Timelines are Expressive Enough to Capture Action-Based Temporal Planning

Nicola Gigante and Angelo Montanari
Dept. of Math, Comp. Science, and Physics
University of Udine, Italy
gigante.nicola@spes.uniud.it
angelo.montanari@uniud.it

Marta Cialdea Mayer
Dept. of Engineering
University of Roma Tre, Italy
cialdea@dia.uniroma3.it

Andrea Orlandini
Inst. of Cognitive Science and Technology
National Research Council
Rome, Italy
andrea.orlandini@istc.cnr.it

***Abstract*—Planning problems are usually expressed by specifying which *actions* can be performed to obtain a given goal. In *temporal* planning problems, actions come with a time duration and can overlap in time, which noticeably increase the complexity of the reasoning process. Action-based temporal planning has been thoroughly studied from the complexity-theoretic point of view, and it has been proved to be EXPSPACE-complete in its general formulation. Conversely, timeline-based planning problems are represented as a collection of variables whose time-varying behavior is governed by a set of temporal constraints, called synchronization rules. Timelines provide a unified framework to reason about planning and execution under uncertainty. Timeline-based systems are being successfully employed in real-world complex tasks, but, in contrast to action-based planning, little is known on their computational complexity and expressiveness. In particular, a comparison of the expressiveness of the action- and timeline-based formalisms is still missing. This paper contributes a first step in this direction by proving that timelines are expressive enough to capture action-based temporal planning, showing as a byproduct the EXPSPACE-completeness of timeline-based planning with no temporal horizon and bounded temporal relations only.**

I. Introduction

Action-based planning languages, such as PDDL [1], [2], represent planning problems from the point of view of the executor by identifying which actions it can choose to perform to reach the given goal. Problems are usually expressed as a set of *fluents*, which describe the world, and actions are characterized by how they change the fluents. When reasoning about automated planning in AI, this has been the mainstream mindset since the beginning of the field, as, for example, in early planning systems such as STRIPS [3], whose heritage is still evident in PDDL.

In addition to be effectively solved by a number of well-known techniques, STRIPS-like *classical* planning has also been studied from a theoretical point of view. The plan existence problem for STRIPS domains has been proved to be PSPACE-complete [4], and it is known that a number of syntactic extensions provided by PDDL, like, for instance, conditional effects, do not increase the complexity and the expressive power of the formalism. Even if we consider *temporally extended goals* [5], supported in PDDL 3 [2], the problem remains PSPACE-complete [6].

Remaining in a deterministic setting, to increase the computational complexity one has to turn to *temporal planning*, where actions are given a *duration* rather than being considered to happen instantaneously. The reasoning is made more complex by the fact that the execution of the actions can now overlap in time, and that the current state may depend on events happened far in the past. In particular, the latter fact makes the problem EXPSPACE-complete, as shown by Rintanen [7], who has also isolated the fragment of temporal planning which is reducible to classical planning.

A different paradigm, called *timeline-based planning*, exists, which models planning problems in a more declarative way than action-based formalisms. It describes the world as a collection of independent, but interacting, components identified by a set of variables, whose behavior over time is constrained by a set of temporal constraints, called *synchronization rules*. This approach was introduced in applications from the space sector [8], where it has been proved very effective in complex real-world tasks [9]–[12]. In these applications, a large number of independent components have to be controlled to reach the goal and to guarantee the satisfaction of operative requirements, and a declarative approach results more natural than the imperative, executor-centric action-based formalisms. Moreover, *flexible* timelines allow one to reason in a unified framework both on planning and execution under uncertainty.

In contrast to action-based planning, theoretical aspects of timeline-based planning were not investigated until very recently. A first formal description of the problem has been proposed in [13], while a formalization with *flexible* timelines appeared in [14], later extended in [15] to account also for controllability issues. Meanwhile, the connection between timelines and Timed Game Automata has been investigated for the purpose of plan verification [16], [17] and robust plan execution [18], [19]. However, a complexity-theoretic characterization of the problem is still missing, and it is not known how this formalism relates to more common action-based ones in terms of expressiveness.

This paper contributes a first step to fill this gap by studying the computational complexity of the plan existence problem for a specific variant of timeline-based planning, which features discrete time, non-flexible timelines, controllable

variables only, tokens of bounded length, and removes the specification of an horizon for solution plans. We compare this fragment to action-based temporal planning *à la* PDDL, showing that even with these simplifications timelines can *capture* action-based temporal planning problems, thus providing a first expressiveness comparison between the two formalisms. This is shown by a polynomial-time reduction from one problem to the other, thus showing that our variant of timeline-based planning is EXPSPACE-hard. We then provide a decision algorithm that runs in exponential space, thus proving it to be EXPSPACE-complete.

The paper is structured as follows. Section II precisely defines the fragment of timeline-based planning we focus on, and it includes some remarks about its syntax that will come useful later. Section III introduces the simple temporal planning language we consider in our comparison, which was already used by Rintanen [7] in his own analysis of temporal planning. Later, Section IV provides the EXPSPACE-hardness result, while Section V shows that the problem belongs to EXPSPACE by providing a suitable decision algorithm. Finally, conclusions in Section VII delineate possible future lines of research on this topic.

II. Timeline-based Planning

In this section, we introduce the basic elements of the timeline-based approach to the planning problem, and define the specific variant of the problem studied in this paper. A formal specification of the general problem has been given in [15], including controllability issues related to temporal uncertainty during actual plan execution. Our work is based on these definitions, adapted and specialized to restrict some aspects in order to isolate the precise definitions needed to capture temporal planning (see Sections IV and V).

As already pointed out, timeline-based planning is a more *declarative* approach to planning than action-based languages. Planning domains are modeled by specifying the possible *behavior* of *state variables*, which represent the main components to be controlled of a system, i.e., logical or physical subsystems whose properties may vary over time.

Definition 1 (State variable)**.** *A state variable x is a triple (V, T, D), where:*

- *V is the* finite domain *of x;*
- *$T : V \to 2^V$ is the* value transition function*, which maps each value $v \in V$ to the set of values that x can take immediately after v;*
- *$D : V \to \mathbb{N} \times \mathbb{N}$ is a function that maps each $v \in V$ to a pair (d_{min}, d_{max}), with $d_{min} \leq d_{max}$, where d_{min} and d_{max} are respectively the minimum and maximum duration of an interval over which x holds the value v.*

In order to specify which values actually takes a variable and for how long, the notion of token has been introduced.

Definition 2 (Token)**.** *Let $x = (V, T, D)$ be a state variable. A* token *for x is a pair (v, d), where $v \in V$ and $d \in \mathbb{N}$ is the* duration *of the token, with $d_{min} \leq d \leq d_{max}$ if $D(v) = (d_{min}, d_{max})$.*

The time-varying behavior of a state variable is represented through a *timeline*.

Definition 3 (Timeline)**.** *A* timeline *for a state variable $x = (V, T, D)$ is a finite sequence $\mathsf{T} = \langle (v_1, d_1), \ldots, (v_k, d_k) \rangle$ of tokens for x, where, for all $i = 1, \ldots, k-1$, $v_{i+1} \in T(v_i)$.*

Notice that we do not require the values v_i and v_{i+1} of x in two consecutive tokens to be different.

For a timeline $\mathsf{T} = \langle (v_1, d_1), \ldots, (v_k, d_k) \rangle$, we define two functions $\mathrm{start\_time}$ and $\mathrm{end\_time}$, that associate an interval with each token in a timeline:

$$\mathrm{start\_time}((v_i, d_i)) = \sum_{j=1}^{i-1} d_j$$
$$\mathrm{end\_time}((v_i, d_i)) = \mathrm{start\_time}((v_i, d_i)) + d_i$$

for $i = 1, \ldots, k$. In the following, we will interchangeably refer to a token and its associated interval when there is no ambiguity. The end time of the *last* token of a timeline is called its *horizon*.

The behavior of the components is constrained by a set of *synchronization rules*, which relate tokens on possibly different timelines through temporal relations among intervals or among intervals and time points. Interval and time point relations that may occur in synchronization rules here are the same as usual timeline-based formulations (*e.g.*, [15, Definition 4]); however, we use a slightly more compact notation. As an example, given two intervals (tokens) a and b, we write $a \leq^{\mathsf{s},\mathsf{s}}_{[l,u]} b$ instead of $a\ \mathrm{starts\_before\_start}_{[l,u]}\ b$. The notation and meaning of all the relations is recalled in Table I (for the sake of brevity, we write s_a for $\mathrm{start\_time}(a)$ and e_a for $\mathrm{end\_time}(a)$). Note that the relations require one to specify lower and upper bounds to the distance between the related points, and that, in contrast to usual formulations of timeline-based planning, the bounds have to be finite (*i.e.*, bounds of the form $[i, +\infty]$ are not allowed). This is an important syntactic feature of the formalism that has major consequences on the complexity of the planning problem, as will be shown in Sections IV and V.

These basic interval relations allow us to define a bounded version of most of Allen's interval relations [20]. One of these, equality between two intervals, will be used extensively in the following sections, so if a and b are two tokens, we define $a = b$ as $a \leq^{\mathsf{s},\mathsf{s}}_{[0,0]} b \wedge a \leq^{\mathsf{e},\mathsf{e}}_{[0,0]} b$. Moreover, the *starts at* and *ends at* time point relations are defined as:

$$a =^{\mathsf{s}} t \ \text{ means } \ a \leq^{\mathsf{s}}_{[0,0]} t$$
$$a =^{\mathsf{e}} t \ \text{ means } \ a \leq^{\mathsf{e}}_{[0,0]} t$$

Definition 4 (Synchronization rules)**.** *Let Σ be an alphabet of symbols, called* token names*. An* atom *is an expression of the form $a\ \rho\ b$, where a and b are token names and ρ is one*

The relation	holds if
$a \leq^{\mathsf{s},\mathsf{s}}_{[l,u]} b$	$l \leq s_b - s_a \leq u$
$a \leq^{\mathsf{e},\mathsf{e}}_{[l,u]} b$	$l \leq e_b - e_a \leq u$
$a \leq^{\mathsf{s},\mathsf{e}}_{[l,u]} b$	$l \leq e_b - s_a \leq u$
$a \leq^{\mathsf{e},\mathsf{s}}_{[l,u]} b$	$l \leq s_b - e_a \leq u$
$a \leq^{\mathsf{s}}_{[l,u]} t$	$l \leq t - s_a \leq u$
$a \geq^{\mathsf{s}}_{[l,u]} t$	$l \leq s_a - t \leq u$
$a \leq^{\mathsf{e}}_{[l,u]} t$	$l \leq t - e_a \leq u$
$a \geq^{\mathsf{e}}_{[l,u]} t$	$l \leq e_a - t \leq u$

Table I
INTERVAL AND TIME POINT RELATIONS, BETWEEN INTERVALS $a = (s_a, e_a)$ AND $b = (s_b, e_b)$, AND TIME POINT $t \in \mathbb{N}$. BOUNDS l AND u ARE GIVEN TO EACH RELATION.

of the interval relations of Table I. An existential statement *is a statement of the form:*

$$\exists a_1[x_1 = v_1] \dots a_n[x_n = v_n] \,.\, \mathcal{C}$$

where $\mathcal{C}$ is a conjunction of atoms, $a_1, \dots, a_n$ are token names, $x_1, \dots, x_n$ are state variables, and $v_1, \dots, v_n$ are possible values of $x_1, \dots, x_n$, respectively.

A synchronization rule *is a clause of the form:*

$$a_0[x_0 = v_0] \longrightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \dots \vee \mathcal{E}_k, \quad \textit{or}$$
$$\top \longrightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \dots \vee \mathcal{E}_k$$

where a_0 is a token name, x_0 a state variable, v_0 a value from the domain of x_0, and $\mathcal{E}_1, \dots, \mathcal{E}_k$ are existential statements *where only the token name a_0 may appear free.*

The $a_0[x_0 = v_0]$ part is called the trigger*, and rules of the second form are called* trigger-less*.*

A formal account of the semantics of synchronization rules can be found in [15], but their meaning is intuitively very simple. The left part (the *trigger*) is a sort of universal quantifier, which says that *for all* the tokens a_0 where the variable x_0 holds 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 $a_1, \dots, a_n$ where the respective state variables hold the specified values, and that satisfy the interval relations specified by $\mathcal{C}$. The trigger-less form just asserts the satisfaction of the existential statements.

Note that the purpose of allowing only finite bounds in interval relations is to ensure that each synchronization rule can only talk about a bounded number of time steps around the triggering token, a fact that will play a crucial role in Section V. For the same reason, the maximum duration of tokens specified by the duration function in Definition 1 must be finite, and the synchronization rules are constrained to be *connected*, as formally stated by the following definition.

Definition 5. *Let R be a* synchronization rule*, a_0 be the token name used in the trigger, if any, and $a_1, \dots, a_k$ be all the token names used in the existential statements of R. Let $G_R = (V, E)$ be an undirected graph, where $V = \{a_0, a_1, \dots, a_k\}$ and E contains an edge $\{a_i, a_j\}$ iff there exists at least one existential statement that contains a clause of the form $a_i \leq^{\mathsf{e}_1,\mathsf{e}_2}_{[l,u]} a_j$ or $a_j \leq^{\mathsf{e}_1,\mathsf{e}_2}_{[l,u]} a_i$, with $\mathsf{e}_1, \mathsf{e}_2 \in \{\mathsf{s}, \mathsf{e}\}$. Then, R is said to be* connected *if G_R is a* connected *graph.*

In a formulation of timeline-based planning like the one in [15], that admits unbounded interval relations, disconnected rules can be trivially rewritten into connected ones. In our formulation this is not possible, and they cause the same issues that lead to the exclusion of unbounded interval relations (see Sections V and VI for details). This restriction is technically not essential for *trigger-less* rules, but disconnected trigger-less rules can be made connected by simply splitting up the connected parts into separate rules, so we assume this restriction for all kinds of rule for uniformity.

A timeline-based planning domain is specified by the set of state variables and the set of synchronization rules representing the admissible behaviors of the involved components. As shown in [15], initial conditions and the goal of the problem can be expressed directly as a set of trigger-less rules, so here we do not treat them differently from other kinds of rules (i.e., we consider them included in the set of synchronization rules S), which leads to a simple definition of planning problem.

Definition 6 (Planning problem)**.** *A timeline-based planning* problem *is a pair $\mathcal{P} = (SV, S)$, where SV is a set of state variables and S is a set of connected synchronization rules defined over variables in SV.*

Definition 7 (Solution plan)**.** *A* solution plan *π for a timeline-based planning problem $\mathcal{P} = (SV, S)$ is a set of timelines, one for each state variable in SV, such that every synchronization rule in S is satisfied.*

Without loss of generality, we can suppose that all the timelines in the solution have the same horizon, that will be the horizon of the whole solution.[1]

Some syntactic features of synchronization rules as defined in Definition 4 can be treated as syntactic sugar, so that their core syntax can be considered simpler. One of these possible simplifications, which will come useful to provide the decision procedure for the problem in Section V, is that any problem $\mathcal{P}$ can be rewritten using only binary state variables, as stated by the following proposition, which is proved in the Appendix.

Proposition 1. *Every timeline-based planning problem can be rewritten, with at most a polynomial increase in size, into an equivalent one that only uses binary state variables.*

[1] If this is not the case, one can add a *don't care* value to the variables domains of a suitable maximum duration.

The reader may have noticed that this definition is far simpler than the corresponding one from [15]. Despite this simplicity, we will show that the problem is still very expressive and computationally hard. In particular, we are concerned with deterministic domains where there is no kind of uncertainty with regard to the environment behavior. In other words, there are no *uncontrollable* components and there is no temporal *flexibility*. This restriction is inline with our aim to compare the problem with *deterministic* temporal planning. Furthermore, while usual formulations require the specification of a finite bound to the horizon of acceptable solutions, we do not impose any such bound. It is worth to note that solution plans are still *finite*, even if we do not put a limit on their length. This is because of the definition of timeline which is a *finite* sequence of tokens. We are not considering a semantics that allows infinite models. Finally, as already noted above, we consider only finite bounds for the interval relations used in synchronization rules.

III. Action-Based Temporal Planning

In this section, we briefly recall the definition of the action-based temporal planning language that will be used in the following comparison with the timeline-based formalism given in Section II. This language has been introduced by Rintanen in [7] as a simpler and formally cleaner equivalent to commonly used temporal planning languages such as PDDL 2 [1]. It can be thought of as an extension of the classical planning language, where preconditions can involve arbitrary points in the past rather than the current one only. In the following, we will refer to problems in this language simply as *temporal planning problems*.

Definition 8 (Precondition formula)**.** *Let Σ be a set of proposition letters. A* precondition formula *over Σ is recursively defined by the following syntax:*

$$\phi := \sigma \in \Sigma \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid [i,j]\psi,$$

where ψ, ψ_1, and ψ_2 are precondition formulae and $i, j \in \mathbb{Z}$, with $i \leq j$. We denote by $\mathcal{L}$ the set of all the precondition formulae.

Precondition formulae pair the connectives of propositional logic with an additional temporal operator $[i, j]$ that, when applied to a formula ϕ, that is, $[i, j]\phi$, states that ϕ must hold in all time points *from i to j steps from now*. The formula $[i, i]\phi$ is written in short as $[i]\phi$. An additional constraint forces precondition formulae to talk about the present and the past only. Hence, while the values i and j of a subformula $[i, j]\phi$ can be greater than 0, the cumulative effect of the temporal operators occurring in a precondition formula must be less than or equal to 0. As an example, the formula $[-5](p \wedge [3]q)$ contains the subformula $[3]q$, but it refers to time points in the past only. It is worth to note that the semantics originally defined for this language consider in a unusual way the past of the *first* time step, so that $[i]p$, for $i < 0$, when valuated at time zero, valuates to the same truth value of p.

Definition 9 (Temporal planning problem)**.** *A* temporal planning problem *is a tuple (A, I, O, R, D, G), where:*

- *A is a finite set of proposition letters, called* state variables*;*
- *I is a valuation of A that represents the* initial state*;*
- *O is a finite set of proposition letters, called* actions *(or* operators*);*
- *$R : O \to \mathcal{L}$ is a function that maps each action to its* precondition*;*
- *D is a finite set of rules of the form (P, E), where P is a precondition formula and E is a set of literals over A, that is, p or $\neg p$ with $p \in A$;*
- *$G \in \mathcal{L}$ is a precondition formula that specifies the* goal *condition.*

Intuitively, a rule (P, E) states that whenever P holds at time point i, the literals in E become true at time point $i+1$, while all the other proposition letters preserve their truth. An action o is applicable (the proposition letter can be true) at time point i if and only if its precondition $R(o)$ is true at i.

Definition 10 (Solution plan)**.** *A* solution plan *for a temporal planning problem (A, I, O, R, D, G) is a sequence of actions, which satisfy the action preconditions R and the rules D of the problem, that, when applied from the initial state I, lead to a state satisfying the goal condition G.*

The following proposition holds [7].

Proposition 2. *Let P be a temporal planning problem. Establishing whether there exists a solution plan for P is an* EXPSPACE*-complete problem.*

As with the timeline-based formalism introduced in Section II, we will benefit from a simplification of the syntax which can be applied without changing its complexity and expressive power. In particular, the following proposition, whose proof can be found in the Appendix, shows that temporal operators of the form $[i]\phi$ are sufficient to express any temporal formula of the extended form $[i, j]\phi$.

Proposition 3. *Any temporal planning problem can be rewritten, with at most a polynomial increase in size, into an equivalent one that only makes use of temporal formulae of the form $[i]\phi$.*

IV. Hardness of Timeline-Based Planning

We now prove the EXPSPACE-hardness of the plan existence problem for timeline-based planning via a polynomial-time reduction from the same problem for the action-based temporal planning.

Theorem 1. *Given a timeline-based planning problem $\mathcal{P}$, establishing whether there exists a solution for $\mathcal{P}$ is an* EXPSPACE*-hard problem.*

Proof: Let $P = (A, I, O, R, D, G)$ be an action-based temporal planning problem. By Proposition 3, without loss of generality, we can assume that all the temporal operators that appear in precondition formulae for rules in D and all action preconditions in R are of the form $[i]\phi$. Moreover, we can assume that all the aforementioned formulae are in *negated normal form*, and that all the temporal operators are pushed down to literals.[2] We will build an equivalent timeline-based problem $\mathcal{P}$ that has a solution if and only if a solution exists for P.

Let F be a set of formulae built as follows:

- $\phi \in F$ for each *subformula* ϕ of each precondition formula from the rules in D and of each precondition $R(o)$, with $o \in O$;
- for each $p \in A \cup O$, $p \in F$ and $\neg p \in F$;
- for each $p \in A$, $[\pm 1]p \in F$ and $[\pm 1]\neg p \in F$;
- for each rule $(P, E) \in D$, $[-1]P \in F$;
- for each formula $\phi \in F$, the negated normal form of its negation is in F, *i.e.*, $\mathrm{NNF}(\neg\phi) \in F$.

The set of state variables for $\mathcal{P}$ contains a state variable x_ϕ for each $\phi \in F$. Each of these state variables is *Boolean* (*i.e.*, its domain is the set $\{0, 1\}$), and its duration is fixed to a unitary length, that is, $D(v) = [1, 1]$ for each $v \in \{0, 1\}$. The transition function does not impose any constraint, so $T(v) = \{0, 1\}$ for $v \in \{0, 1\}$.

For each $p \in A \cup O$, the value of state variables x_p will describe the valuation of the temporal planning variables and actions at a given time point, corresponding to the starting point of the token interval. A set of suitable synchronization rules will ensure that each x_ϕ state variable will be true (resp., false) only when the corresponding formula ϕ would be true (resp., false) given the truth values of literals. As an example, for each formula $\phi \wedge \psi$ appearing in F, there will be a rule of the following form:

$$a[x_{\phi\wedge\psi} = 1] \longrightarrow \exists b[x_\phi = 1]c[x_\psi = 1] \;.\; a = b \wedge a = c$$
$$a[x_{\phi\wedge\psi} = 0] \longrightarrow \exists b[x_{\mathrm{NNF}(\neg(\phi\wedge\psi))} = 1] \;.\; a = b$$

The first rule above ensures that whenever we have an interval where $\phi \wedge \psi$ holds, then both ϕ and ψ hold over that interval. The second rule handles the case where the formula is false, and it delegates the work to the rules governing the variable for its negation. The negated formula does not appear directly because all the formulae in F are in negated normal form. This means that a rule to handle negated formulae is not needed. Negations are instead handled at the bottom level on literals, with rules connecting the tokens of x_p, for each letter p, with the tokens of its negation $x_{\neg p}$. So for each literal ℓ over letters $p \in A \cup O$ we have:

$$a[x_\ell = 1] \longrightarrow \exists b[x_{\bar{\ell}} = 0] \;.\; a = b$$
$$a[x_\ell = 0] \longrightarrow \exists b[x_{\bar{\ell}} = 1] \;.\; a = b$$

[2]The NNF is easily obtained as in propositional logic, with the observation that $\neg[i]\phi \equiv [i]\neg\phi$. To push down temporal operators, observe that both $[i](\phi \vee \psi) \equiv [i]\phi \vee [i]\psi$ and $[i][j]\phi \equiv [i + j]\phi$.

The rules for disjunctions are symmetrical to conjunctions:

$$a[x_{\phi\vee\psi} = 1] \longrightarrow \exists b[x_\phi = 1] \;.\; a = b \;\vee\; \exists b[x_\psi = 1] \;.\; a = b$$
$$a[x_{\phi\vee\psi} = 0] \longrightarrow \exists b[x_{\mathrm{NNF}(\neg(\phi\vee\psi))} = 1] \;.\; a = b$$

The last kind of formula to handle is the temporal operator. For a formula $[i]\ell$, the rules have to ensure that whenever the corresponding variable is true in an interval, then ℓ holds at i time steps *after* that point (or *before* if i is negative). This is easily expressed as:

$$a[x_{[i]\phi} = 1] \longrightarrow \exists b[x_\phi = 1] \;.\; a \leq^{\mathsf{s},\mathsf{s}}_{[i,i]} b$$

if $i \geq 0$. Otherwise, if i is negative, the rule is similar but has to treat in a slightly different way the case where the triggering token is near the start of the timeline and the temporal operator predicates before time step zero, because of how the semantics of the target language treats time points before the first. Thus, in this case the rule is the following:

$$a[x_{[i]\ell} = 1] \longrightarrow \exists b[x_\ell = 1] \;.\; b \leq^{\mathsf{s},\mathsf{s}}_{[i,i]} a \;\vee\; \exists b[x_\ell = 1] \;.\; a = b \wedge b \leq^{\mathsf{s}} i$$

With this infrastructure in place, the timelines of the problem now encode the truth of all the formulae that can possibly be useful to handle the original planning problem, so it is possible to encode the rules of the problem itself. Recall that each rule (P, E) specifies that every time the precondition P is satisfied, literals in E must be true at the next step. This is equivalent to say that every time P is satisfied, the formula $\bigwedge_{\ell\in E}[1]\ell$ holds, and it can thus be expressed as follows, where $\ell_1, \ldots, \ell_n \in E$:

$$a[x_P = 1] \longrightarrow \exists a_1[x_{[1]\ell_1} = 1] \ldots a_n[x_{[1]\ell_n} = 1] \;.\; a = a_1 \wedge \cdots \wedge a = a_n$$

Since we are encoding a *deterministic* planning problem, it is also implicit that every literal not explicitly changed by a rule has to preserve its truth value. Additional synchronization rules are required to ensure this *inertia*. These rules say that if a literal holds a given value at the current time point, it either had the same value at the previous step, or a precondition of some rule involving it was true, causing its change. A special case is needed for the first time point, which has not a predecessor. In detail, for every literal ℓ over A, let $P_1, \ldots, P_n$ be the preconditions of all the rules whose effects include ℓ. Then, the inertia for literal ℓ can be expressed as follows:

$$a[x_\ell = 1] \longrightarrow \bigvee \begin{cases} \exists b[x_\ell = 1] \;.\; a = b \wedge b =^{\mathsf{s}} 0 \\ \exists b[x_{[-1]\ell} = 1] \;.\; a = b \\ \bigvee_{i=1}^{n} \exists b[x_{[-1]P_i} = 1] \;.\; a = b \end{cases}$$

In a similar way, it is possible to encode preconditions of actions, so that when actions are performed their preconditions are ensured to hold. For each action $o \in O$, we have:

$$a[x_o = 1] \longrightarrow \exists b[x_{R(o)} = 1] \,.\, a = b$$

At this point, the domain of the problem is completely encoded by the synchronization rules of the timeline-based domain, and it is now sufficient to express the initial state and the goal condition. Let $\ell_1, \ldots, \ell_n \in I$ be the literals asserted by the initial state, and G be the formula that describes the goal condition. They are encoded by the following trigger-less synchronization rules:

$$\begin{aligned} \top \longrightarrow \exists a_1[x_{\ell_1} = 1] \ldots a_n[x_{\ell_n} = 1] \,. \\ a_1 =^{\mathsf{s}} 0 \wedge \ldots \wedge a_n =^{\mathsf{s}} 0 \\ \top \longrightarrow \exists a[x_G = 1] \,.\, \top \end{aligned}$$

This step completes the encoding. The timelines for the variables x_o, with $o \in O$, from a solution plan to this timeline-based problem, encode the solution plan for the original action-based problem. Plan existence for the original problem is thus reduced to plan existence for the timeline-based one. Moreover, it can be seen that the size of the encoded problem grows polynomially with the size of the input, since it is proportional to the number of state variables x_ϕ, and $|F|$ is linear in the size of all the precondition formulae. Moreover, the encoding can be straightforwardly computed in polynomial time, thus providing a polynomial time reduction.

This completes the proof that the plan existence problem for timeline-based planning is EXPSPACE-hard. ■

V. Complexity of Timeline-Based Planning

This section proves the existence of a decision procedure for timeline-based planning problems which runs in exponential space, thus proving that the problem belongs to the EXPSPACE complexity class.

The proof will involve a few steps, but the reasoning is quite standard. We will first provide an upper bound to the length of solutions, then we will show that a *nondeterministic* Turing machine can find a solution plan using an exponential amount of space, and finally we employ the well-known fact that EXPSPACE = NEXPSPACE to assert that there exists an equivalent deterministic algorithm with the same space complexity.

Definition 11 (Window)**.** *Consider a problem $\mathcal{P} = (SV, S)$. Let k be the product of all the numeric values greater than zero that appear as bounds for the duration functions of state variables and* interval *and* time point *relations used in rules from S. Then, the* window *of $\mathcal{P}$, denoted $w(\mathcal{P})$, is equal to:*

$$w(\mathcal{P}) = 2k + 1$$

Intuitively, $w(\mathcal{P})$ is an upper bound to the maximum number of time steps from the start (or the end) of a token that can affect the satisfaction of a synchronization rule triggered by that token, *i.e.*, k in the past and k in the future. Such a constraint can be understood by observing that the scope of the existential statements of a rule is limited by the bounds used in interval relations, and thus the rule cannot be affected by something happening at a greater distance. Moreover, it can be easily seen that $k < 2^n$, and then:

$$w(\mathcal{P}) < 2^{n+1} + 1 < 2^{2n}$$

Note that $w(\mathcal{P})$ is also the maximum length of the *initial* segment of the solution affected by time point relations.

In the following, let $\mathcal{P} = (SV, S)$ be a timeline-based planning problem, let n be the size of the input description of $\mathcal{P}$, and let w denote the window $w(\mathcal{P})$. By Proposition 1, without loss of generality, we can assume that $\mathcal{P}$ consists of *binary* state variables only.

A solution plan π for $\mathcal{P}$ is a set of timelines, but we need a way to represent it as a string in order to reason in precise terms about its space occupation. This *flattened* representation consists of a sequence of words of $|SV|$ bits. Let us call those words *state words*. The state word at position i in the sequence represents the truth values of the state variables at time i. The following lemma provides an upper bound to the length of such sequences.

Lemma 1. *Consider a timeline-based planning problem $\mathcal{P}$ that only makes use of binary variables. If $\mathcal{P}$ has a solution, then it also has a solution π with an horizon of at most:*

$$2^{2^{n^3}}$$

Proof: Let π be a solution for $\mathcal{P}$, and consider subsegments of π of length w. Thus, there are at most $2^{|SV|}$ possible state words at each time step, so there are at most $w^{2^{|SV|}}$ possible sequences of w state words. Thus, if π is longer than the following amount of time steps

$$w^{2^{|SV|}} \cdot w^{2^{|SV|}} = w^{2^{|SV|+1}}$$

there must be some segment of length $w^{2^{|SV|}}$ that repeats at least twice in the sequence. Thus, there are two time steps $i > w$ and $j > i$ such that the windows of w time steps centered on them is equal. Suppose for now that there are no trigger-less rules in $\mathcal{P}$. Then, if we *remove* from π all the time steps between $i+1$ and j, we obtain a shorter sequence π' which is still a valid solution plan. To see this, observe that there cannot be any synchronization rule triggered by a token before the position $i - w$ or after $j + w$ and satisfied by something happened inside the cut segment. On the other hand, any synchronization rule satisfied by tokens *inside* the segments $(i \ldots i + w)$ and $(j - w \ldots j)$ are still satisfied by the repeated copies $(j \ldots j + w)$ and $(i - w \ldots i)$. Finally, the satisfaction of synchronization rules that use time point relations is preserved by the fact that $i > w$. By iterating this contraction, we can find a solution shorter than $w^{2^{|SV|+1}}$. Now consider again the presence of trigger-less rules. The

bound above is insufficient to guarantee the safety of the contraction, because the part of the cut segment between $i+w$ and $j-w$, may contain some token that was essential to the satisfaction of a trigger-less rule. Let us call this kind of token *goal tokens*, and let h be the number of trigger-less rules of the problem. Observe that, since rules are *connected*, and the two repeating windows are distant more than w time steps, all the tokens involved in the satisfaction of a single trigger-less rule must be inside the potentially cut segment. We cannot cut such a segment, but it is sufficient to wait for other h pairs of repeated windows to be sure to find a repetition without any occurrence of a goal tokens between them, or with an occurrence of goal tokens that satisfy a rule that was already satisfied before, thus cutting that segment preserves the validity of the plan. Thus, if π is longer than $h \cdot w^{2^{|SV|+1}}$, we can find a shorter plan π' which is still a valid solution. Now recall that $w < 2^{2n}$, $|SV| < n$, and $h < n$, so we can contract any plan π longer than:

$$h(2^{2n})^{2^{|SV|+1}} < n2^{n2^{n+2}} \leq 2^{2^{n^3}}$$ ■

To find a solution, a nondeterministic Turing machine can proceed by *guessing* a sequence of state words and checking if it satisfies all the synchronization rules. Since single synchronization rules cannot affect more than w time steps, and $w \in \mathcal{O}(2^n)$, it is possible to execute this *generate-and-check* procedure using only an exponential amount of space. So we can now prove the result stated at the beginning of the section.

Theorem 2. *Given a timeline-based planning problem $\mathcal{P}$, the problem of establishing whether there exists a solution for $\mathcal{P}$ belongs to* EXPSPACE.

Proof: A decision procedure will be provided which can be executed by a *nondeterministic* Turing machine using only an exponential amount of space. In particular, the following pieces of information will be maintained during the execution:

- A subsequence of w state words corresponding to the *current* piece of solution under testing. Such a subsequence can be represented in $w \log |SV| \in \mathcal{O}(2^n)$ bits.
- A subsequence of $\lceil w/2 \rceil$ state words corresponding to the initial segment of the solution, to be able to check synchronization rules that involve time point relations.
- A counter of the current position i at the center of the window (note that w is odd), which has to count up to $2^{2^{n^3}} - \lceil w/2 \rceil$, and thus has to use at most an exponential amount of bits.
- A set of $|SV|$ counters to take care of the duration of the started tokens, to ensure their duration matches the duration function of each variable. These counters use a polynomial amount of bits as they have to count up to w time steps.
- A polynomial amount of bits to count how many trigger-less rules have still to be satisfied.

Here we have supposed, as allowed by Proposition 1, that $\mathcal{P}$ only makes use of binary variables. Then, the exponential-space decision procedure is the following:

1) At start, an *initial* segment of $\lceil w/2 \rceil$ state word is guessed, and set to also be the current window.
2) The counter i is set to zero and the following steps are repeated until i stays less than $2^{2^{n^3}} + 1$:

2.a) If some bits at position i changed since the previous step (at $i = 0$ is as if all bits had changed), and thus some token has started, the corresponding triggered synchronization rules are checked by searching for the satisfying tokens in the current window, and in the initial segment if some time point relation is used.

2.b) If some rule cannot be satisfied, the nondeterministic Turing machine *rejects* the input and terminates.

2.c) If all the triggered rules are satisfied, then the window is checked to look for any satisfied trigger-less rule. If all the trigger-less rules are satisfied, the machine *accepts* the input and terminates. Otherwise, the counter of satisfied trigger-less rules is updated and the execution continues.

2.d) The current position i is incremented and, if $i < 2^{2^{n^3}} - \lceil w/2 \rceil$, the window is advanced by guessing the next state word at position $i + \lceil w/2 \rceil$. If $i > \lfloor w/2 \rfloor$, the last state word at $i - \lfloor w/2 \rfloor$ is forgotten. The current window is thus used as a LIFO queue structure. The new state word is guessed as to satisfy any constraint imposed by the transition and duration functions of the state variables.

3) If not all the trigger-less rule have been satisfied, the machine *rejects* the input and terminates.

Since the provided procedure runs in nondeterministic exponential space and EXPSPACE = NEXPSPACE by Savitch's theorem, we can deduce that a *deterministic* Turing machine exists that can decide whether the given problem $\mathcal{P}$ has a solution in deterministic exponential space. ■

As a consequence of this result and of Theorem 1, we can state the following:

Corollary 1. *Given a timeline-based planning problem $\mathcal{P}$, establishing whether there exists a solution for $\mathcal{P}$ is an* EXPSPACE*-complete problem.*

VI. Discussion

Theorems 1 and 2 prove the EXPSPACE-completeness of the timeline-based planning problem defined in Section II. The hardness result comes from a many-to-one reduction from an action-based temporal planning language that preserves solutions, thus providing also an expressiveness result. In particular, it tells us that our formulation of timelines is expressive enough to capture action-based temporal plan-

ning. In light of these results, a few observations can be made to better understand the picture.

With regard to general timelines formulations, *e.g.*, [15], even if we exclude any form of environmental uncertainty and temporal flexibility, our setting is simplified by a number of syntactic restrictions. In particular, we consider synchronization rules that only use bounded interval relations, while general models usually allow to use interval relations like $\leq_{[0,+\infty]}$, thus being able to express usual unbounded Allen's relations as a particular case. Tokens are similarly mandatorily constrained to have a maximum duration. These restrictions turned out to be essential to our complexity result, as the argument for the proof of Lemma 1 does not hold if we admit synchronization rules that could escape the exponential window of time steps around a single token. It is interesting to learn that a form of timeline-based planning so simplified is still computationally hard and expressive enough to capture action-based temporal planning.

Even if syntactically simplified, our formulation is not a proper restriction of the usual timeline-based models because of the removal of the finite *horizon* from the problem specification. This relaxation plays a crucial role in the *hardness* proof of Theorem 1. That is because, in other formulations, the horizon is specified as *part of the input*, so at most an exponentially large horizon of 2^n time steps can be specified with an input of size n. On the other hand, our target action-based language does not constrain the length of its solutions in any way, and can express solutions of length at most doubly exponential (see [7, Theorems 9 and 10]). This means that a timeline-based formalism with a specified horizon would always loose a whole class of solutions longer than its exponential horizon, unless we allow an exponential growth of the problem description or we consider the horizon to be specified in the input as an exponent (*i.e.*, if the input specifies m then horizon is 2^m). From the expressiveness point of view, this means that a horizon-constrained timeline-based formulation would still be able to capture action-based temporal planning, exactly as in the proof of Theorem 1, but only if a doubly exponential horizon is specified.

A natural question would be whether the converse of Theorem 1 is true, *i.e.*, that our variant of timeline-based planning is not only polynomial-time reducible to action-based temporal planning, which follows from the EXPSPACE-completeness result, but that it is so in a way that is able to preserve solutions and thus to provide an expressiveness equivalence between the two formalisms. While we cannot provide a definitive answer to this question, any attempt by our part so far has failed to encode timeline-based planning into action-based formalisms because of the ability of synchronization rules to *mention* the same time point *multiple times* (*e.g.*, in a clause like $b \subseteq a \wedge c \subseteq a$, where $\subseteq$ is the *contained by* Allen's relation). The attempt to encode this syntactic feature, which is typically found in hybrid temporal logics (see *e.g.*, [21]), into a language that lacks it causes an exponential blowup in the size of formulae because of the need to explicitly enumerate the different linearization of the constraints where the same interval is mentioned multiple times (in the example above, all the possible overlaps between b and c, which both have to be contained in a but are unrelated to each other). This suggests that the action-based temporal language that we studied in this paper might be able to express timeline-based problems, but at the cost of an exponential growth.

VII. Conclusions and Future Work

While timeline-based planning has been successfully employed in complex real-world scenarios, a complete complexity-theoretic analysis of it is still missing, as is a comparison with more common action-based formalisms in terms of expressive power. This paper is a first step in this direction: we identified a simplified formulation of timeline-based planning that is expressive enough to capture action-based temporal planning, and we studied its computational complexity by proving its EXPSPACE-completeness.

Timeline-based planning as formulated in Section II is greatly simplified with regard to usual formulations such as the one from [15], so the natural development of this work will be to study the more complex formulations from a complexity-theoretic point of view, continuing the comparison with action-based formalisms. Interesting extensions that still need to be studied include, but are not limited to: timelines with infinite bounds on interval relations and tokens duration; *flexible* timelines with or without *uncontrollable components*; timelines and flexible timelines over *dense* and/or *continuous* flow of time.

Acknowledgments

The work has been partially supported by the ¡ℕδA‴ GNCS project "Logic, Automata, and Games for Auto-Adaptive Systems".

References

[1] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[2] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, "Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners," *Artificial Intelligence*, vol. 173, no. 5-6, pp. 619–668, 2009.

[3] R. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," in *Proc. of 2nd IJCAI*, 1971, pp. 608–620.

[4] T. Bylander, "The Computational Complexity of Propositional STRIPS Planning," *Artificial Intelligence*, vol. 69, no. 1-2, pp. 165–204, 1994.

[5] F. Bacchus and F. Kabanza, "Planning for Temporally Extended Goals," *Annals of Mathematics in Artificial Intelligence*, vol. 22, no. 1-2, pp. 5–27, 1998.

[6] G. De Giacomo and M. Y. Vardi, "Automata-Theoretic Approach to Planning for Temporally Extended Goals," in *Proc. of 5th ECP*, 1999, 226–238.

[7] J. Rintanen, "Complexity of Concurrent Temporal Planning," in *Proc. of 17th ICAPS*, 2007, pp. 280–287.

[8] N. Muscettola, "HSTS: Integrating Planning and Scheduling," in *Intelligent Scheduling*, M. Zweben and M. S. Fox, Eds., Morgan Kaufmann, 1994, ch. 6, pp. 169–212.

[9] A. K. Jónsson, P. H. Morris, N. Muscettola, K. Rajan, and B. D. Smith, "Planning in Interplanetary Space: Theory and Practice," in *Proc. of 5th AIPS*, 2000, pp. 177–186.

[10] J. Frank and A. Jónsson, "Constraint-Based Attribute and Interval Planning," *Constraints*, vol. 8, no. 4, pp. 339–364, 2003.

[11] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and N. Policella, "An Innovative Product for Space Mission Planning: An A Posteriori Evaluation," in *Proc. of 17th ICAPS*, 2007, pp. 57–64.

[12] S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, and S. Frye, "Timeline-based space operations scheduling with external constraints," in *Proc. of 20th ICAPS*, 2010, pp. 34–41.

[13] A. Cimatti, A. Micheli, and M. Roveri, "Timelines with Temporal Uncertainty," in *Proc. 27th AAAI*, 2013.

[14] M. Cialdea Mayer, A. Orlandini, and A. Umbrico, "A Formal Account of Planning with Flexible Timelines," in *Proc. of 21st TIME*, 2014, pp. 37–46.

[15] M. Cialdea Mayer, A. Orlandini, and A. Ubrico, "Planning and Execution with Flexible Timelines: A Formal Account," *Acta Informatica* (published online, to appear on paper), 2015.

[16] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci, "Flexible Timeline-Based Plan Verification," in *Proc. of 32nd KI*, 2009, pp. 49–56.

[17] ——, "Analyzing Flexible Timeline-Based Plans," in *Proc. of 19th ECAI*, 2010, pp. 471–476.

[18] A. Orlandini, M. Suriano, A. Cesta, and A. Finzi, "Controller Synthesis for Safety Critical Planning," in *Proc. of 25th IEEE ICTAI*, 2013, pp. 306–313.

[19] M. Cialdea Mayer and A. Orlandini, "An Executable Semantics of Flexible Plans in Terms of Timed Game Automata," in *Proc. of 22nd TIME*, 2015, pp. 160–169.

[20] J. F. Allen, "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[21] P. Blackburn and M. Tzakova, "Hybrid Languages and Temporal Logic," *Logic Journal of the IGPL*, vol. 7, no. 1, pp. 27–54, 1999.

APPENDIX

Proposition 1. *Every timeline-based planning problem can be rewritten, with at most a polynomial increase in size, into an equivalent one that only uses binary state variables.*

Proof: Let $\mathcal{P} = (SV, S)$ be a timeline-based planning problem. Let $x = (V, T, D)$ be a state variable with $k = |V|$ and $v_0, \ldots, v_{k-1}$ its possible values from V. We will now construct an equivalent problem $\mathcal{P} = (SV', S')$ where x has been substituted by a suitable number of binary state variables $x_0 = (V_0, T_0, D_0), \ldots, x_{k-1} = (V_{k-1}, T_{k-1}, D_{k-1})$, *i.e.*, one for each possible value that x can hold. Synchronization rules are then introduced to ensure that $x_i = 1$ will hold in a solution or $\mathcal{P}'$ if and only if $x = v_i$ would have held in a solution for $\mathcal{P}$. As a first step, a set of rules is introduced to ensure mutual exclusion between the binary variables, so for all $v_i \in V$:

$$a_i[x_i = 1] \longrightarrow \exists \overbrace{a_0[x_0 = 0] \ldots a_k[x_k = 0]}^{\text{except } a_i} \,.\, \bigwedge_{\substack{j=0 \\ j \neq i}}^{k} a_i = a_j$$

Transitions and duration functions T and D have to be transferred to the new variables. The minimum an maximum duration for each value can be replicated directly to the duration of the positive value of each variable, so $D_i(1) = (d^i_{min}, d^i_{max})$ where $(d^i_{min}, d^i_{max}) = D(v_i)$. The duration function for the negative value has to encompass the possible duration of all the other values, so we will put $D_i(0) = (d_{min}, d_{max})$ for each $i = 0, \ldots, k-1$, where:

$$d_{min} = \min_{i=0,\ldots,k-1} \{d^i_{min}\}$$
$$d_{max} = \max_{i=0,\ldots,k-1} \{d^i_{max}\}$$

The transition function has to be encoded through the use of additional synchronization rules. So for each $i = 0, \ldots, k-1$ we have $T_i(v) = \{0, 1\}$ and:

$$a_i[x_i = 1] \longrightarrow \bigvee_{\substack{j=0,\ldots,k-1 \\ v_j \in T}} \exists a_j[x_j = 1] \,.\, a_i \leq^{\mathsf{e,s}}_{[0,0]} a_j$$

All the synchronization rules of the original problem have to be translated to talk about the newly introduced binary variables. So each appearance of a token quantifier of the form $a[x = v_i]$, both in a trigger or in an existential statement, is translated to a token quantifier $a[x_i = 1]$. For example:

$$a[x = v_1] \longrightarrow \exists b[x = v_2] \,.\, \mathcal{C}$$

gets translated into:

$$a[x_1 = 1] \longrightarrow \exists b[x_2 = 1] \,.\, \mathcal{C}$$

It can be verified that the translated problem $\mathcal{P}'$ is equivalent to $\mathcal{P}$, in the sense that for any solution of $\mathcal{P}$ one for $\mathcal{P}'$ exists where at each time $x = v_i$ iff $x_i = 1$.

However, some concerns may exist about the size of $\mathcal{P}'$. Values of x cannot be represented by a more compact binary notation using $\log |V|$ binary variables because then translating synchronization rules would require to apply universal quantification to multiple variables at a time, which is syntactically not possible. However, this kind of *unary* representation does not significantly increase the size of the problem. To see this consider that that the number of additional synchronization rules is $\mathcal{O}(k^2)$, but $k \in \mathcal{O}(n)$, where n is the size of the input problem, since transition and duration functions already forced values to be enumerated in an extensional way. Thus this translation cause at most a polynomial size growth. ■

Proposition 3. *Any temporal planning problem can be rewritten, with at most a polynomial increase in size, into an equivalent one that only makes use of temporal formulae of the form* $[i]\phi$.

Proof: Let $P = (A, I, O, R, D, G)$ be a temporal planning problem. We will build an equivalent problem $P' = (A', I', O', R', D', G')$ whose precondition formulae only makes use of temporal operators of the form $[i]\phi$, equal to P excepting for what follows.

First, observe that $[i..j]\phi \equiv [j][i-j..0]\phi$ and that $[0]\phi \equiv \phi$, thus we can suppose without loss of generality that all the occurrences of temporal operators are either already of the simple form $[i]\phi$ or of the form $[i..0]\phi$, for $i < 0$.

For any formula ϕ that appears inside an occurrence of a temporal operator, let $[k_1..0]\phi, \ldots, [k_n..0]\phi$ be all such occurrences, and let $k = \max\{-k_1, \ldots, -k_n\} + 1$. The key idea is to encode a counter that increments at each step through all the execution of the plan, from zero up to a maximum of k (and stays at k afterwards), but resets to zero every time $\neg\phi$ holds. Then, to know if $[k_i..0]$ holds it is sufficient to check if the counter is greater than $-k_i$.

The value of the counter c^ϕ, for the formula ϕ, in short only c from now, is represented in binary notation by additional *actions* $c_0, \ldots, c_{w-1} \in O'$ (c_0 the least significant), where $w = \lceil \log_2(k+1) \rceil + 1$. What follows will use a few shorthands for basic formulae that assert useful facts about the counter:

- The formula $c = n$, for $n < k$, asserts the current value of the counter, and is simply a conjunction of literals asserting the truth value of the single bits of n. The formula $c \neq n$ is a shorthand for $\neg(c = n)$.
- The formula $c < n$ asserts that the current value of c is less than the value that c had (or will have) at i steps from now. This shorthand can be defined recursively on the number w of bits. Let $\langle b_0 \ldots b_{w-1} \rangle$ be the bits of n ($\top$ for 1, $\bot$ for 0). For $w = 1$, $c_0 < b_0$ is just $\neg c_0 \wedge b_0$. For $w > 1$, $\langle c_0 \ldots c_{w-1} \rangle < \langle b_0 \ldots b_{w-1} \rangle$ is:
$$(c_{w-1} < b_{w-1}) \vee$$
$$(c_{w-1} \longleftrightarrow b_{w-1} \wedge \langle c_0 \ldots c_{w-2} \rangle < \langle b_0 \ldots b_{w-2} \rangle)$$
Then, $c < n$ is just $\langle c_{w-1} \ldots c_0 \rangle < \langle b_{w-1}, \ldots b_0 \rangle$. Moreover, shorthands $c > n$, $c \geq n$ and $c \leq n$ are defined as one may expect.
- The formula $\mathrm{inc}(c)$ asserts that the counter has incremented its value since the previous step, *i.e.*, if c currently holds the value n, then at the previous step it held the value $n-1$, and vice versa. Again, it can be defined recursively on the number w of bits. For $w = 1$, $\mathrm{inc}(c_0)$ is simply $[-1]c_0 \longleftrightarrow \neg c_0$.
For $w > 1$, $\mathrm{inc}(\langle c_0 \ldots c_{w-1} \rangle)$ is defined as:
$$\begin{aligned} & [-1]c_0 \longleftrightarrow \neg c_0 \\ \wedge \quad & [-1]c_0 \longrightarrow \mathrm{inc}(\langle c_0 \ldots c_{w-2} \rangle) \\ \wedge \, & \neg[-1]c_0 \longrightarrow (\langle c_0 \ldots c_{w-2} \rangle = [-1]\langle c_0 \ldots c_{w-2} \rangle) \end{aligned}$$

With these formulae in place we can write a rule that enforce the counter to increase at each step if less than k, stay still when it reaches k, and reset to zero whenever $\neg\phi$ holds. For this purpose we introduce an additional *fluent* $f_c \in A'$ that we will set to true at the initial state and that we require to be true in the goal condition. In other words:

$$I'(f_c) = 1$$
$$G' \equiv G \wedge f_c$$

This flag will be set to false by the following rule, to invalidate the plan whenever the counter does *not* behave as intended. The rule is thus $(\neg P, \{\neg f_c\})$ where P is the following formula:

$$\begin{aligned} P \equiv & ([-1](\phi \wedge c < k) \longrightarrow \mathrm{inc}(c)) \wedge \\ & ([-1](\phi \wedge c = k) \longrightarrow c = k) \wedge \\ & ([-1]\neg\phi \longrightarrow c = 0) \end{aligned}$$

Clause (A) says that if at the previous step ϕ was true and the counter had not reached its maximum value, then an increment took place. Clause (A) says that if ϕ was true but the counter reached the maximum value, it stayed the same. Finally, clause (A) states that if ϕ *did not* hold at the previous step, then the counter had to be reset to zero. Since this rule set f_c to false whenever P is *false*, any plan containing a sequence of states where the counter does not behave as wanted is invalidated. However, any plan that was valid before is still valid now, when the valuations for the new actions and the new fluent are added accordingly. With the counter in place, we can rewrite any formula of the form $[k_i..0]\phi$ with the formula $c_\phi > -k_i$, stating that the steps passed since the last time ϕ was false are more than $-k_i$.

As can be seen, this encoding only adds a constant number of rules and a single new fluent for each formula ϕ that appears inside a temporal operator. The size of the precondition formula for the new rule is polynomial in the number of *bits* used to represent $k_0, \ldots, k_n$, thus polynomial in the size of the input. ■