University of Udine

Department of Mathematics and Computer Science

PREPRINT

Decidability and Complexity of Timeline-based Planning over Dense Temporal Domains

Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron

Preprint nr.: 1/2018

Reports available from: https://www.dimi.uniud.it/preprints/

Decidability and Complexity of Timeline-based Planning over Dense Temporal Domains

Laura Bozzelli
University of Napoli "Federico II"
Napoli, Italy

Alberto Molinari
University of Udine
Udine, Italy

Angelo Montanari
University of Udine
Udine, Italy

Adriano Peron
University of Napoli "Federico II"
Napoli, Italy

Abstract

Planning is one of the most studied problems in computer science. In the timeline-based approach, the planning domain is modeled as 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, called synchronization rules. The temporal domain is assumed to be discrete, the dense case being dealt with by forcing a suitable discretization. In this paper, we address decidability and complexity issues for timeline-based planning over dense temporal domains without resorting to any form of discretization. We first prove that the general problem is *undecidable* even when a single state variable is involved. Then, we show that *decidability* can be recovered by constraining the logical structure of synchronization rules.

1 Introduction

In this paper, we prove some basic results about decidability and complexity of timeline-based planning over dense temporal domains. Since the 1960s, planning is one of the most studied problems in computer science. In its classical formulation (action-based planning), it can be viewed as the problem of determining a sequence of actions that, given the initial state of the world (domain of interest) and a goal, transforms, step by step, the state of the world until a state that satisfies the goal is reached.

Timeline-based planning is an alternative, more declarative approach to the problem. Unlike action-based planning, it focuses on what has to happen in order to satisfy the goal instead of what an agent has to do. It models the planning domain as a set of independent, but interacting, components, each one consisting of a number of state variables. The evolution of the values of state variables over time is described by means of a set of timelines (sequences of tokens), and it is governed by a set of transition functions, one for each state variable, and a set of synchronization rules, that constrain the temporal relations among state variables.

Timeline-based planning has been successfully exploited in a number of application domains (see, for instance, (Barreiro et al. 2012; Cesta et al. 2007; Chien et al. 2010; Frank and Jónsson 2003; Jónsson et al. 2000; Muscettola 1994)). A systematic study of its expressiveness and complexity has been undertaken only very recently. The temporal domain is assumed to be discrete (the natural numbers), the dense case being commonly dealt with by forcing an artificial discretization of the domain.

In (Gigante et al. 2016), Gigante et al. showed that timeline-based planning 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 (Gigante et al. 2017), they proved that **EXPSPACE**-completeness still holds for timeline-based planning with unbounded interval relations, and that the problem becomes **NEXPTIME**-complete if an upper bound to the temporal horizon is added.

In this paper, we address the timeline-based planning problem over *dense temporal domains without resorting to any form of discretization*. We first show that the general problem is *undecidable* even when a single state variable is used. Then, we prove that decidability can be recovered by suitably constraining the logical structure of synchronization rules, namely, by only admitting trigger-less ones. The achieved results are interesting *per se*; moreover, they identify a large unexplored area of intermediate cases where a good equilibrium between complexity and expressiveness may be expected.

The paper is organized as follows. In Section 2, we provide some background knowledge on timeline-based planning. In Section 3, we prove that planning is undecidable in the general case, by a reduction from the halting problem for Minsky 2-counter machines. Then, in Section 4, we show that decidability can be recovered by restricting to trigger-less synchronization rules: we provide an encoding of the problem into timed automata, obtaining a **PSPACE** planning algorithm, that we expect to be easily implementable by using standard tools based on timed automata, e.g., UPPAAL (Larsen, Pettersson, and Yi 1997), as back-ends. Finally, in Section 5, we outline an **NP** algorithm for planning with trigger-less rules, stemming from the results of the previous section and improving on them.

2 The Timeline-Based Planning Problem

In this section, we give a short account of notation and basic notions of timeline-based planning. For a more detailed illustration, we refer the reader to (Cialdea Mayer, Orlandini, and Umbrico 2016; Gigante et al. 2016).

Hereafter, let $\mathbb{N}$, $\mathbb{R}_+$, and $\mathbb{Q}_+$ be the sets of the naturals, non-negative reals, and non-negative rationals, respectively.

In timeline-based planning, domain knowledge is encoded by a set of state variables, whose behaviour over time is described by transition functions and synchronization rules.

Definition 2.1. A *state variable* x is a triple (V_x, T_x, D_x), where:

- V_x is the *finite domain* of the variable x;
- $T_x : V_x \to 2^{V_x}$ is the *value transition function*, which maps each value $v \in V_x$ to the set of values that x can take immediately after v;
- $D_x : V_x \to \mathbb{I}(\mathbb{Q}_+ \cup \{+\infty\})$ is a function that maps each $v \in V_x$ to an (open or closed, and bounded or unbounded above) interval I with *rational* non-negative bounds.

The value taken by a state variable over a time interval is specified by means of *tokens*.

Definition 2.2. Let $x = (V_x, T_x, D_x)$ be a state variable. A *token* for x is a triple (x, v, d), where $v \in V_x$ is the *value* of x in the token, and $d \in D_x(v)$ is its *duration*.

The sequence of values taken by a state variable is represented by a finite sequence of tokens, called a *timeline*.

Definition 2.3. Let $x = (V_x, T_x, D_x)$ be a state variable. A *timeline* for x is a finite sequence of k tokens (x, v_i, d_i), with $k > 0$, such that $v_{i+1} \in T_x(v_i)$, for $i = 1, \ldots, k-1$.

We define the *start time* and the *end time* of the i-th token of a timeline for x as $\mathsf{s}\big((x, v_i, d_i)\big) = \sum_{j=1}^{i-1} d_j$ and $\mathsf{e}\big((x, v_i, d_i)\big) = \sum_{j=1}^{i} d_j$, respectively.

The behavior of state variables is constrained by a set of *synchronization rules*, which relate tokens, possibly belonging to different timelines, through temporal relations among intervals or among intervals and time points. Let $\Sigma = \{o, o', \ldots\}$ be a set of *token names* used to refer to tokens.

Definition 2.4. An *atom* is either a clause of the form $o \leq^{e_1,e_2}_{[\ell,u]} o'$ (interval), or of the forms $o \leq^{e_1}_{[\ell,u]} t$ or $o \geq^{e_1}_{[\ell,u]} t$ (time-point), where $o, o' \in \Sigma$, $\ell \in \mathbb{Q}_+$, $u \in \mathbb{Q}_+ \cup \{+\infty\}$, $t \in \mathbb{Q}_+$, and e_1 (resp., e_2) is either s or e.

Definition 2.5. Let SV be a set of state variables. An *existential statement* is a statement of the form:

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

where $\mathcal{C} = \rho_0 \wedge \ldots \wedge \rho_m$ is a conjunction of atoms, $o_i \in \Sigma$, $x_i \in SV$, and $v_i \in V_{x_i}$ for each $i = 1, \ldots, n$. The elements $o_i[x_i = v_i]$ are called *quantifiers*. A token name used in $\mathcal{C}$, but not occurring in any quantifier, is said to be *free*.

A *synchronization rule* $\mathcal{R}$ is a clause of one of the forms

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

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* where only o_0 may appear free. In rules of the first form, the quantifier $o_0[x_0 = v_0]$ is called *trigger*. Rules of the second form are said to be *trigger-less*.

Intuitively, the trigger is a universal quantifier, which states that *for all* the tokens o_0, where the variable x_0 takes the value v_0, at least one of the existential statements $\mathcal{E}_i$ must be true. The existential statements in turn assert the existence of tokens $o_1, \ldots, o_n$, where the respective state variables take the specified values, that satisfy the temporal constraints given by $\mathcal{C}$. Trigger-less rules simply assert the satisfaction of the existential statements.

Definition 2.6. Let Γ be a set of tokens and let $\lambda : \Sigma \to \Gamma$ be a function that assigns a token to each token name. An interval atom $o \leq^{e_1,e_2}_{[\ell,u]} o'$ is *satisfied* by λ if $\ell \leq e_2(\lambda(o')) - e_1(\lambda(o)) \leq u$, and a time-point atom $o \leq^{e}_{[\ell,u]} t$ (resp., $o \geq^{e}_{[\ell,u]} t$) is *satisfied* by λ if $\ell \leq t - e(\lambda(o)) \leq u$ (resp., $\ell \leq e(\lambda(o)) - t \leq u$).

Definition 2.7. Given a set of tokens Γ and a function $\lambda : \Sigma \to \Gamma$, a quantifier $o[x = v]$ is *satisfied* by λ if $\lambda(o) = (x, v, d_o)$, for some d_o. An existential statement $\mathcal{E}$, with conjunct clause $\mathcal{C}$, is *satisfied* by λ if all the quantifiers of $\mathcal{E}$ and all the atoms in $\mathcal{C}$ are satisfied by λ.

A synchronization rule of the form $o_0[x_0 = v_0] \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$ is *satisfied* by Γ if, for every token $(x_0, v_0, d) \in \Gamma$, there are an existential statement $\mathcal{E}_i$ and a mapping $\lambda : \Sigma \to \Gamma$ such that $\lambda(o_0) = (x_0, v_0, d)$ and λ satisfies $\mathcal{E}_i$.

A timeline-based planning domain is specified by a set of state variables and a set of synchronization rules modeling their admissible behaviors. Trigger-less rules can be used to express initial conditions and the goals of the problem.

Definition 2.8. A timeline-based planning *problem* is a pair $P = (SV, S)$, where SV is a set of state variables and S is a set of synchronization rules involving variables in SV. A *plan* for P is a set of timelines Π, one for each $x_i \in SV$, such that all the synchronization rules in S are satisfied by the set Γ of all tokens involved in (any of) the timelines of Π.

Note that the 13 Allen's ordering relations between pairs of intervals (Allen 1983) can be defined with interval atoms.

3 Timeline-Based Planning over Dense Time is Undecidable

In this section, we show that timeline-based planning, 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 (Minsky 1967). The proof somehow resembles the one for the satisfiability problem of Metric Temporal Logic with both past and future temporal modalities, interpreted on dense time (Alur and Henzinger 1993).

As a preliminary step, we give a short account of Minsky 2-counter machines. A Minsky 2-counter machine (counter machine for short) is a tuple $M = (\mathsf{Inst}, \ell_{init}, \ell_{halt})$ consisting of a finite set Inst of labeled instructions $\ell : \imath$, where ℓ is a label and $\imath$ is an instruction for either

- *increment*: $c_h := c_h + 1$; `goto` ℓ_r, or
- *decrement*: `if` $c_h > 0$ `then` $c_h := c_h - 1$; `goto` ℓ_s `else` `goto` ℓ_t,

where $h \in \{1, 2\}$, $\ell_s \neq \ell_t$, and ℓ_r (resp., ℓ_s, ℓ_t) is either a label of an instruction in Inst or the halting label ℓ_{halt}. Moreover, ℓ_{init} is the label of a designated instruction in Inst.

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

c_2, respectively. M induces a transition relation, denoted by $\longrightarrow$, over pairs of M-configurations: (i) for an instruction with label ℓ incrementing c_1, $(\ell, n_1, n_2) \longrightarrow (\ell_r, n_1+1, n_2)$, and (ii) for an instruction decrementing c_1, $(\ell, n_1, n_2) \longrightarrow (\ell_s, n_1 - 1, n_2)$, if $n_1 > 0$, and $(\ell, n_1, n_2) \longrightarrow (\ell_t, n_1, n_2)$, otherwise. The analogous for instructions altering c_2. A computation of M is a *finite* sequence $C_1, \ldots, C_k$ of configurations such that $C_i \longrightarrow C_{i+1}$ for all $i \in [1, k-1]$. M *halts* if there is a computation starting at $(\ell_{init}, 0, 0)$ and leading to (ℓ_{halt}, n_1, n_2), for some $n_1, n_2 \in \mathbb{N}$. The halting problem is to decide whether a given machine M halts, and it was proved to be undecidable (Minsky 1967).

Theorem 3.1. *Timeline-based planning over dense time 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 ℓ : `if` $c_h > 0$ `then` $c_h := c_h - 1;$ `goto` ℓ_r `else goto` ℓ_s, we define $c(\ell) := c_h$, $dec(\ell) := \ell_r$, and $zero(\ell) := \ell_s$.

Moreover, let *Lab* 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 (i.e., 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 definition of the sets of *main* values V_{main} and *check* values V_{check} are reported in Figure 1. For each $h = 1, 2$, 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 an M-configuration (ℓ, n_1, n_2), where the instruction label $\ell \in Inc \cup \{\ell_{halt}\}$, $n_1 \geq 0$, and $n_2 \geq 0$, has the form:

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

In the case of a *decrement* instruction label $\ell \in Dec$ such that $c(\ell) = c_1$, the *main*-code w'_{main} for 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.

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

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

In the first case, w'_{main} encodes the configuration $(\ell, 0, n_2)$, in the second case, the configuration (ℓ, 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. Similarly, 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 symbols inc_i, dec_i, and $zero_i$, with $i = 1, 2$: 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 being 0 in C_p).

The *check*-code for an instruction label $\ell \in Lab$ and an inc_1-operation has the form:

$$(\ell, inc_1) \cdot (\ell, inc_1, (c_1, \#)) \cdot \underbrace{(\ell, inc_1, c_1) \ldots (\ell, inc_1, c_1)}_{n_1 \text{ times}} \cdot \underbrace{(\ell, inc_1, c_2) \ldots (\ell, inc_1, c_2)}_{n_2 \text{ times}}$$

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.

The *check*-code for an instruction label $\ell \in Lab$ and an operation $op_1 \in \{dec_1, zero_1\}$ for counter c_1 has the form:

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

where we require that $n_1 = 0$ if $op_1 = zero_1$. The *check*-code for a label $\ell \in Lab$ 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 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 $j \in [1, n-1]$, the following holds (we assume ℓ_i to be the instruction label associated with the configuration code $w^i_{check} \cdot w^i_{main}$):

$$V_{main} := \bigcup_{\ell \in Inc \cup \{\ell_{halt}\}} \bigcup_{h=1,2} \Big(\{\ell\} \cup \{(\ell, c_h)\} \Big) \cup \bigcup_{\ell \in Dec} \bigcup_{\ell' \in \{zero(\ell), dec(\ell)\}} \bigcup_{h=1,2} \Big(\{(\ell, \ell')\} \cup \{(\ell, \ell', c_h)\} \cup \{(\ell, \ell', (c_h, \#))\} \Big)$$

$$V_{check} := \bigcup_{\ell \in Lab} \bigcup_{i,h \in \{1,2\}} \bigcup_{op_i \in \{inc_i, dec_i, zero_i\}} \Big(\{(\ell, op_i)\} \cup \{(\ell, op_i, c_h)\} \cup \{(\ell, op_i, (c_h, \#))\} \Big)$$

Figure 1: Definition of V_{main} and V_{check}.

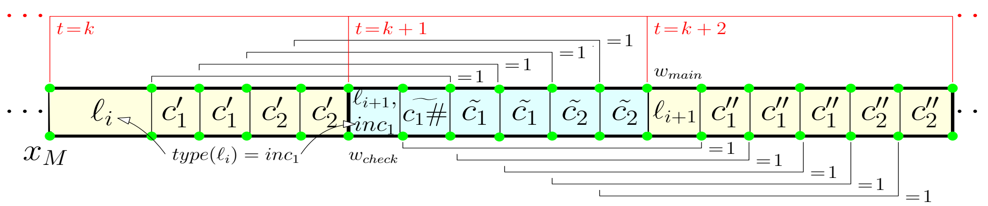

Figure 2: A fragment of a computation code with configuration code for an instruction ℓ_{i+1}. Main-codes are highlighted in yellow and check-codes in cyan. Each square can also be seen as a token of a timeline for x_M (tokens are decorated with their start time and their temporal constraints). In the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

- $\ell_j \neq \ell_{halt}$;
- if $\ell_j \in Inc$ 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 $\ell_j \in Dec$ with $c(\ell_j) = c_h$, and the first symbol of w^j_{main} is $(\ell_j, zero(\ell_j))$ (resp., $(\ell_j, dec(\ell_j))$), then $\ell_{j+1} = zero(\ell_j)$ (resp., $\ell_{j+1} = dec(\ell_j)$) 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 $j \in [1, n-1]$, 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 $\ell_j \in Inc$, 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 $\ell_j \in Dec$, with $c(\ell_j) = c_h$, then $n^{j+1}_{3-h} = n^j_{3-h}$. Moreover, if w^{j+1}_{check} is associated with dec_h, then $n^{j+1}_h = n^j_h - 1$.

Clearly, a well-formed computation code π encodes a computation of the counter machine. A computation-code π is *initial* if it starts with the prefix $(\ell_{init}, zero_1) \cdot \ell_{init}$, 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.

Let us show how to reduce the problem of checking the existence of an initial and halting well-formed computation-code to a planning 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.

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 such that the untimed part of every timeline, i.e., neglecting tokens' durations, for x_M satisfying the rules in R_M is an initial and halting well-formed computation-code. Thus, M halts if and only if there is a timeline of x_M satisfying the rules in R_M.

As for x_M, let $x_M = (V, T, D)$, where, for each $v \in V$, $D(v) = (0, 1]$. This sets the *strict time monotonicity* constraint, i.e., 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 3.2. *The untimed part of each timeline for x_M whose first token has value $(\ell_{init}, zero_1)$ is a prefix of some initial computation-code. Moreover, $(\ell_{init}, zero_1) \notin T(v)$ for all $v \in V$.*

By construction, it is a straightforward task to define T in such a way that the previous requirement is fulfilled (for details, see the appendix).

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

- *Initialization:* every timeline starts with two tokens, the first one having value $(\ell_{init}, zero_1)$, and the second one having value ℓ_{init}. By Claim 3.2 and the fact that no instruction of M leads to ℓ_{init}, it suffices to require that a timeline has a token with value $(\ell_{init}, zero_1)$ and a token with value ℓ_{init}. This is ensured by the following two trigger-less rules: $\top \rightarrow \exists\, o[x_M = (\ell_{init}, zero_1)]. \top$ 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 the initialization and Claim 3.2, 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 $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$. By Claim 3.2, 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 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 3.2, 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 have 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:
$$o[x_M = v] \rightarrow \bigvee_{u \in V_{c_h}} \exists\, o'[x_M = u].\, o' \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o.$$
- *Increment and decrement:* we need to guarantee that the increment and decrement instructions are correctly simulated. By Claim 3.2 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 $\ell_{main} \in \mathit{Inc}$ and $\ell_{check} = \mathit{succ}(\ell_{main})$, or $\ell_{main} \in \mathit{Dec}$ and $\ell_{check} \in \{\mathit{zero}(\ell_{main}), \mathit{dec}(\ell_{main})\}$. Moreover, if $\ell_{main} \in \mathit{Dec}$ and $\ell_{check} = \mathit{zero}(\ell_{main})$, then $n_h^{check} = n_h^{main} = 0$. Thus, it remains to ensure the following two requirements:
  - (*) if $\ell_{main} \in \mathit{Inc}$, then $n_h^{check} = n_h^{main} + 1$ and $n_{3-h}^{check} = n_{3-h}^{main}$;
  - (**) if $\ell_{main} \in \mathit{Dec}$, then $n_{3-h}^{check} = n_{3-h}^{main}$, and whenever $\ell_{check} = \mathit{dec}(\ell_{main})$, then $n_h^{check} = n_h^{main} - 1$.

  First, we observe that if $\ell_{main} \in \mathit{Inc}$, 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 $\ell_{main} \in \mathit{Dec}$, 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 $\ell_{check} = \mathit{dec}(\ell_{main})$, then there is exactly one marked V_{c_h}-value in w_{main}. Then, 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 label ℓ_{init} (resp., ℓ_{halt}). For each $v \in (U_{c_i} \cap V_{main}) \setminus V_{halt}$, we have:
$$o[x_M = v] \rightarrow \bigvee_{u \in U_{c_i}} \exists\, o'[x_M = u].\, o \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o'.$$
  For each $v \in (U_{c_i} \cap V_{check}) \setminus V_{init}$, we have:
$$o[x_M = v] \rightarrow \bigvee_{u \in U_{c_i}} \exists\, o'[x_M = u].\, o' \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o.$$

This concludes the proof of the theorem. □

4 The Trigger-less Case is Decidable

In this section, we show that decidability of the timeline-based planning problem can be recovered if we restrict ourselves to trigger-less synchronization rules. To this aim, we suitably encode the planning problem into a parallel composition of timed automata (TA) whose only communication mean is clock sharing. Each timeline can be seen as a timed word "described" by the TA associated with the corresponding variable. A plan for k variables is then a timed k-multiword, namely, a timed word over a structured alphabet featuring a component for each variable (i.e., a timed word having k timed synchronized traces, one for each timeline). We call k-MWTA the composition of TAs accepting the k-multiwords encoding plans. In particular, we show that each trigger-less rule can be implemented by using shared clocks and diagonal constraints over clock values associated with TA components. The planning problem with trigger-less synchronization rules can thus be naturally reduced to the emptiness problem for the k-MWTA encoding it. By tailoring the standard region-based construction for TAs, we prove that emptiness of a k-MWTA can be solved in **PSPACE**.

We start with a short summary of the standard notions of timed word and TA, and of their semantics. Let w be a finite or infinite word over some alphabet. An *infinite timed word* w over a finite alphabet Σ is an infinite word $w = (a_1, \tau_1)(a_2, \tau_2)\cdots$ over $\Sigma \times \mathbb{R}_+$ (intuitively, τ_i is the time at which the event a_i occurs) such that the sequence $\tau = \tau_1, \tau_2, \ldots$ of timestamps satisfies: (1) $\tau_i \leq \tau_{i+1}$ for all $i \geq 1$ (monotonicity), and (2) for all $t \in \mathbb{R}_+$, $\tau_i \geq t$ for some $i \geq 1$ (divergence/progress). The timed word w is also denoted by the pair (σ, τ), where σ is the (untimed) infinite word $a_1 a_2 \cdots$ and τ is the sequence of timestamps. An *ω-timed language* over Σ is a set of infinite timed words over Σ.

Let us now give a short account of the formalism of *timed automata* (TA, see (Alur and Dill 1994)). Let us fix an alphabet Σ. A *clock constraint* over a set C of clocks is a Boolean combination of atomic formulas of the form $c \bullet c' + cst$ or $c \bullet cst$, where $\bullet \in \{\geq, \leq\}$, $c, c' \in C$, and $cst \in \mathbb{Q}_+$ is a constant. We will often use the interval-based notation instead of a conjunction of two atomic formulas, e.g., $c \in [2, 7.4]$. We denote the set of clock constraints over C by $\Phi(C)$.

A clock valuation $val : C \to \mathbb{R}_+$ for C is a function assigning a real value to each clock of C. Given a clock valuation val for C and a clock constraint θ over C, we say that val satisfies θ, written $val \models \theta$, if θ evaluates to true replacing each occurrence of a clock c in θ by $val(c)$, and interpreting Boolean connectives in the standard way. Given $t \in \mathbb{R}_+$, $(val + t)$ denotes the valuation such that, for all $c \in C$, $(val + t)(c) = val(c) + t$. For $Res \subseteq C$, $val[Res](c) = 0$ if $c \in Res$, and $val[Res](c) = val(c)$ otherwise.

Definition 4.1. *A (Büchi) TA* over Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, C, \Delta, F)$, where Q is a finite set of (control) states, $Q_0 \subseteq Q$ is the set of initial states, C is a finite set of clocks, $F \subseteq Q$ is the set of accepting states, and $\Delta \subseteq Q \times \Sigma \times \Phi(C) \times 2^C \times Q$ is the transition relation.

The intuitive behavior of a Büchi TA $\mathcal{A}$ is the following. Assume that $\mathcal{A}$ is on state $q \in Q$ after reading $i \geq 0$ symbols, the i-th symbol is read at time τ_i and, at that time, the clock valuation is $sval$. On reading the $(i+1)$-th symbol (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 $(sval + t)$, with $t = \tau_{i+1} - \tau_i$. The control then changes from q to q' and $sval$ 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, $sval$ is updated to $(sval + t)[Res]$.

Formally, a configuration of $\mathcal{A}$ is a pair $(q, sval)$, where $q \in Q$ and $sval$ is a clock valuation for C. A run π of $\mathcal{A}$ over $w = (\sigma, \tau)$ is an infinite sequence of configurations $\pi = (q_0, sval_0)(q_1, sval_1)\cdots$ such that $q_0 \in Q_0$, $sval_0(c) = 0$ for all $c \in C$ (*initiation* requirement), and the following constraint holds (*consecution*): for all $i \geq 1$ (we let $\tau_0 = 0$),

- for some $(q_{i-1}, \sigma_i, \theta, Res, q_i) \in \Delta$, $sval_i = (sval_{i-1} + \tau_i - \tau_{i-1})[Res]$ and $(sval_{i-1} + \tau_i - \tau_{i-1}) \models \theta$.

The run π is *accepting* if there are infinitely many positions $i \geq 0$ such that $q_i \in F$. The *timed language* $\mathcal{L}_T(\mathcal{A})$ of $\mathcal{A}$ is

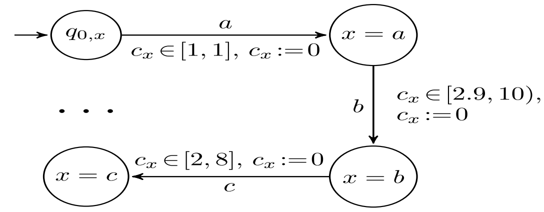

Figure 3: (Part of) a Büchi TA $\mathcal{A}_x$ for a state variable $x = (V_x, T_x, D_x)$, with $V_x = \{a, b, c, \ldots\}$, $b \in T_x(a)$, $c \in T_x(b) \ldots$, $D_x(a) = [2.9, 10)$, $D_x(b) = [2, 8], \ldots$.

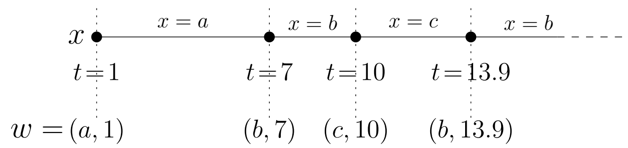

Figure 4: An example of timeline for the state variable x of Figure 3, with the timed word encoding it, accepted by $\mathcal{A}_x$.

the set of infinite timed words w over Σ such that there is an accepting run of $\mathcal{A}$ over w.

We now introduce the notion of *timeline encoded by a timed word*, and of TA *for a state variable*. We assume that, for every x and every $v \in V_x$, we have $T_x(v) \neq \emptyset$ (at the end of the section it is shown how to relax this constraint).

Definition 4.2. Let $x = (V_x, T_x, D_x)$ be a state variable. The *timeline for x encoded by a timed word* $(a_1, \tau_1)(a_2, \tau_2)\cdots$ is the sequence of tokens $(x, a_1, t_1)(x, a_2, t_2)\cdots$, where, for $i \geq 1$, $a_i \in V_x$, $a_{i+1} \in T_x(a_i)$, and $t_i = \tau_{i+1} - \tau_i \in D_x(a_i)$.

Definition 4.3. Let $x = (V_x, T_x, D_x)$ be a state variable. *A TA for x* is a tuple $\mathcal{A}_x = (V_x, Q, \{q_{0,x}\}, \{c_x\}, \Delta, Q)$, where $Q = V_x \cup \{q_{0,x}\}$ ($q_{0,x} \notin V_x$), and $\Delta = \{(v', v, c_x \in D_x(v'), \{c_x\}, v) \mid v', v \in V_x,\ v \in T_x(v')\} \cup \{(q_{0,x}, v, c_x \in [1, 1], \{c_x\}, v) \mid v \in V_x\}$.

Intuitively, this automaton accepts all timed words encoding a timeline for the state variable x. Let us note that all states are accepting. Moreover, the constraints of a transition $\delta \in \Delta$ on the unique clock c_x are determined by the (value of the) source state of δ. For technical reasons, which will be clear in the following, we set $c_x \in [1, 1]$ on all the transitions from the initial state $q_{0,x}$. See Figure 3 and 4 for an example.

In the following, we introduce the formalism of k-multiword TA (k-MWTA). A k-MWTA is the parallel composition of k TAs which share the same clocks for synchronization.

A k-MWTA accepts a language of timed k-multiwords, formally defined as follows. For $k \geq 1$ pairwise disjoint alphabets $\Sigma_1, \ldots, \Sigma_k$, and the symbol $\epsilon \notin \bigcup_{1 \leq i \leq k} \Sigma_i$, k-Σ denotes the multialphabet $\{(a_1, \ldots, a_k) \mid a_i \in \Sigma_i \cup \{\epsilon\},$ for $1 \leq i \leq k\} \setminus \{(\epsilon, \ldots, \epsilon)\}$. A *$k$-multiword* is a word over k-Σ. Intuitively, a k-multiword $\vec{a}_1, \vec{a}_2, \ldots$ is a synchronization of k words over the alphabets $\Sigma_1, \ldots, \Sigma_k$; in $\vec{a}_j = (a_j^1, \ldots a_j^k)$, for $j \geq 1$, all the symbols a_j^i such that $a_j^i \neq \epsilon$, with $1 \leq i \leq k$, occur at the same instant of time, and if $a_j^i = \epsilon$, no symbol (event) of the i-th alphabet Σ_i

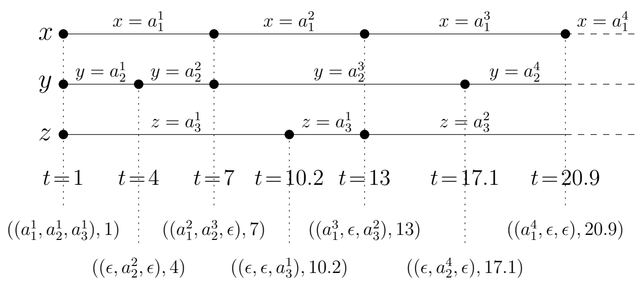

Figure 5: An example of timelines for the state variables x, y, z together with the timed 3-multiword encoding them.

occurs at that time. A timed k-multiword is just a timed word (σ, τ) where the untimed word σ is a k-multiword. The notion of timeline encoded by a timed word can be extended to the (list of) timelines encoded by a timed k-multiword.

Definition 4.4. The timeline for x_j encoded by a timed k-multiword $(\vec{a}_1, \tau_1)(\vec{a}_2, \tau_2)\cdots$ is the timeline encoded by the timed word $\mathsf{d}\epsilon\big((\vec{a}_1[j], \tau_1)(\vec{a}_2[j], \tau_2)\cdots\big)$, where $\mathsf{d}\epsilon((\sigma, \tau))$ is the timed word obtained from (σ, τ) by removing all the occurrences of pairs (ϵ, τ'), with $\tau' \in \mathbb{R}_+$.

See Figure 5 for an example.

A k-MWTA is a suitable parallel composition of TA communicating via shared clocks.

Definition 4.5. Let $\mathcal{A}_i = (\Sigma_i, Q_i, Q_{0,i}, C_i, \Delta_i, F_i)$, with $1 \le i \le k$, be k TA. A k-multiword TA (k-MWTA) for $\mathcal{A}_1, \ldots, \mathcal{A}_k$ is a TA k-$\mathcal{A} = (k\text{-}\Sigma, Q, Q_0, C, \Delta, F)$, where

- $Q = (Q_1 \times \ldots \times Q_k) \cup \{q_f\}$, with q_f an auxiliary state,
- $Q_0 = Q_{0,1} \times \ldots \times Q_{0,k}$,
- $C = C_1 \cup \ldots \cup C_k$,
- $F = (F_1 \times \ldots \times F_k) \cup \{q_f\}$,
- if $\Big((q_1, \ldots, q_k), (a_1, \ldots, a_k), \Theta, \mathcal{C}, (q'_1, \ldots, q'_k)\Big) \in \Delta$ for $(q_1, \ldots, q_k), (q'_1, \ldots, q'_k) \in Q \setminus \{q_f\}$, then $\Theta = \bigwedge_{i=1}^k \theta_i, \mathcal{C} = \bigcup_{i=1}^k \mathit{Res}_i$, and for all $i = 1, \ldots, k$,
  - if $a_i \ne \epsilon$, there exists $(q_i, a_i, \theta_i, \mathit{Res}_i, q'_i) \in \Delta_i$,
  - if $a_i = \epsilon$, it holds $q_i = q'_i$, $\theta_i = \top$ and $\mathit{Res}_i = \emptyset$.

We define now a k-MWTA for a set of state variables $x_1, \ldots, x_k$.

Definition 4.6. Let $x_1, \ldots, x_k$ be k state variables. A k-MWTA for $x_1, \ldots, x_k$ is the k-MWTA k-$\mathcal{A}_{x_1,\ldots,x_k}$ for $\mathcal{A}_{x_i} = (\Sigma_i = V_{x_i}, Q_i, \{q_{0,i}\}, \{c_i\}, \Delta_i, Q_i)$, $1 \le i \le k$, defined as k-$\mathcal{A}_{x_1,\ldots,x_k} = (k\text{-}\Sigma, Q, \{q_0\}, C, \Delta_1 \cup \Delta_2, F)$, where

- $Q = Q_1 \times \ldots \times Q_k$ and $q_0 = (q_{0,1}, \ldots, q_{0,k})$,
- $C = \{c_1, \ldots, c_k\}$ and $F = Q_1 \times \ldots \times Q_k$,
- $\Big((q_1, \ldots, q_k), (a_1, \ldots, a_k), \Theta, \mathcal{C}, (q'_1, \ldots, q'_k)\Big) \in \Delta_1$ iff $(q_1, \ldots, q_k) \ne q_0$, $\Theta = \bigwedge_{i=1}^k \theta_i, \mathcal{C} = \bigcup_{i=1}^k \mathit{Res}_i$, and for all $i = 1, \ldots, k$, we have
  - if $a_i \ne \epsilon$, there exists $(q_i, a_i, \theta_i, \mathit{Res}_i, q'_i) \in \Delta_i$,
  - if $a_i = \epsilon$, it holds $q_i = q'_i$, $\theta_i = \top$ and $\mathit{Res}_i = \emptyset$.
- $\Delta_2 = \{(q_0, (a_1, \ldots, a_k), \bigwedge_{i=1}^k (c_i \in [1,1]), \bigcup_{i=1}^k \{c_i\}, (a_1, \ldots, a_k)) \mid (a_1, \ldots, a_k) \in \Sigma_1 \times \ldots \times \Sigma_k\}$.

The following result clearly holds by construction.

Proposition 4.7. *Given $k \ge 1$ state variables $x_1, \ldots, x_k$, $\mathcal{L}_T(k\text{-}\mathcal{A}_{x_1,\ldots,x_k})$ is a set of k-multiwords, each one encoding a timeline for each of $x_1, \ldots, x_k$.*

Let us now introduce the k-MWTA for the *synchronization rules*. Since each rule has the form $\mathcal{E}_1 \vee \ldots \vee \mathcal{E}_n$, we focus on the construction for a disjunct $\mathcal{E}_i$, taking then the union of the corresponding automata for the sake of the whole rule. Let us consider a disjunct $\mathcal{E}_i$ of the form $\exists o_1[x_1 = v_{1,j_1}] \cdots \exists o_n[x_n = v_{n,j_n}].\mathcal{C}$, where $\mathcal{C}$ is a conjunction of atoms. We associate two clock variables with each quantifier $o_i[x_i = v_{i,j_i}]$—named $c_{o_i,S}$ and $c_{o_i,E}$—which, intuitively, are reset when the token chosen for o_i starts and ends, respectively. In order to select a suitable token along the timeline, $c_{o_i,S}$ and $c_{o_i,E}$ are non-deterministically reset when x_i takes the value $v_{i,j_i} \in V_{x_i}$. Moreover, to deal with atoms involving a time constant (time-point atoms), we also introduce a clock variable c_{glob}, which measures the current time and is *never reset*. For technical reasons, we assume that the start of activities is at time 1 and, consequently, the reset of any $c_{o_i,S}$ and $c_{o_i,E}$ cannot happen *before* 1 time unit has passed from the beginning of the timed word/plan. In fact, in Definition 4.3, we have $c_x \in [1,1]$ on all the transitions from $q_{0,x}$ (for this reason, we must also add 1 to all time constants in all time-point atoms). This assumption implies that the value of $c_{o_i,S}$ is equal to that of c_{glob} if $c_{o_i,S}$ has never been reset, and less otherwise. Since only one token for each quantifier is chosen in a timeline, $c_{o_i,S}$ must be reset only once: a transition resetting $c_{o_i,S}$ is enabled only if the constraint $c_{o_i,S} = c_{glob}$ is satisfied (likewise for $c_{o_i,E}$).

In the following we define a TA for a state variable x, suitably resetting the clocks associated with all the quantifiers over x. For a set of token names O, we denote by $\Lambda(O, x, v)$ the subset of names $o \in O$ such that $o[x = v]$ for a variable x and a value $v \in V_x$, and by $\Lambda(O, x) = \bigcup_{v \in D_x} \Lambda(O, x, v)$. Moreover $C_S(O)$ (resp., $C_E(O)$) represents the set $\{c_{o,S} \mid o \in O\}$ (resp., $\{c_{o,E} \mid o \in O\}$).

Definition 4.8. Given a state variable $x = (V_x, T_x, D_x)$, and quantifiers $o_1[x = v_1], \ldots, o_\ell[x = v_\ell]$, a TA for $x, o_1, \ldots, o_\ell$ is $\mathcal{A}_{x,o_1,\ldots,o_\ell} = (V_x, Q, \{q_0\}, C, \Delta_1 \cup \Delta_2, \emptyset)$, where

- $Q = V_x \cup \{q_0\}$,
- $C = \{c_{glob}\} \cup \{c_{o_i,S}, c_{o_i,E} \mid i = 1, \ldots, \ell\}$,
- Δ_1 is the set of tuples

$$\Big(v, a, \bigwedge_{o \in P} c_{o,S} = c_{glob} \wedge \bigwedge_{o \in R} (c_{o,S} < c_{glob} \wedge c_{o,E} = c_{glob}) \wedge \bigwedge_{o \in \Lambda(\{o_1,\ldots,o_k\},x,v) \setminus R} (c_{o,S} = c_{glob} \vee c_{o,E} < c_{glob}), C_S(P) \cup C_E(R), a\Big)$$

with $v \in V_x$, $P \subseteq \Lambda(\{o_1, \ldots, o_\ell\}, x, a)$ and $R \subseteq \Lambda(\{o_1, \ldots, o_\ell\}, x, v)$;

- $\Delta_2 = \{(q_0, a, c_{glob} \in [1,1], C_S(P), a) \mid a \in V_x, P \subseteq \Lambda(\{o_1, \ldots, o_\ell\}, x, a)\}$.

In the above definition, R (resp., $\Lambda(\{o_1, \ldots, o_k\}, x, v) \setminus R$) is the set of token names whose end clock *must (resp., must not) be reset*. A number k of TAs, each one defined for the quantifiers over a state variable x_i (with $1 \leq i \leq k$) occurring in an existential statement $\mathcal{E}$, are then synchronized by defining a k-MWTA.

Definition 4.9. Given k state variables $x_1, \ldots, x_k$ and $\mathcal{E} = \exists o_1[x_{j_1} = v_1] \cdots \exists o_n[x_{j_n} = v_n].\mathcal{C}$, with $\{j_1, \ldots, j_n\} \subseteq \{1, \ldots, k\}$, a k-MWTA for $x_1, \ldots, x_k, o_1, \ldots, o_n$ and $\mathcal{E}$ is a k-MWTA for the TAs $\mathcal{A}_{x_i, \Lambda(\{o_1, \ldots, o_n\}, x_i)} = (\Sigma_i = V_{x_i}, Q_i, \{q_{0,i}\}, C_i, \Delta_i, \emptyset)$ for $i = 1, \ldots, k$, k-$\mathcal{A}_{x_1, \ldots, x_k, \mathcal{E}} = (k\text{-}\Sigma, Q, \{q_0\}, C, \Delta_1 \cup \Delta_2 \cup \Delta_3 \cup \Delta_4, \{q_f\})$, where

- $Q = (Q_1 \times \ldots \times Q_k) \cup \{q_f\}$, and $q_0 = (q_{0,1}, \ldots, q_{0,k})$,
- $C = \{c_{glob}\} \cup \{c_{o_i,S}, c_{o_i,E} \mid i = 1, \ldots, n\}$,
- Δ_1 is the set of tuples

$$\Big((v_1, \ldots, v_k), (a_1, \ldots, a_k), \bigwedge_{i=1}^{k} \Big(\bigwedge_{o \in P_i} c_{o,S} = c_{glob} \wedge \bigwedge_{o \in R_i} (c_{o,S} < c_{glob} \wedge c_{o,E} = c_{glob}) \wedge \bigwedge_{o \in \overline{R}_i} (c_{o,S} = c_{glob} \vee c_{o,E} < c_{glob})\Big), \bigcup_{i=1}^{k} (C_S(P_i) \cup C_E(R_i)), (v'_1, \ldots, v'_k)\Big)$$

  satisfying the following conditions, for all $i = 1, \ldots, k$:
  - if $a_i = \epsilon$, then $v_i = v'_i$, $P_i = \emptyset$, and $R_i = \overline{R}_i = \emptyset$,
  - if $a_i \neq \epsilon$, then $a_i = v'_i \in D_{x_i}$, $P_i \subseteq \Lambda(\{o_1, \ldots, o_n\}, x_i, v'_i)$, and $R_i \dot\cup \overline{R}_i = \Lambda(\{o_1, \ldots, o_n\}, x_i, v_i)$,
- $\Delta_2 = \{(q_0, (a_1, \ldots, a_k), c_{glob} \in [1,1], \bigcup_{i=1}^{k} C_S(P_i), (a_1, \ldots, a_k)) \mid (a_1, \ldots, a_k) \in V_{x_1} \times \ldots \times V_{x_k}, P_i \subseteq \Lambda(\{o_1, \ldots, o_n\}, x_i, a_i) \text{ for } 1 \leq i \leq k\}$,
- $\Delta_3 = \{(q, (a_1, \ldots, a_k), \Phi_{\mathcal{C}} \wedge \bigwedge_{i=1}^{k} (c_{o_i,S} < c_{glob} \wedge c_{o_i,E} < c_{glob}), \emptyset, q_f) \mid q \in V_{x_1} \times \ldots \times V_{x_k}, (a_1, \ldots, a_k) \in k\text{-}\Sigma\}$, where $\Phi_{\mathcal{C}}$ is the translation of $\mathcal{C}$ into a TA clock constraint suitably obtained by exploiting the correspondences among atoms and TA clock constraints outlined in Table 1,
- $\Delta_4 = \{(q_f, (a_1, \ldots, a_k), \top, \emptyset, q_f) \mid (a_1, \ldots, a_k) \in k\text{-}\Sigma\}$.

Intuitively, q_f is an accepting *sink* state, that the automaton can enter only after all token clocks have been reset (i.e., $\bigwedge_{i=1}^{k} (c_{o_i,S} < c_{glob} \wedge c_{o_i,E} < c_{glob})$) and all $\mathcal{C}$'s conditions are verified. See Figure 6 for an example.

We construct the TA $\tilde{\mathcal{A}}$ for a timeline-based planning problem $P = (\{x_1, \ldots, x_k\}, S)$ by exploiting the standard union and intersection operations of TAs. $\tilde{\mathcal{A}}$ is obtained by intersecting (i) the k-MWTA k-$\mathcal{A}_{x_1, \ldots, x_k}$ for the state variables (see Definition 4.6) with (ii) a TA k-$\mathcal{A}_{x_1, \ldots, x_k, \mathcal{R}}$ for each trigger-less synchronization rule $\mathcal{R} = \top \to \bigvee_{1 \leq i \leq m} \mathcal{E}_i$ in S, which, in turn, is the disjunction of m k-MWTAs k-$\mathcal{A}_{x_1, \ldots, x_k, \mathcal{E}_i}$ as in Definition 4.9 (one for each $\mathcal{E}_i$ in $\mathcal{R}$). The next proposition states the correctness of $\tilde{\mathcal{A}}$.

Proposition 4.10. *$\mathcal{L}_T(\tilde{\mathcal{A}})$ is the set of plans for $P = (\{x_1, \ldots, x_k\}, S)$.*

ρ_i	Φ_{ρ_i}
$o_1 \leq^{s,s}_{[\ell,u]} o_2$	$c_{o_2,S} + \ell \leq c_{o_1,S} \leq c_{o_2,S} + u$
$o_1 \leq^{s,e}_{[\ell,u]} o_2$	$c_{o_2,E} + \ell \leq c_{o_1,S} \leq c_{o_2,E} + u$
$o_1 \leq^{e,s}_{[\ell,u]} o_2$	$c_{o_2,S} + \ell \leq c_{o_1,E} \leq c_{o_2,S} + u$
$o_1 \leq^{e,e}_{[\ell,u]} o_2$	$c_{o_2,E} + \ell \leq c_{o_1,E} \leq c_{o_2,E} + u$
$o \leq^{s}_{[\ell,u]} t$	$(\ell - t') + c_{glob} \leq c_{o,S} \leq (u - t') + c_{glob}$
$o \leq^{e}_{[\ell,u]} t$	$(\ell - t') + c_{glob} \leq c_{o,E} \leq (u - t') + c_{glob}$
$o \geq^{s}_{[\ell,u]} t$	$(\ell + t') + c_{o,S} \leq c_{glob} \leq (u + t') + c_{o,S}$
$o \geq^{e}_{[\ell,u]} t$	$(\ell + t') + c_{o,E} \leq c_{glob} \leq (u + t') + c_{o,E}$

Table 1: For a conjunction of atoms $\mathcal{C} = \rho_1 \wedge \ldots \wedge \rho_m$, we have $\Phi_{\mathcal{C}} = \Phi_{\rho_1} \wedge \ldots \wedge \Phi_{\rho_m}$, where Φ_{ρ_i} is reported in the table and t' stands for $t + 1$.

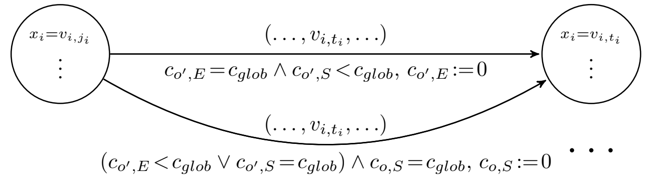

Figure 6: Example of transitions of k-$\mathcal{A}_{x_1, \ldots, x_k, \mathcal{E}}$ in Definition 4.9, taking two quantifiers $o[x_i = v_{i,t_i}]$ and $o'[x_i = v_{i,j_i}]$. The transition above, having the constraint $c_{o',E} = c_{glob} \wedge c_{o',S} < c_{glob}$, resets $c_{o',E}$. The transition below, having constraint $c_{o',E} < c_{glob} \vee c_{o',S} = c_{glob}$, does not reset $c_{o',E}$. Moreover, the transition above does not reset $c_{o,S}$, while the one below resets it, checking that $c_{o,S} = c_{glob}$.

Theorem 4.11. *Let $P = (\{x_1, \ldots, x_k\}, S)$ be a timeline-based planning problem with trigger-less rules only. Checking the existence of a plan for P is a problem in **PSPACE**.*

Proof. Let us refer to the previously defined TA $\tilde{\mathcal{A}}$ for $P = (\{x_1, \ldots, x_k\}, S)$. (i) The number of clocks $|C|$ of $\tilde{\mathcal{A}}$ is $O(k + |S| \cdot d \cdot s)$, where d is the number of disjuncts in the longest trigger-less rule in S, and s the maximum number of quantifiers in an existential statement. (ii) The number of states of $\tilde{\mathcal{A}}$ is $\mathcal{V} = O(\prod_{i=1}^{k} |V_{x_i}|) \cdot O(|S| \cdot (\prod_{i=1}^{k} |V_{x_i}| \cdot d)^{|S|}) = O(|S| \cdot (V^k \cdot d)^{|S|+1})$, where $V = \max_{i=1}^{k} |V_{x_i}|$. (iii) The number of transitions is $\mathcal{U} = O(\mathcal{V}^2 \cdot (2^{\alpha})^{2|S|})$, being α the total number of quantifier occurrences in S rules.

Let us observe that $\tilde{\mathcal{A}}$ can be built on the fly, that is, by looking at the Δ's of Definition 4.6 and 4.9, one can determine, given a state q, a successor q' and the connecting transition, along with the associated constraints and clocks to be reset. Encoding a state or a transition requires $O(|S| \cdot k \cdot \log(|S| \cdot V \cdot d) + |S| \cdot \alpha)$ bits. We also note that the length of constraints on $\tilde{\mathcal{A}}$'s transitions is polynomial.

We now have to inspect the standard emptiness checking algorithm for TAs, in order to verify that the space complexity remains polynomial, even if the TA $\tilde{\mathcal{A}}$ has *exponential size* in the input timeline-based planning problem.

Such a check involves building the so-called *region automaton* $R(\tilde{\mathcal{A}})$ for $\tilde{\mathcal{A}}$, whose states are pairs (q, r), where q is a state of $\tilde{\mathcal{A}}$ and r a *region*: every region specifies, for each clock c of $\tilde{\mathcal{A}}$, 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 number of clock regions is $r = O(|C|! \cdot 2^{|C|} \cdot 2^{2\alpha^2} \cdot \prod_{c\in C}(2K_c+2))$ (Alur and Dill 1994).[1] Thus, to encode a region we need $O(\log(|C|!) + \log(2^{|C|}) + \log(2^{2\alpha^2}) + \log \prod_{c\in C}(2K_c+2)) = O(|C|\log|C| + |C| + 2\alpha^2 + \log(2K+2)^{|C|}) = O(|C|\log|C| + 2\alpha^2 + |C|\log(2K+2))$ bits, where $K = \max_{c\in C} K_c$. Such K is, in our case, the maximum constant occurring in the planning problem, be it either an upper/lower bound of an interval of a token duration, a time constant in an atom, or the upper/lower bound (u or ℓ) at the subscript of an atom (we assume them to be encoded in binary). Thus a region can be encoded in polynomial space. Finally, given a region, it is easy to determine a successor region on the fly. The number of states of $R(\tilde{\mathcal{A}})$ is thus $r \cdot \mathcal{V}$. Every region has at most $\sum_{c\in C}(2K_c+2)$ successors (Alur and Dill 1994), hence the number of transitions of $R(\tilde{\mathcal{A}})$ is $\mathcal{U} \cdot \sum_{c\in C}(2K_c+2)$. The construction concludes by basically considering $R(\tilde{\mathcal{A}})$ as a generalized Büchi automaton, and by performing an emptiness checking over a Büchi automaton of $O(|C| \cdot r \cdot \mathcal{V})$ states derived from the previous one. □

To relax the assumption made in the construction above that, for every x and $v \in V_x$, we have $T_x(v) \neq \emptyset$, we proceed as follows. For every x with $T_x(v) = \emptyset$ for $v \in V_x$, we set $T_x(v) = \{rej_x\}$, $T_x(rej_x) = \{rej_x\}$ and $D_x(rej_x) = [1,1]$, being $rej_x \in V_x$ a fresh domain element ("rejection element") of x. In $\mathcal{A}_x$, we add one clock, $c_{x,rej}$, which is reset on every transition from a state $(x = v)$, with $v \neq rej_x$, into the state $(x = rej_x)$. Then, in any Δ_3-transition of each k-$\mathcal{A}_{x_1,\ldots,x_k,\mathcal{E}}$, we add the constraint $\bigwedge_x((c_{glob} = c_{x,rej}) \vee \bigwedge_{o \text{ in } \mathcal{E}}(c_{o,E} \geq c_{x,rej}))$: this forces every token associated with a quantifier to have its end *before* any variable x gets into its value rej_x.

5 The Trigger-less Case is NP-complete

The given timed automaton-based planning algorithm has a *sub-optimal* complexity: it is possible to show that timeline-based planning with trigger-less rules is in fact **NP**-complete. However, the proposed encoding of a problem into a TA is a preliminary step towards the proof of this stricter complexity result, as it allows us to determine a bounded horizon (namely, the end time of the last token) for the plans of a problem P: if P admits a plan, then it always admits a plan having such a bounded horizon. Analogously, the automaton encoding allows us to fix a bound to the number of tokens in a plan for P. Here we sketch the proof, referring to the appendix for details.

We observe that, if we consider a path among the $g = O(|C| \cdot r \cdot \mathcal{V})$ states of the region (Büchi) automaton built from the timed automaton for P at the end of the proof of Theorem 4.11, each edge/transition in such path corresponds to the start point of at least a token in some timeline of a (candidate) plan for P, and, if more tokens start simultaneously, of at most a token for each timeline. This yields a bound, $O(g \cdot |SV|)$, on the number of tokens. Analogously, we derive a bound on the horizon of the plan, $O(g \cdot |SV| \cdot (K+1))$, being K the maximum constant occurring in P.

[1] $2\alpha^2$ is due to the presence of *diagonal* clock constraints.

Having determined these bounds, we can now describe the algorithm. As a preprocessing step, we reduce to integers all the rational values occurring in P by multiplying them by the lcm γ of all denominators. It is routine to check that, having a plan for the new problem P', we can transform it into a plan for the original P, by dividing the start/end times of all tokens in each timeline by γ.

Then, for every quantifier $o_i[x_i = v_i]$ in the rules of P', the algorithm guesses the integer part of both the start and the end time of the token for x_i to which o_i is mapped. Moreover, it guesses an order of all fractional parts of such start/end times. Being all constants in P' integers, we have the following property: 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 atoms occurring in the rules does not change.

Now we have to check that there exists a legal timeline evolution "connecting" each pair of adjacent guessed tokens over the same variable (two tokens are *adjacent* if there is no other token associated with a quantifier in between). The idea is to interpret each state variable $x_i = (V_i, T_i, D_i)$ as a directed graph $G = (V_i, T_i)$ where D_i associates each $v \in V_i$ with a duration interval. Therefore, for a pair of adjacent guessed tokens (x_i, v, d) and (x_i, v', d'), we have to decide whether there is (i) a path in G, with possibly repeated vertices/edges, $v_0 \cdots v_n$, with $v_0 \in T_i(v)$ and $v' \in T_i(v_n)$, and (ii) a list of $\mathbb{R}_+$ values $d_0, \ldots, d_n$, such that, for all s, $d_s \in D_i(v_s)$ and $\sum_{i=0}^n d_i$ equals the time elapsed from the end of (x_i, v, d) to the start of (x_i, v', d'). To this aim we guess a set of integers $\{\alpha_{u,v} \mid (u,v) \in T_i\}$ where $\alpha_{u,v}$ is the number of times the path traverses (u,v), and check that they specify a directed Eulerian path from v_0 to v_n (Jungnickel 2013). Such a check is "expressed" as a set of constraints of a *linear problem* (solvable in deterministic polynomial time).

As for the **NP**-hardness, there is a trivial reduction from the problem of existence of a Hamiltonian path in a graph.

Theorem 5.1. *$P = (\{x_1, \ldots, x_k\}, S)$ be a timeline-based planning problem with trigger-less rules only. Checking the existence of a plan for P is an **NP**-complete problem.*

6 Conclusions and Future Work

In this paper, we studied the timeline-based planning problem over dense domains. We proved that it is undecidable in its general form, by arguments similar to those of the standard undecidability proof of satisfiability of Metric Temporal Logic. However, if restricted to trigger-less synchronization rules, the problem is showed **NP**-complete: the proposed decision procedure benefits from an encoding of the problem into timed automata (amenable for exploiting model checking technologies available for that model). Future work will be devoted to studying decidability and complexity issues of "intermediate" variants of the problem, which enforce forms of synchronization rules having expressive power in between that of general rules and trigger-less ones. Moreover, we are interested in studying timeline-based model checking, where systems are described by timelines, and properties are specified in interval temporal logics (e.g., MITL or HS).

Acknowledgments

We would like to acknowledge the fundamental contribution by Gerhard Woeginger to the **NP** algorithm for the case of trigger-less rules.

The work has been supported by the GNCS project *Formal methods for verification and synthesis of discrete and hybrid systems*. The work by A. Molinari and A. Montanari has also been supported by the project *(PRID) ENCASE—Efforts in the uNderstanding of Complex interActing SystEms*.

References

Allen, J. F. 1983. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM* 26(11):832–843.

Alur, R., and Dill, D. L. 1994. A theory of timed automata. *Theoretical Computer Science* 126(2):183–235.

Alur, R., and Henzinger, T. A. 1993. Real-Time Logics: Complexity and Expressiveness. *Information and Computation* 104(1):35–77.

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

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Policella, N. 2007. An Innovative Product for Space Mission Planning: An A Posteriori Evaluation. In *Proc. of ICAPS*, 57–64.

Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-based space operations scheduling with external constraints. In *Proc. of ICAPS*, 34–41.

Cialdea Mayer, M.; Orlandini, A.; and Umbrico, A. 2016. Planning and Execution with Flexible Timelines: a Formal Account. *Acta Informatica* 53(6–8):649–680.

Frank, J., and Jónsson, A. 2003. Constraint-Based Attribute and Interval Planning. *Constraints* 8(4):339–364.

Gigante, N.; Montanari, A.; Cialdea Mayer, M.; and Orlandini, A. 2016. Timelines are Expressive Enough to Capture Action-based Temporal Planning. In *Proc. of TIME*, 100–109.

Gigante, N.; Montanari, A.; Cialdea Mayer, M.; and Orlandini, A. 2017. Complexity of timeline-based planning. In *Proc. of ICAPS*, 116–124.

Jónsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. D. 2000. Planning in Interplanetary Space: Theory and Practice. In *Proc. of ICAPS*, 177–186.

Jungnickel, D. 2013. *Graphs, Networks and Algorithms*. Springer-Verlag Berlin Heidelberg.

Larsen, G. K.; Pettersson, P.; and Yi, W. 1997. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer* 1:134–152.

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

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

A Definition of the value transition function T in the proof of Theorem 3.1

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

- For each instruction label $\ell \in \mathit{Inc} \cup \{\ell_{halt}\}$, let $P_\ell = \emptyset$ if $\ell = \ell_{halt}$, and $P_\ell = \{(\mathit{succ}(\ell), \mathit{inc}_h)\}$ 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:

$$\begin{array}{ll} T(\ell) & = \{(\ell, c_1), (\ell, c_2)\} \cup P_\ell \\ T((\ell, c_1)) & = \{(\ell, c_1), (\ell, c_2)\} \cup P_\ell \\ T((\ell, c_2)) & = \{(\ell, c_2)\} \cup P_\ell \end{array}$$

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

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

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

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

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

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

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

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

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

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

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

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

B Timeline-based planning with trigger-less rules is NP-complete

In this section we describe a timeline-based planning algorithm, for planning problems where only trigger-less rules are allowed, which requires a polynomial number of (non-deterministic) computation steps.

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

Example B.1. Let us consider the following planning problem. 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

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

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 B.2. *The timeline-based planning problem with trigger-less rules is **NP**-hard (even when a single state variable is used).*

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 problem. □

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) problem. That is, if a problem $P = (SV, S)$ 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 by inspecting the timed automaton-based planning algorithm. As a matter of fact, the emptiness checking algorithm for the timed automaton generated from P concludes by an emptiness checking of a Büchi automaton of $g = O(|C| \cdot r \cdot \mathcal{V})$ states (we refer to the proof of Theorem 4.11 for the notation used). For this purpose, it is enough to find a finite word uv, where:

- $|u|, |v| \leq g$,
- there is a run of the Büchi automaton that, from an initial state, upon reading u, reaches a state q, and upon reading v from q reaches a final state and gets ultimately back to q.

Finally, we observe that each transition of the Büchi automaton corresponds to the start point of at least a token in some timeline of (a plan for) P, and at most a token for each timeline (when all these tokens start simultaneously). This yields a bound on the number of tokens, which is $2 \cdot g \cdot |SV|$. We can also derive a bound on the horizon of the plan, which is $2 \cdot g \cdot |SV| \cdot (K+1)$, being K the maximum constant occurring in the planning problem (an upper/lower bound of an interval of a token duration, a time constant in an atom, or an upper/lower bound, u or ℓ, at the subscript of an atom). In fact, every transition taken in the timed automaton may let at most $K+1$ time units pass, as K accounts in particular for the maximum constant to which a (any) clock is compared.[2]

Having this pair of bounds, we are now ready to describe the main phases of the algorithm.

Preprocessing As a preliminary preprocessing phase, we consider all rational values occurring in the input planning problem $P = (SV, S)$—be either upper/lower bounds of an interval of a token duration, a time constant in an atom, or upper/lower bounds (u or ℓ) at the subscript of an atom—and make them integers by multiplying them by the least common multiple γ of all denominators. This involves a quadratic blowup in the input size, being all constants encoded in binary.

It is routine to check that, having a plan for P'—where all values are integers—we can obtain one for the original P, by dividing the start/end times of all tokens in each timeline by γ.

Non-deterministic token positioning The algorithm continues by non-deterministically guessing, for every trigger-less rule in S, 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 $\mathsf{e}_{int}(o_i)$.[3] We observe that all start/end time $\mathsf{s}_{int}(o_i)$ and $\mathsf{e}_{int}(o_i)$, being less or equal to $2 \cdot g \cdot |SV| \cdot (K+1)$ (the finite horizon bound), have an integer part that can be

[2] Clearly, and unbounded quantity of time units may pass, but after $K+1$ the last region of the region graph will certainly have been reached.

[3] 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.

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 not change. This is due to all the constants being integers, as a result of the preprocessing step.[4] Therefore we can now check whether all rules are satisfied.

Enforcing legal token durations and timeline evolutions We now conclude 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,

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

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).

[4] 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.

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

$$\sum_{i=0}^{n} d_i = (\mathsf{s}_{int}(o_j) + \alpha_{o_j,s}) - (\mathsf{e}_{int}(o_i) + \alpha_{o_i,e})$$

  and for all $s = 0, \ldots, n$, $d_s \in D_i(v_s)$.

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 $2 \cdot g \cdot |SV|$. 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 B.3. *(Jungnickel 2013) 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.

However, 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 $\xi s - \eta q - k + \beta_t \leq 0$ and $\xi s - \eta q - k - \beta_t \geq 0$, where β_t is an additional fresh non-negative variable, which is *local* to a single constraint. 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 $r \leq \beta_t$, for each of the introduced variables β_t (i.e., r is less than or equal to the minimum of all β'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's are positive (and thus the original strict inequalities hold true).