A Game-Theoretic Approach to Timeline-Based Planning with Uncertainty

Nicola Gigante[1]
University of Udine, Udine, Italy
gigante.nicola@spes.uniud.it

Angelo Montanari[1]
University of Udine, Udine, Italy
angelo.montanari@uniud.it

Marta Cialdea Mayer[1]
University of Roma Tre, Rome, Italy
cialdea@ing.uniroma3.it

Andrea Orlandini
National Research Council, Rome, Italy
andrea.orlandini@istc.cnr.it

Mark Reynolds[2]
The University of Western Australia, Perth, Australia
mark.reynolds@uwa.edu.au

—— **Abstract** ——

In timeline-based planning, domains are described as sets of independent, but interacting, components, whose behaviour over time (the set of timelines) is governed by a set of temporal constraints. A distinguishing feature of timeline-based planning systems is the ability to integrate planning with execution by synthesising control strategies for *flexible plans*. However, flexible plans can only represent *temporal uncertainty*, while more complex forms of nondeterminism are needed to deal with a wider range of realistic problems. In this paper, we propose a novel game-theoretic approach to timeline-based planning problems, generalising the state of the art while uniformly handling temporal uncertainty and nondeterminism. We define a general concept of timeline-based *game* and we show that the notion of winning strategy for these games is strictly more general than that of control strategy for dynamically controllable flexible plans. Moreover, we show that the problem of establishing the existence of such winning strategies is decidable using a doubly exponential amount of space.

---

1   Introduction

In the timeline-based approach to planning [21], the world is modelled as a set of independent, but interacting, components, whose behaviour over time, the *timelines*, is governed by a set of temporal constraints. This approach differs from the classical *action-based* planning, relying on PDDL [13], for its more declarative nature and its focus on temporal reasoning (see, *e.g.* [9, 11, 14, 15, 16, 21]). Timeline-based systems have been successfully deployed in a number of complex scenarios, ranging from space operations [3, 7, 21] to manufacturing [5, 24]. An important feature of timeline-based planning systems is their ability to integrate planning with execution by means of *flexible plans*, which represent envelopes of possible solutions that differ in the execution times and/or the duration of tasks. Flexible plans allow the controller to handle the *temporal uncertainty* involved in dealing with partially controllable elements and the external environment. Cialdea Mayer *et al.* [9] rigorously defined the concept of timeline-based planning specification as well as *dynamically controllable* flexible plans, which can be executed guaranteeing to satisfy the problem constraints while reactively handling any *temporal uncertainty* in the uncontrollable behaviour. A technique for synthesising dynamic control strategies is shown in [8].

Other forms of uncertainty, such as *nondeterminism* (*i.e.*, *which* tasks the environment chooses to perform), are not supported: even for external variables, completely controlled by the environment, their evolution is known up to temporal uncertainty only. This choice to focus on temporal reasoning and temporal uncertainty is coherent with the history and scope of timeline-based systems. However, it is not completely reflected into the grammar of modelling languages used in timeline-based systems, which are expressive enough to model complex scenarios that require the system to handle non-temporal nondeterminism. In such cases, current systems often employ a *re-planning* stage as part of their execution cycle (see, *e.g.* [24]): any mismatch between the expected and actual behaviours of the environment results into a revision of the flexible plan, which then can resume execution. Unfortunately, the cost of such a re-planning phase may be incompatible with the requirements of real-time execution and, more importantly, if a wrong choice is made by the original flexible plan, the re-planning might happen too late to be able to recover a controllable state of the system. Hence, knowledge engineers have to explicitly account for this problem if they want to avoid unnecessary failures and costly re-planning during execution, which make the system less effective and more complex to use.

Nondeterministic planning issues have been extensively investigated within the action-based planning framework following different approaches such as, for instance, reactive planning systems [4], deductive planning [23], POMDP [18], and model checking [12]. More recently, fully observable nondeterministic (FOND) planning problems have been addressed [19, 20] also considering temporally extended goals [6, 22]. However, action-based planning does not support flexible plans and temporal uncertainty, and it does not take into account controllability issues. Recently, SMT-based techniques have been exploited to deal with uncontrollable durations in strong temporal planning [10]; however, dynamic controllability issues are not addressed.

This paper defines the novel concept of *timeline-based planning game*, a game-theoretic generalisation of the timeline-based planning problem with uncertainty, which uniformly treats both temporal uncertainty and general nondeterminism. In these games, the controller tries to satisfy the given temporal constraints no matter what the choices of the environment are. We compare the proposed games with the current approaches based on flexible plans. In particular, we show how current timeline-based modelling languages can express problems

that, only seeming to involve temporal uncertainty at first, in fact model scenarios which would require the controller to handle non-temporal nondeterminism. We show that these problems do not admit dynamically controllable flexible plans (as defined in [9]), but do admit winning strategies when seen as instances of timeline-based games. A study of the decidability and complexity of the problem of establishing the existence of a winning strategy for a given timeline-based planning game concludes the paper.

The paper is structured as follows. Section 2 briefly recaps the basic definitions, and Section 3 discusses the limitations of the approach employed in state-of-the-art timeline-based planning systems. Then, Section 4 defines timeline-based planning games, and shows its greater generality with respect to the current approach. Finally, Section 5 addresses decidability and complexity issues. Section 6 concludes discussing future developments.

2 Timeline-based planning

This section introduces timeline-based planning and describes the state of the art of the field with regards to how uncertainty is handled by current timeline-based systems. As a representative of the modelling languages used by existing systems, we chose the formal language introduced in [9]. We first introduce the basic concepts of the framework, without considering uncertainty, as studied in [15, 16]; then, we add uncertainty to the picture and discuss how it is handled by current systems.

2.1 Basic definitions

State variables are the basic building blocks of the timeline-based planning framework.

▶ **Definition 1** (State variables). A *state variable* is a tuple $x = (V_x, T_x, D_x, \gamma_x)$, where:
- V_x is the *finite domain* of the variable;
- $T_x : V_x \to 2^{V_x}$ is the *value transition function*, which maps each value $v \in V_x$ to the set of values that can follow it;
- $D_x : V_x \to \mathbb{N}^+ \times (\mathbb{N}^+ \cup \{+\infty\})$ is a function that maps each $v \in V_x$ to the pair $(d_{min}^{x=v}, d_{max}^{x=v})$ of the minimum and maximum duration of any interval over which $x = v$;
- $\gamma_x : V_x \to \{c, u\}$ is a function called *controllability tag* (see Section 2.2).

Which value is taken by a state variable over a specified time interval is described by means of *tokens*. The behaviour of a state variable over time is modelled by a finite sequence of tokens, called a *timeline*.

▶ **Definition 2** (Timelines). A *token* for x is a tuple $\tau = (x, v, d)$, where x is a state variable, $v \in V_x$ is the value held by the variable, and $d \in \mathbb{N}^+$ is the *duration* of the token. A *timeline* for a state variable $x = (V_x, T_x, D_x, \gamma_x)$ is a finite sequence $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ of tokens for x.

For any token $\tau_i = (x, v_i, d_i)$ in a timeline $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ we can define the functions $\mathsf{start\text{-}time}(\mathsf{T}, i) = \sum_{j=1}^{i-1} d_j$ and $\mathsf{end\text{-}time}(\mathsf{T}, i) = \mathsf{start\text{-}time}(\mathsf{T}, i) + d_i$, hence mapping each token to the corresponding time interval $[\mathsf{start\text{-}time}, \mathsf{end\text{-}time})$ (right extremum excluded). As an example, the time interval associated with the token $\tau_1 = (x, 2, 5)$ is $[0, 5)$. When there is no ambiguity, we write $\mathsf{start\text{-}time}(\tau_i)$ and $\mathsf{end\text{-}time}(\tau_i)$ to denote, respectively, $\mathsf{start\text{-}time}(\mathsf{T}, i)$ and $\mathsf{end\text{-}time}(\mathsf{T}, i)$. The *horizon* of a timeline $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ is defined as $\mathcal{H}(\mathsf{T}) = \mathsf{end\text{-}time}(\tau_k)$. A timeline T can be empty, in which case we define its horizon as $\mathcal{H}(\mathsf{T}) = 0$.

The problem domain and the goal are modelled by a set of temporal constraints, called *synchronisation rules*. For the sake of space, we do not provide a detailed account of their syntax. Informally, each synchronisation rule has the following form:

$$rule := a_0[x_0 = v_0] \longrightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k, \ with$$
$$\mathcal{E}_i := \exists a_1[x_1 = v_1]a_2[x_2 = v_2]\ldots a_n[x_n = v_n] \ . \ \mathcal{C}$$

where $x_0, \ldots, x_n$ are state variables and $v_0, \ldots, v_n$ are values, with $v_i \in V_{x_i}$ for all i. Each rule thus consists of a *trigger* $(a[x_0 = v_0])$ and a disjunction of *existential statements*. It is satisfied if for each token satisfying the trigger, at least one of the disjuncts is satisfied. The trigger can be empty (T), in which case the rule is said to be *triggerless* and asks for the satisfaction of the body without any precondition. Each existential statement requires the existence of some tokens such that the clause $\mathcal{C}$ is satisfied. The clause is in turn a conjunction of *atoms*, that is, atomic relations between endpoints of the quantified tokens, of the form $a \leq_{[l,u]}^{e_1,e_2} b$, where a and b are token names, $e_1, e_2 \in \{\mathsf{s}, \mathsf{e}\}$, $l \in \mathbb{N}$, and $u \in \mathbb{N} \cup \{+\infty\}$. Intuitively, each atom relates the start (s) or end (e) of the two tokens, and l and u are respectively a lower and upper bound to the distance between the two endpoints. Pointwise atoms relating a single endpoint with a specific point in time are also possible, *e.g.*, $a \leq_{[l,u]}^{\mathsf{s}} t$. An atom $a \leq_{[l,u]}^{e_1,e_2} b$ is *bounded* if $u \neq +\infty$, and *unbounded* otherwise. With these basic atomic relations, one can express all the Allen's relations over time intervals [1]. As an example, one can define $a \, \mathsf{meets} \, b$ as $a \leq_{[0,0]}^{\mathsf{e},\mathsf{s}} b$, or $a \, \mathsf{during} \, b$ as $a \leq_{[0,+\infty]}^{\mathsf{s},\mathsf{s}} b \wedge b \leq_{[0,+\infty]}^{\mathsf{e},\mathsf{e}} a$. Moreover, one can constrain the duration of tokens, *e.g.*, writing $\mathsf{duration}(a) = k$ or $\mathsf{duration}(a) \geq k$ as a shorthand for, respectively, $a \leq_{[k,k]}^{\mathsf{s},\mathsf{e}} a$ and $a \leq_{[k,+\infty]}^{\mathsf{s},\mathsf{e}} a$. Moreover, disjunctions in synchronisation rules allows one to express some forms of conditional (if/then/else) statements.
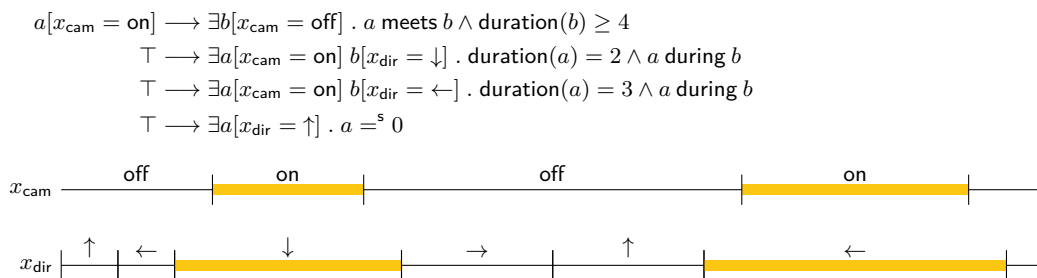
In the simplest setting, a timeline-based planning problem consists of a set of state variables and a set of rules that represent the problem domain and the goal. A solution to such a problem is simply a set of timelines that satisfy the rules.

▶ **Definition 3** (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 4** (Solution plan). A *scheduled solution plan* for a problem $P = (SV, S)$ is a set Γ of scheduled timelines, one for each $x \in SV$, such that $v_{i+1} \in T_x(v_i)$ and $d_{min}^{x=v} \leq d_i \leq d_{max}^{x=v}$ for all tokens $\tau_i = (x_i, v_i, d_i) \in \mathsf{T}_x$, and all the rules in S are satisfied.

Note that the controllability tags γ_x of state variables are ignored in Definition 4, as it only considers the problem of satisfying the synchronisation rules and the issues arising from the execution of the plan are not considered. Even without considering any form of uncertainty, timeline-based planning is already quite a hard problem. Indeed, we know from [16] that the problem of deciding whether there exists a solution plan for a given timeline-based planning problem is EXPSPACE-complete.

As an example, consider Figure 1, which shows a possible solution for a problem with two state variables, x_{cam} and x_{dir}, that respectively represent the on/off state of a camera and its direction. The transition function $T_{x_{\mathsf{dir}}}$ of the second variable is such that the camera can only stay still or move counterclockwise, that is, $T_{x_{\mathsf{dir}}}(\leftarrow) = \{\leftarrow, \downarrow\}$. The first rule states the system requirement that the camera must remain switched off at least four time steps after each use, to let the components cool down. The objective of the system is that of performing some shoots of a given duration, provided the camera is pointed in the right direction: a shoot downwards lasting two time steps, and a shoot toward left lasting three time steps, in

$$a[x_{\mathsf{cam}} = \mathsf{on}] \longrightarrow \exists b[x_{\mathsf{cam}} = \mathsf{off}] \ . \ a \ \mathsf{meets} \ b \wedge \mathsf{duration}(b) \geq 4$$
$$\top \longrightarrow \exists a[x_{\mathsf{cam}} = \mathsf{on}] \ b[x_{\mathsf{dir}} = \downarrow] \ . \ \mathsf{duration}(a) = 2 \wedge a \ \mathsf{during} \ b$$
$$\top \longrightarrow \exists a[x_{\mathsf{cam}} = \mathsf{on}] \ b[x_{\mathsf{dir}} = \leftarrow] \ . \ \mathsf{duration}(a) = 3 \wedge a \ \mathsf{during} \ b$$
$$\top \longrightarrow \exists a[x_{\mathsf{dir}} = \uparrow] \ . \ a =^{\mathsf{s}} 0$$

■ **Figure 1** An example of timeline-based planning problem. Two state variables are used to represent the on/off state of a camera x_{cam} and its pointing direction x_{dir}. The transition function of x_{dir} forces the camera to only move counterclockwise.

an arbitrary order. This objective is encoded by the second and third synchronisation rules, which are triggerless. The last rule expresses the fact that the camera is initially pointed upwards.

2.2 Timeline-based planning with uncertainty

A distinguishing feature of existing timeline-based planning systems is their ability to integrate planning and execution, accounting for the unavoidable uncertainty that comes from the interaction with the environment where the plan is executed. We now recall how timeline-based planning problems with uncertainty are defined in [9].

Two different sources of uncertainty can be represented by this model. The first comes from *external variables*, which are completely under the control of the environment, in contrast to *controlled variables* (also called *planned* variables), which are dealt with the system. The planner is not allowed to decide anything regarding the behaviour of external variables. The second is the duration of *uncontrollable tokens*. The *controllability tag* γ_x, associated with each state variable x, states whether a token where $x = v$ is *controllable* ($\gamma_x(v) = \mathsf{c}$) or *uncontrollable* ($\gamma_x(v) = \mathsf{u}$). The execution of uncontrollable tokens is planned and performed by the system, but their duration cannot be known in advance, *e.g.*, one may go shopping without knowing how much time he/she will have to wait at the counter. In both cases, only *temporal* uncertainty is considered in the current approach.

To deal with the uncertainty inherent in the execution of the plan, timeline-based planning systems make use of the concept of *flexible timeline*.

▶ **Definition 5** (Flexible timeline). A *flexible token* is a tuple $\tau = (x, v, [e, e'], [d, d'])$, where x is a state variable, $v \in V_x$, $[e, e'] \in \mathbb{N} \times \mathbb{N}$ is the interval of possible token *end times*[3], and $[d, d'] \in \mathbb{N}^+ \times \mathbb{N}^+$ is the interval of possible token *durations*.

A *flexible timeline* for a state variable $x = (V_x, T_x, D_x, \gamma_x)$ is a finite sequence $\mathsf{T}_x = \langle \tau_1, \ldots, \tau_k \rangle$ of flexible tokens for x where $[e_1, e_1'] = [d_1, d_1']$, and $[e_i, e_i'] \subseteq [e_{i-1} + d_i, e_{i-1}' + d_i']$ for all $1 < i \leq k$.

A flexible timeline represents a set of different timelines which differ in the precise timings of the described events. Tokens, timelines, and plans will also be referred as *scheduled* tokens, timelines and plans, to better differentiate them from flexible ones. A scheduled timeline

---

[3] Flexible tokens report their end times rather than their start times because in this way a flexible timeline can precisely constrain its horizon.

$\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ is an *instance* of a flexible timeline $\mathsf{T}' = \langle \tau_1', \ldots, \tau_k' \rangle$ if for each $\tau_i = (x, v, d)$ and $\tau_i' = (x', v', [e, e'], [d, d'])$, we have $x = x'$, $v = v'$, $d \in [d, d']$, and $\mathsf{end\text{-}time}(\tau_i) \in [e, e']$.

Flexibility can be naturally lifted from timelines to plans. The notion of *flexible plan* is formally defined as follows.

▶ **Definition 6** (Flexible plan). A *flexible plan* over a set of state variables SV is a pair $\Pi = (\Gamma, \mathcal{R})$, where Γ is a set of *flexible timelines*, exactly one for each $x \in SV$, and $\mathcal{R}$ is a set of *atoms* over the tokens occurring in Γ.

Given a flexible plan $\Pi = (\Gamma, \mathcal{R})$, a scheduled plan Γ' is an *instance* of Π if the timelines in Γ' are instances of those in Γ, and they satisfy the atoms in $\mathcal{R}$.

A flexible plan can be viewed as a tentative set of solutions to a planning problem where the precise timing of execution and the duration of tokens are chosen during execution. The set of atoms that comes together with the set of flexible timelines allow the plan to specify additional constraints over the tokens that compose the timelines. Flexible plans can in particular be used to describe the expected behaviour of external variables, of which one may know the future evolution only up to some temporal uncertainty. It is worth pointing out that a flexible plan is *unconditional*, *i.e.*, a single plan is committed to a specific sequence of state variable values, and the only freedom left concerns token durations.

A timeline-based planning problem with uncertainty is formally defined as follows.

▶ **Definition 7** (Timeline-based planning problem with uncertainty). A *timeline-based planning problem with uncertainty* is a tuple $P = (SV_C, SV_E, S, \mathcal{O})$, where:
- SV_C and SV_E are the sets of, respectively, the *controlled* and the *external* variables;
- S is a set of synchronisation rules over $SV_C \cup SV_E$;
- the *observation* $\mathcal{O} = (\Gamma_E, \mathcal{R}_E)$ is a flexible plan over SV_E specifying the behaviour of external variables.

Definition 7 differs from Definition 3 in two main respects: it splits the set of variables into controlled and external ones and it includes a flexible plan describing the temporally uncertain behaviour of external variables.

To solve the problem, one has to find a set of flexible timelines for the controlled variables such that the rules can be satisfied by a suitable set of instances.

▶ **Definition 8** (Flexible solution plan). Let $P = (SV_C, SV_E, S, \mathcal{O})$, with $\mathcal{O} = (\Gamma_E, \mathcal{R}_E)$ be a timeline-based planning problem with uncertainty. A flexible plan $\Pi = (\Gamma, \mathcal{R})$ over $SV_C \cup SV_E$ is a *flexible solution plan* for P if:
1. Π agrees with $\mathcal{O}$, that is, $\Gamma_E \subseteq \Gamma$ and $\mathcal{R}_E \subseteq \mathcal{R}$;
2. the plan does not restrict the duration of uncontrollable tokens, that is, for any $\mathsf{T} \in \Gamma$ and any token $\tau = (x, v, [e, e'], [d, d']) \in \mathsf{T}$, if $\gamma_x(v) = \mathsf{u}$, then $d = d_{min}^{x=v}$ and $d' = d_{max}^{x=v}$;
3. any *instance* of Γ is a scheduled solution plan for the timeline-based planning problem $P' = (SV_C \cup SV_E, S)$, and there exists at least one such instance.

Note that, despite the name, which is borrowed from [9], the observation $\mathcal{O}$ is rather an *a priori* description of the environment behaviour, which is supposed to be completely known up to the given temporal uncertainty. Usual definitions of planning problems involve the specification of a maximum bound on the *horizon* of the solution plans. For the sake of generality, we omit this parameter, as it can be expressed by suitable synchronisation rules.

According to Item 3 of Definition 8, any instance of a flexible plan satisfies the synchronisation rules of the problem. However, there is no guarantee that one such instance exists for each possible instance of the external timelines. In other words, Definition 8 does

not guarantee that a flexible solution plan can be executed in any possible scenario. Thus, a control strategy is needed to determine how to schedule controllable tasks. Because of space concerns, we cannot present all the details of the definition of control strategy, which is thoroughly illustrated in [9]. Informally, it can be thought of as a function σ which chooses how to schedule the start time of tokens for controlled variables, and the end time of controllable tokens, during execution. A flexible solution plan Π is said to be *weakly controllable* if for each possible schedule of tokens for external variables and of uncontrollable tokens, there is a control strategy σ such that following σ during execution results into an instance of Π. It is said to be *strongly controllable* if, conversely, a single control strategy σ exists which results into an instance of Π whatever the schedule of the endpoints under its control by the environment is. Finally, it is said to be *dynamically controllable* if the controlled endpoints can be scheduled by taking into account, at any given time, only the past history of the execution in such a way that an instance of Π is obtained. Given its generality and wider applicability, dynamic controllability is definitely the most interesting form of controllability. Notice that these concepts, whose definition in the context of flexible plans for timeline-based planning is given in [9], have analogous counterparts in the context of temporal networks [25].

Timeline-based systems which aim at handling both planning and execution cannot simply produce flexible plans, but have to ensure a chosen degree of controllability of the produced plans. As an example, the PLATINUm system [24] employs a two-phase process where a flexible plan is first produced and then checked for the existence of a dynamic control strategy. Dynamic controllability of a flexible plan can be checked, for instance, via a reduction to timed game automata [8]. Since uncontrollable flexible plans are not suitable to be executed, the problem is considered to be solved only when a dynamically controllable flexible solution plan is found, together with its dynamic control strategy.

3   Limitations of the current approach

The focus of timeline-based systems on temporal reasoning and temporal uncertainty clearly emerges from the previous section. This focus has its roots in the history of the paradigm and the typical application scenarios where timeline-based systems have been employed. The exclusive focus on temporal uncertainty is especially evident in the treatment of external variables: their behaviour is supposed to be completely known excepting only the precise timing of specific events. Consider, for example, a satellite in orbit doing some measurements and transmitting the results back to Earth. In such a domain, external variables might be used to represent visibility windows where the different ground stations can be reached by the satellite. The precise timing of those windows is uncertain, but everything else is known (even months) in advance. Nevertheless, to handle the case of a mismatch between the expected and the observed behaviour of the environment, systems such as PLATINUm employ a feedback loop where a *failure manager* is in charge of triggering, if needed, a *re-planning* (or plan repair) phase, which should produce a new flexible plan, with a suitable control strategy, taking into account the newly acquired observations (the name of the $\mathcal{O}$ component in Definition 7 comes from this scenario).

In contrast to domains as the above one, other applications might require to re-plan more frequently. As an example, in robotics scenarios such as those discussed in [24], where the planned system interacts with a human agent, one cannot hope to represent with temporal uncertainty alone all the possible variability of the behaviour of the external environment. In these scenarios, most often re-plans get triggered to handle the unpredicted outcomes

of generally nondeterministic choices that the external agents can make, rather than to fix problems in the domain model. This observation motivates us in proposing an interpretation of timeline-based problems that is able to handle both temporal uncertainty and general nondeterminism, extending and generalising the current approach based on flexible plans.

As a matter of fact, it turns out that the domain description languages commonly in use, here exemplified by the formal syntax defined in Section 2, can easily express scenarios where the need to handle general nondeterminism arise in problems which apparently only involved temporal uncertainty. To see how this may happen, consider a simple timeline-based planning problem with uncertainty $P = (SV_C, SV_E, S, \mathcal{O})$, with a single state variable x, with $V_x = \{v_1, v_2, v_3\}$, $SV_E = \varnothing$, and S consisting of the following synchronisation rules:

$$a[x = v_1] \longrightarrow \exists b[x = v_2] \ . \ a \leq_{[0,0]}^{\mathsf{e,s}} b \wedge a \leq_{[0,5]}^{\mathsf{s,e}} a \vee \exists c[x = v_3] \ . \ a \leq_{[0,0]}^{\mathsf{e,s}} c \wedge a \leq_{[6,10]}^{\mathsf{s,e}} a$$
$$\top \longrightarrow \exists a[x = v_1] \ . \ a =^{\mathsf{s}} 0$$

Suppose that $D_x(v) = [1, 10]$ for all $v \in V_x$, and that $\gamma_x(v_1) = \mathsf{u}$ and $\gamma_x(v_2) = \gamma_x(v_3) = \mathsf{c}$, that is, tokens where $x = v_1$ are uncontrollable. The rules require the execution to start with a token where $x = v_1$, followed by a token where either $x = v_2$ or $x = v_3$ depending on the duration of the first token. This scenario is, intuitively, trivial to control. The system must execute $x = v_1$ as a first token due to the second rule. Then, the environment controls its duration, and the system simply has to wait for the token to end, and then execute either $x = v_2$ or $x = v_3$ depending on how long the first token lasted. However, there are no flexible plans that represent this simple strategy, since each given plan must fix the value of every token in advance. To guarantee the satisfaction of the rules, the choice of which value to assign to x on the second token must be made during the execution, but this is not possible because of the exclusively sequential nature of flexible plans. In this case, therefore, the problem would be considered as unsolvable, even if the goals stated by the rules seem simple to achieve.

The issues discussed here come from the lack of a proper support for general nondeterminism in the framework of flexible plans. However, the last example shows that this is not just a missing feature of current systems, but rather a class of scenarios that can be easily modelled by (the syntax of) timeline-based description languages but whose solutions are not captured by the commonly considered semantics. The *timeline-based planning games* defined in the next section provide a clean way to express the solution to this kind of scenario, providing a semantics to timeline-based planning problems with uncertainty capable of modelling both temporal uncertainty and general nondeterminism in a uniform way. Moreover, they handle the external variables in the most general way, without assuming any *a priori* knowledge of their future behaviour.

4 Timeline-based planning games

Let us now introduce *timeline-based planning games*. They generalise dynamic control strategies for flexible plans while suitably handling the limitations discussed in Section 3.

▶ **Definition 9** (Timeline-based planning game)**.** A *timeline-based planning game* is a tuple $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$, where SV_C and SV_E are the sets of, respectively, the *controlled* and the *external* variables, and $\mathcal{S}$ and $\mathcal{D}$ are two sets of synchronisation rules, respectively called *system* and *domain* rules, involving variables from both SV_C and SV_E.

Intuitively, a timeline-based planning game $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ is a turn-based, two-player game played by the controller, *Charlie*, and the environment, *Eve*. By playing the game, the players progressively build the timelines of a *scheduled plan* (see Definition 4). At
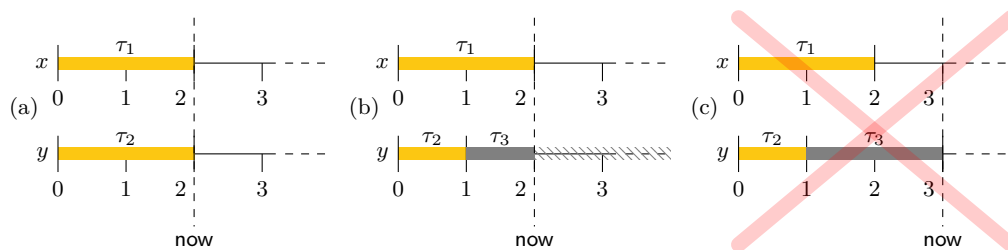
■ **Figure 2** Examples of partial plans: (a) two tokens for variables x and y where $\mathsf{end\text{-}time}(\tau_1) = \mathsf{end\text{-}time}(\tau_2) = 2$ and $\mathsf{now} = 2$; (b) $\mathsf{end\text{-}time}(\tau_1) = 2$, $\mathsf{end\text{-}time}(\tau_2) = 1$, while τ_3 continues; (c) incorrect partial plan, because τ_3 continues over the end of τ_1 and a successor for τ_1 is unspecified.

each round, each player makes a move deciding which tokens to start and/or to stop and at which time. Both players are constrained by the set $\mathcal{D}$ of *domain* rules, which describe the basic rules governing the world. Domain rules replace and generalise the *observation* $\mathcal{O}$ of Definition 7, allowing one to freely model the interaction between the system and the environment. They are not intended to be *Eve*'s (or *Charlie*'s) *goals*, but, rather, a background knowledge about the world that can be assumed to hold at any time. Since neither player can violate $\mathcal{D}$, the strategy of each player may safely assume the validity of such rules. In addition, *Charlie* is responsible for satisfying the set $\mathcal{S}$ of *system* rules, which describe the rules governing the controlled system, including its goals. *Charlie* wins if, assuming *Eve* behaves according to the domain rules, he manages to construct a plan satisfying the system rules. In contrast, *Eve* wins if, while satisfying the domain rules, she prevents *Charlie* from winning, either by forcing him to violate some system rule, or by indefinitely postponing the fulfilment of his goals.

Let us now formally describe the way in which a play of a (timeline-based) planning game evolves. First of all, we observe that at any given time during the play, the plan will be partially built, waiting for some tokens to be completed. A *partial plan* is a plan where the last token on each timeline may be unfinished (*open* token). A timeline whose last token is open is called an *open* timeline.

▶ **Definition 10** (Open timeline). Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a planning game and let $SV = SV_C \cup SV_E$. An *open token* for G is a pair $\tau = (x, v)$, where $x \in SV$ and $v \in V_x$. An *open timeline* for $x \in SV$ is a non-empty finite sequence of tokens $\mathsf{T} = \langle \tau_1, \dots, \tau_{k-1}, \tau_k \rangle$, where $\langle \tau_1, \dots, \tau_{k-1} \rangle$ is a scheduled timeline for x and $\tau_k = (x, v_k)$ is an open token.

We will refer to tokens and timelines as defined in Definition 2 as *closed* tokens and *closed* timelines, respectively. In an open timeline $\mathsf{T} = \langle \tau_1, \dots, \tau_k \rangle$, only $\mathsf{start\text{-}time}(\tau_k)$ is defined for its last open token τ_k: $\mathsf{start\text{-}time}(\tau_0) = 0$ and $\mathsf{start\text{-}time}(\tau_i) = \mathsf{end\text{-}time}(\tau_{i-1})$ for $i > 1$. Recall that $\mathcal{H}(\mathsf{T})$ for a closed timeline is the $\mathsf{end\text{-}time}$ of its last token and that $\mathcal{H}(\mathsf{T}) = 0$ for empty timelines. For an open timeline $\mathsf{T} = \langle \tau_1, \dots, \tau_{k-1}, \tau_k \rangle$, we define its horizon as the $\mathsf{end\text{-}time}$ of its last *closed* token, *i.e.*, $\mathcal{H}(\mathsf{T}) = \mathsf{end\text{-}time}(\tau_{k-1})$ (which is equal to $\mathsf{start\text{-}time}(\tau_k)$).

▶ **Definition 11** (Partial plan). Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a planning game and let $SV = SV_C \cup SV_E$. A *partial plan* for G is a pair $\Pi = (\Gamma, \mathsf{now})$, where Γ is a set of timelines, either open or closed, one for each $x \in SV$, and $\mathsf{now} \in \mathbb{N}$ is the current time, such that:
1. $\mathcal{H}(\mathsf{T}) < \mathsf{now}$ for any *open* timeline $\mathsf{T} \in \Gamma$;
2. $\mathcal{H}(\mathsf{T}) = \mathsf{now}$ for any *closed* timeline $\mathsf{T} \in \Gamma$;

For a partial plan $\Pi = (\Gamma, \mathsf{now})$, we write $\mathsf{T} \in \Pi$ to mean $\mathsf{T} \in \Gamma$, and we define $\mathcal{H}(\Pi) = \max_{\mathsf{T} \in \Pi} \mathcal{H}(\mathsf{T})$ and $\mathsf{now}(\Pi) = \mathsf{now}$. The set of all possible partial plans for a given game G is denoted as $\mathbf{\Pi}_G$, or simply $\mathbf{\Pi}$ where G is understood.

Definition 11 implies that time cannot advance after the end of a token without specifying its successor, and ensures that the partial plan has been built only up to now. Figure 2 shows an example of invalid partial plan where a token continues after the end of another and the successor of the latter is not specified, forming an invalid gap in the description of the timelines. It also implies that if any empty, thus closed, timeline is present, then all timelines are empty. Thus, a unique well-defined *empty* partial plan exists, which we will denote by $\Pi_\varnothing$, with all empty timelines and $\mathsf{now}(\Pi_\varnothing) = 0$.

Players incrementally build a partial plan by extending the initially empty partial plan $\Pi_\varnothing$. This is done by means of actions that specify which tokens to start and/or end.

▶ **Definition 12** (Action). Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a planning game and let $SV = SV_C \cup SV_E$. An *action* α for G is a term of the form $\mathsf{start}(x, v)$ or $\mathsf{end}(x, v)$, where $x \in SV$ and $v \in V_x$. The set of actions is partitioned into the set A_C of *Charlie*'s actions and the set A_E of *Eve*'s actions. An action belongs to A_C (resp., A_E) if it is of the form $\mathsf{start}(x, v)$, for some $x \in SV_C$ (resp., $x \in SV_E$) and $v \in V_x$, or of the form $\mathsf{end}(x, v)$, for some $x \in SV$, $v \in V_x$, and $\gamma_x(v) = \mathsf{c}$ (resp., $\gamma_x(v) = \mathsf{u}$).

Definition 12 can be read as follows. When an action $\mathsf{start}(x, v)$ or $\mathsf{end}(x, v)$ is executed by a player, a token for the variable x, where $x = v$, respectively starts or ends. Ending a task (a token) and starting the next one are two different actions, even if, as it will be precisely stated later, the end of a token must be immediately followed by the start of the next one. Depending on who owns the variable and the involved value, each action can be executed only by a specific player. More precisely, players can start tokens for the variables that they own, and end the tokens that hold values that they control.

It is worth noticing that, in contrast to the original definition of timeline-based planning problems with uncertainty (Definition 7), Definition 12 admits cases where $x \in SV_E$ and $\gamma_x(v) = c$ for some $v \in V_x$, that is, cases where *Charlie* may control the duration of a variable that belongs to *Eve*. This situation is symmetrical to the more common one where *Eve* controls the duration of a variable that belongs to *Charlie*, and there is no need, in our setting, to impose any asymmetry.

Actions are combined into *moves* that can start and/or end multiple tokens at once.

▶ **Definition 13** (Move). Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a timeline-based planning game. A *move* μ is a term of the form $\mathsf{wait}(t)$ or $\mathsf{play}(t, A)$, where $t \in \mathbb{N}$ is the *timestamp* of the move, $A \subseteq \mathsf{A}_C$ or $A \subseteq \mathsf{A}_E$, and for each $x \in SV_C \cup SV_E$, *at most one* action in A involves x. The set of moves is partitioned into the set M_C of *Charlie*'s moves and the set M_E of *Eve*'s moves. A move μ belongs to M_C if it is either of the form $\mu = \mathsf{wait}(t)$ or of the form $\mu = \mathsf{play}(t, A)$ and $A \subseteq \mathsf{A}_C$, while it belongs to M_E only if it is of the form $\mu_E = \mathsf{play}(t, A)$ and $A \subseteq \mathsf{A}_E$.

According to Definition 13, *Charlie* can either execute some set of actions A, by playing a $\mathsf{play}(t, A)$ move, or wait until some given time t, by playing $\mathsf{wait}(t)$. In contrast, *Eve* has a unique kind of move available, *i.e.*, $\mathsf{play}(t, A)$, which executes the actions in A at time t.

We are now ready to introduce the fundamental notion of round.

▶ **Definition 14** (Round). A *round* ρ is a pair of moves $(\mu_C, \mu_E) \in \mathsf{M}_C \times \mathsf{M}_E$. Let Π be a partial plan. A round $\rho = (\mu_C, \mu_E)$ is *applicable* to Π if the following conditions are met.
1. Integrity conditions:
   a. any action of the form $\mathsf{start}(x, v)$ is executed by either μ_C or μ_E if and only if $\mathsf{T}_x \in \Pi$ is a closed timeline;

    b. for any action of the form $\mathsf{end}(x, v)$, executed by μ_C or μ_E, either an action $\mathsf{start}(x, v)$ is being executed by either moves, or $\mathsf{T}_x \in \Pi$ is an open timeline $\mathsf{T}_x = \langle \tau_1, \ldots, \tau_k \rangle$ and its open token is $\tau_k = (x, v)$.

2. Timing conditions:

    a. if $\mu_C = \mathsf{play}(t_C, A_C)$ and $\mu_E = \mathsf{play}(t_E, A_E)$, then $t_C = t_E = \mathsf{now}(\Pi)$;

    b. if $\mu_C = \mathsf{wait}(t_C)$ and $\mu_E = \mathsf{play}(t_E, A_E)$, then $t_C > \mathsf{now}(\Pi)$ and $t_E \le t_C$.

A single move from either player is *applicable* to Π if there exists at least one move from the other player such that the round combining the two moves is applicable to Π.

The conditions that make a round applicable can be interpreted as follows. Item 1 ensures that the actions played by each move of the round are consistent with the current state of the timelines in the partial plan. In particular, a start action has to follow a end one, and start actions cannot be played on timelines that are already open. Note that both $\mathsf{start}(x, v)$ and $\mathsf{end}(x, v)$ for the same x and v can be played at the same round, possibly by two different players, provided that the timeline was previously closed. In this case, the move builds a token of unitary length (see Definition 15). Item 2 constrains the timings of the moves of a round: if *Charlie* does not wait, then he has to play immediately, that is, $t_C = \mathsf{now}(\Pi)$; otherwise, he can wait until an arbitrary future time point t_C. In both cases, *Eve* must play at a timestamp $t_E \le t_C$. This restriction has the following meaning. The advancement of time during the game is determined mostly by *Charlie*, who can make it advance one step at the time, by playing at each round, or skip some time steps at once without playing anything, by waiting. In both cases, *Eve*'s moves must specify what the environment is doing, if anything, in the meantime, hence the requirement that $t_E \le t_C$. If $t_E < t_C$, then time advances only up to t_E, so that at the next round *Charlie* can timely reply to *Eve*'s move.

The next definition specifies the effects of players' moves.

▶ **Definition 15** (Outcome of a round). Let Π be a partial plan and $\rho = (\mu_C, \mu_E)$ be an applicable round. The *outcome* of the application of ρ to Π is a partial plan $\rho(\Pi)$, which is obtained from Π by applying the following ordered sequence of steps:

1. for each $\mu \in \{\mu_C, \mu_E\}$, if $\mu = \mathsf{play}(t, A)$, then, for any $\mathsf{start}(x, v) \in A$, an open token $\tau = (x, v)$ is appended to T_x;

2. for each $\mu \in \{\mu_C, \mu_E\}$, if $\mu = \mathsf{play}(t, A)$, then, for any $\mathsf{end}(x, v) \in A$, the last open token $\tau = (x, v)$ of T_x (possibly added at the previous step) is replaced by a closed token $\tau' = (x, v, d)$, where $d = t - \mathsf{start\text{-}time}(\tau) + 1$;

3. $\mathsf{now}(\rho(\Pi)) = \min(t_C, t_E) + 1$, where either $\mu_C = \mathsf{play}(t_C, A_C)$ or $\mu_C = \mathsf{wait}(t_C)$, and $\mu_E = \mathsf{play}(t_E, A_E)$.

The effects of start and end actions are defined in Items 1 and 2, respectively. The steps are intended to be applied one after the other in order to handle the case where both $\mathsf{start}(x, v)$ and $\mathsf{end}(x, v)$ are played in the same round. Time advances according to Item 3, depending on the timestamps of the moves. Note that if no wait move is played, the new current time corresponds to the horizon of the resulting partial plan.

A play of a planning game is just a sequence of rounds applied to the empty plan $\Pi_\varnothing$.

▶ **Definition 16** (Play). Let G be a planning game and Π_0 be a partial plan (we call it an initial partial plan). A *play* for G from Π_0 is a sequence $\overline{\rho} = \langle \rho_0, \ldots, \rho_k \rangle$ of rounds such that ρ_0 is applicable to Π_0 and ρ_{i+1} is applicable to $\Pi_{i+1} = \rho_i(\Pi_i)$, for $1 \le i < k$.

Let $\overline{\rho} = \langle \rho_0, \ldots, \rho_k \rangle$ be a play and Π be a partial plan. We denote by $\overline{\rho}(\Pi) = \rho_k(\ldots \rho_0(\Pi))$ the *outcome* of $\overline{\rho}$ applied to Π. Where the initial partial plan is not mentioned, it is understood that the play is applied to $\Pi_\varnothing$. Each non-empty partial plan Π can be *closed* to form a

scheduled plan Π' (see Definition 4) by closing all the open tokens of open timelines at time $\mathsf{now}(\Pi)$. In the following, when the context is clear, we will interchangeably speak of a partial plan as a scheduled plan by implicitly referring to its closure. It can be easily checked that rounds as specified in Definition 13 suffice to build any possible scheduled plan over the game variables, *i.e.*, for any partial plan Π, there exists a play $\overline{\rho}$ such that $\Pi = \overline{\rho}(\Pi_{\varnothing})$.

▶ **Definition 17** (Strategy for *Charlie*). A *strategy for Charlie* is a function $\sigma_C : \mathbf{\Pi} \to \mathsf{M}_C$ that maps any given partial plan Π to a move μ_C applicable to Π.

▶ **Definition 18** (Strategy for *Eve*). A *strategy for Eve* is a function $\sigma_E : \mathbf{\Pi} \times \mathsf{M}_C \to \mathsf{M}_E$ that, given a partial plan Π and *Charlie*'s move μ_C applicable to Π, returns the next *Eve*'s move μ_E such that $\rho = (\mu_C, \mu_E)$ is applicable to Π.

A play $\overline{\rho}$ is said to be *played according to* a strategy σ_C for *Charlie*, starting from some initial partial plan Π_0, if $\rho_i = (\sigma_C(\Pi_{i-1}), \mu_E^i)$, for some μ_E^i, for all $0 < i < |\overline{\rho}|$, and to be played according to a strategy σ_E for *Eve* if $\rho_i = (\mu_C^i, \sigma_E(\Pi_{i-1}, \mu_C^i))$, for all $0 < i < |\overline{\rho}|$. For any pair of strategies (σ_C, σ_E) and any $k \geq 0$, there is a unique run $\overline{\rho}_k(\sigma_C, \sigma_E)$ of length k played according to σ_C and σ_E.

It is worth to note that, according to our definition of strategy, *Charlie* can base his decisions only on the previous rounds of the game, not including *Eve*'s move at the current round. Together with the fact that time strictly increases of at least one time unit at each round, this implies that *Charlie* has to wait at least one time unit to react to a move by *Eve*. This models the realistic assumption that the *sense-reason-react* loop of the controller needs a finite amount of time to be executed, and is coherent with the semantics of dynamic control strategies of flexible plans from [9].

Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a planning game. We define two timeline-based planning problems, $P_{\mathcal{D}} = (SV, \mathcal{D})$ and $P_G = (SV, \mathcal{D} \cup \mathcal{S})$, for G. A partial plan is *admissible* if it is a solution plan for $P_{\mathcal{D}}$, and *successful* if it is a solution plan for P_G. Similarly a play $\overline{\rho}$ is admissible or successful if its outcome $\overline{\rho}(\Pi_{\varnothing})$ is, respectively, admissible or successful.

▶ **Definition 19** (Admissible strategy for *Eve*). A strategy σ_E for *Eve* is *admissible* if for each strategy σ_C for *Charlie*, there is a $k \geq 0$ such that the play $\overline{\rho}_k(\sigma_C, \sigma_E)$ is admissible.

▶ **Definition 20** (Winning strategy for *Charlie*). Let σ_C be a strategy for *Charlie*. We say that σ_C is a *winning strategy* for *Charlie* if for any *admissible* strategy σ_E for *Eve*, there exists an $n \geq 0$ such that the play $\overline{\rho}_n(\sigma_C, \sigma_E)$ is successful.

We say that *Charlie wins* the game G if he has a winning strategy, while *Eve wins* the game if such a strategy does not exist. Let us consider again the problem described in Section 3 to show a simple winning strategy. The problem can be viewed as a game with the rules in Figure 1 belonging to $\mathcal{S}$ and $\mathcal{D}$ empty. After playing $\mathsf{start}(x, v_1)$ at the beginning, *Charlie* only has to wait for *Eve* to play $\mathsf{end}(x, v_1)$, and then play $\mathsf{start}(x, v_2)$ or $\mathsf{start}(x, v_3)$ according to the current timestamp.

A more involved example can be obtained by considering two variables $x \in SV_C$ and $y \in SV_E$, with $V_x = V_y = \{go, stop\}$, unit duration, and rules as follows.

$$\mathcal{S} = \left\{ \begin{array}{c} a[x = stop] \longrightarrow \exists b[y = stop] \ . \ b \leq^{\mathsf{e,s}}_{[0,0]} a \\ \top \longrightarrow \exists a[x = stop] \ . \ \top \end{array} \right\}$$

$$\mathcal{D} = \{ \qquad \top \longrightarrow \exists a[y = stop] \ . \ \top \qquad \}$$

Here, *Charlie*'s ultimate goal is to realise $x = stop$, but this can only happen after *Eve* realised $y = stop$. This is guaranteed to happen, since we consider only admissible strategies. Hence, the winning strategy for *Charlie* only chooses $x = go$ until *Eve* chooses $y = stop$, and

then wins by executing $x = stop$. If $\mathcal{D}$ was instead empty, a winning strategy would not exist since a strategy that never chooses $y = stop$ would be admissible. This would therefore be a case where *Charlie* looses because *Eve* can indefinitely postpone his victory.

Let us compare now the concept of *dynamic controllability* of flexible plans, as defined in [9], with the existence of winning strategies for timeline-based planning games, and show the greater generality of the latter concept. Proofs are omitted because of space concerns, but are available in an extended report [17].

The first step is to back the claim of this greater generality. Indeed, it can be shown that for any timeline-based planning problem with uncertainty $P = (SV_C, SV_E, S, \mathcal{O})$ there is an *associated timeline-based planning game* $G_P = (SV_C, SV_E, \mathcal{D}, \mathcal{S})$, such that a dynamically controllable flexible solution plan for P gives us a winning strategy for $\mathcal{G}$. To this aim, we need a way to represent as a game any given planning problem with uncertainty. Intuitively, this can be done by encoding the observations $\mathcal{O}$ into suitable domain rules. The game associated with a problem therefore mimics the exact setting described by the problem. What follows shows that such a game does indeed exist, and that there is a close relationship between its winning strategies and the dynamically controllable flexible plans for the problem.

▶ **Theorem 21.** *Let P be a timeline-based planning problem with uncertainty. If P admits a* dynamically controllable flexible solution plan*, then Charlie has a winning strategy for the associated timeline-based planning game G_P.*

Theorem 21 shows that the proposed framework can represent any timeline-based planning problem with uncertainty, and that the notion of winning strategy for a game subsumes that of dynamically controllable flexible plan. Moreover, it can be seen that the game setting is strictly more expressive, *i.e.*, there are cases where no dynamically controllable flexible plans exist, but a winning strategy can be found. This is the case with the example problem discussed in Section 3, which has an easy winning strategy when seen as a game, while it has no dynamically controllable flexible plan. Therefore, one can prove the following theorem.

▶ **Theorem 22.** *There exists a timeline-based planning problem with uncertainty $P = (SV_C, SV_E, S, \mathcal{O})$ such that there are no dynamically controllable flexible plans for P, but Charlie has a winning strategy for the associated planning game G_P.*
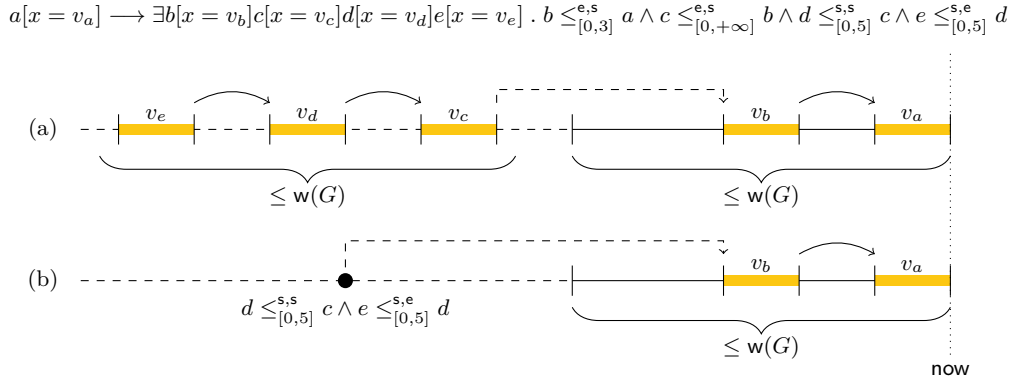
5 Decidability and complexity

A natural question about timeline-based planning games is whether a winning strategy for a given game can effectively be found. In this section, we positively answer the question, showing that a winning strategy, if it exists, can be found with an algorithm that runs in doubly exponential space. Proofs are omitted because of space concerns, but are available in an extended report [17].

The proposed algorithm makes use of the concept of *concurrent game structure* (CGS) [2], in particular, of the specific subclass of *turn-based synchronous game structures* (simply *game structures*, from now on), that, in the case of two players only, can be defined as follows.

▶ **Definition 23** (Turn-based synchronous game structure [2]). A *turn-based synchronous game structure* is a tuple $S = (Q_1, Q_2, M_1, M_2, \mathcal{P}, \pi, \delta_C, \delta_E)$, where:
1. Q_1 and Q_2 are the finite sets of *states* belonging to Player 1 and Player 2, respectively;
2. M_1 and M_2 are the finite sets of *moves* available to Player 2 and Player 2, respectively;
3. $\mathcal{P}$ is a finite set of *proposition letters*;
4. $\pi : Q_1 \cup Q_2 \to 2^{\mathcal{P}}$ provides the set of proposition letters that hold at any given state;
5. $\delta_1 : Q_1 \times M_1 \times Q_2$ and $\delta_2 : Q_2 \times M_2 \times Q_1$ are the *transition relations*, that specify the states reachable from a given one by applying a move by either player.

$$a[x = v_a] \longrightarrow \exists b[x = v_b]c[x = v_c]d[x = v_d]e[x = v_e] \;.\; b \leq^{\mathsf{e,s}}_{[0,3]} a \wedge c \leq^{\mathsf{e,s}}_{[0,+\infty]} b \wedge d \leq^{\mathsf{s,s}}_{[0,5]} c \wedge e \leq^{\mathsf{s,e}}_{[0,5]} d$$

Figure 3 How to represent the distant history of a play of a game G in a compact way: (a) the complete description of the timeline; and (b) the compact description of the distant history.

We proceed in two steps. We first provide a suitable encoding of a timeline-based planning game G into a corresponding game structure, and then we exploit existing machinery for ATL^* model checking [2] to find a winning strategy for G.

Let us focus on the first step. To encode a timeline-based planning game into a game structure, one may think of interpreting partial plans as states, marking with some proposition letter d those which satisfy the domain rules and with w those where the system rules are satisfied. However, such an encoding can, in principle, lead to an infinite state space, as the number of possible partial plans is infinite: the set of moves available to each player is infinite, as they specify a timestamp; moreover, at any given time, a rule triggered by one of the current tokens may require to look arbitrarily back in the past in order to check (the possibility of) its satisfaction. We now show how to constrain the state space to be finite.

First of all, we observe that arbitrarily large timestamps can only be introduced by wait moves played by *Charlie*, as the timestamp of any play move played by *Charlie* is forced to be equal to now and the timestamp of any *Eve*'s move is bounded by the one of a *Charlie*'s move. Now, even though wait moves are useful for *Charlie* to skip a given amount of time without executing a sequence of empty play moves, they can always be replaced by such a sequence, and thus it can be easily shown that if a winning strategy exists, another one exists which does not use any wait move. Hence, w.l.o.g., we can safely restrict ourselves to a *finite* set of moves of form $\mathsf{play}(t, A)$, with $t = \mathsf{now}$.

Let us focus now on the partial plans to show that not every single detail of their distant past has to be remembered. Consider a timeline-based planning game $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$, and let the *window* of G, $\mathsf{w}(G)$, be the product of all the non-zero lower and upper bounds appearing in any bounded atom of any rule, and all the non-zero lower and upper bounds on the length of tokens of any state variable. The window of the game gives a coarse upper bound on how far from a given point a chain of bounded atoms can look, which is a fundamental parameter of any timeline-based planning problem, *e.g.*, it has been exploited in [16] to study the complexity of the plan existence problem.

For the sake of clarification, consider the above half of Figure 3, which shows a timeline satisfying a particular synchronisation rule. The satisfaction of bounded and unbounded atoms of the rule by tokens are represented by solid and dashed arrows, respectively. The depicted tokens can be partitioned in two blocks, which satisfy two different groups of bounded atoms. Once the first group has been witnessed, only its existence need to be remembered, not the precise position or length of its tokens, since any bounded atom involving

the trigger cannot be affected by anything farther than the window $\mathsf{w}(G)$. Hence, while the events happening at most $\mathsf{w}(G)$ time steps before now need to be remembered in detail, the satisfaction of the other group of atoms can be remembered symbolically as shown in the bottom half of Figure 3, *i.e.*, by remembering the existence of the tokens satisfying each required group of atoms, and their relative ordering. The same has to be done to remember future requests triggered by tokens appeared in the past.

It can be easily checked that $\mathsf{w}(G) \in \mathcal{O}(2^{|G|})$, and that the number of bits needed to store the succinct representation of the history and of future requests is exponential as well. Hence, we can build a data structure of size at most *exponential* in the size of G, that, during the whole play, can be used to represent the current state and the past history of the game.

Let Π be a partial plan. We denote by $[\Pi]$ its succinct representation, and by $[\Pi_G]$ the set of the succinct representations of all the possible partial plans on G. For any round ρ, we can easily specify its application to the succinct representation of a partial plan in such a way that $\rho([\Pi]) = [\rho(\Pi)]$. Similarly, given $[\Pi]$, we can easily check whether Π is a solution plan for a given timeline-based planning problem.

We are now ready to define the *game structure* $\mathcal{G}$ associated with a planning game G.

▶ **Definition 24** (Game structure for a planning game). Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a timeline-based planning game. The corresponding game structure $\mathcal{G}$ is a tuple $(Q_C, Q_E, M_C, M_E, \mathcal{P}, \pi, \delta_C, \delta_E)$, where:
1. $Q_C = [\Pi_G]$ and $Q_E \subseteq [\Pi_G] \times \mathsf{M}_C$;
2. $M_C = \mathsf{M}_C$ and $M_E = \mathsf{M}_E$;
3. $\mathcal{P} = \{d, w\}$, where d and w are proposition letters,
4. $\pi(q_E) = \varnothing$, for any $q_E \in Q_E$, and $d \in \pi([\Pi])$ (resp., $w \in \pi([\Pi])$ if and only if Π is a solution plan for $P_{\mathcal{D}}$ (resp., P_G);
5. $([\Pi], \mu_C, ([\Pi], \mu_C)) \in \delta_C$ if and only if $\mu_C = \mathsf{play}(t, A)$ is applicable to Π and $t = \mathsf{now}$;
6. $(([\Pi], \mu_C), \mu_E, [\Pi']) \in \delta_E$ if and only if $\rho = (\mu_C, \mu_E)$ is applicable to Π and $\rho([\Pi]) = [\Pi']$.

The winning condition for *Charlie* (see Definition 20) can be expressed by means of the following ATL^* formula, where Player 1 is interpreted as *Charlie* and Player 2 as *Eve*:[4]

$$\varphi \equiv \langle\!\langle 1 \rangle\!\rangle [\![2]\!] ([\![1]\!] \, \mathsf{F} \, d \rightarrow \mathsf{F}(d \wedge w))$$

The formula asks for the existence of a strategy σ_C for *Charlie* ($\langle\!\langle 1 \rangle\!\rangle$) such that, for all strategies σ_E for *Eve* ($[\![2]\!]$), if σ_E is admissible ($[\![1]\!] \, \mathsf{F} \, d$), *i.e.*, for all *Charlie*'s strategies, there is a future point where d holds, then there is future point where $d \wedge w$ holds ($\mathsf{F}(d \wedge w)$), *i.e.*, *Charlie* wins. By applying known ATL^* model-checking algorithms on the game structure encoding the planning game and the above formula, one may solve the problem.

It can be proved that any strategy satisfying the above ATL^* formula corresponds to a winning strategy for *Charlie* on the original timeline-based planning game. The proposed compact representation of partial plans has exponential size, and thus the game structure has a doubly exponential number of states. Since ATL^* model-checking of fixed-size formulae over game structures is PTIME-complete [2], the following theorem holds.

▶ **Theorem 25.** *Let G be a timeline-based planning game. The problem of establishing whether Charlie has a winning strategy on G belongs to* $\mathsf{2EXPSPACE}$.

---

[4] ATL^* is a branching-time temporal logic similar to CTL^*, that allows one to quantify over the strategies of the different players of a game structure and to differentiate between paths played according to those strategies.

6   Conclusions and future work

This paper defines *timeline-based planning games*, a novel game-theoretic approach to timeline-based planning with uncertainty. Unlike current formulations based on dynamic controllability of flexible plans, the proposed one can uniformly deal with both temporal uncertainty and general nondeterminism, and it is strictly more expressive. We showed that a winning strategy can be found in doubly exponential space. Whether there exists a matching complexity lower bound, how to synthesise a finite-state machine implementing such a strategy, and how hard the synthesis problem is are still open issues.

References

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

2   R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002.

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

4   M. Beetz and D. McDermott. Improving robot plans during their execution. In *Proc. of the International Conference on Artificial Intelligence Planning Systems (AIPS)*, 1994.

5   S. Borgo, A. Cesta, A. Orlandini, and A. Umbrico. A Planning-Based Architecture for a Reconfigurable Manufacturing System. In *Proc. of the 26th International Conference on Automated Planning and Scheduling*, pages 358–366, 2016.

6   A. Camacho, E. Triantafillou, C. Muise, J. A. Baier, and S. A. McIlraith. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *Proc. of the 31st AAAI Conference on Artificial Intelligence*, 2017.

7   A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *Proc. of the 21st Conference on Innovative Applications of Artificial Intelligence*, pages 66–71, 2009.

8   M. Cialdea Mayer and A. Orlandini. An Executable Semantics of Flexible Plans in Terms of Timed Game Automata. In *Proc. of the 22nd International Symposium on Temporal Representation and Reasoning*, pages 160–169, 2015.

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

10   A. Cimatti, M. Do, A. Micheli, M. Roveri, and D. E. Smith. Strong temporal planning with uncontrollable durations. *Artificial Intelligence*, 256:1–34, 2018.

11   A. Cimatti, A. Micheli, and M. Roveri. Timelines with Temporal Uncertainty. In *Proc. the 27th AAAI Conference on Artificial Intelligence*, 2013.

12   A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1):35–84, 2003.

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

14   J. Frank. What is a timeline? In *Proc. of the 4th Workshop on Knowledge Engineering for Planning and Scheduling*, pages 31–38, 2013.

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

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

17   N. Gigante, A. Montanari, M. Cialdea Mayer, A. Orlandini, and M. Reynolds. A game-theoretic approach to timeline-based planning with uncertainty. *CoRR*, abs/1807.04837, 2018. URL: `http://arxiv.org/abs/1807.04837`.

18   L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

19   C. Muise, S. A. McIlraith, and J. C. Beck. Improved non-deterministic planning by exploiting state relevance. In *Proc. of the 22nd International Conference on Automated Planning and Scheduling*, 2012.

20   C. Muise, S.A. McIlraith, and V. Belle. Non-deterministic planning with conditional effects. In *Proc. of the 24th International Conference on Automated Planning and Scheduling*, 2014.

21   N. Muscettola. HSTS: Integrating Planning and Scheduling. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 6, pages 169–212. Morgan Kaufmann, 1994.

22   F. Patrizi, N. Lipovetzky, and H. Geffner. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence*, 2014.

23   S. Steel. Action under uncertainty. *Journal of Logic and Computation*, 4(5):767–795, 1994.

24   A. Umbrico, A. Cesta, M. Cialdea Mayer, and A. Orlandini. 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*, pages 498–512, 2017.

25   T. Vidal and H. Fargier. Handling Contingency in Temporal Constraint Networks: from Consistency to Controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1):23–45, 1999.