On timeline-based games and their complexity

Nicola Gigante, Angelo Montanari

University of Udine, Italy

Andrea Orlandini

CNR-ISTC, Rome, Italy

Marta Cialdea Mayer

University of Roma Tre, Rome, Italy

Mark Reynolds

The University of Western Australia

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 real-world domains. 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 2EXPTIME-complete.

Keywords: Timeline-based planning, Complexity, Games, Planning under uncertainty

1. Introduction

Timeline-based planning is an approach to planning originally proposed in the context of planning and scheduling of space operations. The approach was outlined by Muscettola et al. [40], and deployed soon after in the HSTS system [39], used to schedule and control the operations of the *Hubble Space Telescope*.

Timeline-based planning follows a different modelling perspective, when compared to *action-based* planning paradigms *à la* STRIPS [26]. In the timeline paradigm, there is no explicit separation among states, actions, and goals; rather, the domain is modelled as a set of independent, but interacting, components, whose behaviour over time, described by the *timelines*, is governed by a set of temporal constraints, called *synchronisation rules*. The solution plan consists of a set of timelines describing a possible behaviour of the system's components that satisfies all the rules. This is a more declarative point of view than that of common action-based languages such as PDDL, since it is focused on *what* has or has not to happen, instead of on what the agent has to *do* to achieve a given goal. Furthermore, the modelling of the system can be subdivided among multiple knowledge engineers and domain experts, since the timelines of distinct components can be separately modelled, and the resulting models can better reflect the architecture of the combined system.

In the last decades, timeline-based planning has been adopted and deployed in a number of systems developed by space agencies on both sides of the Atlantic. Systems developed following this paradigm include EUROPA [44], EUROPA 2 [3, 5], and ASPEN [15], developed by NASA, and APSI-TRF, developed for the

European Space Agency [11, 25, 28]. These systems have been repeatedly employed for mid- to long-term mission planning [9, 10, 16, 17, 29, 41], but the approach was also used to handle *on-board* autonomy [16, 29, 41]. Recently, the timeline-based approach has been incarnated also in the PLATINUm system [47, 48], a general purpose framework which is being employed in cooperative robotics [13] and assistive robotics tasks [14].

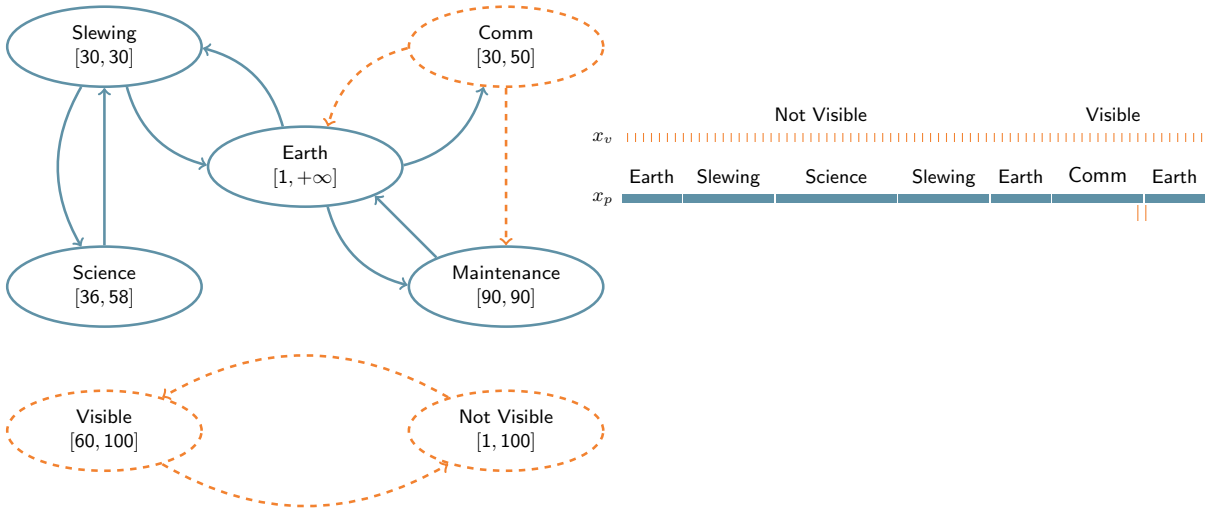
One of the flagship features of timeline-based systems, which makes them particularly suited for such domains, is the ability to integrate the planning phase with the *execution* of the plan. Timeline-based planning domains often model *real-time* systems, whose constraints heavily depend on the precise timing of execution of the tasks. However, ensuring precise timing is often not possible, because of the inherent *temporal uncertainty* that arises in the interaction with the environment. The controller executing the plan can handle temporal uncertainty by the use of *flexible plans*, *i.e.*, sets of different plans that differ in the execution time of the tasks.

Despite the practical success of the approach deployed in many complex real-world problems, little work has been done on timeline-based planning from a foundational perspective, till very recently. The concept of timelines and the main features of the paradigm have been characterised by different authors [6, 8, 21, 27]. A formal framework capturing the concept of timeline-based planning, including aspects regarding uncertainty and controllability issues of plans, has been defined a few years ago by Cialdea Mayer et al. [19]. Building on this framework, recent work explored the timeline-based paradigm from a formal perspective, comparing the expressiveness of timeline-based and action-based modeling languages and studying the computational complexity of the involved planning problems [23, 24, 31, 32]. Gigante [30] provides a comprehensive reference on the recent work on the formal properties of timeline-based planning.

Such a recent work was initially confined to timeline-based planning problems where no uncertainty about the interaction with the environment was considered. In this paper, we add uncertainty back into the picture. Timeline-based planning systems generally focus on handling *temporal uncertainty*, but *nondeterminism* is not supported in general terms: even the behaviour of external variables, completely controlled by the environment, is known up to temporal uncertainty only. The choice on concentrating on temporal reasoning and temporal uncertainty only is coherent with the history and scope of timeline-based systems. However, timeline-based modelling languages are expressive enough to model complex scenarios, such as those faced in cooperative robotics applications [13], that involve non-temporal nondeterminism, such as uncertainty about the task the environment will perform at a certain point in time. In such cases, current systems often employ a *re-planning* stage as part of their execution cycle (see, *e.g.*, [46]): 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.

To address such limitations, this paper introduces the novel concept of *timeline-based planning game*, a game-theoretic generalisation of the timeline-based planning problem with uncertainty, which uniformly deals with both temporal uncertainty and general nondeterminism: 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 [19]), but do admit winning strategies when viewed as instances of timeline-based games. Then, we prove that establishing the existence of a winning strategy for a given timeline-based planning game is 2EXPTIME-complete.

The paper is structured as follows. Section 2 defines timeline-based planning, including the concepts of flexible plan and dynamic controllability, borrowed from the formal framework provided by Cialdea Mayer et al. [19], which forms the basis of our analysis. Then, Section 3 discusses in detail a few limitations of the current approach based on flexible plans, motivating the rest of the paper. Section 4 addresses these issues by defining *timeline-based games*, and proving that the existence of a winning strategy for such games subsumes, and is strictly more general than, the existence of dynamically controllable flexible plans. Finally, Section 5 shows that the problem of establishing whether such a strategy exists is 2EXPTIME-complete. Section 6



(a) Example state variables.

(b) Example timelines.

Figure 1: On the left, values of the example state variables x_p (above) and x_v (below) visualised as state machines. Uncontrollable values are marked in orange. On the right, example timelines for the two variables.

concludes the paper by summarising its results and discussing future lines of research and open problems.

2. Timeline-based planning

In this section, we formally define timeline-based planning problems. At first (Section 2.1), we introduce problems that are not concerned with any uncertainty in the interaction with the external environment. Such an ingredient will be added in Section 2.2, by means of the notion of flexible plan.

2.1. Timelines, plans, synchronization rules, and timeline-based planning problems

In our setting, interesting properties of the modelled system are represented by *state variables*.

Definition 1 (State variable). A *state variable* is a tuple $x = (V_x, T_x, D_x, \gamma_x)$, where V_x is the *finite domain* of x, $T_x : V_x \to 2^{V_x}$ is the *value transition function* of x, $D_x : V_x \to \mathbb{N} \times \mathbb{N} \cup \{+\infty\}$ is the *duration function* of x, and $\gamma_x : V_x \to \{\mathsf{c}, \mathsf{u}\}$ is the *controllability tag*.

Definition 1 can be interpreted as follows. The transition function specifies which values can follow any other during the evolution of the variable. The duration function D_x maps any value $v \in V_x$ into a pair of non-negative integers $(d_{min}^{x=v}, d_{max}^{x=v})$, which respectively specify the minimum and maximum duration of any time interval (more precisely, of any *token*, as defined below) where $x = v$. The maximum duration can be infinite ($d_{max}^{x=v} = +\infty$), in which case there is no upper bound to how long the variable can hold the given value. The controllability tag comes into play when handling *uncertainty*. Intuitively, it states whether the *duration* of any token where $x = v$ is controllable by the system ($\gamma_x(v) = \mathsf{c}$) or not ($\gamma_x(v) = u$).

Two example state variables, that belong to the domain of the operations of a satellite orbiting a planet (a scenario elicited from the ESA *Mars Express* mission [12]), are depicted in Fig. 1. The first variable (x_p) represents the pointing mode of the satellite, *i.e.*, whether it is pointing towards Earth, doing maintenance, doing scientific measurements, slewing between the direction facing Earth and the direction facing the underlying planet, or whether it is transmitting some information. The domain of the variable thus consists of the five depicted values, and the transition function states which task can follow each other, being visualisable as a state machine. Minimum and maximum duration of each value are reported inside the bubbles. The

second variable (x_v) represents the visibility window of the Earth ground station, which determines when the station is visible for transmitting. In this example, $\gamma_{x_p}(\mathsf{Comm}) = \mathsf{u}$, *i.e.*, the Comm value is *uncontrollable*, meaning that the system can decide when to start communicating but can neither decide nor predict how much time will be required by the transmission. All the values of the variable x_v are uncontrollable since, of course, the satellite cannot decide when the ground stations are visible or not. In the rest of the section, the controllability tag will be ignored; it will be considered again in Section 2.2.

The evolution over time of the values of each state variable is modelled by the *timelines*, which are the core concept of the whole formalism.

Definition 2 (Timeline). A *token* for a state variable x is a triple $\tau = (x, v, d)$, where $v \in V_x$ is the value taken by the variable, and $d \in \mathbb{N}_+$ (positive integers) is the *duration* of the token. A *timeline* for a state variable x is a *finite* sequence $\overline{\tau} = \langle \tau_1, \ldots, \tau_k \rangle$ of tokens for x.

A timeline thus represents how a state variable changes its value over time in terms of a sequence of time intervals where the variable keeps the same value. Figure 1b shows two example timelines for the state variables x_p and x_v. Note that $d \in \mathbb{N}_+$, *i.e.*, the duration of tokens cannot be zero. For any token $\tau_i = (x, v_i, d_i)$ in a timeline $\overline{\tau} = \langle \tau_1, \ldots, \tau_k \rangle$, we define $\text{val}(\tau_i) = v_i$ and $\text{duration}(\tau_i) = d_i$. The functions $\text{start-time}(\overline{\tau}, i) = \sum_{j=1}^{i-1} d_j$ and $\text{end-time}(\overline{\tau}, i) = \text{start-time}(\overline{\tau}, i) + d_i$, for all $1 \leq i \leq k$, map each token τ_i to the corresponding $[\text{start-time}(\overline{\tau}, i), \text{end-time}(\overline{\tau}, i))$ time interval (right endpoint excluded). When there is no ambiguity about which timeline we refer to, we write $\text{start-time}(\tau_i)$ and $\text{end-time}(\tau_i)$ to denote, respectively, $\text{start-time}(\overline{\tau}, i)$ and $\text{end-time}(\overline{\tau}, i)$. Note that two consecutive tokens can hold the same value.

Let us denote by $\mathbb{T}_{\mathsf{SV}}$ the set of all the possible timelines for the set of variables in SV. A *plan* is a set of timelines describing the evolution of the considered set of state variables.

Definition 3 (Plan). Let SV be a set of state variables. A *plan* over SV is a function $\pi : \mathsf{SV} \to \mathbb{T}_{\mathsf{SV}}$, that maps each variable to the timeline describing its behaviour, such that all the timelines have the same total duration.

We denote by $\text{tokens}(\pi)$ the set of tokens in $\pi(\mathsf{SV})$ and by $\text{tokens}(\pi(x_i))$ the set of tokens in $\pi(x_i)$. The behaviour of a system over time, described by a set of state variables, is governed by a set of temporal constraints, called *synchronisation rules*. Let $\mathcal{N} = \{a, b, \ldots\}$ be an arbitrary set of *token names*. The building blocks of synchronisation rules are atomic temporal relations.

Definition 4 (Atomic temporal relation). An *atomic temporal relation* (*atom* for short) over $\mathcal{N}$ is defined by the following grammar:

$$\langle term \rangle := t \mid \text{start}(a) \mid \text{end}(a)$$
$$\langle atom \rangle := \langle term \rangle \leq_{[l,u]} \langle term \rangle$$

where $t, l \in \mathbb{N}$, $u \in \mathbb{N} \cup \{+\infty\}$, and $a \in \mathcal{N}$. Terms of the form t, with $t \in \mathbb{N}$, are called *timestamps*.

Notice that the only meaningful atoms are those where at least one of the two terms is not a timestamp.

Let a and b be two token names. Examples of atoms are $\text{start}(b) \leq 5$, $\text{start}(a) \leq_{[3,7]} \text{end}(b)$, and $\text{start}(a) \leq_{[0,+\infty]} \text{start}(b)$. Intuitively, a token name a refers to a specific token in a timeline, and $\text{start}(a)$ and $\text{end}(a)$ to its endpoints. Then, an atom such as $\text{start}(a) \leq_{[l,u]} \text{end}(b)$ constrains a to start before the end of b, and the distance between the two endpoints to be comprised between the lower and upper bounds l and u. Atoms are grouped into quantified clauses called *existential statements*.

Definition 5 (Existential statement). Given a set SV of state variables, an *existential statement* over SV is a statement of the following form:

$$\langle ex.\ statement \rangle := \langle quantifier\ prefix \rangle\ .\ \langle clause \rangle$$
$$\langle quantifier\ prefix \rangle := \exists a_1[x_1 = v_1]a_2[x_2 = v_2] \ldots a_n[x_n = v_n]$$
$$\langle clause \rangle := \langle atom \rangle \wedge \langle atom \rangle \wedge \ldots \wedge \langle atom \rangle$$

where $n \in \mathbb{N}$, $a_1, \ldots, a_n \in \mathcal{N}$, $x_1, \ldots, x_n \in \mathsf{SV}$, and $v_i \in V_{x_i}$ for all $1 \leq i \leq n$.

The blocks of the form $a_i[x_i = v_i]$ are called *quantifiers*. All the token names appearing in the atoms inside the clause $\mathcal{C}$ of an existential statement $\mathcal{E} \equiv \exists a_1[x_1 = v_1] \dots a_n[x_n = v_n] \,.\, \mathcal{C}$, that do not appear in the quantifier prefix, are said to be *free* in $\mathcal{E}$, and all those that do appear are said to be *bound*. An existential statement is *closed* if it does not contain free token names. Note that the quantifier prefix may as well be empty.

The syntax of *synchronisation rules* is defined as follows.

Definition 6 (Synchronisation rules). Given a set of state variables SV, a *synchronisation rule* over SV is an expression matching the following grammar:

$$\begin{aligned} \langle body \rangle &:= \ \langle ex.\ statement \rangle \vee \dots \vee \langle ex.\ statement \rangle \\ \langle rule \rangle &:= a_0[x_0 = v_0] \rightarrow \langle body \rangle \\ \langle rule \rangle &:= \top \rightarrow \langle body \rangle \end{aligned}$$

where $a_0 \in \mathcal{N}$, $x_0 \in \mathsf{SV}$, $v_0 \in V_{x_0}$, and the only token name appearing free in the body is a_0, and only in rules of the first form.

In rules of the first form, the quantifier in the head is called *trigger*; rules of the second form are called *trigger-less rules*. Intuitively, a synchronisation rule requires that, whenever a token exists that satisfies the trigger, at least one of the disjuncts (existential statements) is satisfied, *i.e.*, there exist other tokens, as specified in the quantifier prefix, such that the corresponding clause is satisfied. Trigger-less rules have a trivial universal quantification, which means that they only ask for the existence of some tokens, as specified by the existential statements. Consider, for instance, the timelines in Fig. 1b, and the synchronisation rules:

$$\begin{aligned} a[x_p = \mathsf{Comm}] &\rightarrow \exists b[x_v = \mathsf{Visible}] \,.\, \mathrm{start}(b) \leq \mathrm{start}(a) \wedge \mathrm{end}(a) \leq \mathrm{end}(b) \\ a[x_p = \mathsf{Science}] &\rightarrow \exists b[x_p = \mathsf{Comm}] \,.\, \mathrm{end}(a) \leq_{[0,50]} \mathrm{start}(b) \end{aligned}$$

The first rule expresses an essential guarantee for the satellite system represented by the two example variables, namely, that when the spacecraft is communicating with Earth, the ground station is visible. The timelines in Fig. 1b satisfy this constraint, since the time interval corresponding to the execution of the token where $x_p = \mathsf{Comm}$ is contained in the one of the token where $x_v = \mathsf{Visible}$. The second rule instructs the system to transmit data back to Earth after every measurement session, within a certain time bound.

A trigger-less rule can instead be used to state the goal of the system, namely, to perform some scientific measurement at all:

$$\top \rightarrow a[x_p = \mathsf{Science}]$$

Some simple syntactic sugar can be introduced on top of the basic syntax. A strict version of unbounded atoms can be added by writing $T < T'$ for $T \leq_{[1,+\infty]} T'$. Moreover, one can force two endpoints to coincide in time by writing $\mathrm{start}(a) = \mathrm{start}(b)$ for $\mathrm{start}(a) \leq_{[0,0]} \mathrm{start}(b)$, and two tokens to coincide by writing $a = b$ for $\mathrm{start}(a) = \mathrm{start}(b) \wedge \mathrm{end}(a) = \mathrm{end}(b)$. More generally, all the Allen's interval relations [1] can be expressed in terms of these basic temporal relations.

We conclude the section by providing the notion of *timeline-based planning problem*. As a preliminary step, we give a formal account of the semantics of synchronisation rules, to back up their intuitive meaning.

Definition 7 (Semantics of atomic temporal relations). An *atomic evaluation* is a function $\lambda : \mathcal{N} \rightarrow \mathbb{N}^2$ that maps each token name a to a pair $\lambda(a) = (s, e)$ of natural numbers. Given a term T and an atomic evaluation λ, the *evaluation* of T induced by λ, denoted $[\![T]\!]_\lambda$, is defined as follows:

- $[\![t]\!]_\lambda = t$ for any $t \in \mathbb{N}$;
- for any $a \in \mathcal{N}$, if $\lambda(a) = (s, e)$, then $[\![\mathrm{start}(a)]\!]_\lambda = s$ and $[\![\mathrm{end}(a)]\!]_\lambda = e$.

Given an atomic temporal relation $\alpha \equiv T \leq_{[l,u]} T'$ and an atomic evaluation λ, we say that λ *satisfies* α, written $\lambda \models \alpha$, if and only if $l \leq [\![T']\!]_\lambda - [\![T]\!]_\lambda \leq u$.

Given a *clause* $\mathcal{C} \equiv \alpha_1 \wedge \dots \wedge \alpha_k$, by extension we write $\lambda \models \mathcal{C}$ if $\lambda \models \alpha_i$ for all $1 \leq i \leq k$. Atomic evaluations are extracted from tokens when trying to satisfy a whole existential statement.

Definition 8 (Semantics of existential statements). Let π be a plan over a set of state variables SV, and let $\mathcal{E} \equiv \exists a_1[x_1 = v_1] \dots a_n[x_n = v_n]\ .\ \mathcal{C}$ be an existential statement. A function $\eta : \mathcal{N} \to \text{tokens}(\pi)$ mapping any token name to a token belonging to the plan π is called *token mapping*.

We say that π *satisfies* $\mathcal{E}$ with the token mapping η, written $\pi \models_\eta \mathcal{E}$, if, for all $1 \leq i \leq n$, there is a token $\tau_i \in \text{tokens}(\pi(x_i))$ such that $\eta(a_i) = \tau_i$ and $\text{val}(\tau_i) = v_i$, and $\lambda \models \mathcal{C}$ for an atomic evaluation λ such that $\lambda(a_i) = (\text{start-time}(\tau_i), \text{end-time}(\tau_i))$ for all $1 \leq i \leq n$.

Given a synchronisation rule $\mathcal{R} \equiv a_0[x_0 = v_0] \to \mathcal{E}_1 \vee \dots \vee \mathcal{E}_m$, we say that π *satisfies* $\mathcal{R}$, written $\pi \models \mathcal{R}$, if for any token $\tau_0 \in \pi(x_0)$, if $\text{val}(\tau_0) = v_0$, then there is an existential statement $\mathcal{E}_i$ and a token mapping η such that $\eta(a_0) = v_0$ and $\pi \models_\eta \mathcal{E}_i$. For a *trigger-less* rule $\mathcal{R} \equiv \top \to \mathcal{E}_1 \vee \dots \vee \mathcal{E}_m$, $\pi \models \mathcal{R}$ if there exist an existential statement $\mathcal{E}_i$ and a token mapping η such that $\pi \models_\eta \mathcal{E}_i$.

Definition 9 (Timeline-based planning problems). A *timeline-based planning problem* is a pair $P = (\mathsf{SV}, S)$, where SV is a set of state variables, and S is a set of synchronisation rules over SV. A plan π over SV is a *solution plan* for P if and only if $\pi \models \mathcal{R}$ for all synchronisation rules $\mathcal{R} \in S$.

Let P be a timeline-based planning problem. It has been shown that the problem of establishing whether there exists a plan $\pi \models P$, is EXPSPACE-complete [30, 32].

Definition 9 captures the *deterministic* variant of the problem, where there is no support for modelling the uncertainty coming from the interaction with the external world. The next section defines *timeline-based planning problems with uncertainty*, which account for this important feature.

2.2. Timeline-based planning with uncertainty

In this section, we extend the above definitions with the notion of *temporal uncertainty*, defining *timeline-based planning problems with uncertainty*. We basically follow the way in which they are presented in Cialdea Mayer et al. [19]. The ability of dealing with this form of uncertainty, integrating the planning and execution phases, is one of the key features of timeline-based planning systems, which usually employ the concept of *flexible timeline* and, consequently, of *flexible plan*.

A flexible timeline can be viewed as a set of timelines that differ only in the precise timings of the start and the end of the tokens therein, embodying some *temporal uncertainty* about the events described by the timeline.

Definition 10 (Flexible timeline). A *flexible token* for a state variable x is a tuple $\tau = (x, v, [e, E], [d, D])$, where $v \in V_x$, $[e, E] \in \mathbb{N} \times \mathbb{N}$ is the interval of possible *end times* of the token, and $[d, D] \in \mathbb{N} \times \mathbb{N}_+$ is the interval of possible token *durations*. A *flexible timeline* for x is a *finite* sequence $\overline{\tau} = \langle \tau_1, \dots, \tau_k \rangle$ of flexible tokens $\tau_i = (x, v_i, [e_i, E_i], [d_i, D_i])$ for x, where $[e_1, E_1] = [d_1, D_1]$, $e_i \geq e_{i-1} + d_i$, and $E_1 \leq E_{i-1} + D_i$.

Hence, flexible timelines provide an uncertainty range for the end time and the duration of each flexible token of the timeline. Note that each flexible token reports a range of its *end time*, rather than its start time, because in this way it can explicitly constrain its horizon. Tokens and timelines as specified in Definition 2 are also called *scheduled tokens* and *scheduled timelines*. Similarly to the notation used for scheduled timelines, the set of all the possible flexible timelines for the set of state variables SV is denoted as $\mathbb{F}_{\mathsf{SV}}$.

Let $x = (V_x, T_x, D_x, \gamma_x)$ be a state variable. We now focus on the *controllability tag* γ_x, which has been ignored in Section 2. The controllability tag tells, for each *value* of the domain of each variable, if the duration of tokens that hold the given value are under the control of the planner or not. Hence, a value $v \in V_x$ is said to be *controllable* if $\gamma_x(v) = \mathsf{c}$, and *uncontrollable* if, otherwise, $\gamma_x(v) = \mathsf{u}$.

Given a flexible timeline $\overline{\tau} = \langle \tau_1, \dots, \tau_k \rangle$, with $\tau_i = (x, v_i, [e_i, E_i], [d_i, D_i])$, a scheduled timeline $\overline{\tau}' = \langle \tau'_1, \dots, \tau'_k \rangle$, with $\tau_i = (x, v'_i, d'_i)$ is an *instance* of $\overline{\tau}$ if $d_i \leq d'_i \leq D_i$ and $e_i \leq \text{end-time}(\tau'_i) \leq E_i$. Figure 2 shows a flexible timeline for the example state variable x_p of Fig. 1, and one of its instances. We are now ready to define the concept of *flexible plan*.

Definition 11 (Flexible plan). Given a set of state variables SV, a *flexible plan* over SV is a pair $\Pi = (\pi, \mathsf{R})$, where $\pi : \mathsf{SV} \to \mathbb{F}_{\mathsf{SV}}$ is a function providing a flexible timeline $\pi(x)$ for each state variable x, and R is a set of *atoms* (Definition 4) using as token names the set of tokens of the timelines in π.

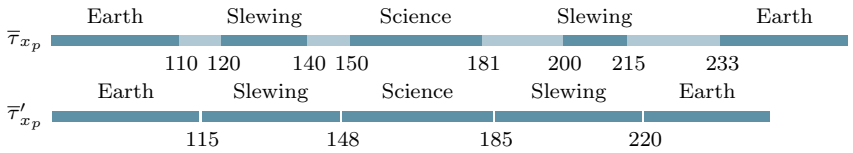


Figure 2: Example of flexible timeline $\overline{\tau}_{x_p}$ and one of its instances $\overline{\tau}'_{x_p}$, for the variable x_p of Fig. 1.

Intuitively, the flexible plan $\Pi = (\pi, \mathsf{R})$ represents a set of instances of the flexible timelines of π which, additionally, satisfy the constraints imposed by the atoms included in R.

Definition 12 (Instances of flexible plans). Let $\Pi = (\pi, \mathsf{R})$ be a flexible plan over SV. A plan π' is an *instance* of Π if $\pi'(x)$ is an instance of $\pi(x)$ for all $x \in \mathsf{SV}$, and all the atoms $T \in \mathsf{R}$ are satisfied by the atomic evaluation λ such that $\lambda(\tau') = (\text{start-time}(\tau'), \text{end-time}(\tau'))$ for all tokens τ' of $\pi'(x)$, for any $x \in \mathsf{SV}$.

To understand the role of the R component in Definition 11, consider the example given in Fig. 3, which shows flexible timelines $\overline{\tau}_x = \langle \tau_0^x, \tau_1^x, \tau_2^x \rangle$ and $\overline{\tau}_y = \langle \tau_0^y, \tau_1^y, \tau_2^y \rangle$ for two variables x and y, that have to be constrained by the shown synchronisation rule. The lower part of the picture shows some example instances of the flexible timelines. Given how the token τ_1^x is instantiated, not all the possible instances of the timeline for y are valid according to the considered rule. The first example instantiation, namely, $\overline{\tau}'_y$, violates the rule, while the second one satisfies it. This happens because a simple set of flexible timelines misses the key information that τ_1^y cannot start before the end of τ_1^x. A flexible plan satisfying such a rule would then have to provide additional constraints ensuring this fact, such as $\mathsf{R} = \{\text{end}(\tau_1^x) = \text{start}(\tau_1^y)\}$ or $\mathsf{R}' = \{\text{end}(\tau_1^x) \leq_{[5,10]} \text{start}(\tau_1^y)\}$.

We are now ready to introduce the timeline-based planning problem *with uncertainty*, as an extension of the timeline-based planning problem of Definition 9. We first provide the definition of the problem and of the *flexible solution plan*, and then discuss in detail their meaning and structure.

Definition 13 (Timeline-based planning problem with uncertainty).
A *timeline-based planning problem with uncertainty* is defined as a tuple $P = (\mathsf{SV}_C, \mathsf{SV}_E, S, O)$, where:

1. SV_C and SV_E are the sets of, respectively, the *controlled* and the *external* variables;
2. S is a set of synchronisation rules over $\mathsf{SV}_C \cup \mathsf{SV}_E$;
3. $O = (\pi_E, \mathsf{R}_E)$ is a flexible plan, called the *observation*, specifying the behaviour of external variables.

Definition 14 (Flexible solution plan). Let $P = (\mathsf{SV}_C, \mathsf{SV}_E, S, O)$, with $O = (\pi_E, \mathsf{R}_E)$, be a timeline-based planning problem with uncertainty. A *flexible solution plan* for P is a flexible plan $\Pi = (\pi, \mathsf{R})$ over $\mathsf{SV}_C \cup \mathsf{SV}_E$ such that:

1. Π agrees with O, *i.e.*, $\pi(x) = \pi_E(x)$, for each $x \in \mathsf{SV}_E$, and $\mathsf{R}_E \subseteq \mathsf{R}$;
2. the plan does not restrict the duration of uncontrollable tokens, *i.e.*, for any state variable x and any flexible token $\tau = (x, v, [e, E], [d, D])$ in $\pi(x)$, if $\gamma_x(v) = \mathsf{u}$, then $d = d_{min}^{x=v}$ and $D = d_{max}^{x=v}$;
3. any *instance* of π is a solution plan for the timeline-based planning problem $P' = (\mathsf{SV}_C \cup \mathsf{SV}_E, S)$, and there is at least one such instance.

The definitions above are worth a detailed explanation. The timeline-based planning problem with uncertainty considers two different sources of uncertainty: the behaviour of *external variables*, and the duration of *uncontrollable tokens*. In contrast to the simple problem without uncertainty (see Definition 9), the set of state variables is split into the *controlled variables* SV_C and the *external variables* SV_E. The behaviour of external variables cannot be constrained by the planner in any way, hence any solution plan is constrained to replicate the flexible timelines given by the *observation* O, which is a flexible plan describing

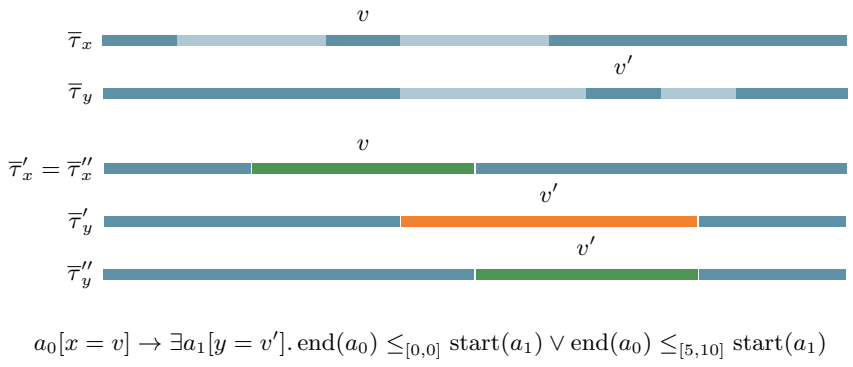


Figure 3: Example flexible timeline for two state variables x and y, where not all the possible instances satisfy the above synchronisation rule.

their behaviour. Since O is a *flexible* plan, there is temporal uncertainty on the start and end times of the involved tokens, but the behaviour of the variables is otherwise known beforehand to the planner. Despite the name, borrowed from Cialdea Mayer et al. [19], the *observation* O is more an *a priori* description of how the external variables will behave during the execution of the plan, up to the given temporal uncertainty on the precise timing of the events. The intended role of the external variables, then, is not so much that of independent components interacting with the planned system, but rather, of external entities useful to represent given facts and invariants that the planner has to account for during the search for a solution.

As an example, consider a satellite seeking the right time to transmit data to Earth. When modelling this scenario as a timeline-based planning problem with uncertainty, the window of visibility of Earth's ground stations can be represented as an external variable with a suitable *observation*. Note that the exact timing of when each station will effectively be available can be uncertain, but each visibility time slots are usually scheduled for the next months to come, and the planner has not to account for any variability in that regard. Hence, specifying the expected behavior of the environment as a flexible plan is usually enough when external variables are used in this way, not so much if a more general specification of the environment behavior is needed, as we will see in the next section.

The second considered source of temporal uncertainty comes from tokens holding *uncontrollable values*. The duration of such tokens cannot be decided by the planner, and thus their minimum and maximum duration in the flexibility range of the timeline has to coincide with that specified by the duration function of the variable. The planner can, however, decide which tokens to start and when, on *controlled* variables, even if $\gamma_x(v) = \mathsf{u}$ (the uncontrollability is specifically limited to the duration of the token). It is worth noting how the formalism has intentionally been tailored to consider only *temporal uncertainty*, both with regards to external variables and to uncontrollable tokens.

2.3. Controllability of flexible plans

As already pointed out, the timeline-based approach to planning is specifically targeted at the integration between the planning phase and the execution of the plan. Hence, it is important to ensure that, once a flexible plan is found for a timeline-based planning problem with uncertainty, the plan can be effectively executed. This is not a trivial requirement given the presence of uncontrollable tokens, whose duration is decided during execution and is unknown beforehand. Definition 11, indeed, ensures that any scheduled instance of the plan is a solution for the problem, but it does not guarantee that (1) such an instance exists for any possible choice of the duration of uncontrollable tokens, and (2) at any time during the execution, the correct choice to keep following an instance of the plan depends only on events already happened and information already known.

For this reason, we must take into consideration the *controllability* of flexible plans, *i.e.*, the property of being effectively executable by a controller. There are three major kinds of controllability that one may want to ensure on a flexible plan, depending on the application, which can be intuitively defined as follows.

Weak controllability For any possible choice of the duration of uncontrollable tokens, there is an instance of the flexible plan respecting that choice.

Strong controllability There is a way of instantiating controllable tokens that results into a valid instance of the flexible plan, no matter which is the duration of uncontrollable ones.

Dynamic controllability A strategy exists to choose how to instantiate each token, which, at any given point in time, can keep the execution in a valid instance of the plan, based only on past events.

Such concepts have been formalized by Cialdea Mayer et al. [19] for flexible plans, but ideas and terminology come from further back to the contributions on *simple temporal networks with uncertainty* (STNU) [50], which face very similar problems. In this paper, we are mostly concerned with *dynamic controllability*, as it represents the most reactive scenario, where the controller can react in real-time to what happens around it in order to achieve its goals or guarantee its safety requirements.

In the following, we briefly remind how dynamically controllable flexible plans can be formally defined. Recall that the set of tokens of all the timelines of a plan is denoted by tokens(π). We extend this notation by distinguishing among the set of tokens of all the timelines of a flexible plan Π, denoted by tokens(Π), the set of *uncontrollable* tokens of Π, denoted by $\text{tokens}_U(\Pi)$, and the set of *controllable* ones, denoted by $\text{tokens}_C(\Pi)$.

Definition 15 (Situations and relevant situations). Let $P = (\mathsf{SV}_C, \mathsf{SV}_E, S, O)$, with $O = (\pi_E, \mathsf{R}_E)$, be a timeline-based planning problem with uncertainty, and let $\Pi = (\pi, \mathsf{R})$ be a flexible plan. A *situation* for Π is a map $\omega : \text{tokens}_U(\Pi) \to \mathbb{N}$ assigning a duration to each uncontrollable token of Π. A situation ω is said to be *relevant* if any instance of Π in $\omega(\Pi)$ satisfies the constraints of R_E.

A situation represents the choices of the environment for the duration of uncontrollable tokens, both of controlled and external variables. Given a flexible plan $\Pi = (\pi, \mathsf{R})$, we denote by $\omega(\Pi)$ the set of instances of Π where the duration of uncontrollable tokens corresponds to what dictated by ω. *Relevant* situations are those where the external variables actually follow the behaviour described by the observation O. Since the controller is allowed to assume that this happens, only relevant situations are considered.

Let us denote by Ω_Π the set of *relevant* situations for Π. If situations represent the decisions of the environment about the duration of uncontrollable tokens, then *scheduling functions* define the controller's counterpart, deciding how to execute the whole plan.

Definition 16 (Scheduling function). Given a timeline-based planning problem $P = (\mathsf{SV}_C, \mathsf{SV}_E, S, O)$ and a flexible plan $\Pi = (\pi, \mathsf{R})$ for P, a *scheduling function* for Π is a map $\theta : \text{tokens}(\Pi) \to \mathbb{N}$, that assigns an *end time* to each token in Π, such that the resulting scheduled plan $\theta(\Pi)$ is an instance of Π.

Let $\mathcal{T}_\Pi$ be the set of scheduling functions for Π. An *execution strategy* for P is a map $\varsigma : \Omega_\Pi \to \mathcal{T}_\Pi$ such that, given $\theta = \varsigma(\omega)$ for any $\omega \in \Omega_\Pi$, if τ is an *uncontrollable* token of $\theta(\Pi)$, then $\text{duration}(\tau) = \omega(\tau)$.

We are now ready to formally define the different concepts of controllability introduced above. We start from the two simplest ones.

Definition 17 (Weak and strong controllability). Let Π be a flexible plan. Then, we say that Π is:

1. *weakly controllable* if there exists an execution strategy ς for Π;
2. *strongly controllable* if there is an execution strategy ς for Π such that $\varsigma(\omega) = \varsigma(\omega')$ for all $\omega, \omega' \in \Omega_\Pi$.

Given a scheduling function θ, let $\theta_{<t}$ be a function mapping any token τ such that $\theta(\tau) < t$ to its duration. Such a function can be viewed as a description of the evolution of the system up to time t, ignoring any token that does not end before it. By exploiting it, we can define *dynamic execution strategies*, and *dynamically controllable* flexible plans, *i.e.*, plans that admit such strategies.

Definition 18 (Dynamic controllability). Let Π be a flexible plan, ς be an execution strategy for Π, $\omega, \omega' \in \Omega_\Pi$ be two relevant situations, and $\tau \in \text{tokens}_C(\Pi)$ be a controllable token in Π. Moreover, let $\varsigma(\omega) = \theta$, $\varsigma(\omega') = \theta'$, and $t = \theta(\tau)$. Then,

1. ς is a *dynamic execution strategy* if $\theta_{<t} = \theta'_{<t}$ implies $\theta(\tau) = \theta'(\tau)$;
2. Π is *dynamically controllable* if it has a dynamic execution strategy.

3. Limitations of the current approach

In this section, we point out some limitations of the current approach to uncertainty in timeline-based planning, based on flexible plans, that we described in the previous section. The whole discussion revolves around the notion of *nondeterminism*. The design of most timeline-based planning systems, and, in particular, of the formal framework by Cialdea Mayer et al. [19], has been intentionally tailored to the handling of *temporal uncertainty*, *i.e.*, uncertainty about *when* things will happen, disregarding general forms of nondeterminism, *i.e.*, uncertainty about *what* will happen. According to the definitions given in Section 2.2, indeed, flexible plans are intrinsically sequential objects, that cannot represent any choice about how the execution of the plan can proceed if not regarding the timing of events (once more, this has been an intentional design choice of these systems).

In the meantime, the action-based planning community studied how to handle general nondeterminism quite extensively in the past years, following different approaches such as, for instance, reactive planning systems [4], deductive planning [45], model checking [20], and, especially, fully observable nondeterministic planning (FOND planning) [7, 37, 38, 42]. However, these approaches to nondeterministic action-based planning do not support flexible plans and temporal uncertainty, and do not account for controllability issues. Recently, SMT-based techniques have been exploited to deal with uncontrollable durations in strong temporal planning [22], but dynamic controllability issues are not addressed.

It seems therefore that the two worlds have evolved in different and incomparable ways. On the one hand, timeline-based planning supports temporal uncertainty, but it does not consider general nondeterminism; on the other hand, action-based planning deals with general nondeterminism, but it does not explicitly support temporal uncertainty.

As a matter of fact, it is worth observing that the explicit focus of timeline-based planning on temporal uncertainty does not mean that handling general nondeterminism is not needed in the common application scenarios of these systems. However, the history of timeline-based planning, with its roots in scheduling and control theory, naturally led over time to this formulation. As explained in Section 2.2, the external variables in timeline-based planning problems with uncertainty are used to express known facts about what will happen, rather than components of a fully-fledged external entity running alongside the planned system. To this end, planning problems include a flexible plan, the *observation*, describing the behaviour of external variables up to the given temporal flexibility. The definition of the various forms of controllability then assumes that the behaviour of the environment follows what is stated by the observation. This is perfectly fine in some scenarios, such as the satellite control example. In other ones, however, the approach can be limiting. As an example, in collaborative robotics domains where the PLATINUm planning system was designed to be deployed [47], the controlled system has to cooperate with human agents, and thus a true reactive behaviour is required and strong assumptions about the environment choices are not possible. To cope with application domains of this nature, many timeline-based systems employ a feedback loop between the planning and execution phases, which includes a *failure manager* that senses when the execution is deviating from the assumed observation, and triggers a *re-planning* phase if necessary, devising a new flexible plan and a dynamic execution strategy that can be used to resume execution. Unfortunately, the re-planning phase can be expensive to perform on-the-fly, limiting the real-time reactivity of the system.

Even ignoring the above issue, the relationships between temporal uncertainty, nondeterminism, and timeline-based planning languages turn out to be more complex than anticipated. As a matter of fact, even explicitly focusing on temporal uncertainty, timeline-based planning languages are still able to express scenarios where handling nondeterminism in a more general way is required. Consider, for instance, a timeline-based planning problem with uncertainty $P = (\mathsf{SV}_C, \mathsf{SV}_E, S, O)$, with a single controlled state variable $x \in \mathsf{SV}_C$, with $V_x = \{v_1, v_2, v_3\}$, $\mathsf{SV}_E = \varnothing$, and S consisting of the following rules:

$$\begin{aligned} a[x = v_1] \rightarrow \exists b[x = v_2] \,.\, \mathrm{end}(a) \leq_{[0,0]} \mathrm{start}(b) \wedge \mathrm{start}(a) \leq_{[0,5]} \mathrm{end}(a) \\ \vee \exists c[x = v_3] \,.\, \mathrm{end}(a) \leq_{[0,0]} \mathrm{start}(c) \wedge \mathrm{start}(a) \leq_{[6,10]} \mathrm{end}(a) \\ \top \rightarrow \exists a[x = v_1] \,.\, \mathrm{start}(a) = 0 \end{aligned}$$

Suppose that $D_x(v_i) = [1, 10]$ for all $v_i \in V_x$, and that tokens where $x = v_1$ are uncontrollable, *i.e.*,

$\gamma_x(v_1) = \mathsf{u}$, while $\gamma_x(v_2) = \gamma_x(v_3) = \mathsf{c}$. The rules require the controller to start the execution 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 value to assign to x on the second token must be chosen during the execution, but this is not possible because of the 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 above simple scenario shows that the inherently sequential nature of flexible plans does not allow one to express the need for a choice to be made during execution other than regarding the timings of events. However, the syntax of the language supports the modelling of scenarios where making qualitative choices depending on the environment nondeterministic behaviour is needed. Note that this is a different situation to that of *deterministic* action-based languages such as PDDL. In these languages, nondeterminism is not supported and simply cannot enter the picture. To allow one to model nondeterministic behaviours, PDDL has to be extended with syntactic elements useful for the purpose, like, *e.g.*, the `anyof` keyword for nondeterministic effects. In this case, instead, the basic syntax of the language is sufficient to express such scenarios, but their possible solutions cannot be represented. We may say that dynamically controllable flexible plans do not provide a *complete semantics* for timeline-based planning with uncertainty. One may suppose that this expressive power comes from *disjunctions* in synchronisation rules, which come into play the above example, but results such as the encoding of action-based temporal planning given by Gigante [30] show how their presence is essential even to express simple *deterministic* scenarios, and thus the gap cannot be filled by removing them.

It can be easily seen that scenarios like the above one would immediately arise when trying to encode any kind of nondeterministic action-based problem such as *fully observable nondeterministic* (FOND) planning problems. Hence, it is impossible to extend the aforementioned encoding of action-based temporal planning to nondeterministic planning. The notable observation, however, is that a syntactic representation of a FOND planning problem as a timeline-based planning problem would be perfectly feasible, similarly to the encoding for classical planning given in [30], but such an encoding would lack a proper semantics, corresponding to FOND *policies*, to express the solutions to the problem.

In this paper, we propose and systematically study an extension to timeline-based planning problems with uncertainty, called *timeline-based games*, which addresses both the issues outlined above by treating temporal uncertainty and general nondeterminism in a uniform way.

Timeline-based games are two-player turn-based perfect-information games where the players play by executing the start and end endpoints of tokens, building a set of timelines. The first player, representing the controller, wins the game if it can build a solution plan for a given timeline-based planning problem, independently from the behaviour of the second player, which represents the environment.

In the next section, we first define the structure of timeline-based games, and then we show that they can capture the semantics of timeline-based planning problems with uncertainty, in the sense that for any such problem there is a game where the controller has a winning strategy if and only if the problem admits a dynamically controllable flexible plan. Moreover, we demonstrate that they strictly subsume the approach based on flexible plans, by showing how the problematic example given above can be modelled by means of a timeline-based game that admits a winning strategy for the controller. Finally, we address the problem of finding a winning strategy for such games, showing that the problem of deciding whether the controller has a winning strategy for a given timeline-based planning game is 2EXPTIME-complete (Section 5). The decision procedure heavily exploits the machinery of *matching records* defined by Gigante [30].

4. Timeline-based games

This section introduces the *timeline-based games*, our game-theoretic approach to the handling of uncertainty in timeline-based planning. We first describe their general structure, including the winning

condition, and then go in detail on how they relate to dynamically controllable flexible plans and the issues brought up in the previous section.

Intuitively, a timeline-based game 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 3). At each round, each player makes a move deciding which tokens to start and/or to end and at which time. Both players are constrained by a set $\mathcal{D}$ of *domain* rules, which describe the basic rules governing the world. Domain rules replace the *observation* carried over by timeline-based planning problems with uncertainty (Definition 13), but generalise them allowing one to freely model the interaction between the system and the environment. Note that domain rules are not intended to be *Eve*'s (nor *Charlie*'s) *goals*, but, rather, a set of background facts about how the world works that can be assumed to hold at any time. Since no 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 a 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.

4.1. Partial plans

Players play the game by building a set of timelines, that is, a plan, in turns. Hence, we need to find a way to describe the partial result of this turn-based plan-building activity, that we call *partial plans*, which are incomplete plans under construction.

We start with the concept of *event sequence*, a different representation of a plan, easier to manipulate from a formal standpoint. Representing plans as event sequences is at the core of recent complexity results about timeline-based planning problems [30, 32]. In particular, we follow here the exposition given by Gigante [30].

In event sequences, instead of focusing on the single timelines as building blocks, plans are flattened over a single sequence of events that mark the start/end of tokens.

Definition 19 (Event sequence). Let SV be a set of state variables, and let $\mathcal{A}_{\mathsf{SV}}$ be the set of all the terms, called *actions*, of the form $\mathrm{start}(x, v)$ or $\mathrm{end}(x, v)$, where $x \in \mathsf{SV}$ and $v \in V_x$.

An *event sequence* over SV is a sequence $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$ of pairs $\mu_i = (A_i, \delta_i)$, called *events*, where $A_i \subseteq \mathcal{A}_{\mathsf{SV}}$ is a *non-empty* set of actions, and $\delta_i \in \mathbb{N}_+$, such that, for any $x \in \mathsf{SV}$:

1. for all $1 \leq i \leq n$, if $\mathrm{start}(x, v) \in A_i$ for some $v \in V_x$, then there are no $\mathrm{start}(x, v')$ in any μ_j before the closest μ_k with $k > i$, if any exists at all, such that $\mathrm{end}(x, v) \in A_k$;
2. for all $1 \leq i \leq n$, if $\mathrm{end}(x, v) \in A_i$ for some $v \in V_x$, then there are no $\mathrm{end}(x, v')$ in any μ_j after the closest μ_k with $k < i$, if any exists at all, such that $\mathrm{start}(x, v) \in A_k$;
3. for all $1 \leq i < n$, if $\mathrm{end}(x, v) \in A_i$ for some $v \in V_x$, then $\mathrm{start}(x, v') \in A_i$ for some $v' \in V_x$.
4. for all $1 < i \leq n$, if $\mathrm{start}(x, v) \in A_i$ for some $v \in V_x$, then $\mathrm{end}(x, v') \in A_i$ for some $v' \in V_x$.

Intuitively, an event $\mu_i = (A_i, \delta_i)$ consists of a set A_i of actions describing the start or the end of some tokens, happening δ_i time steps after the previous one. *Event sequences* collect events to describe a whole plan.

By Definition 19, a started token is not required to end before the end of the sequence, and a token can end without the corresponding starting action to ever have appeared before. In this case, the event sequence is said *open* for the variable x whose start/end event is missing. In event sequences where this does not happen, called *closed* event sequences, both the endpoints of all tokens are specified.

Definition 20 (Open and closed event sequences). An event sequence $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$ is *closed* on the right (left) for a variable x if for each $1 \leq i \leq n$, if $\mathrm{start}(x, v) \in A_i$ ($\mathrm{end}(x, v) \in A_i$), then there is $j > i$ ($j < i$) such that $\mathrm{end}(x, v) \in A_j$ ($\mathrm{start}(x, v) \in A_j$). Otherwise, $\overline{\mu}$ is *open* on the right (left) for x.

An event sequence is simply *open* or *closed* (to the right or to the left) if it is respectively open or closed (to the right or to the left) for any variable x. Note that the empty event sequence is closed on both sides for any variable. Moreover, on closed event sequences, the first event only contains $\text{start}(x, v)$ actions and the last event only contains $\text{end}(x, v)$ actions, and one for each variable x. Given an event sequence $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$ over a set of state variables SV, where $\mu_i = (A_i, \delta_i)$, we define $\delta(\overline{\mu}) = \sum_{1<i\leq n} \delta_i$, that is, $\delta(\overline{\mu})$ is the time passed between the start and the end of the sequence (its duration). The amount of time spanning a subsequence, written as $\delta_{i,j}$ when $\overline{\mu}$ is clear from context, is then $\delta(\overline{\mu}_{[i\ldots j]}) = \sum_{i<k\leq j} \delta_k$.

As a consequence of their definition, closed sequences can be directly mapped to plans.

Definition 21 (Correspondence between event sequences and plans). Let $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$ be a closed event sequence. Then, $\pi_{\overline{\mu}}$ is the *plan* where, for each $x \in \mathsf{SV}$, $\pi_{\overline{\mu}}(x) = \langle \tau_1, \ldots, \tau_k \rangle$ is a timeline such that $\text{start}(x, v) \in A_i$ ($\text{end}(x, v) \in A_i$) if and only if there is a τ_j such that $\text{val}(\tau_j) = v$ and $\text{start-time}(\tau_j) = \delta_{1,i}$ ($\text{end-time}(\tau_j) = \delta_{1,i}$).

In our context, we can assume *w.l.o.g.* that the rules in $\mathcal{S}$ and $\mathcal{D}$ do not use pointwise atoms [30]. In this way, we can forget about any absolute time reference and reason only in terms of distance between events. In mapping an event sequence to the plan it represents, the value of δ_1 of the first event $\mu_1 = (A_1, \delta_1)$ is ignored, since it would represent the time passed after a non-existent previous event. By fixing an arbitrary value for δ_1, the converse mapping from plans to event sequences can also be defined. Hence, we denote by $\overline{\mu}_\pi$ the event sequence such that $\pi_{\overline{\mu}_\pi} = \pi$. By admitting open event sequences, we can represent plans that are under construction, which was our original need.

Definition 22 (Partial plan). Let SV be a set of state variables. A *partial plan* over SV is an event sequence $\overline{\mu}$ over SV, *closed on the left*.

Partial plans can be either open or closed on the right depending on the particular moment of the game, but they are always closed on the left. Since there is no ambiguity, we will simply say *open* or *closed* to mean open or closed on the *right*.

4.2. The game arena

Let us start by defining the key notion of *timeline-based games*.

Definition 23 (Timeline-based game). A *timeline-based game* is a tuple $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$, 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 SV_C and SV_E.

A partial plan for G is a partial plan over the state variables $\mathsf{SV}_C \cup \mathsf{SV}_E$. Let $\mathbf{\Pi}_G$ be the set of all possible partial plans for G, or simply $\mathbf{\Pi}$ when there is no ambiguity. It is worth stressing again that the plan being built by the players, represented by the partial plan, is a *scheduled* plan, not a flexible one. The uncertainty is moved to the ignorance about what the next moves of *Eve* will be at each step. Recall that $\delta(\overline{\mu})$ denotes the duration of $\overline{\mu}$, that is, the distance in time between the last and the first events of the sequence, hence in our settings it can be interpreted as the time elapsed from the start of the game.

Since ε is a closed event sequence and $\delta(\varepsilon) = 0$, the *empty* partial plan ε is a good starting point for the game. Players incrementally build a partial plan, starting from ε, by playing actions that specify which tokens to start and/or end, producing an event that extends the event sequence, or complementing the already existing last event of the sequence. Recall from Definition 19 that actions are terms of the form $\text{start}(x, v)$ or $\text{end}(x, v)$, where $x \in \mathsf{SV}$ and $v \in V_x$, and that the set of possible actions over SV is denoted as $\mathcal{A}_{\mathsf{SV}}$, here just $\mathcal{A}$ for simplicity. Actions of the former kind are called *starting* actions, and those of the latter kind are called *ending* actions. Then, we partition all the available actions into those that are playable by either of the two players.

Definition 24 (Partition of player actions). The set $\mathcal{A}$ of available actions over the set of state variables $\mathsf{SV} = \mathsf{SV}_C \cup \mathsf{SV}_E$ is partitioned into the set $\mathcal{A}_C$ of *Charlie*'s actions, and the set $\mathcal{A}_E$ of *Eve*'s actions, defined as:

$$\mathcal{A}_C = \underbrace{\{\text{start}(x, v) \mid x \in \mathsf{SV}_C,\ v \in V_x\}}_{\text{start tokens on } \textit{Charlie}\text{'s timelines}} \cup \underbrace{\{\text{end}(x, v) \mid x \in \mathsf{SV},\ v \in V_x,\ \gamma_x(v) = \mathsf{c}\}}_{\text{end controllable tokens}}$$

$$\mathcal{A}_E = \underbrace{\{\text{start}(x,v) \mid x \in \mathsf{SV}_E,\ v \in V_x\}}_{\text{start tokens on } \textit{Eve}\text{'s timelines}} \cup \underbrace{\{\text{end}(x,v) \mid x \in \mathsf{SV},\ v \in V_x,\ \gamma_x(v) = \mathsf{u}\}}_{\text{end uncontrollable tokens}}$$

Hence, players can start tokens for the variables that they own, and end the tokens that hold values that they control. It is worth noting that, in contrast to the original definition of timeline-based planning problems with uncertainty (Definition 13), Definition 24 admits cases where $x \in \mathsf{SV}_E$ and $\gamma_x(v) = \mathsf{c}$ 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* (*i.e.*, uncontrollable tokens), and we have no need to impose any asymmetry.

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

Definition 25 (Moves). A *move* m_C for *Charlie* is a term of the form $\text{wait}(\delta_C)$ or $\text{play}(A_C)$, where $\delta_C \in \mathbb{N}$ and $\varnothing \neq A_C \subseteq \mathcal{A}_C$ is either a set of *starting* actions or a set of *ending* actions.

A *move* m_E for *Eve* is a term of the form $\text{play}(A_E)$ or $\text{play}(\delta_E, A_E)$, where $\delta_E \in \mathbb{N}$ and $A_E \subseteq \mathcal{A}_E$ is either a set of *starting* actions or a set of *ending* actions.

Two different aspects of the mechanics of the game influence the above definitions.

First, moves such as $\text{play}(A_C)$ and $\text{play}(\delta_E, A_E)$ can play either $\text{start}(x,v)$ actions only or $\text{end}(x,v)$ actions only. A move of the former kind is called a *starting* move, while a move of the latter kind is called an *ending* move. Note that empty moves $\text{play}(\delta_E, \varnothing)$ can be considered both starting or ending moves. Moreover, we consider wait moves as *ending* moves. In some sense, starting and ending moves have to be alternated during the game.

Second, the two players can play the two different sets of moves defined above, hence we denote as $\mathcal{M}_C$ the set of moves playable by *Charlie*, and as $\mathcal{M}_E$ the set of moves playable by *Eve*. *Charlie* can choose to play some actions to start/end a set of tokens, by playing a $\text{play}(A_C)$ move, or to do nothing and wait a certain amount of time by playing a $\text{wait}(\delta_C)$ move. *Charlie* plays first at each round, as will be formally stated later, and *Eve* can reply to *Charlie*'s move by playing a $\text{play}(A_E)$ move in response to a $\text{play}(A_C)$ move by *Charlie*, and a $\text{play}(\delta_E, A_E)$ move in response to a $\text{wait}(\delta_C)$ move by *Charlie*. If *Charlie* plays a $\text{play}(A_C)$ move, the given actions are applied *immediately*, for some specific sense defined later, and *Eve* replies by specifying what happens to her variables at the same time point. Instead, if *Charlie* plays a $\text{wait}(\delta_C)$ move to wait some amount of time δ_C, there is no reason why *Eve* should be forced to wait the same amount of time without doing nothing, so she can play a $\text{play}(\delta_E, A_E)$ move, specifying an amount of time $\delta_E \leq \delta_C$, so that actions in A_E will be applied accordingly, *interrupting* the wait of *Charlie* who can then timely reply to *Eve*'s actions. This is formalised by the following notion of *round*.

Definition 26 (Round). A *round* ρ is a pair $(\mathsf{m}_C, \mathsf{m}_E) \in \mathcal{M}_C \times \mathcal{M}_E$ of moves such that:

1. m_C and m_E are either both *starting* or both *ending* moves;
2. either $\rho = (\text{play}(A_C), \text{play}(A_E))$, or $\rho = (\text{wait}(\delta_C), \text{play}(\delta_E, A_E))$, with $\delta_E \leq \delta_C$;

A *starting* (*ending*) round is one made of starting (ending) moves. Note that since *Charlie* cannot play empty moves and wait moves are considered ending moves, each round is unambiguously either a starting or an ending round. We can now define how a round is applied to the current partial plan to obtain the new one.

Definition 27 (Outcome of rounds). Let $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$ be a partial plan, with $\mu_n = (A_n, \delta_n)$, let $\rho = (\mathsf{m}_C, \mathsf{m}_E)$ be a round, let δ_E and δ_C be the time increments of the moves, with $\delta_C = \delta_E = 1$ for $\text{play}(A)$ moves, and let A_E and A_C be the set of actions of the two moves (A_C is empty if m_C is a wait move).

The *outcome* of ρ on $\overline{\mu}$ is the event sequence $\rho(\overline{\mu})$ defined as follows:

1. if ρ is a starting round, then $\rho(\overline{\mu}) = \overline{\mu}_{<n}\mu'_n$, where $\mu'_n = (A_n \cup A_C \cup A_E, \delta_n)$;
2. if ρ is an ending round, then $\rho(\overline{\mu}) = \overline{\mu}\mu'$, where $\mu' = (A_C \cup A_E, \delta_E)$;

We say that ρ is *applicable* to $\overline{\mu}$ if:

a) the above construction is well-defined, *i.e.*, $\rho(\overline{\mu})$ is a valid event sequence by Definition 19;

b) ρ is an ending round if and only if $\overline{\mu}$ is open for all variables.

We say that a single move by either player is applicable to $\overline{\mu}$ if there is a move for the other player such that the resulting round is applicable to $\overline{\mu}$.

Together, Definitions 26 and 27 define the mechanics of the game, that can now be fully clarified. The game starts from the empty partial plan ε, and players play in turn, composing a round from the move of each one, which is applied to the current partial plan to obtain the new one. Let $\overline{\mu}$ be the current partial plan. At each step of the game, both players can either stop the execution of a set of tokens, by playing an ending round, or start the execution of a set of others, by playing a starting round (Item 1 of Definition 26). This does *not* mean that at each time point in the constructed plan only one of the two things can happen, but that the ending and starting actions of each events are contributed separately in two phases. When a starting round is played, its actions are added to the last event of the round (since no time amount needs to be specified, starting rounds can only consist of play(A) moves). In contrast, when an ending round is played, the corresponding actions form an event that is appended to $\overline{\mu}$, obtaining that $\delta(\rho(\overline{\mu})) > \delta(\overline{\mu})$. Then, the next round, which must be a starting round by Item b) of Definition 27, can start the new tokens following the ones that were just closed. Note that Items a) and b) of Definition 27 together ensure that a) the played actions make sense with regards to the current partial plan being built (such as the fact that a token can be closed only if it was open *etc.*, see Definition 19), and b) that time cannot stall, by forcing starting rounds to be immediately followed by ending ones.

4.3. The winning condition

It is now time to define the notion of *strategy* for each player, and of *winning strategy* for *Charlie*.

Definition 28 (Strategies). A *strategy for Charlie* is a function $\sigma_C : \mathbf{\Pi} \to \mathcal{M}_C$ that maps any given partial plan $\overline{\mu}$ to a move m_C applicable to $\overline{\mu}$. A *strategy for Eve* is a function $\sigma_E : \mathbf{\Pi} \times \mathcal{M}_C \to \mathcal{M}_E$ that maps a partial plan $\overline{\mu}$ and a move $\mathsf{m}_C \in \mathcal{M}_C$ applicable to $\overline{\mu}$, to a m_E such that $\rho = (\mathsf{m}_C, \mathsf{m}_E)$ is applicable to $\overline{\mu}$.

A sequence $\overline{\rho} = \langle \rho_0, \dots, \rho_n \rangle$ of rounds is called a *play* of the game. A play is said to be *played according to* some strategy σ_C for *Charlie*, if, starting from the initial partial plan $\overline{\mu}_0 = \varepsilon$, it holds that $\rho_i = (\sigma_C(\Pi_{i-1}), \mathsf{m}^i_E)$, for some m^i_E, for all $0 < i \leq n$, and to be played according to some strategy σ_E for *Eve* if $\rho_i = (\mathsf{m}^i_C, \sigma_E(\Pi_{i-1}, \mathsf{m}^i_C))$, for all $0 < i \leq n$. It can be seen that for any pair of strategies (σ_C, σ_E) and any $n \geq 0$, there is a unique run $\overline{\rho}_n(\sigma_C, \sigma_E)$ of length n played according both to σ_C and σ_E.

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. However, *Charlie* can still react *immediately*, in some sense, to decide which token to start after an uncontrollable one closed by *Eve*, because of the alternation between starting and ending rounds. Hence *Charlie* can choose the starting actions of an event depending on the ending actions of that same event, but the contrary is not true: after *Eve* closes a token, *Charlie* has to wait at least one time step to react to that move with an *ending* action. This design choice is crucial to replicate and capture the semantics of dynamically controllable flexible plans, as will be detailed in Section 4.4.

As for the winning condition, we have to formalise the intuition given at the beginning of the section, regarding the role of domain rules and system rules. *Charlie* wins if, *assuming* domain rules are respected, he manages to satisfy the system rules no matter how *Eve* plays.

Let $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$ be a planning game. To evaluate the satisfaction of the two sets of rules over the current partial plan, we proceed as follows. First, we define from G two timeline-based planning problems (as for Definition 9), $P_{\mathcal{D}} = (\mathsf{SV}, \mathcal{D})$ and $P_{\mathcal{S}} = (\mathsf{SV}, \mathcal{S})$. Then, given a partial plan $\overline{\mu}$, we consider the scheduled plan $\pi_{\overline{\mu}'}$ corresponding to an event sequence $\overline{\mu}'$ obtained by closing $\overline{\mu}$ at time $\delta(\overline{\mu})$, *i.e.*, completing the last event of $\overline{\mu}$ in such a way to close any open token. Then, we say that a partial plan $\overline{\mu}$, and the play $\overline{\rho}$ such that $\overline{\mu} = \overline{\rho}(\varepsilon)$, are *admissible*, if $\pi_{\overline{\mu}'} \models P_{\mathcal{D}}$, *i.e.*, if the partial plan satisfies the domain rules, and are *successful* if $\pi_{\overline{\mu}'} \models P_{\mathcal{S}}$, *i.e.*, if the partial plan satisfies the system rules.

Definition 29 (Admissible strategy for *Eve*). A strategy σ_E for *Eve* is *admissible* if for each strategy σ_C for *Charlie*, there is $k \geq 0$ such that the play $\overline{\rho}_k(\sigma_C, \sigma_E)$ is admissible.

Definition 30 (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 $n \geq 0$ 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 a winning strategy does not exist.

As an example, consider a timeline-based game $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$ with two variables $x \in \mathsf{SV}_C$ and $y \in \mathsf{SV}_E$, $V_x = V_y = \{go, stop\}$, unit duration, and the sets of rules defined as follows:

$$\mathcal{S} = \left\{ \begin{array}{c} a[x = stop] \rightarrow \exists b[y = stop] \,.\, \mathrm{end}(b) = \mathrm{start}(a) \\ \top \rightarrow \exists a[x = stop] \,.\, \top \end{array} \right\}$$

$$\mathcal{D} = \left\{ \top \rightarrow \exists a[y = stop] \,.\, \top \right\}$$

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* loses because *Eve* can indefinitely postpone his victory.

This is a simple example of a kind of problems and solutions that are not approachable with flexible plans. The next section will formalize and prove the greater generality of this approach. As pointed out in Section 3, the inherent sequentiality of flexible plans poses some limitations in interactive application scenarios, where a feeback loop involving a *re-planning* phase is often needed. In contrast, in timeline-based games, the interaction between the environment and the controlled system can be modeled in a rich and expressive way, by means of the domain rules. Note that domain rules can mention both controlled and external variables, allowing for the specification of complex dynamics and interactions. A winning strategy for such a game is then able to cope with the maximum generality to any execution of such a specification, without the need of any sort of re-planning (hence without the need to implement the planner itself as part of the executive system). Of course, as in any model-based approach, the modelling task is crucial, as a badly modeled game would result into a strategy unable to really react to the environment when executed in the real world. However, this is rather a problem of knowledge engineering and domain modeling: as far as the world is correctly modelled, the game-theoretic approach avoids the need of a run-time feedback loop involving any kind of re-planning phase.

Moreover, the high complexity of the strategy existence problem, proved in the next section, does not confute by itself any of the above claims: the search for a winning strategy and the synthesis of a controller implementing such a strategy are done off-line, while during the execution a blind execution of the strategy suffices. The costly re-planning phases, in contrast, takes place during execution, impairing the applicability of the approach to real-time scenarios.

4.4. Timeline-based games and flexible plans

Let us compare now the concept of *dynamic controllability* of flexible plans, as defined in [19], with the existence of winning strategies for timeline-based planning games.

The first step is to back the claim of the greater generality of the latter with respect to the former. We prove that, given a flexible solution plan for a timeline-based planning problem with uncertainty, we can reduce the problem of the dynamic controllability of the plan to the existence of a winning strategy for a particular game. To this aim, we need a way to represent as a game any given planning problem with uncertainty together with its flexible plan. 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 it. What follows shows how such a game is built and which relationship exists between its winning strategies and dynamically controllable flexible plans for the original problem.

Theorem 1 (Winning strategies vs. dynamic controllability)**.** *Let P be a timeline-based planning problem with uncertainty, and suppose that P admits a flexible solution plan Π. Then, a timeline-based game $G_{P,\Pi}$ can be built, in polynomial time, such that Π is dynamically controllable iff* Charlie *has a winning strategy for $G_{P,\Pi}$.*

Proof. Let $P = (\mathsf{SV}_C, \mathsf{SV}_E, S, O)$ be a timeline-based planning problem with uncertainty and let $\Pi = (\pi, \mathsf{R})$ be a flexible solution plan for P. We can build an equivalent timeline-based game $G_{P,\Pi} = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$, by keeping SV_C and SV_E unchanged, and suitably encoding the *observation* O and the flexible plan Π into, respectively, the set of domain rules $\mathcal{D}$ and of system rules $\mathcal{S}$. In this way, *Eve*'s behaviour will be constrained to follow what is dictated by the observation, replicating the semantics of timeline-based planning problems with uncertainty, and the behaviour of *Charlie* will follow by construction what is stated by the flexible plan.

To proceed, let $\mathsf{SV}_E = \{x_1, \ldots, x_n\}$, $O = (\pi_E, \mathsf{R}_E)$, and $\overline{\tau}_i = \pi_E(x_i) = \langle \tau_1^i, \ldots, \tau_{k_i}^i \rangle$, for some k_i and all $x_i \in \mathsf{SV}_E$, with $\tau_j^i = (x_i, v_j^i, [e_j^i, E_j^i], [d_j^i, D_j^i])$. Finally, let $\mathsf{R}_E = \{\alpha_1, \ldots, \alpha_m\}$. The set $\mathcal{D}$ can encode the whole observation by a single trigger-less rule stating that: (1) the tokens τ_j^i are required to exist, (2) their (a) position in the sequence, and (b) the end time and duration flexibility ranges correspond to the plan, and (3) the atoms in $\mathsf{R} = \{\alpha_1, \ldots, \alpha_{|\mathsf{R}|}\}$ are satisfied.

Such a rule can be written as follows:

$$\top \rightarrow \exists \tau_1^1[x_1 = v_1^1], \ldots, \exists \tau_{k_n}^n[x_n = v_{k_n}^n] \,. \tag{1}$$

$$\wedge \bigwedge_{\substack{1 \leq i \leq n \\ 1 \leq j < k_i}} \mathrm{end}(\tau_j^i) = \mathrm{start}(\tau_{j+1}^i) \tag{2a}$$

$$\wedge \bigwedge_{\substack{1 \leq i \leq n \\ 1 \leq j \leq k_i}} e_j^i \leq \mathrm{end}(\tau_j^i) \leq E_j^i \wedge d_j^i \leq \mathrm{duration}(\tau_j^i) \leq D_j^i \wedge \bigwedge_{1 \leq i \leq m} \alpha_i \tag{2b,3}$$

Adding the above rule to the set $\mathcal{D}$ of domain rules ensures that any admissible play of the game follows the observation O. In a completely similar way, we can encode the flexible plan Π into a rule to add to the system rules $\mathcal{S}$. Note that, by definition of flexible plan, following the plan satisfying R is sufficient to satisfy the set S of problem rules, which thus can be discarded and replaced by the single rule that encodes the plan. Now, we prove that *Charlie* has a winning strategy for $G_{P,\Pi}$ if and only if Π is dynamically controllable.

($\longrightarrow$). Suppose that there exists a dynamic execution strategy ς for Π. We show how to obtain a winning strategy σ for $G_{P,\Pi}$ by combining the flexible plan with the dynamic execution strategy. The strategy σ is built as follows. Let $\overline{\mu}$ be an event sequence.

1. If $\overline{\mu}$ is not open for all variables, then players have to produce a starting round. Hence, $\sigma(\overline{\mu}) = \mathrm{play}(A_C)$, where A_C contains an action $\mathrm{start}(x, v)$ for any variable x that is open in $\overline{\mu}$, and $v \in V_x$ is chosen following the flexible plan Π, as enforced by the system rules $\mathcal{S}$, which uniquely determine the sequence of tokens on the timeline of each $x \in \mathsf{SV}_C$.

2. If $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$ is open for all variables, then players have to produce an ending round. To decide the end time of the currently open tokens, we can mimic the dynamic execution strategy ς. Execution strategies map *relevant situations* (Definition 15) to scheduling functions (Definition 16). Situations describe the duration of all the uncontrollable tokens in the plan, and thus we cannot directly construct a situation from the current partial plan, since only the duration of tokens ended before $\delta(\overline{\mu})$ is known. However, a relevant situation ω can be obtained by choosing the duration of missing tokens arbitrarily, as long as the result projects an instance of the observation O. The winning strategy we are looking for can assume that Charlie is playing against an *admissible Eve* strategy, and hence the existence of such a relevant situation is guaranteed. Then, the resulting scheduling function $\theta = \varsigma(\omega)$ can be used to decide the next move. Among all the flexible tokens in Π whose instance is currently open in $\overline{\mu}$, let $\langle \tau_1 = (x_1, v_1, [e_1, E_1], [d_1, D_1]), \ldots, \tau_k = (x_k, v_k, [e_k, E_k], [d_k, D_k]) \rangle$ be those such that $t = \theta(\tau_i)$ is minimum. Then, if $t = \delta(\overline{\mu}) + 1$, the strategy plays the end of those tokens, *i.e.*, $\sigma(\overline{\mu}) = \mathrm{play}(A_C)$, where $A_C = \mathrm{end}(x_1, v_1), \ldots, \mathrm{end}(x_k, v_k)$. Otherwise, it is not yet time to end them, and thus $\sigma(\overline{\mu}) = \mathrm{wait}(t_\delta)$, with $t_\delta = t - \delta(\overline{\mu})$. Note that the arbitrary completion of the situation ω for future tokens was only a formal obligation: since ς is a dynamic execution strategy (Definition 18), the consequent choice only depended on the tokens ended before $\delta(\overline{\mu})$, anyway.

($\longleftarrow$). We now show that if a winning strategy σ_C for $G_{P,\Pi}$ exists, then there exists a dynamic execution strategy ς for Π, defined as follows. Let ω be a relevant situation. An *admissible* strategy σ_E for *Eve* is

induced by ω as follows. For variables $x \in \mathsf{SV}_E$, the strategy σ_E starts tokens in the order specified by O, and ends them with the timings specified by ω. Since ω is relevant, we are sure that a valid instance of O is obtained, hence σ_E is admissible, as it satisfies the domain rule in $\mathcal{D}$, that encodes O. Then, for variables $y \in \mathsf{SV}_C$, *if Charlie builds an instance of* Π, then σ_E ends the uncontrollable tokens of the plan according to ω, behaving arbitrarily otherwise. Note that $\mathcal{D}$ does not involve any variable in SV_C, and thus the behavior of the strategy on those variable does not affect its admissibility. Now, since σ_C is a winning strategy, and σ_E is admissible, there is a natural number k such that the play $\overline{\rho}_k$ of k rounds played according to σ_C and σ_E produces an event sequence $\overline{\mu} = \overline{\rho}_k(\varepsilon)$ that satisfies $\mathcal{D}$ and $\mathcal{S}$. Since $\mathcal{S}$ faithfully encodes the flexible plan Π, the plan $\pi_{\overline{\mu}}$ induced by $\overline{\mu}$ is an instance of Π. Hence, we can define $\varsigma(\omega)$ as the scheduling function θ such that, for each token τ in Π which ends at time t_τ in $\pi_{\overline{\mu}}$, $\theta(\tau) = t_\tau$. Since $\pi_{\overline{\mu}}$ is an instance of Π, θ is a scheduling function for Π. Moreover, θ is based on $\overline{\mu}$, which is the result of playing the strategy σ_C. By how the game is defined, at each ending round, σ_C has only access to the previous history of the game up to the *previous* time step. Hence, let τ be a token and $t = \theta(\tau)$. The decision by θ of ending τ at time t only depends on the prefix of $\overline{\mu}$ happening before t, and thus, given any other situation ω', with $\theta' = \varsigma(\omega')$, if $\theta_{<t} = \theta_{<t}$, we have that $\theta'(\tau) = t$ as well, by construction. This confirms that ς is a dynamic execution strategy for Π. □

Theorem 1 shows that given a flexible solution plan Π, we can decide its dynamic controllability by looking for a winning strategy for the game $G_{P,\Pi}$. More generally, given the timeline-based planning problem with uncertainty $P = (\mathsf{SV}_C, \mathsf{SV}_E, S, O)$, we can similarly build a game $G_P = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$ such that the existence of a dynamically controllable flexible plan for P implies the existence of a winning strategy for G_P. This is done by encoding the observation O into the set of domain rules $\mathcal{D}$ exactly as done in Theorem 1, but setting $\mathcal{S} = S$, without constraining the game to any specific plan. Then, if a plan exists, and it is dynamically controllable, it can be checked that a winning strategy for G_P must exist as well.

Corollary 1 (Generality of timeline-based games)**.** *Let P be a timeline-based planning problem with uncertainty. Then, a timeline-based game G_P can be built, in polynomial time, such that if P admits a dynamically controllable flexible solution plan, then* Charlie *has a winning strategy for G_P.*

The converse is not true, however, because winning strategies for timeline-based games are strictly more expressive than flexible plans. Hence, there can be some problems P that do *not* have any dynamically controllable flexible plan, but such that there is a winning strategy for G_P. 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. We can encode the example problem P with the game G_P, in which the shown synchronisation rules are included as system rules, and the set of domain rules is empty (since there are no external variables and thus the observation is empty as well). The winning strategy is simple: after playing $\mathrm{start}(x, v_1)$ at the beginning, *Charlie* only has to wait for *Eve* to play $\mathrm{end}(x, v_1)$, and then play $\mathrm{start}(x, v_2)$ or $\mathrm{start}(x, v_3)$ according to the current timestamp. Therefore, one can prove the following theorem.

Theorem 2. *A timeline-based planning problem with uncertainty P exists 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. Complexity of finding winning strategies

In previous sections, we introduced and formally defined the notion of *timeline-based game*, and showed how the existence of a winning strategy for such a game subsumes the existence of a dynamically controllable flexible plan for the equivalent timeline-based planning problem with uncertainty. In this section, we show that deciding whether such a strategy exists is a 2EXPTIME-complete problem.

5.1. Finite representation of game plays

From the definitions given in Section 4 and, in particular, the definitions of strategies for the two players (Definition 28), it can be seen that a timeline-based game provides an implicit representation for a potentially infinite state space consisting of all possible partial plans $\mathbf{\Pi}$. To solve the game, we first reduce the game to a *finite* game. The key observation here is that, although each synchronisation rule can potentially speak

about events arbitrarily far in the past and in the future, a finite representation of the history of the game is possible. The same issue has been met already in the study of the computational complexity of the plan existence problem for timeline-based planning problems [30, 32], leading to the development of a few conceptual tools that we are going to reuse here: a graph-theoretic representation of synchronisation rules, called the *rule graphs*, and a data structure, called *matching records*, that, by using rule graphs, can finitely represent an infinite set of similar partial plans.

What follows briefly recaps a minimal set of definitions that make it possible to understand how these concepts can be leveraged to obtain a finite state space for timeline-based games. The exposition is borrowed from [30], where many additional details can be found.

Definition 31 (Rule graphs). Let $\mathcal{E} \equiv \exists a_1[x_1 = v_1] \dots a_k[x_k = v_k]\ .\ \mathcal{C}$ be one of the existential statements of a synchronisation rule $\mathcal{R} \equiv a_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \dots \vee \mathcal{E}_m$.

Then, the *rule graph* of $\mathcal{E}$ is an edge-labelled graph $\mathcal{G}_\mathcal{E} = (V, E, \beta)$ where:

1. the set of nodes V is made of *terms* (as per Definition 4) such that:
   (a) $\text{start}(a) \in V$ or $\text{end}(a) \in V$ if and only if $a \in \{a_0, \dots, a_k\}$, for any $a \in \mathcal{N}$;
   (b) if the term T is used in $\mathcal{C}$, then $T \in V$;
2. $E \subseteq V \times V$ is the edge relation such that, for each pair of nodes $T, T' \in V$, there is an edge $(T, T') \in E$ if and only if $\mathcal{C}$ contains an atom of the form $T \leq_{[l,u]} T'$, or $T = \text{start}(a_i)$ and $T' = \text{end}(a_i)$ for some $0 \leq i \leq k$;
3. $\beta : E \rightarrow \mathbb{N} \times \mathbb{N}_{+\infty}$ is the edge-labelling function, such that for each $e \in E$, if e is associated with the atom $T \leq_{[l,u]} T'$ in $\mathcal{C}$, then $\beta(e) = (l, u)$.

Intuitively, the rule graph $\mathcal{G}_\mathcal{E}$ for an existential statement $\mathcal{E}$ has a node for each term that appears in $\mathcal{E}$, including both endpoints of each mentioned token, and an edge for each temporal constraint imposed between any two nodes. An edge e is said to be *unbounded*, if $\beta(e) = (l, +\infty)$ for some $l \in \mathbb{N}$, and *bounded* otherwise. Note that the nodes representing the token a_0 quantified in the *trigger* of the rule are included in the rule graph of all the existential statements of the rule. Given a rule graph $\mathcal{G} = (V, E)$ and an event sequence $\overline{\mu} = \langle \mu_1, \dots, \mu_n \rangle$, a *matching function* $\gamma : V \rightarrow [1, \dots, n]$ can be used to match the nodes of $\mathcal{G}$ to the events of $\overline{\mu}$. If a matching function γ exists such that all the temporal constraints are satisfied (with satisfaction defined in the standard way), written $\overline{\mu}, \gamma \models \mathcal{G}$, then we say that $\mathcal{G}$ *matches* over $\overline{\mu}$, written $\overline{\mu} \models \mathcal{G}$. If $\text{start}(a_0)$ appears in μ_i (a rule containing $\mathcal{E}$ would be triggered by μ_i), we write $\overline{\mu}, \gamma \models_i \mathcal{G}$ if $\gamma(\text{start}(a_0)) = i$, and $\overline{\mu} \models_i \mathcal{G}$ if there exists such a γ. In general, we can rephrase the satisfaction of synchronization rules in terms of matching of rule graphs.

We can thus reason about the satisfaction of the synchronization rules of the game in terms of matching of the rule graphs of their existential statements. A few useful observations can be made about rule graphs. Given a rule graph $\mathcal{G}_\mathcal{E} = (V, E, \beta)$, a *subgraph* of $\mathcal{G}$ is a graph $\mathcal{G}' = (V', E', \beta')$ such that $V' \subseteq V$, $E' \subseteq E$, and $\beta' = \beta|_{E'}$. A subgraph of a rule graph can be seen itself as the rule graph of a simpler existential statement. A particularly important kind of subgraphs are the *bounded components*: subgraphs that are connected by bounded edges. Given a bounded component B of $\mathcal{G}$, we can compute the maximum distance of any two events involved in the matching of B on any event sequence.

Proposition 1. *Let B be a bounded component of a rule graph $\mathcal{G} = (V, E, \beta)$, let γ be a matching function, and let $\overline{\mu} = \langle \mu_1, \dots, \mu_n \rangle$ be an event sequence such that $\mathcal{G}, \gamma \models \overline{\mu}$. Then, a positive quantity, denoted by* $\text{window}(B)$*, can be computed such that:*

1. *for any $T, T' \in V$, with $\gamma(T) = i$ and $\gamma(T') = j$, it holds that $\delta(\overline{\mu}_{[i \dots j]}) \leq \text{window}(B)$;*
2. $\text{window}(B) \in \mathcal{O}(2^{|\mathcal{G}|})$*, where $|\mathcal{G}|$ is the size of the representation of $\mathcal{G}$.*

A reasonable upper bound to window(B) can be given by the sum of all the upper bounds of the bounded edges of B. A much tighter bound can be computed as shown in [30]. Here, were are interested in highlighting that window(B) is at most exponential in the size of $\mathcal{G}$, since we consider numeric coefficients to be expressed in binary notation. Then, we can extend the concept to a generic set S of synchronization rules, by defining window(S) as the sum of window(B) for all the bounded components B of all the rule graphs of every rule in S. Note that window(S) $\in \mathcal{O}(2^{|S|})$, where $|S|$ is the size of S (defined in the natural way).

The definition of window(S) allows us recall a basic property of event sequences.

Proposition 2 (Bounded distance in event sequences [30])**.** *Let S be a set of synchronization rules over a set of state variables* SV*, and let $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$ be an event sequence satisfying S. Then, there exists another event sequence $\overline{\mu}' = \langle \mu'_1, \ldots, \mu'_m \rangle$, satisfying S, such that $\delta'_i \leq \text{window}(S)$ for all $i \geq 0$.*

Intuitively, the distance between two events of an event sequence does not need to exceed window(S) because no rule in the set has any way to discriminate two consecutive events so far in time.

The above-introduced concepts allow us to define the notion of *matching record*. Intuitively, given a set of synchronisation rules S and any event sequence $\overline{\mu}$, the matching record $[\overline{\mu}]$ of $\overline{\mu}$ is a structure of bounded size that allows us to effectively test whether $\overline{\mu}$ satisfies any of the rules in S. Furthermore, given an event μ, it is possible to effectively build the matching record $[\overline{\mu}\mu]$ starting from $[\overline{\mu}]$.

Definition 32 (Matching record). Let S be a set of synchronization rules over a set SV of state variables, and let $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$ be an event sequence over SV, closed to the left, such that $\delta(\overline{\mu}) \geq 2\,\text{window}(S)$.

The *matching record* of $\overline{\mu}$ is a tuple $[\overline{\mu}] = (\overline{\omega}, \Gamma, \Delta)$, where:

1. $\overline{\omega}$ is the shortest suffix $\overline{\mu}_{\geq h}$ of $\overline{\mu}$ that can be split into two subsequences spanning at least window(S) time steps, *i.e.*, $\overline{\omega} = \overline{\omega}_-\overline{\omega}_+$, where $\overline{\omega}_- = \overline{\mu}_{[h \ldots h_+ - 1]}$, $\overline{\omega}_+ = \overline{\mu}_{\geq h_+}$, and both $\delta(\overline{\omega}_-) \geq \text{window}(S)$ and $\delta(\overline{\omega}_+) \geq \text{window}(S)$;
2. Γ is a function that maps any *existential statement* $\mathcal{E}$ of any $\mathcal{R} \in S$ and $1 \leq k \leq |\overline{\omega}_-|$ to the *maximal* subgraph $\Gamma(\mathcal{E}, k)$ of $G_{\mathcal{E}}$ such that:
   (a) $\overline{\mu}_{\leq h+k}, \gamma \models \Gamma(\mathcal{E}, k)$ for some matching function γ,
   (b) $\Gamma(\mathcal{E}, k)$ does not contain the trigger node start(a_0),
   (c) any edge going out of $\Gamma(\mathcal{E}, k)$ is *unbounded*;
3. Δ is a function that maps an *existential statement* $\mathcal{E}$ of any $\mathcal{R} \in S$ and $1 \leq k \leq |\overline{\omega}_+|$ to the *maximal* subgraph $\Delta(\mathcal{E}, k)$ of $G_{\mathcal{E}}$ such that:
   (a) for each position t in $\overline{\mu}_{<h_+}$ where $\mathcal{R}$ is triggered, $\overline{\mu}_{\leq h_+ + k} \models_t \Delta(\mathcal{E}, k)$,
   (b) any edge going in or out from $\Delta(\mathcal{E}, k)$ is *unbounded*.

If, instead, $\overline{\mu}$ is empty, made of only one event, or $\delta(\overline{\mu}) < 2\,\text{window}(S)$, then $[\overline{\mu}] = \overline{\mu}$.

A detailed account of the above definition can be found in [30]. Here, we will briefly summarize the role of the components of a matching record $[\overline{\mu}] = (\omega, \Gamma, \Delta)$. The first component is a *suffix* ω of the actual event sequence $\overline{\mu}$, which is considered as composed of two parts, ω_- and ω_+, each spanning at least a window(S) amount of time. ω records the *recent history* of the sequence in an exact way. The rest of the sequence does not need to be stored completely, as the essential information about it is represented by Γ and Δ. In particular, Γ records which parts of the rule triggered inside ω_- match over $\overline{\mu}$, including those that matched in the past, before the recent history recorded by ω. For each newly triggered rule, Γ is queried to know which parts of the rule matched in the distant past. Then, Δ, records the parts that matched in each instance of the rule triggered in the whole $\overline{\mu}$. The missing parts will need to be satisfied in the future in order to fulfill the rule. In both cases, the subgraphs recorded are required to only have *unbounded* outgoing edges, or, in other words, to be only made of whole *bounded components*. Since ω_- and ω_+ span at least window(S), this ensures that quantitative constraints can be fully matched inside ω when building Γ and Δ.

The above definition is relatively abstract, but matching records can be nonethelesss computationally manipulated in useful way, as formally stated by the next proposition.

Proposition 3 (Matching records (see Gigante [30], Chapter 4))**.** *Let S be a set of synchronization rules over a set SV of state variables, and let $\overline{\mu}$ be an event sequence over SV. The following statements hold:*

1. *the size of $[\overline{\mu}]$ is at most* exponential *in the size of S;*
2. *given $[\overline{\mu}]$ and $\mathcal{R} \in S$, whether $\overline{\mu} \models \mathcal{R}$ can be decided in* exponential time*;*
3. *given $[\overline{\mu}]$ and an event μ, whether $\overline{\mu}\mu$ would be a valid event sequence can be checked in polynomial time;*
4. *given $[\overline{\mu}]$ and an event μ, the matching record $[\overline{\mu}\mu]$ can be effectively built in* exponential time*.*

5.2. Deciding the existence of winning strategies

We can use matching records to reduce the state space of our games to a finite size, that is, given a timeline-based game, we can build a structure representing a finite-state equivalent game. In particular, we can build a *turn-based synchronous game structure*, as introduced by Alur et al. [2].

Definition 33 (Turn-based synchronous game structure). A *turn-based synchronous game structure* is a tuple $S = \langle \mathcal{P}, Q, \Sigma, \nu, \lambda, R \rangle$, where $\mathcal{P} = \{1, \ldots, k\}$ is the set of players, Q is the finite set of *states*, Σ is the finite set of *propositions*, $\nu : Q \to 2^{\Sigma}$ specifies the set $\nu(q)$ of propositions true at any state $q \in Q$; $\lambda : Q \to \mathcal{P}$ is a function telling which player owns any given state, and $R \subseteq Q \times Q$ is the transition relation.

Turn-based synchronous game structures, simply called *game structures* hereinafter, represent games where players play in turn, not concurrently, since each state $q \in Q$ is owned by the player $\lambda(q)$, who plays when the game reaches one of its states. A path of the game is an infinite sequence of states $\overline{q} = \langle q_0, q_1, \ldots \rangle$ such that $(q_i, q_{i+1}) \in R$ for all $i \geq 0$. Given a player $a \in \mathcal{P}$, a *strategy* for a is a function $f_a : Q^+ \to Q$ that maps any non-empty finite prefix $\overline{q} = \langle q_0, \ldots, q_n \rangle$ of a path (the history of the game play), where $\lambda(q_n) = a$, to the next state $f_a(q_n)$ chosen among the successors of q_n. A play such that $q_{i+1} = f_a(q_i)$ for any q_i such that $\lambda(q_i) = a$ is said to be *played according to* the strategy f_a. Given a set of players $A \subseteq \mathcal{P}$, and a set of strategies F_A, one for each $a \in A$, the sequence $\overline{q}$ is played according to F_A if it is played according to all the strategies in F_A.

Let $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$ be a timeline-based game. If $\mathbf{\Pi}$ is the set of all the possible event sequences over $\mathcal{S} \cup \mathcal{D}$, let $[\mathbf{\Pi}]$ be the set of all the matching records over $\mathbf{\Pi}$. Note that, by Proposition 3, $[\mathbf{\Pi}]$ is a finite set. Hence, we can use $[\mathbf{\Pi}]$ to build the state space of a game structure representing G.

Definition 34 (Game structure of a timeline-based game). Let $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$ be a timeline-based game. The *turn-based asynchronous game structure* $S_G = \langle \mathcal{P}, Q, \Sigma, \nu, \lambda, R \rangle$ associated with G is defined as follows:

1. the set of players is $\mathcal{P} = \{1, 2\}$, where player 1 represents *Charlie* and player 2 represents *Eve*;
2. $Q \subseteq [\mathbf{\Pi}] \cup ([\mathbf{\Pi}] \times \mathcal{M}_C)$ is the set of states, partitioned into the set $Q_1 = [\mathbf{\Pi}]$ and the set $Q_2 \subseteq [\mathbf{\Pi}] \times \mathcal{M}_C$ of pairs $([\overline{\mu}], \mathsf{m}_C)$, where $[\overline{\mu}]$ is a matching record and m_C is a move for *Charlie* applicable to $[\overline{\mu}]$;
3. $\Sigma = \{d, w\}$ is a set of two propositions;
4. the valuation ν is such that for all $q \in Q_2$, $\nu(q) = \varnothing$, and for all $[\overline{\mu}] \in Q_1$, $d \in \nu([\overline{\mu}])$ iff $\overline{\mu} \models P_{\mathcal{D}}$ and $w \in \nu([\overline{\mu}])$ iff $\overline{\mu} \models P_{\mathcal{S}}$;
5. $\lambda(q) = 1$ if $q \in Q_1$ and $\lambda(q) = 2$ if $q \in Q_2$;
6. the transition relation is bipartite, relating only states from Q_1 to Q_2 or *vice versa*, and is defined as:
   (a) $([\overline{\mu}], ([\overline{\mu}'], \mathsf{m}_C)) \in R$ if and only if $[\overline{\mu}] = [\overline{\mu}']$;
   (b) $(([\overline{\mu}], \mathsf{m}_C), [\overline{\mu}']) \in R$ if and only if there is a move m_E for *Eve* such that the round $\rho = (\mathsf{m}_C, \mathsf{m}_E)$ is applicable to $\overline{\mu}$, and $[\rho(\overline{\mu})] = [\overline{\mu}']$.

Lemma 1 (Construction of the associated game structure)**.** *Let $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$ be a timeline-based game. The associated game structure S_G can be built in* doubly exponential time*, and its size is doubly exponential in the size of G.*

Proof. The size and construction complexity of the game structure S_G associated with a game G directly follows from the properties of matching records stated in Proposition 3. In particular, since each matching record built on top of $\mathcal{S} \cup \mathcal{D}$ has exponential size in (the size of) $\mathcal{S} \cup \mathcal{D}$ (Item 1 of Proposition 3), there is at most a doubly exponential number of possible matching records. By Item 2 of Definition 34, for each matching record $[\overline{\mu}]$, there is a doubly exponential number of states in S_G corresponding to the possible moves by *Charlie* applicable to $\overline{\mu}$, because of the time amounts δ_C in wait(δ) moves (recall that, by Proposition 2, we can restrict *w.l.o.g.* to $\delta \leq \text{window}(\mathcal{S} \cup \mathcal{D})$). Hence, S_G contains a doubly exponential (hence, most notably, *finite*) number of states. Now, each state can be labelled either with the w or d propositions (or both) in *exponential time*, by Item 2 of Proposition 3, and the successor $[\overline{\mu}\mu]$ of each node $([\overline{\mu}], \mathsf{m}_C)$ can be obtained in exponential time as well because of Items 3 and 4 of Proposition 3. Hence, S_G can be built in doubly exponential time. □

In order to formally tie to S_G the existence of winning strategies for G, we introduce a logical formulation of Definition 28 in terms of *alternating-time temporal logic* (ATL) or, more precisely, its extension ATL*. Introduced by Alur et al. [2], ATL and ATL* are strategic logics that are interpreted over concurrent game structures, of which turn-based synchronous structures are a special case. Given a set $\mathcal{P} = 1, \ldots, k$ of players and a finite set Σ of propositions, the syntax of ATL* is given in terms of *state formulas* and *path formulas*, defined as follows:

$$\begin{aligned} \phi &:= p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle\!\langle \mathsf{A} \rangle\!\rangle \psi && \text{state formulas} \\ \psi &:= \phi \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \mathsf{X}\psi_1 \mid \psi_1 \,\mathcal{U}\, \psi_2 && \text{path formulas} \end{aligned}$$

ATL* formulas are all the *state formulas* defined above. Given a game structure $S = \langle \mathcal{P}, Q, \Sigma, \nu, \lambda, R \rangle$ and a state $q \in Q$, the formula $\langle\!\langle \mathsf{A} \rangle\!\rangle \psi$ holds over S and q, written $S, q \models \langle\!\langle \mathsf{A} \rangle\!\rangle \psi$, if there exists a set of strategies F_A, one for each $a \in A$, such that $S, \overline{q} \models \psi$ for all paths $\overline{q} = \langle q, q', \ldots \rangle$ starting from q played according to F_A. The other connectives and temporal operators are defined as expected. See Alur et al. [2] for the complete semantics of the logic.

Lemma 2 (ATL* formulation of winning strategies)**.** *Let $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$ be a timeline-based game, and let S_G be its associated game structure. Then,* Charlie *has a winning strategy for G iff it holds that:*[1]

$$S_G, [\varepsilon] \models \langle\!\langle 1 \rangle\!\rangle (\mathsf{F}d \rightarrow \mathsf{F}w)$$

Proof ($\longrightarrow$). First, we prove that if *Charlie* has a winning strategy for G, then the given formula holds on the $[\varepsilon]$ state of S. Let $\sigma_C : \mathbf{\Pi} \rightarrow \mathcal{M}_C$ be such a strategy. A strategy for Player 1 of S_G is a function $\sigma_1 : Q^+ \rightarrow Q$. Let $\overline{q} = (q_0, \ldots, q_n)$ be a path in S_G where $q_0 = [\varepsilon]$. Suppose $\lambda(q_n) = 1$, *i.e.*, it is Player 1's turn to play. Given the particular transition relation of S_G, this means that n is even. By construction, states in even positions $\langle q_0, q_2, q_4, \ldots \rangle$ are a sequence of matching records $\langle [\overline{\mu}_0], [\overline{\mu}_2], \ldots \rangle$, whereas for the odd ones, $\langle q_1, q_3, q_5, \ldots \rangle$, we have $q_i = ([\overline{\mu}_{i-1}], \mathsf{m}_C)$, where m_C is a move for *Charlie* applicable to $\overline{\mu}_{i-1}$. *Vice versa*, states in odd positions are related to even ones by moves for *Eve*. Hence, from the path we can reconstruct the actual event sequence $\overline{\mu}_n$ built by the full play of the game. Suppose *Charlie* has a winning strategy σ_C for G, and let $\mathsf{m}_C = \sigma_C(\overline{\mu}_n)$. We can define a strategy σ_1 for Player 1 in S_G as $\sigma_1(\overline{q}) = \mathsf{m}_C$.

Now, let $\overline{w} = \langle w_0, w_1, \ldots \rangle$ be a path played according to the strategy σ_1 defined above and some strategy σ_2 for Player 2. We show that $\overline{w} \models \mathsf{F}d \rightarrow \mathsf{F}w$. Suppose $\mathsf{F}d$ holds. This means that there is $k \geq 0$ such that $S, w_k \models d$. Note that k is even, since, by definition, only states in Q_1 are labelled, hence let $w_k = [\overline{\mu}_k]$. Then, by construction, it follows that domain rules are satisfied by $\overline{\mu}_k$, hence σ_2 corresponds to an *admissible*

[1] The formula as shown in [33] contained an error, stemming from interpreting ATL* strategy quantifiers in the style of Strategy Logic.

strategy for *Eve* in the game G. Since σ_C is a winning strategy, we know that any play played according to it and any admissible strategy leads to a k' such that the rules in $\mathcal{S}$ are satisfied by $\overline{\mu}_{k'}$, and consequently w'_k is labelled by w, meaning that $\overline{w}$ satisfied $\mathsf{F}w$. We can conclude that all paths starting from $[\varepsilon]$ and played according to σ_1 satisfy $\mathsf{F}d \rightarrow \mathsf{F}w$, hence $S, [\varepsilon] \models \langle\!\langle 1 \rangle\!\rangle (\mathsf{F}d \rightarrow \mathsf{F}w)$.

($\longleftarrow$). Conversely, let us show that if the formula holds on S_G, then a winning strategy for *Charlie* exists on G. If $\langle\!\langle 1 \rangle\!\rangle (\mathsf{F}d \rightarrow \mathsf{F}w)$ holds on $[\varepsilon]$, there exists a strategy σ_1 for Player 1 such that for all paths $\overline{q} = \langle q_0, q_1, \ldots \rangle$ played according to σ_1, it holds that $S, \overline{q} \models \mathsf{F}d \rightarrow \mathsf{F}w$. From σ_1, we can define a corresponding strategy σ_C for *Charlie* similarly to the converse operation defined above: for each $\overline{\mu}$, a path $\overline{w}$ is defined that reconstructs $\overline{\mu}$, and then $\sigma_C(\overline{\mu}) = \sigma_1(\overline{w})$. Then, we can see that σ_C is a winning strategy: a play played according to an admissible strategy σ_E for *Eve* would correspond to a path $\overline{u} = \langle u_0, \ldots \rangle$, with $u_0 = [\varepsilon]$, such that $S, u_k \models d$ for some $k \geq 0$, which means that the domain rules are satisfied at the k-th round of the play of G. Since the path $\overline{w}$ is played according to σ_1, *i.e.*, the play is played according to σ_C, there is a k' where $S, u_{k'} \models w$ (because $\mathsf{F}d \rightarrow \mathsf{F}w$ holds), which means in turn that system rules are satisfied as well at the k' game round. Hence, σ_C is a winning strategy. □

Everything is now in place to prove the complexity of finding a winning strategy for a given game.

Theorem 3 (Complexity of finding winning strategies)**.** *Whether a timeline-based game G admits a winning strategy for* Charlie *can be decided in* doubly exponential time*.*

Proof. Let G be a timeline-based game. Thanks to Lemma 2, we can verify whether G admits a winning strategy for *Charlie* by building the corresponding turn-based synchronous game structure S_G and checking whether $S, [\varepsilon] \models \langle\!\langle 1 \rangle\!\rangle (\mathsf{F}d \rightarrow \mathsf{F}w)$. It is known from Alur et al. [2] that model checking an ATL^* formula on a concurrent game structure has *polynomial* time complexity in terms of the size of the structure, for formulas of *bounded* size. This is our case, since the formula that we need to check is fixed, and always the same for any G. Hence, as the structure can be built in doubly exponential time (Lemma 1), such is the complexity of checking whether G admits a winning strategy for *Charlie*. □

The ATL^* formula used by Theorem 3 is fixed and very simple, and most of the complexity of the procedure comes from the construction of the game structure. However, having framed the problem in logical terms gives us much flexibility in how to *extend* the current setting to more complex or expressive variants, or to different winning conditions. Exploring this potential is left as future work.

5.3. Finding whether winning strategies exist is 2EXPTIME*-complete*

We will now prove that deciding whether a winning strategy exists for *Charlie* in a given timeline-based game is 2EXPTIME-hard (hence 2EXPTIME-complete, as well). The proof is based on a reduction from a particular kind of *tiling games*, introduced by Chlebus [18] as a 2-player variant of common *tiling problems*. These kind of problems have been used for a long time as a source of reductions to study the computational complexity of many problems in logic and combinatorics [34, 35, 36, 43, 49, 51].

Definition 35 (Tiling structures and tilings). A *tiling structure* is a tuple $\mathcal{T} = (T, t^0, t^*, H, V, n)$, where T is a set of elements called *tiles*, $t_0 \in T$ is the *initial* tile, $t_* \in T$ is the *final* tile, $H, V \subseteq T \times T$ are the *horizontal* and *vertical adjacency relations*, and $n \in \mathbb{N}_+$ is a positive number, encoded in *binary*.

A *k-tiling* of the tiling structure $\mathcal{T}$, for $k > 1$, is a function $f : [n] \times [k] \rightarrow T$, mapping any position (x, y) of the rectangle of size $n \times k$ to a tile $f(x, y) \in T$ such that:

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

The *exponential rectangle tiling problem* is the problem of deciding, given a tiling structure $\mathcal{T}$, if there exists a k-tiling for $\mathcal{T}$ for some $k > 1$. The problem is known to be EXPSPACE-complete [49]. The exponential rectangle tiling *game*, is a 2-player variant of the problem, where a player, the *Constructor*, tries to build a k-tiling for a given tiling structure, and the opposite player, the *Saboteur*, tries to prevent it to happen. The two players play in turn, starting from *Constructor*, choosing one tile at the time, starting from the one in position $(0, 0)$, filling one row after the other. A strategy for *Constructor* is a function $\sigma : T^* \to T$, that given the sequence of tiles positioned up to the current time, gives the next tile to play. Given a tiling structure $\mathcal{T}$, a *winning strategy* lets *Constructor* build a k-tiling, for some $k > 1$, no matter which tiles are chosen by *Saboteur*.

Given a tiling structure $\mathcal{T}$, the problem of deciding whether *Constructor* has a winning strategy can be seen to be 2EXPTIME-complete: Chlebus [18] proves that the problem is EXPTIME-complete if n is encoded in *unary*, while here it is encoded in *binary*. Following the same proof with this difference, we can obtain the 2EXPTIME-completeness result. We can now prove how to reduce tiling games to timeline-based games.

Theorem 4 (Deciding the existence of winning strategies is 2EXPTIME-hard)**.** *Let $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$ be a timeline-based game. The problem of deciding whether* Charlie *has a winning strategy for G is* 2EXPTIME*-hard.*

Proof. As anticipated, the proof goes by reduction from *exponential rectangle tiling games*. We prove that, given any tiling structure $\mathcal{T} = (T, t_0, t^*, H, V, n)$, we can build in polynomial time a corresponding timeline-based game $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathcal{S}, \mathcal{D})$ such that *Charlie* has a winning strategy for G if and only if *Constructor* has a winning strategy in the tiling game over $\mathcal{T}$. The timeline-based game encoding needs the implementation of a binary counter, repeatedly counting from 0 to n. The bits of the counter are represented by a number of variables $c_1, \ldots, c_m \in \mathsf{SV}_E$, where $m = \lceil \log_2(n) \rceil$. The binary variables are *uncontrollable*, *i.e.*, $\gamma(c_i) = \mathsf{u}$ for all $1 \leq i \leq m$, have all domain $V_{c_i} = \{0, 1\}$, and have trivial transition function and unit duration, *i.e.*, $T_{c_i}(0) = T_{c_i}(1) = \{0, 1\}$ and $D_{c_i}(0) = D_{c_i}(1) = (1, 1)$ for all $1 \leq i \leq m$. It can be seen that, with a polynomial number of synchronisation rules of polynomial size, it is possible to force these variables to encode the correct behaviour of the counter. Such rules are placed in $\mathcal{D}$, so that the evolution of the counter is completely handled by *Eve*. Other rules can look at them to query the current value.

Then, the rectangle to be tiled is represented by two variables $x \in \mathsf{SV}_C$ and $y \in \mathsf{SV}_E$, defined as follows. For $x = (V_x, T_x, D_x, \gamma_x) \in \mathsf{SV}_C$, the domain is defined as $V_x = \{t_{even}, t_{odd} \mid t \in T\}$, *i.e.*, an *even* and *odd* version of each tile $t \in T$. The transition function forces a strict alternation between *even* and *odd* values, *i.e.*, $T_x(t_{even}) = \{t'_{odd} \mid t' \in T\}$ and $T_x(t_{odd}) = \{t'_{even} \mid t' \in T\}$ for each $t \in T$. Any token for x is controllable and is forced to be of unit duration, *i.e.*, $D_x(v) = (1, 1)$ and $\gamma(x) = \mathsf{c}$ for each $v \in V_x$. In contrast, the domain of y contains a single value for each tile, with the addition of a special symbol $\perp$, *i.e.*, $V_y = T \cup \{\perp\}$, and any token for y is *uncontrollable* and forced to last two units of time, *i.e.*, $D_y(v) = (2, 2)$ and $\gamma_y(v) = \mathsf{u}$ for all $v \in V_y$. The transition function is trivial, defined as $T_y(v) = V_y$ for all $v \in V_y$.

With these variables and suitable synchronisation rules, we can simulate the tiling game. Tiles chosen by *Charlie* (in the role of *Constructor*) are directly put on his timelines. Tiles chosen by *Eve* (in the role of *Saboteur*) are put on her timeline, and then replicated by *Charlie*, in order to turn the timeline for x into a row-major representation of the current partially tiled rectangular area. The separation between even and odd values in the domain of x is needed to tell *Charlie* whether in the current turn it is time to freely choose the next tile, or to blindly replicate *Eve*'s choice. As remarked in Section 4, *Charlie* needs at least one time step of delay to replicate *Eve*'s moves, while *Eve* can reply immediately. For this reason, *Eve*'s tokens last two time steps, so that *Charlie* has time to see *Eve*'s choice and replicate. Tokens on the two timelines remains aligned: *Eve*'s tokens span over the last *Charlie* choice and *Eve*'s move replica. Now, we show the domain and system rules that can enforce such a dynamics, starting with the basic construction of the grid. The first token of *Charlie*'s timeline must be t^0_{even}, to enforce the base case of the tiling, and the fact that the token starting at time 0 is, in fact, marked as *even*:

$$\top \to a[x = t^0_{even}] \,.\, \mathrm{start}(a) = 0 \qquad (\text{in } \mathcal{S})$$

Then, *Charlie* is instructed to blindly replicate *Eve*'s choice with an odd token after every even one, unless the current tile is t^* and the counter says this is the end of the row, in which case *Charlie* have won:

$$a[x = t_{even}] \to \bigvee_{t' \in T} \exists b[y = t']c[x = t'_{odd}] \,.\, \mathrm{start}(a) = \mathrm{start}(b) \wedge \mathrm{end}(a) = \mathrm{start}(c) \qquad (\text{in } \mathcal{S}, \text{ for } t \neq t^* \in T)$$

$$a[x = t^*_{even}] \to \bigvee_{t' \in T} \exists b[y = t']c[x = t'_{odd}] \,.\, \text{start}(a) = \text{start}(b) \wedge \text{end}(a) = \text{start}(c) \qquad \text{(in } \mathcal{S}\text{)}$$
$$\vee \;\; \exists b_1[c_1 = n_1] \ldots b_m[c_m = n_m] \,.\, b_0 = a \wedge \ldots \wedge b_m = a \qquad \text{(where } \langle n_1, \ldots, n_m \rangle = n\text{)}$$

Let us shorthand the second disjunct of the above rule simply as $\overline{c} = n$, and as $\overline{c} = 0$ the similar disjunct that requires all c_i to be zero. Then, we have to enforce the adjacency conditions for the tiling, which must be obeyed by both players. For *Charlie*, this can be done as follows, for the horizontal relation:

$$a[x = t_{even}] \to \overline{c} = n \vee \bigvee_{\substack{t' \in T \\ tHt'}} \exists b[x = t'_{odd}] \,.\, \text{end}(a) = \text{start}(b) \qquad \text{(in } \mathcal{S}\text{, and with even/odd swapped)}$$
$$a[x = t_{even}] \to \overline{c} = 0 \vee \bigvee_{\substack{t' \in T \\ t'Ht}} \exists b[x = t'_{odd}] \,.\, \text{end}(b) = \text{start}(a) \qquad \text{(in } \mathcal{S}\text{, and with even/odd swapped)}$$

and for the vertical relation:

$$a[x = t_{even}] \to \bigvee_{\substack{t' \in T \\ tVt'}} \exists b[x = t'_{even}] \,.\, \text{end}(a) \leq_{[n,n]} \text{start}(b) \qquad \text{(in } \mathcal{S}\text{)}$$
$$\vee \;\; \exists g[x = t^*_{even}] \underbrace{b_i[c_i = n_i]}_{1 \leq i \leq m} \,.\, g = b_1 \wedge \ldots \wedge g = b_m \wedge a \leq_{[0,n]} g$$
$$a[x = t_{even}] \to \text{start}(a) \leq n \vee \bigvee_{\substack{t' \in T \\ t'Vt}} \text{start}(a) \leq n \vee \exists b[x = t'_{even}] \,.\, \text{end}(b) \leq_{[n,n]} \text{start}(a) \qquad \text{(in } \mathcal{S}\text{)}$$

Note how in the rules above, the last column and the last row are detected using, respectively, the value of the counter and the presence of the final tile.

For *Eve*, the rules that enforce the adjacency relations are similar, but with two differences. First of all, the rules are triggered by tokens on the y variable, which implies that they do not check the consistency of *Charlie*'s choices. This is because a wrong choice by *Charlie* has to make it lose, not to make the run not admissible. The second difference is that, by means of additional disjuncts, *Eve* is admitted to play a token with value $y = \bot$ in place of any other tile. This value, however, is allowed only when no other viable choice is available, by means of a rule such as:

$$a[y = \bot] \to \bigvee_{\substack{t' \in T \\ \text{not } t' \; V \; t}} \exists b[x = t'_{even}] \,.\, b \text{ is } \textit{above } a \text{ in the tiling} \qquad \text{(in } \mathcal{D}\text{)}$$
$$\vee \bigvee_{\substack{t' \in T \\ \text{not } t' \; H \; t}} \exists b[x = t'] \,.\, b \text{ is } \textit{on the left of } a \text{ in the tiling} \qquad \text{(in } \mathcal{D}\text{)}$$

All the rules encoding the behaviour of *Eve* are written in such a way that they do not hold after $\bot$ is played. Since $\bot \notin V_x$, when *Eve* plays it, *Charlie* cannot replicate her move, hence violating his rules and losing the game. In this way, the inability of *Eve* to progress in the tiling is turned into a lost game by *Charlie* instead of an inadmissible run, accordingly to the semantics of the tiling game. Finally, we can state the final goal of *Charlie*, namely, that of placing the final tile as the top tile of the current row:

$$\top \to \exists g[x = t^*_{even}] \underbrace{b_i[c_i = n_i]}_{1 \leq i \leq m} \,.\, g = b_1 \wedge \ldots \wedge g = b_m \qquad \text{(in } \mathcal{S}\text{)}$$
$$\vee \;\; \exists g[x = t^*_{odd}] \underbrace{b_i[c_i = n_i]}_{1 \leq i \leq m} \,.\, g = b_1 \wedge \ldots \wedge g = b_m \qquad \text{(in } \mathcal{S}\text{)}$$

It can be easily checked that the number and size of all the rules described above is polynomial in the size of the tiling problem, and that a winning strategy for *Charlie* effectively corresponds to a winning strategy for *Constructor*, hence concluding the proof. □

6. Conclusions and future work

In this paper, we introduced *uncertainty* in the recent body of work devoted to the investigation of formal properties of timeline-based planning problems [23, 30, 31, 32]. Rather than studying the complexity of the problem of finding dynamically controllable flexible plans – a research direction which will be worth exploring anyway – we took a more proactive approach, analysing some issues of the current approach based on flexible plans, and proposing a more general game-theoretic formulation of the problem.

We generalised timeline-based planning problems with uncertainty by defining a novel concept of *timeline-based game*, where the controller tries to execute some tasks as dictated by a timeline-based model, independently of the choices of the environment. In comparing this approach to the state-of-the-art one, we showed that the existence of winning strategies for timeline-based games is strictly more general than the existence of dynamically controllable flexible plans: the latter implies the former, but there are some problems that, when stated as games, have easy winning strategies but do not admit dynamically controllable flexible plans. Then, we analysed the computational complexity of checking whether a winning strategy exists for a given timeline-based planning game, proving that the problem is 2EXPTIME-complete.

This work opens the way for further interesting developments. First of all, the problem of how to efficiently *synthesize* a controller implementing a winning strategy for a given game is still open, with a look as well at the *quality* of the synthesized strategy. Work in this direction may exploit existing machinery from the field of reactive synthesis of logical specifications, by means of the logical encoding of timeline-based problems given by Della Monica et al. [23].

Then, timeline-based games may be extended to multi-agent scenarios, where multiple players are involved, each with its own objectives and constraints, all playing in the surrounding environment. Strategies may be synthesized for single players or for coalitions, sharing some objectives while pursuing also individual goals. This setting could be further extended to *distributed* games, where players do not share a single clock, and communicate via message passing. Variants with partial observability are also an interesting direction. Having framed the problem in terms of model checking of ATL* formulas will allow us to extend our work to other settings while exploring the full potential of such logics and of the framework of concurrent game structures.

Finally, extending the modelling language to cope with much-needed features such as representation and handling of *resources* might be fundamental to handle complex real-world scenarios. On the other hand, given the high computational complexity of solving the games, the pursuit of easier fragments, such as with bounded durations or bounded horizon of the game, is an important step towards the application of this approach.

Acknowledgements

We thank Nicolas Markey for his help in improving our complexity analysis, leading to a tight upper bound. Thanks also to reviewers for their useful remarks.

Bibliography

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. doi: 10.1145/182.358434.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002. doi: 10.1145/585265.585270.

[3] T. Bedrax-Weiss, C. McGann, A. Bachmann, W. Edgington, and M. Iatauro. Europa2: User and contributor guide. Technical report, NASA Ames Research Center, 2005.

[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. Bernardini and D. E. Smith. Developing domain-independent search control for europa2. In *Proceedings of the ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning*, 2007.

[6] S. Bernardini and D. E. Smith. Translating pddl2.2. into a constraint-based variable/value language. In *Proceedings of the ICAPS 2008 Workshop on Heuristics for Domain-Independent Planning*, 2008.

[7] 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.

[8] A. Cesta and A. Oddi. Ddl.1: A formal description of a constraint representation language for physical domains. In M. Ghallab and A. Milani, editors, *New directions in AI planning*. IOS Press, 1996.

[9] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and N. Policella. Software companion: The mexar2 support to space mission planners. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *Proceedings of the 17th European Conference on Artificial Intelligence*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 622–626. IOS Press, 2006.

[10] A. Cesta, G. Cortellessa, M. Denis, A. Donati, S. Fratini, A. Oddi, N. Policella, E. Rabenau, and J. Schulster. Mexar2: AI solves mission planner problems. *IEEE Intelligent Systems*, 22(4):12–19, 2007. doi: 10.1109/MIS.2007.75.

[11] A. Cesta, S. Fratini, and F. Pecora. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Science*, 18(2):231–271, 2008. ISSN 1230-2384.

[12] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Analyzing flexible timeline-based plans. In H. Coelho, R. Studer, and M. Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 471–476. IOS Press, 2010. doi: 10.3233/978-1-60750-606-5-471.

[13] A. Cesta, L. M. Tosatti, A. Orlandini, N. Pedrocchi, S. Pellegrinelli, T. Tolio, and A. Umbrico. Planning and execution with robot trajectory generation in industrial human-robot collaboration. In S. M. Anzalone, A. Farinelli, A. Finzi, and F. Mastrogiovanni, editors, *Proceedings of the 4th Italian Workshop on Artificial Intelligence and Robotics*, volume 2054 of *CEUR Workshop Proceedings*, pages 47–52. CEUR-WS.org, 2017. URL `http://ceur-ws.org/Vol-2054/paper8.pdf`.

[14] A. Cesta, G. Cortellessa, A. Orlandini, and A. Umbrico. A cognitive architecture for autonomous assistive robots. *ERCIM News*, 2018(114), 2018. URL `https://ercim-news.ercim.eu/en114/special/a-cognitive-architecture-for-autonomous-assistive-robots`.

[15] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. Aspen - automating space mission operations using automated planning and scheduling. In *Proceedings of the International Conference on Space Operations*, 2000.

[16] S. A. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castaño, A. Davies, R. Lee, D. Mandl, S. Frye, B. Trout, J. Hengemihle, J. D'Agostino, S. Shulman, S. G. Ungar, T. Brakke, D. Boyer, J. V. Gaasbeck, R. Greeley, T. Doggett, V. R. Baker, J. M. Dohm, and F. Ip. The EO-1 autonomous science agent. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 420–427. IEEE Computer Society, 2004. doi: 10.1109/AAMAS.2004.10022.

[17] S. A. Chien, G. Rabideau, D. Tran, M. Troesch, J. Doubleday, F. Nespoli, M. P. Ayucar, M. C. Sitja, C. Vallat, B. Geiger, N. Altobelli, M. Fernandez, F. Vallejo, R. Andres, and M. Kueppers. Activity-based scheduling of science campaigns for the rosetta orbiter. In Q. Yang and M. Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 4416–4422. AAAI Press, 2015. URL `http://ijcai.org/Abstract/15/655`.

[18] B. S. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986. doi: 10.1016/0022-0000(86)90036-X.

[19] 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. doi: 10.1007/s00236-015-0252-z.

[20] 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.

[21] A. Cimatti, A. Micheli, and M. Roveri. Timelines with temporal uncertainty. In M. desJardins and M. L. Littman, editors, *Proceedings of the 27th AAAI Conference on Artificial Intelligence*. AAAI Press, 2013. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6319`.

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

[23] D. Della Monica, N. Gigante, A. Montanari, P. Sala, and G. Sciavicco. Bounded timed propositional temporal logic with past captures timeline-based planning with bounded constraints. In C. Sierra, editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1008–1014, 2017. doi: 10.24963/ijcai.2017/140.

[24] D. Della Monica, N. Gigante, A. Montanari, and P. Sala. A novel automata-theoretic approach to timeline-based planning. In M. Thielscher, F. Toni, and F. Wolter, editors, *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning*, pages 541–550. AAAI Press, 2018. URL `https://aaai.org/ocs/index.php/KR/KR18/paper/view/18024`.

[25] European Space Agency. Apsi - advanced planning and scheduling initiative. URL `https://essr.esa.int/project/apsi-advanced-planning-and-scheduling-initiative`.

[26] R. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971. doi: 10.1016/0004-3702(71)90010-5.

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

[28] S. Fratini and L. Donati. Apsi timeline representation framework v. 3.0. Technical report, European Space Agency - ESOC, 2011.

[29] S. Fratini, A. Cesta, A. Orlandini, R. Rasconi, and R. De Benedictis. Apsi-based deliberation in goal oriented autonomous controllers. In *ASTRA 2011*, volume 11. ESA, 2011.

[30] N. Gigante. *Timeline-based Planning: Expressiveness and Complexity*. PhD thesis, University of Udine, Italy, 2019. Available on *arXiv*.

[31] N. Gigante, A. Montanari, M. Cialdea Mayer, and A. Orlandini. Timelines are expressive enough to capture action-based temporal planning. In C. E. Dyreson, M. R. Hansen, and L. Hunsberger, editors, *Proceedings of the 23rd International Symposium on Temporal Representation and Reasoning*, pages 100–109. IEEE Computer Society, 2016. doi: 10.1109/TIME.2016.18.

[32] N. Gigante, A. Montanari, M. Cialdea Mayer, and A. Orlandini. Complexity of timeline-based planning. In L. Barbulescu,

J. Frank, Mausam, and S. F. Smith, editors, *Proceedings of the 27th International Conference on Automated Planning and Scheduling*, pages 116–124. AAAI Press, 2017. URL `https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15758`.

[33] N. Gigante, A. Montanari, M. Cialdea Mayer, A. Orlandini, and M. Reynolds. A game-theoretic approach to timeline-based planning with uncertainty. In N. Alechina, K. Nørvåg, and W. Penczek, editors, *Proceedings of the 25th International Symposium on Temporal Representation and Reasoning*, volume 120 of *LIPIcs*, pages 13:1–13:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi: 10.4230/LIPIcs.TIME.2018.13.

[34] D. S. Johnson. A Catalog of Complexity Classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 67–161. 1990.

[35] L. Levin. Universal sequential search problems. *Problems in Information Transmission*, 9:265–266, 1973.

[36] H. R. Lewis. *Unsolvable Classes of Quantificational Formulas*. Addison-Wesley, Reading, Mass., 1979. ISBN 0201040697.

[37] 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.

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

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

[40] N. Muscettola, S. F. Smith, A. Cesta, and D. D'Aloisi. Coordinating space telescope operations in an integrated planning and scheduling architecture. *IEEE Control Systems*, 12:28–37, 1992.

[41] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998. doi: 10.1016/S0004-3702(98)00068-X.

[42] 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.

[43] M. P. W. Savelsbergh and P. van Embde Boas. Bounded tiling, an alternative to satisfiability? In *Proc. of the 2nd Frege Conference*, volume 20 of *Mathematische Forschung*, pages 354–363. Akademic Verlag, 1984.

[44] D. E. Smith, J. Frank, and A. K. Jónsson. Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, 15(1):47–83, 2000.

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

[46] 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.

[47] A. Umbrico, A. Cesta, M. Cialdea Mayer, and A. Orlandini. Platinum: A new framework for planning and acting. In F. Esposito, R. Basili, S. Ferilli, and F. A. Lisi, editors, *Proceedings of the 16th International Conference of the Italian Association for Artificial Intelligence*, volume 10640 of *Lecture Notes in Computer Science*, pages 498–512. Springer, 2017. doi: 10.1007/978-3-319-70169-1_37.

[48] A. Umbrico, A. Cesta, M. Cialdea Mayer, and A. Orlandini. Integrating resource management and timeline-based planning. In M. de Weerdt, S. Koenig, G. Röger, and M. T. J. Spaan, editors, *Proceedings of the 28th International Conference on Automated Planning and Scheduling*, pages 264–272. AAAI Press, 2018. URL `https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17773`.

[49] P. van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture Notes in Pure and Applied Mathematics*, pages 331–363. Marcel Dekker Inc., 1997.

[50] 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. doi: 10.1080/095281399146607.

[51] H. Wang. Proving theorems by pattern recognition I. *Communications of the ACM*, 3(4):220–234, 1960. doi: 10.1145/367177.367224.