Proceedings of the Sixteenth International Conference on
Principles of Knowledge Representation and Reasoning (KR 2018)

A Novel Automata-Theoretic Approach to Timeline-Based Planning

Dario Della Monica,[*†‡] Nicola Gigante,[§] Angelo Montanari,[§] Pietro Sala[¶]
[*] Istituto Nazionale di Alta Matematica "F. Severi" (INdAM), Italy
[†] Università di Napoli, Italy
[‡] Universidad Complutense de Madrid, Spain
[§] Università di Udine, Italy
[¶] Università di Verona, Italy

Abstract

Timeline-based planning is a well-established approach successfully employed in a number of application domains. A very restricted fragment, featuring only bounded temporal relations and token durations, is expressive enough to capture action-based temporal planning. As for computational complexity, it has been shown to be EXPSPACE-complete when unbounded temporal relations, but only bounded token durations, are allowed. In this paper, we present a novel automata-theoretic characterisation of timeline-based planning where the existence of a plan is shown to be equivalent to the nonemptiness of the language recognised by a nondeterministic finite-state automaton that suitably encodes all the problem constraints (timelines and synchronisation rules). Besides allowing us to restate known complexity results in a fairly natural and compact way, such an alternative characterisation makes it possible to finally establish the exact complexity of the full version of the problem with unbounded temporal relations and token durations, which was still open and turns out to be EXPSPACE-complete. Moreover, the proposed technique is general enough to cope with (infinite) recurrent goals, which received little attention so far, despite being quite common in real-word application scenarios.

Introduction

In this paper, we illustrate a novel automata-theoretic approach to timeline-based planning, which turns out to be beneficial in a twofold perspective. On the one hand, known complexity results can be restated in a clean and compact way by making use of nondeterministic, finite state automata (NFAs); on the other hand, it allows us to fill the last complexity gap. Moreover, an easy generalisation of the proposed solution, that replaces NFAs by Büchi automata, makes it possible to cope with recurring goals when the problem is interpreted over infinite timelines.

In the timeline-based framework, the planning domain is modelled as a set of independent, but interacting, components. Each component is represented by a finite number of *state variables*, whose behaviour over time (the *timelines*) is governed by a set of temporal constraints (*transition functions* and *synchronisation rules*).

The timeline-based approach to planning was initially introduced in the context of planning and scheduling of *space operations* (Muscettola 1994). Since then, it has been successfully employed in a number of complex real-world scenarios, including both long-term mission planning and onboard spacecraft autonomy (Jónsson et al. 2000; Frank and Jónsson 2003; Cesta et al. 2007; Chien et al. 2010; Barreiro et al. 2012). Recently, timeline-based systems have also been exploited in cooperative robotics applications (Umbrico et al. 2017).

Compared to the well-known action-based planning paradigm, the timeline-based approach allows for a more declarative description of the problem domain, which does not focus on what an agent has to *do*, but rather on what has to *happen*. In particular, there is no explicit separation between *states* and *actions*, but, rather, the timelines represent both the current state of the system and the ongoing tasks being performed. This point of view makes the approach particularly well suited to model the interaction of many components rather than the behaviour of a single agent.

Despite the successful application of timeline-based planning systems, an in-depth theoretical understanding of the paradigm was missing until very recently (Gigante et al. 2016; 2017).

As a first step, Gigante et al. (2016) showed that timeline-based planning with *bounded* temporal relations and token durations, and no temporal horizon, is EXPSPACE-complete and *expressive* enough to capture action-based temporal planning, that is, PDDL 2 with durative actions (Fox and Long 2003). To show this result, they first provide a polynomial-time reduction of action-based temporal planning, which is known to be EXPSPACE-complete (Rintanen 2007), to timeline-based planning, and then develop an EXPSPACE procedure for the latter (a nondeterministic Turing machine using only an exponential amount of space).

Later, in (Gigante et al. 2017), it has been proved

N. Gigante and A. Montanari worked on these results mainly while on leave at The University of Western Australia. Nicola Gigante was supported by the AIxIA Outgoing mobility grant 2017. This work was also partially supported by the Italian INdAM GNCS project Formal methods for verification and synthesis of discrete and hybrid systems (N. Gigante, A. Montanari, and P. Sala), a Marie Curie INdAM-COFUND-2012 Outgoing Fellowship (D. Della Monica), and the PRID project ENCASE - Efforts in the uNderstanding of Complex interActing SystEms (N. Gigante and A. Montanari).

that EXPSPACE-completeness still holds for timeline-based planning with unbounded interval relations. The core of the proof of EXPSPACE-completeness is a small model theorem showing that any satisfiable timeline-based planning problem has a solution at most doubly exponentially long. Such a result is then exploited to build a nondeterministic guess-and-check procedure, that runs in nondeterministic exponential space. The very same procedure gives a NEXPTIME bound when we restrict ourselves to exponentially sized solutions, that is, the problem is proved to be NEXPTIME-complete when a bound on the solution horizon is required as part of the input. As a matter of fact, there seems to be no easy way to accommodate such a small model argument to unbounded token durations.

Here, we prove that timeline-based planning with both unbounded temporal relations and unbounded token durations is EXPSPACE-complete by using a completely different automata-theoretic construction. More precisely, we show that any instance of the timeline-based planning problem can be encoded into a suitable nondeterministic finite-state automaton (NFA) $\mathcal{A}$ such that the language recognised by $\mathcal{A}$, say, $\mathcal{L}(\mathcal{A})$, is nonempty if and only if a solution plan exists.

The proof consists of two main steps: we first build the NFA $\mathcal{A}$ and show that it recognises exactly those finite words that represent solution plans for the given problem; then, we check the nonemptiness of $\mathcal{L}(\mathcal{A})$ by solving a suitable reachability problem over $\mathcal{A}$, as usual.

In order to respect the EXPSPACE complexity bound, we must control both the structural complexity of $\mathcal{A}$ and the complexity of the reachability algorithm. To this end, we merge the construction of the automaton and the reachability check into a unique on-the-fly procedure. This technique allows us to finally characterise the computational complexity of the full problem without introducing any artificial restriction on the syntax of transition functions and synchronisation rules.

Compared to the combinatorial argument used in (Gigante et al. 2017), the automata-theoretic proof given here is definitely simpler and cleaner. Moreover, it can be easily extended to the case of infinite timelines, where synchronisation rules can be exploited to express recurrent goals.

Despite the existence of a number of natural application scenarios, to the best of our knowledge, the case of infinite plans has not been investigated in the context of timeline-based planning. Here, we show that a natural extension of the proposed approach, that replaces NFAs by Büchi automata, actually solves the problem with no increase in computational complexity.

The paper is organised as follows. The next section provides some background knowledge on timeline-based planning. The third one describes the encoding of an instance of the timeline-based planning problem into an NFA, and then it shows how to exploit the resulting automaton to solve it. The fourth section generalises the proposed solution to the case of infinite timelines. Conclusions provide a short assessment of the work done and outline future research directions. Proof details are omitted because of lack of space.

Timeline-based planning

This section introduces basic concepts and terminology of timeline-based planning. The notation is mostly borrowed from (Gigante et al. 2016; Della Monica et al. 2017), and was originally introduced and extensively discussed in (Cialdea Mayer, Orlandini, and Umbrico 2016).

Definition 1 (State variable) *A* state variable *x is a triple (V_x, T_x, D_x), where:*

- *V_x is the* finite domain *of the variable x;*
- *$T_x : V_x \to 2^{V_x}$ is the* value transition function*, which maps each value $v \in V_x$ to the set of values that x can take immediately after v;*
- *$D_x : V_x \to \mathbb{N} \times (\mathbb{N} \cup \{+\infty\})$ is a function that maps each $v \in V_x$ to a pair of values $0 < d_{min}^{x=v} \leq d_{max}^{x=v}$, which represent respectively the minimum and maximum duration of an interval over which x takes value v.*

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

Definition 2 (Token) *A* token *for x is a tuple $\tau = (x, v, d)$, where x is a state variable, $v \in V_x$, and $d \in \mathbb{N}$ is the* duration *of the token, with $d_{min}^{x=v} \leq d \leq d_{max}^{x=v}$.*

It is worth pointing out that, according to Definition 1, the duration of a token can be arbitrarily long (this is the case when $+\infty$ is taken as the maximum duration of the token). The proof of EXPSPACE-completeness given in (Gigante et al. 2017) excludes such a possibility: it assumes the duration of tokens to be bounded, and there is no way to generalise it in order to admit arbitrarily long tokens. The proof given in this paper naturally handles this case, without any special treatment.

The time-varying behaviour of a state variable is modelled by means of a finite sequence of tokens, called a *timeline*.

Definition 3 (Timeline) *A* timeline *for a state variable $x = (V_x, T_x, D_x)$ is a finite sequence $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ of tokens for x, where $v_{i+1} \in T_x(v_i)$ for $i \in \{1, \ldots, k-1\}$.*

Notice that the values of x in two consecutive tokens do not need to be different.

A time interval can be associated with any token $\tau_i = (x, v_i, d_i)$ in a timeline $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ by means of the functions $\mathrm{start\_time}(\mathsf{T}, i) = \sum_{j=1}^{i-1} d_j$ and $\mathrm{end\_time}(\mathsf{T}, i) = \mathrm{start\_time}(\mathsf{T}, i) + d_i$. The end time of the last token of a timeline is called the *horizon* of the timeline. For the sake of simplicity, when the timeline T is clear from context, we will write $\mathrm{start\_time}(\tau_i)$ and $\mathrm{end\_time}(\tau_i)$ for $\mathrm{start\_time}(\mathsf{T}, i)$ and $\mathrm{end\_time}(\mathsf{T}, i)$, respectively.

In the following, when there is no ambiguity, we will interchangeably refer to a token and to the associated time interval. Moreover, to refer to a token $\tau = (x, v, d)$, we will often write $\tau = (v, d)$ when the intended state variable is understood.

The behaviour of state variables is constrained by a set of *synchronisation rules*, which relate tokens, possibly belonging to different timelines, through temporal relations among intervals or among intervals and time points.

To express these temporal constraints, we adopt the compact notation proposed in (Gigante et al. 2016). Let $\Sigma = \{a, b, c, \dots\}$ be a set of *token names* used to refer to tokens.

Definition 4 (Atom) *An* atom *is either a clause of the form* $a \leq_I^{*_1,*_2} b$ *(interval), or of the form* $a \leq_I^{*_1} t$, $a \geq_I^{*_1} t$ *(time-point), where* $a, b \in \Sigma$, $t \in \mathbb{N}$, I *is either* $[l, u]$ *or* $[l, +\infty)$, *with* $l, u \in \mathbb{N}$, *and* $*_1, *_2$ *are either* start_time *or* end_time, *respectively* s *and* e *for short.*

As an example, the atom $a \leq_{[l,u]}^{\mathsf{s},\mathsf{e}} b$ constrains the token a to *start* before the *end* of b, and the distance between the two related endpoints to be between l and u.

Definition 5 (Synchronization rule) *Let* SV *be a set of state variables.*

An existential statement *is an expression of the form:*

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

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

A synchronisation rule $\mathcal{R}$ *is a clause of one of the following two forms:*

$$a_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \dots \vee \mathcal{E}_k$$
$$\top \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \dots \vee \mathcal{E}_k$$

where $a_0 \in \Sigma$, $x_0 \in SV$, $v_0 \in V_{x_0}$, *and* $\mathcal{E}_1, \dots, \mathcal{E}_k$ *are* existential statements *where only* a_0 *may appear free. In rules of the first form, the quantifier* $a_0[x_0 = v_0]$ *is called* trigger. *Rules of the second form are said to be* trigger-less.

Definition 5 can be read as follows. The trigger is a universal quantifier, which states that *for all* the tokens a_0, where the variable x_0 takes the value v_0, at least one of the existential statements $\mathcal{E}_i$ must be true. The existential statements in turn assert the existence of tokens $a_1, \dots, a_n$, where the respective state variables take the specified values, that satisfy the temporal constraints given by C. Trigger-less rules simply assert the satisfaction of the existential statements.

Given an existential statement $\mathcal{E} \equiv \exists a_1[x_1 = v_1] \dots a_n[x_n = v_n] \,.\, C$, its *existential prefix* is the sequence of quantifiers $\exists a_1[x_1 = v_1] \dots a_n[x_n = v_n]$. The *length* of the existential prefix of $\mathcal{E}$ is defined as the number of quantifiers occurring in it.

The semantics of atoms and synchronisation rules is defined as follows.

Definition 6 (Semantics of atoms) *Let* Γ *be a set of tokens and let* $\lambda : \Sigma \rightarrow \Gamma$ *be a function that assigns a token to each token name.*

An interval atom $a \leq_{l,u}^{*_1,*_2} b$ *is* satisfied *by* λ *if* $l \leq *_2(\lambda(b)) - *_1(\lambda(a)) \leq u$.

A time-point atom of the form $a \leq_{[l,u]}^{*} t$ *(resp.,* $a \geq_{[l,u]}^{*} t$*) is* satisfied *by* λ *if* $l \leq t - *(\lambda(a)) \leq u$ *(resp.,* $l \leq *(\lambda(a)) - t \leq u$*).*

Definition 7 (Semantics of synchronization rules) *Let* Γ *be a set of tokens and let* $\lambda : \Sigma \rightarrow \Gamma$ *be a function that assigns a token to each token name.*

A quantifier $a[x = v]$ *is* satisfied *by* λ *if* $\lambda(a) = (x_a, v_a, d_a)$, *with* $x_a = x$ *and* $v_a = v$.

An existential statement $\mathcal{E}$, *with conjunct clause* C, *is satisfied by* λ *if all the quantifiers of* $\mathcal{E}$ *and all the atoms in* C *are satisfied by* λ.

A synchronisation rule $\mathcal{R}$ *of the form* $a_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \dots \vee \mathcal{E}_k$ *is* satisfied *by* Γ *if, for every token* $\tau = (x_\tau, v_\tau, d_\tau) \in \Gamma$, *with* $x_\tau = x_0$ *and* $v_\tau = v_0$, *there is an existential statement* $\mathcal{E}_i$ *and a mapping* $\lambda : \Sigma \rightarrow \Gamma$ *such that* $\lambda(a_0) = \tau$ *and* λ *satisfies* $\mathcal{E}_i$.

A timeline-based planning domain is specified by a set of state variables and a set of synchronisation rules representing their admissible behaviours. Trigger-less rules can be used to express initial conditions, domain invariants, and goals.

Definition 8 (Timeline-based planning problem) *A timeline-based planning* problem *is a pair* $P = (SV, S)$, *where* SV *is a set of state variables and* S *is a set of synchronisation rules over* SV.

Definition 9 (Solution plan) *Let* $SV = \{x_1, \dots, x_n\}$ *be a set of state variables.*

A plan *over* SV *is a set of timelines* $\pi = \{\mathsf{T}_1, \dots, \mathsf{T}_n\}$, *one for each* $x_i \in SV$.

A solution plan *(or, simply, a* solution*) for a timeline-based planning problem* $P = (SV, S)$ *is a plan* π *over* SV *such that all the synchronisation rules in* S *are satisfied by the set* Γ *of all the tokens involved in* π, *that is,* $\Gamma = \{\tau \mid \tau \in \mathsf{T}, \mathsf{T} \in \pi\}$.

A timeline-based planning problem is *satisfiable* if a solution for it exists.

We conclude the section with a short account of known complexity results. In (Gigante et al. 2017), it has been shown that establishing whether a solution plan exists for a given timeline-based planning problem is EXPSPACE-complete, but only if tokens with unbounded duration are forbidden, that is, only if $d_{max}^{x=v} \neq +\infty$ for all $x \in SV$ and $v \in V_x$.

The EXPSPACE decision procedure is based on the following small-model property.

Proposition 1 (Gigante et al. 2017) *Let* P *be a timeline-based planning problem where unbounded durations are forbidden. If* P *admits any solution plan, then there exists at least one solution plan of length at most doubly exponential in the size of* P.

Such a small-model property allows one to devise a nondeterministic *guess-and-check* algorithm that nondeterministically guesses a doubly exponentially long candidate solution plan for the problem and then checks if it satisfies all the constraints of the problem in exponential space. Doing the check in exponential space is possible by sliding an exponentially long window over the solution that is being guessed, without storing the entire candidate plan at once.

The above result is proven by a contraction procedure, where a plan supposedly longer than the bound is cut in suitable parts to obtain a shorter one, which is justified by a rather involved combinatorial argument that cannot be applied to the case of unbounded token durations. The complexity of the full problem has therefore remained open until now.

EXPSPACE-completeness

In this section, we first define the encoding of a timeline-based planning problem, with both unbounded temporal relations and unbounded tokens, into a nondeterministic finite state automaton (NFA), and then we provide an EXPSPACE decision procedure to check the resulting NFA for nonemptiness.

Throughout the paper, let $P = (SV, S)$ be a specific instance of the planning problem, where we let $x = (V_x, T_x, D_x)$ and $(d_{min}^{x=v}, d_{max}^{x=v}) = D_x(v)$ for all $x \in SV$ and $v \in V_x$; moreover, let $\mathcal{V} = \bigcup_{x \in SV} V_x$.

The alphabet Σ of the automaton is defined as the set of functions assigning a legitimate value (or the special symbol $\circlearrowleft$) to each variable in SV. Formally, $\Sigma = \{\sigma : SV \to \mathcal{V} \cup \{\circlearrowleft\} \mid \sigma(x) \in V_x \cup \{\circlearrowleft\} \text{ for all } x \in SV\}$. Notice that we can equivalently define Σ as the cartesian product $\times_{x \in SV}(V_x \cup \{\circlearrowleft\})$. The elements of Σ are called *token symbols*. Among them, we identify the *starting token symbols* and collect them in the set $\Sigma_S = \{\sigma \in \Sigma \mid \sigma(x) \neq \circlearrowleft \text{ for all } x \in SV\}$. Observe that $|\Sigma| \leq (|\mathcal{V}|+1)^{|SV|}$, that is, the number of token symbols is at most exponential in the size of P.

The following correspondence can be naturally established between finite words in $\Sigma_S\Sigma^*$ and finite plans. First of all, we associate a natural number (an index) with each position in a word, that is, the first element of a word w is indexed by 0 and denoted by $w[0]$, the second element is indexed by 1 and denoted by $w[1]$, and so on. Given a word $w \in \Sigma_S\Sigma^* \cup \{\varepsilon\}$ and a state variable x, we define the *timeline for x associated with w*, denoted by $\mathbf{T}_x^w$, as the timeline $\langle (v_{i_0}, i_1 - i_0), (v_{i_1}, i_2 - i_1), \ldots, (v_{i_{k-1}}, i_k - i_{k-1})\rangle$, where $\{i_0, i_1, \ldots, i_{k-1}\} = \{i \mid w[i](x) \neq \circlearrowleft\}$, $i_k = |w|$, and $i_{j-1} < i_j$ for all $j \in \{1, \ldots, k\}$. We define the *plan associated with w*, denoted by $\mathbf{T}^w$, as the set of timelines $\{\mathbf{T}_x^w \mid x \in SV\}$.

We say that a word in Σ^* is a *solution* for P if the plan associated with it is a solution for P. Hence, it holds that P admits a solution plan if and only if the language $\{w \in \Sigma^* \mid w \text{ is a solution for } P\}$ is not empty.

Automaton construction

We now show how to build, for any given planning problem P, an NFA over Σ that accepts exactly those words that are solutions for P. Notice that the ith element of a word $w \in \Sigma^*$ can be viewed as a snapshot of the values of the variables at the ith time-point of the plan associated with w.

An informal account. Intuitively speaking, the states of the automaton contain patterns of tokens (called *blueprints*, see below for their formal definition) with some pieces of information about their positioning along the plan. While blueprints specify the qualitative information about the relative position of such tokens (e.g., token a occurs before token b), they abstract away, to some extent, quantitative information about how far apart tokens are, by making use of a *pumping machinery*. This will allow us to deal with unbounded temporal relations and durations.

As an example, consider blueprint B_1 in Figure 1. It is an abstraction representing all scenarios where the three tokens a_0 (for x_0, with value v_0), a_1 (for x_1, with value v_3), and a_2 (for x_0, with value v_1) occur in the following relative positions: a_1 occurs entirely before than a_0, which, in turn, occurs entirely before than a_2. As already mentioned, information about how long and how far apart the intervals are is abstracted away. Technically, this is done thanks to the presence of *pumping points* (filled circles in the picture), which allow for stretching the distances through the addition of points. Roughly speaking, a pumping point can be replaced by a series of points.

Such a blueprint can be instantiated in two different ways in the plan depicted in Figure 1:

- by instantiating tokens a_0, a_1, a_2 in B_1 with tokens $[6, 8]$, $[0, 4]$, and $[14, 18]$ in the plan, respectively;
- by instantiating tokens a_0, a_1, a_2 in B_1 with tokens $[8, 12]$, $[0, 4]$, and $[14, 18]$ in the plan, respectively.

Contrarily, blueprint B_1 cannot be instantiated by associating a_0 with $[1, 6]$ in the plan because, even though state variable and value of a_0 match the ones of $[1, 6]$, there is no token for x_1 ending before point 1 in the plan, meaning that the relative positioning of tokens imposed by B_1 cannot be fulfilled.

Each blueprint is functional to the satisfaction of a synchronisation rule. Pushing forward with our running example, consider the following synchronisation rule:

$$a_0[x_0 = v_0] \to \exists a_1[x_1 = v_3]a_2[x_0 = v_1]. \\ (a_1 \leq^{s,s}_{[1,+\infty)} a_0 \wedge a_0 \leq^{e,s}_{[1,+\infty)} a_2). \qquad (\mathcal{R})$$

Its meaning is the following: for every token a_0 for x_0 with value v_0, there must exist a token a_1 for x_1 with value v_3 and a token a_2 for x_0 with value v_1 such that the starting point of token a_1 occurs strictly before the starting point of a_0 and the ending point of a_0 occurs strictly before the starting point of a_2. Such a rule is triggered by tokens $[1, 6]$, $[6, 8]$, and $[8, 12]$ in the plan in Figure 1. Blueprint B_1 can be used to certify the fulfilment of such a rule when triggered by $[6, 8]$ (resp., $[8, 12]$), because B_1 satisfies $(\mathcal{R})$, that is, it conforms with the constraints $(\mathcal{R})$ imposes, and there is an instantiation of B_1 which associates a_0 with $[6, 8]$ (resp., $[8, 12]$).

The automaton scans the plan (associated with the input word w) from left to right, nondeterministically instantiating (abstract) tokens in blueprints with (concrete) ones as they are discovered along the plan, while obeying to (qualitative and quantitative) positioning constraints. A word is accepted if every token that triggers a synchronisation rule is involved in some instantiation of a blueprint that satisfies the rule.

A formal definition. The intuitive account given above can be turned into a formal definition as follows.

Blueprints. Let us fix the following two quantities:

- N is the largest constant appearing in P for temporal relation and duration bounds;
- M is the length of the largest existential prefix of an existential statement occurring inside a synchronisation rule in P (e.g., if P only features rule $(\mathcal{R})$, then $M = 2$).

Notice that N may be exponential in the size of P if constants are expressed in binary while M is linear in the size of P. We let $K = 2 \cdot (N + 1) + 2 \cdot (N + 1) \cdot (M + 1)$ and, for $i, j \in \mathbb{N}$, we let $[i, j] = \{i, i + 1, \ldots, j - 1, j\}$ and $[i] = [0, i]$.

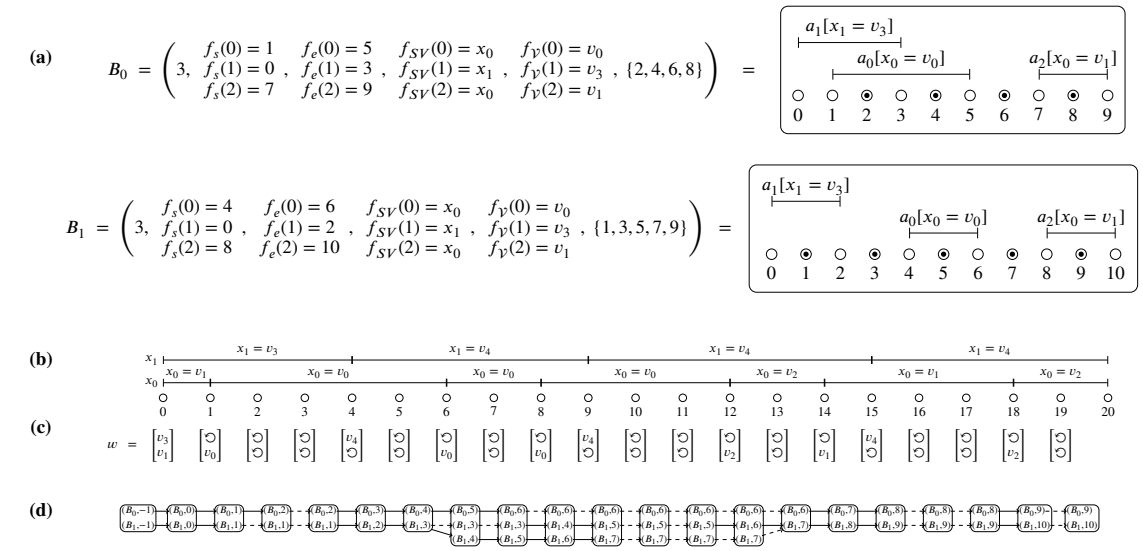

Figure 1: A run of the automaton **(d)** over the plan **(b)** associated with word w **(c)**; states contain blueprints B_0 and B_1 **(a)**.

Definition 10 (Blueprint) *A* blueprint B *is a tuple* $B = (m, f_s, f_e, f_{SV}, f_{\mathcal{V}}, \mathcal{P})$*, with* $m \in [M]$*,* $f_s, f_e : [m] \to [K]$*,* $f_{SV} : [m] \to SV$*,* $f_{\mathcal{V}} : [m] \to \mathcal{V}$*, and* $\mathcal{P} \subseteq \{x - 1 \in \mathbb{N} \setminus (Img(f_s) \cup Img(f_e)) \mid x \in [2(N+1), K] \cap (Img(f_s) \cup Img(f_e))\}$*, such that for all* $i \in [m]$*:*

1. $f_s(i) < f_e(i)$*;*
2. *if* $f_{SV}(i) = (V_x, T_x, D_x)$*, then* $f_{\mathcal{V}}(i) \in V_x$*;*
3. (disjointness) *for each* $j \in [m]$*, if* $f_{SV}(i) = f_{SV}(j)$*, then exactly one of the following holds:*
   - $f_e(i) \leq f_s(j)$*,*
   - $f_s(i) \geq f_e(j)$*,*
   - $[f_s(i), f_e(i)] = [f_s(j), f_e(j)]$ *and* $f_{\mathcal{V}}(i) = f_{\mathcal{V}}(j)$*.*

We refer to $\mathcal{P}$ *as the set of* pumping points *of* B*.*

Two blueprints, B_0 and B_1, are shown in Figure 1 (top).

Let $\mathbb{B}$ be the set of blueprints. By Definition 10, it follows that $|\mathbb{B}| \in \mathcal{O}(M \cdot K^M \cdot K^M \cdot |SV|^M \cdot |\mathcal{V}|^M \cdot 2^{2M})$, i.e., the number of blueprints is at most exponential in the size of P.

Given a synchronisation rule $R = a_0[x_0 = v_0] \to \mathcal{E}_1 \vee \ldots \vee \mathcal{E}_k$ and a blueprint $B = (m, f_s, f_e, f_{SV}, f_{\mathcal{V}}, \mathcal{P})$, we say that B *fulfils* R, written $B \vDash R$, if and only if there exists $i \in [1, k]$, with $\mathcal{E}_i = \exists a_1[x_1 = v_1] \ldots a_m[x_m = v_m](A_1 \wedge \ldots \wedge A_h)$, such that for each $j \in [m]$, $f_{SV}(j) = x_j$ and $f_{\mathcal{V}}(j) = v_j$, and for each $j \in [1, h]$, the following conditions hold:

- if $A_j = a_{k'} \leq^{*_1,*_2}_{[l,u]} a_{k''}$ with $k' \in [m]$ and $k'' \in [m]$, then $l \leq f_{*_2}(k'') - f_{*_1}(k') \leq u$ and $[f_{*_1}(k'), f_{*_2}(k'')] \cap \mathcal{P} = \emptyset$;
- if $A_j = a_{k'} \leq^{*_1,*_2}_{[l,+\infty)} a_{k''}$ with $k' \in [m]$ and $k'' \in [m]$, then $l \leq f_{*_2}(k'') - f_{*_1}(k')$;
- if $A_j = a_{k'} \leq^{*}_{[l,u]} t$ with $k' \in [m]$, then $l \leq t - f_*(k') \leq u$;
- if $A_j = a_{k'} \geq^{*}_{[l,u]} t$ with $k' \in [m]$, then $l \leq f_*(k') - t \leq u$;
- if $A_j = a_{k'} \leq^{*}_{[l,+\infty)} t$ with $k' \in [m]$, then $l \leq t - f_*(k')$;
- if $A_j = a_{k'} \geq^{*}_{[l,+\infty)} t$ with $k' \in [m]$, then $l \leq f_*(k') - t$.

We define $B \vDash \mathcal{R}$ in a similar way when $\mathcal{R}$ is trigger-less.

Viewpoints. A *viewpoint* is a pair (B, k), where B is a blueprint and $-1 \leq k \leq K$. States of the automaton are sets of viewpoints (examples of automaton states are given in Figure 1, bottom).

Intuitively, a viewpoint is a blueprint equipped with a *pointer* (the second element of the pair) that maps (abstract) points/tokens in a blueprint to (concrete) ones in the plan (word) that is being processed by the automaton.

Such a connection is as follows: suppose that, after reading the token symbol $w[i]$ in the input word w, the automaton reaches a state that contains the viewpoint (B, k); then, point k of B is matched with point i of the input plan/word.

The set of viewpoints is denoted by $\mathbb{V}$. Its size is at most exponential in the size of P, that is, $|\mathbb{V}| \in \mathcal{O}(K \cdot |\mathbb{B}|)$. We say that a viewpoint $((m, f_s, f_e, f_{SV}, f_{\mathcal{V}}, \mathcal{P}), k)$ is *closed* if $k \geq \max(\mathrm{Img}(f_e))$; it is an *initial* viewpoint if $k = -1$.

Pumping and stepping. Since blueprints are compact representation of (parts of) a plan, a point in a blueprint may correspond to several points in the plan. In particular, this is the case with pumping points, that is, if k is a pumping point of B, then the automaton computation can (nondeterministically) move to a state where viewpoint (B, k) is left unchanged, while the input head move to from $w[i]$ to $w[i+1]$, meaning that the kth point of the blueprint is associated with both points i and $i+1$ of the plan: it abstracts several points of the plan.

Being in a pumping point k is a necessary condition for a viewpoint to be allowed to *pump* the blueprint (at k), that is, to keep its pointer still while the input head moves to the next token symbol and the computation moves to the next step; however, it is not a sufficient one. To precisely state when a viewpoint can pump the blueprint, we introduce the notions of *active* variables (and their values) in a viewpoint, as well as the one of *pumping symbol* for a viewpoint.

Let $value : \mathbb{V} \times SV \to \mathcal{V}$ be a partial function that returns the *value* of a state variable x in a viewpoint V, defined as:

$$value(V,x)=\begin{cases} v & \exists i\begin{pmatrix} f_s(i)\le k<f_e(i)\wedge \\ \wedge f_{SV}(i)=x\wedge f_{\mathcal{V}}(i)=v\end{pmatrix} \\ \text{undefined} & \text{otherwise}\end{cases}$$

Thanks to the disjointness of the tokens for x in the blueprint (condition 3 in Definition 10), *value* is well defined.[1] A variable $x \in SV$ is *active* in V if $value(V,x)$ is defined.

Definition 11 (Pumping symbol) *A token symbol $\sigma \in \Sigma$ is a* pumping symbol *for a viewpoint $V=(B,k)$, with $\mathcal{P}$ being the set of pumping points of B, if $k \in \mathcal{P}\cup\{-1\}$ and $\sigma(x)=\circlearrowleft$ holds for every x that is active in V.*

The above definition captures the fact that a viewpoint V is allowed to pump when the next symbol read by the automaton is a pumping symbol for V. The second part of the condition ($\sigma(x)=\circlearrowleft$ for all active variables x) reflects the fact that a point in a blueprint cannot hide a point that correspond in the input plan to the end of the active token: this would cause the active token to span two or more tokens in the plan, thus violating the (qualitative) structure of the plan itself.

Besides pumping, a viewpoint can evolve during the computation by *synchronising* with the plan, that is, the pointer of the viewpoint moves along the blueprint as the input head moves along the plan. In this case, we say that the viewpoint *steps* upon reading a given token symbol. This is captured by means of the following notions. A viewpoint $V=((m,f_s,f_e,f_{SV},f_{\mathcal{V}},\mathcal{P}),k)$ is said to *start* (resp., *end*) a state variable x if there exists i such $f_{SV}(i)=x$ and $f_s(i)=k$ (resp., $f_e(i)=k$).

Definition 12 (Step symbol) *Let $V=(B,k)$ be a viewpoint. A token symbol $\sigma \in \Sigma$ is a* step symbol *for V if the following conditions hold:*

1. *for all $x \in SV$, if $(B,k+1)$ starts x, then it holds that $value((B,k+1),x)=\sigma(x)$, and*
2. *for all $x \in SV$ with x being active in V, $(B,k+1)$ ends x if and only if $\sigma(x)\neq\circlearrowleft$.*

The evolution of viewpoints along computations can be described by means of a ternary relation $\rightarrow\in \mathbb{V}\times\Sigma\times\mathbb{V}$, defined as follows: $(V,\sigma,V')\in\rightarrow$, with $V=(B,k)$, if and only if one of the following holds:

- σ is a pumping symbol for V and $V'=V$,
- σ is a step symbol for V and $V'=(B,k+1)$,
- V is closed and $V'=V$.

In the following, we use the more intuitive notation $V\xrightarrow{\sigma}V'$ for $(V,\sigma,V')\in\rightarrow$. It is worth noticing that the notions of pumping and step symbols are not mutually exclusive, meaning that a symbol can be both a pumping and a step symbol for a given viewpoint. Consequently, a viewpoint $V=(B,k)$ can pump and step at the same time, thus leading to a state containing the two viewpoints $V_1=V$ (obtained through pumping) and $V_2=(B,k+1)$ (obtained through stepping). Roughly speaking, in such a case the viewpoint splits into two *copies*, that is, two viewpoints with the same blueprint but different pointers. Such a splitting corresponds to different instantiations of the same blueprint in order to satisfies a synchronisation rule that is triggered by different tokens in the plan.

Figure 1**(d)** depicts how viewpoints based on blueprints B_0 and B_1 (Figure 1**(a)**) evolve with respect to word w (Figure 1**(c)**)—stepping is represented through solid arrows, pumping is represented through dashed ones. Let us focus on blueprint B_1. As already pointed out, it can be used to certify the fulfilment of synchronisation rule ($\mathcal{R}$), when triggered by [6, 8] and [8, 12], through different instantiations: a_0 is associated with [6, 8] in one of them, while it is associated with [8, 12] in the other. This is done by exploiting the above described splitting mechanism. At the beginning, after reading the first token symbol ($w[0]=\{x_0\mapsto v_1, x_1\mapsto v_3\}$), the viewpoint $V_0=(B_1,0)$ is created, thus establishing the correspondence between point 0 in B_1 and point 0 in the plan. It is easy to check that $w[1]=\{x_0\mapsto v_0, x_1\mapsto \circlearrowleft\}$ is a step symbol (but it is not a pumping one) for V_0, thus the computation leads to viewpoint $V_1=(B_1,1)$, and so on. Before reading token symbol $w[6]=\{x_0\mapsto v_0, x_1\mapsto \circlearrowleft\}$, the state contains viewpoint $V_3=(B_1,3)$; since $w[6]$ is both a pumping and a step symbol for V_3, viewpoint V_3 splits into $V_3'=V_3$ and $V_4=(B_1,4)$. Viewpoint V_4 (created through stepping) establishes a correspondence between point 4 in B_1 and point 6 in the plan, thus commencing the instantiation of token [4, 6] in B_1 with [6, 8] in the plan (which is a trigger for ($\mathcal{R}$)). On the other hand, viewpoint V_3' (created through pumping) will be used to fulfil ($\mathcal{R}$) when it will be triggered by token [8, 12] in the plan. To this end, V_3' will pump point 3 until token symbol $w[8]=\{x_0\mapsto v_0, x_1\mapsto \circlearrowleft\}$ is reached; at that point, it steps to viewpoint $V_4'=(B_1,4)$, thus paving the way to relating token [4, 6] in B_1 with [8, 12] in the plan (which is another trigger for ($\mathcal{R}$)).

We are now ready to define the NFA we are looking for. As a matter of fact, it is obtained by intersecting two basic automata, $\mathcal{A}_P$ and $\mathcal{E}_P$, that respectively deal with the following two variants of the timeline-based planning problem.

Definition 13 (Relaxed planning problem) *Let $P=(SV,S)$ be a timeline-based planning problem. The* relaxation *of P is the problem $relaxed(P)=(SV',S)$, where SV' is obtained from SV by replacing the transition and duration functions T_x and D_x of each $x \in SV$ by, respectively, the trivial functions T_x' and D_x' such that $T_x'(v)=V_x$ and $D_x'(v)=(0,+\infty)$, for all $v \in V_x$.*

Definition 14 (Unconstrained planning problem) *Let $P=(SV,S)$ be a timeline-based planning problem. The* unconstrained *version of P is the problem $basic(P)=(SV,\emptyset)$.*

Automaton $\mathcal{A}_P$. Given a timeline-based planning problem P, we first build an automaton $\mathcal{A}_P=(2^{\mathbb{V}},\Sigma,Q_0,\mathcal{F},\Delta)$, where:

- $2^{\mathbb{V}}$ is the finite set of states (states are sets of viewpoints), whose size is $2^{|\mathbb{V}|}$, that is, at most doubly exponential in the size of P;
- Σ is the input alphabet;
- $Q_0\subseteq 2^{\mathbb{V}}$ is the set of initial states, such that $\Upsilon\in Q_0$ if and only if $k=-1$ for all $(B,k)\in\Upsilon$ and for every trigger-less

[1] This disjointness condition directly follows from the definition of timeline, and plays a major role in the construction of the NFA.

synchronisation rule $\mathcal{R}$, there exists a viewpoint $(B, -1) \in \Upsilon$ such that $B \vDash \mathcal{R}$;

- $\mathcal{F} \subseteq 2^{\mathbb{V}}$ is the set of final states, defined as:

$$\mathcal{F} = \{\Upsilon \subseteq 2^{\mathbb{V}} \mid V \text{ is closed for all } V \in \Upsilon\};$$

- $\Delta \subseteq 2^{\mathbb{V}} \times \Sigma \times 2^{\mathbb{V}}$ is the transition relation, which reflects the aforementioned transition $\rightarrow$. Roughly speaking, a set of viewpoints Υ evolves into another one, say it Υ', upon reading token symbol σ if, and only if, each viewpoint $V \in \Upsilon$ evolves into a viewpoint $V' \in \Upsilon'$ according to relation $\rightarrow$ and, vice versa, each viewpoint $V' \in \Upsilon'$ is the image of a viewpoint $V \in \Upsilon$ with respect to relation $\rightarrow$.

  Formally, $(\Upsilon, \sigma, \Upsilon') \in \Delta$ if and only if it holds that:

  1. if $\Upsilon \in Q_0$, then σ is a starting symbol;
  2. for each $V \in \Upsilon$, there exists $V' \in \Upsilon'$ such that $V \xrightarrow{\sigma} V'$;
  3. for each $V' \in \Upsilon'$, there exists $V \in \Upsilon$ such that $V \xrightarrow{\sigma} V'$;
  4. if there is a synchronisation rule $R = a_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \ldots \vee \mathcal{E}_h$ in S such that $\sigma(x_0) = v_0$, then there exists $(B, k) \in \Upsilon'$ such that $B \vDash R$, (B, k) starts x, and $value((B, k), x) = v_0$.

For every token in the plan that triggers a synchronisation rule, condition 4 forces the existence of a viewpoint (B, k) to be used to satisfy that rule when triggered by that token. In particular, this condition forces the split of a viewpoint in situations analogous to the one described above, that is, when the blueprint of the viewpoint must be used to satisfy a rule triggered by two different tokens of the plan: indeed, if the viewpoint were not split, then one of the two triggers would never be instantiated and the computation would halt when the token symbol corresponding to the beginning of such token is read, without reaching a final state. Instead, satisfaction of trigger-less rules is guaranteed by the presence of suitable viewpoints in every initial state belonging to Q_0.

It is worth pointing out that $\mathcal{A}_P$ has two sources of nondeterminism: one is represented by the nondeterministic choice of the set of viewpoints contained in the initial states, while the other is given by viewpoints that can both pump and step in correspondence of some token symbol.

It holds that $|\mathcal{A}_P| \in \mathcal{O}(|\Sigma| \cdot 2^{|\mathbb{V}|})$, that is, the size of $\mathcal{A}_P$ is at most doubly exponential in the one of P.

It is possible to show that the language accepted by $\mathcal{A}_P$, denoted by $\mathcal{L}(\mathcal{A}_P)$, characterises the solutions of $relaxed(P)$ (see Definition 13).

Lemma 1 *Let P be a planning problem. $\mathcal{L}(\mathcal{A}_P) = \{w \in \Sigma_S\Sigma^* \cup \{\varepsilon\} \mid \mathbf{T}^w$ is a solution for $relaxed(P)\}$.*

Automaton $\mathcal{E}_P$. We now show how to build an NFA $\mathcal{E}_P$ that characterises the solutions of $basic(P)$ (see Definition 14), that is, that captures the evolution of values and durations along timelines, as imposed by functions T_x and D_x, for all $x \in SV$.

Let $\mathcal{E}_P = (Q, \Sigma, q_0, F, \Delta')$, where:

- $Q = ((\mathcal{V} \cup \{\epsilon\}) \times [N+1])^{|SV|}$ is the finite set of states, that is, states are pairs of functions (v, l), where $v : SV \rightarrow \mathcal{V} \cup \{\epsilon\}$ and $l : SV \rightarrow \{0, \ldots, N+1\}$;
- Σ is the input alphabet;
- $q_0 = (\epsilon, 0)^{|SV|} \in Q$ is the initial state;
- F is the set of final states, defined as:
  $F = \{(v, l) \in Q \setminus \{q_0\} \mid d_{min}^{x=v(x)} \leq l(x) \leq d_{max}^{x=v(x)}$ for all $x \in SV\}$;
- the transition relation $\Delta' \subseteq Q \times \Sigma \times Q$ is such that $((v, l), \sigma, (v', l')) \in \Delta'$ if and only if it holds that:
  1. if $(v, l) = q_0$, then $\sigma \in \Sigma_S$, $v'(x) = \sigma(x)$ and $l'(x) = 1$ for all $x \in SV$;
  2. if $(v, l) \neq q_0$, then, for each $x \in SV$:
     - if $\sigma(x) \neq \circlearrowleft$, then $\sigma(x) \in T_x(v(x))$, $d_{min}^{x=v(x)} \leq l(x) \leq d_{max}^{x=v(x)}$, $v'(x) = \sigma(x)$, and $l'(x) = 1$;
     - if $\sigma(x) = \circlearrowleft$, then $v'(x) = v(x)$ and

$$l'(x) = \begin{cases} l(x) + 1 & \text{if } l(x) < N + 1 \\ N + 1 & \text{otherwise} \end{cases}$$

It is easy to see that $|\mathcal{E}_P| \in \mathcal{O}(|\Sigma| \cdot (N \cdot |\mathcal{V}|)^{|SV|})$, that is, it is at most exponential in the size of P.

Lemma 2 *Let P be a planning problem. $\mathcal{L}(\mathcal{E}_P) = \{w \in \Sigma_S\Sigma^* \cup \{\varepsilon\} \mid \mathbf{T}^w$ is a solution for $basic(P)\}$.*

The main technical result of the paper is the next theorem, which is an immediate consequence of the following observation: a plan is a solution for P if and only if it is a solution for both $relaxed(P)$ and $basic(P)$.

Theorem 1 *P is satisfiable iff $\mathcal{L}(\mathcal{A}_P) \cap \mathcal{L}(\mathcal{E}_P) \neq \emptyset$.*

On-the-fly reachability check

The size of the automaton $\mathcal{A}_P \sqcap \mathcal{E}_P$, obtained by intersecting $\mathcal{A}_P$ and $\mathcal{E}_P$, is at most doubly exponential in the size of P. Therefore, a trivial procedure that first builds the automaton and then checks it for (non)emptiness would require too much space.

However, an EXPSPACE procedure can be obtained by merging the construction of the automaton and the reachability check into a unique on-the-fly procedure. A *nondeterministic* procedure has simply to guess a sequence of automaton states, starting from the initial one, choosing each among the possible successors of the current one. If a final state is reached within n steps, with $n = |\mathcal{A}_P \sqcap \mathcal{E}_P|$, then the procedure returns a positive answer; otherwise, the procedure terminates with a negative answer after $n + 1$ steps. The procedure only uses exponential space because it only needs to count (in binary) up to n. A *deterministic* equivalent procedure can be obtained by the well-known fact that EXPSPACE = NEXPSPACE.

Coupling our analysis with the known hardness result for the satisfiability checking for timeline-based planning problems (Gigante et al. 2016), we can state our main theorem.

Theorem 2 *Deciding whether a given timeline-based planning problem admits a solution plan is EXPSPACE-complete.*

A solution plan for the problem can be effectively extracted from the sequence of automaton states found by the above procedure. As a consequence, if the problem admits any solution, this procedure finds a solution of length bounded above by the doubly exponential size of the automaton. This is formally stated by the following corollary.

Corollary 1 *Let P be a timeline-based planning problem. If P admits any solution plan, then there exists at least one solution plan of length at most doubly exponential in the size of P.*

It is worth pointing out that a similar small-model theorem is the core of the result provided by (Gigante et al. 2017), whose proof, however, does *not* hold in the case of unbounded token durations and it involves non-trivial combinatorial arguments. Here, instead, it follows naturally from our construction.

Recurrent goals and infinite timelines

In this section, we show that the construction described in the previous section can be suitably adapted to deal with recurrent goals and infinite timelines. The formulation of infinite solution plans for recurrent goals (*recurrent solution plans*) to a large extent coincides with the standard one. The only change is the replacement of (finite) timelines with infinite ones.

First of all, Definition 3 is replaced by the following one.

Definition 15 (Infinite timeline) *An* infinite timeline *for a state variable $x = (V_x, T_x, D_x)$ is an infinite sequence $\mathsf{T} = \langle \tau_1, \tau_2 \ldots \rangle$ of tokens for x, where $v_{i+1} \in T_x(v_i)$ for $i \in \mathbb{N}^+$.*

The notion of recurrent solution plan is a simple reformulation of the standard notion of solution plan (see Definition 9) where (finite) timelines have been replaced by infinite ones.

Definition 16 (Recurrent solution plan) *Let $SV = \{x_1, \ldots, x_n\}$ be a set of state variables.*

A recurrent plan *over SV is a set of infinite timelines $\pi = \{\mathsf{T}_1, \ldots, \mathsf{T}_n\}$, one for each $x_i \in SV$.*

A recurrent solution plan *(or, simply,* recurrent solution*) for a timeline-based planning problem $P = (SV, S)$ is a recurrent plan π over SV such that all the synchronisation rules in S are satisfied by the set Γ of all the tokens involved in π, that is, $\Gamma = \{\tau \mid \tau \in \mathsf{T}, \mathsf{T} \in \pi\}$.*

In order to deal with infinite timelines, nondeterministic finite state automata must be replaced by Büchi ones. Let $P = (SV, S)$ be a timeline-based planning problem. We show that the existence of a recurrent solution plan for P can be reduced to the nonemptiness of the intersection language of two Büchi automata, namely $\mathcal{B}_P$ and $\mathcal{E}^\omega_P$, that suitably generalise the NFAs $\mathcal{A}_P$ and $\mathcal{E}_P$ from the previous section. The finite case can actually be recovered as a special case.

The automaton $\mathcal{E}^\omega_P$, which is meant to characterise infinite solutions of $basic(P)$ (see Definition 14), that is, which captures the evolution of values and durations along timelines, as imposed by functions T_x and D_x, for all $x \in SV$, is the same as $\mathcal{E}_P$, the only difference dwelling in the acceptance condition (a Büchi condition for the acceptance of infinite words, *ω-words* for short, rather than finite ones).

We turn now to the definition of the automaton $\mathcal{B}_P$, based on $\mathcal{A}_P$. In the following, we assume that $\Sigma, Q_0, \mathcal{F}$, and Δ are defined exactly as they have been defined in the previous section for $\mathcal{A}_P$. Given a planning problem P, let $\mathcal{B}_P = (2^{\mathbb{V}} \times 2^{\mathbb{V}}, \Sigma, Q_0 \times Q_0, 2^{\mathbb{V}} \times \mathcal{F}, \Delta_\omega)$, where $\Delta_\omega \subseteq (2^{\mathbb{V}} \times 2^{\mathbb{V}}) \times \Sigma \times (2^{\mathbb{V}} \times 2^{\mathbb{V}})$ is the transition relation between states of $\mathcal{B}_P$ or, equivalently, between *pairs of states* of $\mathcal{A}_P$. Formally, $((\Upsilon, \overline{\Upsilon}), \sigma, (\Upsilon', \overline{\Upsilon}')) \in \Delta_\omega$ if and only if it holds that:

1. $(\Upsilon, \sigma, \Upsilon') \in \Delta$;
2. if $\overline{\Upsilon} \notin \mathcal{F}$, then $(\overline{\Upsilon}, \sigma, \overline{\Upsilon}') \in \Delta$;
3. if $\overline{\Upsilon} \in \mathcal{F}$, then $\overline{\Upsilon}' = \Upsilon'$.

Informally, to deal with recurrent plans we run two copies of $\mathcal{A}_P$ in parallel (and thus the states of $\mathcal{B}_P$ are pairs $(\Upsilon, \overline{\Upsilon})$).

The first computation, whose state is represented by Υ, that is, the first component of the pair, proceeds exactly like a run of $\mathcal{A}_P$ and its duty is to ensure that every token that triggers a synchronisation rule is involved in some instantiation of a blueprint that satisfies the rule. However, due to the possible presence of *recurrent goals*, that is, rules imposing constraints to be verified infinitely often along the plan (see the example in Figure 2), such a component might be forced to never reach states in $\mathcal{F}$.

To overcome this problem, we introduce a second copy of $\mathcal{A}_P$, that is, the second component $\overline{\Upsilon}$, that works on separate, adjacent *chunks* of the input word, focusing each time on (previously triggered) synchronisation rules that have not yet been fulfilled at the beginning of the current chunk. When all these synchronisation rules (ignoring further synchronisation rules that might be triggered after the beginning of the chuck) have been satisfied (*i.e.*, $\overline{\Upsilon} \in \mathcal{F}$), the state $(\Upsilon, \overline{\Upsilon})$ is final for the automaton $\mathcal{B}_P$. With the next transition, leading to a new state $(\Upsilon', \overline{\Upsilon}')$, a new chunk is started by "taking a snapshot" of the first state component Υ', that is, Υ' is copied into $\overline{\Upsilon}'$.

Let us now consider the example depicted in Figure 2. The timeline-based planning problem is given over a single timeline for the variable x which can assume two possible values, v_0 and v_1. Moreover, tokens for x are constrained to have duration exactly 1. The problem basically forces the timeline for x to represent a word on the alphabet $\{v_0, v_1\}$. By means of two simple synchronisation rules, we may impose every v_0 token to be eventually followed by a v_1 token and, vice versa, every v_1 token to be eventually followed by a v_0 one. This means that there cannot be a finite solution for such a timeline-based planning problem. However, we can easily find a recurrent solution for it; a trivial one is represented by the infinite timeline $\mathsf{T} = \langle (v_0, 1), (v_1, 1), (v_0, 1), (v_1, 1) \ldots \rangle$ or, equivalently, the ω-word $(v_0 v_1)^\omega$, which presents an infinite strict alternation of v_0 and v_1 tokens. In Figure 2, we depicts the prefix of a run of the Büchi automaton over the ω-word $w = (v_0 v_0 v_0 v_1)^\omega$, which represents another recurrent solution to the problem. For the sake of completeness, we show in Figure 2 four blueprints, even though only blueprints B_2 and B_4 are needed for the solution. Intuitively, blueprint B_1 (resp., B_3) matches with parts of the timeline where a v_0 (resp., v_1) token is immediately followed by a v_1 (resp., v_0) one, while blueprint B_2 (resp., B_4) captures patterns where a v_0 (resp., v_1) token is eventually followed by a v_1 (resp., v_0) one, with at least one time point in between the two tokens. Final states are highlighted in Figure 2: we have the final

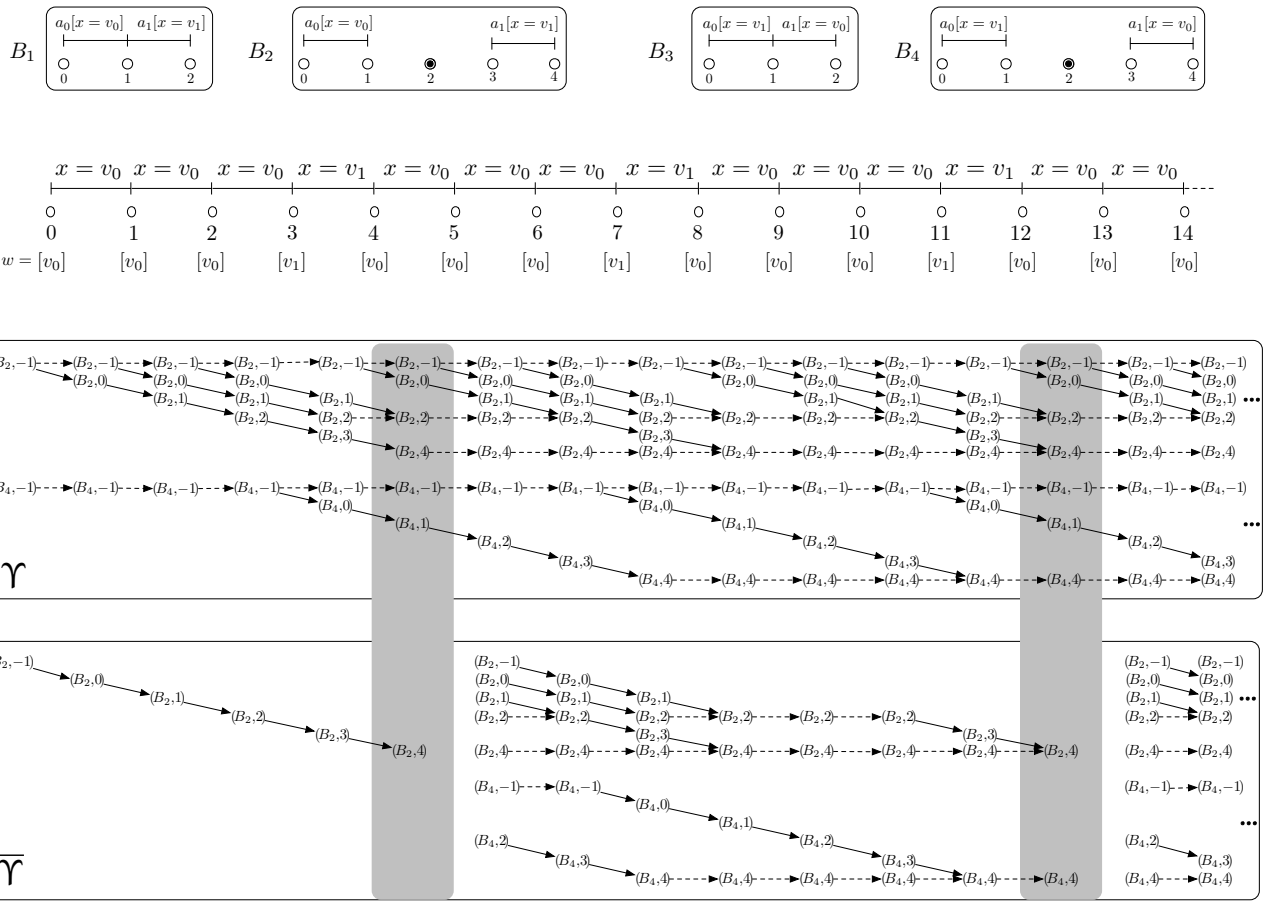

Figure 2: A run of the Büchi automaton for the planning problem $P = (\{(\{v_0, v_1\}, T_x, D_x)\}, \{a_0[x = v_0] \rightarrow \exists a_1[x = v_1](a_0 \leq^{s,e}_{[0,+\infty)} a_1), a_0[x = v_1] \rightarrow \exists a_1[x = v_0](a_0 \leq^{s,e}_{[0,+\infty)} a_1)\})$, where $T_x(v_i) = \{v_0, v_1\}$ and $D_x(v_i) = (1, 1)$ for all $i \in \{0, 1\}$. Final states are highlighted.

states

$$\left(\Upsilon = \left\{\begin{matrix}(B_2,-1),(B_2,0),(B_2,2),\\(B_2,4),(B_4,-1),(B_4,1)\end{matrix}\right\},\ \overline{\Upsilon} = \left\{(B_2,4)\right\}\right)$$

and

$$\left(\begin{array}{ll}\Upsilon & = \left\{\begin{matrix}(B_2,-1),(B_2,0),(B_2,2),(B_2,4),\\(B_4,-1),(B_4,1),(B_4,4)\end{matrix}\right\},\\ \overline{\Upsilon} & = \left\{(B_2,4),(B_4,4)\right\}\end{array}\right)$$

occurring, respectively, after reading the prefixes $v_0v_0v_0v_1v_0$ and $v_0v_0v_0v_1v_0v_0v_0v_1v_0v_0v_0v_1v_0$.

It is worth noticing that final states in $\mathcal{B}_P$ are determined only by the second component $\overline{\Upsilon}$, that is, $(\Upsilon, \overline{\Upsilon})$ is a final state of $\mathcal{B}_P$ if and only if $\overline{\Upsilon}$ is a final state of $\mathcal{A}_P$, and that the state that follows a final one along a run of $\mathcal{B}_P$, say it $(\Upsilon, \overline{\Upsilon})$, satisfies $\Upsilon = \overline{\Upsilon}$.

Moreover, note that finding a recurrent solution for a timeline-based planning problem (if any) is more general than finding a solution for it. As a matter of fact, it is possible to show that a timeline-based planning problem P can be translated into a timeline-based planning problem P' that only admits recurrent solutions, for which there exists an isomorphism between solutions for P and solutions for P' such that the former ones correspond to finite prefixes of the latter ones.

Finally, observe that the considerations made in the previous section on the size and the final state reachability check for the automaton $\mathcal{A}_P \sqcap \mathcal{E}_P$ (obtained by intersecting $\mathcal{A}_P$ and $\mathcal{E}_P$) still hold for the size and the final state recurrent reachability check for automaton $\mathcal{B}_P$, thus yielding the following complexity characterisation for the problem of finding (infinite) recurrent solutions for timeline-based planning problems.

Theorem 3 *Deciding whether a given timeline-based planning problem admits a recurrent solution plan is EXPSPACE-complete.*

Conclusions

In this work, we completely characterised the complexity of timeline-based planning problems showing that establishing whether a solution plan exists for a given problem is EXPSPACE-complete. Such a complexity bound was previously found in (Gigante et al. 2017) but only for problems

where unbounded token duration were forbidden. Here, we address the complexity of the problem in its full generality, without any artificial restriction on the syntax of synchronisation rules.

The automata-theoretic proof technique employed here is simpler and cleaner than the rather involved combinatorial argument used in (Gigante et al. 2017), and can be easily generalised to the case of infinite timelines, which was still an unexplored variant of the problem. The complexity of the plan existence problem is proved to be EXPSPACE-complete also in this case.

References

Barreiro, J.; Boyce, M.; Do, M.; Frank, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Ong, J.; Remolina, E.; Smith, T.; and Smith, D. 2012. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *Proc. of the 4th International Competition on Knowledge Engineering for Planning and Scheduling*.

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Policella, N. 2007. An Innovative Product for Space Mission Planning: An A Posteriori Evaluation. In *Proc. of the 17th International Conference on Automated Planning and Scheduling*, 57–64.

Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-based space operations scheduling with external constraints. In *Proc. of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 34–41.

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

Della Monica, D.; Gigante, N.; Montanari, A.; Sciavicco, G.; and Sala, P. 2017. Bounded timed propositional temporal logic with past captures timeline-based planning with bounded constraints. In *Proc. of the 26th International Joint Conference on Artificial Intelligence*, 1008–1014.

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

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

Gigante, N.; Montanari, A.; Cialdea Mayer, M.; and Orlandini, A. 2016. Timelines are Expressive Enough to Capture Action-based Temporal Planning. In *Proc. of the 23rd International Symposium on Temporal Representation and Reasoning*, 100–109.

Gigante, N.; Montanari, A.; Cialdea Mayer, M.; and Orlandini, A. 2017. Complexity of timeline-based planning. In *Proc. of the 27th International Conference on Automated Planning and Scheduling*, 116–124.

Jónsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. D. 2000. Planning in Interplanetary Space: Theory and Practice. In *Proc. of 5th International Conference on Artificial Intelligence Planning and Scheduling*, 177–186.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M., and Fox, M. S., eds., *Intelligent Scheduling*. Morgan Kaufmann. chapter 6, 169–212.

Rintanen, J. 2007. Complexity of Concurrent Temporal Planning. In *Proc. of the 17th International Conference on Automated Planning and Scheduling*, 280–287.

Umbrico, A.; Cesta, A.; Cialdea Mayer, M.; and Orlandini, A. 2017. PLATINUm: A New Framework for Planning and Acting in Human-Robot Collaborative Scenarios. In *Proc. of the 16th International Conference of the Italian Association for Artificial Intelligence*, 498–512.