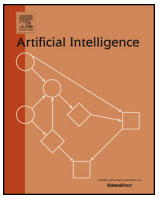
Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



# Decidability and complexity of action-based temporal planning over dense time

Nicola Gigante [a,*], Andrea Micheli [b], Angelo Montanari [c], Enrico Scala [d]

[a] *Free University of Bozen-Bolzano, Italy*
[b] *Fondazione Bruno Kessler, Trento, Italy*
[c] *University of Udine, Italy*
[d] *University of Brescia, Italy*

ARTICLE INFO

*Article history:*
Received 10 December 2020
Received in revised form 23 February 2022
Accepted 1 March 2022
Available online 4 March 2022

*Keywords:*
Temporal planning
Computational complexity
Timed automata
Tiling problems

ABSTRACT

In this paper, we study the computational complexity of action-based temporal planning interpreted over *dense* time. When time is assumed to be discrete, the problem is known to be EXPSPACE-complete. However, the official PDDL 2.1 semantics and many implementations interpret time as a dense domain. This work provides several results about the complexity of the problem, focusing on some particularly interesting cases: whether a minimum amount $\varepsilon$ of separation between mutually exclusive events is given, in contrast to the separation being simply required to be non-zero, and whether or not actions are allowed to overlap already running instances of themselves. We prove the problem to be PSPACE-complete when self-overlap is forbidden, whereas, when it is allowed, it becomes EXPSPACE-complete with $\varepsilon$-separation and even undecidable with non-zero separation. These results clarify the computational consequences of different choices in the definition at the core of the PDDL 2.1 semantics, which have been vague until now.[1]



## 1. Introduction

Domain-independent planning [33] is one of the classical fields of Artificial Intelligence and received considerable attention over the years. One of the most active research directions in this context is *temporal planning*, which represents and reasons about the flow of time explicitly. A popular modeling language for such problems is PDDL 2.1 [28], an action-centered formalism that extends classical planning by explicitly modeling the *duration* of actions. A temporal planning problem in PDDL 2.1 consists of looking for a sequence of actions that is not only *causally* executable (as in *classical* planning), but also *schedulable*, in accordance to a given set of constraints on action duration, along a timeline of unbounded length. Several planning systems [18,27,31,46] as well as various international planning competitions [19,52] adopt or have adopted PDDL 2.1 for the specification of the temporal planning problem.

Here, we study the computational complexity of temporal planning problems over a *dense* temporal domain, that is, when time points are interpreted as elements of a dense linear order (one where there is always a third element between any

* Corresponding author.
*E-mail addresses:* nicola.gigante@unibz.it (N. Gigante), amicheli@fbk.eu (A. Micheli), angelo.montanari@uniud.it (A. Montanari), enrico.scala@unibs.it (E. Scala).

[1] This paper is a considerably extended and revised version of [36].

https://doi.org/10.1016/j.artint.2022.103686

**Table 1**
Complexity bounds for the different considered dense-time semantics. $L_a = U_a$ is the case where duration is fixed, while $[L_a, U_a]$ is the case where the duration can be any value in the interval. Bold font indicates novel results. We recall that Rintanen [48] proves PSPACE-completeness and EXPSPACE-completeness for the case of discrete-time for the $>0$-separation-semantics and $\varepsilon$-separation-semantics, respectively, while all cases in this table assume that time is dense.

| | $\varepsilon$-separation ($L_a = U_a$) | $\varepsilon$-separation ($[L_a, U_a]$) | $>0$-separation ($[L_a, U_a]$) |
|---|---|---|---|
| w/o self-overlap | PSPACE-complete | **PSPACE-complete** | |
| self-overlap | EXPSPACE-complete | **EXPSPACE-complete** | **undecidable** |

two). The problem we are studying includes the essential temporal aspects of the full syntax of PDDL 2.1. More precisely, we restrict our attention on fully ground action representations with simple unconditional effects, and conjunctive preconditions and goals.

To the best of our knowledge, to date, only Rintanen [48], Cushing et al. [21], and Cushing [20] approached temporal planning from a theoretical point of view; however, they focused their attention on a temporal model that is substantially discrete. In particular, Rintanen [48] proves the problem to be EXPSPACE-complete over *discrete* time in the general case, and PSPACE-complete when actions are disallowed to self-overlap with already running instances of themselves. These results apply to the dense setting if a specific $\varepsilon$ is given as the minimum amount of time separating mutually exclusive (*mutex*) events, and if actions are given only a specific fixed duration, whereas PDDL 2.1 generally specifies actions with an interval of admissible values for the duration.

The computational complexity arising from using a dense temporal model with no $\varepsilon$ value given upfront remains still poorly understood. It is worth noticing that the formal specification of PDDL 2.1 [28, Section 8] only requires mutex events to be separated by a non-zero amount of time; the idea of accepting an $\varepsilon$-separation value as input comes later in the text as an expedient to facilitate plan validation (Section 10 - Plan Validation). The very same authors do however admit that this ambiguity was at that time problematic and they did not find a definitive and principled way to account for it. Clarifying this aspect is relevant not only because it could be at times impractical to provide the right $\varepsilon$ value upfront, but also because many planners do not use a discrete temporal model at all [18,51]. The distance between practice and theory seems unnecessarily high.

In order to work out these issues, this paper analyzes the computational complexity of temporal planning problems over dense time taking into account, in a comprehensive manner, a number of variants: the case with $\varepsilon$-separation, where an $\varepsilon$ value of separation between mutex events is given upfront, and the case with $>0$-separation, where mutex events are only required to not appear at the same time. Both cases are studied either allowing or forbidding self-overlap of actions.

The results of our work can be summarized as follows: when self-overlap of actions is forbidden temporal planning over dense time is not harder than classical planning (PSPACE-complete), regardless of the mutex separation criterion. On the other hand, allowing actions to self-overlap makes the problem harder: EXPSPACE-complete with $\varepsilon$-separation and, perhaps most surprisingly, *undecidable* in the $>0$-separation case. We prove these results by studying the problems with and without self-overlapping separately. For the case with no self-overlap, we devise a novel polynomial reduction to *updatable timed automata* [2,10], which gives us the PSPACE upper bound for both the $\varepsilon$-separation and $>0$-separation cases. For the case with self-overlap, we provide a reduction from two specific variants of the *corridor tiling problem* [53], known to respectively be EXPSPACE-complete and undecidable, to temporal planning problems with $\varepsilon$-separation and $>0$-separation semantics, respectively. We work with actions with a non-fixed duration, thus extending the bound found by Rintanen [48] to a more general setting. Table 1 summarizes the results of the paper.

Our theoretical results highlight that, at least from a computational complexity standpoint, the adoption of a truly continuous representation of time does not necessarily make temporal planning harder than classical planning; indeed, when self-overlapping is forbidden, there is no reason for the user to anticipate some domain dependent epsilon before planning, regardless of whether the duration is fixed (equality) or simply formulated as a min-max interval. On the other hand, our undecidability result can be seen as a further indication that the interpretation with self-overlapping is not only of dubious usefulness, but is likely also an unintended feature of the formalism.

The paper is structured as follows. Section 2 formally defines the problem, and Section 3 surveys related work. Then, Section 4 provides a complexity analysis of the problem when self-overlap of actions is forbidden, and Section 5 when it is allowed. Section 6 concludes the paper with some final remarks.

## 2. Dense-time temporal planning

This section introduces the temporal planning problem we are interested in. Our analysis deals with the core aspects of PDDL 2.1 [28], limiting the full syntax to the STRIPS fragment with a set-theoretical representation [32].

**Definition 2.1** *(Temporal planning problem).* A temporal planning problem is a tuple $\mathcal{P} = \langle P, A, I, G \rangle$, where $P$ is a set of propositions, $A$ is a set of durative actions, $I \subseteq P$ is the initial state, and $G \subseteq P$ is the goal condition. A snap (instantaneous) action is a tuple $h = \langle \text{pre}(h), \text{eff}^+(h), \text{eff}^-(h) \rangle$, where $\text{pre}(h) \subseteq P$ is the set of preconditions and $\text{eff}^+(h), \text{eff}^-(h) \subseteq P$ are two disjoint sets of propositions, called the *positive* and *negative* effects of $h$, respectively. We write $\text{eff}(h)$ for $\text{eff}^+(h) \cup \text{eff}^-(h)$.

A durative action $a \in A$ is a tuple $\langle a_{\vdash}, a_{\dashv}, \text{pre}^{\leftrightarrow}(a), [L_a, U_a]\rangle$, where $a_{\vdash}$ and $a_{\dashv}$ are the *start* and *end* snap actions, respectively, $\text{pre}^{\leftrightarrow}(a) \subseteq P$ is the *over-all condition*, and $L_a \in \mathbb{Q}_{>0}$ and $U_a \in \mathbb{Q}_{\geq 0} \cup \{\infty\}$ are the bounds on the action duration.

As we are using a set-theoretic representation, both states and conditions are subsets of $P$, which is the universe of facts over which one can determine the status of things, and the possible transitions that may take place in the system. When interpreted as a state, a subset $s$ of $P$ lists those atoms which hold true in it and implicitly asserts false those which are not part of it (closed-world assumption). When interpreted as a condition, the subset lists those atoms that need to be true in order for the condition to be satisfied. The initial state $I$ specifies what holds at the beginning, before execution, while action tuples in $A$ specify the dynamics of the system, that is, how a state can change, and under which propositional and temporal conditions such changes may happen. An action tuple delegates the specification of the state transition to two snap actions: one is relative to when the durative action starts, and one to when the durative action ends. As classical planning actions, these instantaneous transitions are represented by a pair, which encodes the applicability of the transition ($\text{pre}(h)$), and the effects that the transition has on the state when applied ($\text{eff}(h)$). Unlike in classical planning problems, however, in temporal planning actions last for a certain time (they are durative), and their duration has to satisfy the given lower and upper time limits, that is, $[L_a, U_a]$. Moreover, since the state can change while the action is under execution, we can further require that all intermediate states satisfy a given invariant condition ($\text{pre}^{\leftrightarrow}(a)$). Note that PDDL 2.1 allows for instantaneous classical actions to be defined alongside durative actions. Here, for simplicity we do not allow for this feature, but it would be straightforward to do so. Finally, all the propositions in $G$ determine what needs to be achieved in order for the problem to be solved.

A collection of action tuples from $A$, together with a starting time and a duration, is called a plan.

**Definition 2.2** *(Plan).* Let $\mathcal{P} = \langle P, A, I, G\rangle$ be a planning problem. A plan for $\mathcal{P}$ is a set of tuples $\pi = \{\langle a_1, t_1, d_1\rangle, \cdots, \langle a_n, t_n, d_n\rangle\}$, where, for each $1 \leq i \leq n$, $a_i \in A$ is a durative action, $t_i \in \mathbb{Q}_{\geq 0}$ is its start time, and $d_i \in \mathbb{Q}_{>0}$ is its duration.

A plan can be understood as a set of timed decisions the agent can take over time. Indeed, each tuple of a plan defines what action needs to start ($a_n$), when it must be initiated ($t_n$), and how long it has to last ($d_n$).

In order to precisely state whether a given plan is valid with respect to the temporal planning problem it represents a candidate solution for, in the following we recall and summarize the state-transition model interpretation of a temporal plan given by Fox and Long [28].

**Definition 2.3** *(Set of timed snap actions).* A timed snap action (TSA) is a pair $\langle t, h\rangle$, where $t \in \mathbb{Q}_{\geq 0}$ and $h$ is a snap action. Given a plan $\pi = \{\langle a_1, t_1, d_1\rangle, \cdots, \langle a_n, t_n, d_n\rangle\}$, the set of TSAs of $\pi$ is defined as:

$$H(\pi) = \{\langle t_1, a_{1\vdash}\rangle, \langle t_1 + d_1, a_{1\dashv}\rangle, \cdots, \langle t_n, a_{n\vdash}\rangle, \langle t_n + d_n, a_{n\dashv}\rangle\}$$

Given a set of timed snap actions, we define the induced parallel plan, as the sequence of sets of timed snap actions sharing the same time index. As we will see, the validity of plans can be stated by defining constraints over the induced parallel plan that can be extracted from the timed snap actions of a plan.

**Definition 2.4** *(Induced parallel plan).* Let $\pi$ be a plan and let $H(\pi) = \{\langle t'_1, h_1\rangle, \cdots, \langle t'_m, h_m\rangle\}$ be the set of TSAs of $\pi$. The induced parallel plan for $\pi$ is the sequence $\pi^{ind} = \langle\langle t''_1, \{h \mid \langle t''_1, h\rangle \in H(\pi)\}\rangle, \cdots, \langle t''_k, \{h \mid \langle t''_k, h\rangle \in H(\pi)\}\rangle\rangle$, which is ordered and grouped with respect to the time index, that is, $\forall i, j \in \{1, \cdots, k\}$, $i < j$ if and only if $t''_i < t''_j$, and if $t''_i = t''_j$, then $i = j$. Given $c_i = \langle a_i, t_i, d_i\rangle \in \pi$, we denote by $\pi^{ind}_{\vdash}(c_i) = x$, with $t''_x = t_i$, and $\pi^{ind}_{\dashv}(c_i) = y$, with $t''_y = t_i + d_i$, the indexes of the pairs in $\pi^{ind}$ containing, in the right hand side, the snap actions $a_{i\vdash}$ and $a_{i\dashv}$ relative to $c_i$, respectively.

The semantics variants of temporal plans that we study in this paper rely on classical planning and the mutex relation between pairs of snap actions. We say that two snap actions are mutex whenever one interferes with at least one effect or precondition of the other.[2] They are formally defined as follows.

**Definition 2.5** *(Mutex snap actions).* Two snap actions $h$ and $z$ are *mutually exclusive* (mutex), denoted by $\text{mutex}(h, z)$, if either $\text{pre}(h) \cap \text{eff}(z) \neq \varnothing$, or $\text{pre}(z) \cap \text{eff}(h) \neq \varnothing$, or $\text{eff}^{+}(h) \cap \text{eff}^{-}(z) \neq \varnothing$, or $\text{eff}^{+}(z) \cap \text{eff}^{-}(h) \neq \varnothing$.

We are now ready to define the notion of plan validity. Intuitively, a plan is said to be valid if (i) the induced plan is a classical goal-reaching execution where all overall and duration constraints are satisfied, and (ii) mutex snap actions do not appear at the same time in a plan. Depending on whether a minimum amount $\varepsilon$ of separation between any pair of

[2] The notion of mutex between snap actions has been introduced by Fox and Long [28] as a conservative measure to enforce the *no moving targets* rule: *no two actions can simultaneously make use of a value if one of the two is accessing the value to update it – the value is a moving target for the other action to access.*

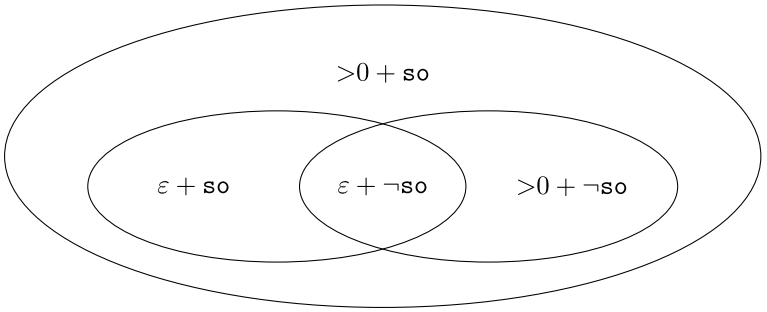



**Fig. 1.** Accepted plans according to the semantics variants: $\varepsilon$ stands for the $\varepsilon$-separation semantics (analogously, $>0$ stands for the $>0$-separation); so indicates whether self-overlapping is allowed or not ($\neg$so).

mutex snap actions must be enforced or any separation suffices, different ways of meeting requirement (ii) are possible. Two different notions of *plan validity* can be therefore defined.

**Definition 2.6** *($>0$-separation plan validity).* Let $\mathcal{P} = \langle P, A, I, G\rangle$ be a temporal planning problem, $\pi = \{\langle a_1, t_1, d_1\rangle, \cdots, \langle a_n, t_n, d_n\rangle\}$ be a plan for $\mathcal{P}$, and $\pi^{ind} = \langle\langle t'_1, B_1\rangle, \cdots, \langle t'_m, B_m\rangle\rangle$ be its induced plan. Then, $\pi$ is valid under the *$>0$-separation semantics* if the following statements hold:

(i) $\forall i \in \{1, \cdots, n\}\ L_{a_i} \leq d_i \leq U_{a_i}$,
(ii) there are no $h, z \in B_i$, with $h \neq z$, for some $i \in \{1, \cdots, m\}$, such that mutex$(h, z)$,
(iii) given $s_0 = I$, for all $i \in \{1, \ldots, m\}$, it holds that:
  (a) $\bigcup_{h\in B_i} \text{pre}(h) \subseteq s_i$;
  (b) $s_i = (s_{i-1} \setminus \bigcup_{h\in B_i} \text{eff}^-(h)) \cup \bigcup_{h\in B_i} \text{eff}^+(h)$;
  (c) $G \subseteq s_m$,
(iv) for all $c = \langle a, t, d\rangle \in \pi$ and all $\pi^{ind}_{\vdash}(c) \leq k < \pi^{ind}_{\dashv}(c)$, we have $\text{pre}^{\leftrightarrow}(a) \subseteq s_k$.

**Definition 2.7** *($\varepsilon$-separation plan validity).* Let $\mathcal{P} = \langle P, A, I, G\rangle$ be a temporal planning problem, $\pi = \{\langle a_1, t_1, d_1\rangle, \cdots, \langle a_n, t_n, d_n\rangle\}$ be a plan for $\mathcal{P}$, $\pi^{ind} = \langle\langle t'_1, B_1\rangle, \cdots, \langle t'_m, B_m\rangle\rangle$ be its induced plan, and $\varepsilon \in \mathbb{Q}_{>0}$. Then, $\pi$ is valid under the *$\varepsilon$-separation semantics* if it is valid under the $>0$-separation semantics and the following additional condition holds as well:

(v) for all $i, j \in \{1, \cdots, m\}$, with $i \neq j$, such that there exist $h \in B_i$ and $z \in B_j$, with mutex$(h, z)$, we have that $|t'_i - t'_j| \geq \varepsilon$.

Another feature has relevant consequences from a computational perspective, both under $\varepsilon$-separation and $>0$-separation semantics: self-overlap of actions.

**Definition 2.8** *(Self-overlap of actions).* Let $\mathcal{P} = \langle P, A, I, G\rangle$ be a temporal planning problem and $\pi = \{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$ be a plan for $\mathcal{P}$. An action $a \in A$ is said to *self-overlap* in $\pi$ if there exist $1 \leq i, j \leq n$, with $i \neq j$, such that $a = a_i = a_j$ and $t_i \leq t_j \leq t_i + d_i$.

Fig. 1 shows the relationships among the plans accepted by the various notions of plan validity defined so far. In the following, we study the computational complexity of deciding whether a solution plan exists for a given planning problem $\mathcal{P}$, under both the above-defined notions of plan validity, considering the cases where self-overlap is allowed and where it is not.

We conclude the section by introducing a syntactic variant of the temporal planning problem that allows general Boolean formulas as conditions of actions. Such a variant will be exploited in Section 5, where a reduction from a tiling problem to temporal planning will be defined.

Let $P$ be a set of propositions. A *Boolean condition* is a Boolean formula over $P$. Temporal planning with Boolean conditions is defined as follows.

**Definition 2.9** *(Temporal planning problem with Boolean conditions).* A temporal planning problem with Boolean conditions is a temporal planning problem where the precondition pre$(h)$ of each snap action $h$, the *over-all* condition $\text{pre}^{\leftrightarrow}(a)$ of each action $a$, and the goal $G$ are Boolean conditions.

Planning problems with Boolean conditions can be turned into planning problems with set-theoretic conditions. The semantics of Boolean conditions immediately follows from such a translation. It is well known that the temporal planning

problem with Boolean conditions of Definition 2.9 is equivalent to the temporal planning problem with set-theoretic conditions of Definition 2.1 in terms of expressive power, but it can be exponentially more succinct. In other words, translating Boolean conditions into set-theoretic conditions can incur into an exponential blowup *in the size of the Boolean conditions*. However, we are not going to be affected by this blowup, as explained in Section 5.

## 3. Related work

In this section, we discuss related work on the computational complexity of timed systems represented by planning languages (both action-based and timeline-based), verification formalisms, temporal logics, and process algebras.

### 3.1. Action-based planning

*Classical* planning, which is devoid of an explicit (metric) temporal dimension, has received considerable attention over the years. The basic planning problem has been proved to be PSPACE-complete [14]; moreover, fixed-parameter tractable fragments have been studied [7], and strict time and space bounds have been identified [6]. Complexity, decidability and undecidability results have been also given for many variants of the problem [25]. A parameterized complexity analysis of cost-optimal planning has been done in [1]. Hierarchical planning, that is, planning with action decomposition, has been studied as well [26]. Planning problems with nondeterministic aspects have been investigated and thoroughly classified in a series of papers [40,41,44,47], including planning problems under full observability of the current state (FOND planning, e.g., [16]), under partial observability [47], and with *probabilistic* domains [42].

Two major approaches have been proposed in the literature to deal with *temporal* planning problems, where timing aspects explicitly come into play: classical action-based temporal planning and timeline-based planning.

Temporal planning adds a temporal dimension to classical action-based planning [28]. Cushing [20] discusses at length the philosophical subtleties of some semantic aspects, including the impact of the non-zero vs. $\varepsilon$-separation issue in PDDL 2.1. However, he diverges significantly from the PDDL 2.1 modeling formulation, and gives no complexity results. Relevant to our analysis is also the work by Shin and Davis [51], who raise the ambiguity between $\varepsilon$-separation and $>0$-separation, and favor the latter in their SMT encoding of the problem. Note that, despite the confusion on this matter, PDDL 2.1 semantics does not prescribe to accept an $\varepsilon$ separation value as input to the planner. The $\varepsilon$ input is only suggested by Fox and Long [28] as a way to alleviate the burden of a potentially complex plan validation task. Rintanen [48] focuses on temporal planning over discrete time, showing the problem to be EXPSPACE-complete. Then, he proceeds showing that forbidding action self-overlap, with constant action durations ($L_a = U_a$), makes the problem PSPACE-complete, that is, reducible to classical planning. The result improves the quite restrictive conditions of *temporally simple languages* previously identified by Cushing et al. [21]. Transferring these results to dense time is *a priori* possible only assuming $\varepsilon$-separation, since then problems can be suitably scaled and discretized at will. As we will show, discretization is not always possible under $>0$-separation, as the problem becomes *undecidable* when self-overlap is allowed. Moreover, while the restriction to fixed action durations is just syntactic convenience in the discrete case, since an action with an interval $[L_a, U_a]$ of possible durations can be replaced by a finite number of copies, with dense time this is, in general, not possible.

Other action-based languages have been proposed to model temporal planning problems. The NDL language [49] is a formalism that allows actions with timed effects, and supports action concurrency by using a resource-based model. Each action can be associated with a set of resource requests, each representing which resource is used by the action, and over which interval. As resources can have finite capacity, a valid plan has to ensure that resources are not used by multiple actions at the same time. PDDL+ [29] is another extension of classical planning capable of dealing with time. Unlike PDDL 2.1, where actions are durative and defined over intervals, PDDL+ semantics is based on instantaneous actions representing choices and autonomous *processes* that model the evolution of the system through time. In addition, PDDL+ also supports exogenous *events*. In its general form, PDDL+ is undecidable [29], while the computational complexity of NDL is still an open problem.

### 3.2. Timeline-based planning

In timeline-based planning (TP for short), planning domains are described as collections of independent, but interacting, components, each one consisting of a set of state variables. The evolution of the values of the variables is modeled 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. A systematic analysis of the computational complexity of timeline-based planning, over both discrete [23,34,35] and dense time [13], has been undertaken in the last years. In [34], Gigante et al. showed that (discrete) TP with bounded temporal relations and token durations, and no temporal horizon, is EXPSPACE-complete and expressive enough to capture action-based temporal planning. Later, Gigante et al. proved that TP with unbounded interval relations is still EXPSPACE-complete [35] (it becomes NEXPTIME-complete if an upper bound is added to the temporal horizon), and that the same holds for TP with recurrent goals [43].

TP over dense time has been studied in [13], where it has been shown to be undecidable even when a single state variable is used. Decidability can be recovered by suitably constraining the logical structure of synchronization rules. In general, synchronization rules make it possible to universally quantify over the tokens of a timeline (triggers). By restricting

to rules in purely existential form, the TP problem becomes NP-complete. In [13], various intermediate cases have been studied. A first possibility is to constrain any non-trigger token to appear exactly once in the body of the rule (simple trigger rules). Such a restriction prevents temporal comparisons of a single, non-trigger token with many others. A second one concerns future and past tokens. When a token is selected by a trigger, the synchronization rule allows one to compare such a token with other tokens of the timelines both preceding (past) and following (future) it. One can restrict the comparison only to future tokens (future semantics of trigger rules). It has been shown that the TP problem restricted to simple trigger rules [13] or to trigger rules with the future semantic [12] is still undecidable. Decidability can be recovered by restricting to future TP with simple trigger rules, which is non-primitive recursive-hard. Better complexity results can be obtained by restricting also the type of intervals used in the simple trigger rules to compare tokens. In particular, future TP with simple trigger rules without singular intervals (intervals of the form $[a, a]$, for $a \in \mathbb{N}$) is EXPSPACE-complete, PSPACE-complete if one only allows intervals of the forms $[0, a]$ and $[b, +\infty[$.

In this paper, we focus on action-based temporal planning in the dense-time setting, and extend the results obtained for the discrete case by naturally handling actions of non-constant durations and by covering both $\varepsilon$-separation and $>0$-separation semantics, with or without self-overlap. Table 1 summarizes the old and, in bold, the new results. When comparing the results in Table 1 with those for the timeline-based planning paradigm, it is interesting to note a similar complexity jump, with a problem that is EXPSPACE-complete over discrete time [35], but becomes undecidable over dense time [13].[3]

### 3.3. Timed systems verification

The formal verification of systems that evolve over time, in particular the analysis of the computational complexity of the problem of checking their properties, is a key topic in many research areas.

In the verification setting, timed automata are commonly used to model timed systems and to check their properties [2]. The fundamental problem of finding a path leading to a certain location (reachability problem) in a timed automaton is known to be PSPACE-complete [2]. Variants of timed automata have been studied [10], in particular the reachability problem for timed automata with constant updates has been shown to be PSPACE-complete as well. In analogy to other approaches [9,11,38], to provide suitable complexity bounds to action-based temporal planning problem, we will reduce it to a reachability problem for a timed automaton; however, unlike previous work, we will show how to express the semantics of PDDL 2.1 with a *polynomial-size* timed automaton. Extensions of classical Petri nets, that make it possible to explicitly represent time and to deal with timing constraints, have also been proposed in the literature (Timed Petri nets [55]). They have been used, for instance, to model and analyze communicating systems with timing constraints and delays. Their reachability problem is, in general, undecidable; however, fragments and variants with better complexity have been studied [30]. Besides timed automata and Petri nets, other formalisms have been developed to model timed systems. A fully-symbolic representation of infinite-state timed systems with clock constraints can be found in [17], and it allows one to efficiently perform model-checking analyses, while Max-Plus linear systems, which are able to model repetitive temporal behaviors and synchronizations, have been successfully applied to the analyses of production lines and railway schedules [37].

### 3.4. Temporal logics and process algebras

The problem of representing real-time constraints and system dynamics have also been addressed in the field of temporal logic. One of the most successful proposals is Metric Temporal Logic (MTL) [45], that generalizes LTL operators to cope with different interpretations of time. Another meaningful extension of LTL is Timed Propositional Temporal Logic (TPTL) [3,4], which introduces explicit clocks and allows one to constrain their values. The satisfiability problem for both MTL and TPTL is undecidable when continuous time and an unbounded horizon are assumed. Suitable restrictions must be imposed to recover decidability. As an example, in [22], it has been shown that timeline-based planning with bounded temporal constraints, over a discrete temporal domain, can be captured by a bounded version of TPTL, augmented with past operators, whose satisfiability problem is EXPSPACE-complete. Similar extensions have been developed for the branching time logic CTL yielding Timed CTL [5].

Finally, various extensions of process algebras with timing operators have been proposed in the literature. In particular, extensions of CCS and ACP are given in [39] and [8], respectively. They can be used to model and analyze concurrent behaviors of protocols and programs where timings, delays, and synchronizations are crucial aspects.

## 4. Forbidding self-overlap of actions

In this section, we determine the complexity of temporal planning over dense time when actions are not allowed to overlap with themselves. Since temporal planning extends classical one, which is known to be PSPACE-complete [14], it is trivially PSPACE-hard. We prove that the problem can be solved in polynomial space by encoding it into a particular kind of *timed automata*.

[3] As a matter of fact, neither self-overlap nor $\varepsilon$-separation/$>0$-separation is supported in TP, making a direct comparison of complexity results difficult.

### 4.1. Updatable timed automata

A *timed automaton* (TA) is a finite automaton augmented with a set of *clocks* [2], which explicitly track the flow of *time*. Each transition in a TA may include temporal constraints, called *guards*, that disable the transition if not satisfied by the current clock values. Each transition may also include *clock resets* that cause specified clocks to be reset to zero whenever the transition is taken. Finally, each location may include an *invariant*—that is, a constraint specifying the conditions under which the automaton may stay in that location. Here, we use the more general *updatable timed automata* [10], that allow clocks to be reset to any constant rational value, not only to zero.

Let $\mathcal{X}$ be a set of elements, called *clocks*. The set $\mathcal{C}(\mathcal{X})$ of *constraints* over the clocks in $\mathcal{X}$ contains conjunctions of constraints of the form $x \bowtie k$ or $y - x \bowtie k$, where $x, y \in \mathcal{X}$, $k \in \mathbb{Q}$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. The set $\mathcal{U}(\mathcal{X})$ of *updates* on the clocks in $\mathcal{X}$ is the set of all basic statements of the form $x := k$, with $x \in \mathcal{X}$ and $k \in \mathbb{Q}$.

**Definition 4.1** *(Updatable timed automaton).* An *updatable timed automaton* is a tuple $\mathcal{T} = \langle \Sigma, \mathcal{L}, \ell_0, \mathcal{X}, \Delta, Inv \rangle$, where:

1. $\Sigma$ is the alphabet;
2. $\mathcal{L}$ is a finite set of locations;
3. $\ell_0 \in \mathcal{L}$ is the initial location;
4. $\mathcal{X}$ is a finite set of *clocks*;
5. $\Delta \subseteq \mathcal{L} \times \mathcal{C}(\mathcal{X}) \times \Sigma \times 2^{\mathcal{U}(\mathcal{X})} \times \mathcal{L}$ is the transition relation;
6. $Inv : \mathcal{L} \to \mathcal{C}(\mathcal{X})$ maps each location to its *invariant*.

**Definition 4.2** *(Semantics of an updatable timed automaton).* Given an updatable timed automaton $\mathcal{T} = \langle \Sigma, \mathcal{L}, \ell_0, \mathcal{X}, \Delta, Inv \rangle$, a *state* is a pair $\langle \ell, v \rangle$, where $\ell \in \mathcal{L}$ and $v : \mathcal{X} \to \mathbb{Q}_{\geq 0}$. Intuitively, a state represents a location where the automaton is in and a value assignment for each of the clocks.

Given a set of clock updates $U$ and a valuation of clocks $v$, we indicate the valuation obtained by applying the updates in $U$ to $v$ as $v[U]$, defined as:

$$v[U](x) = \begin{cases} k & \text{if } x := k \in U \\ v(x) & \text{otherwise} \end{cases}$$

Furthermore, we define $v + \delta$, with $\delta \in \mathbb{Q}_{\geq 0}$, as the valuation such that, for all $x \in \mathcal{X}$, $(v + \delta)(x) = v(x) + \delta$. Moreover, we define the valuation $\vec{0}$ such that $\vec{0}(x) = 0$, for all $x \in \mathcal{X}$.

From a state that satisfies the location invariant ($v \models Inv(\ell)$), the automaton can either make a transition to a different location or let time elapse. This is formally captured by a transition relation $\to$ defined as follows:

**timed transition:** $\langle \ell, v \rangle \xrightarrow{\delta} \langle \ell, v' \rangle$, with $\delta \in \mathbb{Q}_{\geq 0}$, $v' = v + \delta$ and $v' \models Inv(\ell)$;
**discrete transition:** $\langle \ell, v \rangle \xrightarrow{a} \langle \ell', v' \rangle$, with $a = \langle \ell, C, \sigma, U, \ell' \rangle \in \Delta$, $v \models C$, $v' = v[U]$, and $v' \models Inv(\ell')$.

The semantics of an updatable timed automaton $\mathcal{T}$ is a labeled transition system $\langle Q, q_0, \xrightarrow{s} \rangle$, where $Q$ is the set of states of the automaton, $q_0 = \langle \ell_0, \vec{0} \rangle$, and $s$ takes value in the set of labels $\Delta \cup \mathbb{Q}_{\geq 0}$. A run of $\mathcal{T}$ is a sequence of alternating timed and discrete transitions in the labeled transition system.

The reachability problem for a TA $\mathcal{T} = \langle \Sigma, \mathcal{L}, \ell_0, \mathcal{X}, \Delta, Inv \rangle$ and a goal $\mathcal{G} \subseteq \mathcal{L}$ is the problem of deciding whether there exists a run of $\mathcal{T}$ that starts from $\ell_0$ and ends in a location $\ell^* \in \mathcal{G}$. The problem is PSPACE-complete with standard resets and constant updates [10].

To simplify the exposition, we make use of the concept of *urgent* locations, that is, locations where time is stationary, which are encoded by adding an extra clock that is reset to zero by each incoming transition and forced to be zero by the location invariant. Since the size of the automaton increases polynomially, the complexity of the reachability problem remains unchanged.

The intuition behind the encoding comes from decision-epoch planners [24,50]: at each step, the automaton can either execute a set of snap actions (by checking their preconditions and applying their effects) or decide to wait a certain amount of time (delta-transition). Unlike decision-epoch planners, however, the amount of time to wait is decided symbolically and is not forced to be aligned with a future event, avoiding the incompleteness problems of decision-epoch planners [20]. Crucially, to keep the size of the resulting automaton polynomial, there cannot be a distinct location for each propositional state (that is, for every possible truth assignment to the predicates) of the planning problem. Instead, we symbolically encode the predicates using clocks that maintain a truth value recognizable in the guards of the automaton. We use constant updates to apply the effects of actions on such clocks. As we will see, the encoding can be adapted to support either the $\varepsilon$-separation or the $>0$-separation semantics, hence proving the complexity of both cases, without self-overlap of actions.

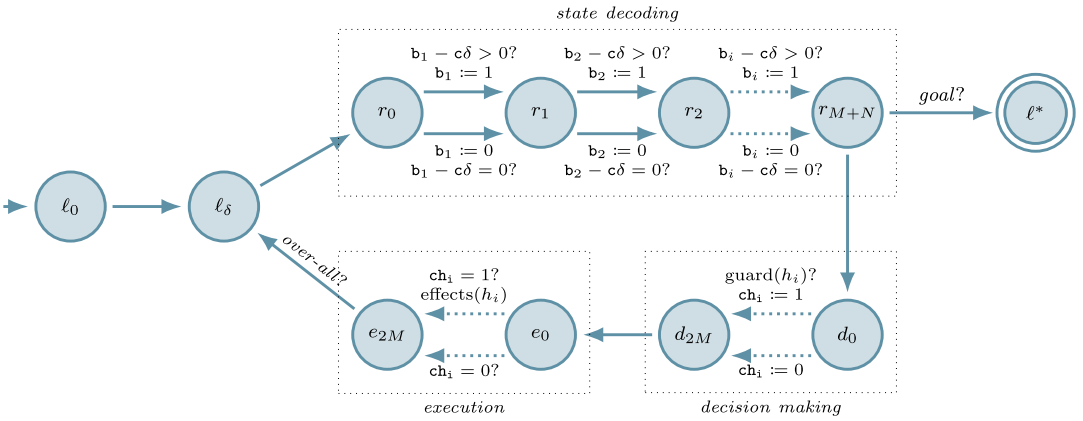



**Fig. 2.** The *updatable timed automaton* of Theorem 4.1. Note that $h_i$ indicates any snap action, *i.e.*, either $a_{i\vdash}$ or $a_{i\dashv}$, for some $a_i \in A$, and, similarly, $\mathtt{ch}_i$ stands for $\mathtt{cs}_{\mathtt{a}_i}$ or $\mathtt{ce}_{\mathtt{a}_i}$, for some $a_i \in A$.

### 4.2. Encoding temporal planning by means of updatable timed automata

We start by defining the updatable timed automaton that captures the semantics of a given planning problem. The description of the automaton proceeds as follows. First, we define the alphabet, the set of locations, and the set of clock variables used in the encoding; then, we define the transition relation. A graphical account of the resulting automaton is given in Fig. 2.

Let $\mathcal{P} = \langle P, A, I, G\rangle$ be a temporal planning problem, without self-overlap of actions. The Updatable Timed Automaton $\mathcal{T}[\mathcal{P}] = \langle \Sigma, \mathcal{L}, \ell_0, \mathcal{X}, \Delta, \mathit{Inv}\rangle$, that encodes it, is formally defined as follows.

Let $A = \{a_1, \ldots, a_M\}$ and $P = \{p_1, \ldots, p_N\}$.

1. The alphabet $\Sigma$ is an arbitrary singleton $\{\textvisiblespace\}$. In the proposed encoding, the words accepted by the automaton are indeed irrelevant, since all pieces of information are encoded by the timestamps of the accepted trace.
2. The set of locations $\mathcal{L}$ consists of:
   - the three locations $\ell_0$ (the *initial* location), $\ell^*$ (the *goal* location), and $\ell_\delta$ (the *time-passing* location);
   - a set $\{r_0, r_1, \ldots, r_{M+N}\}$ of $M+N+1$ *state decoding* locations[4];
   - a set $\{d_0, d_1, \ldots, d_{2M}\}$ of $2M+1$ *decision making* locations;
   - a set $\{e_0, e_1, \ldots, e_{2M}\}$ of $2M+1$ *execution* locations.

   All locations, excepting $\ell_\delta$, are *urgent*.
3. The set of clock variables $\mathcal{X}$ includes:
   - a clock $\mathtt{c}\gamma$, that is never reset, called the *global* clock;
   - a clock $\mathtt{c}\delta$;
   - a clock $\mathtt{cp}_\mathtt{i}$, for each $p_i \in P$;
   - five clocks, $\mathtt{cx}_{\mathtt{a}\vdash}$, $\mathtt{cx}_{\mathtt{a}\dashv}$, $\mathtt{cr}_\mathtt{a}$, $\mathtt{cs}_\mathtt{a}$, and $\mathtt{ce}_\mathtt{a}$, for each $a \in A$.
4. No invariant conditions are needed (except for the ones that are implicitly defined by urgent states), and thus $\mathit{Inv} = \varnothing$.

We now define the transition relation $\Delta$, explaining how the automaton works and how the above-defined locations are connected. A schema of the construction is depicted in Fig. 2. Remember that the time-passing location $\ell_\delta$ is the only location where time can pass, all the other locations being urgent. The initial location immediately transitions to the time-passing location, setting some of the $\mathtt{cp}_\mathtt{i}$ clocks to one, depending on the *initial state*:

$$\langle \ell_0, \top, \textvisiblespace, \{\mathtt{cp}_\mathtt{i} := 1 \mid p_i \in I\}, \ell_\delta\rangle \in \Delta$$

Let $B = \{\mathtt{cp}_\mathtt{i} \mid p_i \in P\} \cup \{\mathtt{cr}_\mathtt{a} \mid a \in A\}$ be a subset of the clocks, that we call the *binary* clocks, and let us denote them, with an arbitrary order, as $B = \{\mathtt{b}_1, \ldots, \mathtt{b}_{M+N}\}$. The initial transition establishes an invariant that is kept by construction throughout the automaton: when the execution *enters* $\ell_\delta$, it holds that $\mathtt{c}\delta = 0$ and, for $\mathtt{b}_k \in B$, either $\mathtt{b}_k = 0$ or $\mathtt{b}_k = 1$. Since clocks advance together while the automaton stays in the time-passing location, the *difference* between any $\mathtt{b}_k \in B$ and $\mathtt{c}\delta$ will always be either zero or one, accordingly. In this way, these clocks can be used as *binary variables*, and, in particular, the $\mathtt{cp}_\mathtt{i}$ clocks can be used to represent the state of the planning problem propositions.

[4] Note that $M+N+1$ locations are necessary because we need to decode each proposition in $P$ as well as the auxiliary binary clock $\mathtt{cr}_\mathtt{a}$, for each action $a \in A$.

In the time-passing location, the automaton can decide at any time to make a transition to the first *state decoding* location $r_0$:

$$\langle \ell_\delta, \top, \sqcup, \varnothing, r_0 \rangle \in \Delta$$

The $r_0$ location starts a chain of $M+N$ locations, that goes up to $r_{M+N}$, called the *state decoding* path. Its purpose is to reset each clock $\mathtt{b}_k \in B$ to a binary value, to allow the subsequent transitions to use such clocks as binary variables: at each step of the state decoding path, say in the transition from $r_{k-1}$ to $r_k$, the $\mathtt{b}_k$ clock is reset to zero if $\mathtt{b}_k - \mathtt{c}\delta = 0$, and it is reset to one if $\mathtt{b}_k - \mathtt{c}\delta = 1$, hence we have:

$$\langle r_{k-1}, \mathtt{b}_k - \mathtt{c}\delta = 0, \sqcup, \{\mathtt{b}_k := 0\}, r_k \rangle \in \Delta$$
$$\langle r_{k-1}, \mathtt{b}_k - \mathtt{c}\delta > 0, \sqcup, \{\mathtt{b}_k := 1\}, r_k \rangle \in \Delta$$

for all $k \in \{1, \ldots, M+N\}$. By the time the automaton reaches $r_{M+N}$, the value of the $\mathtt{cp}_\mathtt{i}$ clocks directly corresponds to the binary values of the planning problem propositions at the current time in the encoded plan. Hence, the guards of later transitions can directly encode any propositional formula over those propositions. Instead, the other binary clocks $\mathtt{cr}_\mathtt{a}$ will be used to keep track of whether the $a$ action is being executed, that is, "running", at the current time (we exclude self-overlap of actions).

The automaton traverses the state decoding path either at the start or at the end of an action or when the execution ends because the goal was reached. In the former case, we move to the location $d_0$ unconditionally:

$$\langle r_{M+N}, \top, \sqcup, \varnothing, d_0 \rangle \in \Delta$$

Starting from $d_0$, the automaton can decide which subset of the snap actions to execute in parallel by setting the execution clock $\mathtt{cx}_\mathtt{h}$ either to 1 or to 0, for each snap $h$. This can be done polynomially by a construction analogous to the state decoding path. For each $a_k \in A$, we have the following transitions:

$$\langle d_{2k-2}, \mathrm{guard}(a_{k\vdash}), \sqcup, \{\mathtt{cx}_{\mathtt{a}_{k\vdash}} := 1\}, d_{2k-1} \rangle \in \Delta$$
$$\langle d_{2k-2}, \top, \sqcup, \{\mathtt{cx}_{\mathtt{a}_{k\vdash}} := 0\}, d_{2k-1} \rangle \in \Delta$$
$$\langle d_{2k-1}, \mathrm{guard}(a_{k\dashv}), \sqcup, \{\mathtt{cx}_{\mathtt{a}_{k\dashv}} := 1\}, d_{2k} \rangle \in \Delta$$
$$\langle d_{2k-1}, \top, \sqcup, \{\mathtt{cx}_{\mathtt{a}_{k\dashv}} := 0\}, d_{2k} \rangle \in \Delta$$

Intuitively, the automaton can either take a transition that sets to zero the clock relative to a snap action, stating that such an action is not to be executed at the current time, or take the transition checking the guard of the snap action and setting the clock to one. The guard of a starting snap action is:

$$\mathrm{guard}(a_\vdash) = \bigwedge_{p_i \in \mathrm{pre}(a_\vdash)} \mathtt{cp}_\mathtt{i} = 1 \wedge \mathtt{cr}_\mathtt{a} = 0 \wedge \mathrm{sep}(a_\vdash)$$

where:

$$\mathrm{sep}(h) = \bigwedge_{\mathrm{mutex}(b_\vdash, h)} \mathtt{cs}_\mathtt{b} > 0 \wedge \bigwedge_{\mathrm{mutex}(b_\dashv, h)} \mathtt{ce}_\mathtt{b} > 0 \wedge \bigwedge_{\mathrm{mutex}(b_\vdash, h)} \mathtt{cx}_{\mathtt{b}_\vdash} = 0 \wedge \bigwedge_{\mathrm{mutex}(b_\dashv, h)} \mathtt{cx}_{\mathtt{b}_\dashv} = 0$$

The condition expressed by $\mathrm{guard}(a_\vdash)$ checks that the preconditions of $a_\vdash$ are satisfied, and that the action is not already running. Then, $\mathrm{sep}(a_\vdash)$ encodes the time separation between mutex snap actions and prevents the decision to execute two mutex snap actions in the same round. By checking that the corresponding clocks of each mutex snap actions are positive, we enforce the $>0$-separation condition. By replacing the $\mathtt{cs}_\mathtt{b} > 0$ and $\mathtt{ce}_\mathtt{b} > 0$ conditions with $\mathtt{cs}_\mathtt{b} \geq \varepsilon$ and $\mathtt{ce}_\mathtt{b} \geq \varepsilon$, we can easily capture the $\varepsilon$-separation semantics. We indicate the encoding with the former constraints as $\mathcal{T}[\mathcal{P}]_{>0}$ and the latter as $\mathcal{T}[\mathcal{P}]_{\geq\varepsilon}$.

The guard for the end of an action is very similar, but it checks that the action is actually already running and that the action duration is compatible with its duration constraints:

$$\mathrm{guard}(a_\dashv) = \bigwedge_{p_i \in \mathrm{pre}(a_\dashv)} \mathtt{cp}_\mathtt{i} = 1 \wedge \mathtt{cr}_\mathtt{a} = 1 \wedge \mathrm{sep}(a_\dashv) \wedge \mathrm{dur}(a_\dashv)$$

where:

$$\mathrm{dur}(a_\dashv) = \mathtt{cs}_\mathtt{a} \geq L_a \wedge \mathtt{cs}_\mathtt{a} \leq U_a$$

At this point, for each snap action $h$, the value of the execution clock $\mathtt{cx}_\mathtt{h}$ is either 0 or 1 depending on whether $h$ must be executed or not. To apply the effects, we traverse another sequence of locations $e_0, \cdots, e_{2M}$ that apply the effects of each snap action if the corresponding execution clock is set to 1.

$$\langle d_{2M}, \top, \sqcup, \varnothing, e_0\rangle \in \Delta$$
$$\langle e_{2k-2}, \mathtt{cx}_{\mathtt{a}_{k\vdash}} = 1, \sqcup, \text{effects}(a_{k\vdash}), e_{2k-1}\rangle \in \Delta$$
$$\langle e_{2k-2}, \mathtt{cx}_{\mathtt{a}_{k\vdash}} = 0, \sqcup, \varnothing, e_{2k-1}\rangle \in \Delta$$
$$\langle e_{2k-1}, \mathtt{cx}_{\mathtt{a}_{k\dashv}} = 1, \sqcup, \text{effects}(a_{k\dashv}), e_{2k}\rangle \in \Delta$$
$$\langle e_{2k-1}, \mathtt{cx}_{\mathtt{a}_{k\dashv}} = 0, \sqcup, \varnothing, e_{2k}\rangle \in \Delta$$

Note that guard$(\cdot)$ and effects$(\cdot)$ use the $\mathtt{cp}_\mathtt{i}$ clocks to, respectively, enforce the preconditions and produce the effects of the snap action at hand. The effects of the start of an action are defined as follows:

$$\begin{aligned}\text{effects}(a_\vdash) = &\{\mathtt{cr}_\mathtt{a} := 1, \mathtt{cs}_\mathtt{a} := 0\}\\ &\cup \{\mathtt{cp}_\mathtt{i} := 1 \mid p_i \in \text{eff}^+(a_\vdash)\}\\ &\cup \{\mathtt{cp}_\mathtt{i} := 0 \mid p_i \in \text{eff}^-(a_\vdash)\}\end{aligned}$$

Hence, effects$(a_\vdash)$ sets $\mathtt{cr}_\mathtt{a}$ to record that $a$ is executing and resets the clock $\mathtt{cs}_\mathtt{a}$, which is used to record the time elapsed since the last time the $a$ action was started. It also sets the binary clocks $\mathtt{cp}_\mathtt{i}$ according to the positive or negative effects of the action. The effects for the *end* of actions, effects$(a_\dashv)$, are defined similarly, but the clock $\mathtt{ce}_\mathtt{a}$, instead of $\mathtt{cs}_\mathtt{a}$, is reset, to record the time elapsed since the last time the action $a$ ended, and $\mathtt{cr}_\mathtt{a}$ is set to 0.

When all effects have been produced (location $e_{2M}$ has been reached), we can return to location $\ell_\delta$ by resetting the clock $\mathtt{c}\delta$. In this way, we reset the invariant for binary clocks, and the automaton can decide how much time to wait until the next subset of snap actions is executed. In such a transition, we perform a final check to ensure that the over-all conditions of each running action are respected:

$$\langle e_{2M}, \bigwedge_{a\in A} \text{oc}(a), \sqcup, \{\mathtt{c}\delta := 0\}, \ell_\delta\rangle \in \Delta$$

where:

$$\text{oc}(a) = \bigwedge_{p_i \in \text{pre}^{\leftrightarrow}(a)} \mathtt{cr}_\mathtt{a} - \mathtt{cp}_\mathtt{i} \leq 0$$

Intuitively, the formula oc$(a)$ checks that if $a$ is running (that is, $\mathtt{cr}_\mathtt{a} = 1$), then the over-all conditions of $a$ must be true (that is, each $\mathtt{cp}_\mathtt{i}$ must be 1 for each $p_i \in \text{pre}^{\leftrightarrow}(a)$). This implication is captured by the above difference that is false only when $\mathtt{cr}_\mathtt{a} = 1$ and $\mathtt{cp}_\mathtt{i} = 0$. This ensures that, in any accepted run, the guard cannot be false due to a condition being violated.

Finally, if the automaton takes the state decoding path when the goal condition is satisfied and no action is running, then it can transition to the goal state, reaching its objective:

$$\langle r_{M+N}, \bigwedge_{p_i\in G} \mathtt{cp}_\mathtt{i} = 1 \wedge \bigwedge_{a\in A} \mathtt{cr}_\mathtt{a} = 0, \sqcup, \varnothing, \ell^*\rangle \in \Delta$$

Now we can note that the number of locations is polynomial in the size of $\mathcal{P}$. In particular, we have $M + N$ locations in the *state decoding* path, $2M$ locations in the *decision making* path, and $2M$ locations in the *execution path*. Recall that $M$ is the number of actions and $N$ is the number of propositions of the planning problem, hence we have a number of locations that is linear in the number of actions and propositions.

### *4.3. PSPACE-completeness when self-overlap of actions is disallowed*

In the previous section, we introduced the updatable timed automaton $\mathcal{T}[\mathcal{P}]$ corresponding to a temporal planning problem $\mathcal{P}$. Now, we show that every execution reaching $\ell^*$ from $\ell_0$ corresponds to a plan for $\mathcal{P}$, and vice versa.

We start by defining the plan induced by a run of the automaton.

**Definition 4.3** *(Plan induced by a run).* Let $R = \langle \ell_0, \vec{0}\rangle \xrightarrow{\delta_1} \langle \ell_1, v_1\rangle \xrightarrow{a_2} \langle \ell_2, v_2\rangle \cdots \langle \ell^*, v_n\rangle$ be a finite run of $\mathcal{T}[\mathcal{P}]$ starting at $\ell_0$ and ending in $\ell^*$. The plan $\pi[R]$ induced by $R$ is defined as follows:

$$\pi[R] = \{\langle a, t, d\rangle \mid \langle \ell_i, v_i\rangle \xrightarrow{l_{a_\dashv}} \langle \ell_{i+1}, v_{i+1}\rangle,\ d = v_i(\mathtt{cs}_\mathtt{a}),\ t = v_i(\mathtt{c}\gamma) - d\},$$

with $l_{a_\dashv} = \langle \ell_i, \mathtt{cx}_{\mathtt{a}_\dashv} = 1, \sqcup, \text{effects}(a_\dashv), \ell_{i+1}\rangle$.

Each label $l_{a_\dashv}$ corresponds to a decision to terminate an action instance (setting the $\mathtt{cx}_{\mathtt{a}_\dashv}$ clock to 0). It is easy to see that in each state $\langle \ell_i, v_i\rangle$, the value of the global clock $v_i(\mathtt{c}\gamma)$ indicates the time elapsed since the beginning of the run, while $v_i(\mathtt{cs}_\mathtt{a})$ indicates the time elapsed since $a$ has been started, and, in a state where $l_a$ is taken, $v_i(\mathtt{cs}_\mathtt{a})$ indicates the chosen duration of the action $a$.

**Lemma 4.1.** *For any run $R$ of $\mathcal{T}[\mathcal{P}]_{>0}$ reaching $\ell^*$ from $\ell_0$, the plan $\pi[R]$ is valid for $\mathcal{P} = \langle P, A, I, G\rangle$ according to the non-zero separation semantics.*

**Proof.** We proceed by showing that all the conditions of Definition 2.6 hold. Complete proof in the appendix. □

A similar result holds for the $\varepsilon$-separation case.

**Lemma 4.2.** *For any run $R$ of $\mathcal{T}[\mathcal{P}]_{\geq\varepsilon}$ reaching $\ell^*$ from $\ell_0$, the plan $\pi[R]$ is valid for $\mathcal{P}$ according to the $\varepsilon$-separation semantics.*

**Proof.** We proceed as in the proof of the previous lemma, and then we add what is necessary to satisfy Definition 2.7. Complete proof in the appendix. □

**Lemma 4.3.** *Let $\pi = \{\langle t_1, a_1, d_1\rangle, \cdots, \langle t_n, a_n, d_n\rangle\}$ be a valid plan for $\mathcal{P} = \langle P, A, I, G\rangle$ according to the $>0$-separation semantics. Then, there exists a run $R[\pi]$ of $\mathcal{T}[\mathcal{P}]_{>0}$ that reaches $\ell^*$ from $\ell_0$.*

**Proof.** We construct the run $R[\pi]$ by reversing the argument of Lemma 4.1. Complete proof in the appendix. □

**Lemma 4.4.** *Let $\pi = \{\langle t_1, a_1, d_1\rangle, \cdots, \langle t_n, a_n, d_n\rangle\}$ be a valid plan for $\mathcal{P} = \langle P, A, I, G\rangle$ according to the $\varepsilon$-separation semantics. Then, there exists a run $R[\pi]$ of $\mathcal{T}[\mathcal{P}]_{\geq\varepsilon}$ that reaches $\ell^*$ from $\ell_0$.*

**Proof.** The proof is identical to that of Lemma 4.3. □

We conclude the section by formally stating the computational complexity of temporal planning over dense time, when self-overlap of actions is excluded.

**Theorem 4.1.** *Temporal planning over dense time, without self-overlap of actions, is PSPACE-complete for both $>0$-separation and $\varepsilon$-separation semantics.*

**Proof.** PSPACE-hardness follows from PSPACE-completeness of classical planning [14], as it can be easily shown that any classical planning problem can be polynomially encoded into our formulation of temporal planning.

The standard representation of classical planning in STRIPS [32] is, indeed, very close to our encoding of temporal planning except for (i) all actions are assumed to be instantaneous, instead of durative, (ii) there is no distinction between starting and ending conditions or effects and (iii) there are no overall conditions. In particular, classical planning actions are syntactically identical to snap actions. The semantics of STRIPS in its turn is a subset of the temporal one: a STRIPS plan is a sequence of actions $\langle a_1, \cdots, a_n\rangle$ that reaches the goal from the initial state. Formally, it amounts to say that there is a sequence of states $s_0 = I, s_1, \cdots, s_n \subseteq G$ such that $s_i = (s_{i-1} \setminus \text{eff}^-(a_i)) \cup \text{eff}^+(a_i)$. Given the standard formulation of the classical planning problem, it suffices to transform each STRIPS instantaneous action $a = \langle \text{pre}(a), \text{eff}^+(a), \text{eff}^-(a)\rangle$ into a durative action $da = \langle a, \langle\varnothing, \varnothing, \varnothing\rangle, \varnothing, [1, 1]\rangle$. Any valid plan for the resulting temporal planning problem encodes a valid STRIPS plan, where the sequence of starting snap actions, ordered according to the starting time indicated in the temporal plan, are the only actions being considered. Indeed, all the ending snap actions have neither conditions nor effects and rule (iii) of Definitions 2.6 and 2.7 subsumes the validity of STRIPS.

As for the opposite direction, given a valid STRIPS plan $\pi_{STRIPS} = \langle a_1, \cdots, a_n\rangle$, we can turn it into a temporal plan $\pi = \{(da_i, i, 1) \mid a_i \in \pi_{STRIPS}\}$ which is valid either for the $\varepsilon$-separation or for the $>0$-separation semantics, because satisfaction of condition (iii) follows from the validity of the STRIPS plan and all the other rules of Definitions 2.6 and 2.7 are vacuously satisfied.

As for membership, consider any temporal planning problem $\mathcal{P}$. We use the definition of the $\mathcal{T}[\mathcal{P}]$ automaton for one of the two semantics: its size is obviously polynomial and we showed that $\ell^*$ is reachable from $\ell_0$ if and only if $\mathcal{P}$ admits a solution plan according that semantics. Being the reachability problem PSPACE-complete for Updatable Timed Automata, it follows that temporal planning without self-overlapping actions is in PSPACE as well. □

## 5. Allowing self-overlap of actions

This section focuses on the problem of dense-time temporal planning in the case where actions are allowed to overlap with themselves. Here, the distinction between the $\varepsilon$-separation and $>0$-separation semantics plays an important role. Indeed, the problem turns out to be EXPSPACE-complete in the former case and *undecidable* in the latter. The section is structured as follows. First, we recall the definition of two variants of the *tiling problem*, a problem commonly used for hardness proofs in logic and combinatorics (a detailed survey on the topic is provided by van Emde Boas [53]). Then, we prove that the undecidable *unbounded (corridor) tiling problem* can be reduced to temporal planning under $>0$-separation semantics This allows us to conclude its *undecidability*. Finally, by a suitable adaptation of such a reduction, we show

that the simpler *exponential corridor tiling problem* can be reduced to temporal planning under $\varepsilon$-separation semantics, thus proving its EXPSPACE-hardness.

### 5.1. Tiling problems

Generally speaking, tiling problems ask whether it is possible to tile (a portion of) the two-dimensional plane with tiles of a finite set of types in such a way that adjacent tiles satisfy a certain relation, *e.g.*, same color on matching sides. In what follows, for all $n \geq 0$, we denote the set $\{0 \dots n\}$ by $[n]$.

**Definition 5.1** *(Tiling structures).* A (finite) *tiling structure* is a tuple $\mathcal{T} = \langle T, t_0, t^*, H, V \rangle$, where $T$ is a finite set of *tile types*, $t_0 \in T$ and $t_* \in T$ are, respectively, the *initial* and *final* tile types, and $H, V \subseteq T \times T$, are the *horizontal* and *vertical adjacency relations*.

**Definition 5.2** *(Tilings).* A *tiling* for a tiling structure $\mathcal{T} = \langle T, t_0, t^*, H, V \rangle$ is a function $f : [n] \times [h] \to T$, for some $n \geq 0$ and $h \geq 0$, mapping any pair $(i, j) \in [n] \times [h]$ to a tile type $f(i, j) \in T$ such that:

1. $f(0, 0) = t_0$;
2. $f(n, h) = t^*$;
3. for all $x \in [n-1]$ and $y \in [h]$, $f(x, y)\ H\ f(x+1, y)$;
4. for all $x \in [n]$ and $y \in [h-1]$, $f(x, y)\ V\ f(x, y+1)$.

Several interesting combinatorial problems can be defined, with complexities ranging from NP to highly undecidable, depending on which portion of the plane we are asked to tile, in particular, whether some bound on the size of such a portion is given. Here, we introduce the two instances of the tiling problem that are used in the following.

**Definition 5.3** *(Unbounded (corridor) tiling problem).* Let $\mathcal{T} = \langle T, t_0, t^*, H, V \rangle$ be a tiling structure. The *unbounded (corridor) tiling problem* asks whether there exists a tiling $f : [m] \times [h] \to T$, for some $m \geq 0$ and $h \geq 0$.

The unbounded (corridor) tiling problem is as simple to state as complex to solve. Indeed, it can be shown to be undecidable by a direct reduction from the halting problem of Turing machines [53]. By restricting the width of the corridor beforehand, we obtain a decidable variant of it.

**Definition 5.4** *(Exponential corridor tiling problem).* Let $\mathcal{T} = \langle T, t_0, t^*, H, V \rangle$ be a tiling structure, and let $n \geq 0$ be an integer. The *exponential corridor tiling problem* asks whether there exists a tiling $f : [m] \times [h] \to T$, for some $0 \leq m \leq n$ and $h \geq 0$.

Intuitively, the exponential corridor tiling problem recovers decidability by restricting beforehand the width of the area to be tiled. Here, *exponential* refer to the fact that the natural encoding of $n$ in the problem input is binary, hence the upper bound on the corridor width is exponential in the size of the input. For this reason, the problem can be shown to be EXPSPACE-complete, as opposed to the simpler *corridor tiling problem*, where a *unary* encoding of $n$ is assumed, which is PSPACE-complete [53].

### 5.2. The reduction

In the following, we show how to reduce the above-considered tiling problems to temporal planning ones by encoding the tiling functions into plans. In the general case, the reduction introduces an unbounded number of self-overlaps, which is the source of undecidability in the $>0$-separation case.

Let $\mathcal{T} = \langle T, t_0, t^*, H, V \rangle$ be a tiling structure. Fig. 3 shows how an example tiling of nine tiles (top-left of the figure) is flattened into the corresponding temporal plan, which is represented by a set of rectangular intertwined blocks. Underneath the temporal plan, the figure shows the evolution of the values of a number of variables whose role is explained below.

In order to encode a tiling for $\mathcal{T}$ into a temporal plan, we make use of two actions $a^t$ and $a^t_\ell$ for each tile type $t \in T$, assuming a unit duration for each of them. The sequence of such actions in the plan represents the corresponding tiling row-by-row, with actions $a^t_\ell$ positioned only at places that correspond to the left border of the tiling. At the beginning of the plan, a certain number of actions get started before the end of the first one. Such actions represent the first row of the tiling, and thus their number is equal to the number of columns of the tiling. Then, after the end of the first action, the plan proceeds with a strict alternation of starts and ends of actions, in such a way that each action starts immediately after the end of the action corresponding to the tile at the same position in the preceding row. For example, in Fig. 3, action n. 5 represents the tile at position $(1, 1)$, that is, second column and second row, and starts immediately after action n. 2, which represents the tile above it, at position $(0, 1)$, that is, second column and first row. When the final tile has been placed, a special action *end*, which does not correspond to any tile, marks the end of the tiling; the start/end alternation ends with

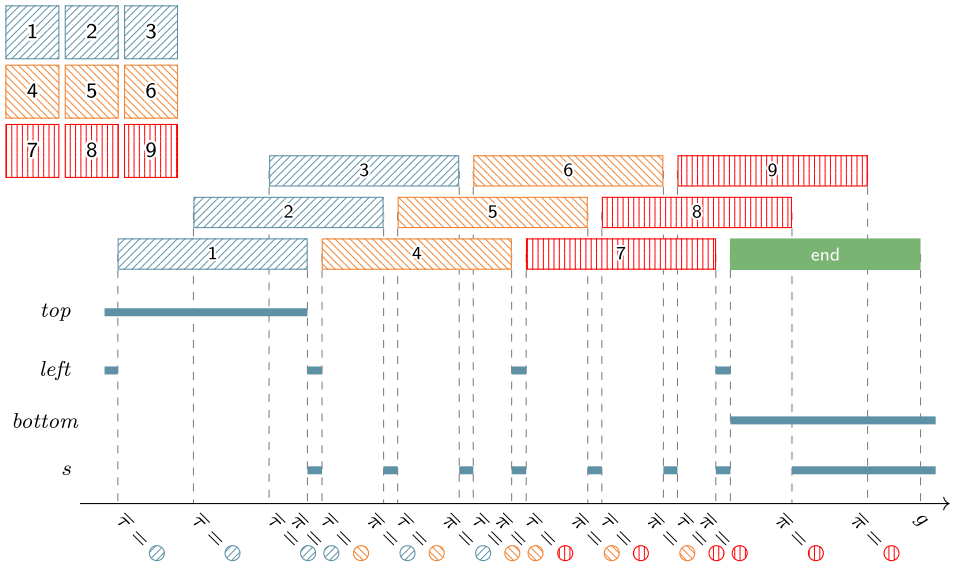



**Fig. 3.** Example of temporal plan encoding a tiling. On the top-left, a square tiling of nine tiles $\{1, \ldots, 9\}$. On the bottom-right, the corresponding temporal plan, with the executed actions on top, and the values of auxiliary fluents below. Colors identify different tile types, while numbers are used to highlight the correspondence between the single tiles in the example tiling and the actions in the plan. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

the end of all the actions that are still running. The end of the *end* action has the effect to set the *goal* fluent, which is the goal of the planning problem, so the plan can terminate.

Two sets of *bookkeeping fluents*, $\bar{\tau} = \langle \tau_0, \ldots, \tau_{n-1} \rangle$ and $\bar{\pi} = \langle \pi_0, \ldots, \pi_{n-1} \rangle$, where $n = \lceil \log_2(|T|) \rceil$, are used to enforce the horizontal and vertical adjacency relations as follows. The $\bar{\tau}$ fluents are updated at the start of each action, so that each action knows the tile placed by the action before, and can enforce the horizontal adjacency relation. On the converse, the $\bar{\pi}$ fluents are updated at the *end* of each action, so that each action knows as well the tile placed by the action immediately above it, and the vertical adjacency relation can be checked.

A number of auxiliary fluents are used to build this machinery. The *top* and *bottom* fluents are used to know whether the execution of the plan is currently positioned at the top or bottom row of the tiling, respectively. The *s* fluent is used to enforce the start/end alternance in the middle part of the plan. Finally, the *left* is used to mark the first column of the plan.

To formally specify such an encoding, we need to introduce some notation. To this end, we make use of the syntax of temporal planning problems with Boolean conditions, as defined in Definition 2.9. For each $t \in T$, a formula $\overline{\tau} = t$ (resp., $\overline{\pi} = t$) is defined as a simple conjunction stating the truth value of the $\tau_i$ (resp., $\pi_i$) fluents corresponding to $t$ (similarly to SAS$^+$ planning state variables [15]). Analogously, the shorthands $\overline{\tau} := t$ and $\overline{\pi} := t$ are used to denote the effect of setting the $\tau_i$ and $\pi_i$ fluents to the tuple of values corresponding to $t$. Finally, we write $(\overline{\tau}, t) \in H$ and $(\overline{\pi}, t) \in V$ to say that the current values of the $\tau_i$ and $\pi_i$ fluents, respectively, are related to $t$ by the $H$ and $V$ relations:

$$(\overline{\tau}, t) \in H \leftrightarrow \bigvee_{\substack{t' \in T \\ (t', t) \in H}} \overline{\tau} = t' \qquad (\overline{\pi}, t) \in V \leftrightarrow \bigvee_{\substack{t' \in T \\ (t', t) \in V}} \overline{\pi} = t'$$

Now, we are ready to formally describe the reduction.

**Definition 5.5** *(Planning problem for a tiling structure).*

Let $\mathcal{T} = \langle T, t_0, t^*, H, V \rangle$ be a tiling structure. The *temporal planning problem associated with* $\mathcal{T}$ is the problem $\mathcal{P}_\mathcal{T} = \langle P_\mathcal{T}, A_\mathcal{T}, I_\mathcal{T}, G_\mathcal{T} \rangle$, where

1. the set of propositions is
$$P_\mathcal{T} = \{\tau_0, \ldots, \tau_{n-1}\} \cup \{\pi_0, \ldots, \pi_{n-1}\} \cup \{s, \mathit{top}, \mathit{bottom}, \mathit{left}, g\},$$
with $n = \lceil \log_2(|T|) \rceil$;
2. the initial state is $I_\mathcal{T} = \{\mathit{top}, \mathit{left}\}$;
3. the goal condition is $G_\mathcal{T} = \{g\}$;
4. the set of *actions* is $A_\mathcal{T} = \{a^t \mid t \in T\} \cup \{a^t_\ell \mid t \in T\} \cup \{\mathit{end}\}$, where each action is defined as follows:
   (a) for each $a \in A_\mathcal{T}$, we have $L_a = U_a = 1$ and $\mathrm{pre}^{\leftrightarrow}(a) = \top$;

(b) for each $t \in T$, conditions and effects of the actions $a^t$ and $a^t_\ell$ (both denoted as $a^t_*$ below) are defined as follows:

$$\begin{aligned}
\text{pre}(a^t_{*\vdash}) &= \neg bottom \\
&\wedge left && \leftarrow \text{only for } a^t_\ell \\
&\wedge \neg left && \leftarrow \text{only for } a^t \\
&\wedge \neg(top \wedge left) && \leftarrow \text{except for } a^{t_0} \\
&\wedge \neg top \rightarrow s \\
&\wedge \neg top \rightarrow (\overline{\pi}, t) \in V \\
&\wedge \neg left \rightarrow (\overline{\tau}, t) \in H \\
\text{eff}(a^t_{*\vdash}) &= \neg s \wedge \neg left \wedge \overline{\tau} := t \\
\text{pre}(a^t_{*\dashv}) &= \neg bottom \rightarrow \neg s \\
\text{eff}(a^t_{*\dashv}) &= s \wedge \neg top \wedge \overline{\pi} := t \\
&\wedge left && \leftarrow \text{only for } a^t_\ell
\end{aligned}$$

Finally, conditions and effects of the *end* action are defined as follows:

$$\begin{aligned}
\text{pre}(end_\vdash) &= left \\
\text{eff}(end_\vdash) &= bottom \\
\text{pre}(end_\dashv) &= (\overline{\pi} = t^*) \\
\text{eff}(end_\dashv) &= g
\end{aligned}$$

A note on the size of the encoding of Definition 5.5 is due. As mentioned, we used the syntactic extension of Boolean conditions as defined in Definition 2.9, which may in principle require an exponential blowup to be translated into the set-theoretic conditions of Definition 2.1. Notably, the proof of undecidability of Theorem 5.1 would not be affected by any sort of size blowup, since it provides a reduction from an undecidable problem. However, for this encoding to be used as a polynomial reduction in the proof of Theorem 5.2, we need to prove that the exponential blowup does *not* happen in this case.

**Lemma 5.1.** *Let $\mathcal{T}$ be a tiling structure. The temporal planning problem $\mathcal{P}_\mathcal{T}$ associated with $\mathcal{T}$, as defined in Definition 5.5, can be mapped into a temporal planning problem without Boolean conditions with only a polynomial increase in size.*

**Proof.** The well-known compilation of Boolean conditions into set-theoretic conditions involves the translation of the conditions of each snap action into *disjunctive normal form* (DNF), and a duplication of each action with a new copy of itself for each disjunct of the resulting formula. This step is the source of the exponential blowup that may arise in general. To see why it does not happen in this case, observe that (i) all the conditions of the problem in Definition 5.5 are in essence in *conjunctive normal form*, and (ii) the number of clauses is fixed and it does not depend on the size of the input problem. In particular, in $\text{pre}(a^t_\vdash)$, there are always only five clauses, and only the last two clauses grow linearly in the number of tiles of $\mathcal{T}$. The number of disjuncts in the DNF of $\text{pre}(a^t_\vdash)$ thus grows quadratically in the number of tiles. □

By making use of the encoding of Definition 5.5, we prove the undecidability of the temporal planning problem when the *>0-separation* semantics is assumed.

**Theorem 5.1** *(Undecidability). Temporal planning over dense time, with* >0-separation *semantics and* self-overlap *of actions, is* undecidable.

**Proof.** We prove that the encoding described in Definition 5.5 is a sound and complete reduction from the unbounded (corridor) tiling problem to temporal planning with >0-separation semantics and self-overlap of actions. Since the former problem is undecidable [53], the latter is undecidable as well.

Let $\mathcal{T} = \langle T, t_0, t^*, H, V\rangle$ be a tiling structure and let $\mathcal{P}_\mathcal{T} = \langle P_\mathcal{T}, A_\mathcal{T}, I_\mathcal{T}, G_\mathcal{T}\rangle$ be its associated temporal planning problem as specified in Definition 5.5.

($\Rightarrow$) Suppose there is a tiling $f : [n] \times [h] \rightarrow T$ for $\mathcal{T}$, for some $n, h > 0$. We build a solution plan for $\mathcal{P}_\mathcal{T}$ by laying out the actions that represent the tiling in a row-by-row layout (recall Fig. 3). For $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, h\}$, let $t^{i,j} = f(i, j)$ be the tile placed at position $(i, j)$ by the tiling $f$ and let $a^{i,j} = a^{t_{i,j}}$, for $j > 0$, and $a^{i,j} = a^{t_{i,j}}_\ell$, for $j = 0$, be the corresponding

action. Note that actions $a^t_\ell$ are placed along the left border of the tiling. Moreover, let $a^{0,h+1} = end$ and $\delta = 1/n$. Formally, the plan $\pi_{\mathcal{T}}$ representing $\mathcal{T}$ can be defined as follows:

$$\pi_{\mathcal{T}} = \bigcup_{\substack{i\in\{0,\dots,n\}\\ j\in\{0,\dots,h\}}} \{\langle a^{i,j}, j\cdot(1+\frac{\delta}{2}) + \delta\cdot i, 1\rangle\} \cup \{\langle a^{0,h+1}, (h+1)\cdot(1+\frac{\delta}{2}), 1\rangle\}$$

It can be checked that mutex events in $\pi_{\mathcal{T}}$ are correctly separated as required by the $>0$-separation semantics defined in Definition 2.6.

Intuitively, $\delta$ is the amount of time that separates the start of consecutive actions of a given row $j$. In this way, the start of all the actions of the $j$-th row are contained inside its first action $a^{0,j}$. When $a^{0,j}$ ends, the next row then begins with $a^{0,j+1}$, which starts $\frac{\delta}{2}$ later, just before the end of $a^{1,j}$. All the actions have unit duration as specified by $\mathcal{P}_{\mathcal{T}}$.

To start with, we observe a few facts.

1. In the whole plan, excepting during $a^{0,0}$ and $end = a^{0,h+1}$, there is a strict alternation of starts and ends of actions, that is, each start is placed between two ends and each end is placed between two starts.
2. The effect of $a^t_{\ell\dashv}$ makes proposition *left* hold only between the end of $a^{0,j}$ and the start of $a^{0,j+1}$, for all $j \in [h]$.

Then, we can check that the preconditions of all the actions are satisfied.

Let us consider $\mathrm{pre}(a_\vdash)$, for all $a \in A_{\mathcal{T}}$.

1. $\neg bottom$ is always satisfied since *bottom* is initially false and it is only set to true by the start of *end*, which, by construction, is the last action to start in the plan.
2. *left* (for $a^t_\ell$) and $\neg left$ (for $a^t$) are satisfied by the way the actions are placed, with $a^t_\ell$ placed always and only at the left border of the tiling.
3. *top* and *left* are both initially true, but $\neg(top \wedge left)$ is *not* required by $a^{t_0} = a^{0,0}$. After that, all the $a^{i,j}{}_\vdash$ set *left* to false. Proposition *left* is set to true again by all the $a^{0,j}{}_\dashv$, which, however, also set $\neg top$, and thus *top* and *left* never become true together again.
4. $\neg top \rightarrow s$ is satisfied thanks to the strict start/end alternation enforced in the plan starting from $a^{0,0}{}_\dashv$, which is the first ending snap action in the plan and, among its effects, sets $\neg top$.
5. By construction, for all $i > 0$ and $j \geq 0$, the action $a^{i,j}$ starts immediately after the end of $a^{i-1,j}$, hence after $\bar{\pi}$ have just been set to $t^{i-1,j}$. Consequently, since $f$ is a tiling for $\mathcal{T}$, we know that $(t^{i-1,j}, t^{i,j}) \in H$, and thus $(\bar{\pi}, t^{i,j}) \in H$ is satisfied.
6. By construction, for all $i \geq 0$ and $j > 0$, the action $a^{i,j}$ is the first to start after the start of $a^{i,j-1}$, hence after $\bar{\tau}$ have been set to $t^{i,j-1}$. Consequently, since $f$ is a tiling for $\mathcal{T}$, we know that $(t^{i,j-1}, t^{i,j}) \in V$, and thus $(\bar{\tau}, t^{i,j}) \in V$ is satisfied.

The case of $\mathrm{pre}(a_\dashv)$ is similar: $\neg s$ is set as the effect of each $a^{i,j}{}_\vdash$, and each action ends after the start of some other, excepting after $end_\vdash$, which, however, sets *bottom* to true. Hence, $\neg bottom \rightarrow \neg s$ is satisfied at the end of each action.

Let us consider now start and end conditions of the *end* action. Since $end = a^{0,h+1}$, the action starts when *left* is true, and thus $\mathrm{pre}(end_\dashv)$ is satisfied. Moreover, *end* ends just after $a^{n,h}$, which sets $\bar{\pi}$ to $t^{n,h} = t^*$, hence $\mathrm{pre}(end_\dashv)$ is satisfied.

Finally, we observe that the goal condition is reached, since *end* is executed, which implies that proposition $g$ is set to true as required by the goal condition.

($\Leftarrow$) Suppose there is a solution plan $\pi$ for $\mathcal{P}_{\mathcal{T}}$. We show how to extract a candidate tiling $f$ from $\pi$, and then that $f$ is indeed a tiling for $\mathcal{T}$. We first show that $\pi$ necessarily has a certain structure, that we use afterwards to extract the tiling $f$. In particular, we show that $\pi$ consists of a sequence of starts of a number of actions, followed by a segment of strict alternation of starts and ends of actions, followed by a sequence of ends. To this end, we observe that:

1. The start of any action requires $s$ only if $\neg top$, and thus, at the beginning of the plan, we may have an unbounded sequence of starts.
2. Such a sequence stops at the end of the first action, which sets $\neg top$. From there, each start requires $s$ and sets $\neg s$, while each end requires $\neg s$ and sets $s$, thus requiring a strict alternation of starts and ends.
3. The end of the distinguished *end* action is the only event that sets the $g$ fluent, which is the goal of the problem. Since $\pi$ is a solution plan, sooner or later *end* is executed.
4. The *end* action is the last action to start, since its start sets *bottom*, and the start of any other action requires $\neg bottom$. Moreover, the end of *end* is the last event of the plan, since all the actions have the same unit duration, and thus all the actions started before *end* have time to conclude.
5. During the execution of *end*, the sequence of the ends of all the actions started before and not yet concluded takes place.

Now, let $a_0, \dots, a_{k-1}$ be the sequence of actions of the plan in the order in which they are started, and let $n$ be the number of actions that start before the end of $a_0$, that is, the first sequence of starts. The *left* fluent, initially true, is set to

false by the start of any action, and thus it is immediately set to false by $a_0$. However, actions $a^t_\ell$, which start only when *left* is true, set *left* true again at their ends. Note that $a_0$ must be of the form $a^t_\ell$, hence the end of $a_0$ sets *left* again. This happens after $n$ starts. Then, *left* is set to false again by the $n+1$ start event, to be set true again by the end following the $2n$-th start, and so on. Hence, *left* is set to true after every $n$ start events. Finally, note that since *end* requires *left*, it can only be started after $m \cdot n$ starts, for some $m > 0$, and thus the total number $k$ of actions is of the form $k = m \cdot n + 1$.

We can now explain how the tiling $f$ is extracted from $\pi$. It is an $n \times m$ tiling $f : [n] \times [m] \to T$ such that $f(i, j) = t_{i,j}$, where $a_{i \cdot n+j} = a^{t_{i,j}}_*$ (where $a^{t_{i,j}}_*$ is either $a^{t_{i,j}}$ or $a^{t_{i,j}}_\ell$). Intuitively, $n$ is the number of columns of the tiling, the start of each action places a tile, and *left* is true at the start of every action at the leftmost border of the tiling.

It remains to check that $f$ is indeed a tiling for $\mathcal{T}$. To do this, consider that the start of each action $a^t_*$ sets the $\bar{\tau}$ fluents to $t$, and requires that $(\bar{\tau}, t) \in H$. The $\bar{\tau}$ fluents thus track which tile is placed in the current position and are used to check the horizontal adjacency relation. The vertical adjacency relation is enforced in a similar, but slightly more complex, way. The end of $a^t$ sets $\bar{\pi}$ to $t$. If $a^t = a_i$, then this happens right before the start of $a_{i+n}$, that is, before the start of the action/tile at the same column in the following row. Thus, when the start of $a_{i+n} = a^{t'}$ checks that $(\bar{\pi}, t') \in V$, it correctly checks the vertical adjacency relation. Thus, $f$ is a tiling for $\mathcal{T}$. □

The above construction can be adapted to the *$\varepsilon$-separation* semantics, using a reduction from the computationally simpler *exponential corridor tiling problem*.

**Theorem 5.2.** *Temporal planning over dense time, with* $\varepsilon$-separation *semantics and* self-overlap *of actions, is* EXPSPACE*-complete.*

**Proof.** Let $\mathcal{P} = (P, A, I, G)$ be a dense-time temporal planning problem with *$\varepsilon$-separation* semantics and *self-overlap* of actions. It can be easily shown that it belongs to EXPSPACE. An equivalent problem of exponential size $\mathcal{P}' = (P', A', I', G')$, with $|\mathcal{P}'| \in \mathcal{O}(2^{|\mathcal{P}|})$, can indeed be obtained from it such that $\mathcal{P}$ has a solution plan admitting self-overlap of actions if and only if $\mathcal{P}'$ admits a plan without any self-overlap. Then, by Theorem 4.1, $\mathcal{P}'$ can in turn be solved in space polynomial in $|\mathcal{P}'|$, hence exponential in $|\mathcal{P}|$. Let $D_{max}$ be the maximum duration allowed for any action in $A$. $\mathcal{P}'$ can obtained from $\mathcal{P}$ by duplicating each action $a \in A$ into $k$ copies $a_1, \ldots, a_k$, where $k = D_{max}/\varepsilon$, which corresponds to the maximum number of overlapping instances of the same action (note that the start of an action is mutex with itself).

To show that the problem is EXPSPACE-hard, we exploit a reduction from the *exponential corridor tiling problem*. Given a tiling structure $\mathcal{T}$ and an integer $n > 0$, the structure can be encoded into a temporal planning problem $\mathcal{P}$ that admits a solution plan if and only if $\mathcal{T}$ admits an $m$-tiling for some $m \leq n$. The encoding is the same as the one described in Definition 5.5, and thus we do not repeat it here. We just remark that, by Lemma 5.1, the size of the resulting planning problem is polynomial in the size of the tiling structure, even when expressed with set-theoretic conditions. Now, since the maximum number of columns in the tiling is now known in advance, the resulting planning problem can be solved with $\varepsilon$-separation by choosing a suitable value for $\varepsilon$. First, note that all snap actions produced in the reduction are *mutex*, since their effects and preconditions share at least the flag bit $s$. Then, at worst, the action corresponding to the first tile of each row has to contain the end of the $m$ actions of the previous row, and the start of the other $m-1$ tiles of the row. This divides the unit time interval of the action into $2m-2$ subintervals, hence $\varepsilon = \frac{1}{(2n-2)}$ ensures enough granularity for any tiling with only and at most $n$ columns. □

## 6. Conclusions

The paper studies the computational complexity of temporal planning, as specified by PDDL 2.1, and it provides a comprehensive picture of the complexity of the problem under different semantic interpretations. More precisely, we distinguish between two ways of separating mutually exclusive events: they can be constrained to be separated by a given minimum quantum of time ($\varepsilon$-separation) or just by any positive amount of time ($>$0-separation). Moreover, we consider both the case where actions are allowed to overlap with running instances of themselves and the case where such an overlapping is disallowed.

The outcome of the analysis is the following. Dense-time temporal planning is PSPACE-complete, that is, no harder than classical planning, when self-overlap is forbidden, regardless of the adopted form of separation of mutually exclusive events. When self-overlap is allowed, the problem becomes EXPSPACE-complete with $\varepsilon$-separation and even *undecidable* with $>$0-separation. These results clarify the computational consequences of different choices in the PDDL 2.1 semantics, which were vague until now.

It is worth pointing out that the proof of PSPACE-membership makes use of an original polynomial-size encoding of action-based temporal planning into *timed automata*. In analogy to (or in combination with) other contributions, where timed automata or their extensions have been exploited to deal with planning problems [9,54], this may lead to novel approaches of practical interest.

We would like to remark that we focused our attention on the PDDL 2.1 fragment, where only conjunctive preconditions and goals are allowed, and actions do not contain conditional effects. Much as it happens in the case of STRIPS [14], an interesting research question is whether the obtained results carry on to the more general setting. We expect the problem to be challenging, in particular, the support for conditional effects, as their specification leads to reasoning over many

contextual situations, and it is not clear how such a generalization, and the necessary reformulation of mutex conditions, may affect the temporal reasoning task. Still looking at action-based languages for temporal planning, we would also like to understand whether our constructions can be adapted for analyzing the computational complexity of NDL [49] that, to the best of our knowledge, has not yet been explored.

Last but not least, we would like to systematically compare the achieved results with those obtained for timeline-based planning over dense time [11,13], in analogy to what has been already done in the discrete setting [34].

## Declaration of competing interest

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

## Acknowledgements

Andrea Micheli and Enrico Scala have been supported by AIPlan4EU, a project funded by EU Horizon 2020 research and innovation programme under GA n. 101016442 (since 2021), by the Autonomous Province of Trento in the scope of L.P. n. 6/1999 with grant MAIS (n. 2017-D323-00056 del. n. 941 of 16/06/2017, until 2019) and by EIT Digital within the *AWARD* project (until 2019). Nicola Gigante and Angelo Montanari have been supported by the PRID project *ENCASE - Efforts in the uNderstanding of Complex interActing SystEms*, funded by the University of Udine, and by the $\mathfrak{i}\mathbb{N}\delta\mathrm{A}\wp$ GNCS project *Strategic Reasoning and Automatic Synthesis of Multi-Agent Systems*.

## Appendix A. Proofs of Section 4

**Lemma 4.1.** *For any run $R$ of $\mathcal{T}[\mathcal{P}]_{>0}$ reaching $\ell^*$ from $\ell_0$, the plan $\pi[R]$ is valid for $\mathcal{P} = \langle P, A, I, G\rangle$ according to the non-zero separation semantics.*

**Proof.** We proceed by showing that all the conditions of Definition 2.6 hold for $\pi[R] = \{\langle t_1, a_1, d_1\rangle, \cdots, \langle t_n, a_n, d_n\rangle\}$. Let $\pi[R]^{ind} = \langle\langle t'_1, B_1\rangle, \cdots, \langle t'_m, B_m\rangle\rangle$.

(i) We have to show that $\forall i \in \{1 \cdots n\}. L_{a_i} \leq d_i \leq U_{a_i}$.
The duration of each action instance $a$ is set (by Definition 4.3) to the value of $\mathtt{cs_a}$ when transition $l_{a_\dashv} = \langle \ell_i, \mathtt{cx_{a_\dashv}} = 1, \llcorner, \mathrm{effects}(a_\dashv), \ell_{i+1}\rangle$ is taken. Then, the only way to satisfy the guard $\mathtt{cx_{a_\dashv}} = 1$ is to have a previous transition labeled with $g_a = \langle \ell_k, \mathrm{guard}(a_\dashv), \llcorner, \{\mathtt{cx_{a_\dashv}} := 1\}, \ell_w\rangle$ without a positive time elapse between the two transitions. Since the run $R$ is valid, then the guard of $g_a$ is satisfied and $\mathrm{guard}(a_\dashv)$ imposes the condition $\mathrm{dur}(a_\dashv) = \mathtt{cs_a} \geq L_a \wedge \mathtt{cs_a} \leq U_a$, hence fulfilling the duration constraint of the plan.

(ii) We have to show that there are no $h, z \in B_i$, with $h \neq z$, for some $i \in \{1, \cdots, m\}$, such that $\mathrm{mutex}(h, z)$.
By contradiction, suppose that there are such $h$ and $z$. Then, there must be a transition $\langle \ell_i, v_i\rangle \xrightarrow{l} \langle \ell_{i+1}, v_{i+1}\rangle$ in $R$ with $l = \langle \ell_i, \mathtt{cx_h} = 1, \llcorner, \mathrm{effects}(h), \ell_{i+1}\rangle$. In turn, the only way to satisfy the guard $\mathtt{cx_h} = 1$ is to have a previous transition with label $l' = \langle d_k, \mathrm{guard}(h), \llcorner, \{\mathtt{cx_h} := 1\}, d_{k+1}\rangle$ for some $k$, without a positive time elapse between $l$ and $l'$. To satisfy the guard of $l'$, the formula $\mathrm{sep}(h)$ must be satisfied. This implies that $\mathtt{cz} > 0$, because $\mathrm{mutex}(h, z)$, but then $z \notin B_i$, hence the contradiction.

(iii) We have to show that, given $s_0 = I$, for all $i \in \{1, \ldots, m\}$, it holds that:
(a) $\bigcup_{h \in B_i} \mathrm{pre}(h) \subseteq s_i$,
(b) $s_i = (s_{i-1} \setminus \bigcup_{h \in B_i} \mathrm{eff}^-(h)) \cup \bigcup_{h \in B_i} \mathrm{eff}^+(h)$, and
(c) $G \subseteq s_m$.
Given a clock valuation $v_i$, we define the decoded state as $s[v_i] = \{p_j \mid p_j \in P \wedge v_i(\mathtt{cp_j}) = 1\}$. The first transition of $R$ must obviously be $\langle \ell_0, \vec{0}\rangle \xrightarrow{l} \langle \ell_\delta, v_0\rangle$, with $l = \langle \ell_0, \top, \llcorner, \{\mathtt{cp_i} := 1 \mid p_i \in I\}, \ell_\delta\rangle$. Clearly, $s_0 = s[v_0] = I$ given the updates of $l$. Now, for each $\langle t, B\rangle \in \pi[R]^{ind}$, let $s_i = s[v_k]$, where $\langle \ell_{k-1}, v_{k-1}\rangle \xrightarrow{l_h} \langle \ell_k, v_k\rangle$ with $l_h = \langle \ell_i, \mathtt{cx_h} = 1, \llcorner, \mathrm{effects}(h), \ell_{i+1}\rangle$, $v_k(\mathtt{c}\gamma) = t$, and $h \in B$. Note that $s_i$ does exist by Definition 4.3.
It is easy to see that Item (b) holds, that is, $s_i = (s_{i-1} \setminus \bigcup_{h \in B_i} \mathrm{eff}^-(h)) \cup \bigcup_{h \in B_i} \mathrm{eff}^+(h))$: it directly follows from the formulae $\mathrm{effects}(h)$, for each $h \in B$, and the inertia of the clocks, that remain still during each transition unless explicitly updated. No other update touches the binary clocks, except for the state update that only scales them back to either 0 or 1 without changing their binary meaning.
Let us show now that Item (a), *i.e.*, $\bigcup_{h \in B_i} \mathrm{pre}(h) \subseteq s_i$, for all $i \in \{1, \ldots, m\}$, holds as well. Following the same reasoning path of point (iii), there is a transition $\langle \ell_i, v_i\rangle \xrightarrow{l} \langle \ell_{j+1}, v_{j+1}\rangle$ in $R$ with $l = \langle \ell_j, \mathtt{cx_h} = 1, \llcorner, \mathrm{effects}(h), \ell_{j+1}\rangle$. In turn, the only way to satisfy the guard $\mathtt{cx_h} = 1$ is to have a previous transition with label $l' = \langle d_k, \mathrm{guard}(h), \llcorner, \{\mathtt{cx_h} := 1\}, d_{k+1}\rangle$ for some $k$, without a positive time elapse between $l$ and $l'$. Now, due to the formula $\mathrm{guard}(h)$, it must be the case that $\mathrm{pre}(h) \subseteq s_i$, because no snap action mutex with $h$ can be executed between $l$ and $l'$ in $R$.

Finally, we show that Item (c), *i.e.*, $G \subseteq s_m$, holds. Since the run $R$ terminates in $\ell^*$, the last transition is labeled as $\langle r_{M+N}, \bigwedge_{p_i \in G} \mathtt{cp_i} = 1 \wedge \bigwedge_{a \in A} \mathtt{cr_a} = 0, \_, \varnothing, \ell^* \rangle$, and thus $G \subseteq s_m$, as the guard $\bigwedge_{p_i \in G} \mathtt{cp_i} = 1$ must be satisfied.

(iv) To complete the proof, we need to show that, for all $c = \langle t, a, d \rangle \in \pi$, we have that $\forall \pi^{ind}_{\vdash}(c) \leq k < \pi^{ind}_{\dashv}(c).\, \mathrm{pre}^{\leftrightarrow}(a) \subseteq s_k$. Along run $R$, $\mathtt{cr_a} - \mathtt{c}\delta = 1$ between consecutive transitions $\langle \ell_i, \mathtt{cx}_{a_\vdash} = 1, \_, \mathrm{effects}(a_\vdash), \ell_{i+1} \rangle$ and $\langle \ell_j, \mathtt{cx}_{a_\dashv} = 1, \_, \mathrm{effects}(a_\dashv), \ell_{j+1} \rangle$ (these transitions must come in pairs due to the guard($a_\vdash$) formula). Each of these pairs corresponds to $\pi^{ind}_{\vdash}(c)$ and $\pi^{ind}_{\dashv}(c)$. Therefore, $\mathrm{pre}^{\leftrightarrow}(a) \subseteq s_k$ for each $\pi^{ind}_{\vdash}(c) \leq k < \pi^{ind}_{\dashv}(c)$ because in all the clock valuations $v_k$ corresponding to each $s_k$, $v_k(\mathtt{cr_a}) = 1$ and the formula oc($a$) must be satisfied, forcing $v_k(\mathtt{cp_i})$ to be 1 for all $p_i \in \mathrm{pre}^{\leftrightarrow}(a)$. □

**Lemma 4.2.** *For any run $R$ of $\mathcal{T}[\mathcal{P}]_{\geq\varepsilon}$ reaching $\ell^*$ from $\ell_0$, the plan $\pi[R]$ is valid for $\mathcal{P}$ according to the $\varepsilon$-separation semantics.*

**Proof.** We proceed as in the proof of the previous lemma, then we add the following part to prove the satisfaction of Definition 2.7.

(v) For all $i, j \in \{1, \cdots, m\}$, with $i \neq j$, such that there exist $h \in B_i$ and $z \in B_j$, with mutex($h, z$), we have that $|t'_i - t'_j| \geq \varepsilon$. By contradiction, suppose that there are $i, j \in \{1, \cdots, m\}$, with $i \neq j$, such that there exist $h \in B_i$ and $z \in B_j$ with mutex($h, z$) and $|t_i - t_j| < \varepsilon$. Without loss of generality, suppose $i < j$. Then, there is a transition $\langle \ell_i, \vec{i} \rangle \xrightarrow{l} \langle \ell_{i+1}, v_{i+1} \rangle$ in $R$ with $l = \langle \ell_i, \mathtt{cx_h} = 1, \_, \mathrm{effects}(h), \ell_{i+1} \rangle$. In turn, the only way to satisfy the guard $\mathtt{cx_h} = 1$ is to have a previous transition with label $l' = \langle d_{2k-2}, \mathrm{guard}(h), \_, \{\mathtt{cx_h} := 1\}, d_{2k-1} \rangle$ without a positive time elapse between $l$ and $l'$. To satisfy the guard of $l'$, the formula sep($h$) must be satisfied. This implies that $\mathtt{cz} \geq \varepsilon$, as mutex($h, z$), hence the contradiction, because the only way to satisfy $\mathtt{cz} \geq \varepsilon$ is to have $|t_i - t_j| \geq \varepsilon$. □

**Lemma 4.3.** *Let $\pi = \{\langle t_1, a_1, d_1 \rangle, \cdots, \langle t_n, a_n, d_n \rangle\}$ be a valid plan for $\mathcal{P} = \langle P, A, I, G \rangle$ according to the $>0$-separation semantics. Then, there exists a run $R[\pi]$ of $\mathcal{T}[\mathcal{P}]_{>0}$ that reaches $\ell^*$ from $\ell_0$.*

**Proof.** We construct the run $R[\pi]$ by reversing the argument of Lemma 4.1. We will omit the time-elapse transitions with duration 0, since all the locations of $\mathcal{T}[\mathcal{P}]_{>0}$, with the exception of $\ell_\delta$, are urgent. Let $\pi^{ind} = \langle \langle t_1, B_1 \rangle, \cdots, \langle t_m, B_m \rangle \rangle$.

By Definition 2.6, we know that there exists a sequence of states $s_0 \cdots, s_m$ such that $s_0 = I$, $s_i = (s_{i-1} \setminus \bigcup_{h \in B_i} \mathrm{eff}^-(h)) \cup \bigcup_{h \in B_i} \mathrm{eff}^+(h)$, and $G \subseteq s_m$.

Moreover, let $\langle \rho_0, \cdots, \rho_k \rangle$ a sequence of sets of actions where $\rho_i = \{a \mid \#\{j \mid a_\vdash \in B_j, j \leq i\} - \#\{j \mid a_\dashv \in B_j, j \leq i\} = 1\}$. Intuitively, since we are disallowing self-overlapping of actions, $\rho_i$ contains all the actions running while the system is in state $s_i$. Equivalently, one can assume to have a fresh predicate for each action, and have it set to true when the action it refers to is started and set to false when it does terminate; in this case, $\rho_i$ would be the set of actions having their fluent set to true in $s_i$. Clearly, $\rho_0 = \rho_k = \varnothing$, because at the beginning and at the end of the plan no action can be running.

We proceed by induction on $\pi^{ind}$ showing that starting from an automaton state $\langle \ell_\delta, v_i \rangle$, with $s_i = s[v_i]$ and $\rho_i = \{a \mid v_i(\mathtt{cr_a}) = 1\}$, we can construct the run $R[\pi]$ that reaches $\ell^*$.

The run $R[\pi]$ invariantly starts with the transition $\langle \ell_0, \vec{0} \rangle \xrightarrow{l} \langle \ell_\delta, v_1 \rangle$, with $l = \langle \ell_0, \top, \_, \{\mathtt{cp_i} := 1 \mid p_i \in I\}, \ell_\delta \rangle$. This transition can always be taken because the guard is a tautology. It is easy to see that $s[v_1] = s_0$ as in the proof of Lemma 4.1.

Let $\langle \ell_\delta, v_i \rangle$ be the last state of the prefix of the run constructed so far. We can deterministically add to $R[\pi]$ a sequence of transitions that traverse the state-decoding portion of the timed automaton till location $r_{M+N}$. This is always possible, because the two alternative transitions between location $r_i$ and $r_{i+1}$ are incompatible one another due to their guards and either of the two guards is surely satisfied.

Let the new last state of the run $R[\pi]$ be $\langle r_{M+N}, v_j \rangle$. It is easy to see that $s[v_i] = s[v_j]$ because the only modification operated by the state-decoding part of the automaton is to bring back the binary clocks to either 0 or 1.

*Base case.* If $\pi^{ind}$ is empty, then $s_0 = s_k$ and we can add to $R[\pi]$ the transition $\langle r_{M+N}, v_j \rangle \xrightarrow{l} \langle \ell^*, v_k \rangle$ with $l = \langle r_{M+N}, \bigwedge_{p_i \in G} \mathtt{cp_i} = 1 \wedge \bigwedge_{a \in A} \mathtt{cr_a} = 0, \_, \varnothing, \ell^* \rangle$. This transition must be possible, as, being the plan valid according to Definition 4.3, $G \subseteq s[v_j]$ and $\bigwedge_{a \in A} \mathtt{cr_a} = 0$ ($\mathtt{cr_a}$ is set to 1 when starting an action and reset to 0 upon action termination, and in $\pi^{ind}$ each $a_\vdash$ is paired with a $a_\dashv$). Hence, if $\pi^{ind}$ is empty, all $\mathtt{cr_a}$ must be 0.

*Inductive case.* Let $\langle t_i, B_i \rangle$ the first element of $\pi^{ind}$. First, we add to the run $R[\pi]$ a sequence of transitions bringing the automaton from $\langle r_{M+N}, v_j \rangle$ to $\langle d_{2M}, v_q \rangle$. From $\langle r_{M+N}, v_j \rangle$, we can immediately move to $\langle d_0, v_j \rangle$ as the guard is true and no clock is reset. Then, for each $h \in B_i$, we take the transition $\langle d_k, \mathrm{guard}(h), \_, \{\mathtt{cx_h} := 1\}, d_{k+1} \rangle$, and for each $z \notin B_i$, we take the transition $\langle d_k, \top, \_, \{\mathtt{cx_z} := 0\}, d_{k+1} \rangle$. Note that the latter is trivially enabled, as the guard is a tautology, while, in order to take the former transition. we need to show that guard($h$) is satisfied. Since $(\bigcup_{h \in B_i} \mathrm{pre}(h)) \subseteq s_i$, it follows that $\mathrm{pre}(h) \subseteq s_i$. Moreover, it is clear that in all the states $\langle d_i, w_i \rangle$ belonging to the decision-making section of the automaton, the valuation of all the clocks, except for $\mathtt{cx_h}$, is the same as the one in $v_j$, and thus, since $s_i = s[v_j] = s[w_i]$, guard($h$) must be satisfied. Note that by constraint (ii) and constraint (i) of Definition 2.6, subformula sep($h$) and subformula $dur(h)$ are satisfied, respectively.

Since moving from $d_{2M}$ to $e_0$ is always possible, we can conclude that from $\langle r_{M+N}, v_j\rangle$ we can reach a state $\langle e_0, w_0\rangle$. It is then possible to traverse the states from $\langle e_0, w_0\rangle$ to $\langle e_{2M}, w_{2M}\rangle$ deterministically, because either the guard of the transition $\langle e_{k-1}, \mathtt{cx_h} = 1, \llcorner, \text{effects}(h), e_k\rangle$ or the one of $\langle e_k, \mathtt{cx_h} = 0, \llcorner, \varnothing, e_k\rangle$ is enabled at each step. The run constructed so far, indeed, sets all $\mathtt{cx_h}$ either to 1 (if $h \in B_i$) or to 0 (otherwise). Now, it is easy to see that $s_{i+1} = s[w_{2M}]$, as we applied all the updates in effects($h$), for each $b \in B_i$. Moreover, $\rho_{i+1} = \{a \mid v_i(\mathtt{cr_a}) = 1\}$, as $\mathtt{cr_a}$ is set to 1 when $h = a_{\vdash}$, and to 0 otherwise.

Let us show now that, at $\langle e_{2M}, w_{2M}\rangle$, transition $\langle e_{2M}, \bigwedge_{a\in A} \text{oc}(a), \llcorner, \{\mathtt{c}\delta := 0\}, \ell_\delta\rangle$ can be taken. By constraint (iv) of Definition 2.6, we know that, for all $c = \langle t, a, d\rangle \in \pi$, it holds that $\forall \pi^{ind}_{\vdash}(c) \le k < \pi^{ind}_{\dashv}(c).\, \text{pre}^{\leftrightarrow}(a) \subseteq s_k$. Therefore, when $a \in \rho_i$ $\text{pre}^{\leftrightarrow}(a) \subseteq s_i$. This directly satisfies the guard formula oc($a$).

Finally, we add a timed transition $\langle \ell_\delta, v_i\rangle \xrightarrow{t_{i+1}-t_i} \langle \ell_\delta, v_j\rangle$ to the run $R[\pi]$ so that time advances and $v_j(\mathtt{c}\gamma) = t_{i+1}$. Now, as before, we can deterministically traverse the state-decoding portion and arrive at $\langle r_{M+N}, v_j\rangle$, with $s_{i+1} = s[v_j]$ and $\rho_{i+1} = \{a \mid v_i(\mathtt{cr_a}) = 1\}$, where, by the inductive hypothesis, we can extend the run $R[\pi]$ till $\ell^*$. □

## References


[1] Meysam Aghighi, Christer Bäckström, Cost-optimal and net-benefit planning - a parameterized complexity view, in: Proc. of the Twenty-Fourth International Joint Conference on Artificial Intelligence, AAAI Press, 2015, pp. 1487–1493.
[2] Rajeev Alur, David L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (2) (1994) 183–235.
[3] Rajeev Alur, Thomas A. Henzinger, Real-time logics: complexity and expressiveness, Inf. Comput. 104 (1) (1993) 35–77.
[4] Rajeev Alur, Thomas A. Henzinger, A really temporal logic, J. ACM 41 (1) (1994) 181–203.
[5] Rajeev Alur, Costas Courcoubetis, David Dill, Model-checking for real-time systems, in: Proc. of the Fifth Annual IEEE Symposium on Logic in Computer Science, IEEE, 1990, pp. 414–425.
[6] Christer Bäckström, Peter Jonsson, Time and space bounds for planning, J. Artif. Intell. Res. 60 (2017) 595–638, https://doi.org/10.1613/jair.5535.
[7] Christer Bäckström, Peter Jonsson, Sebastian Ordyniak, Stefan Szeider, A complete parameterized complexity analysis of bounded planning, J. Comput. Syst. Sci. 81 (7) (2015) 1311–1332, https://doi.org/10.1016/j.jcss.2015.04.002.
[8] J.C.M. Baeten, C.A. Middelburg, Process algebra with timing: real time and discrete time, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, ISBN 978-0-444-82830-9, 2001, pp. 627–684.
[9] Sergiy Bogomolov, Daniele Magazzeni, Stefano Minopoli, Martin Wehrle, PDDL+ planning with hybrid automata: foundations of translating must behavior, in: Proc. of the 25th International Conference on Automated Planning and Scheduling, 2015, pp. 42–46.
[10] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, Antoine Petit, Updatable timed automata, Theor. Comput. Sci. (ISSN 0304-3975) 321 (2) (2004) 291–345, https://doi.org/10.1016/j.tcs.2004.04.003.
[11] Laura Bozzelli, Angelo Montanari, Adriano Peron, Taming the complexity of timeline-based planning over dense temporal domains, in: Proc. of the 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, in: LIPIcs, vol. 150, 2019, pp. 34:1–34:14.
[12] Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, Undecidability of future timeline-based planning over dense temporal domains?, in: Gennaro Cordasco, Luisa Gargano, Adele A. Rescigno (Eds.), Proceedings of the 21st Italian Conference on Theoretical Computer Science, Ischia, Italy, September 14–16, 2020, in: CEUR Workshop Proceedings, vol. 2756, CEUR-WS.org, 2020, pp. 155–166, http://ceur-ws.org/Vol-2756/paper_15.pdf.
[13] Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, Gerhard J. Woeginger, Timeline-based planning over dense temporal domains, Theor. Comput. Sci. 813 (2020) 305–326, https://doi.org/10.1016/j.tcs.2019.12.030.
[14] Tom Bylander, The computational complexity of propositional STRIPS planning, Artif. Intell. 69 (1–2) (1994) 165–204.
[15] Christer Bäckström, Bernhard Nebel, Complexity results for SAS+ planning, Comput. Intell. 11 (4) (1995) 625–655, https://doi.org/10.1111/j.1467-8640.1995.tb00052.x.
[16] Alessandro Cimatti, Marco Pistore, Marco Roveri, Paolo Traverso, Weak, strong, and strong cyclic planning via symbolic model checking, Artif. Intell. 147 (1) (2003) 35–84.
[17] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, Stefano Tonetta, Extending nuXmv with timed transition systems and timed temporal properties, in: Proc. of the 31st International Conference on Computer Aided Verification, in: Lecture Notes in Computer Science, vol. 11561, Springer, 2019, pp. 376–386.
[18] Amanda J. Coles, Andrew Coles, Maria Fox, Derek Long, Forward-chaining partial-order planning, in: Proc. of the 20th International Conference on Automated Planning and Scheduling, AAAI, 2010, pp. 42–49.
[19] Amanda J. Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez Celorrio, Carlos Linares López, Scott Sanner, Sungwook Yoon, A survey of the seventh international planning competition, AI Mag. 33 (1) (2012).
[20] William A. Cushing, When is temporal planning really temporal?, PhD thesis, Arizona State University, 2012.
[21] William A. Cushing, Subbarao Kambhampati Mausam, Daniel S. Weld, When is temporal planning really temporal?, in: Proc. of the 20th International Joint Conference on Artificial Intelligence, 2007, pp. 1852–1859.
[22] Dario Della Monica, Nicola Gigante, Angelo Montanari, Pietro Sala, Guido Sciavicco, Bounded timed propositional temporal logic with past captures timeline-based planning with bounded constraints, in: Carles Sierra (Ed.), Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19–25, 2017, 2017.
[23] Dario Della Monica, Nicola Gigante, Salvatore La Torre, Angelo Montanari, Complexity of qualitative timeline-based planning, in: Proc. of the 27th International Symposium on Temporal Representation and Reasoning, in: LIPIcs, vol. 178, 2020, pp. 16:1–16:13.
[24] Minh Binh Do, Subbarao Kambhampati, Sapa: a multi-objective metric temporal planner, J. Artif. Intell. Res. 20 (2003) 155–194, https://doi.org/10.1613/jair.1156.
[25] Kutluhan Erol, Dana S. Nau, V.S. Subrahmanian, Complexity, decidability and undecidability results for domain-independent planning, Artif. Intell. 76 (1–2) (1995) 75–88, https://doi.org/10.1016/0004-3702(94)00080-K.
[26] Kutluhan Erol, James A. Hendler, Dana S. Nau, Complexity results for HTN planning, Ann. Math. Artif. Intell. 18 (1) (1996) 69–93.
[27] Patrick Eyerich, Robert Mattmüller, Gabriele Röger, Using the context-enhanced additive heuristic for temporal and numeric planning, in: Towards Service Robots for Everyday Environments, in: Springer Tracts in Advanced Robotics, vol. 76, Springer, 2012, pp. 49–64.
[28] M. Fox, D. Long, PDDL2.1: an extension to PDDL for expressing temporal planning domains, J. Artif. Intell. Res. 20 (2003) 61–124.
[29] Maria Fox, Derek Long, Modelling mixed discrete-continuous domains for planning, J. Artif. Intell. Res. 27 (2006) 235–297.
[30] Estíbaliz Fraca, Serge Haddad, Complexity analysis of continuous Petri nets, Fundam. Inform. 137 (1) (2015) 1–28.
[31] Alfonso Gerevini, Alessandro Saetti, Ivan Serina, Planning through stochastic local search and temporal action graphs in LPG, J. Artif. Intell. Res. 20 (2003) 239–290.

[32] Malik Ghallab, Dana S. Nau, Paolo Traverso, Automated Planning - Theory and Practice, Elsevier, 2004.
[33] Malik Ghallab, Dana S. Nau, Paolo Traverso, Automated Planning and Acting, Cambridge University Press, ISBN 978-1-107-03727-4, 2016.
[34] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, Andrea Orlandini, Timelines are expressive enough to capture action-based temporal planning, in: Proc. of the 23rd International Symposium on Temporal Representation and Reasoning, IEEE Computer Society, 2016, pp. 100–109.
[35] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, Andrea Orlandini, Complexity of timeline-based planning, in: Proc. of the 27th International Conference on Automated Planning and Scheduling, AAAI Press, 2017, pp. 116–124.
[36] Nicola Gigante, Andrea Micheli, Angelo Montanari, Enrico Scala, Decidability and complexity of action-based temporal planning over dense time, in: Proc. of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI Press, 2020, pp. 9859–9866.
[37] Bernd Heidergott, Geert Jan Olsder, Jacob Van der Woude, Max Plus at Work: Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications, Princeton University Press, 2014.
[38] Alexander Heinz, Martin Wehrle, Sergiy Bogomolov, Daniele Magazzeni, Marius Greitschus, Andreas Podelski, Temporal planning as refinement-based model checking, in: Proc. of the 29th International Conference on Automated Planning and Scheduling, AAAI Press, 2019, pp. 195–199.
[39] M. Hennessy, T. Regan, A process algebra for timed systems, Inf. Comput. (ISSN 0890-5401) 117 (2) (1995) 221–239, https://doi.org/10.1006/inco.1995.1041.
[40] Michael L. Littman, Probabilistic propositional planning: representations and complexity, in: Proc. of the 14th National Conference on Artificial Intelligence and the 9th Innovative Applications of Artificial Intelligence Conference, AAAI Press, 1997, pp. 748–754.
[41] Michael L. Littman, Judy Goldsmith, Martin Mundhenk, The computational complexity of probabilistic planning, J. Artif. Intell. Res. 9 (1998) 1–36, https://doi.org/10.1613/jair.505.
[42] Omid Madani, Steve Hanks, Anne Condon, On the undecidability of probabilistic planning and related stochastic optimization problems, Artif. Intell. 147 (1–2) (2003) 5–34, https://doi.org/10.1016/S0004-3702(02)00378-8.
[43] Dario Della Monica, Nicola Gigante, Angelo Montanari, Pietro Sala, A novel automata-theoretic approach to timeline-based planning, in: Michael Thielscher, Francesca Toni, Frank Wolter (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October, 2 November 2018, AAAI Press, 2018, pp. 541–550, https://aaai.org/ocs/index.php/KR/KR18/paper/view/18024.
[44] Martin Mundhenk, Judy Goldsmith, Christopher Lusena, Eric Allender, Complexity of finite-horizon Markov decision process problems, J. ACM 47 (4) (2000) 681–720, https://doi.org/10.1145/347476.347480.
[45] Joël Ouaknine, James Worrell, On the decidability of metric temporal logic, in: Proc. of the 20th Annual IEEE Symposium on Logic in Computer Science, IEEE, 2005, pp. 188–197.
[46] Masood Feyzbakhsh Rankooh, Gholamreza Ghassem-Sani, ITSAT: an efficient SAT-based temporal planner, J. Artif. Intell. Res. 53 (2015) 541–632, https://doi.org/10.1613/jair.4697.
[47] Jussi Rintanen, Complexity of planning with partial observability, in: Proc. of the 14th International Conference on Automated Planning and Scheduling, AAAI, 2004, pp. 345–354.
[48] Jussi Rintanen, Complexity of concurrent temporal planning, in: Proc. of the 17th International Conference on Automated Planning and Scheduling, 2007, pp. 280–287.
[49] Jussi Rintanen, Models of action concurrency in temporal planning, in: Qiang Yang, Michael J. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015, AAAI Press, 2015, pp. 1659–1665, http://ijcai.org/Abstract/15/237.
[50] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, Miquel Ramírez, Interval-based relaxation for general numeric planning, in: Proc. of the 22nd European Conference on Artificial Intelligence, in: Frontiers in Artificial Intelligence and Applications, vol. 285, IOS Press, 2016, pp. 655–663.
[51] Ji-Ae Shin, Ernest Davis, Processes and continuous change in a sat-based planner, Artif. Intell. 166 (1–2) (2005) 194–253.
[52] Mauro Vallati, Lukas Chrpa, Marek Grześ, Thomas Leo McCluskey, Mark Roberts, Scott Sanner, et al., The 2014 international planning competition: progress and trends, AI Mag. 36 (3) (2015) 90–98.
[53] Peter van Emde Boas, The convenience of tiling, in: Andrea Sorbi (Ed.), Complexity, Logic and Recursion Theory, in: Lecture Notes in Pure and Applied Mathematics, vol. 187, Marcel Dekker Inc., 1997, pp. 331–363.
[54] David Wang, Brian Charles Williams, tBurton: a divide and conquer temporal planner, in: AAAI, AAAI Press, 2015, pp. 3409–3417.
[55] Jiacun Wang, Timed Petri Nets: Theory and Application, vol. 9, Springer Science & Business Media, 2012.