Controller Synthesis for Timeline-based Games

Renato Acampora
University of Udine, Italy
acampora.renato@spes.uniud.it

Luca Geatti   Nicola Gigante
Free University of Bozen-Bolzano
{geatti,gigante}@inf.unibz.it

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

Valentino Picotti
University of Southern Denmark
picotti@imada.sdu.dk

In the timeline-based approach to planning, originally born in the space sector, the evolution over time of a set of state variables (the timelines) is governed by a set of temporal constraints. Traditional timeline-based planning systems excel at the integration of planning with execution by handling *temporal uncertainty*. In order to handle general nondeterminism as well, the concept of *timeline-based games* has been recently introduced. It has been proved that finding whether a winning strategy exists for such games is 2EXPTIME-complete. However, a concrete approach to synthesize controllers implementing such strategies is missing. This paper fills this gap, outlining an approach to controller synthesis for timeline-based games.

1 Introduction

In the timeline-based approach to planning, the world is viewed as a system made of a set of independent but interacting components whose behaviour over time (the timelines) is governed by a set of temporal constraints, called *synchronization rules*. Timeline-based planning has been originally introduced in the space industry [19], with timeline-based planners developed and used by space agencies on both sides of the Atlantic [5, 4, 13, 2, 6], both for short- to long-term mission planning [7] and on-board autonomy [14].

While successful in practice, only recently timeline-based planning has been studied from a theoretical perspective. The formalism has been at first compared with traditional action-based languages *à la* STRIPS, proving that they can be expressed by means of timeline-based languages [16]. Then, the complexity of the timeline-based plan existence problem has been studied: the problem is EXPSPACE-complete [17] over discrete time in the general case, and PSPACE-complete with qualitative constraints [11]. On dense time, the problem goes from being NP-complete to undecidable, depending on the syntactic restrictions applied [3]. The expressiveness of timeline-based languages has also been studied from a logical perspective [10], and an automata-theoretic point of view [9].

Traditional timeline-based planning systems excel at the integration of planning with *execution* by treating explicitly the concept of *temporal uncertainty*: the exact timings of the events under control of the environment need not to be precisely known in advance. However, general nondeterminism, where the environment can also decide *what* to do (instead of only *when* to do it) is usually not handled by these systems. To overcome this limitation, the concept of *timeline-based games* has been recently introduced [18]. In these games, the state variables are partitioned between the controller and the environment, and the latter has the freedom to play arbitrarily as long as a set of *domain* rules, that define the game arena, are satisfied. The controller plays to satisfy his set of *system* rules. A strategy for controller is winning if it allows him/her to win independently from the choices of the environment.

Establishing whether a winning strategy exists for these games has been proved to be 2EXPTIME-complete [18]. However, no concrete way to synthesize a controller implementing such strategies is

D. Della Monica and P. Ganty (Eds.): 13th International Symposium on Games, Automata, Logics and Formal Verification (GandALF 22)
EPTCS 370, 2022, pp. 131–146, doi:10.4204/EPTCS.370.9

known. The proof technique of the aforementioned complexity result involves the construction of a huge (doubly exponential) *concurrent game structure*, which is used to model check some Alternating-time Temporal Logic (ATL) formulas [1]. While this structure is deterministic and can be in principle used as an arena to solve a reachability game and synthesize a controller, its construction is based on theoretical nondeterministic procedures which have no hope to be ever concretely implemented. On the other hand, the automata-theoretic approach by Della Monica *et al*. [9] provides a concrete and effective construction of an automaton that accepts a word if and only if the original planning problem has a solution plan. However, the automaton is *nondeterministic* and already doubly exponential, and the determinization needed to use it as an arena would result into a further blow up and a non-optimal procedure.

In this paper, we provide a concrete and computationally optimal approach to controller synthesis for timeline-based games. We overcome the limitations of both the above-mentioned approaches by devising a direct construction for a *deterministic* finite-state automaton that recognizes solution plans, which is doubly exponential in size (thus not requiring the determinization of a nondeterministic automaton). This automaton is then used as the arena of a reachability game for which plenty of controller synthesis techniques are known in the literature.

The paper is structured as follows. In Section 2 we introduce the needed background on timeline-based planning and timeline-based games. Then, Section 3 provides the core technical contribution of the paper, namely the construction of the deterministic automaton recognizing solution plans. Section 4 uses this automaton as the game arena to solve the controller synthesis problem. Last, Section 5 summarizes the main contributions of the work and discuss future developments.

2 Timeline-based games

In this section, we introduce timeline-based games, as defined in [18].

2.1 State variables, event sequences, synchronization rules

The first basic concept is that of *state variable*.

Definition 1 (State variable)**.** *A* state variable *is a tuple* $x = (V_x, T_x, D_x, \gamma)$*, where:*

- V_x *is the* finite domain *of* x*;*
- $T_x : V_x \to 2^{V_x}$ *is the* value transition function *of* x*, which maps each value* $v \in V_x$ *to the set of values that can immediately follow it;*
- $D_x : V_x \to \mathbb{N} \times \mathbb{N}$ *is the* duration function *of* x*, mapping each value* $v \in V_x$ *to a pair* $(d^{x=v}_{min}, d^{x=v}_{max})$ *specifying respectively the minimum and maximum duration of any interval where* $x = v$*;*
- $\gamma : V_x \to \{\mathsf{c}, \mathsf{u}\}$ *is the* controllability tag*, that, for each value* $v \in V_x$*, specifies whether it* controllable *(*$\gamma(v) = \mathsf{c}$*) or* uncontrollable *(*$\gamma(v) = \mathsf{u}$*).*

Intuitively, a state variable x takes a value from a finite domain and represents a simple finite-state machine, whose transition function is T_x. The behaviour over time of a set of state variables SV is defined by a set of *timelines*, one for each variable. Instead of reasoning about timelines directly, though, in this paper we follow the approach outlined in [18] and represent the whole execution of a system, modeled by means of a set of state variables, with a single word, called *event sequence*.

Definition 2 (Event sequence [18])**.** *Let* SV *be a set of state variables. Let* A_{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* $A_i \subseteq \mathsf{A}_{\mathsf{SV}}$ *is a set of actions, and* $\delta_i \in \mathbb{N}_+$*, such that, for any* $x \in \mathsf{SV}$*:*

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

Intuitively, an event sequence represents the evolution over time of the state variables of the system by representing the *start* and the *end* of *tokens*, *i.e.*, a sequence of adjacent intervals where a given variable takes a given value. 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. In an *event sequence*, events are collected to describe a whole plan.

Definition 2 intentionally implies that 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 have ever appeared before. In this case we say the event sequence is *open* (on the right or on the left, respectively). Otherwise, it is said to be *closed*. An event sequence closed on the left and open on the right is also called a *partial plan*. 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, 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, with $\mu_i = (A_i, \delta_i)$, we define $\delta(\overline{\mu}) = \sum_{1<i\le n} \delta_i$, that is, $\delta(\overline{\mu})$ is the time elapsed from the start to 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 $\delta(\overline{\mu}_{i,j}) = \sum_{i<k\le j} \delta_k$. Finally, given an event sequence $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$, for each $1 < i \le n$, we define $\overline{\mu}_{<i}$ as $\langle \mu_1, \ldots, \mu_{i-1} \rangle$.

In timeline-based games, the controller plays to satisfy a set of *synchronization rules*, which describe the desired behavior of the system. Synchronization rules relate tokens, possibly belonging to different timelines, through temporal relations among token endpoints. Let SV be a set of state variables and $\mathsf{N} = \{a, b, \ldots\}$ be an arbitrary set of *token names*. Moreover, let an *atomic temporal relation*, or simply *atom*, be an expression of the form $\langle term \rangle \le_{l,u} \langle term \rangle$, where $l \in \mathbb{N}$, $u \in \mathbb{N} \cup \{\infty\}$, and a *term* is either $\text{start}(a)$ or $\text{end}(a)$, for some $a \in \mathsf{N}$. A synchronisation rule R takes the following form:

$$\begin{aligned} \mathsf{R} \;&:=\; a_0[x_0 = v_0] \rightarrow \mathsf{E}_1 \vee \mathsf{E}_2 \vee \cdots \vee \mathsf{E}_k \quad \textit{where} \\ \mathsf{E}_i \;&:=\; \exists a_1[x_1 = v_1] a_2[x_2 = v_2] \ldots a_n[x_n = v_n] \,.\, \mathsf{C}_i \end{aligned}$$

where $a_0, \ldots, a_n \in \mathsf{N}$, $x_0, \ldots, x_n \in \mathsf{SV}$, $v_0, \ldots, v_n$ are such that $v_i \in V_{x_i}$, for all $0 \le i \le n$, and C_i is a conjunction of atomic temporal relations (a clause). The elements $a_i[x_i = v_i]$ are called *quantifiers* and the quantifier $a_0[v_0 = x_0]$ is called the *trigger*. The disjuncts in the body are called *existential statements*.

We say that a token $\tau = (x, v, d)$ *satisfies* a quantifier $a_i[x_i = v_i]$ if $x = x_i$ and $v = v_i$. The semantics of a synchronisation rule R states that for every token satisfying the trigger, at least one of the existential statements is satisfied. Each existential statement E_i requires the existence of some tokens, satisfying the quantifiers in its prefix, such that the clause C_i is satisfied. When a token satisfies the trigger of a rule, it is said to *trigger* such a rule.

For space concerns, we do not provide all the details of the semantics of synchronization rules. The reader can find them in [18]. Intuitively, each time there is a token that satisfies the trigger of a rule, one of its existential statements must be satisfied as well. The existential statements in turn assert the existence of other tokens that satisfy a conjunction of atoms.

If a and b are two token names, then examples of atomic relations are $\mathsf{start}(a) \leq_{3,7} \mathsf{end}(b)$ and $\mathsf{start}(a) \leq_{0,+\infty} \mathsf{start}(b)$. Intuitively, a token name a refers to a specific token, that is, a pair of $\mathsf{start}(x,v)$ and $\mathsf{end}(x,v)$ actions in an event sequence, and $\mathsf{start}(a)$ and $\mathsf{end}(a)$ to its endpoints. Then, an atom such as $\mathsf{start}(a) \leq_{l,u} \mathsf{end}(b)$ constrains a to start before the end of b, with the distance between the two endpoints to be comprised between the lower and upper bounds l and u.

Examples of synchronization rules are the following, where the relations $=$ and $\leq$ are respectively syntactic sugar for $\leq_{0,0}$ and $\leq_{0,+\infty}$:

$$a[x_s = \mathsf{Comm}] \rightarrow \exists b[x_g = \mathsf{Available}] \,.\, \mathsf{start}(b) \leq \mathsf{start}(a) \wedge \mathsf{end}(a) \leq \mathsf{end}(b)$$
$$a[x_s = \mathsf{Science}] \rightarrow \exists b[x_s = \mathsf{Slewing}]c[x_s = \mathsf{Earth}]d[x_s = \mathsf{Comm}] \,.$$
$$\mathsf{end}(a) = \mathsf{start}(b) \wedge \mathsf{end}(b) = \mathsf{start}(c) \wedge \mathsf{end}(c) = \mathsf{start}(d)$$

where the variables x_s and x_g represent respectively the state of a spacecraft and the visibility of the communication ground station. The first rule requires the satellite and the ground station to synchronise their communications, so that when the satellite is transmitting the ground station is available for reception. The second rule instructs the system to transmit data back to Earth after every measurement session, interleaved by the required slewing operation. A rule whose trigger is empty ($\top$), called *triggerless rule*, can be used to state the *goal* of the system. As an example, they allow one to force the spacecraft to perform some scientific measurement at all:

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

Triggerless rules have a trivial universal quantification, which means they only demand the existence of some tokens, as specified by the existential statements. Although triggerless rules are meant to specify the goals of a planning problem, they can be regarded as syntactic sugar on top of the syntax described above. Indeed, triggerless rules can be translated into triggered rules [18], and thus we do not consider them from here onwards.

Finally, even though our focus is on timeline-based games, we conclude the section by formally defining *timeline-based planning problems*.

Definition 3 (Timeline-based planning problem)**.** *A* timeline-based planning problem *is a pair* $P = (\mathsf{SV}, \mathsf{S})$*, where* SV *is a set of state variables and* S *is a set of synchronization rules over* SV*. An event sequence* $\overline{\mu}$ *over* SV *is a solution plan for* P *if all the rules in* S *are satisfied by* $\overline{\mu}$*.*

2.2 The game arena

We are now ready to introduce *timeline-based games*. Their definition is quite involved, as their structure has been designed with the goal of being strictly more general than *timeline-based planning with uncertainty* [8] while being able to capture its semantics precisely. For space concerns, we keep the exposition quite terse, but the reader can refer to [18] for details.

Definition 4 (Timeline-based game)**.** *A* timeline-based game *is a tuple* $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathsf{S}, \mathsf{D})$*, where* SV_C *and* SV_E *are the sets of* controlled *and* external *variables, respectively, and* S *and* D *are the sets of* system *and* domain *synchronisation rules, respectively, both 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 Π_G be the set of all possible partial plans for G, simply Π when there is no ambiguity.

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 richer partial plan, starting from ε, by playing actions that specify which tokens to start and/or to end, adding an event that extends the event sequence, or complementing the existing last event of the sequence. We partition all the available actions into those that are playable by either of the two players.

Definition 5 (Partition of player actions)**.** *Let* $\mathsf{SV} = \mathsf{SV}_C \cup \mathsf{SV}_E$*. The set* A *of available actions over* SV *is partitioned into the sets* A_C *of* Charlie*'s actions and* A_E *of* Eve*'s actions, which are defined as follows:*

$$\mathsf{A}_C = \underbrace{\{\, \mathrm{start}(x,v) \mid x \in \mathsf{SV}_C,\ v \in V_x \,\}}_{\textit{start tokens on } \text{Charlie}\textit{'s timelines}} \cup \underbrace{\{\, \mathrm{end}(x,v) \mid x \in \mathsf{SV},\ v \in V_x,\ \gamma_x(v) = \mathsf{c} \,\}}_{\textit{end controllable tokens}} \tag{1}$$

$$\mathsf{A}_E = \underbrace{\{\, \mathrm{start}(x,v) \mid x \in \mathsf{SV}_E,\ v \in V_x \,\}}_{\textit{start tokens on } \text{Eve}\textit{'s timelines}} \cup \underbrace{\{\, \mathrm{end}(x,v) \mid x \in \mathsf{SV},\ v \in V_x,\ \gamma_x(v) = \mathsf{u} \,\}}_{\textit{end uncontrollable tokens}} \tag{2}$$

Hence, players can start tokens for the variables that they own, and end the tokens that hold values that they control. Actions are combined into *moves* that can start/end multiple tokens at once.

Definition 6 (Moves)**.** *A* move μ_C *for* Charlie *is a term of the form* $\mathrm{wait}(\delta_C)$ *or* $\mathrm{play}(A_C)$*, where* $\delta_C \in \mathbb{N}^+$ *and* $\varnothing \neq A_C \subseteq \mathsf{A}_C$ *is either a set of* starting *actions or a set of* ending *actions.*

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

We denote by M_C and M_E the set of moves playable by *Charlie* and *Eve*, respectively. Moves such as $\mathrm{play}(A_C)$ and $\mathrm{play}(\delta_E, A_E)$ can play either $\mathrm{start}(x,v)$ actions only or $\mathrm{end}(x,v)$ 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. We consider wait moves as *ending* moves. Moreover, Starting and ending moves have to be alternated during the game.

Definition 7 (Round)**.** *A* round ρ *is a pair* $(\mu_C, \mu_E) \in \mathsf{M}_C \times \mathsf{M}_E$ *of moves such that:*

1. μ_C *and* μ_E *are either both* starting *or both* ending *moves;*
2. *either* $\rho = (\mathrm{play}(A_C), \mathrm{play}(A_E))$*, or* $\rho = (\mathrm{wait}(\delta_C), \mathrm{play}(\delta_E, A_E))$*, 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. Also note that since $\mathrm{play}(\delta_E, A_E)$ moves are played only in rounds together with $\mathrm{wait}(\delta_C)$, and $\mathrm{wait}(\delta_C)$ is always an ending move, then any $\mathrm{play}(\delta_E, A_E)$ must be an ending move. We can now define how a round is applied to the current partial plan to obtain the new one. The game always starts with a single *starting round*.

Definition 8 (Outcome of rounds)**.** *Let* $\overline{\mu} = \langle \mu_1, \ldots, \mu_n \rangle$ *be an event sequence, with* $\mu_n = (A_n, \delta_n)$ *or* $\mu_n = (\varnothing, 0)$ *if* $\overline{\mu} = \varepsilon$*. Let* $\rho = (\mu_C, \mu_E)$ *be a round, let* δ_E *and* δ_C *be the time increments of the moves, with* $\delta_C = \delta_E = 1$ *for* $\mathrm{play}(A)$ *moves, and let* A_E *and* A_C *be the set of actions of the two moves (*A_C *is empty if* μ_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 2;*

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 $\overline{\mu}$.

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.

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

Definition 9 (Strategies)**.** *A* strategy for Charlie *is a function* $\sigma_C : \Pi \to \mathsf{M}_C$ *that maps any given partial plan* $\overline{\mu}$ *to a move* μ_C *applicable to* $\overline{\mu}$*. A* strategy for Eve *is a function* $\sigma_E : \Pi \times \mathsf{M}_C \to \mathsf{M}_E$ *that maps a partial plan* $\overline{\mu}$ *and a move* $\mu_C \in \mathsf{M}_C$ *applicable to* $\overline{\mu}$*, to a* μ_E *such that* $\rho = (\mu_C, \mu_E)$ *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 $\rho_i = (\sigma_C(\Pi_{i-1}), \mu_E^i)$, for some μ_E^i, for all $0 < i \le n$, and to be played according to some strategy σ_E for *Eve* if $\rho_i = (\mu_C^i, \sigma_E(\Pi_{i-1}, \mu_C^i))$, for all $0 < i \le n$. It can be seen that for any pair of strategies (σ_C, σ_E) and any $n \ge 0$, there is a unique run $\overline{\rho}_n(\sigma_C, \sigma_E)$ of length n played according both to σ_C and σ_E.

Then, we say that a partial plan $\overline{\mu}$, and the play $\overline{\rho}$ such that $\overline{\mu} = \overline{\rho}(\varepsilon)$, are *admissible*, if the partial plan satisfies the domain rules, and are *successful* if the partial plan satisfies the system rules.

Definition 10 (Admissible strategy for *Eve*)**.** *A strategy* σ_E *for* Eve *is* admissible *if for each strategy* σ_C *for* Charlie*, there is* $k \ge 0$ *such that the play* $\overline{\rho}_k(\sigma_C, \sigma_E)$ *is admissible.*

Charlie wins if, *assuming* domain rules are respected, he manages to satisfy the system rules no matter how *Eve* plays.

Definition 11 (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* $n \ge 0$ *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 for *Charlie* does not exist.

3 A deterministic automaton for timeline-based planning

In this section we encode a timeline-based planning problem into a *deterministic* finite state automaton (DFA) that recognises all and only those event sequences that represent solution plans for such problem. This automaton will form the basis for the game arena solved in the next section. The words accepted by the automaton are *event sequences* representing solution plans.

Let $P = (\mathsf{SV}, S)$ be a timeline-based planning problem. To get a finite alphabet, we define $d = \max(L, U) + 1$, where L and U are in turn the *maximum* lower and (finite) upper bounds appearing in any rule of P, and we account only for event sequences such that the distance between two consecutive events is at most d. It can be easily seen that this assumption does not loose generality (for a proof, see Lemma 4.8 in [15]). Hence, the symbols of the alphabet Σ are *events* of the form $\mu = (A, \delta)$, where $A \subseteq \mathsf{A}_{\mathsf{SV}}$ and $1 \le \delta \le d$. Formally, $\Sigma = 2^{\mathsf{A}_{\mathsf{SV}}} \times [d]$, where $[d] = \{1, \dots, d\}$. Note that the size of Σ is exponential in the size of the problem. Moreover, we define the amount $\mathrm{window}(P)$ as the product of all the non-zero coefficients appearing as upper bounds in rules of P. Intuitively, $\mathrm{window}(P)$ is the maximum amount of time a rule of P can *count* far away from the occurrence of the quantified tokens. For example, consider the following rule:

$$a_0[x_0 = v_0] \to \exists a_1[x_1 = v_1] a_2[x_2 = v_2] a_3[x_3 = v_3] \,. \\ \mathrm{start}(a_1) \le_{[4,14]} \mathrm{end}(a_0) \wedge \mathrm{end}(a_0) \le_{[0,+\infty]} \mathrm{end}(a_2) \wedge \mathrm{start}(a_2) \le_{[0,3]} \mathrm{end}(a_3)$$

	start(a_0)	end(a_0)	start(a_1)	end(a_1)	start(a_2)	end(a_2)	start(a_3)	end(a_3)
start(a_0)								
end(a_0)			-4					
start(a_1)		14						
end(a_1)								
start(a_2)								3
end(a_2)		0						
start(a_3)								
end(a_3)					0			

Figure 1: DBM of an example synchronization rule. Missing entries are intended to be $+\infty$.

In this case, window(P) (assuming this is the only rule of the problem), would be $3 \cdot 14 = 42$. This means the rule can precisely account for what happens at most 42 time point from the occurrence of its quantified tokens. For example, if the token a_1 appears at a given distance from a_0, it has to be at less than 42 time points (less than 14, in particular), and any modification of the plan that changes such distance has the potential to break the satisfaction of the rule. Instead, what happens further away from a_0 only affects the satisfaction of the rule *qualitatively*. Suppose the tokens a_2 and a_3 lie at 100 time points from a_0 (at most 3 time steps from each other). Changing this distance (while maintaining the qualitative order between tokens) cannot ever break the satisfaction of the rule. See [15] for a precise account of the properties of window(P).

A key observation underlying our construction is that every atomic temporal relation $T \leq_{l,u} T'$ can be rewritten as the conjunction of two inequalities $T' - T \leq u$ and $T - T' \leq -l$, and that the clause C of an existential statement E can be rewritten as a system of difference constraints $\nu(\mathsf{C})$ of the form $T - T' \leq n$, with $n \in \mathbb{Z}_{+\infty}$. Then, the system $\nu(\mathsf{C})$ can be conveniently represented by a squared matrix D indexed by terms, where the entry associated with $D[T,T']$ gives the upper bound on $T - T'$. Such matrices, which take the name of *Difference Bound Matrices* (DBMs) [12, 20], can be conveniently updated as the plan evolves to keep track of the satisfaction of the atomic temporal relations among terms. In building a DBM for the system of constraints $\nu(\mathsf{C})$, we augment the system with constraints of kind $\text{start}(a_i) - \text{end}(a_i) \leq -d_{min}^{x_i = v_i}$ and $\text{end}(a_i) - \text{start}(a_i) \leq d_{max}^{x_i = v_i}$, for any quantified token $a_i[x_i = v_i]$ of E. Moreover, if two different bounds $T - T' \leq n$ and $T - T' \leq n'$ with $n' < n$ belong to $\nu(\mathsf{C})$, we keep only $T - T' \leq n'$. As an example, the DBM for the existential graph of the rule above is the one in Fig. 1. Note that, when the bounds of the temporal relations are translated into a DBM, there is no longer a distinction between *lower* and *upper* bounds. However, for some of the entries we can retrieve their original meaning. Indeed, if $D[T,T'] < 0$, then such entry is the lower bound of a temporal relation $T \leq_{l,u} T'$, whereas, if $D[T,T'] > 0$, it is the upper bound of a relation $T' \leq_{l,u} T$.

On top of DBMs, we define the concept of *matching structure*, a data structure that allows us to manipulate and reason about partially matched existential statements, *i.e.*, existential statements of which only a part of the requests has already been satisfied by the part of the word already read, while the rest can be still potentially matched in the future.

Definition 12 (Matching Structure)**.** *Let* $\mathsf{E} \equiv \exists a_1[x_1 = v_1] \ldots a_m[x_m = v_m] \,.\, \mathsf{C}$ *be the existential statement of a synchronisation rule* $\mathsf{R} \equiv a_0[x_0 = v_0] \rightarrow \mathsf{E}_1 \vee \cdots \vee \mathsf{E}_k$ *over the set of state variables* SV.

The matching structure *for* E *is a tuple* $\mathsf{M}_{\mathsf{E}} = (V, D, M, t)$ *where:*

- *V is the set of terms* $\mathrm{start}(a)$ *and* $\mathrm{end}(a)$ *for* $a \in \{a_0, \dots, a_m\}$*;*
- $D \in \mathbb{Z}_{+\infty}^{|V|^2}$ *is a DBM indexed by terms of V where* $D[T,T'] = n$ *if* $(T - T' \leq n) \in \nu(\mathsf{C})$*,* $D[T,T'] = 0$ *if* $T = T'$*, and* $D[T,T'] = +\infty$ *otherwise;*
- $M \subseteq V$ *and* $0 \leq t \leq \mathrm{window}(P)$.

The set M contains the terms of V that the matching structure has correctly matched over the event sequence read so far. With $\overline{M} = V \setminus M$ we denote the actions that we have yet to see. Then, we say that a matching structure M is *closed* if $M = V$, it is *initial* if $M = \varnothing$ and it is *active* if it is not closed and $\mathrm{start}(a_0) \in M$. Intuitively, a matching structure is *active* if its trigger has been matched over the word the automaton is reading. Then, when all the terms have been matched over the word, the matching structure becomes *closed*. The component t is the time elapsed since $\mathrm{start}(a_0)$ has been matched. When time flows, a matching structure can then be updated as follows.

Definition 13 (Time shifting)**.** *Let $\delta > 0$ be a positive amount of time, and $\mathsf{M} = (V,D,M,t)$ be a matching structure. The result of shifting M by δ time units, written $\mathsf{M} + \delta$, is the matching structure $\mathsf{M}' = (V,D',M,t')$ where:*

- *for all $T,T' \in V$:*

$$D'[T,T'] = \begin{cases} D[T,T'] + \delta & \text{if } T \in M \text{ and } T' \in \overline{M} \\ D[T,T'] - \delta & \text{if } T \in \overline{M} \text{ and } T' \in M \\ D[T,T'] & \text{otherwise} \end{cases}$$

- *and*

$$t' = \begin{cases} t + \delta & \text{if } \mathsf{M} \text{ is active} \\ t & \text{otherwise} \end{cases}$$

Definition 14 (Matching)**.** *Let $\mathsf{M} = (V,D,M,t)$ be a matching structure and $I \subseteq \overline{M}$ a set of matched terms. A matching structure $\mathsf{M}' = (V,D,M',t)$ is the result of matching the set I, written $\mathsf{M} \cup I$, if $M' = M \cup I$.*

Beside updating the reference t to the trigger occurrence of an active matching structure, Definition 13 dictates how to update the entries of the DBM. In particular, the distance bounds between any pair of terms T and T' where one is in M and the other is not are tighten by the elapsing of time: when $T \in M$ and $T' \in \overline{M}$, $D[T,T']$ is a lower bound loosen by adding the elapsed time δ, when $T \in \overline{M}$ and $T' \in M$, $D[T,T']$ is an upper bound tighten by subtracting δ. For example, consider the DBM in Fig. 1 and consider the pair of terms $\mathrm{start}(a_1)$ and $\mathrm{end}(a_0)$. $D[\mathrm{start}(a_1), \mathrm{end}(a_0)] = -4$, meaning that $\mathrm{end}(a_0) - \mathrm{start}(a_1) \leq 4$ must hold. Suppose $\mathrm{start}(a_1) \in M$ (*i.e.*, it has been matched), and $\mathrm{end}(a_0) \notin M$ (it still has to). Now, if 1 time point passes, the entry in the DBM is incremented and updated to $-4 + 1 = -3$, which corresponds to the constraint $\mathrm{end}(a_0) - \mathrm{start}(a_1) \leq 3$. This reflects the fact that to be able to satisfy the constraint, $\mathrm{end}(a_0)$ has now only 3 time steps left before it is too late. Definition 14 tells us how to update the set M of a matching structure.

To correctly match an existential statement while reading an event sequence, a matching structure is updated only as long as no violations of temporal constraints are witnessed. As such, an event is classified from the standpoint of a matching structure as *admissible* or not.

Definition 15 (Admissible Event)**.** *An event $\mu = (A, \delta)$ is* admissible *for a matching structure $\mathsf{M}_{\mathsf{E}} = (V,D,M,t)$ if and only if, for every $T \in M$ and $T' \in \overline{M}$, $\delta \leq D[T',T]$,* i.e.*, the elapsing of δ time units does not exceed the upper bound of some term T' not yet matched by M_{E}.*

Each admissible event μ read from the word can be matched with a subset of the terms of the matching structure. There are usually more than one way to match events and terms. The following definition makes this choice explicit.

Definition 16 (*I*-match Event)**.** *Let* $\mathsf{M}_\mathsf{E} = (V, D, M, t)$ *be a matching structure and* $I \subseteq \overline{M}$*. An* I-match event *is an admissible event* $\mu = (A, \delta)$ *for* M_E *such that:*

1. *for all token names* $a \in \mathsf{N}$ *quantified as* $a[x = v]$ *in* E *we have that:*
   (a) *if* $\mathrm{start}(a) \in I$*, then* $\mathrm{start}(x, v) \in A$*;*
   (b) $\mathrm{end}(a) \in I$ *if and only if* $\mathrm{start}(a) \in M$ *and* $\mathrm{end}(x, v) \in A$*;*
2. *and for all* $T \in I$ *it holds that:*
   (a) *for every other term* $T' \in V$*, if* $D[T', T] \leq 0$*, then* $T' \in M \cup I$*;*
   (b) *for all* $T' \in M$*,* $\delta \geq -D[T', T]$, i.e.*, all the lower bounds on* T *are satisfied;*
   (c) *for each other term* $T' \in I$*, either* $D[T', T] = 0$*,* $D[T, T'] = 0$*, or* $D[T', T] = D[T, T'] = +\infty$*.*

Intuitively, an event is an I-match event if the actions in the event correctly match the terms in I. Item 1 ensures that each term is correctly matched over an action it represents and, most importantly, that the endpoints of a quantified token correctly identify the endpoints of a token in the event sequence. Item 2 ensures that matching the terms in I does not violate any atomic temporal relation. In particular, Item 2a deals with the qualitative aspect of an "happens before" relation, while Items 2b and 2c deal with the quantitative aspects of the lower bounds of these relations. Note that an $\varnothing$-event is admitted.

Let $\mathbb{M}_P$ be the set of all the matching structures for a planning problem P. By Definition 16, a single event can represent several I-match events for a matching structure, hence a matching structure can evolve into several matching structures, one for each I-match event. Such evolution is defined as a ternary relation $S \subseteq \mathbb{M}_P \times \Sigma \times \mathbb{M}_P$ such that $(\mathsf{M}, (A, \delta), \mathsf{M}') \in S$ if and only if (A, δ) is an I-match event for M and $\mathsf{M}' = (\mathsf{M} + \delta) \cup I$. To deal with the nondeterministic nature of this relation, states of the automaton will comprise sets of matching structures collecting all the possible outcomes of S, so that suitable notation for working with sets of matching structures, denoted by Υ hereafter, is introduced. We define $\Upsilon_t^\mathsf{R} \subseteq \Upsilon$ as the set of all the *active* matching structures $\mathsf{M} \in \Upsilon$ with timestamp t, associated with any existential statement of R. Intuitively, matching structures in Υ_t^R contribute to the fulfilment of the same triggering event for the rule R (because they have the same timestamp), regardless of the existential statement they represent. We also define $\Upsilon_\perp \subseteq \Upsilon$ as the set of *non active* matching structures of Υ. A set Υ is *closed* if there exists $\mathsf{M} \in \Upsilon$ such that M is *closed*. Lastly, a function step_μ extends the relation S to sets of matching structures: $\mathrm{step}_\mu(\Upsilon) = \{\, \mathsf{M}' \mid (\mathsf{M}, \mu, \mathsf{M}') \in S, \text{ for some } \mathsf{M} \in \Upsilon \,\}$.

We are now ready to define the automaton. If E is an existential statement, let $\mathbb{E}_\mathsf{E}$ be the set of all the existential statements of the same rule of E. Let $\mathbb{F}_P$ be the set of functions mapping each existential statement of P to a set of existential statements, and let $\mathbb{D}_P$ be the set of functions mapping each existential statement to a set of matching structures. A simple automaton T_P that checks the transition function and duration functions of the variables is easy to define. Then, given a timeline-based planning problem $P = (\mathsf{SV}, S)$, the corresponding automaton is $A_P = \mathsf{T}_P \cap \mathsf{S}_P$ where S_P, the automaton that checks the satisfaction of the synchronization rules, is defined as $\mathsf{S}_P = (Q, \Sigma, q_0, F, \tau)$, where:

1. $Q = 2^\mathbb{M} \times \mathbb{D} \times \mathbb{F} \cup \{\perp\}$ is the finite set of states, *i.e.*, states are tuples of the form $\langle \Upsilon, \Delta, \Phi \rangle \in 2^\mathbb{M} \times \mathbb{D} \times \mathbb{F}$, plus a sink state $\perp$;
2. Σ is the input alphabet defined above;
3. the initial state $q_0 = \langle \Upsilon_0, \Delta_0, \Phi_0 \rangle$ is such that Υ_0 is the set of initial matching structures of the existential statements of P and, for all existential statements E of P, we have $\Delta_0(\mathsf{E}) = \varnothing$ and $\Phi_0(\mathsf{E}) = \mathbb{E}_\mathsf{E}$;

4. $F \subseteq Q$ is the set of final states defined as:

$$F = \left\{ \langle \Upsilon, \Delta, \Phi \rangle \in Q \;\middle|\; \begin{array}{l} \mathsf{M} \text{ is not } \textit{active} \text{ for all } \mathsf{M} \in \Upsilon \\ \text{and } \Delta(\mathsf{E}) = \varnothing \text{ for all } \mathsf{E} \text{ of } P \end{array} \right\}$$

5. $\tau : Q \times \Sigma \to Q$ is the transition function that given a state $q = \langle \Upsilon, \Delta, \Phi \rangle$ and a symbol $\mu = (A, \delta)$ computes the new state $\tau(q, \mu)$. Let $\mathrm{step}^{\mathsf{E}}_{\mu}(\Upsilon^{\mathsf{R}}_t) = \{ \mathsf{M}_{\mathsf{E}} \mid \mathsf{M}_{\mathsf{E}} \in \mathrm{step}_{\mu}(\Upsilon^{\mathsf{R}}_t) \}$. Moreover, let $\Psi^{\mathsf{R}}_t = \{ \mathsf{E} \mid \mathsf{M}_{\mathsf{E}} \in \mathrm{step}_{\mu}(\Upsilon^{\mathsf{R}}_t) \}$. Then, the updated components of the state are based on what follows, where $W = \mathrm{window}(P)$:

$$\Upsilon' = \mathrm{step}_{\mu}(\Upsilon_{\perp}) \cup \bigcup \left\{ \mathrm{step}_{\mu}(\Upsilon^{\mathsf{R}}_t) \;\middle|\; t < W - \delta \text{ and } \mathrm{step}_{\mu}(\Upsilon^{\mathsf{R}}_t) \text{ is not } \textit{closed} \right\}$$

$$\Delta'(\mathsf{E}) = \begin{cases} \mathrm{step}^{\mathsf{E}}_{\mu}(\Upsilon^{\mathsf{R}}_t) & \text{where } t \text{ is the minimum such that } t \geq W - \delta \text{ and } \mathrm{step}^{\mathsf{E}}_{\mu}(\Upsilon^{\mathsf{R}}_t) \neq \varnothing \\ \mathrm{step}_{\mu}(\Delta(\mathsf{E})) & \text{if such } t \text{ does not exist} \end{cases}$$

$$\Phi'(\mathsf{E}) = \begin{cases} \mathbb{E}_{\mathsf{E}} \quad \text{if } \mathsf{E} \in \Psi(\mathsf{E}') \text{ for some } \mathsf{E}' \text{ such that } \Delta'(\mathsf{E}') \text{ is } \textit{closed} \\ \Phi(\mathsf{E}) \setminus \{ \mathsf{E}' \mid \exists t \geq W - \delta \,.\, \mathsf{E}' \in \Psi^{\mathsf{R}}_t \wedge \mathsf{E} \notin \Psi^{\mathsf{R}}_t \} \quad \text{otherwise} \end{cases}$$

Let $\Delta''(\mathsf{E}) = \Delta'(\mathsf{E})$ unless there is an E' with $\mathsf{E} \in \Phi'(\mathsf{E}')$ such that $\Delta'(\mathsf{E}')$ is *closed*, in which case $\Delta''(\mathsf{E}) = \varnothing$. Then, $\tau(q, \mu) = \langle \Upsilon', \Delta'', \Phi' \rangle$ if the following holds:

(a) for every Υ^{R}_t, $\mathrm{step}_{\mu}(\Upsilon^{\mathsf{R}}_t) \neq \varnothing$, and

(b) for every synchronisation rule $\mathsf{R} \equiv a_0[x_0 = v_0] \to \mathsf{E}_1 \vee \cdots \vee \mathsf{E}_n$ in S, if $\mathrm{start}(x_0, v_0) \in A$, then there exists $\mathsf{M}_{\mathsf{E}_i} = (V, D, M, 0) \in \Upsilon'$, with $i \in \{1 \ldots n\}$, such that $\mathrm{start}(a_0) \in M$;

Otherwise, $\tau(q, \mu) = \perp$.

Let us explain what is going on. The first component Υ of an automaton state $q = (\Upsilon, \Delta, \Phi)$ is a set of matching structures that keeps track of what have been tracked so far. Intuitively, the automaton precisely keeps track of what happened to the last $\mathrm{window}(P)$ time points, and only summarizes what happened before that window, which is what allows us to keep the size under control. Any matching structure in Υ has $t < \mathrm{window}(P)$. Matching structures in Υ evolve following the step function, until they are closed or the t component reaches $\mathrm{window}(P)$. Matching structures that reach $\mathrm{window}(P)$ are promoted to a new role. Their new task is to record the pieces of existential statements that still have to be matched in order to satisfy all the trigger events of R that no longer fit into the *recent history* of the event sequence (*i.e.*, the last $\mathrm{window}(P)$ time points). These matching structures are not stored in Υ though, they are summarized by the function Δ that maps each existential statement E of a rule R to the set of matching structures for E with $t = \mathrm{window}(P)$.

When a set Υ^{R}_t exceeds the bound $\mathrm{window}(P)$, the Δ function must be updated by merging the information of Υ^{R}_t to the information already present in Δ. Now, it has to be noted that, by closing a set $\Delta(\mathsf{E})$, we can not conclude that every event that triggered R actually satisfies R. Indeed, there can be sets $\Delta(\mathsf{E})$ and $\Delta(\mathsf{E}')$ that are in charge of the satisfaction of the same rule R, but for different trigger events, and closing $\Delta(\mathsf{E})$ does not imply that R has been satisfied. The opposite case may also arise, in which $\Delta(\mathsf{E})$ and $\Delta(\mathsf{E}')$ contribute to the fulfilment of the same trigger events and closing either set suffices to satisfy R. To overcome the information lost when a set of matching structures gets added to the Δ function, the Φ function (the third component of the automaton states) maps each existential statement E to the set of existential statements E' such that $\Delta(\mathsf{E}')$ tracks the fulfilment of the same trigger events of the set $\Delta(\mathsf{E})$. We use Φ as follows: when a set $\Delta(\mathsf{E})$ gets closed, we can discard its matching structures and all the matching structures of the sets $\Delta(\mathsf{E}')$, with $\mathsf{E}' \in \Phi(\mathsf{E})$.

One can prove the soundness and completeness of our construction.

Theorem 1. *(Soundness and completeness) Let $P = (\mathsf{SV}, S)$ be a timeline-based planning problem and let A_P be the associated automaton. Then, any event sequence $\overline{\mu}$ is a solution plan for P if and only if $\overline{\mu}$ is accepted by A_P.*

Recall that we assumed the timestamp of each event of event sequences to be bounded, but since events can have an empty set of actions, Theorem 1 can actually deal with arbitrary event sequences, after adding suitable empty events. Now, let us look at the size of the automaton. Let E be the overall number of existential statements in P, which is linear in the size of P. It can be seen that $|\mathbb{D}_P| \in \mathscr{O}((2^{|\mathbb{M}_P|})^E) = \mathscr{O}(2^{E\cdot|\mathbb{M}_P|})$, *i.e.*, the number of Δ functions is doubly exponential in the size of P. Then, observe that $|\mathbb{F}_P| \in \mathscr{O}((2^E)^E) = \mathscr{O}(2^{E^2})$. Then, $|\mathsf{S}_P| \in \mathscr{O}(|\Sigma|\cdot 2^{|\mathbb{M}_P|})$, that is, the size of S_P is at most exponential in the number of possible matching structures. To bound this number, let N be the largest finite constant appearing in P as bound in any atom or value duration function and let L be the length of the largest existential prefix of an existential statement occurring inside a rule of P. Notice that N is exponential in the size of P, since constants are expressed in binary, while $L \in \mathscr{O}(|P|)$. Then, the entries of a DBM for P, of which there is a number quadratic in L, are constrained to take values within the interval $[-N, N]$ (excluding the infinitary value $+\infty$), whose size is linear in N. By Definition 12, it follows that, for the planning problem P, $|\mathbb{M}_P| \in \mathscr{O}(N^{L^2}\cdot 2^L \cdot \mathrm{window}(P))$, *i.e.*, the number of matching structures is at most exponential in the size of P. Hence, we proved the following:

Theorem 2 (Size of the automaton)**.** *Let $P = (\mathsf{SV}, S)$ be a timeline-based planning problem and let A_P be the associated automaton. Then, the size of A_P is at most doubly-exponential in the size of P.*

Note that this is the same size as the automaton built by Della Monica *et al.* [9], but their automaton was *nondeterministic*, while ours is by construction deterministic, essential for its use as a game arena.

4 Controller synthesis

In this section we use the deterministic automaton constructed above to obtain a deterministic arena where we can solve a simple reachability game for checking the existence of (and, in this case, to synthesize) a controller for the corresponding timeline-based game.

4.1 From the automaton to the arena

Let $G = (\mathsf{SV}_C, \mathsf{SV}_E, \mathsf{S}, \mathsf{D})$ be a timeline-based game. We use the construction of the automaton explained in the previous section in order to obtain a game arena. However, the automaton construction considers a planning problem with a single set of synchronization rules, while here we have to account for the roles of both S and D.

To do that, let A_S and A_D be the deterministic automata built over the timeline-based planning problem $P_\mathsf{S} = (\mathsf{SV}_C \cup \mathsf{SV}_E, \mathsf{S})$ and $P_\mathsf{D} = (\mathsf{SV}_C \cup \mathsf{SV}_E, \mathsf{D})$, respectively. We define the automaton A_G as $\overline{A_\mathsf{D}} \cup A_\mathsf{S}$, *i.e.*, the union of A_S with the complement of A_D. Note that these are all standard automata-theoretic constructions over DFAs. Any accepting run of A_G represents either a plan that violates the domain rules or a plan that satisfies both the domain and the system rules, in conformance with Definition 11. Note that A_G is deterministic and can be built from A_D and A_S with only a polynomial increase in size.

Now, the A_G automaton is still not suitable as a game arena, because the moves of the timeline-based game are not directly visible in the labels of the transitions. In other words, the A_G automaton reads events, while we need an automaton that reads game *moves*. In particular, a single transition in

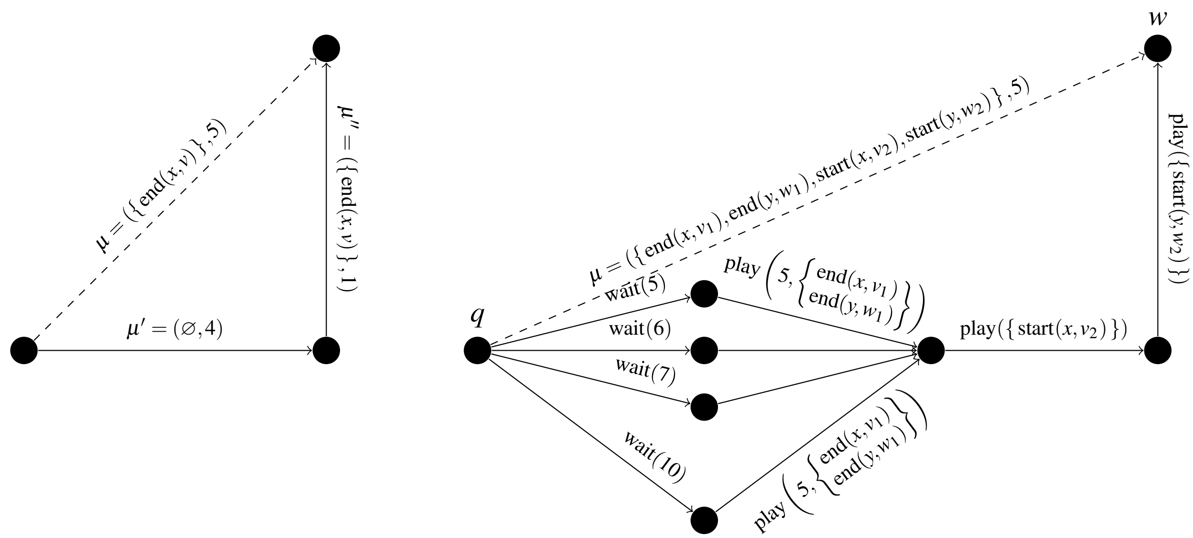

Figure 2: On the left, the removal of transitions $\mu = (A,\delta)$ with $\delta > 1$ and ending actions of controllable tokens in A. On the right, the transformation of a transition of the A_G into a sequence of transitions in A_G^a, with $x \in \mathsf{SV}_C$, $y \in \mathsf{SV}_E$, and $\gamma_x(v_1) = \gamma_y(w_1) = \mathsf{u}$.

the automaton can correspond to different combinations of rounds, since the presence of wait(δ) moves is not explicit in the transition. For example, an event $\mu = (A,5)$ can be the result of a wait(5) move by *Charlie* followed by a play$(5,A)$ move by *Eve*, or by any wait(δ) move with $\delta > 5$ followed by play$(5,A)$. Hence, we need to further adapt A_G to obtain a suitable arena.

Let $A_G = (Q,\Sigma,q_0,F,\tau)$ be the automaton built as described before. Let $\mu = (A,\delta)$ be an event. If $\delta > 1$, this transition must have resulted from *Charlie* playing a wait(δ') move with $\delta' \geq \delta$. However, if A contains any end(x,v) action with $x \in \mathsf{SV}_C$, this is for sure the result of more than one pair of starting/closing rounds. In order to simplify the construction below, we remove this possibility beforehand. More formally, we define a slightly different automaton $A'_G = (Q,\Sigma,q_0,F,\tau')$ where τ' is now a *partial* transition function (*i.e.*, the automaton becomes *incomplete*) that agrees with τ on everything excepting that transitions $\tau(q,(\mathsf{A},\delta))$ is *undefined* if $\delta > 1$ and A contains any end(x,v) action with $x \in \mathsf{SV}_C$. You can see an example of this operation in Fig. 2, on the left. Note that this removal does not change the plans accepted by the automaton because for each transition $\tau(q,(\mathsf{A},\delta)) = q'$ with $\delta > 1$ there are two transitions $\tau(q,(\varnothing,\delta-1)) = q''$ and $\tau(q'',(\mathsf{A},1)) = q'$.

Now we can transform the automaton in order to make the game rounds, and especially wait(δ) moves, explicit. Intuively, each transition of the automaton is split into four transitions explicitating the four moves of the two rounds. Given the automaton $A'_G = (Q,\Sigma,q_0,F,\tau')$, we define the automaton $A_G^a = (Q^a,\Sigma^a,q_0^a,F^a,\tau^a)$, which will be the arena of our game, as follows:

1. $Q^a = Q \cup \{\, q_\delta \mid 1 \leq \delta \leq d \,\} \cup \{\, q_{\delta,A} \mid 1 \leq \delta \leq d, A \subseteq \mathsf{A} \,\}$ is the set of states;
2. $\Sigma^a = \mathsf{M}_C \cup \mathsf{M}_E$, *i.e.*, the alphabet is turned into the set of moves of the two players;
3. $q_0^a = q_0$ and $F^a = F$, *i.e.*, initial and final states do not change;
4. the (partial) transition function τ^a is defined as follows. Let $w = \tau(q,\mu)$ with $\mu = (\mathsf{A},\delta)$. We distinguish the case where $\delta = 1$ or $\delta > 1$.

(a) if $\delta = 1$, let $\mathsf{A}_C \subseteq \mathsf{A}$ and $\mathsf{A}_E \subseteq \mathsf{A}$ be the set of actions in A playable by *Charlie* and by *Eve*, respectively. Then:

i. $\tau(q, \text{play}(\mathsf{A}_C^e)) = q_{1,\mathsf{A}_C^e}$, where A_C^e is the set of *ending* actions in A_C;
ii. $\tau(q_{1,\mathsf{A}_C^e}, \text{play}(\mathsf{A}_E^e)) = q_{1,\mathsf{A}_C^e \cup \mathsf{A}_E^e}$, where A_E^e is the set of *ending* actions in A_E;
iii. $\tau(q_{1,\mathsf{A}_C^e \cup \mathsf{A}_E^e}, \text{play}(\mathsf{A}_C^s)) = q_{1,\mathsf{A}_C^e \cup \mathsf{A}_E^e \cup \mathsf{A}_C^s}$, where A_C^s is the set of *starting* actions in A_C;
iv. $\tau(q_{1,\mathsf{A}_C^e \cup \mathsf{A}_E^e \cup \mathsf{A}_C^s}, \text{play}(\mathsf{A}_E^s)) = w$, where A_E^s is the set of *starting* actions in A_E;

where the mentioned states are added to Q^a as needed.

(b) if $\delta > 1$, let $\mathsf{A}_C \subseteq \mathsf{A}$ and $\mathsf{A}_E \subseteq \mathsf{A}$ be the set of actions in A playable by *Charlie* and by *Eve*, respectively. Note that by construction, A_C only contains *starting* actions. Then:

i. $\tau(q, \text{wait}(\delta_C)) = q_{\delta_C}$ for all $\delta \le \delta_C \le d$;
ii. $\tau(q_{\delta_C}, \text{play}(\delta, \mathsf{A}_E^e)) = q_{\delta,\mathsf{A}_E^e}$ where A_E^e is the set of *ending* actions in A_E;
iii. $\tau(q_{\delta,\mathsf{A}_E^e}, \text{play}(\mathsf{A}_C)) = q_{\delta,\mathsf{A}_E^e \cup \mathsf{A}_C}$;
iv. $\tau(q_{\delta,\mathsf{A}_E^e \cup \mathsf{A}_C}, \text{play}(\mathsf{A}_E^s)) = w$ where A_E^s is the set of *starting* actions in A_E;

where the mentioned states are added to Q^a as needed.

All the transitions not explicitly defined above are undefined.

A graphical example of the above construction can be seen in Fig. 2, on the right. Note that the structure of the original A_G automaton is preserved by A_G^a. In particular, one can see that for each $q \in Q$ and event $\mu = (A, \delta)$, any sequence of moves whose outcome would append μ to the partial plan (see Definition 8) reach from q the same state w in A_G^a that is reached in A_G by reading μ. Hence, one can consider A_G^a to also being able to *read* event sequences, even though its alphabet is different. We denote as $[\overline{\mu}]$ the state $q \in Q^a$ reached by reading $\overline{\mu}$ in A_G^a.

Moreover, note that, with a minimal abuse of notation, any play $\overline{\rho}$ for the game G can be seen as a word readable by the automaton A_G^a. Hence, we can prove the following.

Theorem 3. *If G is a timeline-based game, for any play $\overline{\rho}$ for G, $\overline{\rho}$ is successful if and only if it is accepted by A_G^a.*

4.2 Computing the Winning Strategy

Once built the arena, we can focus on computing the winning region W_C for *Charlie*, that is, the set of states of the arena from which *Charlie* can force the play to reach a final state of A_G^a, no matter of the strategy of *Eve*. These games are called *reachability games* [21]. If the winning region W_C is not empty, a winning strategy of *Charlie* can be simply derived from W_C. As a consequence of Theorems 1 and 3, the computed winning strategy σ_C for A_G^a respects Definition 11.

As stated in [21, Theorem 4.1], rechability games are determined, and the winning region W_C along with the corresponding positional winning strategy s are computable. Let $A_G^a = (Q^a, \Sigma^a, q_0^a, F^a, \tau^a)$ be the automaton built from G as described in the previous section. Note that, by construction, in any state $q \in Q^a$ only one of the players has available moves. Let $Q_C^a \subseteq Q^a$ be the set of states *belonging* to *Charlie*, *i.e.*, states from which *Charlie* can move, and let $Q_E^a = Q^a \setminus Q_C^a$. Moreover, let $E = \{ (q, q') \in Q^a \times Q^a \mid \exists \mu \,.\, \tau^a(q, \mu) = q' \}$, *i.e.*, the set of all the edges of A_G^a.

Now, for each $i \ge 0$, we can compute the i-th attractor of F^a, written $Attr_C^i(F^a)$, that is, the set of states from which *Charlie* can win in at most *i* steps. $Attr_C^i(F^a)$ is defined as follows:

$$\begin{aligned}
Attr_C^0(F^a) &= F^a \\
Attr_C^{i+1}(F^a) &= Attr_C^i(F^a) \\
&\quad \cup \{\, q^a \in Q_C^a \mid \exists r\big((q^a,r) \in E \wedge r \in Attr_C^i(F^a)\big) \,\} \\
&\quad \cup \{\, q^a \in Q_E^a \mid \forall r\big((q^a,r) \in E \to r \in Attr_C^i(F^a)\big) \,\}
\end{aligned}$$

As remarked in [21], the sequence $Attr_C^0(F^a) \subseteq Attr_C^1(F^a) \subseteq Attr_C^2(F^a) \subseteq \dots$ becomes stationary for some index $k \le |Q^a|$. Thus, we define $Attr_C(F^a) = \bigcup_{i=0}^{|Q^a|} Attr_C^i(F^a)$. In order to prove that $W_C = Attr_C(F^a)$, it suffices to use the proof of [21, Theorem 4.1] for showing that $Attr_C(F^a) \subseteq W_C$ and $W_C \subseteq Attr_C(F^a)$.

To compute a winning strategy for *Charlie* in the case that $q_0^a \in W_C$, it is sufficient to define $s(q) = \mu$ for any μ such that $\tau^a(q,\mu) = q'$ with $q,q' \in W_C$ (which is guaranteed to exist by construction of the attractor). Then, the strategy σ_C for *Charlie* in G (see Definition 11) is defined as $\sigma_C(\overline{\mu}) = s([\overline{\mu}])$.

Theorem 4. *Given* $A_G^a = (Q^a, \Sigma^a, q_0^a, F^a, \tau^a)$, $q_0^a \in W_C$ *if and only if* Charlie *has a winning strategy* σ_C *for* G.

Proof. We first prove *soundness*, that is, $q_0^a \in W_C$ implies that *Charlie* has a winning strategy σ_C for G. If $q_0^a \in W_C$, then it means that there exists a positional winning strategy s for *Charlie* for the reachability game over the arena A_G^a. By Theorem 3 and by the definition of reachability game, we know that each play generated by s corresponds to a successful play for game G. Let $\sigma_C(\overline{\mu}) = s([\overline{\mu}])$ be the winning strategy for *Charlie* in game G as defined above. By construction of σ_C and by Definition 11, this means that σ_C is a winning strategy of *Charlie* for G.

To prove *completeness* (*i.e.*, if *Charlie* has a winning strategy σ_C for G then $q_0^a \in W_C$), we proceed as follows. From Definition 11 we know that a winning strategy σ_C for *Charlie* is a strategy such that for every admissible strategy σ_E for *Eve*, there exists $n \ge 0$ such that the play $\overline{\rho}_n(\sigma_C, \sigma_E)$ is successful. From Theorem 3, we know that $\overline{\rho}_n(\sigma_C, \sigma_E)$ is accepted by A_G^a. Therefore, $\overline{\rho}_n(\sigma_C, \sigma_E)$ reaches a state in the set F^a starting from q_0^a. By definition of reachability game, this means that $q_0^a \in W_C$. □

5 Conclusions

In this paper, we completed the picture about timeline-based games by providing an effective procedure for controller synthesis, whereas before only a proof of the complexity of the strategy existence problem was known. Previous approaches either provided a deterministic concurrent game structure which was however not built effectively, or an effectively built automata which was, however, nondeterministic and thus unsuitable for use as a game arena without a costly determinization. Our approach surpasses the limits of both previous ones by providing a deterministic construction, of optimal asymptotic size, suitable to be used as a game arena. Then, we solve the reachability game on the arena with standard methods to effectively compute the winning strategy for the game, if it exists.

This work paves the way to interesting future developments. On the one hand, the effective procedure shown here can be finally implemented, bringing timeline-based games from theory to practice. On the other hand, developing an effective system based on such games requires to answer many interesting questions, from which concrete modeling language to adopt, to which algorithmic improvements are needed to make the approach feasible. For example, it can be foreseen that, to solve the fixpoint computation that leads to the strategy with reasonable performance, the application of *symbolic techniques* would be needed.

Acknowledgements

Nicola Gigante and Luca Geatti acknowledge the support of the Free University of Bozen-Bolzano, Faculty of Computer Science, by means of the projects TOTA (*Temporal Ontologies and Tableaux Algorithms*) and STAGE (*Synthesis of Timeline-based Planning Games*).

References

[1] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-time temporal logic*. J. ACM 49(5), pp. 672–713, doi:10.1145/585265.585270.

[2] Sara Bernardini & David E. Smith (2007): *Developing Domain-Independent Search Control for Europa2*. In: Proceedings of the ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning.

[3] Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron & Gerhard J. Woeginger (2020): *Timeline-based planning over dense temporal domains*. Theor. Comput. Sci. 813, pp. 305–326, doi:10.1016/j.tcs.2019.12.030.

[4] Amedeo Cesta, Gabriella Cortellessa, Michel Denis, Alessandro Donati, Simone Fratini, Angelo Oddi, Nicola Policella, Erhard Rabenau & Jonathan Schulster (2007): *Mexar2: AI Solves Mission Planner Problems*. IEEE Intelligent Systems 22(4), pp. 12–19, doi:10.1109/MIS.2007.75.

[5] Amedeo Cesta, Gabriella Cortellessa, Simone Fratini, Angelo Oddi & Nicola Policella (2006): *Software Companion: The Mexar2 Support to Space Mission Planners*. In Gerhard Brewka, Silvia Coradeschi, Anna Perini & Paolo Traverso, editors: Proceedings of the 17th European Conference on Artificial Intelligence, Frontiers in Artificial Intelligence and Applications 141, IOS Press, pp. 622–626.

[6] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins & D. Tran (2000): *ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling*. In: Proceedings of the International Conference on Space Operations.

[7] Steve A. Chien, Gregg Rabideau, Daniel Tran, Martina Troesch, Joshua Doubleday, Federico Nespoli, Miguel Perez Ayucar, Marc Costa Sitja, Claire Vallat, Bernhard Geiger, Nico Altobelli, Manuel Fernandez, Fran Vallejo, Rafael Andres & Michael Kueppers (2015): *Activity-Based Scheduling of Science Campaigns for the Rosetta Orbiter*. In Qiang Yang & Michael Wooldridge, editors: Proceedings of the 24th International Joint Conference on Artificial Intelligence, AAAI Press, pp. 4416–4422. Available at http://ijcai.org/Abstract/15/655.

[8] Marta Cialdea Mayer, Andrea Orlandini & Alessandro Umbrico (2016): *Planning and execution with flexible timelines: a formal account*. Acta Informatica 53(6-8), pp. 649–680, doi:10.1007/s00236-015-0252-z.

[9] Dario Della Monica, Nicola Gigante, Angelo Montanari & Pietro Sala (2018): *A Novel Automata-Theoretic Approach to Timeline-Based Planning*. In Michael Thielscher, Francesca Toni & Frank Wolter, editors: Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press, pp. 541–550. Available at https://aaai.org/ocs/index.php/KR/KR18/paper/view/18024.

[10] Dario Della Monica, Nicola Gigante, Angelo Montanari, Pietro Sala & Guido Sciavicco (2017): *Bounded Timed Propositional Temporal Logic with Past Captures Timeline-based Planning with Bounded Constraints*. In Carles Sierra, editor: Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 1008–1014, doi:10.24963/ijcai.2017/140.

[11] Dario Della Monica, Nicola Gigante, Salvatore La Torre & Angelo Montanari (2020): *Complexity of Qualitative Timeline-Based Planning*. In: Proceedings of the 27th International Symposium on Temporal Representation and Reasoning, LIPIcs 178, pp. 16:1–16:13, doi:10.4230/LIPIcs.TIME.2020.16.

[12] David L. Dill (1989): *Timing Assumptions and Verification of Finite-State Concurrent Systems*. In Joseph Sifakis, editor: Proceedings of the International Workshop on Automatic Verification Methods for Finite State

Systems, *Lecture Notes in Computer Science* 407, Springer, pp. 197–212, doi:10.1007/3-540-52148-8_17.

[13] Jeremy Frank & Ari K. Jónsson (2003): *Constraint-Based Attribute and Interval Planning*. *Constraints* 8(4), pp. 339–364, doi:10.1023/A:1025842019552.

[14] Simone Fratini, Amedeo Cesta, Andrea Orlandini, Riccardo Rasconi & Riccardo De Benedictis (2011): *APSI-based Deliberation in Goal Oriented Autonomous Controllers*. In: *ASTRA 2011*, 11, ESA.

[15] Nicola Gigante (2019): *Timeline-based Planning: Expressiveness and Complexity*. Ph.D. thesis, University of Udine, Italy. Available on *arXiv* at: https://arxiv.org/abs/1902.06123.

[16] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer & Andrea Orlandini (2016): *Timelines Are Expressive Enough to Capture Action-Based Temporal Planning*. In Curtis E. Dyreson, Michael R. Hansen & Luke Hunsberger, editors: *Proceedings of the 23rd International Symposium on Temporal Representation and Reasoning*, IEEE Computer Society, pp. 100–109, doi:10.1109/TIME.2016.18.

[17] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer & Andrea Orlandini (2017): *Complexity of Timeline-Based Planning*. In Laura Barbulescu, Jeremy Frank, Mausam & Stephen F. Smith, editors: *Proceedings of the 27th International Conference on Automated Planning and Scheduling*, AAAI Press, pp. 116–124. Available at https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15758.

[18] Nicola Gigante, Angelo Montanari, Andrea Orlandini, Marta Cialdea Mayer & Mark Reynolds (2020): *On timeline-based games and their complexity*. *Theor. Comput. Sci.* 815, pp. 247–269, doi:10.1016/j.tcs.2020.02.011.

[19] Nicola Muscettola (1994): *HSTS: Integrating Planning and Scheduling*. In Monte Zweben & Mark S. Fox, editors: *Intelligent Scheduling*, chapter 6, Morgan Kaufmann, pp. 169–212.

[20] Mathias Péron & Nicolas Halbwachs (2007): *An abstract domain extending difference-bound matrices with disequality constraints*. In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*, Springer, pp. 268–282, doi:10.1007/978-3-540-69738-1_20.

[21] Wolfgang Thomas (2008): *Solution of Church's Problem: A tutorial*. *New Perspectives on Games and Interaction. Texts on Logic and Games* 5.