SYNTHESIS OF TIMELINE-BASED PLANNING STRATEGIES AVOIDING DETERMINIZATION

DARIO DELLA MONICA [a], ANGELO MONTANARI [a], AND PIETRO SALA [b]

[a] University of Udine, Italy
e-mail address: dario.dellamonica@uniud.it, angelo.montanari@uniud.it

[b] University of Verona, Italy
e-mail address: pietro.sala@univr.it

Abstract. Qualitative timeline-based planning models domains as sets of independent, but interacting, components whose behaviors over time, the timelines, are governed by sets of qualitative temporal constraints (ordering relations), called synchronization rules. Its plan-existence problem has been shown to be PSPACE-complete; in particular, PSPACE-membership has been proved via reduction to the nonemptiness problem for nondeterministic finite automata. However, nondeterministic automata cannot be directly used to synthesize planning strategies as a costly determinization step is needed. In this paper, we identify a fragment of qualitative timeline-based planning whose plan-existence problem can be directly mapped into the nonemptiness problem of deterministic finite automata, which can then be used to synthesize strategies. In addition, we identify a maximal subset of Allen's relations that fits into such a deterministic fragment.

1. Introduction

Timeline-based planning is an approach that originally emerged and developed in the context of planning and scheduling of *space* operations [Mus94]. In contrast to common action-based formalisms, such as PDDL [FL03], timeline-based languages do not distinguish among

Key words and phrases: planning, synthesis, determinism, automata.

* This paper is an extended and revised version of [ADMG+24].

Angelo Montanari acknowledges the support from the Interconnected Nord-Est Innovation Ecosystem (iNEST), which received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.5 – D.D. 1058 23/06/2022, ECS00000043) and from the MUR PNRR project FAIR - Future AI Research (PE00000013) also funded by the European Union Next-GenerationEU. Dario Della Monica acknowledges the partial support from the M4C2 I1.3 "SEcurity and RIghts In the CyberSpace – SERICS" (PE00000014 - CUP D33C22001300002), under the National Recovery and Resilience Plan (NRRP) funded by the European Union-NextGenerationEU and from the Unione europea-Next Generation EU, missione 4 componente 2, project MaPSART "Future Artificial Intelligence (FAIR)", PNRR, PE00000013-CUP C63C22000770006. Dario Della Monica and Angelo Montanari acknowledge the partial support from the 2024 INdAM-GNCS project "Certificazione, monitoraggio, ed interpretabilità in sistemi di intelligenza artificiale" (project n. CUP E53C23001670001). This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

actions, states, and goals. Rather, the domain is modeled as a set of independent, but interacting, components, the timelines, whose behavior over time is governed by a set of temporal constraints. It is worth pointing out that timeline-based planning was born with an application-oriented flavor, with various successful stories, and only relatively recently foundational work about its expressiveness and complexity has been done. The present paper aims at bringing back theory to practice by searching for expressive enough and computationally well-behaved fragments.

Timeline-based planning has been successfully employed by planning systems developed at NASA [CRK+00, CST+04] and at ESA [FCO+11] for both short- to long-term mission planning and on-board autonomy. More recently, timeline-based planning systems such as PLATINUm [UCCO17] are being employed in collaborative robotics applications [UCO23]. All these applications share a deep reliance on *temporal reasoning* and the need for a tight integration of planning with *execution*, both features of the timeline-based framework. The latter feature is usually achieved by the use of *flexible timelines*, which represent a set of possible executions of the system that differ in the precise timing of the events, hence handling the intrinsic *temporal uncertainty* of the environment. A formal account of timeline-based planning with uncertainty has been provided by [COU16]. A lot of theoretical research followed, including *complexity* [BMMP18a, BMMP18b, GMCO17] and *expressiveness* [DMGMS18, GMMO16] analyses, based on the formalization given in [COU16], which is the one we use here as well.

To extend reactivity and adaptability of timeline-based systems beyond temporal uncertainty, the framework of (two-player) *timeline-based games* has been recently proposed. In timeline-based games, the system player tries to build a set of timelines satisfying the constraints independently from the choices of the environment player. This framework allows one to handle general nondeterministic environments in the timeline-based setting. However, its expressive power comes at the cost of increasing the complexity of the problem. While the plan-existence problem for timeline-based planning is EXPTIME-complete [GMCO17], deciding the existence of strategies for timeline-based games is 2EXPTIME-complete [GMO+20], and a controller synthesis algorithm exists that runs in doubly exponential time [AGG+22].

Such a high complexity motivates the search for simpler fragments that can nevertheless be useful in practical scenarios. One of them is the *qualitative* fragment, where temporal constraints only concern the relative order between pairs of events and not their distance. The qualitative fragment already proved itself to be easier for the plan-existence problem, being PSPACE-complete [DMGLTM20], and this makes it a natural candidate for the search of a good fragment for the strategy-existence problem. Unfortunately, a *deterministic* arena is crucial to synthesize a non-clairvoyant strategy in *reactive synthesis* problems (see, for instance, [PR89]), and determinizing the nondeterministic automaton of exponential size built for the qualitative case in [DMGLTM20] would cause an exponential blowup, thus resulting in a procedure of doubly-exponential complexity. The strategy synthesis procedure from a deterministic arena is a well-known, standard construction in reactive synthesis [PR89]. Once the deterministic arena is obtained, one solves a reachability/safety game on it via classical fixpoint algorithms.

In this paper, we identify a class of qualitative timeline-based planning problems, called the *eager fragment*, that admits a characterization of solution plans in terms of *deterministic* finite automata (DFA) of size at most exponential, thereby enabling an exponential solution

to the strategy-existence problem (the synthesis problem for short).[1] Intuitively, the eager fragment aims to remove *nondeterminism* from qualitative timeline-based planning problems. We identify two sources of nondeterminism: disjunctions, which require automata to *guess* the disjuncts witnessing their satisfaction, and some particular conjunctions of constraints, imposing partial ordering on events, which require automata to *guess* the exact (linear) order of the events in advance.

On the one hand, we prove that restricting to the eager fragment is a *sufficient* condition to directly synthesize a DFA of singly-exponential size, thus usable as an arena to play the game in an asymptotically optimal way. On the other hand, we give evidence of the expressive power of such a fragment by showing that eager synchronization rules can systematically encode all major control flow patterns of Business Process Model and Notation (BPMN), including sequential execution, parallel branching, exclusive choice, and iterative loops (Section 8); moreover, we demonstrate that the eager fragment is expressive enough to capture a large subset of Allen's relations [All83] (Section 9).

It is not known, instead, whether both restrictions imposed by the eager fragment to remove the two source of nondeterminism (disjunctions and particular conjunctions imposing a partial ordering on events) are *necessary*, or one suffices. Towards an answer to this question, we identify a class of (non-eager) qualitative timeline-based planning problems for which a characterization of solution plans in terms of DFA of size at most exponential does not exist (Section 5). Such a class relaxes one of the two restrictions (by allowing for disjunctions). However, to show that both restrictions are necessary, a similar result should be proved for a class of problems obtained by relaxing the other restriction.

The rest of the paper is organized as follows. Section 2 recalls some background knowledge on timeline-based planning. Section 3 defines the eager fragment, that directly maps into a DFA of singly exponential size. Section 4 gives a word encoding of timelines, and vice versa. Section 5 proves that it is not possible to encode the solution plans for (non-eager) qualitative timeline-based planning problems using deterministic finite automata of exponential size. Section 6 builds an automaton to recognize plans, and Section 7 shows how to construct an automaton that accepts solution plans. Section 8 demonstrates the practical relevance of the eager fragment through a comprehensive case study that systematically translates BPMN diagrams into eager timeline-based qualitative planning problems, showing that the fragment can capture all major control flow patterns. Section 9 identifies the maximal subset of Allen's relations which is captured by the eager fragment. Finally, Section 10 summarizes the main contributions of the work and discusses possible future developments.

This paper is a considerably revised and extended version of [ADMG+24]. In particular, missing proofs are added, Section 5 and Section 8 are completely new, Section 3 has been enriched with more examples that make it clear the intuition behind eager rules, and Section 9 has been considerably extended with a summarizing table and with several explanatory examples.

2. Background

In this section, we recall the basic notions of timeline-based planning and of its qualitative variant. As usual, $\mathbb{N}$ is the set of natural numbers, and $\mathbb{N}^{>0}$ stands for $\mathbb{N} \setminus \{0\}$.

[1] As a matter of fact, the eager fragment was introduced in [ADMG+24]. However, there was a mistake in its definition, because it allows disjunctions (inside rules), while they must be disallowed for the result to hold.

2.1. **Timeline-based planning.** The key notion is that of *state variable*.

Definition 2.1 (State variable)**.** A *state variable* is a tuple $x = (V_x, T_x, D_x)$, where:

- V_x is the *finite domain* of the variable;
- $T_x : V_x \to 2^{V_x}$ is the *value transition function*, which maps each value $v \in V_x$ to the set of values that can (immediately) follow it;
- $D_x : V_x \to \mathbb{N}^{>0} \times (\mathbb{N}^{>0} \cup \{+\infty\})$ is a function that maps each $v \in V_x$ to the pair $(d_{min}^{x=v}, d_{max}^{x=v})$ of minimum and maximum durations allowed for intervals where $x = v$.

A *timeline* describes how the value of a state variable x evolves over time. It consists of a finite sequence of *tokens*, each denoting a value v and (the duration of) a time interval d.

Definition 2.2 (Token and timeline)**.** A *token* for x is a tuple $\tau = (x, v, d)$, where x is a state variable, $v \in V_x$ is the value held by the variable, and $d \in \mathbb{N}^{>0}$ is the *duration* of the token, with $D_x(v) = (d_{min}^{x=v}, d_{max}^{x=v})$ and $d_{min}^{x=v} \leq d \leq d_{max}^{x=v}$. A *timeline* for a state variable x is a finite sequence $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ of tokens for x, for some $k \in \mathbb{N}$, such that, for any $1 \leq i < k$, if $\tau_i = (x, v_i, d_i)$, then $v_{i+1} \in T_x(v_i)$.

For every timeline $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ and token $\tau_i = (x, v_i, d_i)$ in T, we define the functions $\mathrm{start}(\mathsf{T}, i) = \sum_{j=1}^{i-1} d_j$ and $\mathrm{end}(\mathsf{T}, i) = \mathrm{start}(\mathsf{T}, i) + d_i$. We call the *horizon* of T the end time of the last token in T, that is, $\mathrm{end}(\mathsf{T}, k)$. We write $\mathrm{start}(\tau_i)$ and $\mathrm{end}(\tau_i)$ to indicate $\mathrm{start}(\mathsf{T}, i)$ and $\mathrm{end}(\mathsf{T}, i)$, respectively, when there is no ambiguity.

The overall behavior of state variables is subject to a set of temporal constraints known as *synchronization rules* (or simply *rules*). We start by defining their basic building blocks. Let $\mathcal{N}$ be a finite set of *token names*. *Atoms* are formulas of the following form:

$$\begin{aligned} \mathit{atom} &\coloneqq \mathit{term} \leq_{[l,u]} \mathit{term} \mid \mathit{term} <_{[l,u]} \mathit{term} \\ \mathit{term} &\coloneqq \mathtt{s}(a) \mid \mathtt{e}(a) \mid t \end{aligned}$$

where $a \in \mathcal{N}$, $l, t \in \mathbb{N}$, and $u \in \mathbb{N} \cup \{+\infty\}$. Terms $\mathtt{s}(a)$ and $\mathtt{e}(a)$ respectively denote the start and the end of the token associated with the token name a. As an example, atom $\mathtt{s}(a) \leq_{[l,u]} \mathtt{e}(b)$ (resp., $\mathtt{s}(a) <_{[l,u]} \mathtt{e}(b)$) relates tokens a and b by stating that the end of b cannot precede (resp., must succeed) the beginning of a, and the distance between these two endpoints must be at least l and at most u. An atom $\mathit{term}_1 \leq_{[l,u]} \mathit{term}_2$, with $l = 0$, $u = +\infty$, and $\mathit{term}_1, \mathit{term}_2 \notin \mathbb{N}$, is *qualitative* (the subscript is usually omitted in this case). We sometimes use the abbreviation $\mathit{term}_1 = \mathit{term}_2$ for $\mathit{term}_1 \leq \mathit{term}_2 \wedge \mathit{term}_2 \leq \mathit{term}_1$.

An *existential statement* $\mathcal{E}$ is a constraint of the form:

$$\exists a_1[x_1 = v_1]a_2[x_2 = v_2] \ldots a_n[x_n = v_n].\ \mathcal{C}$$

where $x_1, \ldots, x_n$ are state variables, $v_1, \ldots, v_n$ are values, with $v_i \in V_{x_i}$, for $i = 1, \ldots, n$, $a_1, \ldots, a_n$ are token names from $\mathcal{N}$, and $\mathcal{C}$ is a finite conjunction of atoms, called a *clause*, involving only tokens $a_1, \ldots, a_n$, plus, possibly, the *trigger token* (usually denoted by a_0) of the *synchronization rule* in which the existential statement is embedded, as shown below.[2] Intuitively, an existential statement asks for the existence of tokens $a_1, a_2, \ldots, a_n$ whose state variables take the corresponding values $v_1, v_2, \ldots, v_n$ and are such that their start and end times satisfy the atoms in $\mathcal{C}$.

[2]Without loss of generality, we assume that *(i)* if a token a appears in the quantification prefix $\exists a_1[x_1 = v_1]a_2[x_2 = v_2] \ldots a_n[x_n = v_n]$ of $\mathcal{E}$, then at least one among $\mathtt{s}(a)$ and $\mathtt{e}(a)$ occurs in one of its atoms, and *(ii)* trivial atoms, i.e., atoms of the form $\mathtt{s}(a) \leq \mathtt{s}(a)$, $\mathtt{e}(a) \leq \mathtt{e}(a)$, $\mathtt{s}(a) \leq \mathtt{e}(a)$, or $\mathtt{s}(a) < \mathtt{e}(a)$, never occur in existential statements, even though they clearly hold by the definition of token.

Synchronization rules have one of the following forms:

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

where $a_0 \in \mathcal{N}$, x_0 is a state variable, $v_0 \in V_{x_0}$, and $\mathcal{E}_i$ is an existential statement, for each $1 \leq i \leq k$. In the former case, $a_0[x_0 = v_0]$ is called *trigger* and a_0 is the *trigger token*, and the rule is considered *satisfied* if *for all* the tokens for x_0 with value v_0, at least one of the existential statements is satisfied. In the latter case, the rule is said to be *triggerless*, and it states the truth of the body without any precondition.[3] We refer the reader to [COU16] for a formal account of the semantics of the rules. A synchronization rule is *disjunction-free* if it contains only one existential statement, that is, $k = 1$.

A *timeline-based planning problem* consists of a set of state variables and a set of rules that represent the problem domain and the goal.

Definition 2.3 (Timeline-based planning problem)**.** A *timeline-based planning problem* is defined as a pair $P = (\mathsf{SV}, S)$, where SV is a set of state variables and S is a set of synchronization rules involving state variables in SV. Moreover, P is *disjunction-free* if so are all of its synchronization rules.

A *solution plan* for a given timeline-based planning problem is a set of timelines, one for each state variable, that satisfies all the synchronization rules.

Definition 2.4 (Plan and solution plan)**.** A *plan* over a set of state variables SV is a finite set of timelines with the same horizon, one for each state variable $x \in \mathsf{SV}$. A *solution plan* (or, simply, *solution*) for a timeline-based planning problem $P = (\mathsf{SV}, S)$ is a plan over SV such that all the rules in S are satisfied.

The problem of determining whether a solution plan exists for a given timeline-based planning problem is EXPSPACE-complete [GMCO17].

Definition 2.5 (Qualitative timeline-based planning)**.** A timeline-based planning problem $P = (\mathsf{SV}, S)$ is said to be *qualitative* if the following conditions hold:

(1) $D_x(v) = (1, +\infty)$, for all state variables $x \in \mathsf{SV}$ and $v \in V_x$.
(2) synchronization rules in S involve *qualitative* atoms only.

In the rest of the paper, we focus on qualitative timeline-based planning. Its complexity is shown to be PSPACE-complete in [DMGLTM20], where a reduction to the nonemptiness problem for non-deterministic finite automata (NFA) is given.

3. A well-behaved fragment

In this section, we introduce a meaningful fragment of qualitative timeline-based planning for which it is possible to construct a DFA of singly exponential size. The fragment is characterized by conditions on the admissible patterns of synchronization rules (*eager rules*). The distinctive feature of eager rules is that their satisfaction with respect to a given plan can be checked using an eager/greedy strategy, that is, when a relevant event (start/end of a token involved in some atom) occurs, we are guaranteed that the starting/ending point of

[3]Without loss of generality, if a_0 is the trigger token of a non-triggerless rule, then both $\mathtt{s}(a_0)$ and $\mathtt{e}(a_0)$ occur in the existential statements of the rule. In particular, as an exception, we allow trivial atoms over trigger tokens (e.g., $\mathtt{s}(a_0) < \mathtt{e}(a_0)$).

such a token is useful for rule satisfaction. With non-eager rules, instead, a relevant event may occur that is not useful for rule satisfaction: some analogous event in the future will be.

As a preliminary step, we define a sort of reflexive and transitive closure of a clause. By slightly abusing the notation, we identify a clause $\mathcal{C}$ with the finite set of atoms occurring in it. Let t, t_1, t_2, and t_3 be terms of the form $\mathtt{s}(a)$ or $\mathtt{e}(a)$, with $a \in \mathcal{N}$. The *closure* of $\mathcal{C}$, denoted by $\hat{\mathcal{C}}$, is defined as the smallest set of atoms including $\mathcal{C}$ such that: *(i)* if term t occurs in $\mathcal{C}$, then atom $t \leq t$ belongs to $\hat{\mathcal{C}}$, *(ii)* if both terms $\mathtt{s}(a)$ and $\mathtt{e}(a)$ occur in $\mathcal{C}$ for some token name a, then atom $\mathtt{s}(a) < \mathtt{e}(a)$ belongs to $\hat{\mathcal{C}}$, *(iii)* if atom $t_1 < t_2$ belongs to $\hat{\mathcal{C}}$, then atom $t_1 \leq t_2$ belongs to $\hat{\mathcal{C}}$ as well, *(iv)* if atoms $t_1 \leq t_2$ and $t_2 \leq t_3$ belong to $\hat{\mathcal{C}}$, then atom $t_1 \leq t_3$ belongs to $\hat{\mathcal{C}}$ as well, *(v)* if atoms $t_1 < t_2$ and $t_2 \leq t_3$ belong to $\hat{\mathcal{C}}$, then atom $t_1 < t_3$ belongs to $\hat{\mathcal{C}}$ as well, *(vi)* if atoms $t_1 \leq t_2$ and $t_2 < t_3$ belong to $\hat{\mathcal{C}}$, then atom $t_1 < t_3$ belongs to $\hat{\mathcal{C}}$ as well.[4] We write $t_1 \equiv t_2 \in \hat{\mathcal{C}}$ as an abbreviation to mean that both atoms $t_1 \leq t_2$ and $t_2 \leq t_1$ belong to $\hat{\mathcal{C}}$, and $t_1 \equiv t_2 \notin \hat{\mathcal{C}}$ for its negation. Clearly, the set (of terms occurring in) $\hat{\mathcal{C}}$ and the relation $\leq$ define a preorder.

Intuitively, the closure $\hat{\mathcal{C}}$ captures all the temporal ordering relationships that are logically implied by the explicit constraints in $\mathcal{C}$. It includes not only the directly stated ordering constraints, but also their transitive consequences and the implicit relationships that follow from the nature of tokens (such as the fact that every token has a start time before its end time). As an example, if $\mathcal{C}$ contains atoms $\mathtt{s}(a) \leq \mathtt{s}(b)$ and $\mathtt{s}(b) \leq \mathtt{e}(c)$, then $\hat{\mathcal{C}}$ will also contain the atom $\mathtt{s}(a) \leq \mathtt{e}(c)$, which follows by transitivity, even though this relationship was not explicitly stated. Similarly, if both $\mathtt{s}(a)$ and $\mathtt{e}(a)$ appear in $\mathcal{C}$, then $\hat{\mathcal{C}}$ automatically includes $\mathtt{s}(a) < \mathtt{e}(a)$, reflecting the fundamental property that tokens have positive duration. On the other hand, the absence of a relationship between token endpoints indicate that either the opposite relationship holds or the relationship between such endpoints is not implied by the existing constraints. For instance, if $\mathtt{e}(a) \leq \mathtt{s}(b)$ does not belong to the closure, then it can be that $\mathtt{s}(b) < \mathtt{e}(a)$ is in the closure (and thus b must begin before the end of a) or, simply, that the end of a is not constrained to occur before the beginning of b. This closure operation ensures that we have a complete picture of all temporal relationships that must hold in any plan satisfying the rule, which is essential for determining whether a rule can be checked eagerly or not. Notice that trivial atoms over non-trigger tokens are allowed in the closure of a clause (while they are disallowed in the clause itself).

Let us now define the fundamental notion of *eager rule*.

Definition 3.1 (Ambiguous token, eager rule, eager planning problem)**.** Let $\mathcal{R}$ be a synchronization rule, $\mathcal{C}$ one of the clauses of $\mathcal{R}$, and a a token name occurring in $\mathcal{C}$. Then,

(A) a is *left-ambiguous* if all of the following are verified:
  (A1) if a_0 is the trigger token of $\mathcal{R}$, then $\mathtt{s}(a) \equiv \mathtt{s}(a_0) \notin \hat{\mathcal{C}}$ and $\mathtt{s}(a) \equiv \mathtt{e}(a_0) \notin \hat{\mathcal{C}}$, and
  (A2) there is a term $t \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$ such that $\mathtt{s}(a) \leq t \in \hat{\mathcal{C}}$ and $\mathtt{e}(a) \leq t \notin \hat{\mathcal{C}}$;

(B) a is *right-ambiguous* if there is a term $t \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$ such that $t \leq \mathtt{e}(a) \in \hat{\mathcal{C}}$ and $t \leq \mathtt{s}(a) \notin \hat{\mathcal{C}}$;

(C) a is *ambiguous* if it is not the trigger token of $\mathcal{R}$, and it is both left- and right-ambiguous.

Rule $\mathcal{R}$ is *unambiguous* if none of the token names occurring in it is ambiguous.

[4]Without loss of generality, we assume that $\hat{\mathcal{C}}$ is consistent, *i.e.*, it admits at least a solution. This check can be done in polynomial time, since it is an instance of linear programming.

Rule $\mathcal{R}$ is *eager* if it is unambiguous and disjunction-free.
Finally, a qualitative timeline-based planning problem $P = (\mathsf{SV}, S)$ is *eager* if S only contains eager rules.

Clearly, deciding whether a rule is disjunction-free is linear in its size. Moreover, deciding whether a disjunction-free rule is also eager amounts to computing the closure of its unique clause (which requires time at most polynomial in the size of the rule), and then verifying that none of its the token names is ambiguous (which ultimately reduces to a series of membership tests, and thus requires time linear in the size of the closure). Therefore, deciding whether a rule is eager can be done in time at most polynomial in the size of the rule.

From now on, we focus on eager qualitative timeline-based planning problems. For the sake of brevity, we sometimes refer to them simply as *planning problems*. Intuitively, eager rules remove the *nondeterminism* (a sort of ambiguity) that comes from two factors: disjunctions, which require to *guess* a disjunct, and some patterns of conjunctions of atoms, which require to *guess* the right occurrences of events relevant for satisfaction. We claim that restricting to eager rules (thus, removing these two sources of nondeterminism) suffices to obtain a singly exponential DFA, whose construction will be illustrated in the next sections. We give here a short intuitive account of the rationale behind the conditions of Definition 3.1.

Consider the following synchronization rule:

$$a_0[x_0 = v_0] \rightarrow \exists a_1[x_1 = v_1].\ (\mathtt{s}(a_0) = \mathtt{s}(a_1) \wedge \mathtt{e}(a_0) \leq \mathtt{e}(a_1)).$$

According to Definition 3.1, it is an eager rule because a_0 is not ambiguous (trivially, as a_0 is the trigger token), and neither is a_1, since $\mathtt{s}(a_1) \equiv \mathtt{s}(a_0) \in \hat{\mathcal{C}}$, which implies that a_1 is not left-ambiguous (see Item *A1* in Definition 3.1). In particular, having $\mathtt{s}(a_0) = \mathtt{s}(a_1)$ is crucial for any DFA $\mathcal{A}$ recognizing solution plans, because, when $\mathcal{A}$ reads the event $\mathtt{s}(a_0)$, it can *eagerly* and *deterministically* go to a state representing the fact that both $\mathtt{s}(a_0)$ and $\mathtt{s}(a_1)$ have happened. Moreover, if later it reads the event $\mathtt{e}(a_1)$, but it has not read $\mathtt{e}(a_0)$ yet, then it transitions to a rejecting state, that is, a state from which it cannot accept any plan; if, instead, it reads the event $\mathtt{e}(a_1)$ only after reading $\mathtt{e}(a_0)$ (or at the same time), then it transitions to an accepting state.

As a more subtle example, consider the following rule, obtained from the previous one by replacing $=$ by $\leq$ in the clause:

$$a_0[x_0 = v_0] \rightarrow \exists a_1[x_1 = v_1].\ (\mathtt{s}(a_0) \leq \mathtt{s}(a_1) \wedge \mathtt{e}(a_0) \leq \mathtt{e}(a_1)). \tag{1}$$

This rule is eager as well, because, once again, trigger token a_0 is trivially not ambiguous and token name a_1 is not left-ambiguous (Item *A2* in Definition 3.1 is falsified). It is not immediate, though, to see that this rule can always be handled *eagerly* by a DFA. Indeed, consider the plan (partially) depicted in Figure 1. The picture shows a token of value v_0 for timeline x_0 that starts at time 1 and ends a time 4. Such a token matches the trigger token a_0, thus triggering the rule. Figure 1 also depicts two tokens matching value (v_1) and timeline (x_1) associated with token name a_1: the first one starting at time 2 and the second one starting at time 5. The plan satisfies the rule from Equation (1) above, thanks to the token starting at time 5. Now, consider a DFA that tries to *eagerly* verify the rule. Initially, the DFA idles until the beginning of a token that matches the trigger token a_0. When, at time 1, the token for x_0 starts, the DFA transitions into a state where $\mathtt{s}(a_0)$ has happened, and the automaton waits for the beginning of a token for x_1 of value v_1 or the end of the

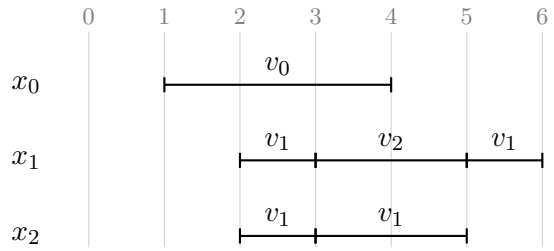


Figure 1: A (partial) plan satisfying the rule from Equation (1), which is eager, and the one from Equation (2), which is not. The token starting at time 1 triggers both rules. In both cases, the satisfaction of the rule is witnessed by the second token of value v_1 in the relevant timeline (not the first one). However, a DFA that behaves eagerly can be instructed to identify the witnessing token for the eager rule, but not for the non-eager one. Note that the token for x_1 starting at time 3 has value $v_2 \neq v_1$; thus, it does not match token name a_1, and cannot be used to fulfill the rule from Equation (1).

token for x_0 (the two events can also happen at the same time). At time 2, a token starts that matches token name a_1; even if this token is not useful to certify the fulfillment of the rule (since it ends before the ending of the token for x_0), a DFA that behaves eagerly will match this event with $\mathtt{s}(a_1)$, and will thus transition into a state where it waits for the token for x_0 to end before (or at the same time of) the token for x_1. Since the token for x_1 ends at time 3, strictly before the token for x_0, it seems that the eager choice of the DFA will lead to a rejection. Fortunately, since $\mathtt{s}(a_1)$ is not constraint to happen before some other event (there is no atom $\mathtt{s}(a_1) \leq t$ in $\hat{\mathcal{C}}$, with $t \in \{\mathtt{s}(a_0), \mathtt{e}(a_0)\}$ – see Item *A2* in Definition 3.1), the DFA can be instructed to ignore the ending, at time 3, of the token for x_1, thus, as a matter of fact, *adjusting* the first match of $\mathtt{s}(a_1)$: it will be (tacitly) re-matched with some future token start. At time 4, the token for x_0 ends, and the DFA transitions into a state where it waits for the end of a token for x_1 of value v_1, which will happen at time 6. Tacitly, term $\mathtt{s}(a_1)$ is necessarily re-matched with the beginning of the token ending at time 6.

Let us provide now an example of a non-eager rule, thus not amenable to being checked in an eager/greedy fashion:

$$a_0[x_0 = v_0] \rightarrow \exists a_3[x_2 = v_1].\ (\mathtt{s}(a_0) \leq \mathtt{s}(a_3) \wedge \mathtt{s}(a_3) \leq \mathtt{e}(a_0) \wedge \mathtt{e}(a_0) \leq \mathtt{e}(a_3)). \tag{2}$$

This rule is *not* eager, because token name a_3 is ambiguous (as it is not the trigger token, and it is both left-and right-ambiguous), and thus cannot be handled by a DFA that acts eagerly like the one described above. As a matter of fact, such a DFA would not accept the plan (partially) depicted in Figure 1, even though it satisfies the rule. In particular, due to a_3 being left-ambiguous (there is $t \in \{\mathtt{s}(a_0), \mathtt{e}(a_0)\}$ such that $\mathtt{s}(a_3) \leq t \in \hat{\mathcal{C}}$ and $\mathtt{e}(a_3) \leq t \notin \hat{\mathcal{C}}$ – see Item *A2* in Definition 3.1), it is not possible to instruct a DFA, as before, to somehow match $\mathtt{s}(a_3)$ with the beginning of the token for x_2 starting at time 2 and $\mathtt{e}(a_3)$ with the end of a different token for x_2, the one ending at time 5. In other words, the eager

choice of the DFA to match $\mathtt{s}(a_3)$ with the beginning of the token for x_2 starting at time 2 is final and cannot be adjusted, and the DFA is not able to deterministically establish that the token witnessing the rule is the second one for x_2 (rather than the first one): it is something that must be *guessed nondeterministically*.

In what follows, we give a reduction from the plan-existence problem for the eager fragment of qualitative timeline-based planning to the nonemptiness problem of DFAs of *singly exponential* size with respect to the original problem. The approach is inspired by those in [DMGLTM20, DMGMS18] for non-eager timeline-based planning, where an NFA of exponential size is built for any given timeline-based planning problem. It is important to note that there is a bijective correspondence between the set of solutions of a planning problem and the language accepted by the automaton built from that problem. However, the reductions presented in [DMGLTM20, DMGMS18] produce nondeterministic automata, which cannot be used as arenas to solve timeline-based games without a preliminary determinization step that would cause a second exponential blowup.

In the following, we first show how to encode timelines and plans as finite words, and vice versa (Section 4). Using such an encoding, we show that it is not possible to characterize the solution plans for non-eager qualitative timeline-based planning problems using DFA of exponential size (Section 5). Then, given an eager qualitative timeline-based planning problem $P = (\mathsf{SV}, S)$, we show how to build a DFA whose language encodes the set of solution plans for P. The DFA consists of the intersection of two DFAs: one checks that the input word correctly encodes a (candidate) plan over SV that fulfills the constraint on the alternation of token values expressed by functions T_x, for $x \in \mathsf{SV}$ (Section 6); the other one verifies that the encoded plan is indeed a solution plan for P, i.e., synchronization rules in S are fulfilled (Section 7).

4. From plans to finite words and vice versa

In this section, as a first step towards the construction of the DFA corresponding to an eager qualitative timeline-based planning problem, we show how to encode timelines and plans as *words* that can be recognized by an automaton, and *vice versa*.

Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem, and let $V = \cup_{x \in \mathsf{SV}} V_x$. We define the *initial alphabet* Σ^I_{SV} as $(\{-\} \times V)^{\mathsf{SV}}$, that is the set of functions from SV to $(\{-\} \times V)$.[5] Similarly, we define the *non-initial alphabet* Σ^N_{SV} as $((V \times V) \cup \{\circlearrowleft\})^{\mathsf{SV}}$, where the pairs $(v, v') \in V \times V$ are supposed to encode the value v of the token that has just ended and the value v' of the token that has just started, and $\circlearrowleft$ represents the fact that the value for the state variable has not changed. The *input alphabet* (or, simply, *alphabet*) associated with SV, denoted by Σ_{SV}, is the union $\Sigma^I_{\mathsf{SV}} \cup \Sigma^N_{\mathsf{SV}}$. Observe that the size of the alphabet Σ_{SV} is at most exponential in the size of SV, precisely $|\Sigma_{\mathsf{SV}}| = |\Sigma^I_{\mathsf{SV}}| + |\Sigma^N_{\mathsf{SV}}| = |V|^{|\mathsf{SV}|} + (|V|^2 + 1)^{|\mathsf{SV}|}$.

We now show how to encode the basic structure[6] underlying each plan over SV as a word in $\Sigma^I_{\mathsf{SV}} \cdot (\Sigma^N_{\mathsf{SV}})^* \cup \{\epsilon\}$, where ϵ is the empty word (corresponding to the empty plan), $(\Sigma^N_{\mathsf{SV}})^*$ is the Kleene's closure of Σ^N_{SV}, and $\cdot$ denotes the concatenation operation. Intuitively,

[5]The symbol $\{-\}$ is a technicality that allows us to consider pairs instead of single values in V, to be uniform with symbols of the non-initial alphabet.

[6]With "basic structure" we refer to the fact that, in this section, we neither take into account the transition functions T_x of state variables nor their domains V_x (*cf.* Definition 2.1), which will be dealt with in Section 6. Recall that functions D_x are irrelevant as we only consider qualitative planning problems (*cf.* Definition 2.5).

let σ be the symbol at position i of a word $\lambda \in \Sigma^I_{\mathsf{SV}} \cdot (\Sigma^N_{\mathsf{SV}})^* \cup \{\epsilon\}$. Then, if $\sigma(x) = (v, v')$ for some $x \in \mathsf{SV}$, then at time i a new token begins in the timeline for x with value v'; instead, if $\sigma(x) = \circlearrowleft$, then no change happens at time i in the timeline for x, meaning that no token ends at that time point in the timeline for x. The value v of the token ending at time i will come in handy later in the construction of the automata.

We remark that not all words in $\Sigma^I_{\mathsf{SV}} \cdot (\Sigma^N_{\mathsf{SV}})^* \cup \{\epsilon\}$ correspond to plans over SV: for a word to correctly encode a plan, the information carried by the word about the value of a starting token and the one associated to the end of the same token must coincide. Formally, given a word $\lambda = \langle \sigma_0, \ldots, \sigma_{|\lambda|-1} \rangle \in \Sigma^I_{\mathsf{SV}} \cdot (\Sigma^N_{\mathsf{SV}})^* \cup \{\epsilon\}$ and a state variable $x \in \mathsf{SV}$, let $\mathit{changes}(x) = (i^x_0, i^x_1, \ldots, i^x_{k^x-1})$, for some $k^x \in \mathbb{N}$, be the increasing sequence of positions where x changes, *i.e.*, $i \in \mathit{changes}(x)$ if and only if $\sigma_i(x) \neq \circlearrowleft$, for all $i \in \{0, \ldots, |\lambda| - 1\}$. We denote by $\overleftarrow{\sigma}^x_i$ and $\vec{\sigma}^x_i$ the first and the second component of $\sigma_i(x)$, respectively, for all $x \in \mathsf{SV}$ and $i \in \mathit{changes}(x)$. We omit superscripts ^{x} when there is no risk of ambiguity.

Definition 4.1 (Words weakly-encoding plans)**.** Let $\lambda \in \Sigma^I_{\mathsf{SV}} \cdot (\Sigma^N_{\mathsf{SV}})^* \cup \{\epsilon\}$ and let $\mathit{changes}(x) = (i_0, i_1, \ldots, i_{k-1})$. We say that λ *weakly-encodes a plan over* SV if $\vec{\sigma}^x_{i_{h-1}} = \overleftarrow{\sigma}^x_{i_h}$ for all $x \in \mathsf{SV}$ and $h \in \{1, \ldots, k-1\}$. If this is the case, then the *plan induced by* λ is the set $\{\mathsf{T}_x \mid x \in \mathsf{SV}\}$, where $\mathsf{T}_x = \langle (x, \vec{\sigma}^x_{i_0}, i_1 - i_0), (x, \vec{\sigma}^x_{i_1}, i_2 - i_1), \ldots, (x, \vec{\sigma}^x_{i_{k-1}}, i_k - i_{k-1}) \rangle$ and $i_k = |\lambda|$, for all $x \in \mathsf{SV}$.

Intuitively, if a word weakly-encodes a plan, then it captures the dynamics of a state variable, but it ignores its domain and transition function, which will be taken care of in the next sections. A converse correspondence from plans to words can be defined accordingly.

Before concluding the section, we introduce another couple of notions that will come in handy later. We denote by $\mathit{events}(\sigma)$ the set of events (beginning/ending of a token) occurring at a given time, encoded in the alphabet symbol σ. Formally, $\mathit{events}(\sigma)$ is the smallest set such that:

- if $\sigma(x) = (v, v')$ for some x, then $\{\mathit{end}(x, v), \mathit{start}(x, v')\} \subseteq \mathit{events}(\sigma)$, and
- if $\sigma(x) = (-, v')$ for some x, then $\mathit{start}(x, v') \in \mathit{events}(\sigma)$.

Finally, we say that σ *triggers* a rule $\mathcal{R}$ if $\mathit{start}(x_0, v_0) \in \mathit{events}(\sigma)$ and $a_0[x_0 = v_0]$ is the trigger of $\mathcal{R}$.

5. Instances for which no exponential automata exist

We let $[n] = \{0, 1, \ldots, n\}$ and $[n]^{>0} = \{1, \ldots, n\}$, for all $n \in \mathbb{N}$. For a planning problem P, we denote by $|P|$ its size, that is, the length of its representation[7], and by $\mathcal{L}(P)$ the language of words encoding solution plans for P.

In this section, we show that a characterization of the solution plans for (non-eager) qualitative timeline-based planning problems using deterministic finite automata of exponential size does not exist. More precisely, we define a schema $\{P_n\}_{n \in \mathbb{N}^{>0}}$ of (non-eager) qualitative timeline-based planning problems and we show that, for n large enough, the smallest automata accepting $\mathcal{L}(P_n)$ has size more than exponential in the size of P_n (Theorem 5.1 below).

For all $n \in \mathbb{N}^{>0}$, we let $P_n = (\mathsf{SV}_n, S_n)$, where

[7]The length of the representation of a qualitative planning problem (SV, S) is the sum of the length of the representation of SV, which is at most polynomial in the number of variables in it and the size of the domains V_x, for $x \in \mathsf{SV}$, and the one of the representation of S, which is at most polynomial in the number of synchronization rules and the size of the longest rule in it.

- $\mathsf{SV}_n = \{x_i\}_{i\in[n]}$, with
  - $x_i = (V_{x_i}, T_{x_i}, D_{x_i})$ for all $i \in [n]$,
  - $V_{x_0} = \{v_0, \overline{v}_0\}$,
  - $V_{x_i} = \{v_i, v_i', \overline{v}_i\}$ for all $i \in [n]^{>0}$,
  - $T_{x_i}(v) = V_{x_i}$ for all $i \in [n]$ and $v \in V_{x_i}$,
  - $D_{x_i}(v) = (1, +\infty)$ for all $i \in [n]$ and $v \in V_{x_i}$.
- S_n is a singleton containing the following synchronization rule

$$\begin{aligned} a_0[x_0 = v_0] \;\rightarrow\; & \exists a_1[x_1 = v_1]a_1'[x_1 = v_1'].\, \mathtt{s}(a_0) \leq \mathtt{s}(a_1) \wedge \mathtt{s}(a_1) \leq \mathtt{e}(a_0) \wedge \mathtt{e}(a_0) \leq \mathtt{s}(a_1') \\ \vee\; & \exists a_2[x_2 = v_2]a_2'[x_2 = v_2'].\, \mathtt{s}(a_0) \leq \mathtt{s}(a_2) \wedge \mathtt{s}(a_2) \leq \mathtt{e}(a_0) \wedge \mathtt{e}(a_0) \leq \mathtt{s}(a_2') \\ \vee\; & \ldots \\ \vee\; & \exists a_n[x_n = v_n]a_n'[x_n = v_n'].\, \mathtt{s}(a_0) \leq \mathtt{s}(a_n) \wedge \mathtt{s}(a_n) \leq \mathtt{e}(a_0) \wedge \mathtt{e}(a_0) \leq \mathtt{s}(a_n') \end{aligned}$$

In what follows, we identify, for any given $n \in \mathbb{N}^{>0}$, a set of plans (equivalently, words encoding plans) over SV_n that contains more than 2^n plans that are pairwise distinguished by some extension, in the sense clarified later. The thesis (*cf.* Theorem 5.1) then follows from Myhill-Nerode theorem [HMU06]. Note also that the above synchronization rule is not eager, as it is not disjunction-free (it is unambiguous, though – *cf.* Definition 3.1). Therefore, while restricting to disjunction-free and unambiguous (thus, eager) rules is a sufficient condition towards the construction of the desired deterministic automata, it is unclear whether both restrictions are necessary or if restricting only to disjunction-free rule – while still allowing ambiguous ones – is already a sufficient condition.

Now, consider plans over SV_n, for $n \in \mathbb{N}^{>0}$, such that:

- the timeline associated with x_0 is a sequence of v_0-valued tokens of duration two, followed by a $\overline{v}_0$-valued final token of duration one;
- each other timeline, associated with variable x_j for some $j \in [n]^{>0}$,
  - begins with a $\overline{v}_j$-valued token whose duration is one and
  - continues with a sequence of tokens of duration two and whose value is either v_j or $\overline{v}_j$.

Formally, for every $n \in \mathbb{N}^{>0}$, let W_n be the set of words of odd length encoding plans over SV_n (that is, every word $\lambda \in W_n$ has the form $\sigma_0\sigma_1 \ldots \sigma_h$ for some even positive natural number h) such that:

- the timeline associated with state variable x_0 evolves as follows:
  - $\sigma_0(x_0) = (-, v_0)$ (a v_0-valued token starts at the beginning of the timeline),
  - $\sigma_i(x_0) = \circlearrowleft$, for all odd $i \in [h]$ (at odd time instants no token starts/ends),
  - $\sigma_i(x_0) = (v_0, v_0)$, for all even $i \in \{2, \ldots, h-2\}$ (at even time instants, except for h, a new v_0-valued token starts),
  - $\sigma_h(x_0) = (v_0, \overline{v}_0)$ (the last token has value $\overline{v}_0$), and
- other timelines, associated with state variables x_j, for $j \in [n]^{>0}$, evolve as follows:
  - $\sigma_0(x_j) = (-, \overline{v}_j)$ (a $\overline{v}_j$-valued token starts at the beginning of the timeline),
  - $\sigma_i(x_j) = \circlearrowleft$, for all even $i \in [h]^{>0}$ (at even time instants, except for zero, no token starts/ends),
  - $\sigma_i(x_j) = (v_{old}, v_{new})$ for some $v_{new} \in \{v_j, \overline{v}_j\}$ and some $v_{old} \in V_{x_j}$, and for all odd $i \in [h]$ (at odd time instants a new token starts whose value can be either v_j or $\overline{v}_j$).

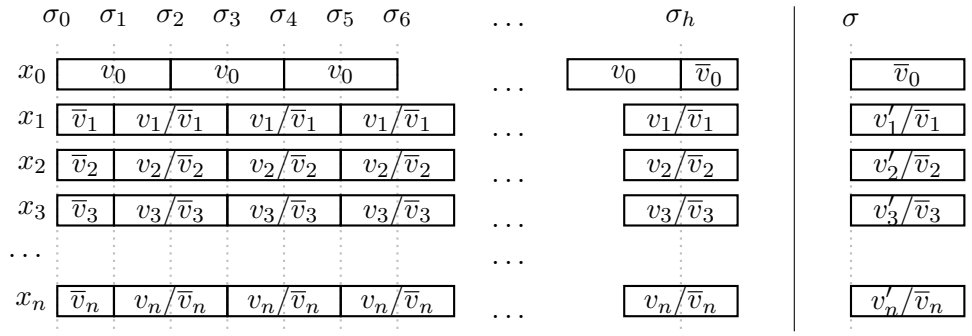


Figure 2: Graphical representation of sets W_n, with $n \in \mathbb{N}$. W_n includes words $\lambda = \sigma_0\sigma_1 \dots \sigma_h$, for all even positive natural numbers h, encoding plans over SV_n and such that: the timeline associated with variable x_0 features $h/2$ v_0-valued tokens and a final $\overline{v}_0$-valued token; timelines associated with variables other than x_0 feature an initial $\overline{v}_j$-valued token, followed by $h/2$ tokens labeled with either v_j or $\overline{v}_j$, for $j \in [n]^{>0}$.

A graphical account of words in W_n, for $n \in \mathbb{N}$, is given in Figure 2, where h is any even positive natural number. It is easy to see that none of the words in W_n represents a solution plan for P_n. However, each of them can be completed into a solution plan by adding only one more token in each timeline. Formally, let Λ be the set of all functions $\sigma : \mathsf{SV} \to \bigcup_{i\in[n]} V_{x_i}$ such that $\sigma(x_0) = \overline{v}_0$ and $\sigma(x_i) \in \{v'_i, \overline{v}_i\}$, for all $i \in [n]^{>0}$ (see Figure 2, right-hand side). We have that

the concatenation of a plan $\sigma_0\sigma_1 \dots \sigma_h$ in W_n with a function $\sigma \in \Lambda$ is a solution plan for P_n if and only if for all odd natural numbers $i \in [h]$ there is $j \in [n]^{>0}$ such that $\sigma_i(x_j) = v_j$ and $\sigma(x_j) = v'_j$. (3)

Two plans $\sigma_0\sigma_1 \dots \sigma_h$ and $\sigma'_0\sigma'_1 \dots \sigma'_{h'}$ in W_n are *distinguished* by Λ if and only if there is $\sigma \in \Lambda$ such that exactly one between $\sigma_0\sigma_1 \dots \sigma_h\sigma$ and $\sigma'_0\sigma'_1 \dots \sigma'_{h'}\sigma$ is a solution plan for P_n. In what follows, we show that, for n large enough, W_n contains more than 2^n plans that are pairwise distinguished by Λ. The thesis then follows from Myhill-Nerode theorem [HMU06].

To this end, notice that timeline associated with x_0 is the same for all plans in W_n of a given length, and thus a plan in W_n is univocally identified by its length and the sequences of $v_j/\overline{v}_j$ tokens in all the other timelines. Therefore, there is a bijection ξ between plans $\sigma_0\sigma_1 \dots \sigma_h$ in W_n of length $h+1$ and sequences $\mu_1\mu_2 \dots \mu_{\frac{h}{2}}$, of length $\frac{h}{2}$, of sets of natural numbers, defined as follows: $\xi(\sigma_0\sigma_1 \dots \sigma_h) = \mu_1\mu_2 \dots \mu_{\frac{h}{2}}$, where $\mu_i \subseteq [n]^{>0}$ for all $i \in [\frac{h}{2}]^{>0}$ and $j \in \mu_i$ if and only if in the timeline associated with x_j the ith token after the initial $\overline{v}_j$-valued token has value v_j. (Observe that it is essential for the existence of bijection ξ to have state variables with disjoint domains, i.e., $V_{x_i} \cap V_{x_j} = \varnothing$ for all $i \neq j$.) Analogously, all functions in Λ have the same value at x_0; therefore, there is another bijection, which we call ξ abusing the notation, between Λ and the powerset of $[n]^{>0}$ (namely, $2^{\{1,2,\dots,n\}}$), which maps every $\sigma \in \Lambda$ to set $\xi(\sigma) = \mu \subseteq [n]^{>0}$, with $j \in \mu$ if and only if $\sigma(x_j) = v'_j$ (and thus $j \notin \mu$ if and only if $\sigma(x_j) = \overline{v}_j$). Now, given a plan $\sigma_0\sigma_1 \dots \sigma_h$ in W_n and $\sigma \in \Lambda$ such that

$\xi(\sigma_0\sigma_1\dots\sigma_h) = \mu_1\mu_2\dots\mu_{\frac{h}{2}}$ and $\xi(\sigma) = \mu$, we can rephrase statement (3) above as:

$$\textit{the concatenation of } \mu_1\mu_2\dots\mu_{\frac{h}{2}} \textit{ with } \mu \textit{ represents a solution plan for } P_n \textit{ if and only if } \mu_i \cap \mu \neq \varnothing \textit{ for all } i \in [\tfrac{h}{2}]^{>0}. \tag{4}$$

Clearly, the order of the elements in sequence $\mu_1\mu_2\dots\mu_{\frac{h}{2}}$, as well as the presence of repetitions therein, is irrelevant. To make this formal, for a plan $\vec{\sigma} = \sigma_0\sigma_1\dots\sigma_h$ in W_n, let $set(\vec{\sigma})$ be the support of the sequence of sets $\xi(\vec{\sigma})$, i.e., $set(\vec{\sigma}) = \{\mu \mid \mu \in \xi(\vec{\sigma})\}$ is the set of elements occurring in $\xi(\vec{\sigma})$, unsorted and devoid of repetitions. Then, if two plans $\vec{\sigma}, \vec{\sigma'} \in W_n$ are such that $set(\vec{\sigma}) = set(\vec{\sigma'})$, then they are not distinguished by Λ. The converse implication does not hold in general, but it does if we restrict ourselves to plans $\vec{\sigma}$ where at every odd time instant there are exactly $\lfloor \frac{n}{2} \rfloor$ different timelines where a v_j-valued token starts (and, consequently, exactly $\lceil \frac{n}{2} \rceil$ different timelines where a $\overline{v}_j$-valued token starts instead – see Figure 2). Formally, we have the following result.

Proposition 5.1. *Let $\vec{\sigma}, \vec{\sigma'} \in W_n$ be such that*

- $|\mu| = \lfloor \frac{n}{2} \rfloor$ *for all* $\mu \in \xi(\vec{\sigma}) \cup \xi(\vec{\sigma'})$, *and*
- $set(\vec{\sigma}) \neq set(\vec{\sigma'})$.

Then, there is $\sigma \in \Lambda$ that distinguishes them.

Proof. Due to $set(\vec{\sigma}) \neq set(\vec{\sigma'})$, we have that:

- there is $\mu \in \xi(\vec{\sigma})$ such that $\mu \neq \mu'$ for all $\mu' \in \xi(\vec{\sigma'})$, or
- there is $\mu' \in \xi(\vec{\sigma'})$ such that $\mu \neq \mu'$ for all $\mu \in \xi(\vec{\sigma})$.

Assume, without loss of generality, that the former holds (the other case is dealt with analogously). Since μ has the same cardinality as every set $\mu' \in \xi(\vec{\sigma'})$, there must be an element $e_{\mu'} \in \mu' \setminus \mu$ for each $\mu' \in \xi(\vec{\sigma'})$. Let us collect these elements in a set $\mu'' = \{e_{\mu'} \mid \mu' \in \xi(\vec{\sigma'})\}$. Clearly, $\mu'' \cap \mu = \varnothing$; thus, according to (4), the concatenation of $\xi(\vec{\sigma})$ with μ'' does not represent a solution plan for P_n. In addition, for all $\mu' \in \xi(\vec{\sigma'})$ it holds that $\mu'' \cap \mu' \neq \varnothing$; thus, according to (4), the concatenation of $\xi(\vec{\sigma'})$ with μ'' represents a solution plan for P_n. The thesis follows, since $\mu'' = \xi(\sigma)$, for some $\sigma \in \Lambda$. □

Using the above proposition, showing that, for n large enough, there are more than 2^n plans in W_n that are pairwise distinguished by Λ amounts to showing that for any given set of cardinality n there are more than 2^n distinct sets of subsets of cardinality $\lfloor \frac{n}{2} \rfloor$, as formally stated in the next proposition (note that $set(\vec{\sigma})$ is a set of subsets of $[n]^{>0}$, for all $\vec{\sigma} \in W_n$).

Proposition 5.2. *For every $n \in \mathbb{N}$, with $n \geq 4$, the cardinality of the powerset of the set $\{S \subseteq [n]^{>0} \mid |S| = \lfloor \frac{n}{2} \rfloor\}$ is greater than 2^n.*

Proof. The cardinality of $\{S \subseteq [n]^{>0} \mid |S| = \lfloor \frac{n}{2} \rfloor\}$ is $\binom{n}{\lfloor \frac{n}{2} \rfloor}$. Thus, the cardinality of its powerset is $2^{\binom{n}{\lfloor \frac{n}{2} \rfloor}}$, which is strictly greater than 2^n for all $n \geq 4$. □

Theorem 5.1. *There exists a set $\mathcal{P}$ of (non-eager) qualitative timeline-based planning problems of unbounded size such that for all problems $P \in \mathcal{P}$ it holds that the smallest automaton recognizing $\mathcal{L}(P)$ has size greater than $2^{p(|P|)}$, for any polynomial p.*

6. DFA accepting plans

Given an eager qualitative timeline-based planning problem $P = (\mathsf{SV}, S)$, we show how to build a DFA $\mathcal{T}_{\mathsf{SV}}$, of size at most exponential in the size of SV (and thus the one of P), accepting words that *encode plans over* SV, i.e., words that weakly-encode plans (*cf.* Definition 4.1) and, additionally, are compatible with the domain functions V_x and comply with the constraints on the alternation of token values expressed by functions T_x, for $x \in \mathsf{SV}$. In the next section, we show how to obtain a DFA, once again of size at most exponential in the size of P, that accepts exactly the *solution plans* for P.

For every planning problem $P = (\mathsf{SV}, S)$, the DFA $\mathcal{T}_{\mathsf{SV}}$ is the tuple $\langle Q_{\mathsf{SV}}, \Sigma_{\mathsf{SV}}, \delta_{\mathsf{SV}}, q^0_{\mathsf{SV}}, F_{\mathsf{SV}}\rangle$, whose components are defined as follows.

- Q_{SV} is the finite set of *states*. Intuitively, a state of $\mathcal{T}_{\mathsf{SV}}$ keeps track of the token values of the timelines at the current and the previous step of the run. Therefore, a state is a function mapping each state variable x into a pair (v, v'), where v' (resp., v) denotes the token value of timeline x at the current (resp., previous) step. To formally define Q_{SV}, we exploit the alphabet Σ_{SV} (*cf.* Section 4): states are the alphabet symbols except for those functions $\sigma \in \Sigma_{\mathsf{SV}}$ that assign $\circlearrowleft$ to at least one state variable $x \in \mathsf{SV}$. For technical reasons, we also need a fresh *initial state* q^0_{SV} and a fresh *rejecting sink state* $\mathbf{s}_{\mathsf{SV}}$. Formally, $Q_{\mathsf{SV}} = \big(\Sigma_{\mathsf{SV}} \setminus \overline{Q}_{\mathsf{SV}}\big) \cup \{q^0_{\mathsf{SV}}, \mathbf{s}_{\mathsf{SV}}\}$, where $\overline{Q}_{\mathsf{SV}} = \{\sigma \in \Sigma_{\mathsf{SV}} \mid \sigma(x) = \circlearrowleft \text{ for some } x \in \mathsf{SV}\}$. Clearly, the size of Q_{SV} is at most as the size of Σ_{SV}, which is in turn at most exponential in the size of P.
- Σ_{SV} is the *input alphabet*, defined as in Section 4.
- $\delta_{\mathsf{SV}} : Q_{\mathsf{SV}} \times \Sigma_{\mathsf{SV}} \to Q_{\mathsf{SV}}$ is the *transition function*. Towards a definition of δ_{SV}, we say that an alphabet symbol $\sigma \in \Sigma_{\mathsf{SV}}$ is *compatible* with a state $\sigma_1 \in Q_{\mathsf{SV}}$ (we use for states the same symbols as for the alphabet, i.e., $\sigma, \sigma_1, \sigma_2, \ldots$, to stress the fact that states are closely related to alphabet symbols) if one of the following holds: *(i)* $\sigma_1 = q^0_{\mathsf{SV}}$ is the initial state and $\sigma \in \Sigma^I_{\mathsf{SV}}$ is an initial symbol such that for each $x \in \mathsf{SV}$ it holds that $\sigma(x) = (-, v)$ with $v \in V_x$; *(ii)* $\sigma_1 \in \Sigma_{\mathsf{SV}} \setminus \overline{Q}_{\mathsf{SV}}$ and $\sigma \in \Sigma^N_{\mathsf{SV}}$ is a non-initial alphabet symbol such that for each $x \in \mathsf{SV}$ either $\sigma(x) = \circlearrowleft$ or $\sigma_1(x) = (v, v')$ and $\sigma(x) = (v', v'')$ with $v'' \in T_x(v') \cap V_x$. Now, $\delta_{\mathsf{SV}} : Q_{\mathsf{SV}} \times \Sigma_{\mathsf{SV}} \to Q_{\mathsf{SV}}$ is defined as follows. For all $\sigma_1 \in Q_{\mathsf{SV}}$ and $\sigma \in \Sigma_{\mathsf{SV}}$, if σ is not compatible with σ_1 or σ_1 is the sink state (i.e., $\sigma_1 = \mathbf{s}_{\mathsf{SV}}$), then $\delta(\sigma_1, \sigma) = \mathbf{s}_{\mathsf{SV}}$; otherwise
  - if σ_1 is the initial state (i.e., $\sigma_1 = q^0_{\mathsf{SV}}$), then $\delta(\sigma_1, \sigma) = \sigma$; in other words, in this case the automaton transitions to the state represented by the letter that is being read;
  - if $\sigma_1 \in \Sigma_{\mathsf{SV}} \setminus \overline{Q}_{\mathsf{SV}}$, then $\delta(\sigma_1, \sigma) = \sigma_2$, where $\sigma_2(x) = \sigma_1(x)$ if $\sigma(x) = \circlearrowleft$, and $\sigma_2(x) = \sigma(x)$ otherwise, for all $x \in \mathsf{SV}$; intuitively, the automaton transitions into a state that updates the information about tokens whose value is changed, according to the letter that is being read.

  We point out that, in both cases, the automaton transitions to the next state in a deterministic fashion.
- $F_{\mathsf{SV}} = Q_{\mathsf{SV}} \setminus \{\mathbf{s}_{\mathsf{SV}}\}$ is the set of *final states*.

Correctness of the DFA $\mathcal{T}_{\mathsf{SV}}$ is proved by the next lemma.

Lemma 6.1. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem. Then, words accepted by $\mathcal{T}_{\mathsf{SV}}$ are exactly those encoding plans over SV. Moreover the size of $\mathcal{T}_{\mathsf{SV}}$ is at most exponential in the size of P.*

Proof. Let W denote the set of words that weakly-encode plans over SV (in the sense of Definition 4.1) and additionally satisfy the domain and transition constraints, that is, for each word $\lambda = \langle \sigma_0, \dots, \sigma_{|\lambda|-1} \rangle \in W$, each $x \in \mathsf{SV}$, and each position $i \in \mathit{changes}(x)$, we have $\vec{\sigma}_i^x \in V_x$ and, if i is not the first position in $\mathit{changes}(x)$, then $\vec{\sigma}_i^x \in T_x(\overleftarrow{\sigma}_i^x)$. We establish that $\mathcal{L}(\mathcal{T}_{\mathsf{SV}}) = W$, which precisely corresponds to the set of words encoding plans over SV. We prove the equivalence by showing both containments $\mathcal{L}(\mathcal{T}_{\mathsf{SV}}) \subseteq W$ and $W \subseteq \mathcal{L}(\mathcal{T}_{\mathsf{SV}})$ using contradiction arguments based on the minimum index where violations occur.

To show $\mathcal{L}(\mathcal{T}_{\mathsf{SV}}) \subseteq W$, let us suppose, towards a contradiction, that there exists a word $\lambda = \langle \sigma_0, \sigma_1, \dots, \sigma_{n-1} \rangle \in \mathcal{L}(\mathcal{T}_{\mathsf{SV}})$ such that $\lambda \notin W$. Since $\mathcal{T}_{\mathsf{SV}}$ accepts λ, there is an accepting run $q^0_{\mathsf{SV}} \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{n-1}} q_n$ with $q_n \in F_{\mathsf{SV}}$. Let i be the minimum index such that the constraints defining W are violated at position i. We consider the possible violations. If $i = 0$ and either $\sigma_0 \notin \Sigma^I_{\mathsf{SV}}$ or $\sigma_0(x) = (-, v)$ with $v \notin V_x$ for some $x \in \mathsf{SV}$, then symbol σ_0 is not compatible with the initial state q^0_{SV}, and thus the transition $\delta_{\mathsf{SV}}(q^0_{\mathsf{SV}}, \sigma_0)$ leads to the sink state $\mathbf{s}_{\mathsf{SV}}$, contradicting the fact that λ is accepted. If $i > 0$, then q_i is not the initial state, and in order for symbol σ_i to be compatible with q_i, it must be $\sigma_i \in \Sigma^N_{\mathsf{SV}}$ (σ_i is a non-initial alphabet symbol). If the violation occurs because the weak encoding condition fails (i.e., $\vec{\sigma}^x_{i_{h-1}} \neq \overleftarrow{\sigma}^x_{i_h}$ for some consecutive changes), or because domain constraints are violated ($\vec{\sigma}_i^x \notin V_x$), or because transition constraints fail ($\vec{\sigma}_i^x \notin T_x(\overleftarrow{\sigma}_i^x)$), then again symbol σ_i is not compatible with state q_i, and transition $\delta_{\mathsf{SV}}(q_i, \sigma_i)$ leads to the sink state $\mathbf{s}_{\mathsf{SV}}$, again contradicting acceptance of λ. Since we reach a contradiction in all cases, we conclude $\mathcal{L}(\mathcal{T}_{\mathsf{SV}}) \subseteq W$.

To show $W \subseteq \mathcal{L}(\mathcal{T}_{\mathsf{SV}})$, let us suppose, towards a contradiction, that there exists a word $\lambda = \langle \sigma_0, \sigma_1, \dots, \sigma_{n-1} \rangle \in W$ such that $\lambda \notin \mathcal{L}(\mathcal{T}_{\mathsf{SV}})$. Consider the unique run of $\mathcal{T}_{\mathsf{SV}}$ on λ: $q^0_{\mathsf{SV}} \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{n-1}} q_n$. Since λ is not accepted, from the final states being $Q_{\mathsf{SV}} \setminus \{\mathbf{s}_{\mathsf{SV}}\}$ (i.e., all the states but the sink one) we have that the run eventually hits the sink state $\mathbf{s}_{\mathsf{SV}}$. Let i be the minimum index such that $q_i = \mathbf{s}_{\mathsf{SV}}$. We can assume that $i > 0$ since the fresh initial state q^0_{SV} is distinct from the sink one. Then, we have $\delta_{\mathsf{SV}}(q_i, \sigma_i) = \mathbf{s}_{\mathsf{SV}}$. This happens exactly when σ_i is not compatible with state q_i. However, since $\lambda \in W$, all the constraints that define compatibility are satisfied: the weak encoding condition and the domain/transition constraints ensure compatibility. This contradicts the incompatibility detected by the automaton, and thus we can conclude $W \subseteq \mathcal{L}(\mathcal{T}_{\mathsf{SV}})$.

Finally, for the bound on the size of the automaton, we have that the size of $\mathcal{T}_{\mathsf{SV}}$ is at most polynomial in $|Q_{\mathsf{SV}}|$ and $|\Sigma_{\mathsf{SV}}|$. From the construction, we have $|Q_{\mathsf{SV}}| \leq |\Sigma_{\mathsf{SV}}| = |\Sigma^I_{\mathsf{SV}}| + |\Sigma^N_{\mathsf{SV}}| = |V|^{|\mathsf{SV}|} + (|V|^2 + 1)^{|\mathsf{SV}|}$, where $V = \cup_{x \in \mathsf{SV}} V_x$. Since each domain V_x has size polynomial in the size of P, the total size $|V| = |\bigcup_{x \in \mathsf{SV}} V_x|$ is polynomial in $|P|$. Then, $|\Sigma_{\mathsf{SV}}|$, and thus $|Q_{\mathsf{SV}}|$ as well, is at most exponential in $|P|$. □

7. DFA accepting solution plans

In this section, we go through the construction of an automaton recognizing solution plans for a planning problem. Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem, and let $V = \cup_{x \in \mathsf{SV}} V_x$. We show how to build a DFA $\mathcal{A}_P$, whose size is at most exponential in the size of P, that accepts exactly those words encoding solutions plans for P when restricted to words encoding plans over SV. In different terms, if a word encodes a plan over SV, then it is accepted by $\mathcal{A}_P$ if and only if it encodes a solution plan for P.

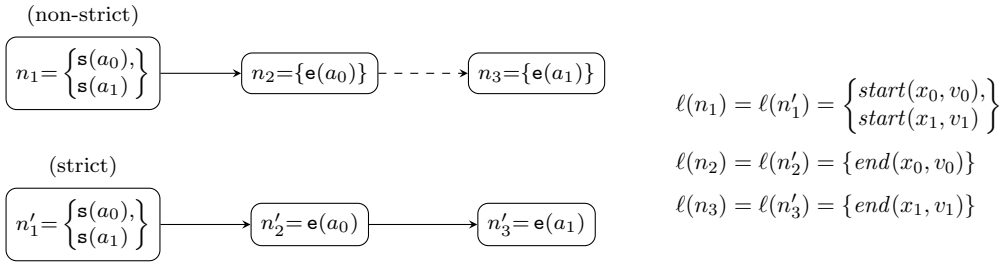


Figure 3: The picture shows two DAGs on the left-hand side. The one on the top is associated with (the existential statement of) the rule $a_0[x_0 = v_0] \rightarrow \exists a_1[x_1 = v_1].(\mathtt{s}(a_0) \leq \mathtt{s}(a_1) \wedge \mathtt{s}(a_1) \leq \mathtt{s}(a_0) \wedge \mathtt{e}(a_0) \leq \mathtt{e}(a_1))$. It forces token a_0 to either be a prefix of or coincide with token a_1. The one on the bottom is associated with the existential statement obtained replacing $\mathtt{e}(a_0) \leq \mathtt{e}(a_1)$ with $\mathtt{e}(a_0) < \mathtt{e}(a_1)$, that forces a_0 to be a (strict) prefix of a_1. The labeling function is the same for both DAGs, and it is shown on the right-hand side.

However, $\mathcal{A}_P$ may also accept words that do not encode a plan over SV. Therefore, we need the intersection of such a DFA $\mathcal{A}_P$ with DFA $\mathcal{T}_{\mathsf{SV}}$ from the previous section.

Since, as already noticed, the ordering relations imposed by conjunctions of atoms in synchronization rules induce preorders, in the following we use labeled directed acyclic graphs (labeled DAGs) to represent them. Intuitively, each conjunction of atoms $\mathcal{C}$ identifies a DAG whose vertices are the equivalence classes of terms $\mathtt{s}(a)/\,\mathtt{e}(a)$ occurring in it. Formally, let $\mathcal{R}$ be a synchronization rule, and let $\mathcal{E}$ and $\mathcal{C}$ be, respectively, the existential statement and the conjunction of atoms of $\mathcal{R}$. For each term t occurring in $\mathcal{C}$ we denote by $[t]_{\equiv_\mathcal{C}}$ its *equivalence class with respect to* $\mathcal{C}$, defined as the set $\{u \mid t \equiv u \in \hat{\mathcal{C}}\}$. We omit the subscript $_\mathcal{C}$ when it is clear from the context. Moreover, if $t = \mathtt{s}(a)$ (resp., $t = \mathtt{e}(a)$) and $a[x = v]$ either occurs in (the quantifier prefix of) $\mathcal{E}$ or is the trigger of $\mathcal{R}$, then *term-to-event*$(t) = start(x, v)$ (resp., *term-to-event*$(t) = end(x, v)$); with an abuse of notation, we lift function *term-to-event* to the domain of sets of terms, i.e., *term-to-event*$(T) = \{$*term-to-event*$(t) \mid t \in T\}$, for all sets of terms T. The *(labeled) DAG associated with* $\mathcal{R}$, denoted $G_\mathcal{R}$, is defined as the tuple $(V_\mathcal{R}, A_\mathcal{R}, A^<_\mathcal{R}, \ell_\mathcal{R})$, where (we omit the subscript $_\mathcal{R}$ when clear from the context):

- $V = \{[t]_\equiv \mid t$ is a term occurring in $\mathcal{C}\}$,
- $A = \{([t]_\equiv, [u]_\equiv) \in V \times V \mid [t]_\equiv \neq [u]_\equiv$ and $t \leq u \in \hat{\mathcal{C}}\}$,
- $A^< = \{([t]_\equiv, [u]_\equiv) \in A \mid t < u \in \hat{\mathcal{C}}\}$, and
- ℓ is the *labeling function* assigning to each vertex $[t]_\equiv \in V$ its *label* $\ell([t]_\equiv) =$ *term-to-event*$([t]_\equiv)$.

Notice that labeling function ℓ is crucial to establish a correspondence between vertices of the DAG and sets of events *events*(σ) associated with alphabet symbols σ. It is convenient to further lift function *term-to-event* to deal with sets of vertices (i.e., sets of sets of terms): *term-to-event*$(V') = \bigcup_{[t]_\equiv \in V'}$ *term-to-event*$([t]_\equiv)$ for all $V' \subseteq V$. When convenient, we may use dashed and solid arrows to denote arcs in A and $A^<$, respectively, thus writing $[t]_\equiv \dashrightarrow [u]_\equiv$ and $[t]_\equiv \rightarrow [u]_\equiv$ for $([t]_\equiv, [u]_\equiv) \in A$ and $([t]_\equiv, [u]_\equiv) \in A_<$, respectively. We

likewise use $\not\dashrightarrow$ and $\not\Rightarrow$ to denote the absence of the corresponding arcs. Figure 3 shows such a difference.

7.1. **Viewpoints.** In order to obtain the desired automaton construction, it is useful to introduce an auxiliary structure, called viewpoint. A *viewpoint* $\mathbb{V}$ for $\mathcal{R}$ is a pair (G, K), where $G = (V, A, A^<, \ell)$ is the (labeled) DAG associated with $\mathcal{R}$ and $K \subseteq V$ is a *downward closed* subset of vertices of G, that is, $n \in K$ implies $n' \in K$ for all $n, n' \in V$ with $n' \dashrightarrow n$. The number of different viewpoints for $\mathcal{R}$ is $2^{|V|}$, hence at most exponential in the size of P. If $K = \varnothing$, then $\mathbb{V} = (G, K)$ is *initial*; analogously, if K is the entire set of vertices of G, then $\mathbb{V}$ is *final*. If $\mathbb{V}$ is a viewpoint for $\mathcal{R}$, then we use *rule*$(\mathbb{V})$ to refer to $\mathcal{R}$. Intuitively, the downward closed component of a viewpoint for a synchronization rule is meant to collect the events that are relevant to witness the satisfaction of the rule, among those that are encoded in the letters of the input word that have been read so far. In other words, a viewpoint retains information about relevant tokens encountered so far along the plan. In what follows, we describe how viewpoints evolve.

According to the formalization provided below, states of automata $\mathcal{A}_P$ are sets of viewpoints containing at least one viewpoint for each rule of P (besides a fresh rejecting sink state $\mathbf{s}_P$). Therefore, to define automata runs, we first show how viewpoints evolve upon reading an alphabet symbol. To this end, we introduce the following notions. Let $\mathbb{V} = (G, K)$ be a viewpoint, with $G = (V, A, A^<, \ell)$. We set *next-available*$(\mathbb{V}) = K'$, where K' is the largest downward closed subset of V for which there is no pair of vertices $v, v' \in K' \setminus K$ with $v \to v'$. Notice that it is always the case that *next-available*$(\mathbb{V}) \supseteq K$. Moreover, given an alphabet symbol $\sigma \in \Sigma_{\mathsf{SV}}$, we define *consumed*$(\mathbb{V}, \sigma) = K'$, where K' is the largest downward closed subset of vertices of *next-available*$(\mathbb{V})$ such that *term-to-event*$(K' \setminus K) \subseteq$ *events*(σ). Once again, it holds that *consumed*$(\mathbb{V}, \sigma) \supseteq K$. Intuitively, a viewpoint $\mathbb{V} = (G, K)$ evolves by suitably extending K; to this end, *next-available*$(\mathbb{V})$ identifies the only vertices that can possibly be added to K (independently from the alphabet symbol read), that is, vertices of $V \setminus K$ reachable from K with no arcs in $A_<$ (solid arrows) among them. The exact extension, however, depends on the actual symbol σ read by the automaton: K cannot be extended with events that are not included in σ. Therefore, *consumed*$(\mathbb{V}, \sigma)$ identifies precisely how viewpoint $\mathbb{V}$ evolves upon reading σ.

There are events that *must* be consumed by a viewpoint $\mathbb{V}$ when certain symbols (i.e., symbols that encode, among others, such events) are read by the automaton. We aim to collect such events in a set called the *waiting list* of $\mathbb{V}$, defined in the following. The idea is that an automaton must reject if an event in the waiting list of $\mathbb{V}$ is read (through the symbol σ) and the viewpoint is not *ready* to consume it (i.e., the event is not featured in *consumed*$(\mathbb{V}, \sigma)$). Let the *past* of $\mathbb{V}$, denoted *past*$(\mathbb{V})$, be the set $\bigcup_{[t]_\equiv \in K} [t]_\equiv$, and the *future* of $\mathbb{V}$, denoted *future*$(\mathbb{V})$, be the set $\bigcup_{[t]_\equiv \in V \setminus K} [t]_\equiv$. Then, one would expect the waiting list of $\mathbb{V}$ to include those events $\mathtt{e}(a)$ (token ending) that are in the future of $\mathbb{V}$, but whose corresponding $\mathtt{s}(a)$ (token beginning) is in the past of $\mathbb{V}$, that is, the beginning of a token has been read (and consumed), but its ending has not yet. This intuition is quite correct, but there is an exception. Let $\mathcal{C}$ be the unique clause occurring in the rule corresponding to $\mathbb{V}$, namely *rule*$(\mathbb{V})$, and let $\mathtt{s}(a) \in$ *past*$(\mathbb{V})$ and $\mathtt{e}(a) \in$ *future*$(\mathbb{V})$. Then, $\mathtt{e}(a)$ is *not* included

in the waiting list of $\mathbb{V}$ if a is not the trigger token of *rule*$(\mathbb{V})$ and there is no term $t \neq \mathtt{s}(a)$ such that $\mathtt{s}(a) \leq t \in \hat{\mathcal{C}}$ and $\mathtt{e}(a) \leq t \notin \hat{\mathcal{C}}$. The idea is that if there is no term t that is required to occur after (or together with) $\mathtt{s}(a)$ and possibly before than $\mathtt{e}(a)$, then it is safe for the automaton to be indulgent, and allow the viewpoint to ignore the event $\mathtt{e}(a)$ that matches with $\mathtt{s}(a)$ and wait instead for some future $\mathtt{e}(a)$ event (that refers to the end of a different token than the one $\mathtt{s}(a)$ refers to), thus having non-matching $\mathtt{s}(a)$ and $\mathtt{e}(a)$. Intuitively, the reason why it is safe for the automaton to be indulgent is that, under these conditions, the positions of the non-matching event $\mathtt{s}(a)$ and the event $\mathtt{s}(a)$ that matches the future $\mathtt{e}(a)$ coincide relative to the other events. This is formally captured by defining the *waiting list* of $\mathbb{V}$, denoted *waiting*$(\mathbb{V})$, as the set

$$\left\{ \mathtt{e}(a) \mid \begin{array}{l} \mathtt{s}(a) \in \mathit{past}(\mathbb{V}), \mathtt{e}(a) \in \mathit{future}(\mathbb{V}), \text{ and at least one of the following holds:} \\ \quad \bullet\ a \text{ is the trigger token of } \mathit{rule}(\mathbb{V}), \\ \quad \bullet\ \text{there is a term } t \notin \{\mathtt{s}(a), \mathtt{e}(a)\} \text{ such that } \mathtt{s}(a) \leq t \in \hat{\mathcal{C}} \text{ and } \mathtt{e}(a) \leq t \notin \hat{\mathcal{C}} \end{array} \right\}$$

where $\mathcal{C}$ is the unique clause occurring in *rule*$(\mathbb{V})$. We finally say that $\mathbb{V}$ is *compatible* with symbol σ if *term-to-event*(*waiting*$(\mathbb{V})) \cap$ *events*$(\sigma) \subseteq$ *term-to-event*(*consumed*$(\mathbb{V}, \sigma) \setminus K)$. Observe that this last notion of compatibility ensures that a viewpoint is allowed to evolve upon reading a symbol only if no token ending event in the waiting list is overlooked (in other words, events read by the automaton that belong to the waiting list *must* be consumed).

We can now define the *evolution of a viewpoint* $\mathbb{V} = (G, K)$ upon reading an alphabet symbol $\sigma \in \Sigma_{\mathsf{SV}}$, denoted *evol*$(\mathbb{V}, \sigma)$, as the viewpoint $(G,$ *consumed*$(\mathbb{V}, \sigma))$, if $\mathbb{V}$ is compatible with σ; *evol*$(\mathbb{V}, \sigma)$ is undefined otherwise.

7.2. **States, initial state, and final states of $\mathcal{A}_P$.** We have already mentioned that *states* of $\mathcal{A}_P$ are sets of viewpoints containing at least one viewpoint for each rule $\mathcal{R} \in S$ (recall that S is the set of rules in planning problem P), besides a fresh rejecting sink state $\mathbf{s}_P$. However, since it is crucial for us to bound the size of $\mathcal{A}_P$ to be at most exponential in the one of P, we impose the *linearity condition*, formalized in what follows. To this end, it is useful to assume, without loss of generality, that different rules of P involves disjoint sets of token names. For each rule $\mathcal{R} \in S$, let $\Upsilon_{\mathcal{R}}$ be the set of all viewpoints for $\mathcal{R}$, and let $\Upsilon_P = \bigcup_{\mathcal{R} \in S} \Upsilon_{\mathcal{R}}$. We define an ordering relation $\preceq$ between viewpoints: for all $\mathbb{V}, \mathbb{V}' \in \Upsilon_P$, with $\mathbb{V} = (G, K)$ and $\mathbb{V}' = (G', K')$, it holds that $\mathbb{V} \preceq \mathbb{V}'$ if and only if *(i)* $G = G'$, that is, $\mathbb{V}, \mathbb{V}' \in \Upsilon_{\mathcal{R}}$ for some $\mathcal{R} \in S$, and *(ii)* $K \subseteq K'$. Intuitively, $\mathbb{V} \preceq \mathbb{V}'$ captures the fact that $\mathbb{V}'$ has gone further than $\mathbb{V}$ in matching input symbols with the beginning/end of a token, in order to satisfy a rule.

At this point, we can formalize the linearity condition, crucial to constrain the size of $\mathcal{A}_P$ (Lemma 7.1).

Definition 7.1 (Linearity condition)**.** A set of viewpoints Υ satisfies the *linearity condition* if for all viewpoints $\mathbb{V}, \mathbb{V}' \in \Upsilon$ and rules $\mathcal{R} \in S$, if $\mathbb{V}, \mathbb{V}' \in \Upsilon_{\mathcal{R}}$, then $\mathbb{V} \preceq \mathbb{V}'$ or $\mathbb{V}' \preceq \mathbb{V}$ holds.

Intuitively, we impose all viewpoints for the same rule in a state of $\mathcal{A}_P$ to be linearly ordered.

We are now ready to formally characterize the set of *states* of $\mathcal{A}_P$, consisting of the sets $\Upsilon \subseteq \Upsilon_P$ of viewpoints that contain at least one viewpoint for each rule $\mathcal{R} \in S$ and that satisfy the linearity condition, and including, in addition, a fresh *rejecting sink state* $\mathbf{s}_P$. We denote it by Q_P; formally, $Q_P = \{\mathbf{s}_P\} \cup \{\Upsilon \subseteq \Upsilon_P \mid \Upsilon \cap \Upsilon_{\mathcal{R}} \neq \varnothing \text{ for all } \mathcal{R} \in S \text{ and } \Upsilon \text{ satisfies the linearity condition}\}$. The *initial state* q_P^0 of $\mathcal{A}_P$ is the set $\{\mathbb{V}_{\mathcal{R}}^0 \mid \mathcal{R} \in$

$S\}$, where $\mathbb{V}^0_{\mathcal{R}}$ is the initial viewpoint of rule $\mathcal{R}$. Towards a definition of the set F_P of *final states* of $\mathcal{A}_P$, we introduce the notion of *enabled viewpoints*. A viewpoint $\mathbb{V} = (G, K)$ for rule $\mathcal{R} \in S$ is *enabled* if either $\mathcal{R}$ is triggerless or $\mathcal{R}$ has trigger token a_0 and $[\mathtt{s}(a_0)]_{\equiv} \in K$. A state q of $\mathcal{A}_P$ is *final* if it is not the rejecting sink state $\mathbf{s}_P$ and every enabled viewpoint therein is final.

7.3. **Transition function of $\mathcal{A}_P$.** The last step of our construction is the definition of the *transition function* δ_P for automaton $\mathcal{A}_P$. To this end, we first introduce the notion of alphabet symbol *enabling* a viewpoint $\mathbb{V}$, along with the one of state of $\mathcal{A}_P$ *compatible* with an alphabet symbol. Let $\mathbb{V}$ be a viewpoint for a non-triggerless rule $\mathcal{R}$ with trigger token a_0 and $\sigma \in \Sigma_{\mathsf{SV}}$ an alphabet symbol. We say that σ *enables* $\mathbb{V} = (G, K)$ if $[\mathtt{s}(a_0)]_{\equiv} \in \mathit{consumed}(\mathbb{V}, \sigma) \setminus K$. Moreover, we say that a state $q \in Q_P \setminus \{\mathbf{s}_P\}$ is *compatible* with σ if *(i)* all viewpoints in q are compatible with σ and *(ii)* for all non-triggerless rules $\mathcal{R} \in S$, if σ *triggers* $\mathcal{R}$ (i.e., $\mathit{start}(x_0, v_0) \in \mathit{events}(\sigma)$ with $a_0[x_0 = v_0]$ trigger token of $\mathcal{R}$), then there is a viewpoint $\mathbb{V} \in q$ for $\mathcal{R}$ such that σ enables $\mathbb{V}$.

We are now ready to define the *transition function* δ_P of $\mathcal{A}_P$. For all $q \in Q_P$ and alphabet symbol $\sigma \in \Sigma_{\mathsf{SV}}$:

- if $q = \mathbf{s}_P$ or q is not compatible with σ, then $\delta(q, \sigma) = \mathbf{s}_P$;
- otherwise, $\delta(q, \sigma) = q'$, where q' is the smallest set such that for all $\mathbb{V} = (G, K) \in q$
  - $\mathit{evol}(\mathbb{V}, \sigma) \in q'$ and
  - if σ enables $\mathbb{V}$, then $(G, K') \in q'$, where $K' = \mathit{consumed}(\mathbb{V}, \sigma) \setminus \{[t]_{\equiv} \mid \mathtt{s}(a_0) \leq t \in \hat{\mathcal{C}}\}$ and a_0 is the trigger token of $\mathit{rule}(\mathbb{V})$.

As an example, Figure 4 and Figure 5 depict the automata $\mathcal{A}_P$ for two single-rule planning problems whose associated DAGs are shown in Figure 3 (non-strict and strict case, respectively). Each state q_i is annotated with the set of downward closed subsets of DAG vertices representing the viewpoint component K (the graph of the viewpoints is always the same, i.e., the one shown in Figure 3). Transition labels encode conditions on the alphabet symbol σ being read, expressed as pairs of constraints on events for timelines x_0 and x_1; v_i denotes the presence of the relevant event, $\neg v_i$ its absence, $*$ no constraints, and $\circlearrowleft$ that the corresponding timeline preserves its current value. The strict case (Figure 5) requires six non-sink states, compared to four non-sink states for the non-strict case (Figure 4), reflecting the finer tracking needed to enforce the strict ordering constraint $\mathtt{e}(a_0) < \mathtt{e}(a_1)$.

The proof of the following lemma is quite technical and notationally heavy. Therefore, its proof is given in Appendix A.

Lemma 7.1. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem. Each finite word over Σ_{SV} that encodes a plan over SV is accepted by $\mathcal{A}_P$ if and only if it encodes a solution plan for P. Moreover, the size of $\mathcal{A}_P$ is at most exponential in the size of P.*

Our final result follows from Lemma 6.1 and Lemma 7.1.

Theorem 7.1. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem. Then, the words accepted by the intersection automaton of $\mathcal{A}_P$ and $\mathcal{T}_{\mathsf{SV}}$ are exactly those encoding solution plans for P. Moreover, the size of the intersection automaton of $\mathcal{A}_P$ and $\mathcal{T}_{\mathsf{SV}}$ is at most* exponential *in the size of P.*

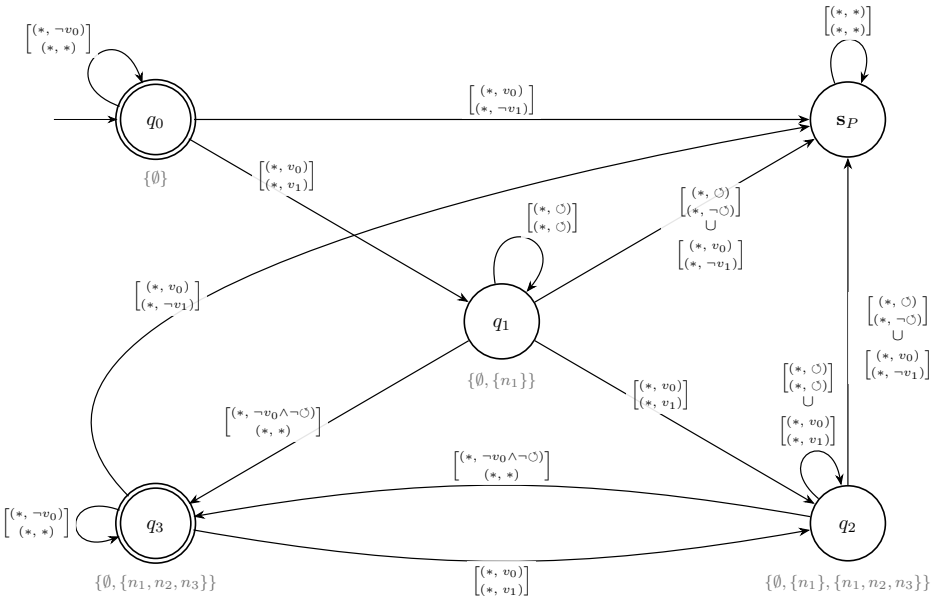


Figure 4: The automaton $\mathcal{A}_P$ for the non-strict rule whose DAG is depicted in Figure 3 (non-strict), with four non-sink states.

Theorem 7.1 enables the classical strategy synthesis procedure from a DFA arena [PR89]. Once the DFA is obtained, one solves a reachability/safety game on it via classical fixpoint algorithms.

8. Expressiveness of the eager fragment: A BPMN case study

Having established the theoretical foundations of the eager fragment and demonstrated its decidability properties through deterministic finite automata, we now turn to examining its practical expressiveness. To illustrate the modeling capabilities of the eager fragment, we present a comprehensive case study involving the translation of Business Process Model and Notation (BPMN) diagrams into timeline-based planning problems. BPMN provides a rich set of control flow constructs including sequential flows, parallel execution, exclusive choice (XOR), and loops, making it an ideal test-bed for demonstrating that the eager fragment can capture complex real-world process semantics. Through this translation, we show how various BPMN constructs can be systematically encoded using eager synchronization rules, thereby demonstrating the expressiveness of the fragment for representing sophisticated temporal and control flow constraints.

More precisely, the section is articulated in four stages. First, we introduce BPMN through a concrete example that illustrates the key modeling constructs and their intended semantics. Second, we introduce the concept of parse trees that capture the hierarchical decomposition of the well-structured class of BPMN diagrams into Single Entry Single Exit

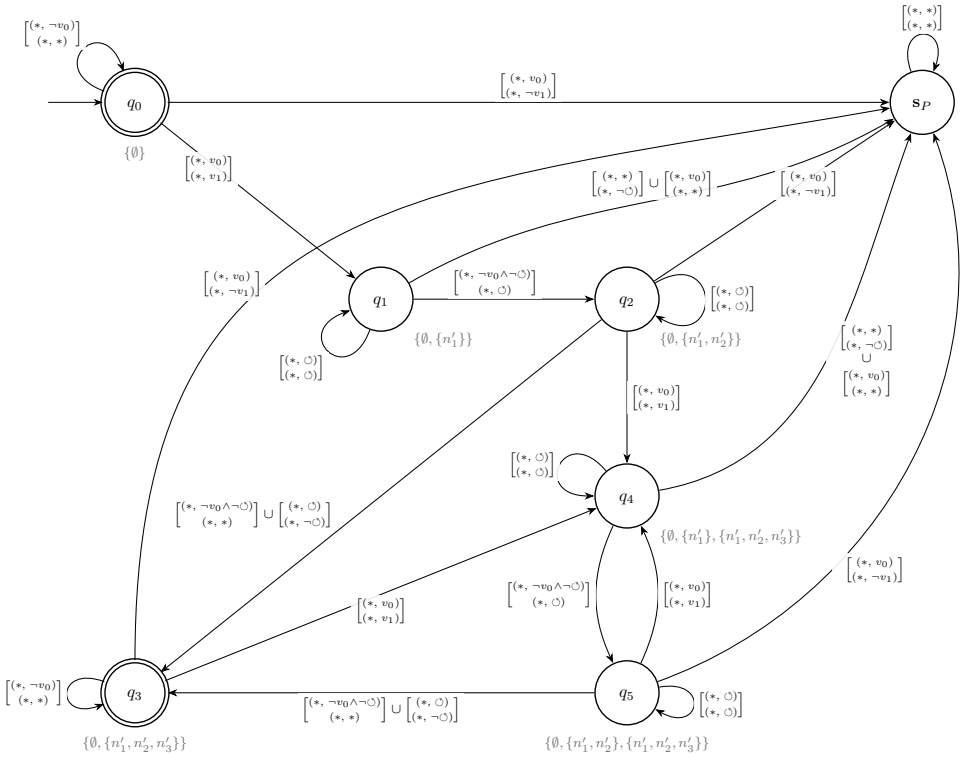


Figure 5: The automaton $\mathcal{A}_P$ for the strict rule whose DAG is depicted in Figure 3 (strict), with six non-sink states.

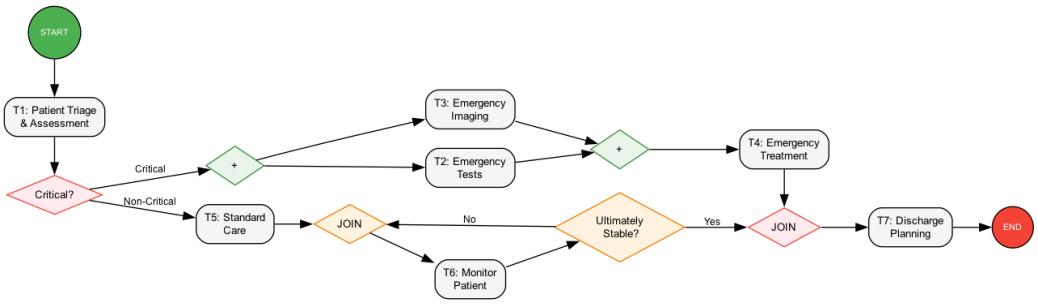


Figure 6: Emergency Department BPMN Process Diagram

(SESE) blocks, which are the building blocks that will be compiled into the eager fragment. Third, we present the general translation methodology that systematically maps each type of BPMN construct to corresponding eager timeline synchronization rules. Finally, we apply this translation framework to our working example, demonstrating how the resulting timeline-based planning problem preserves the original process semantics while remaining within the eager fragment.

To illustrate our translation approach, we present a simplified Emergency Department process that demonstrates all major BPMN control flow patterns. The process shown in Figure 6 models patient flow through an emergency department. Upon arrival, every patient

undergoes an initial triage and assessment phase (T1) where medical staff evaluate the patient condition and assign a priority level. Based on this assessment, patients are classified through an XOR gateway that routes them along different treatment paths according to the severity of their condition. Critical patients follow an intensive care pathway where emergency tests (T2) and emergency imaging (T3) are performed concurrently to rapidly diagnose the condition, after which emergency treatment (T4) is administered. Non-critical patients instead follow a standard care pathway where they receive routine medical care (T5) and are subsequently monitored (T6) in a loop structure that continues until their condition stabilizes. Regardless of which treatment path is followed, all patients eventually proceed to discharge planning (T7) where arrangements are made for their release or transfer to appropriate care facilities.

To systematically translate BPMN diagrams into timeline-based planning problems, we employ a structural decomposition approach based on parse trees and Single Entry Single Exit (SESE) blocks. A SESE block is a process fragment that has exactly one entry point and one exit point, meaning that control flow can only enter the block through a single incoming edge and can only leave through a single outgoing edge. This structural constraint ensures that SESE blocks are compositional units that can be translated independently as we will see.

The hierarchical structure of well-formed BPMN diagrams naturally decomposes into a parse tree where each node represents a SESE block of a specific type. Figure 7 shows the complete SESE decomposition of our Emergency Department process. We distinguish five fundamental block types that capture the essential BPMN control flow patterns. Task blocks (b_6, b_8, b_{11}, b_{12}, b_{13}, b_{15}, b_{16}) represent atomic activities that cannot be further decomposed. Flow blocks (b_1, b_2, b_4, b_7) capture sequential execution where exactly two sub-blocks are executed in strict temporal order, with a "before" child that must complete prior to the "after" child beginning execution. Parallel blocks (b_5) model concurrent execution where exactly two branches are activated simultaneously and execution continues only after both branches complete. Loop blocks (b_9) represent iterative structures where a body block may be executed multiple times based on continuation conditions. XOR blocks (b_3) implement exclusive choice where exactly one of two alternative branches is selected for execution based on process state or external conditions. When more than two sequential steps, parallel branches, or choice alternatives are required, they can be obtained by chaining multiple blocks of the same type, just as we demonstrate with the flow blocks b_1 and b_2 in our decomposition of Figure 7.

The parse tree structure in Figure 7 reveals the hierarchical organization of the process through its binary decomposition. The root block b_1 represents the entire process as a flow block containing two sequential phases: the assessment and classification phase (b_2) executed before the final discharge planning (b_{16}). The assessment phase b_2 further decomposes into initial patient triage (b_{15}) followed by critical classification (b_3). The XOR block b_3 implements the branching logic that routes patients to either the critical path (b_4) or non-critical path (b_7) based on their condition severity. The critical path b_4 sequences emergency procedures (b_5) before emergency treatment (b_6), where the emergency procedures block coordinates concurrent execution of emergency tests (b_{11}) and imaging (b_{12}). The non-critical

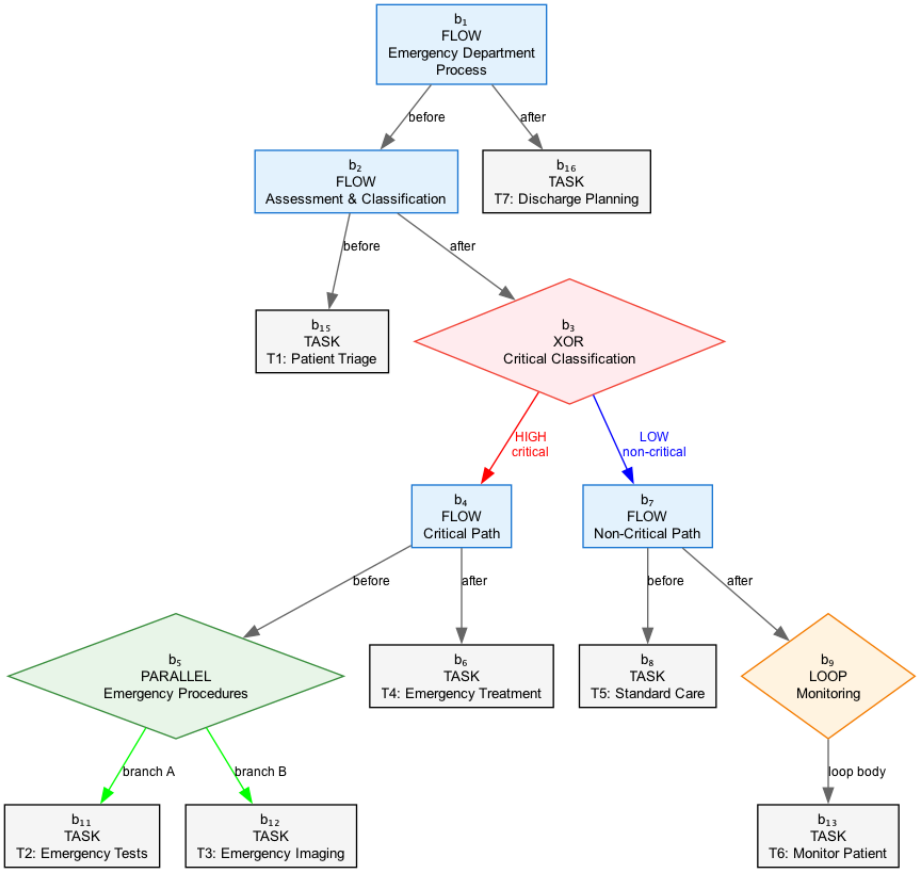


Figure 7: SESE Block Decomposition Tree for Emergency Department Process

path b_7 sequences standard care (b_8) before the monitoring loop (b_9) containing the patient monitoring task (b_{13}). This decomposition enables systematic translation where each SESE block type maps to specific eager timeline synchronization rules that preserve the intended control flow semantics.

Let us provide now the general translation rules, then we will apply them to our working example. The construction of BPMN semantics into timeline-based planning is not a novel concept. In [COS19], the authors present a comprehensive approach that utilizes the full quantitative fragment of timeline-based planning, which allows for disjunctions and encompasses all possible relations among temporal points within a clause. However, this comprehensive expressiveness comes at a significant computational cost, as the resulting planning problems are EXPSPACE-complete. In contrast, the encoding proposed in this work adopts a more restrictive but computationally efficient approach. By limiting ourselves to the eager fragment of qualitative timeline-based planning, we achieve a more tractable complexity class of PSPACE-complete, while still maintaining sufficient expressiveness to capture all major BPMN control flow patterns as we demonstrate in this section.

To systematically translate BPMN diagrams into timeline-based planning problems, we define a translation function $\mathcal{R}$ that maps each SESE block to a set of eager timeline

synchronization rules. This function decomposes into two complementary components:

$$\mathcal{R}(b) = \mathcal{R}_{\text{forward}}(b) \cup \mathcal{R}_{\text{backward}}(b).$$

This decomposition into forward and backward rules serves just for conceptual clarity with the dichotomy determined by which block serves as the trigger in each synchronization rule:

- rules in $\mathcal{R}_{\text{forward}}(b)$ are triggered by parent blocks (rules of the form $a_0[x_{\text{parent}} = \top] \to \ldots$) and capture the natural control flow semantics where parent blocks orchestrate the activation patterns of their children, implementing the specific behavioral constraints of each block type.
- rules in $\mathcal{R}_{\text{backward}}(b)$ are triggered by child blocks (rules of the form $a_0[x_{\text{child}} = \top] \to \ldots$) and establish the essential parent-child dependency relationships, ensuring that child blocks can only execute when their parent blocks permit such activation.

Given a SESE tree decomposition with blocks $B = \{b_1, b_2, \ldots, b_n\}$, the complete timeline-based planning problem encoding the original BPMN diagram is defined as:

$$P = (\mathcal{SV}, \mathcal{S})$$

where $\mathcal{SV}$ is the set of all state variables introduced for the blocks, and $\mathcal{S}$ is the complete set of synchronization rules:

$$\mathcal{S} = \bigcup_{b \in B} \mathcal{R}(b) \cup \{\exists t[x_{b_{\text{root}}} = \top].\top\} = \bigcup_{b \in B} (\mathcal{R}_{\text{forward}}(b) \cup \mathcal{R}_{\text{backward}}(b)) \cup \{\exists t[x_{b_{\text{root}}} = \top].\top\}$$

The singleton triggerless rule $\exists t[x_{b_{\text{root}}} = \top].\top$ added to $\mathcal{S}$ will be discussed below as it constitutes a technical requirement for proper process initiation.

Let us define the state variables first. For each SESE block b in the decomposition, we introduce state variables that capture the activation status and control flow decisions. The specific variables as well as their domain depend on the block type. For each block b in the SESE tree, we introduce a state variable

$$x_b \text{ with domain } V_{x_b} = \{\top, \bot\}$$

where $\top$ indicates that the block is active (executing) and $\bot$ indicates that the block is inactive. This applies uniformly to all block types: TASK, FLOW, PARALLEL, LOOP, and XOR blocks. The following transition functions for block state avoids consecutive disabled intervals that make no sense in process execution:

$$T_{x_b}(\top) = V_{x_b} = \{\top, \bot\}$$
$$T_{x_b}(\bot) = \{\top\}$$

For XOR blocks, which implement exclusive choice semantics, we require an additional state variable to track which branch has been selected. Therefore, for each XOR block b, we introduce a second state variable

$$x_{b_\text{dec}} \text{ with domain } V_{x_{b_\text{dec}}} = \{\bot, \top_{\text{high}}, \top_{\text{low}}\}$$

This decision variable determines which branch is selected, where $\top_{\text{high}}$ selects the high-priority branch, $\top_{\text{low}}$ selects the low-priority branch, and $\bot$ indicates no decision has been made. The transition functions for all state variables mentioned above allow transitions between any values in their respective domains, that is, $T_{x_b}(v) = V_{x_b}$ for all values v in the domain.

Finally for FLOW blocks, which implement sequential execution semantics, we require an additional control flow timeline to ensure proper sequential execution and mutual exclusion between phases. Therefore, for each FLOW block b, we introduce a second state variable

$$x_{b_\text{flow}} \text{ with domain } V_{x_{b_\text{flow}}} = \{\bot, \top_{\text{before}}, \top_{\text{after}}\}$$

This control variable enforces the sequential progression from the before phase to the after phase, with automatic mutual exclusion between the two phases. For such a reason this is the only variable that has a constrained transition function which enforces sequential execution:

$$\begin{aligned} T_{x_{b_\text{flow}}}(\bot) &= \{\top_{\text{before}}\} \\ T_{x_{b_\text{flow}}}(\top_{\text{before}}) &= \{\top_{\text{after}}\} \\ T_{x_{b_\text{flow}}}(\top_{\text{after}}) &= \{\bot, \top_{\text{before}}\} \end{aligned}$$

This transition structure ensures that the flow begins in the disabled state, must transition to the before phase first, can transition from before to after, and prohibits direct transitions from disabled to after.
Given a SESE tree with blocks $B = \{b_1, b_2, \ldots, b_n\}$, the complete set of state variables is:

$$\mathcal{SV} = \{x_b \mid b \in B\} \cup \begin{array}{c} \{x_{b_\text{dec}} \mid b \in B \text{ and } b \text{ is an XOR block}\} \\ \cup \\ \{x_{b_\text{flow}} \mid b \in B \text{ and } b \text{ is a FLOW block}\} \end{array}$$

We must address a special case in our encoding: the root region of the SESE decomposition. Unlike other blocks that can transition between enabled and disabled states, the root region has a unique semantics. The root region state variable x_b (where b denotes the root block) has only the $\top$ (enabled) value in its domain $V_{x_b} = \{\top\}$, and its transition function is defined as $T_{x_b}(\top) = \varnothing$, meaning that once the root region becomes active, it remains active throughout the entire process execution. This represents a semantic constraint ensuring that each plan corresponds to exactly one complete process computation.

As a goal specification, we represent the requirement for process initiation as a triggerless rule: $\exists t[x_b = \top].\top$. This triggerless rule mandates that the root region must be activated, which will cascadingly trigger all due synchronization rules according to the hierarchical structure of the SESE decomposition. This design choice ensures that every valid plan represents a complete execution of the BPMN process from start to finish, rather than partial or incomplete executions.
We now define the functions $\mathcal{R}_{\text{forward}}$ and $\mathcal{R}_{\text{backward}}$ responsible of generating all the synchronization rules for each block type. It is crucial to note that all synchronization rules involved in our BPMN encoding are eager, which ensures the deterministic automaton construction presented in this paper. The eager nature of these rules typically manifests in three common patterns: rules where the trigger token equals another token (trigger = another token), rules where the trigger token is started by or starts another token (corresponding to specific rows in Table 1 of Section 9), and rules where the trigger token is ended by a non-trigger token, which amounts to non-trigger token ending the trigger token. When rules of these kinds appear for the first time in the encoding below, we reference the corresponding row in Table 1 of Section 9. The fact that all such rows have "Overall ambiguous" value equal to "no" establishes the eagerness of the encoding. A notable aspect of our encoding is the use of reflexive versions of Allen relations. For instance, when we say that token a starts token b, we include the case where $a = b$, meaning that the loose end constraint is $e(a) \leq e(b)$ rather

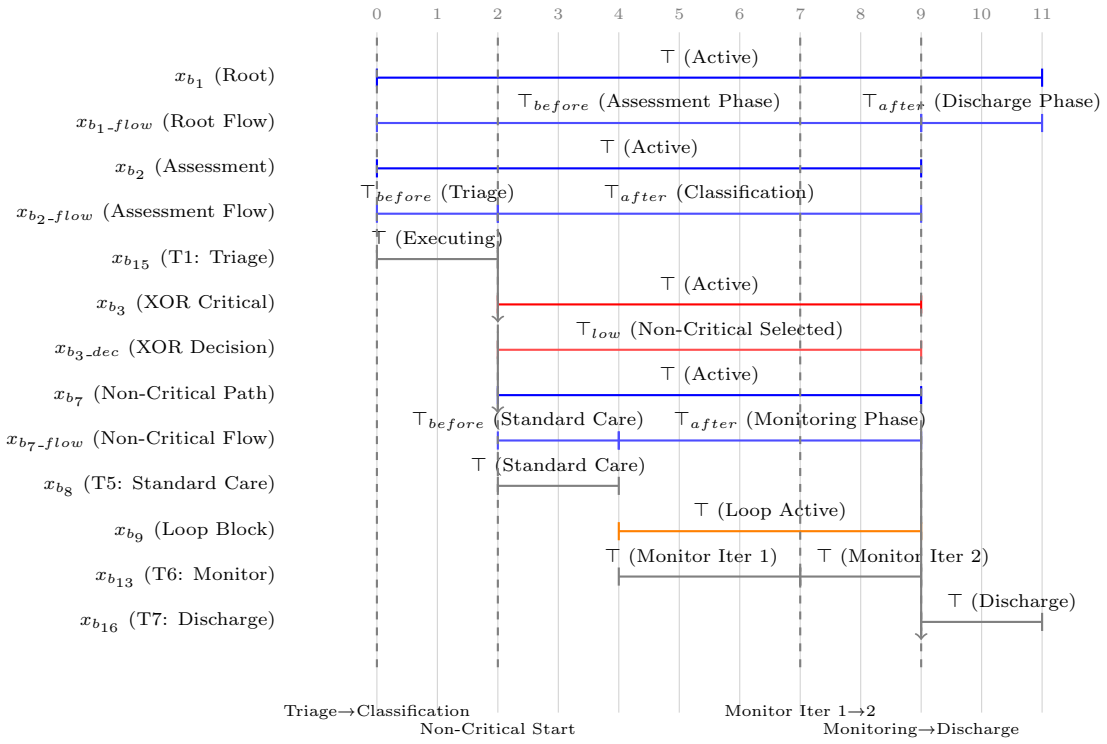


Figure 8: Timeline execution for the non-critical path in the Emergency Department BPMN process with two iterations of the monitoring loop. The diagram shows the execution flow when the XOR decision selects the low branch (non-critical patient). The monitoring loop (b_9) executes twice with consecutive monitoring task iterations: first iteration from time 4-7 (length 3), second iteration from time 7-9 (length 2). The loop block covers exactly the concatenation of the two monitoring executions with no gaps, with the patient becoming stable after the second monitoring cycle, allowing progression to discharge planning.

than the strict inequality $e(a) < e(b)$ required by the traditional Allen relation. As we will point out in Section 9, reflexivity for relations such as starts, ends, and during (where both endpoints have non-strict equality) does not affect the eagerness status: if a rule is eager with strict relations, it remains eager with reflexive relations, and if it is not eager with strict relations, it remains non-eager with reflexive relations.

To illustrate the behavior of the synchronization rules across different block types, we present two concrete timeline executions that demonstrate alternative paths through the Emergency Department process depicted in Figure 6. The first execution, depicted in Figure 8, represents a plan satisfying the non-critical path, where patients receive standard care followed by iterative monitoring until their condition stabilizes. The second execution, shown in Figure 9, represents a plan satisfying the critical path of the BPMN diagram,

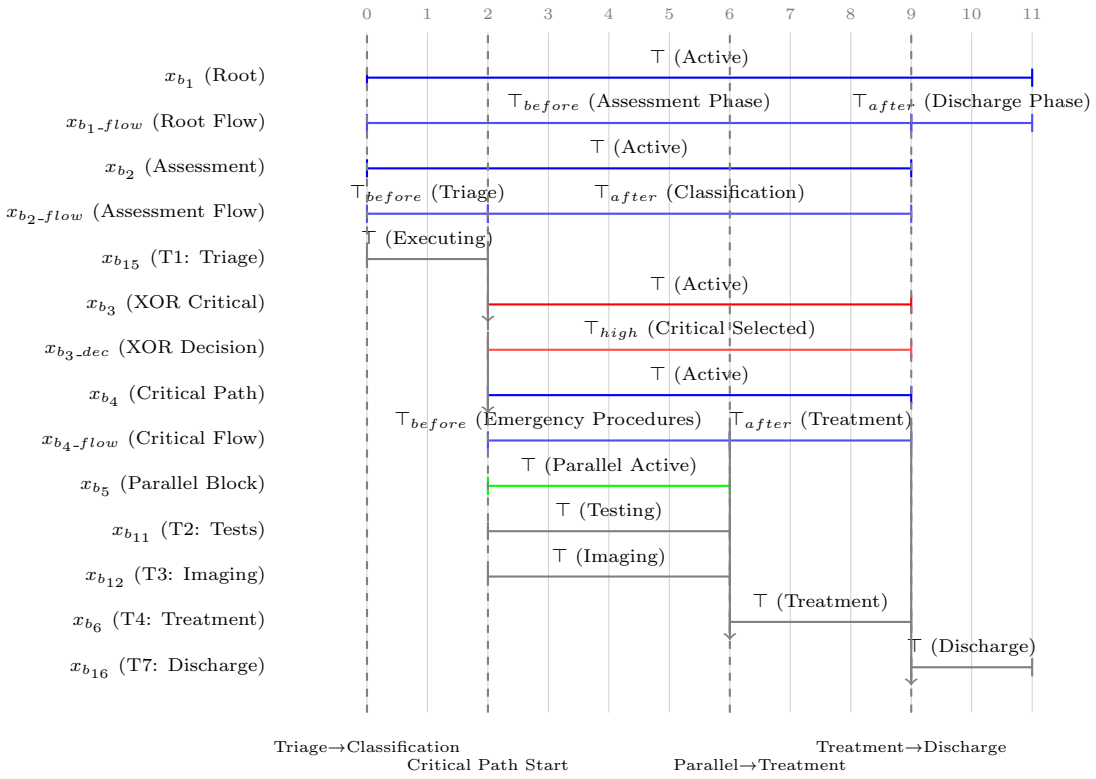


Figure 9: Timeline execution for the critical path in the Emergency Department BPMN process, including all state variables and flow control timelines. The diagram shows both the main block activations and their corresponding flow control variables that enforce sequential execution semantics. Flow variables with subscript "flow" implement the before/after phases for FLOW blocks, while the decision variable for the XOR block tracks branch selection. Tasks T2 (Emergency Tests) and T3 (Emergency Imaging) execute in parallel between time points 2 and 6, synchronized by the parallel block constraint, both completing simultaneously before proceeding to emergency treatment (T4).

where patients classified as critical undergo emergency procedures including concurrent testing and imaging followed by emergency treatment. These timeline figures will be referenced throughout the following block definitions to demonstrate how each type of synchronization rule enforces the intended BPMN semantics.

For TASK blocks b:

$$\mathcal{R}_{\text{forward}}(b) = \mathcal{R}_{backward}(b) = \varnothing$$

Task blocks generate no synchronization rules as they represent atomic activities that do not decompose into sub-blocks. The positive duration constraint is automatically enforced by the qualitative timeline-based planning framework.

For FLOW blocks b with children b_{before} and b_{after}, we define:

$$\mathcal{R}_{\text{forward}}(b) = \left\{ \begin{array}{ll} \text{(Ff.1)} & a_0[x_b = \top] \rightarrow \exists a_1[x_{b_\text{flow}} = \top_{\text{before}}]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_1) \leq \text{end}(a_0)), \\ \text{(Ff.2)} & a_0[x_b = \top] \rightarrow \exists a_1[x_{b_\text{flow}} = \top_{\text{after}}]. \\ & \qquad (\text{start}(a_1) \leq \text{start}(a_0) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Ff.3)} & a_0[x_{b_\text{flow}} = \top_{\text{before}}] \rightarrow \exists a_1[x_b = \top]. \\ & \qquad (\text{start}(a_1) = \text{start}(a_0) \wedge \text{end}(a_0) \leq \text{end}(a_1)), \\ \text{(Ff.4)} & a_0[x_b = \bot] \rightarrow \exists a_1[x_{b_\text{flow}} = \bot]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Ff.5)} & a_0[x_{b_\text{flow}} = \bot] \rightarrow \exists a_1[x_b = \bot]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Ff.6)} & a_0[x_{b_\text{flow}} = \top_{\text{before}}] \rightarrow \exists a_1[x_{b_{\text{before}}} = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Ff.7)} & a_0[x_{b_\text{flow}} = \top_{\text{after}}] \rightarrow \exists a_1[x_{b_{\text{after}}} = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)) \end{array} \right\}$$

$$\mathcal{R}_{\text{backward}}(b) = \left\{ \begin{array}{ll} \text{(Fb.1)} & a_0[x_{b_{\text{before}}} = \top] \rightarrow \exists a_1[x_{b_\text{flow}} = \top_{\text{before}}]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Fb.2)} & a_0[x_{b_{\text{after}}} = \top] \rightarrow \exists a_1[x_{b_\text{flow}} = \top_{\text{after}}]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)) \end{array} \right\}$$

Rules Ff.1-3, when paired with the transition function of x_{b_flow}, effectively impose that block b represents the sequential concatenation of b_{before} and b_{after}. This constraint emerges through several key mechanisms. The if-and-only-if correspondence forced by rule Ff.6 paired with Fb.1, and rule Ff.7 paired with Fb.2, establishes a bidirectional relationship between the flow control phases and the actual child block executions. Additionally, rule Ff.3 imposes that any b_flow interval with value $\top_{\text{before}}$ must be a prefix of some b interval, which, combined with the transition function constraint that $\top_{\text{before}}$ can only transition to $\top_{\text{after}}$, ensures that we can have exactly one $\top_{\text{before}}$ phase followed by just $\top_{\text{before}}$ phase for $x_{b_{flow}}$ inside any $\top$ interval of b. The remaining rules (Ff.4-5) provide the necessary constraints for proper disabled state management, ensuring consistency between the parent block and its control flow variable when neither is active.

This sequential execution behavior enforced by FLOW blocks can be observed in the timeline executions depicted in Figures 9 and 8. In both execution scenarios, the flow control variables (such as $x_{b_1_\text{flow}}$, $x_{b_2_\text{flow}}$, $x_{b_4_\text{flow}}$, and $x_{b_7_\text{flow}}$) demonstrate the strict sequential progression from $\top_{\text{before}}$ to $\top_{\text{after}}$ phases, with no overlap between consecutive phases and automatic mutual exclusion ensuring that only one phase can be active at any given time within each flow block execution interval.

The Allen relations appearing in the FLOW block encoding establish its eagerness according to Table 1. Rule Ff.1 implements the reflexive relation a_1 starts a_0 with a_0 as trigger (row 11), rule Ff.2 implements the reflexive relation a_1 ends a_0 with a_0 as trigger (row 8), and rule Ff.3 implements the reflexive relation a_0 starts a_1 with a_0 as trigger (row 10). All remaining rules (Ff.4, Ff.5, Ff.6, Ff.7, Fb.1, Fb.2) implement equality relations a_0 equals a_1 with the respective trigger token (row 19). According to Table 1, all these configurations yield "no" in the "Overall ambiguous" column, confirming that the entire

FLOW block encoding maintains eagerness properties essential for deterministic automaton construction. As noted earlier, the reflexive nature of these relations does not affect their eagerness status.

For PARALLEL blocks b with children b^1 and b^2, we define:

$$\mathcal{R}_{\text{forward}}(b) = \left\{ \begin{array}{ll} \text{(Pf.1)} & a_0[x_b = \top] \rightarrow \exists a_1[x_{b^1} = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Pf.2)} & a_0[x_b = \top] \rightarrow \exists a_1[x_{b^2} = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)) \end{array} \right\}$$

$$\mathcal{R}_{\text{backward}}(b) = \left\{ \begin{array}{ll} \text{(Pb.1)} & a_0[x_{b^1} = \top] \rightarrow \exists a_1[x_b = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Pb.2)} & a_0[x_{b^2} = \top] \rightarrow \exists a_1[x_b = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)) \end{array} \right\}$$

These rules establish the bidirectional equivalence where block b is enabled if and only if both children b^1 and b^2 are enabled on exactly the same interval. Note that explicit disabled state constraints are unnecessary since the enabled and disabled states are mutually exclusive by definition of block state variables.

This parallel execution behavior can be observed in Figure 9, where the parallel block x_{b_5} coordinates the simultaneous execution of emergency tests ($x_{b_{11}}$) and emergency imaging ($x_{b_{12}}$) between time points 2 and 6. Both child tasks execute concurrently with identical start and end times, synchronized by the parallel block constraint that enforces their exact temporal alignment. The PARALLEL block encoding introduces no new types of Allen relations beyond those discussed for FLOW blocks, as all rules implement equality relations between tokens.

For LOOP blocks b with body child b_{body}, we define:

$$\mathcal{R}_{\text{forward}}(b) = \left\{ \begin{array}{ll} \text{(Lf.1)} & a_0[x_b = \top] \rightarrow \exists a_1[x_{b_{\text{body}}} = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_1) \leq \text{end}(a_0)), \\ \text{(Lf.2)} & a_0[x_b = \top] \rightarrow \exists a_1[x_{b_{\text{body}}} = \top]. \\ & \qquad (\text{start}(a_1) \leq \text{start}(a_0) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Lf.3)} & a_0[x_b = \bot] \rightarrow \exists a_1[x_{b_{\text{body}}} = \bot]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)) \end{array} \right\}$$

$$\mathcal{R}_{\text{backward}}(b) = \left\{ \begin{array}{ll} \text{(Lb.1)} & a_0[x_{b_{\text{body}}} = \bot] \rightarrow \exists a_1[x_b = \bot]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)) \end{array} \right\}$$

The bidirectional correspondence between disabled states (Lf.3 and Lb.1) ensures that the loop and its body are disabled on exactly the same intervals, meaning that when one is enabled the other must be too without gaps due to mutual exclusion of enabled and disabled states on block state variables. With only the bidirectional disabled correspondence, we would allow for chains of body executions inside loop intervals and chains of loop intervals inside body executions. To avoid this erroneous latter case, we explicitly require that any

enabled loop admits both a prefix body execution (Lf.1) and a suffix body execution (Lf.2), thereby establishing the proper containment ordering where body executions are nested inside loop intervals rather than vice versa.

This iterative execution behavior can be observed in Figure 8, where the loop block x_{b_9} enables multiple consecutive iterations of the monitoring task ($x_{b_{13}}$). The figure demonstrates two monitoring iterations: the first from time 4-7 (length 3) and the second from time 7-9 (length 2), with the loop block spanning exactly the concatenation of both iterations from time 4-9. The LOOP block encoding introduces no new types of Allen relations beyond those discussed for FLOW blocks, utilizing the same starts, ends, and equality relations to establish proper containment and sequencing constraints.

For XOR blocks b with children b^{high} and b^{low}, we define:

$$\mathcal{R}_{\text{forward}}(b) = \left\{ \begin{array}{ll} \text{(Xf.1)} & a_0[x_b = \bot] \rightarrow \exists a_1[x_{b_\text{decision}} = \bot]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Xf.2)} & a_0[x_{b_\text{decision}} = \bot] \rightarrow \exists a_1[x_b = \bot]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Xf.3)} & a_0[x_{b_\text{decision}} = \top_{\text{high}}] \rightarrow \exists a_1[x_b = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Xf.4)} & a_0[x_{b_\text{decision}} = \top_{\text{low}}] \rightarrow \exists a_1[x_b = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Xf.5)} & a_0[x_{b_\text{decision}} = \top_{\text{high}}] \rightarrow \exists a_1[x_{b^{\text{high}}} = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Xf.6)} & a_0[x_{b_\text{decision}} = \top_{\text{low}}] \rightarrow \exists a_1[x_{b^{\text{low}}} = \top]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)) \end{array} \right\}$$

$$\mathcal{R}_{\text{backward}}(b) = \left\{ \begin{array}{ll} \text{(Xb.1)} & a_0[x_{b^{\text{high}}} = \top] \rightarrow \exists a_1[x_{b_\text{decision}} = \top_{\text{high}}]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)), \\ \text{(Xb.2)} & a_0[x_{b^{\text{low}}} = \top] \rightarrow \exists a_1[x_{b_\text{decision}} = \top_{\text{low}}]. \\ & \qquad (\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1)) \end{array} \right\}$$

The forward rules establish the decision-driven behavior where the decision variable and main block are synchronized in their disabled states (Xf.1-2), and where decision values trigger both the main block (Xf.3-4) and the corresponding child branches (Xf.5-6). The backward rules ensure that child block activations trigger their respective decision values (Xb.1-2). The values on the decision variable are mutually exclusive by definition since they reside on the same state variable, which automatically ensures that child blocks are disabled when the decision is disabled.

This exclusive choice behavior can be observed in both Figures 9 and 8, where the XOR block x_{b_3} implements the critical classification decision. In the critical path scenario, the decision variable $x_{b_3_\text{dec}}$ takes value $\top_{\text{high}}$ to select the critical path (x_{b_4}) with emergency procedures, while in the non-critical scenario it takes value $\top_{\text{low}}$ to select the non-critical path (x_{b_7}) with standard care and monitoring. The XOR block encoding introduces no new types of Allen relations beyond those discussed for FLOW blocks, as all rules implement

equality relations that ensure perfect synchronization between the decision variable, main block, and selected child branch.

This concludes our encoding of BPMN constructs into the eager fragment. Returning to the comparison with [COS19], we note that while limited to the interval relations discussed in Section 9, we can also express constraints in the style of [COS19] through eager rules by exploiting the compositionality of timelines. This means that plans may always be enriched through additional constraints implemented via synchronization rules that involve the newly introduced timelines and the existing ones, as we will show in the following. As an oversimplified example, consider adding a timeline for patient condition that transitions between unstable and stable states, where once stability is reached it persists: $T_{\text{condition}}(\text{unstable}) = \{\text{stable}, \text{unstable}\}$ and $T_{\text{condition}}(\text{stable}) = \{\text{stable}\}$. We can then specify that the critical path decision always starts with an unstable patient condition through the rule:

$$a_0[x_{b_4} = \top] \to \exists a_1[x_{\text{condition}} = \text{unstable}].(\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_1) \leq \text{end}(a_0))$$

This rule is activated only in case of critical path execution. In contrast, we place no constraints on the non-critical path (patients may be unstable but only mildly so). In such situations, we contemplate the case that the monitoring loop, activated only in case of non-critical execution, may cycle through stable conditions to verify that the patient remains stable, and finally we can require that discharge always begins with a stable patient condition through the rule:

$$a_0[x_{b_{16}} = \top] \to \exists a_1[x_{\text{condition}} = \text{stable}].(\text{start}(a_0) = \text{start}(a_1) \wedge \text{end}(a_0) = \text{end}(a_1))$$

This second rule is eventually activated for all patients, regardless of their treatment path. Such domain-specific constraints demonstrate how the eager fragment can still express some interesting temporal constraints beyond pure control flow through compositional timeline integration.

8.1. **Scope, extensibility, and limitations of the encoding.** The BPMN encoding presented above is intended as an illustrative case study demonstrating the expressiveness of the eager fragment on the major control flow patterns. The BPMN constructs not explicitly covered can, in principle, be encoded within our formalism; however, several limitations and drawbacks should be taken into account, as they may significantly affect the size of the resulting encoding and, consequently, the efficiency of the subsequent decidability goal, whether plan existence, strategy synthesis, or others.

SESE decomposition. The efficiency and succinctness of our encoding relies heavily on the SESE property of the input BPMN diagram. Since disjunctions are not allowed in eager rules, the tree structure of the SESE decomposition is essential: it guarantees that, upon completion of the current block, the next block to be executed can be identified with certainty, whether it is the next sibling in a flow or the next sibling of the parent block, without requiring any nondeterministic guessing. Well-structuredness is widely regarded as a desirable design principle in the BPMN community, as it promotes readability, reduces modeling errors, and facilitates formal analysis [PGBD12]. Moreover, in most practical cases, unstructured diagrams can be transformed into equivalent SESE ones by duplicating components: this has been shown to be always possible for acyclic process models [PGBD12], and partial results exist for cyclic models as well [KH04, CKJZ15, PCH22], although some pathological loop constructions are known to resist structuring. All in all, the SESE requirement is

not restrictive in practice; however, when restructuring of a non-SESE diagram is needed, the transformation may introduce additional components, resulting in a larger parse tree and, consequently, a higher number of state variables. Since $|\mathcal{SV}|$ is the parameter that governs the complexity of the resulting planning problem, this duplication directly impacts the efficiency.

Swimlanes and message passing. Swimlanes and pools are naturally accommodated by our encoding: they simply partition the state variables into organizational groups, resulting in a collection of independent SESE trees that do not affect the encoding in any way. Similarly, message passing constructs — such as sending and receiving messages, signals, or events — can be handled by introducing a dedicated timeline for each communication channel. Such a timeline behaves like a task and is shared between the two SESE trees of the communicating pools. The flow constraints on each side determine when the respective pool is ready to send or receive, and synchronization between the two pools happens in a natural and elegant way, practically for free.

Clock events and duration constraints. Although our framework is qualitative, clock events and duration constraints can be encoded within the eager fragment, at the cost of a more involved construction. We briefly sketch the idea for clock events; duration constraints for tasks can be handled similarly. First, a *global clock* timeline x_{clk} is introduced, with a single value *tick* (i.e., $V_{x_{clk}} = \{tick\}$). Every token on every other timeline is then required to be *started by* a clock tick, corresponding to row 11 of Table 1 in Section 9, so that any value change on any timeline is aligned with a fresh tick of the global clock. For each clock event of duration d, two dedicated timelines are introduced. The first, the *counter* timeline, has domain $\{\bot, 0, 1, \ldots, d\}$ with value transition function $T(\bot) = \{0\}$, $T(i) = \{i{+}1\}$ for $0 \leq i < d$, and $T(d) = \{\bot\}$, so that the counter cycles through exactly d ticks before resetting. The second, the *envelope* timeline, alternates between two values: one that spans the entire $\bot$ phase and one that spans the full counting phase from 0 to d, providing a coarser view of the timer's active and inactive periods. It should be noted that this construction encodes durations in *unary*: a clock event of duration d requires $d{+}2$ values in the domain of the counter timeline, so the size of the encoding grows linearly with the magnitude of the max durations expressed in the process. A more compact binary encoding of clocks would not be feasible within the eager fragment, as it would necessarily require *disjunctions* to express the bit-manipulation logic of binary counting, and *equality constraints between non-trigger existential tokens* to synchronize the individual bit timelines, the latter corresponding to row 21 of Table 1, which is ambiguous. Both features are incompatible with eager rules. Another key observation is that, due to the qualitative nature of the synchronization rules, individual ticks may have different actual durations; however, since all tokens across all timelines are forced to respect the same global clock, a successful plan exists in the qualitative encoding if and only if a successful plan exists in the intended real-time interpretation.

Gateways. The most notable BPMN gateway not covered by our encoding is the *inclusive OR gateway*, which allows one or more branches to be activated depending on runtime conditions. A binary inclusive OR with branches A and B can in principle be encoded as a three-way XOR over the activation patterns $\{A\}$, $\{B\}$, and $\{A, B\}$, thus avoiding disjunctions. However, this encoding requires duplicating the entire downstream process subtree for each activation pattern, with all child timelines in the duplicated regions renamed to fresh state variables. Since n-ary inclusive OR gateways can always be decomposed

into a tree of binary ones, the critical cost factor is the nesting depth: k nested binary inclusive OR gateways result in 3^k copies of the innermost subtree, yielding an encoding that is exponential in the nesting depth. The *event-based gateway* (deferred choice), where the routing decision depends on which among several competing events occurs first, can instead be treated as an XOR gateway: the decision is resolved by the plan itself, which selects the winning event. In the presence of clock events, the encoding becomes more involved due to the need for the timer machinery described above, but this additional complexity does not affect the size of the plan in the way that inclusive OR gateways do.

In summary, the eager fragment can accommodate a broader range of BPMN constructs than those explicitly covered in our case study. Swimlanes, message passing, and event-based gateways are handled naturally and essentially for free, without increasing the complexity of the encoding. Clock events and duration constraints can be encoded at the cost of a unary representation of durations and a more involved rule structure, but without departing from the eager fragment. The main limitations arise from inclusive OR gateways, whose encoding requires an exponential blowup in the nesting depth, and from unstructured (non-SESE) diagrams, whose restructuring, when possible, may introduce additional components. In both cases, the resulting increase in the number of state variables directly impacts the complexity of the subsequent decidability tasks, as $|\mathcal{SV}|$ is the governing parameter of our construction.

9. A maximal subset of Allen's relations

Allen's interval algebra is a formalism for temporal reasoning introduced in [All83]. It identifies all possible relations between pairs of time intervals over a linear order and specifies a machinery to reason about them. In this section, we isolate the maximal subset of Allen's relations captured by the eager fragment of qualitative timeline-based planning. To this end, we show how to map Allen's relations over tokens in terms of their endpoints, that is, as conjunctions of atoms over terms $\mathtt{s}(a), \mathtt{s}(b), \mathtt{e}(a), \mathtt{e}(b)$, for token names a and b. Then, we identify the relation encodings that can be expressed by the eager fragment, according to Definition 3.1. Let $a, b \in \mathcal{N}$ and $t_1 = t_2$ be an abbreviation for $t_1 \leq t_2 \wedge t_2 \leq t_1$, for all pairs of terms t_1, t_2.

- a before b (b after a) can be defined as $\mathtt{e}(a) < \mathtt{s}(b)$.
- a meets b (b is-met-by a) can be defined as $\mathtt{e}(a) = \mathtt{s}(b)$.
- a ends b (b is-ended-by a) can be defined as $\mathtt{s}(b) < \mathtt{s}(a) \wedge \mathtt{e}(a) = \mathtt{e}(b)$.
- a starts b (b is-started-by a) can be defined as $\mathtt{s}(a) = \mathtt{s}(b) \wedge \mathtt{e}(a) < \mathtt{e}(b)$.
- a overlaps b (b is-overlapped-by a) can be defined as $\mathtt{s}(a) < \mathtt{s}(b) \wedge \mathtt{s}(b) < \mathtt{e}(a) \wedge \mathtt{e}(a) < \mathtt{e}(b)$.
- a during b (b contains a) can be defined as $\mathtt{s}(b) < \mathtt{s}(a) \wedge \mathtt{e}(a) < \mathtt{e}(b)$.
- $a = b$ can be defined as $\mathtt{s}(a) = \mathtt{s}(b) \wedge \mathtt{e}(a) = \mathtt{e}(b)$.

It is not difficult to see that, if one of the tokens, say a, is the trigger token, then the encodings not complying with the definition of eager rule (Definition 3.1) are the ones for Allen's relations a ends b, a overlaps b, a is-overlapped-by b, and a during b (see also Table 1). Thus, the maximal subset of Allen's relations that can be captured by an instance of the eager fragment of the timeline-based planning problem consists of relations a before b, a after b, a meets b, a is-met-by b, a is-ended-by b, a starts b, a is-started-by b, a contains b, and $a = b$. As an example, consider the constraint a overlaps b and let $\mathcal{C} = \{\mathtt{s}(a) < \mathtt{s}(b), \mathtt{s}(b) <$

$\mathsf{e}(a), \mathsf{e}(a) < \mathsf{e}(b)\}$ be its encoding. Clearly, the transitive closure $\hat{\mathcal{C}}$ of $\mathcal{C}$ (cf. Section 3) includes also $\mathsf{s}(b) \leq \mathsf{e}(a)$, but it does not include any of $\mathsf{s}(b) \leq \mathsf{s}(a)$, $\mathsf{e}(a) \leq \mathsf{s}(b)$, and $\mathsf{e}(b) \leq \mathsf{e}(a)$, implying that token name b is left-ambiguous (Definition 3.1*(A)*). Moreover, we have that $\hat{\mathcal{C}}$ includes $\mathsf{e}(a) \leq \mathsf{e}(b)$ but, as already pointed out, it does not include $\mathsf{e}(a) \leq \mathsf{s}(b)$, which means that b is right-ambiguous as well (Definition 3.1*(B)*). Therefore, b is ambiguous (since it is not the trigger token – see Definition 3.1*(C)*), and thus a overlaps b is not captured by an eager rule. A similar argument can be used for a ends b, a is-overlapped-by b, and a during b. If, instead, none of the tokens is a trigger token, then the only Allen's relations that can be captured by eager rules are are before, after, meets, and is-met-by.

Table 1 provide a complete picture of the status of the Allen's interval relations with respect to the property of being expressible by means of eager rules. As a matter of fact, the table only includes Allen's relations before, meets, ends, starts, overlaps, during, and $=$. The status of the remaining relations (after, is-met-by, is-ended-by, is-started-by, is-overlapped-by, contains) can be easily derived, as each is the inverse of one of the listed relations.

Every Allen relation can be expressed as a conjunction of atoms, without using disjunction. Therefore, being expressible by means of an eager rule amounts to being expressible by means of an unambiguous one (*cf.* Definition 3.1). Moreover, according to Definition 3.1, the property of being unambiguous depends on the role (being or not a trigger token) of the token names involved in the rule. Therefore, every relation, which establishes constraints between two token names a and b, is considered with respect to three different scenarios, according to which token name is the trigger one (column "Trigger token"): *(i)* a is the trigger token, *(ii)* b is the trigger token, *(iii)* neither a nor b is the trigger token. For example, the first of the three lines for Allen relation ends (value "a" in the column "Trigger token") depicts the status of the corresponding synchronization rule, which constrains token name a to be a strict suffix of token name b, when a is the trigger token:

$$a[x_a = v_a] \rightarrow \exists b[x_b = v_b].\ \mathsf{s}(b) < \mathsf{s}(a) \wedge \mathsf{e}(a) = \mathsf{e}(b).$$

In this case, being a the trigger token, it is trivially not ambiguous. This is expressed in the table by the character "$-$". The table also shows, in the columns relevant for token b, that b is left-ambiguous, right-ambiguous, and ambiguous. Since at least one among a and b is ambiguous, the entire rule is ambiguous, as indicated by the string "yes" in the last column (meaning that the rule does not belong to the eager fragment of qualitative timeline-based planning). The next line of the table (value "b" in the column "Trigger token") considers the rule obtained from the above one, using b as trigger token:

$$b[x_b = v_b] \rightarrow \exists a[x_a = v_a].\ \mathsf{s}(b) < \mathsf{s}(a) \wedge \mathsf{e}(a) = \mathsf{e}(b).$$

In this case, b is trivially not ambiguous. Token name a is not ambiguous as well, since it is not left-ambiguous (even though it is right-ambiguous). Since neither a nor b is ambiguous, the entire rule is unambiguous, as indicated by the string "no" in the last column (and thus it belongs to the eager fragment of qualitative timeline-based planning). Furthermore, the next line of the table (value "none" in the column "Trigger token") considers the triggerless rule for the same Allen relation:

$$\top \rightarrow \exists a[x_a = v_a] b[x_b = v_b].\ \mathsf{s}(b) < \mathsf{s}(a) \wedge \mathsf{e}(a) = \mathsf{e}(b).$$

As shown in the table, a is not ambiguous (as it is not left-ambiguous), while b is ambiguous (because it is not the trigger token, and it is both left- and right-ambiguous); therefore, the entire rule is ambiguous (and thus it does not belong to the eager fragment of qualitative timeline-based planning). As a last example, consider the following rule for Allen relation

is-ended-by, constraining token name b to be a strict suffix of token name a, when a is the trigger token:

$$a[x_a = v_a] \rightarrow \exists b[x_b = v_b].\ \mathtt{s}(a) < \mathtt{s}(b) \wedge \mathtt{e}(a) = \mathtt{e}(b).$$

Saying a is-ended-by b, with the first token name being the trigger one, amounts to saying b ends a, with the second token name being the trigger one; therefore, the rule is not ambiguous, since it corresponds to the second of the three lines in the table for Allen relation ends, the one where the second token name is the trigger token. More precisely, a is not ambiguous since it is the trigger token, and b is not ambiguous since it is not left-ambiguous (it is right-ambiguous, though).

Finally, we observe that *reflexive* variants of the Allen's relations can also be considered. These variants are obtained by replacing $<$ with $\leq$ in each of the mappings (of Allen's relations in terms of conjunctions of atoms) given above. The status of the reflexive variants does not change, for any Allen relation.

Table 1: Eagerness analysis for Allen's relations

#	Allen Relation	Trigger token	Token a			Token b			Overall ambiguous
			Left-amb	Right-amb	Ambiguous	Left-amb	Right-amb	Ambiguous	
1	a before b	a	-	-	-	no	no	no	no
2	a before b	b	no	no	no	-	-	-	no
3	a before b	none	no	no	no	no	no	no	no
4	a meets b	a	-	-	-	no	no	no	no
5	a meets b	b	no	yes	no	-	-	-	no
6	a meets b	none	no	yes	no	yes	no	no	no
7	a ends b	a	-	-	-	yes	yes	yes	yes
8	a ends b	b	no	yes	no	-	-	-	no
9	a ends b	none	no	yes	no	yes	yes	yes	yes
10	a starts b	a	-	-	-	no	yes	no	no
11	a starts b	b	no	no	no	-	-	-	no
12	a starts b	none	yes	no	no	yes	yes	yes	yes
13	a overlaps b	a	-	-	-	yes	yes	yes	yes
14	a overlaps b	b	yes	yes	yes	-	-	-	yes
15	a overlaps b	none	yes	yes	yes	yes	yes	yes	yes
16	a during b	a	-	-	-	yes	yes	yes	yes
17	a during b	b	no	no	no	-	-	-	no
18	a during b	none	no	no	no	yes	yes	yes	yes
19	$a = b$	a	-	-	-	no	yes	no	no
20	$a = b$	b	no	yes	no	-	-	-	no
21	$a = b$	none	yes	yes	yes	yes	yes	yes	yes

10. Conclusions

In this paper, we identified a meaningful fragment of qualitative timeline-based planning (the *eager* fragment) whose solutions can be recognized by DFAs of singly exponential size. Specifically, we identified restrictions on the allowed synchronization rules, which we called *eager* rules, for which we showed how to build the corresponding deterministic automaton of exponential size, that can then be directly exploited to synthesize strategies. We also showed that it is not possible to encode the solution plans for qualitative timeline-based planning problems (when non-eager rules are allowed) using deterministic finite automata of exponential size. Moreover, we demonstrated the practical relevance of the eager fragment

through a comprehensive BPMN case study that systematically translates all major control flow patterns into eager synchronization rules. Last but not least, we isolated a maximal subset of Allen's relations captured by the eager fragment.

Whether the eager fragment of qualitative timeline-based planning is maximal or not is an open question currently under investigation. Further research directions include a parameterized complexity analysis over the number of synchronization rules and a characterization in terms of temporal logics, like the one in [DMGM$^+$17]. Another interesting direction for future work concerns the extension of our approach to dense temporal domains. Timeline-based planning over dense time has been studied by Bozzelli et al. [BMMP18b], who showed that the problem is undecidable in full generality, even with a single state variable. Our work deliberately focuses on the qualitative fragment, which abstracts away metric temporal information and retains only the relative order of events. The DFA construction at the core of our approach fundamentally relies on this discrete, qualitative setting, where each letter of the input word encodes a single time step and the automaton tracks rule satisfaction incrementally. Lifting the eager fragment to dense time would require fundamentally different techniques, and we regard this as a challenging and rewarding avenue for future investigation.

Acknowledgments

Partially supported by the GNCS 2024 project "Certificazione, monitoraggio, ed interpretabilità in sistemi di intelligenza artificiale".

Appendix A. Proof of Lemma 7.1

In this section, we proof Lemma 7.1 from Section 7. We restate the lemma for the reader's convenience.

Lemma 7.1. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem. Each finite word over Σ_{SV} that encodes a plan over SV is accepted by $\mathcal{A}_P$ if and only if it encodes a solution plan for P. Moreover, the size of $\mathcal{A}_P$ is at most exponential in the size of P.*

We first proof the exponential upper bound for the automata size, which is easier. Then, we focus on the main claim of the statement, concerning the correctness of the proposed automata construction.

A.1. Exponential upper bound.

Lemma A.1. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem. The size of $\mathcal{A}_P$ is at most exponential in the size of P.*

Proof. Let k be the largest number of token names in a rule of P. Thanks to the linearity condition enjoyed by states of $\mathcal{A}_P$ (cf. Definition 7.1), it is not difficult to convince oneself that the number of different viewpoints for the same rule in a state $q \in Q_P$ to be at most k. Thus, each state in Q_P contains at most $|S| \times k$ different viewpoints.

Therefore, the size of Q_P is at most $|\Upsilon_P|^{(|S|\times k)}$. Clearly, $|S| \times k$ is at most polynomial in the size of P. Since $|\Upsilon_P| \leq \sum_{\mathcal{R}\in S} |\Upsilon_{\mathcal{R}}|$ and, as already pointed out, $|\Upsilon_{\mathcal{R}}|$ is at most exponential in the size of P, we can conclude that the size of Q_P is at most exponential in the size of P. □

A.2. **Correctness of the automata construction.**

Lemma A.2. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem. Each finite word over Σ_{SV} that encodes a plan over SV is accepted by $\mathcal{A}_P$ if and only if it encodes a solution plan for P.*

First, we define a notion of witness that characterizes solution plans for (not necessarily eager) disjunction-free qualitative timeline-based planning problems (cf. Definition 2.3). Then, in Section A.2.2 we introduce a particular canonical witness, which characterizes solution plans for eager qualitative timeline-based planning problems. The thesis follows from the fact that the automata that we defined in Section 6 and Section 7 recognize such canonical witnesses.

A.2.1. *Matching and relaxed witnesses for (not necessarily eager) planning problems.* Let $P = (\mathsf{SV}, S)$ be a (not necessarily eager) qualitative timeline-based planning problem, $\mathcal{R} \in S$ a rule with trigger $a_0[x_0 = v_0]$, and $\lambda = \langle \sigma_0, \dots, \sigma_{|\lambda|-1} \rangle$ a finite word over Σ_{SV}. We denote by $\mathit{triggers}^\lambda(\mathcal{R})$ the set of pairs of natural numbers that correspond to tokens matching the trigger a_0 of $\mathcal{R}$, i.e., tokens for timeline x_0 of value v_0. Formally, for every $m, n \in \mathbb{N}$, with $m < n$, we have that $(m, n) \in \mathit{triggers}^\lambda(\mathcal{R})$ if and only if

- $\mathit{start}(x_0, v_0) \in \mathit{events}(\sigma_m)$,
- $\mathit{end}(x_0, v_0) \in \mathit{events}(\sigma_n)$, and
- $\{\mathit{start}(x_0, v_0), \mathit{end}(x_0, v_0)\} \cap \mathit{events}(\sigma_i) = \varnothing$ for all $i \in \mathbb{N}$ with $m < i < n$.

For the sake of a uniform treatment of triggerless and non-triggerless rules, we set $\mathit{triggers}^\lambda(\mathcal{R}) = \{\top\}$ for every triggerless rule $\mathcal{R}$ and $\lambda \in \Sigma^*_{\mathsf{SV}}$.

Definition A.1 (trigger-level witness)**.** Let $P = (\mathsf{SV}, S)$ be a (not necessarily eager) disjunction-free qualitative timeline-based planning problem, $\mathcal{R} \in S$, and $\lambda \in \Sigma^*_{\mathsf{SV}}$. Moreover, let $\mathcal{C}$ be the only clause of $\mathcal{R}$ and T be the set of terms (of the form $\mathtt{s}(a)$ and $\mathtt{e}(a)$) appearing in $\mathcal{R}$. For every $tr \in \mathit{triggers}^\lambda(\mathcal{R})$, a *(trigger-level) witness* of tr in λ is a function $f : T \to \mathbb{N}$ such that:

(i) f is *compatible* with λ, that is, if $\lambda = \langle \sigma_0, \dots, \sigma_{|\lambda|-1} \rangle$ and $a[x = v]$ occurs in $\mathcal{R}$, then
- $\mathtt{s}(a) \in T$ implies $\mathit{start}(x, v) \in \mathit{events}(\sigma_{f(\mathtt{s}(a))})$ and
- $\mathtt{e}(a) \in T$ implies $\mathit{end}(x, v) \in \mathit{events}(\sigma_{f(\mathtt{e}(a))})$

(ii) f *satisfies* $\mathcal{C}$, that is, replacing in $\mathcal{C}$ every occurrence of every term $t \in T$ with $f(t)$ makes all atoms in $\mathcal{C}$ true, and

(iii) if $tr = (m, n) \in \mathbb{N} \times \mathbb{N}$, then $f(\mathtt{s}(a_0)) = m$ and $f(\mathtt{e}(a_0)) = n$, where $a_0[x_0 = v_0]$ is the trigger of $\mathcal{R}$ (this condition does not apply if $tr = \top \notin \mathbb{N} \times \mathbb{N}$, i.e, when $\mathcal{R}$ is triggerless).

Definition A.2 (matching and relaxed (trigger-level) witness)**.** Let $P = (\mathsf{SV}, S)$ be a (not necessarily eager) disjunction-free qualitative timeline-based planning problem, $\mathcal{R} \in S$, $\lambda = \langle \sigma_0, \dots, \sigma_{|\lambda|-1} \rangle \in \Sigma^*_{\mathsf{SV}}$, $tr \in \mathit{triggers}^\lambda(\mathcal{R})$, and $f : T \to \mathbb{N}$ a witness of tr in λ, and suppose that $a[x = v]$ occurs in $\mathcal{R}$.

Then, token name a is *matching* (in f and λ) if and only if $\{\mathtt{s}(a), \mathtt{e}(a)\} \not\subseteq T$ or $\{\mathit{start}(x, v), \mathit{end}(x, v)\} \cap \mathit{events}(\sigma_i) = \varnothing$ for all $i \in \mathbb{N}$ with $f(\mathtt{s}(a)) < i < f(\mathtt{e}(a))$. Note that if a is the trigger token of $\mathcal{R}$, then it is matching.

If a is not matching, then we define $\mathit{matching\text{-}start}_f^\lambda(a) = \max_i\{i \in \mathbb{N} \mid f(\mathtt{s}(a)) < i < f(\mathtt{e}(a))$ and $\mathit{start}(x, v) \in \mathit{events}(\sigma_i)\}$. Observe that if λ encodes a plan over SV, then $\mathit{matching\text{-}start}_f^\lambda$ is well defined and $f(\mathtt{s}(a)) < \mathit{matching\text{-}start}_f^\lambda(a)$.

Moreover, witness f is a *matching witness* of tr in λ if and only if all token names appearing in T are matching. Instead, it is a *relaxed witness* of tr in λ if and only if all non-matching token names a appearing in T violate Item *A2* of Definition 3.1, that is, for all terms $t \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$ we have that $\mathtt{s}(a) \leq t \in \hat{\mathcal{C}}$ implies $\mathtt{e}(a) \leq t \in \hat{\mathcal{C}}$.

We show in the following how to obtain matching trigger-level witnesses from relaxed ones.

Definition A.3. Let $P = (\mathsf{SV}, S)$ be a (not necessarily eager) disjunction-free qualitative timeline-based planning problem and $f : T \to \mathbb{N}$ a relaxed witness of tr in λ, for some finite word λ over Σ_{SV} that encodes a plan over SV and $tr \in \mathit{triggers}^\lambda(\mathcal{R})$, with $\mathcal{R} \in S$. We define the *matching version* of f, denoted $\hat{f}$, as the function from T to $\mathbb{N}$ such that:

- $\hat{f}(\mathtt{e}(a)) = f(\mathtt{e}(a))$ for all $\mathtt{e}(a) \in T$,
- $\hat{f}(\mathtt{s}(a)) = f(\mathtt{s}(a))$ for all $\mathtt{s}(a) \in T$ where a is a matching token name,
- $\hat{f}(\mathtt{s}(a)) = \mathit{matching\text{-}start}_f^\lambda(a)$ for all $\mathtt{s}(a) \in T$ where a is not a matching token name.

Lemma A.3. *Let $P = (\mathsf{SV}, S)$ be a (not necessarily eager) disjunction-free qualitative timeline-based planning problem and $f : T \to \mathbb{N}$ a relaxed witness of tr in λ, for some finite word λ over Σ_{SV} that encodes a plan over SV and $tr \in \mathit{triggers}^\lambda(\mathcal{R})$, with $\mathcal{R} \in S$. Then, $\hat{f}$ is a matching witness of tr in λ.*

Proof. We only need to show that $\hat{f}$ is a witness of tr in λ. Being a matching one follows from the definition of function $\mathit{matching\text{-}start}_f^\lambda$. Item *(i)* and Item *(iii)* of Definition A.1 are clearly fulfilled.

Thus we focus on Item *(ii)*. Let $\mathcal{C}$ be the only clause of $\mathcal{R}$ and $\alpha \in \mathcal{C}$ be $t_1 \triangleleft t_2$ for some $t_1, t_2 \in T$ and $\triangleleft \in \{\leq, <\}$. We show that $\hat{f}$ *satisfies* α, that is $\hat{f}(t_1) \triangleleft \hat{f}(t_2)$ holds true. By definition of $\hat{f}$, it holds that $f(t) \leq \hat{f}(t)$ for all $t \in T$. If $f(t_1) = \hat{f}(t_1)$, then we have $\hat{f}(t_1) = f(t_1) \triangleleft f(t_2) \leq \hat{f}(t_2)$, which implies $\hat{f}(t_1) \triangleleft \hat{f}(t_2)$. Thus, let us assume $f(t_1) < \hat{f}(t_1)$. Since $\hat{f}(\mathtt{e}(a)) = f(\mathtt{e}(a))$ for all token names a, it must be $t_1 = \mathtt{s}(a)$ for some token name a; moreover, since $\hat{f}(\mathtt{s}(a)) = f(\mathtt{s}(a))$ for all $\mathtt{s}(a) \in T$ where a is a matching token name, we have that a is not matching. If $t_2 = t_1$, then $\triangleleft = \leq$ (indeed, if it was $\triangleleft = <$, then we would have a contradiction with the hypothesis that f satisfies $t_1 < t_2$), and $\hat{f}(t_1) \leq \hat{f}(t_2)$ trivially holds. If $t_2 = \mathtt{e}(a)$, then, by definition of $\hat{f}$, it holds that $\hat{f}(t_1) = \hat{f}(\mathtt{s}(a)) < \hat{f}(\mathtt{e}(a)) = \hat{f}(t_2)$, which implies $\hat{f}(t_1) < \hat{f}(t_2)$, and thus $\hat{f}(t_1) \triangleleft \hat{f}(t_2)$. Finally, let us consider the last case, with $t_2 \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$. Since f is a relaxed witness and a is not matching, for all terms $t \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$ we have that $\mathtt{s}(a) \leq t \in \hat{\mathcal{C}}$ implies $\mathtt{e}(a) \leq t \in \hat{\mathcal{C}}$, which means that $\mathtt{e}(a) \leq t_2 \in \hat{\mathcal{C}}$. Because f satisfies $\mathcal{C}$, we have that $f(\mathtt{e}(a)) \leq f(t_2)$, and thus it holds that $\hat{f}(\mathtt{s}(a)) < \hat{f}(\mathtt{e}(a)) = f(\mathtt{e}(a)) \leq f(t_2) \leq \hat{f}(t_2)$, which, in turn, implies $\hat{f}(\mathtt{s}(a)) < \hat{f}(t_2)$. Hence, $\hat{f}(t_1) \triangleleft \hat{f}(t_2)$ holds true. □

Next, we lift the notion of witness in order to characterize solution plans for (not necessarily eager) disjunction-free qualitative timeline-based planning problem.

Definition A.4 (rule-level witness)**.** Let $P = (\mathsf{SV}, S)$ be a (not necessarily eager) disjunction-free qualitative timeline-based planning problem, $\mathcal{R} \in S$, and $\lambda \in \Sigma_{\mathsf{SV}}^*$. A *(rule-level) witness*

of $\mathcal{R}$ in λ is a sequence w of functions $\langle f^{tr} \rangle_{tr \in triggers^{\lambda}(\mathcal{R})}$, where f^{tr} is a (trigger-level) witness of tr in λ for all $tr \in triggers^{\lambda}(\mathcal{R})$. Moreover, w is matching (resp., relaxed) if so are all its components.

Definition A.5 (problem-level witness)**.** Let $P = (\mathsf{SV}, S)$ be a (not necessarily eager) disjunction-free qualitative timeline-based planning problem and $\lambda \in \Sigma^*_{\mathsf{SV}}$. A *(problem-level) witness* of S in λ is a sequence $\mathfrak{w}$ of functions $\langle w^{\mathcal{R}} \rangle_{\mathcal{R} \in S}$, where $w^{\mathcal{R}}$ is a (rule-level) witness of $\mathcal{R}$ in λ for all $\mathcal{R} \in S$. Moreover, $\mathfrak{w}$ is matching (resp., relaxed) if so are all its components.

Clearly, a (trigger-/rule-/problem-level) matching witness is also a (trigger-/rule-/problem-level) relaxed one, but the converse does not hold in general. Moreover, it is obvious that problem-level matching witnesses characterize solution plans, also for non-eager instances of the planning problem.

Remark A.1. *Let $P = (\mathsf{SV}, S)$ be a (not necessarily eager) disjunction-free qualitative timeline-based planning problem and λ a finite word over Σ_{SV} that encodes a plan over SV. Then, λ encodes a solution plan for P if and only if there is a matching witness of S in λ.*

It is less obvious that relaxed witnesses characterize solution plans as well.

Lemma A.4. *Let $P = (\mathsf{SV}, S)$ be a (not necessarily eager) disjunction-free qualitative timeline-based planning problem and λ a finite word over Σ_{SV} that encodes a plan over SV. Then, λ encodes a solution plan for P if and only if there is a relaxed witness of S in λ.*

Proof. Thanks to Remark A.1, it is enough to show that if a relaxed witness of S in λ exists, then a matching one exists as well. Let $\mathfrak{w} = \langle w^{\mathcal{R}} \rangle_{\mathcal{R} \in S}$ be a (problem-level) relaxed witness of S in λ, where $w^{\mathcal{R}}$ is a (rule-level) relaxed witness of $\mathcal{R}$ in λ, for all $\mathcal{R} \in S$. Then, the sequence $\hat{w^{\mathcal{R}}}$, obtained from $w^{\mathcal{R}}$ by replacing each (trigger-level) witness f with its matching version $\hat{f}$, is a (rule-level) matching witness of $\mathcal{R}$ in λ, for all $\mathcal{R} \in S$. Finally, $\hat{\mathfrak{w}} = \langle \hat{w^{\mathcal{R}}} \rangle_{\mathcal{R} \in S}$ is a matching witness of S in λ. □

A.2.2. *Canonical eager witnesses for eager planning problems.* For an eager synchronization rule $\mathcal{R}$, whose unique clause is $\mathcal{C}$, and a term t occurring in it, we define sets $pre^{\leq}_{\mathcal{R}}(t)$ of *predecessors* and $pre^{<}_{\mathcal{R}}(t)$ of *strict predecessors* of t:

$$pre^{<}_{\mathcal{R}}(t) = \{s \mid s < t \in \hat{\mathcal{C}}\}$$

$$pre^{\leq}_{\mathcal{R}}(t) = \{s \mid s \leq t \in \hat{\mathcal{C}} \text{ and } t \leq s \notin \hat{\mathcal{C}}, \}$$

Clearly, if $s \leq t \in \hat{\mathcal{C}}$, then $pre^{<}_{R}(s) \subseteq pre^{<}_{\mathcal{R}}(t)$ and $pre^{\leq}_{R}(s) \subseteq pre^{\leq}_{\mathcal{R}}(t)$; therefore, $t_1 \equiv t_2 \in \hat{\mathcal{C}}$ implies $pre^{<}_{R}(s) = pre^{<}_{\mathcal{R}}(t)$ and $pre^{\leq}_{R}(s) = pre^{\leq}_{\mathcal{R}}(t)$.

Now, let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem, $\mathcal{R} \in S$, and $\lambda = \langle \sigma_0, \ldots, \sigma_{|\lambda|-1} \rangle \in \Sigma^*_{\mathsf{SV}}$. Moreover, let $\mathcal{C}$ be the only clause of $\mathcal{R}$ and T be the set of terms (of the form $\mathtt{s}(a)$ and $\mathtt{e}(a)$) appearing in $\mathcal{R}$. For every $tr \in triggers^{\lambda}(\mathcal{R})$, we define a function $f^{tr}_{\lambda} : T \to \mathbb{N}$, and then we show that (since P is eager) if there is a relaxed witness of tr in λ, then f^{tr}_{λ} is one such witness. To this end we need to define an auxiliary functions, namely $l\text{-}bound^{tr}_{\lambda} : T \to \mathbb{N}$. Functions f^{tr}_{λ} and $l\text{-}bound^{tr}_{\lambda}$ are defined by mutual induction as follows. Let a_0 be the trigger token of $\mathcal{R}$, unless $\mathcal{R}$ is triggerless, and $t \in T \setminus \{\mathtt{s}(a_0), \mathtt{e}(a_0)\}$.

- If $\mathcal{R}$ is not triggerless (thus $tr \neq \top$)), let $tr = (m, n) \in \mathbb{N} \times \mathbb{N}$; then $f^{tr}_\lambda(\mathtt{s}(a_0)) = m$ and $f^{tr}_\lambda(\mathtt{e}(a_0)) = n$.
- If $pre^{\leq}_{\mathcal{R}}(t) \cup pre^{<}_{\mathcal{R}}(t) = \varnothing$, then
  - $l\text{-}bound^{tr}_\lambda(t) = 0$ and
  - $f^{tr}_\lambda(t) = \min\{x \in \mathbb{N} \mid term\text{-}to\text{-}event([t]_\equiv) \subseteq events(\sigma_x)\}$.
- Otherwise (i.e., $pre^{\leq}_{\mathcal{R}}(t) \cup pre^{<}_{\mathcal{R}}(t) \neq \varnothing$)
  - $l\text{-}bound^{tr}_\lambda(t) = \max(\{f^{tr}_\lambda(s) \mid s \in pre^{\leq}_{\mathcal{R}}(t)\} \cup \{f^{tr}_\lambda(s) + 1 \mid s \in pre^{<}_{\mathcal{R}}(t)\})$ and
  - $f^{tr}_\lambda(t) = \min\{x \in \mathbb{N} \mid x \geq l\text{-}bound^{tr}_\lambda(t)$ and $term\text{-}to\text{-}event([t]_\equiv) \subseteq events(\sigma_x)\}$.

It is immediate to see that when f^{tr}_λ is well defined (in particular, it is total), it satisfies all atoms $t_1 \leq t_2$ and $t_1 < t_2$ in $\mathcal{C}$, as long as t_2 does not involve a trigger token, and thus, in order to check whether it is a witness of tr in λ (cf. Definition A.1), it suffices to verify atoms $t_1 \leq t_2$ and $t_1 < t_2$ in $\mathcal{C}$, where t_2 involves a trigger token. The other conditions of Definition A.1 are clearly satisfied.

Lemma A.5. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem, $\mathcal{R} \in S$, $\lambda = \langle \sigma_0, \ldots, \sigma_{|\lambda|-1} \rangle \in \Sigma^*_{\mathsf{SV}}$, and $tr \in triggers^\lambda(\mathcal{R})$. If a witness $f : T \to \mathbb{N}$ of tr in λ exists, then $f^{tr}_\lambda : T \to \mathbb{N}$ is well defined and such that $f^{tr}_\lambda(t) \leq f(t)$ for all $t \in T$.*

Proof. Let $t : T \to \mathbb{N}$ be a witness of tr in λ and a_0 be the trigger token of $\mathcal{R}$, unless $\mathcal{R}$ is triggerless. If $\mathtt{s}(a_0), \mathtt{e}(a_0) \in T$ (i.e., $\mathcal{R}$ is not triggerless and thus $tr = (m, n)$ for some natural numbers m and n), then $f^{tr}_\lambda(\mathtt{s}(a_0)) = m = f(\mathtt{s}(a_0))$ and $f^{tr}_\lambda(\mathtt{e}(a_0)) = n = f(\mathtt{e}(a_0))$.

Let $t \in T \setminus \{\mathtt{s}(a_0), \mathtt{e}(a_0)\}$. Since f is a witness of tr in λ, we have that

$$f(t) = i \text{ implies } term\text{-}to\text{-}event([t]_\equiv) \subseteq events(\sigma_i). \tag{5}$$

We show by induction that *(i)* $l\text{-}bound^{tr}_\lambda(t)$ id defined and $l\text{-}bound^{tr}_\lambda(t) \leq f(t)$, and *(ii)* $f^{tr}_\lambda(t)$ is defined and $f^{tr}_\lambda(t) \leq f(t)$. If $pre^{\leq}_{\mathcal{R}}(t) \cup pre^{<}_{\mathcal{R}}(t) = \varnothing$, then *(i)* holds trivially, while *(ii)* follows from Equation 5. If, instead, $pre^{\leq}_{\mathcal{R}}(t) \cup pre^{<}_{\mathcal{R}}(t) \neq \varnothing$, then *(ii)* immediately follows from *(i)* and Equation 5, while we proceed as follows to prove *(i)*. By definition of $l\text{-}bound^{tr}_\lambda$, we have that $l\text{-}bound^{tr}_\lambda(t) = f^{tr}_\lambda(s)$ for some s such that $s \leq t \in \hat{\mathcal{C}}$ or $l\text{-}bound^{tr}_\lambda(t) = f^{tr}_\lambda(s)+1$ for some s such that $s < t \in \hat{\mathcal{C}}$. Observe that $f^{tr}_\lambda(s)$ is defined by induction, and thus $l\text{-}bound^{tr}_\lambda(t)$ is defined, too. By induction, it also holds that $f^{tr}_\lambda(s) \leq f(s)$. In the former case ($l\text{-}bound^{tr}_\lambda(t) = f^{tr}_\lambda(s)$ and $s \leq t \in \hat{\mathcal{C}}$), we have that $f(s) \leq f(t)$, and thus it holds:

$$l\text{-}bound^{tr}_\lambda(t) = f^{tr}_\lambda(s) \leq f(s) \leq f(t);$$

hence *(i)*. In the latter case ($l\text{-}bound^{tr}_\lambda(t) = f^{tr}_\lambda(s) + 1$ and $s < t \in \hat{\mathcal{C}}$), we have that $f(s) < f(t)$, and thus it holds:

$$l\text{-}bound^{tr}_\lambda(t) = f^{tr}_\lambda(s) + 1 \leq f(s) + 1 < f(t) + 1;$$

hence *(i)*. □

Thanks to Lemma A.5, if a witness of tr in λ exists, then f^{tr}_λ is a witness of tr in λ, too. If, in addition, it is a relaxed one (cf. Definition A.2), then we call it the *canonical eager witness* of tr in λ.

Lemma A.6. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem, $\mathcal{R} \in S$, λ a finite word over Σ_{SV} that encodes a plan over SV, and $tr \in triggers^\lambda(\mathcal{R})$. Then, there is a relaxed witness of tr in λ if and only if $f^{tr}_\lambda : T \to \mathbb{N}$ is the canonical eager witness of tr in λ.*

Proof. The "if" direction of the claim trivially holds, since the canonical eager witness of tr in λ is, by definition, a relaxed one. Therefore, it suffices to prove that if a relaxed witness of tr in λ exists, then f_λ^{tr} is the canonical eager witness of tr in λ. As already observed, since a relaxed witness of tr in λ exists, f_λ^{tr} is indeed a witness of tr in λ. In the following, we show that all non-matching token names a appearing in T violate Item *A2* of Definition 3.1. Towards a contradiction, assume that there is a token name a that is not matching in f_λ^{tr} and λ for which Item *A2* of Definition 3.1 holds true. Notice that a is matching in f and λ, or f would not be a relaxed witness. Therefore it must be $f_\lambda^{tr}(\mathtt{s}(a)) < f(\mathtt{s}(a))$: indeed, if it was $f_\lambda^{tr}(\mathtt{s}(a)) = f(\mathtt{s}(a))$, then it would be $f_\lambda^{tr}(\mathtt{e}(a)) > f(\mathtt{e}(a))$ (otherwise a is matching in f_λ^{tr} and λ), which is in contradiction with Lemma A.5. We show that a satisfies also Item *A1* and Item B of Definition 3.1. Since a is not a matching token name, it is not a trigger token either (cf. Definition A.2), and thus it is ambiguous, which means that $\mathcal{R}$, and thus P, are not eager (cf. Definition 3.1), which contradicts the hypothesis of P being eager.

First, we focus on Item *A1* of Definition 3.1. If $\mathcal{R}$ is triggerless, then the condition is vacuously verified. Otherwise, let a_0 be the trigger token of $\mathcal{R}$ and let $tr = (m, n)$ for $m, n \in \mathbb{N}$. If $\mathtt{s}(a) \equiv \mathtt{s}(a_0) \in \hat{\mathcal{C}}$, then $f_\lambda^{tr}(\mathtt{s}(a)) = f_\lambda^{tr}(\mathtt{s}(a_0)) = m = f(\mathtt{s}(a_0)) = f(\mathtt{s}(a))$, which is in contradiction with $f_\lambda^{tr}(\mathtt{s}(a)) < f(\mathtt{s}(a))$. If $\mathtt{s}(a) \equiv \mathtt{e}(a_0) \in \hat{\mathcal{C}}$, then $f_\lambda^{tr}(\mathtt{s}(a)) = f_\lambda^{tr}(\mathtt{e}(a_0)) = n = f(\mathtt{e}(a_0)) = f(\mathtt{s}(a))$, which is in contradiction with $f_\lambda^{tr}(\mathtt{s}(a)) < f(\mathtt{s}(a))$. Therefore, it must be $\mathtt{s}(a) \equiv \mathtt{s}(a_0) \notin \hat{\mathcal{C}}$ and $\mathtt{s}(a) \equiv \mathtt{e}(a_0) \notin \hat{\mathcal{C}}$, which means that Item *A1* of Definition 3.1 is satisfied.

We turn now to Item B of Definition 3.1. Let $\lambda = \langle \sigma_0, \dots, \sigma_{|\lambda|-1} \rangle$ and suppose $a[x = v]$ occurs in $\mathcal{R}$. Since λ encodes a plan and a is not a matching token name in f_λ^{tr} and λ, there is a natural number i such that $f_\lambda^{tr}(\mathtt{s}(a)) < i < f_\lambda^{tr}(\mathtt{e}(a))$ and $\mathit{end}(x, v) \in \mathit{events}(\sigma_i)$. Let $\hat{i}$ be the smallest among such numbers, that is, $\hat{i} = \min\{i \in \mathbb{N} \mid f_\lambda^{tr}(\mathtt{s}(a)) < i < f_\lambda^{tr}(\mathtt{e}(a))$ and $\mathit{end}(x, v) \in \mathit{events}(\sigma_i)\}$. There are two possible reason for $f_\lambda^{tr}(\mathtt{e}(a)) > \hat{i}$:

- $\hat{i} < \mathit{l\text{-}bound}_\lambda^{tr}(\mathtt{e}(a))$ or
- $\mathit{term\text{-}to\text{-}event}([\mathtt{e}(a)]_\equiv) \not\subseteq \mathit{events}(\sigma_{\hat{i}})$.

In the former case, by the definition of $\mathit{l\text{-}bound}_\lambda^{tr}$, we have that there exists a term t such that $t \leq \mathtt{e}(a) \in \hat{\mathcal{C}}$, $\mathtt{e}(a) \leq t \notin \hat{\mathcal{C}}$, and $f_\lambda^{tr}(t) \geq \hat{i}$. By $\mathtt{e}(a) \leq t \notin \hat{\mathcal{C}}$, it holds that $\mathtt{e}(a) \equiv t \notin \hat{\mathcal{C}}$, and thus $t \neq \mathtt{e}(a)$. It also holds that $t \leq \mathtt{s}(a) \notin \hat{\mathcal{C}}$, or f_λ^{tr} would not satisfy $\mathcal{C}$, because of $f_\lambda^{tr}(t) \geq \hat{i} > f_\lambda^{tr}(\mathtt{s}(a))$. Therefore, $t \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$, and Item B of Definition 3.1 is verified. In the latter case (i.e., $\mathit{term\text{-}to\text{-}event}([\mathtt{e}(a)]_\equiv) \not\subseteq \mathit{events}(\sigma_{\hat{i}})$), since $\mathit{end}(x, v) \in \mathit{events}(\sigma_{\hat{i}})$, there is $t \neq \mathtt{e}(a)$ such that $t \equiv \mathtt{e}(a) \in \hat{\mathcal{C}}$ and $\mathit{term\text{-}to\text{-}event}(t) \notin \mathit{events}(\sigma_{\hat{i}})$. Clearly, $t \neq \mathtt{s}(a)$ and $t \leq \mathtt{s}(a) \notin \hat{\mathcal{C}}$ (because $\mathtt{s}(a) < \mathtt{e}(a) \in \hat{\mathcal{C}}$). Therefore, $t \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$, and Item B of Definition 3.1 is verified. □

Corollary A.1. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem and λ a finite word over Σ_{SV} that encodes a plan over SV. Then, λ encodes a solution plan for P if and only if f_λ^{tr} is the canonical eager witness of tr in λ for all $\mathcal{R} \in S$ and $tr \in \mathit{triggers}^\lambda(\mathcal{R})$.*

A.2.3. *Correspondence between automata and witnesses.* In this section, we establish a correspondence between automata and witnesses. The claim then follows. Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem, $\lambda = \langle \sigma_0, \dots, \sigma_{n-1} \rangle$ a finite word over Σ_{SV} that encodes a plan over SV, and $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_{n-1}} q_n$ the run of $\mathcal{A}_P$ on λ.

We associate to π a set $\mathcal{F}(\pi)$ of function and we show that $\mathcal{A}_P$ accepts λ if and only if $\mathcal{F}(\pi)$ is the set of canonical eager witnesses of tr in λ for all $\mathcal{R} \in S$ and $tr \in \mathcal{R}$.

First, for every $i \in \{1, \ldots, n\}$, with $q_i \neq \mathbf{s}_P$, and every $\mathbb{V} = (G, K) \in q_i$, we define the *pre-image* of $\mathbb{V}$ in q_i, denoted $pre_{q_i}(\mathbb{V})$, as the set $\{\mathbb{V}' \in q_{i-1} \mid evol(\mathbb{V}', \sigma_{i-1}) = \mathbb{V}\}$. Notice that $pre_{q_i}(\mathbb{V}) \neq \varnothing$. Then, $\mathcal{F}_{q_i}(\mathbb{V})$ is a set of functions from terms (of the forms $\mathtt{s}(a)$ and $\mathtt{e}(a)$) to natural numbers defined inductively as follows.

- $\mathcal{F}_{q_0}(\mathbb{V}) = \{\varnothing\}$ for all $\mathbb{V} \in q_0$ (the empty function is associated to $\mathcal{F}_{q_0}(\mathbb{V})$).
- $\mathcal{F}_{q_i}(\mathbb{V}) = \{f[K \setminus K' \mapsto i-1] \mid f \in \mathbb{V}', \mathbb{V}' = (G, K') \in pre_{q_i}(\mathbb{V})\}$, where, for a function f, a set of sets of terms K, with $\kappa \cap dom(f) = \varnothing$ for all $\kappa \in K$, and a natural number n, we denote by $f[K \mapsto n]$ the function obtained from f by assigning value n to all terms $t \in \kappa$ for some set of terms $\kappa \in K$. Formally, $f[K \mapsto n](t) = f(t)$ if $t \in dom(f)$, $f[K \mapsto n](t) = n$ if $t \in \bigcup_{\kappa \in K} \kappa$, $f[K \mapsto n](t)$ is undefined otherwise.

Notice that, for every $f \in \mathcal{F}_{q_i}(\mathbb{V})$, it holds that $dom(f) = \bigcup_{[t]_\equiv \in K} [t]_\equiv$, and that $f(t) < i$, for all $t \in dom(f)$. Consequently, it is not difficult to see that f satisfies all atoms $t_1 \leq t_2 \in \mathcal{C}$ (resp., $t_1 < t_2 \in \mathcal{C}$), where $t_1, t_2 \in dom(f)$ and $\mathcal{C}$ is the unique clause of $rule(\mathbb{V})$. Moreover, if $rule(\mathbb{V})$ has trigger $a_0[x_0 = v_0]$ and $\mathbb{V}$ is enabled (i.e., $[\mathtt{s}(a_0)]_\equiv \in K$), then f identifies univocally a trigger in $triggers^\lambda(\mathcal{R})$, denoted by $trgr^\lambda(f)$ and defined as $trgr^\lambda(f) = (f(\mathtt{s}(a_0)), matching\text{-}end_f^\lambda(a_0)) \in \mathbb{N} \times \mathbb{N}$, where, $matching\text{-}end_f^\lambda(a_0)$ is defined analogously to $matching\text{-}start_f^\lambda(a_0)$, that is, $matching\text{-}end_f^\lambda(a_0) = \min_i\{i \in \mathbb{N} \mid f(\mathtt{s}(a_0)) < i \text{ and } end(x_0, v_0) \in events(\sigma_i)\}$. If $rule(\mathbb{V})$ is triggerless, then we set $trgr^\lambda(f) = \top$.

Lemma A.7. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem, $\lambda = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle$ a finite word over Σ_{SV} that encodes a plan over SV, and $\pi = q_0 \overset{\sigma_0}{\to} q_1 \overset{\sigma_1}{\to} \ldots \overset{\sigma_{n-1}}{\to} q_n$ the run of $\mathcal{A}_P$ on λ. For all $i \in \{0, \ldots, n\}$, with $q_i \neq \mathbf{s}_P$, all $\mathbb{V} \in q_i$, with $\mathbb{V}$ enabled, and all $f \in \mathcal{F}_{q_i}(\mathbb{V})$, it holds that $f(t) = j$ if and only if $f_\lambda^{tr}(t) = j$ for all $t \in T$ and all natural numbers $j < i$, where $tr = trgr^\lambda(f)$ and T is the set of terms (of the form $\mathtt{s}(a)$ and $\mathtt{e}(a)$) appearing in $rule(\mathbb{V})$.*

Proof. We prove, by induction on t, the following equivalent statement. For all $t \in T$ and all $j \in \{0, \ldots, i\}$, it holds that $f(t) < j$ if and only if $f_\lambda^{tr}(t) < j$. Assume $f(t') < j$ if and only if $f_\lambda^{tr}(t') < j$ holds true for all $j \in \{0, \ldots, i\}$ and all terms t' such that $t' \leq t \in \hat{\mathcal{C}}$ and $t < t' \notin \hat{\mathcal{C}}$ (inductive hypothesis).

"only if" direction. Assume $f(t) < j$. We show that $f_\lambda^{tr}(t) < j$ holds. From $f(t) < j$, there are a natural number $h < j$, viewpoints $\mathbb{V}_h = (G, K_h) \in q_h$ and $\mathbb{V}_{h+1} = (G, K_{h+1}) \in q_{h+1}$, and functions $f_h \in \mathcal{F}_{q_h}(\mathbb{V}_h)$ and $f_{h+1} \in \mathcal{F}_{q_{h+1}}(\mathbb{V}_{h+1})$ such that $evol(\mathbb{V}_h, \sigma_h) = \mathbb{V}_{h+1}$, $f_{h+1} = f_h[K_{h+1} \setminus K_h \mapsto h]$, and $t \in \bigcup_{[s]_\equiv \in K_{h+1} \setminus K_h} [s]_\equiv$ (i.e., $t \in [s]_\equiv$ for some $[s]_\equiv \in K_{h+1} \setminus K_h$). To prove that $f_\lambda^{tr}(t) < j$ holds, we show the following:

(i) $h \geq f_\lambda^{tr}(t')$ for all t' with $t' \leq t \in \hat{\mathcal{C}}$ and $t \leq t' \notin \hat{\mathcal{C}}$. By the downward closure of K_{h+1}, we have that $t' \in \bigcup_{[s]_\equiv \in K_{h+1}} [s]_\equiv$. Thus, $f(t') \leq h$. By inductive hypothesis, $f(t') \leq h$ implies $f_\lambda^{tr}(t') \leq h$.

(ii) $h > f_\lambda^{tr}(t')$ for all t' with $t' < t \in \hat{\mathcal{C}}$. By the definition of *next-available*$(\mathbb{V}_h)$ in Section 7.1, we have $t' \in \bigcup_{[s]_\equiv \in K_h} [s]_\equiv$. Thus, $f(t') < h$. By inductive hypothesis, $f(t') < h$ implies $f_\lambda^{tr}(t') < h$.

(iii) *term-to-event*$([t]_\equiv) \subseteq events(\sigma_h)$. It immediately follows from *term-to-event*$(K_{h+1} \setminus K_h) \subseteq events(\sigma_h)$ (cf. Section 7.1) and $[t]_\equiv \in K_{h+1} \setminus K_h$

From *(i)*, *(ii)*, and *(iii)* above, it follows that $f^{tr}(t) \leq h < j$.

"if" direction. Assume $f^{tr}_{\lambda}(t) = h < j$. We show that $f(t) < j$ holds. The assumption $f^{tr}_{\lambda}(t) = h$ implies that *(i)* $h \geq f^{tr}_{\lambda}(t')$ for all t' with $t' \leq t \in \hat{\mathcal{C}}$ and $t \leq t' \notin \hat{\mathcal{C}}$, *(ii)* $h > f^{tr}_{\lambda}(t')$ for all t' with $t' < t \in \hat{\mathcal{C}}$, and *(iii)* *term-to-event*$([t]_{\equiv}) \subseteq$ *events*(σ_h). From *(i)* and *(ii)* above, it follows that there are viewpoints $\mathbb{V}_h = (G, K_h) \in q_h$ and $\mathbb{V}_{h+1} = (G, K_{h+1}) \in q_{h+1}$ such that $t' \in \bigcup_{[s]_{\equiv} \in K_{h+1}} [s]_{\equiv}$ for all t' with $t' \leq t \in \hat{\mathcal{C}}$ and $t \leq t' \notin \hat{\mathcal{C}}$, (due to *(i)*), $t' \in \bigcup_{[s]_{\equiv} \in K_h} [s]_{\equiv}$ for all t' with $t' < t \in \hat{\mathcal{C}}$, and (due to *(ii)*). Therefore, $[t]_{\equiv} \in$ *next-available*$(\mathbb{V}_h)$. From *(iii)* above, it follows that $[t]_{\equiv} \in$ *consumed*$(\mathbb{V}_h, \sigma_h) = K_{h+1}$. Hence, $f(t) = h < j$. □

The following corollary immediately follows (recall that the last state of an accepting run is not the rejecting sink state $\mathbf{s}_P$ and it is such that all enabled viewpoints in it are final).

Corollary A.2. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem, $\lambda = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle$ a finite word over Σ_{SV} that encodes a plan over SV, and $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_{n-1}} q_n$ the run of $\mathcal{A}_P$ on λ. If π is accepting, then for all $\mathbb{V} \in q_n$, with $\mathbb{V}$ enabled, and all $f \in \mathcal{F}_{q_n}(\mathbb{V})$, we have that $f = f^{tr}_{\lambda}$, where $tr = trgr^{\lambda}(f)$.*

The next lemma establishes the correctness of out automata construction.

Lemma A.8. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem, $\lambda = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle$ a finite word over Σ_{SV} that encodes a plan over SV, and $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_{n-1}} q_n$ the run of $\mathcal{A}_P$ on λ.*

(1) *If π is accepting, then $\bigcup_{\mathbb{V} \in q_n, \mathbb{V} enabled} \mathcal{F}_{q_n}(\mathbb{V}) = \{f^{tr}_{\lambda} \mid \mathcal{R} \in S, tr \in triggers^{\lambda}(\mathcal{R})\}$.*
(2) *If π is accepting, then for every $f \in \bigcup_{\mathbb{V} \in q_n, \mathbb{V} enabled} \mathcal{F}_{q_n}(\mathbb{V})$, it holds that f is the canonical eager witness of $trgr^{\lambda}(f)$ in λ.*
(3) *If π is rejecting, then there are $\mathcal{R} \in S$ and $tr \in triggers^{\lambda}(\mathcal{R})$ such that f^{tr}_{λ} is not the canonical eager witness of tr in λ.*

Proof. (1) The left-to-right inclusion immediately follows from Corollary A.2. In order to proof the converse inclusion, we proceed as follows. Let $\mathcal{R} \in S$ and $tr \in triggers^{\lambda}(\mathcal{R})$. We show that there are $\mathbb{V}_n \in q_n$, with $\mathbb{V}_n$ enabled, and $f \in \mathcal{F}_{q_n}(\mathbb{V}_n)$ such that $f = f^{tr}_{\lambda}$. If $\mathcal{R}$ is triggerless, then the claim follows immediately from Corollary A.2 (recall that viewpoints for triggerless rules are always enabled). If, instead, $\mathcal{R}$ is not triggerless, with trigger $a_0[x_0 = v_0]$, then $tr = (h, k)$, for some $h, k \in \mathbb{N} \times \mathbb{N}$, with $h < n - 1$, and $start(x_0, v_0) \in events(\sigma_h)$ (i.e., σ_h triggers $\mathcal{R}$). Note that, from π being accepting, it follows that $q_i \neq \mathbf{s}_P$ for all $i \in \{0, 1, \ldots, n\}$ and that q_i is compatible with σ_i for all $i \in \{0, 1, \ldots, n-1\}$. Then, since q_h is compatible with σ_h and σ_h triggers $\mathcal{R}$, there is $\mathbb{V}_h = (G, K_h) \in q_h$ for $\mathcal{R}$ such that $[\mathtt{s}(a_0)]_{\equiv} \in consumed(\mathbb{V}_h, \sigma_h) \setminus K_h$. Therefore, there is $\mathbb{V}_{h+1} = (G, K_{h+1}) \in q_{h+1}$ such that $[\mathtt{s}(a_0)]_{\equiv} \in K_{h+1} \setminus K_h$, which means that there are $\mathbb{V}_n \in q_n$ and $f \in \mathcal{F}_{q_n}(\mathbb{V}_n)$ such that $\mathbb{V}_n$ is enabled and $f(\mathtt{s}(a_0)) = h$. By Corollary A.2, $f = f^{tr'}_{\lambda}$, where $tr' = tr$ (because $tr' = trgr^{\lambda}(f) = (f(\mathtt{s}(a_0)), matching\text{-}end^{\lambda}_f(a_0)) = (h, k) = tr$).
(2) Let $\mathbb{V} \in q_n$, with $\mathbb{V}$ enabled, and $f \in \mathcal{F}_{q_n}(\mathbb{V})$. By the previous item, $f = f^{tr}_{\lambda}$, with $tr = trgr^{\lambda}(f)$. Moreover, by definition, f is a witness of $trgr^{\lambda}(f)$ in λ. We show that it is a relaxed one, i.e., for all non-matching token names a appearing in $dom(f)$ and all terms $t \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$ it holds that $\mathtt{s}(a) \leq t \in \hat{\mathcal{C}}$ implies $\mathtt{e}(a) \leq t \in \hat{\mathcal{C}}$, where $\mathcal{C}$ is the

unique clause of $\mathit{rule}(\mathbb{V})$ (cf. Definition A.2). Let a be a non-matching token name in f and λ, with $a[x = v]$ appearing in $\mathit{rule}(\mathbb{V})$, and let $m = \mathit{matching\text{-}end}_f^{\lambda}(a) < f(\mathtt{e}(a))$. Then, $\mathtt{s}(a), \mathtt{e}(a) \in \mathit{dom}(f)$ and $\mathit{end}(x, v) \in \mathit{events}(\sigma_m)$. Since π is accepting and given that $f(\mathtt{e}(a)) > m$, there are a viewpoint $\mathbb{V} \in q_m$ and a function $f' \in \mathcal{F}_{q_m}(\mathbb{V})$ such that $f'(\mathtt{s}(a)) = f(\mathtt{s}(a))$ and $\mathtt{e}(a) \notin \mathit{waiting}(\mathbb{V})$ (or it would be $f(\mathtt{e}(a)) = m$, leading to a contradiction). Therefore, by the definition of $\mathit{waiting}(\mathbb{V})$, it must be that $\mathtt{s}(a) \leq t \in \hat{\mathcal{C}}$ implies $\mathtt{e}(a) \leq t \in \hat{\mathcal{C}}$ for all terms $t \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$, which concludes the proof.

(3) Since π is rejecting, there are two possibilities:

A. $q_n \neq \mathbf{s}_P$ and there is $\mathbb{V} \in q_n$ that is enabled but not final;

B. there is $i \in \{0, 1, n-1\}$ such that $q_i \neq \mathbf{s}_P$ and $q_{i+1} = \mathbf{s}_P$.

In the former case, there is $f \in \mathcal{F}_{q_n}(\mathbb{V})$ that is not defined for some term t appearing in $\mathit{rule}(\mathbb{V})$. By Lemma A.7, we have that $f = f_\lambda^{tr}$, where $tr = \mathit{trgr}^{\lambda}(f)$. Therefore, f_λ^{tr} is a partial function, and thus it is not the canonical eager witness of tr in λ (as a matter of fact f_λ^{tr} is not even a witness tr in λ).

In the latter case, we further distinguish two cases:

B.i there is a viewpoint $\mathbb{V} = (G, K) \in q_i$ that is not compatible with σ_i;

B.ii there is a rule $\mathcal{R} \in S$ such that $\mathcal{R}$ is not triggerless, σ_i triggers $\mathcal{R}$, and there is no viewpoint $\mathbb{V} \in q_i$ for $\mathcal{R}$ such that σ_i enables $\mathbb{V}$.

We deal with these two cases separately.

B.i $\mathbb{V}$ being not compatible with σ_i implies that $\mathit{term\text{-}to\text{-}event}(\mathit{waiting}(\mathbb{V})) \cap \mathit{events}(\sigma_i) \subseteq \mathit{term\text{-}to\text{-}event}(\mathit{consumed}(\mathbb{V}, \sigma_i) \setminus K)$. Thus, there is a token name a such that $a[x = v]$ appears in $\mathit{rule}(\mathbb{V})$, $\mathtt{e}(a) \in \mathit{waiting}(\mathbb{V})$, $\mathit{end}(x, v) \in \mathit{events}(\sigma_i)$, and $[\mathtt{e}(a)]_{\equiv} \notin \mathit{consumed}(\mathbb{V}, \sigma_i) \setminus K$.

Let $f \in \mathcal{F}_{q_i}(\mathbb{V})$. By $\mathtt{e}(a) \in \mathit{waiting}(\mathbb{V})$, we know that $[\mathtt{s}(a)]_{\equiv} \in K$, meaning that $\mathbb{V}$ is enabled. Therefore, f identifies univocally the trigger $\mathit{trgr}^{\lambda}(f)$ in $\mathit{triggers}^{\lambda}(\mathcal{R})$. Moreover, by Lemma A.7, we have $f(t) = j$ if and only if $f_\lambda^{tr}(t) = j$, for all terms t appearing in $\mathit{rule}(\mathbb{V})$ and all $j \in \mathbb{N}$, with $j < i$. We show that f_λ^{tr} is not the canonical eager witness of tr in λ.

From $\mathtt{e}(a) \in \mathit{waiting}(\mathbb{V})$, it follows $[\mathtt{e}(a)]_{\equiv} \notin K$; thus, since $[\mathtt{e}(a)]_{\equiv} \notin \mathit{consumed}(\mathbb{V}, \sigma_i) \setminus K$, it holds that $[\mathtt{e}(a)]_{\equiv} \notin \mathit{consumed}(\mathbb{V}, \sigma_i)$. By the definition of $\mathit{consumed}(\mathbb{V}, \sigma_i)$, there are two possibilities, both leading to $f_\lambda^{tr}(\mathtt{e}(a)) > i$.

- If $[\mathtt{e}(a)]_{\equiv} \notin \mathit{next\text{-}available}(\mathbb{V})$, then there is t such that $t < \mathtt{e}(a) \in \hat{\mathcal{C}}$ and $[t]_{\equiv} \notin K$. In this case, we have $f(t) \geq i$, that implies $f_\lambda^{tr}(t) \geq i$. If f_λ^{tr} was a witness of tr in λ, then we would have $f_\lambda^{tr}(\mathtt{e}(a)) > f_\lambda^{tr}(t) \geq i$.
- Otherwise, there must be t such that $t \leq \mathtt{e}(a) \in \hat{\mathcal{C}}$, $[t]_{\equiv} \notin \mathit{consumed}(\mathbb{V}, \sigma_i)$, and $\mathit{term\text{-}to\text{-}event}(t) \notin \mathit{events}(\sigma_i)$. Therefore, also in this case, we have $f_\lambda^{tr}(\mathtt{e}(a)) > i$.

From $f_\lambda^{tr}(\mathtt{e}(a)) > i$, it follows that a is not matching in f_λ^{tr} and λ. Therefore, a is not the trigger token of $\mathit{rule}(\mathbb{V})$. By $\mathtt{e}(a) \in \mathit{waiting}(\mathbb{V})$, it must then be the case that there is a term $t \notin \{\mathtt{s}(a), \mathtt{e}(a)\}$ such that $\mathtt{s}(a) \leq t \in \hat{\mathcal{C}}$ and $\mathtt{e}(a) \leq t \notin \hat{\mathcal{C}}$. This, together with a being not matching, implies that f_λ^{tr} is not a relaxed witness, and thus it is not the canonical eager witness of tr in λ.

B.ii Assume there is a rule $\mathcal{R} \in S$ such that $\mathcal{R}$ is not triggerless, with trigger $a_0[x_0 = v_0]$, σ_i triggers $\mathcal{R}$, and there is no viewpoint $\mathbb{V} \in q_i$ for $\mathcal{R}$ such that σ_i enables $\mathbb{V}$. Then, from the fact that σ_i triggers $\mathcal{R}$, it follows $\mathit{term\text{-}to\text{-}event}(\mathtt{s}(a_0)) \in \mathit{events}(\sigma_i)$, and thus $(i, j) \in \mathit{triggers}^{\lambda}(\mathcal{R})$, for some $j \in \mathbb{N}$. Assume, towards a

contradiction, that $f_\lambda^{(i,j)}$ is the canonical eager witness of (i,j) in λ. Therefore, $f_\lambda^{(i,j)}(\mathtt{s}(a)) = i$. However, let $\mathbb{V} \in q_i$ be a viewpoint for $\mathcal{R}$ and let $f \in \mathcal{F}_{q_i}(\mathbb{V})$. Since σ_i does not enable $\mathbb{V}$, we distinguish two scenarios.

- If there is t such that $t < \mathtt{s}(a) \in \hat{\mathcal{C}}$ and $[t]_\equiv \notin K$, then $f(t) \geq i$, that implies $f_\lambda^{(i,j)}(t) \geq i$. If $f_\lambda^{(i,j)}$ was a witness of (i,j) in λ, then we would have $f_\lambda^{(i,j)}(\mathtt{s}(a)) > f_\lambda^{(i,j)}(t) \geq i$, which is in contradiction with $f_\lambda^{(i,j)}(\mathtt{s}(a)) = i$.
- Otherwise, there must be t such that $t \leq \mathtt{s}(a) \in \hat{\mathcal{C}}$, $[t]_\equiv \notin \mathit{consumed}(\mathbb{V}, \sigma_i)$, and $\mathit{term\text{-}to\text{-}event}(t) \notin \mathit{events}(\sigma_i)$. Therefore, also in this case, we have $f_\lambda^{tr}(\mathtt{s}(a)) > i$, which is in contradiction with $f_\lambda^{(i,j)}(\mathtt{s}(a)) = i$. □

Corollary A.3. *Let $P = (\mathsf{SV}, S)$ be an eager qualitative timeline-based planning problem and λ a finite word over Σ_{SV} that encodes a plan over SV. $\mathcal{A}_P$ accepts λ if and only if f_λ^{tr} is the canonical eager witness of tr in λ for all $\mathcal{R} \in S$ and $tr \in \mathit{triggers}^\lambda(\mathcal{R})$.*

Lemma A.2 now follows from Corollary A.1 and Corollary A.3, thus completing the proof of Lemma 7.1.

References

[ADMG+24] Renato Acampora, Dario Della Monica, Luca Geatti, Nicola Gigante, and Angelo Montanari. Synthesis of timeline-based planning strategies avoiding determinization. In *Proc. of the 15th International Symposium on Games, Automata, Logics and Formal Verification (GandALF)*, 2024.

[AGG+22] Renato Acampora, Luca Geatti, Nicola Gigante, Angelo Montanari, and Valentino Picotti. Controller synthesis for timeline-based games. In Pierre Ganty and Dario Della Monica, editors, *Proceedings of the 13th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2022, Madrid, Spain, September 21-23, 2022*, volume 370 of *EPTCS*, pages 131–146, 2022. doi:10.4204/EPTCS.370.9.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.

[BMMP18a] Laura Bozzelli, Alberto Molinari, Angelo Montanari, and Adriano Peron. Complexity of timeline-based planning over dense temporal domains: Exploring the middle ground. In Andrea Orlandini and Martin Zimmermann, editors, *Proceedings of the 9th International Symposium on Games, Automata, Logics, and Formal Verification*, volume 277 of *EPTCS*, pages 191–205, 2018. doi:10.4204/EPTCS.277.14.

[BMMP18b] Laura Bozzelli, Alberto Molinari, Angelo Montanari, and Adriano Peron. Decidability and complexity of timeline-based planning over dense temporal domains. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning*, pages 627–628. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/KR/KR18/paper/view/17995.

[CKJZ15] Yongsun Choi, Petcharat Kongsuwan, Cheol Min Joo, and J. Leon Zhao. Stepwise structural verification of cyclic workflow models with acyclic decomposition and reduction of loops. *Data & Knowledge Engineering*, 95:39–65, 2015. doi:10.1016/j.datak.2014.11.003.

[COS19] Carlo Combi, Barbara Oliboni, and Pietro Sala. Customizing BPMN diagrams using timelines. In Johann Gamper, Sophie Pinchinat, and Guido Sciavicco, editors, *26th International Symposium on Temporal Representation and Reasoning, TIME 2019, October 16-19, 2019, Málaga, Spain*, volume 147 of *LIPIcs*, pages 5:1–5:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.TIME.2019.5, doi:10.4230/LIPICS.TIME.2019.5.

[COU16] Marta Cialdea Mayer, Andrea Orlandini, and Alessandro 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.

[CRK+00] Steve A. Chien, Gregg Rabideau, Russell L. Knight, Rob Sherwood, Barbara E. Engelhardt, D. Mutz, Tara Estlin, B. Smith, Forest Fisher, T. Barrett, G. Stebbins, and Daniel Tran. Aspen - automating space mission operations using automated planning and scheduling, 2000.

[CST+04] Steve A. Chien, Rob Sherwood, Daniel Tran, Benjamin Cichy, Gregg Rabideau, Rebecca Castaño, Ashley Davies, Rachel Lee, Dan Mandl, Stuart Frye, Bruce Trout, Jerry Hengemihle, Jeff D'Agostino, Seth Shulman, Stephen G. Ungar, Thomas Brakke, Darrell Boyer, Jim Van Gaasbeck, Ronald Greeley, Thomas Doggett, Victor R. Baker, James M. Dohm, and Felipe 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.

[DMGLTM20] Dario Della Monica, Nicola Gigante, Salvatore La Torre, and Angelo Montanari. Complexity of qualitative timeline-based planning. In Emilio Muñoz-Velasco, Ana Ozaki, and Martin Theobald, editors, *27th International Symposium on Temporal Representation and Reasoning, TIME 2020, September 23-25, 2020, Bozen-Bolzano, Italy*, volume 178 of *LIPIcs*, pages 16:1–16:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.TIME.2020.16.

[DMGM+17] Dario Della Monica, Nicola Gigante, Angelo Montanari, Pietro Sala, Guido Sciavicco, et al. Bounded timed propositional temporal logic with past captures timeline-based planning with bounded constraints. In *IJCAI*, pages 1008–1014. International Joint Conferences on Artificial Intelligence, 2017. doi:10.24963/IJCAI.2017/140.

[DMGMS18] Dario Della Monica, Nicola Gigante, Angelo Montanari, and Pietro Sala. A novel automata-theoretic approach to timeline-based planning. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning*, pages 541–550. AAAI Press, 2018.

[FCO+11] Simone Fratini, Amedeo Cesta, Andrea Orlandini, Riccardo Rasconi, and Riccardo De Benedictis. Apsi-based deliberation in goal oriented autonomous controllers, 2011.

[FL03] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003. doi:10.1613/jair.1129.

[GMCO17] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, and Andrea Orlandini. Complexity of timeline-based planning. In Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith, editors, *Proceedings of the 27th International Conference on Automated Planning and Scheduling*, pages 116–124. AAAI Press, 2017. doi:10.1609/icaps.v27i1.13830.

[GMMO16] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer, and Andrea Orlandini. Timelines are expressive enough to capture action-based temporal planning. In Curtis E. Dyreson, Michael R. Hansen, and Luke Hunsberger, editors, *23rd International Symposium on Temporal Representation and Reasoning,*, pages 100–109. IEEE Computer Society, 2016. doi:10.1109/TIME.2016.18.

[GMO+20] Nicola Gigante, Angelo Montanari, Andrea Orlandini, Marta Cialdea Mayer, and Mark Reynolds. On timeline-based games and their complexity. *Theoretical Computer Science*, 815:247–269, 5 2020. doi:10.1016/j.tcs.2020.02.011.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, 3rd edition, 2006.

[KH04] Jana Koehler and Rainer Hauser. Untangling unstructured cyclic flows – A solution based on continuations. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3290 of *Lecture Notes in Computer Science*, pages 121–138. Springer, 2004. doi:10.1007/978-3-540-30468-5_10.

[Mus94] Nicola Muscettola. HSTS: Integrating Planning and Scheduling. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 6, pages 169–212. Morgan Kaufmann, 1994.

[PCH22] Thomas M. Prinz, Yongsun Choi, and N. Long Ha. Understanding and decomposing control-flow loops in business process models. In *Business Process Management*, volume 13420 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2022. doi:10.1007/978-3-031-16103-2_21.

[PGBD12] Artem Polyvyanyy, Luciano García-Bañuelos, and Marlon Dumas. Structuring acyclic process models. *Information Systems*, 37(6):518–538, 2012. doi:10.1016/j.is.2011.10.005.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *16th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989. doi:10.1007/BFB0035790.

[UCCO17] Alessandro Umbrico, Amedeo Cesta, Marta Cialdea Mayer, and Andrea Orlandini. Platinum: A new framework for planning and acting. In Floriana Esposito, Roberto Basili, Stefano Ferilli, and Francesca A. Lisi, editors, *Proceedings of the 16th International Conference of the Italian Association for Artificial Intelligence*, volume 10640 of *LNCS*, pages 498–512. Springer, 2017. doi:10.1007/978-3-319-70169-1_37.

[UCO23] Alessandro Umbrico, Amedeo Cesta, and Andrea Orlandini. Human-aware goal-oriented autonomy through ros-integrated timeline-based planning and execution. In *32nd IEEE International Conference on Robot and Human Interactive Communication*, pages 1164–1169. IEEE, 2023. doi:10.1109/RO-MAN57019.2023.10309516.