Validating Domains and Plans for Temporal Planning via Encoding into Infinite-State Linear Temporal Logic

Alessandro Cimatti
Fondazione Bruno Kessler
Trento, 38123, Italy
cimatti@fbk.eu

Andrea Micheli
Fondazione Bruno Kessler
Trento, 38123, Italy
amicheli@fbk.eu

Marco Roveri
Fondazione Bruno Kessler
Trento, 38123, Italy
roveri@fbk.eu

Abstract

Temporal planning is an active research area of Artificial Intelligence because of its many applications ranging from robotics to logistics and beyond. Traditionally, authors focused on the automatic synthesis of plans given a formal representation of the domain and of the problem. However, the effectiveness of such techniques is limited by the complexity of the modeling phase: it is hard to produce a correct model for the planning problem at hand.

In this paper, we present a technique to simplify the creation of correct models by leveraging formal-verification tools for automatic validation. We start by using the ANML language, a very expressive language for temporal planning problems that has been recently presented. We chose ANML because of its usability and readability. Then, we present a sound-and-complete, formal encoding of the language into Linear Temporal Logic over predicates with infinite-state variables. Thanks to this reduction, we enable the formal verification of several relevant properties over the planning problem, providing useful feedback to the modeler.

Introduction

Automated planning aims at synthesizing a course of actions suitable to achieve a desired objective, given a domain model describing the effect of the actions. Planning is a mature research field, that has known an impressive period of improvement. Several tools and techniques able to handle increasingly larger problems exist (Vallati et al. 2015).

Planners rely on the quality of the domain model (e.g. how an exploration rover responds to internal commands and external stimuli) to produce useful plans (e.g. to collect an interesting sample and transmit suitable information within daylight). Planning domains are described by means of general-purpose action description languages. For the case of temporal planning, where actions have durations, and temporal constraints are involved, we mention PDDL2.1 (Fox and Long 2003) and the Action Notation Modeling Language (ANML) (Smith, Frank, and Cushing 2008).

Unfortunately, writing planning domains is known to be a hard and error-prone task (Long, Fox, and Howey 2009). This is particularly relevant in temporal planning: the complexity of temporal constraints, continuous time, and issues related to the concurrency, may cause common errors, such as forgetting necessary conditions, or imposing overly restrictive constraints. Similarly, temporal plans are very hard to understand and to inspect manually.

In this paper, we tackle the problem of validation in the setting of Temporal Planning, making the following contributions. First, we formally define a semantics for the temporal fragment of ANML. ANML is equipped with a continuous-time semantics, and it natively offers a very intuitive syntax and support for advanced features such as timed goals, conditions and effects at arbitrary times during actions. Interestingly, ANML is able to capture the temporal fragment (level 3, without instantaneous actions) of PDDL 2.1, for which a translation into ANML exists. Second, we propose an encoding of the behaviors associated to the ANML temporal domain and goal in a corresponding formulation into Linear Temporal Logic (Pnueli 1977) modulo Rational Arithmetic (LTL_{RA}). This is an extension of the classical LTL, where it is possible to reason about rational-valued variables. We use the rational variables from LTL_{RA} to represent time points and temporal expressions from the continuous-time semantics of the ANML language. Third, we formalize as queries in LTL_{RA} several structural properties of planning problems, that are intended to pinpoint the most common flaws in temporal domain descriptions. Queries of interest include checks for action executability and mutual exclusion, plan validation and plan completion. The proposed queries are made practical by leveraging recent, efficient infinite-state verification techniques and tools based on Satisfiability Modulo Theory (Barrett et al. 2009).

Related Work

The difficulty of writing domain descriptions is clearly recognized in the planning community. In fact, there has been a line of research focusing on the development of knowledge-based approaches to support the formalization of planning domains and problems (Simpson, Kitchin, and McCluskey 2007; Jilani et al. 2014). Several Integrated Development Environment tools have been developed, that provide substantial features like domain visualization and plan animation. These works are primarily oriented to classical planning, and provide virtually no support for temporal aspects. Furthermore, the approaches concentrate on supporting the conversion of informal knowledge into domain models. This

line of work is radically different from our approach in two respects: we consider the case of temporal planning, and our focus is on the formal validation of planning models via property checking. We see the two approaches as strongly complementary, and believe that our techniques could be integrated in some of the tools used for model development in order to allow for early debugging.

There are several approaches to validation based on formal methods and temporal logics. Some focus on classical planning, in which actions are instantaneous. Calvanese, De Giacomo, and Vardi (2002) explored an LTL formalization of planning in nondeterministic domains. Other works considering LTL encodings of classical planning are (Mayer et al. 2007) and (Cerrito and Mayer 1998). Raimondi, Pecheur, and Brat (2009) cast some verification queries for classical planning into planning problems. The fundamental difference with respect to our work is that we tackle the case of *temporal* planning, which raises significant issues due to durations and concurrency.

The problem of validation in temporal planning is largely unexplored (see (Bensalem, Havelund, and Orlandini 2014) for a survey). Most works address the problem of *plan* validation, i.e. checking whether a given plan is a solution for a given planning problem. The state-of-the-art is represented by the VAL tool (Howey, Long, and Fox 2004), a fast validation procedure for the full PDDL language, including continuous dynamics and time. The technique is limited to validating complete plans only, due to its simulation approach based on concrete, numerical methods. Our approach also supports other queries (such as domain validation), and can also tackle a more expressive form of plan validation, dealing with incomplete initial conditions and partial plans. In fact, the two approaches can be seen as complementary, and their integration is the objective of future research.

Many formal approaches for validation in the temporal case are based on a reduction to timed automata. Our reduction, based on temporal logic, greatly simplifies the encoding, and allows us to handle a very rich set of constructs and queries in a single framework. The works (Orlandini et al. 2014; Cesta et al. 2010) focus on plan validation, with particular emphasis on the analysis of temporal flexibility (the plan is not represented as a time-triggered set of actions with duration, but as a set of constraints: each model of the constraints yield a plan), without considering the problem of domain validation.

The domain validation problem is tackled in the temporal setting by (Khatib, Muscettola, and Havelund 2000; 2001). These works reduce the HSTS planner language to a Timed Automaton to be verified with the UPPAAL model checker (Behrmann et al. 2006). This work also discusses two structural queries: checking the mutual exclusion of two predicates, and checking the reachability of a predicate (either from a defined initial state, or from all the initial states). Our work deals with a significantly more expressive language, and covers in a single framework a much larger set of queries. In (Penix, Pecheur, and Havelund 1998), abstraction is used to encode some queries over the HSTS language into finite-state model-checking.

Finally, we mention the plan generation approach of (Siminiceanu, Butler, and Muñoz 2008): a temporal planning language is encoded into a symbolic transition system in the SAL (Bensalem et al. 2000) modeling language, and model checking techniques are used for planning. Our focus is not on plan generation but on many forms of validation. Furthermore, thanks to the use of a rich temporal logic, we are able to deal with a richer domain description, and to succinctly specify very expressive properties.

LTL Modulo Rational Arithmetic

We use an extension of the usual Linear Temporal Logic (LTL) (Pnueli 1977) beyond the Boolean case. We allow for the use of variables having domain over the rational numbers and a first-order signature comprising arithmetic constants, the operators $+$ and $\times$ and the relations $=$ and $<$. Moreover, we allow for terms built with a special operator v' where v is a variable. This operator allows to express conditions on the value of the variables in the next state of a trace. We call this logic LTL Modulo Rational Arithmetic (LTL_{RA}). This logic is a fragment of the extended $RELTL$ logic introduced in (Cimatti, Roveri, and Tonetta 2015).

Let $V_{\mathbb{R}}$ be a set of rational variables, and $V_{\mathbb{B}}$ be a set of Boolean variables. A formula λ in LTL_{RA} over $V \doteq V_{\mathbb{R}} \cup V_{\mathbb{B}}$ is defined by the following grammar.

$$\tau \doteq c \mid v_{\mathbb{R}} \mid v'_{\mathbb{R}} \mid \tau_1 + \tau_2 \mid \tau_1 \times \tau_2$$
$$\alpha \doteq v_{\mathbb{B}} \mid \tau_1 = \tau_2 \mid \tau_1 < \tau_2$$
$$\lambda \doteq \alpha \mid \neg\lambda \mid \lambda_1 \wedge \lambda_2 \mid \mathbf{X}\,\lambda \mid \lambda_1\,\mathbf{U}\,\lambda_2$$

Where c is an arithmetic constant, $v_{\mathbb{R}} \in V_{\mathbb{R}}$, and $v_{\mathbb{B}} \in V_{\mathbb{B}}$. We use the following abbreviations: $\top \doteq (1 = 1)$, $\bot \doteq \neg\top$, $\lambda_1 \vee \lambda_2 \doteq \neg(\neg\lambda_1 \wedge \neg\lambda_2)$, $\lambda_1 \rightarrow \lambda_2 \doteq \neg\lambda_1 \vee \lambda_2$, $\tau_1 \leq \tau_2 \doteq \tau_1 < \tau_2 \vee \tau_1 = \tau_2$, $\tau_1 > \tau_2 \doteq \tau_2 < \tau_1$, $\tau_1 \geq \tau_2 \doteq \tau_2 < \tau_1 \vee \tau_1 = \tau_2$, $\mathbf{F}\,\lambda \doteq \top\,\mathbf{U}\,\lambda$, and $\mathbf{G}\,\lambda \doteq \neg\mathbf{F}\,\neg\lambda$. Moreover, given a set of formulae $\Lambda \doteq \{\lambda_1, \cdots, \lambda_n\}$, we write ExactlyOne($\Lambda$) for the formula $(\bigvee_{i=1}^{n} \lambda_i) \wedge (\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \neg(\lambda_i \wedge \lambda_j))$ that holds if and only if exactly one of the $\lambda_i \in \Lambda$ holds.

We interpret terms (τ) and atoms (α) using standard first-order semantics $[\![]\!]$ over the real arithmetic $RA \doteq \langle \mathbb{R}, +, \times, < \rangle$. We slightly abuse the notation by using c for $[\![c]\!]_{RA}$ for any $c \in \mathbb{R}$, $+$ for $[\![+]\!]_{RA}$ and $\times$ for $[\![\times]\!]_{RA}$. An assignment for $V \doteq V_{\mathbb{R}} \cup V_{\mathbb{B}}$ is a function μ mapping each $v_{\mathbb{R}} \in V_{\mathbb{B}}$ to $\mathbb{B}$ and each $v_{\mathbb{R}} \in V_{\mathbb{R}}$ to $\mathbb{R}$. We write $\mu \models_{RA} \alpha$ to indicate that μ satisfies α. We write μ' for the assignment $\mu'(v') = \mu(v)$ for all $v \in V$ that assigns all the next operators applied to variables. We write $\mu \cup \mu'$ to indicate the assignment obtained by the union of functions μ and μ' having disjoint domain. LTL_{RA} is interpreted over discrete traces with rational values. A trace over V is an infinite sequence of assignments $\sigma : \mathbb{N} \rightarrow \Sigma(V)$, where $\Sigma(V)$ indicates all the possible assignments to V. Since we allow for the next operator we interpret the logic on a pair of consecutive time points instead of a single one.

- $\sigma, i \models \alpha$ iff $\sigma(i) \cup \sigma'(i+1) \models_{RA} \alpha$;
- $\sigma, i \models \neg\lambda$ iff $\sigma, i \not\models \lambda$;
- $\sigma, i \models \lambda_1 \wedge \lambda_2$ iff $\sigma, i \models \lambda_1$ and $\sigma, i \models \lambda_2$;
- $\sigma, i \models \mathbf{X}\,\lambda$ iff $\sigma, i+1 \models \lambda$;

- $\sigma, i \models \lambda_1 \mathbf{U} \lambda_2$ iff exists $j \geq i$ s.t. $\sigma, j \models \lambda_2$ and for all $i \leq k < j, \sigma, k \models \lambda_1$.

In general, the satisfiability problem for LTL_{RA} is undecidable. However, infinite-state model-checking techniques can be applied to tackle the satisfiability problem in a sound-but-incomplete way (Cimatti, Roveri, and Tonetta 2015). Such technologies are based on efficient Satisfiability Modulo Theory (SMT) decision procedures and are becoming increasingly efficient (Cimatti et al. 2014; Cavada et al. 2014).

Problem Definition

We now formalize the temporal planning fragment of the ANML language that we consider in this work. We chose to adopt the temporal fragment of ANML for two main reasons. First, we believe that ANML is a good language for hand-modeling complex scenarios thanks to its expressive and simple syntax. Second, the temporal part of the widely used PDDL 2.1 language (that is the fragment in which no instantaneous actions and continuous dynamics are involved) is subsumed by ANML, allowing us to experiment also with PDDL domains[1].

As a running example, we consider the classical Match Cellar domain (Coles et al. 2009) taken from the temporal track of the 2014 International Planning Competition (IPC). The problem consists in fixing a number of fuses with the "MEND_FUSE" action, but in order to mend a fuse, we need light throughout the execution of the mending action. Light can be provided by lighting matches with the "LIGHT_MATCH" action. For our example, we assume to have a single match that can provide light for 5 time units and two fuses, each needing 2 time units to be fixed. The resulting planning problem in the ANML syntax is reported in figure 1.

We now formalize the abstract syntax and the corresponding semantics of a planning problem. Our language is inspired by the ANML language: we allow for rich temporal conditions as well as infinite-state fluents[2].

Definition 1. *Given a finite set of rational fluents $F_{\mathbb{R}}$, a **rational expression** over $F_{\mathbb{R}}$ is:*

$$\rho \doteq c \mid f_{\mathbb{R}} \mid \text{START} \mid \text{DUR} \mid \rho_1 + \rho_2 \mid \rho_1 \times \rho_2$$

where $c \in \mathbb{R}$ and $f_{\mathbb{R}} \in F_{\mathbb{R}}$.

Definition 2. *Given a set of rational fluents $F_{\mathbb{R}}$ and a set of Boolean fluents $F_{\mathbb{B}}$, a **Boolean expression** over $F_{\mathbb{R}} \cup F_{\mathbb{B}}$ is:*

$$\beta \doteq f_{\mathbb{B}} \mid \rho_1 = \rho_2 \mid \rho_1 < \rho_2 \mid \beta_1 \wedge \beta_2 \mid \neg\beta$$

where ρ_1 and ρ_2 rational formulae over $F_{\mathbb{R}}$ and $f_{\mathbb{B}} \in F_{\mathbb{B}}$.

Analogously to the LTL_{RA} case, we define the following rewritings: $\top \doteq (1 = 1)$, $\bot \doteq \neg\top$, $\beta_1 \vee \beta_2 \doteq \neg(\neg\beta_1 \wedge \neg\beta_2)$, $\beta_1 \rightarrow \beta_2 \doteq \neg\beta_1 \vee \beta_2$, $\rho_1 \leq \rho_2 \doteq \rho_1 < \rho_2 \vee \rho_1 = \rho_2$, $\rho_1 > \rho_2 \doteq \rho_2 < \rho_1$, $\rho_1 \geq \rho_2 \doteq \rho_2 < \rho_1 \vee \rho_1 = \rho_2$, and $\text{END} \doteq \text{START} + \text{DUR}$.

[1]The syntactical reduction from the temporal fragment of PDDL 2.1 to ANML can be found online at `https://es.fbk.eu/people/amicheli/resources/aaai17`.

[2]Following ANML terminology, we call "fluent" any variable of the problem, rational or Boolean.

```
1  type fuse;
2  type match;
3
4  fluent boolean handfree;
5  fluent boolean light(match m);
6  fluent boolean mended(fuse f);
7  fluent boolean unused(match m);
8
9  action LIGHT_MATCH(match m) {
10     duration := 5;
11     [start] unused(m);
12     [start] unused(m) := false;
13     [start] light(m) := true;
14     [end] light(m) := false;
15 };
16
17 action MEND_FUSE(fuse f, match m) {
18     duration := 2;
19     [start] handfree; (start, end) light(m);
20     [start] handfree := false;
21     [end] mended(f) := true;
22     [end] handfree := true;
23 };
24
25 instance fuse f1, f2;
26 instance match m1;
27
28 [start] handfree := true;
29 [start] unused(m1) := true;
30 [end] mended(f1) and mended(f2);
```

Figure 1: The Match Cellar running example in ANML.

We say that an expression is constant if it contains no fluents and that an expression is time-independent if it contains no START nor DUR terms.

Definition 3. *Given two rational expressions e_1 and e_2, we define the four possible **intervals**:*

1. $[e_1, e_2]$ *closed;*
2. $(e_1, e_2]$ *left-open;*
3. $[e_1, e_2)$ *right-open;*
4. (e_1, e_2) *open.*

We write $\llbracket e_1, e_2 \rrbracket$ to indicate an instance of the above possibilities without distinguishing the type, similarly $\llbracket e_1, e_2 \rrbracket$ indicates an interval that can be open or close on the left, but close on the right, and so on. Moreover, we write $[e]$ to indicate the single-point interval $[e, e]$.

Definition 4. *A **planning problem** $\mathcal{P} \doteq \langle F, T, A, G \rangle$ is s.t.:*

- *$F = F_{\mathbb{B}} \cup F_{\mathbb{R}}$ is a finite set of Boolean and rational fluents.*
- *T is a finite set of Timed-Initial-Literals (TILs), each of the form $\langle [e_1]\ f := e_2 \rangle$ where $f \in F$, e_1 is a constant, rational expression and e_2 is a constant time-independent expression with the same type as f.*
- *A is a set of durative actions of the form $a \doteq \langle C, E \rangle$ where:*
  - *C is the set of conditions of the form $\langle \llbracket e_1, e_2 \rrbracket\ e_3 \rangle$ with e_1 and e_2 being constant, rational expressions and e_3 being a Boolean expression;*
  - *E is a set of instantaneous effects of the form $\langle [e_1]\ f := e_2 \rangle$ where $f \in F$, e_1 is a constant, rational expression and e_2 is an expression of the same type of f.*
- *G is a set of timed goals each of the form $\langle \llbracket e_1, e_2 \rrbracket\ e_3 \rangle$ with e_1 and e_2 being constant,rational expressions and e_3 being a Boolean expression.*

Intuitively, the language allows the description of temporal planning domains that can modify a finite set of either Boolean or rational valued fluents. The initial state and the

exogenous temporal evolution can be specified by means of TILs: in fact, a TIL at time 0 is an initial condition. Actions can have arbitrary conditions on their duration and we can express conditions having starting or ending times at any point within an action. Note that we do not allow for effects and condition outside the interval in which an action occurs, and this fact is checked by the semantics below. This abstract syntax closely corresponds to the ANML one (Smith, Frank, and Cushing 2008). The only important difference here is that we described a ground language, while ANML allows for lifted specifications (and our example in figure 1 show a use of this feature). In the following, we report the grounded, abstract syntax for the example problem. (Here and in the following, we indicate the fluent `handfree` as hf, `light(m1)` as l_1, `mended(f1)` as m_1, `mended(f2)` as m_2, `unused(m1)` as u_1, the action `LIGHT_MATCH(m1)` as LM_1, `MEND_FUSE(f1)` as MF_1, and `MEND_FUSE(f2)` as MF_2).

$$F \doteq F_{\mathbb{B}} \doteq \{hf, l_1, m_1, m_2, u_1\} \quad G \doteq \{\langle [\text{END}]\ m_1 \wedge m_2\rangle\}$$
$$T \doteq \{\langle [0]\ hf := \top\rangle, \langle [0]\ light(u_1) := \top\rangle, \langle [0]\ l_1 := \bot\rangle, \langle [0]\ m_1 := \bot\rangle, \langle [0]\ m_2 := \bot\rangle\}$$
$$A \doteq \{\langle C_1, E_1\rangle, \langle C_2, E_2\rangle, \langle C_2, E_3\rangle\}$$
$$C_1 \doteq \{\langle [\text{START}]\ \text{DUR} = 5\rangle, \langle [\text{START}]\ u_1\rangle\}$$
$$E_1 \doteq \{\langle [\text{START}]\ u_1 := \bot\rangle, \langle [\text{START}]\ l_1 := \top\rangle, \langle [\text{END}]\ l_1 := \bot\rangle\}$$
$$C_2 \doteq \{\langle [\text{START}]\ \text{DUR} = 2\rangle, \langle [\text{START}]\ hf\rangle, \langle (\text{START}, \text{END})\ l_1\rangle\}$$
$$E_2 \doteq \{\langle [\text{START}]\ hf := \bot\rangle, \langle [\text{END}]\ hf := \top\rangle, \langle [\text{END}]\ m_1 := \top\rangle\}$$
$$E_3 \doteq \{\langle [\text{START}]\ hf := \bot\rangle, \langle [\text{END}]\ hf := \top\rangle, \langle [\text{END}]\ m_2 := \top\rangle\}$$

For the sake of this paper, we focus on time-triggered plans, namely plans comprising a finite number of actions to be executed at specified times each with specified duration.

Definition 5. *A **time triggered plan** π for $\mathcal{P}$ is a sequence $\langle\langle s_1, a_1, d_1\rangle, \langle s_2, a_2, d_2\rangle, \cdots, \langle s_n, a_n, d_n\rangle\rangle$, where $s_i \in \mathbb{R}^+$, $a_i \in A$, $d_i \in \mathbb{R}^+$ and $s_i \leq s_{i+1}$.*

We give the semantics of the planning language by defining the validity of a plan π for a given problem $\mathcal{P}$. As usual, we say that $\mathcal{P}$ admits a solution if there exists a valid plan, otherwise the problem is said to be unsolvable.

The basic element of our semantics is a chronicle, that is used to assign a value to each fluent in F for each time instant $x \geq 0 \in \mathbb{R}^+$.

Definition 6. *A **chronicle** τ for a given problem instance $\mathcal{P} \doteq \langle F, T, A, G\rangle$ is a set of functions $\{\tau_f \mid f \in F\}$. Each τ_f maps a positive rational into a value for f.*

Definition 7. *The **value of an expression** e in a chronicle τ at a time $t \in \mathbb{R}^+$ in a context starting at time $s \in \mathbb{R}^+$ with duration $d \in \mathbb{R}^+$ (written $[\![e]\!]^\tau_{s,d}(t)$) is defined as follows:*

1. $[\![c]\!]^\tau_{s,d}(t) \doteq c$, *with* $c \in \mathbb{R}$;
2. $[\![f]\!]^\tau_{s,d}(t) \doteq \tau_f(t)$;
3. $[\![\text{START}]\!]^\tau_{s,d}(t) \doteq s$;
4. $[\![\text{DUR}]\!]^\tau_{s,d}(t) \doteq d$;
5. $[\![e_1 + e_2]\!]^\tau_{s,d}(t) \doteq [\![e_1]\!]^\tau_{s,d}(t) + [\![e_2]\!]^\tau_{s,d}(t)$;
6. $[\![e_1 \times e_2]\!]^\tau_{s,d}(t) \doteq [\![e_1]\!]^\tau_{s,d}(t) \times [\![e_2]\!]^\tau_{s,d}(t)$;
7. $[\![e_1 = e_2]\!]^\tau_{s,d}(t) \doteq \top$ *iff* $[\![e_1]\!]^\tau_{s,d}(t) = [\![e_2]\!]^\tau_{s,d}(t)$;
8. $[\![e_1 < e_2]\!]^\tau_{s,d}(t) \doteq \top$ *iff* $[\![e_1]\!]^\tau_{s,d}(t) < [\![e_2]\!]^\tau_{s,d}(t)$;
9. $[\![e_1 \wedge e_2]\!]^\tau_{s,d}(t) \doteq \top$ *iff* $[\![e_1]\!]^\tau_{s,d}(t)$ *and* $[\![e_2]\!]^\tau_{s,d}(t)$;
10. $[\![\neg e]\!]^\tau_{s,d}(t) \doteq \top$ *iff* $[\![e]\!]^\tau_{s,d}(t)$ *is false.*

For constant expressions we write $[\![e]\!]_{s,d}$ (τ and t are not needed). For constant, time independent expressions we write $[\![e]\!]$.

Definition 8. *Given a plan π we define the **make-span** ms_π as $max(\{t + d \mid \langle t, a, d\rangle \in \pi\})$.*

Given a plan, we can now define the chronicle induced by it. In our language we have two components that contribute to change the state of a fluent, namely the TILs and the action effects. Apart for these events, each fluent is assumed to maintain its value in the other time instants. To formalize this concept we start by collecting the set of change events (with the temporal extremes of the intervals they appear in) in the execution of the plan.

Definition 9. *Given a planning problem $\mathcal{P} \doteq \langle F, T, A, G\rangle$ and a plan $\pi \doteq \{\langle t_i, a_i, d_i\rangle \mid i \in [1, n]\}$, the **set of changes** induced by π is a multi-set $CH(\mathcal{P}, \pi)$ s.t.:*

- *for all $\langle [e_1]\ f := e_2\rangle \in T$, $\langle [\![e_1]\!]_{0,ms_\pi}, f, e_2, 0, ms_\pi\rangle \in CH(\mathcal{P}, \pi)$;*
- *for all $\langle t, a, d\rangle \in \pi$ with $a \doteq \langle C, E\rangle$, $\forall \langle [e_1]\ f := e_2\rangle \in E$, $\langle [\![e_1]\!]_{t,d}, f, e_2, t, d\rangle \in CH(\mathcal{P}, \pi)$ if $t \leq [\![e_1]\!]_{t,d} \leq t + d$.*

The set of changes is defined as a multi-set because it is possible for two identical effects to happen at the same time. This will be a reason to declare a plan not-valid in Definition 12, but here we have to take this possibility into account. Note the last condition: we are requiring that each effect of each action happens within the action itself. In this way, an hypothetical effect $\langle [\text{START} + 6]\ hf := \top\rangle$ in the action LM_1 of the running example would make the problem badly-defined. This also implies that each change $\langle t, f, v, t_0, d_0\rangle$ is such that $t \geq t_0$. Now we can define the chronicle induced by a plan by imposing that at each change point the chronicle changes its value and between any two changes, the chronicle maintains the "older" value.

Definition 10. *Given a planning problem $\mathcal{P}$, the **chronicle** τ^π **induced by a plan** π is s.t. for each $x \in \mathbb{R}^+$ and each $\langle t_s, f, v, t_0, d_0\rangle \in CH(\mathcal{P}, \pi)$ s.t $t_s < x \leq min(\{t_e \mid \langle t_e, f, v', t', d'\rangle \in CH(\mathcal{P}, \pi) \wedge t_e > t_s\})$, $\tau^\pi_f(x) = [\![v]\!]^\tau_{t_0,d_0}(t_0)$.*

Intuitively, each fluent in each time point assumes the value imposed by the last change until another effect is applied. Note the strict inequality in the definition: we impose the value of an effect immediately after the change is scheduled to happen. Also, there is no minimum time quantum (as in PDDL 2.1): we only require a positive amount of time between an effect and a condition requiring it.

As an auxiliary definition, we introduce the absolute-time interval of a condition of an action appearing in a plan. The idea is to define the subset of the time points in which each condition is required to hold given a plan.

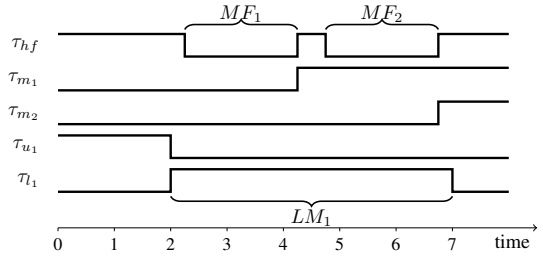

Figure 2: Evolution of the running example on plan π_{ex}.

Definition 11. *The **absolute-time interval** $\Omega_\tau(c, s, d)$ of a condition c in a context starting at s with duration d is:*

$$\Omega_\tau(c,s,d) \doteq \begin{cases} [[\![e_1]\!]^\tau_{s,d}(s), [\![e_2]\!]^\tau_{s,d}(s)] & \text{if } c = \langle [e_1, e_2]\ \phi \rangle \\ ([\![e_1]\!]^\tau_{s,d}(s), [\![e_2]\!]^\tau_{s,d}(s)] & \text{if } c = \langle (e_1, e_2]\ \phi \rangle \\ [[\![e_1]\!]^\tau_{s,d}(s), [\![e_2]\!]^\tau_{s,d}(s)) & \text{if } c = \langle [e_1, e_2)\ \phi \rangle \\ ([\![e_1]\!]^\tau_{s,d}(s), [\![e_2]\!]^\tau_{s,d}(s)) & \text{if } c = \langle (e_1, e_2)\ \phi \rangle \end{cases}$$

We now define the validity of a plan for a problem: it suffices to check that no pair of changes are applied in the same instant (1), that each action condition holds in its absolute-time interval (2), and that each goal is also satisfied (3).

Definition 12. *Given a problem $\mathcal{P} \doteq \langle F, T, A, G \rangle$, a **plan** π **is valid**, if the chronicle τ^π is s.t:*

1. *for each $t \in \mathbb{R}$ and each $f \in F$, $|\langle [\![v]\!]^{\tau^\pi}_{t_0,d_0}(t) \mid \langle t, f, v, t_0, d_0 \rangle \in CH(\mathcal{P}, \pi)\rangle| \leq 1$;*
2. *for each $\langle t, a, d \rangle \in \pi$ with $a \doteq \langle C, E \rangle$, and for each $c \in C$ (with $c = \langle [\![e_1, e_2]\!]\ e_3 \rangle$) and each $x \in \Omega_{\tau^\pi}(c, t, d)$, $t \leq x \leq t + d$ and $[\![e_3]\!]^{\tau^\pi}_{t,d}(x)$ holds;*
3. *for each goal condition $c = \langle [\![e_1, e_2]\!]\ e_3 \rangle \in G$, $[\![e_3]\!]^{\tau^\pi}_{0,ms_\pi}(x)$ holds for any $x \in \Omega_{\tau^\pi}(c, 0, ms_\pi)$.*

Note that we are requiring that each condition of each action happens during the action itself. Figure 2 shows the chronicle for the running example problem on the following (valid) plan:

$$\pi_{ex} \doteq \{\langle 2, LM(1), 5\rangle, \langle 2.25, MF(1), 2\rangle, \langle 4.75, MF(2), 2\rangle\}.$$

Encoding in LTL_{RA}

Given a planning problem $\mathcal{P} \doteq \langle F = F_{\mathbb{B}} \cup F_{\mathbb{R}}, T, A, G \rangle$, we now describe how to encode $\mathcal{P}$ into an LTL_{RA} formula such that the models of such formula encode the executions of $\mathcal{P}$ for any valid plan. First, we define the variables of the encoding.

Definition 13. *The set of **variables appearing in the encoding** of $\mathcal{P}$ is $V^\mathcal{P} \doteq V^\mathcal{P}_\mathbb{B} \cup V^\mathcal{P}_\mathbb{R}$ where:*

- $V^\mathcal{P}_\mathbb{B} \doteq \{\overline{a} \mid a \in A\} \cup \{\overline{f} \mid f \in F_\mathbb{B}\}$;
- $V^\mathcal{P}_\mathbb{R} \doteq \{t, \omega\} \cup \{\delta_a, s_a \mid a \in A\} \cup \{\overline{f} \mid f \in F_\mathbb{R}\}$.

Intuitively, t will record the current absolute time in the execution, ω will mark the time at which the plan is terminated (i.e. the make-span), each $\overline{f}$ with $f \in F$ records the value of the corresponding fluent and $\overline{a}$, δ_a and s_a are used to encode the execution of the action $a \in A$.

The intuition behind the encoding is as follows. We abstract the chronicles in the semantics of the planning language by representing only the discrete points in which an effect happens (this amounts to encode the set of changes in Definition 9). This constitutes a trace that is accepted by our LTL_{RA} encoding, while all invalid traces are rejected. Using the variables introduced above, we can constrain the behaviors according to the planning problem semantics. For example, the evolution depicted in figure 2 corresponds to the trace σ_{ex} defined as follows. (To save space, we only report the Boolean variables set to $\top$ and the rational variables set to a value different from 0.)

$$\begin{aligned}
\sigma_{ex}(0) &\doteq \{\overline{hf}, \overline{u_1}, \omega = 8\} \\
\sigma_{ex}(1) &\doteq \{\overline{hf}, \overline{l_1}, \overline{LM_1}, s_{LM_1} = 2, \delta_{LM_1} = 5, t = 2, \omega = 8\} \\
\sigma_{ex}(2) &\doteq \{\overline{l_1}, \overline{MF_1 LM_1}, s_{MF_1} = 2.25, s_{LM_1} = 2, \delta_{MF_1} = 2, \\
&\qquad \delta_{LM_1} = 5, t = 2.25, \omega = 8\} \\
\sigma_{ex}(3) &\doteq \{\overline{hf}, \overline{m_1}, \overline{l_1}, \overline{MF_1}, \overline{LM_1}, s_{LM_1} = 2, \delta_{LM_1} = 5, t = 4.25, \omega = 8\} \\
\sigma_{ex}(4) &\doteq \{\overline{m_1}, \overline{l_1}, \overline{MF_2}, \overline{LM_1}, s_{MF_2} = 4.75, s_{LM_1} = 2, \delta_{MF_2} = 2, \\
&\qquad \delta_{LM_1} = 5, t = 4.75, \omega = 8\} \\
\sigma_{ex}(5) &\doteq \{\overline{hf}, \overline{m_1}, \overline{m_2}, \overline{l_1}, \overline{MF_2}, \overline{LM_1}, s_{LM_1} = 2, \delta_{LM_1} = 5, \\
&\qquad t = 6.75, \omega = 8\} \\
\sigma_{ex}(6) &\doteq \{\overline{hf}, \overline{m_1}, \overline{m_2}, \overline{LM_1}, t = 7, \omega = 8\} \\
\sigma_{ex}(7) &\doteq \{\overline{hf}, \overline{m_1}, \overline{m_2}, t = 7.5, \omega = 8\} \\
\sigma_{ex}(i) &\doteq \{\overline{hf}, \overline{m_1}, \overline{m_2}, t = 8, \omega = 8\} \text{for each } i \geq 8
\end{aligned}$$

We now introduce the encoding of expressions into LTL_{RA}. Intuitively, an expression in the planning language is transposed in LTL_{RA} by keeping its operators, but interpreting the references to fluents as variables of the formula and by interpreting the START and DUR temporal references depending on the context of the expression.

Definition 14. *Let e be any expression in the planning problem $\mathcal{P}$, and let $a \in A \cup \{\emptyset\}$, the **encoding of** e in the context of a (written $(\!|e|\!)_a$) is:*

1. $(\!|c|\!)_a \doteq c$, *with* $c \in \mathbb{R}$;
2. $(\!|f|\!)_a \doteq \overline{f}$, *with* $f \in F$;
3. $(\!|\text{START}|\!)_\emptyset \doteq 0$;
4. $(\!|\text{START}|\!)_a \doteq s_a$ *with* $a \in A$;
5. $(\!|\text{DUR}|\!)_\emptyset \doteq \omega$;
6. $(\!|\text{DUR}|\!)_a \doteq \delta_a$ *with* $a \in A$;
7. $(\!|e_1 + e_2|\!)_a \doteq (\!|e_1|\!)_a + (\!|e_2|\!)_a$;
8. $(\!|e_1 \times e_2|\!)_a \doteq (\!|e_1|\!)_a \times (\!|e_2|\!)_a$;
9. $(\!|e_1 = e_2|\!)_a \doteq (\!|e_1|\!)_a = (\!|e_2|\!)_a$;
10. $(\!|e_1 < e_2|\!)_a \doteq (\!|e_1|\!)_a < (\!|e_2|\!)_a$;
11. $(\!|e_1 \wedge e_2|\!)_a \doteq (\!|e_1|\!)_a \wedge (\!|e_2|\!)_a$;
12. $(\!|\neg e|\!)_a \doteq \neg (\!|e|\!)_a$.

We can now define the domain encoding $\epsilon^d_\mathcal{P}$ of $\mathcal{P}$ as the conjunction of the following LTL_{RA} formulae.

1. $t = 0$; i.e. the initial time is 0.
2. $\mathbf{G}\ (\omega = \omega')$; i.e. the time horizon never changes.

3. $\mathbf{G}\,((t < \omega) \rightarrow (t < t'))$; i.e. the time always increases before the time horizon.
4. $\mathbf{G}\,((t \geq \omega) \rightarrow ((\bigwedge_{f \in V^{\mathcal{P}}} \overline{f} = \overline{f}') \wedge (\bigwedge_{a \in A} \neg \overline{a}) \wedge (t = t'))$; i.e. after the time horizon, all the variables keep their values and no action is executing.
5. $\bigwedge_{a \in A} \mathbf{G}\,(\overline{a} \rightarrow (\delta'_a = \delta_a \wedge s'_a = s_a))$; i.e. the δ_a and s_a variables are kept during an action execution.
6. $\bigwedge_{a \in A} \mathbf{G}\,((\neg \overline{a} \wedge \mathbf{X}\,\overline{a}) \rightarrow \mathbf{X}\,(\overline{a}\,\mathbf{U}\,(\mathbf{X} \neg \overline{a} \wedge t = \delta_a + s_a)))$; i.e. a terminates exactly at time $t = \delta_a + s_a$.
7. $\mathbf{G}\,((\neg \overline{a} \wedge \mathbf{X}\,\overline{a}) \rightarrow \mathbf{X}\,(\overline{a}\,\mathbf{U}\,(t = \langle\!| e_1 |\!\rangle_a \wedge \overline{f} = \langle\!| e_2 |\!\rangle_a)))$ for each effect $[e_1]f := e_2$ of each action a; i.e. each effect changes the value of a fluent in a step that is forced to happen at the time specified by e_1.
8. $\mathbf{G}\,(((\mathbf{X}\,\overline{a} \wedge t < \langle\!| e_1 |\!\rangle'_a \wedge \mathbf{X}\,(t \geq \langle\!| e_1 |\!\rangle_a)) \rightarrow \langle\!| e_3 |\!\rangle_a) \wedge ((\overline{a} \wedge t \geq \langle\!| e_1 |\!\rangle_a \wedge t < \langle\!| e_2 |\!\rangle_a) \rightarrow \langle\!| e_3 |\!\rangle_a))$ for each condition $\langle [e_1, e_2] \, e_3 \rangle$ of each action a; i.e. each durative condition closed on the left is achieved before the time specified by e_1 and kept during the whole interval.
9. $\mathbf{G}\,(((\mathbf{X}\,\overline{a} \wedge t \leq \langle\!| e_1 |\!\rangle'_a \wedge \mathbf{X}\,(t > \langle\!| e_1 |\!\rangle_a)) \rightarrow \langle\!| e_3 |\!\rangle_a) \wedge ((\overline{a} \wedge t > \langle\!| e_1 |\!\rangle_a \wedge t < \langle\!| e_2 |\!\rangle_a) \rightarrow \langle\!| e_3 |\!\rangle_a))$ for each condition $\langle (e_1, e_2] \, e_3 \rangle$ of each action a; i.e. each durative condition open on the left is achieved before or at the time specified by e_1 and kept during the whole interval.

This is the "domain" part of the encoding: we are defining the evolution of time (conjuncts 1 to 3), then we impose a strictly looping trace in which the system stops moving after the horizon that is fixed at ω (constraint 4). We encode each action with an associated Boolean variable that can be set to $\top$ when the action starts and is kept $\top$ during the action execution. Conjunct 5 imposes that the values of the starting time s_a and of the duration δ_a are kept during the execution of the action. In this way, the system is free to choose suitable values for the duration and the starting time (before the action starts), but those cannot be changed during an action execution. Then, we impose (constraint 6) that the action terminates exactly at time $s_a + \delta_a$. Finally, constraint 7 imposes that there must be a step in which each effect of the action is realized, while constraints 8 and 9 ensure that all the action conditions are satisfied. Note that, we do not need to encode dedicated steps for the conditions bounds because we check the value of each fluent before the starting of the condition and during it, this is enough for the condition to be satisfied.

We define the problem encoding of $\mathcal{P}$ as the LTL_{RA} formula $\epsilon^p_{\mathcal{P}} \doteq \tau_{\mathcal{P}} \wedge \gamma_{\mathcal{P}}$; where:
$\tau_{\mathcal{P}} \doteq \bigwedge_{\langle [e_1]\, f := e_2 \rangle \in T} \omega > \langle\!| e_1 |\!\rangle_\emptyset \wedge \mathbf{F}\,(t = \langle\!| e_1 |\!\rangle_\emptyset \wedge \overline{f} = \langle\!| e_2 |\!\rangle_\emptyset)$ and $\gamma_{\mathcal{P}}$ is the conjunction of the following formulae:

1. $\bigwedge_{\langle e_1, e_2 \rangle e_3 \in G} \omega \geq \langle\!| e_2 |\!\rangle_\emptyset$;
2. $\mathbf{G}(((t < \langle\!| e_1 |\!\rangle_\emptyset \wedge \mathbf{X}(t \geq \langle\!| e_1 |\!\rangle_\emptyset)) \rightarrow \langle\!| e_3 |\!\rangle_\emptyset) \wedge ((t \geq \langle\!| e_1 |\!\rangle_\emptyset \wedge t < \langle\!| e_2 |\!\rangle_\emptyset) \rightarrow \langle\!| e_3 |\!\rangle_\emptyset) \wedge ((t = \langle\!| e_1 |\!\rangle_\emptyset \wedge t = 0) \rightarrow \langle\!| e_3 |\!\rangle_\emptyset))$ for each goal $\langle e_1, e_2 \rangle \, e_3 \rangle$;
3. $\bigwedge_{(e_1, e_2) e_3 \in G} \mathbf{G}\,(((t \leq \langle\!| e_1 |\!\rangle_\emptyset \wedge \mathbf{X}\,(t > \langle\!| e_1 |\!\rangle_\emptyset)) \rightarrow \langle\!| e_3 |\!\rangle_\emptyset) \wedge ((t > \langle\!| e_1 |\!\rangle_\emptyset \wedge t < \langle\!| e_2 |\!\rangle_\emptyset) \rightarrow \langle\!| e_3 |\!\rangle_\emptyset))$ for each goal $\langle [e_1, e_2] \, e_3 \rangle$.

Intuitively, $\tau_{\mathcal{P}}$ encodes the TILs of the problem by forcing that we will eventually have a step corresponding to the time of the TILs in which the corresponding effect is applied (this is similar to an action effect). The formula $\gamma_{\mathcal{P}}$ requires all the timed goals to be reached: the LTL_{RA} encoding is analogous to the condition of actions, but here we allow for goals to be specified starting from time 0 and we interpret the START and DUR time references relatively to time 0 and the make-span of the execution, respectively.

Finally, we define the frame condition. In fact, we must enforce that each change in the value of each fluent is motivated by an effect, otherwise it would be possible for a fluent to change value without a cause. For the subsequent analyses, we define two different frame condition formulae, one that only considers the domain actions ($\phi^d_{\mathcal{P}}$) and one that considers the full problem ($\phi^p_{\mathcal{P}}$).
$\phi^d_{\mathcal{P}} \doteq \bigwedge_{f \in V^{\mathcal{P}}} \mathbf{G}\,((\overline{f} \neq \overline{f}') \rightarrow \textsc{ExactlyOne}(Acts))$
$\phi^p_{\mathcal{P}} \doteq \bigwedge_{f \in V^{\mathcal{P}}} \mathbf{G}\,((\overline{f} \neq \overline{f}') \rightarrow \textsc{ExactlyOne}(Acts \cup Tils))$
where $Acts \doteq \{\mathbf{X}\,(\overline{a} \wedge (\langle\!| e_1 |\!\rangle_a = t) \mid [e_1]f := e_2 \in E, a = \langle C, E \rangle \in A\}$ and $Tils \doteq \{\mathbf{X}(\langle\!| e_1 |\!\rangle_\emptyset = t) \mid [e_1]f := e_2 \in \mathcal{T}\}$.

The difference between these two formulations is only in the consideration of TILs: $\phi^d_{\mathcal{P}}$ does not consider them because we want to encode only the domain, without the initial state and the timed effects, while $\phi^p_{\mathcal{P}}$ fully considers them.

The correctness of the domain-validation queries is based on the following theorem[3] stating that regardless of the initial state and the goal, the $\epsilon^d_{\mathcal{P}}$ formula captures any possible evolution of the planning problem.

Theorem 1. *Let $\mathcal{P}$ be a planning problem admitting a solution, then $\epsilon^d_{\mathcal{P}}$ is satisfiable.*

Note that, the converse is not true: depending on the initial state and the goals, it is possible that $\mathcal{P}$ is unsolvable while $\epsilon^d_{\mathcal{P}}$ is satisfiable. The correctness of the problem- and plan-validation queries relays on the following, stronger property.

Theorem 2. *$\epsilon^d_{\mathcal{P}} \wedge \epsilon^p_{\mathcal{P}} \wedge \phi^p_{\mathcal{P}}$ is satisfiable if and only if the planning problem $\mathcal{P}$ admits a solution.*

The encoding does not bound the number of time points, nor it requires the specification of an horizon (the horizon exists for every execution of a plan, but the encoding decides the horizon by setting the ω variable). However, the encoding disallows self-concurrency: we have a Boolean variable dedicated to each action that is kept true in the time points in which the action is executing and this prevents another instance of the same action to be executed concurrently. This is a common limitation and a possible solution is to bound the number of self-concurrent actions and instantiate the encoding on a problem with multiple copies of such actions.

Validation Queries

In this section, we show how to exploit our LTL_{RA} encoding to perform several validation queries. The user can clearly specify any LTL_{RA} property of interest and check its satisfiability or validity, but here we indicate how to encode some common properties of interest. We believe that

[3] All the theorem proofs are available at `https://es.fbk.eu/people/amicheli/resources/aaai17`.

this could be a reasonable set of checks that a modeling tool could perform automatically to help the domain-expert during the development of domains and problems.

Action Executability. One common modeling error that can arise in practice is the non-applicability of an action a due to non-realizable conditions. This can be easily checked in our framework by checking the property $\epsilon^d_{\mathcal{P}} \wedge \phi^d_{\mathcal{P}} \wedge \mathbf{F}\ \overline{a}$. If the property is unsatisfiable, it means that regardless of the initial state or the goal, the action a cannot be executed. Otherwise, any satisfying trace shows a path that eventually starts the action a. Note that, this formulation does not consider TILs: due to the definition of $\phi^d_{\mathcal{P}}$, we are only giving the freedom to pick an arbitrary initial state, not to change the values of fluents during the execution without an action effect. If TILs have to be considered, we can use the $\phi^p_{\mathcal{P}}$ formulation in place of $\phi^d_{\mathcal{P}}$ by filtering only the TILs of interest.

This check is important if an action has an effect or a condition that happens outside of the action. In fact the encoding forbids those, making the action non-executable (i.e. the corresponding Boolean variable cannot be set to $\top$).

Action Mutex. A second domain validation query is to check if two given actions are mutually exclusive. Given two actions a_1 and a_2, we can prove that the actions are mutually exclusive in a given planning problem $\mathcal{P}$ by checking the validity of $\epsilon^d_{\mathcal{P}} \wedge \phi^d_{\mathcal{P}} \rightarrow \neg\mathbf{F}\ (\overline{a_1} \wedge \overline{a_2})$. Intuitively, we check that for any execution respecting the domain constraints, it is impossible to reach a state in which both a_1 and a_2 are executing simultaneously.

Plan Generation. The plan generation problem can be reshaped as an LTL_{RA} satisfiability check exploiting Theorem 2. If $\epsilon^d_{\mathcal{P}} \wedge \epsilon^p_{\mathcal{P}} \wedge \phi^p_{\mathcal{P}}$ is satisfiable, then the planning problem is solvable and any satisfying trace yields a plan. In order to extract a plan from the trace, we can simply search for the states in which an action a is set to $\top$ being previously $\bot$: the starting time of the action will be the value of s_a and the prescribed duration will be the value of δ_a.

We highlight that, the performance of current model-checkers in finding temporal plans is not comparable to heuristic-based temporal planners, but this query is still very useful for at least two purposes. First, differently from most planners, if the problem admits no plan, the model checker is able to eventually terminate proving that no plan exist. Second, we can use this query to validate a planning algorithm using any model-checker for LTL_{RA}.

Plan Completion. We can check whether a given plan $\pi \doteq \{\langle s_1, a_1, d_1\rangle, \cdots \langle s_n, a_n, d_n\rangle\}$ can be extended to a valid plan by checking the satisfiability of the LTL_{RA} formula $\epsilon^d_{\mathcal{P}} \wedge \epsilon^p_{\mathcal{P}} \wedge \phi^p_{\mathcal{P}} \wedge \chi_\pi$, where χ_π is defined as the formula $\bigwedge_{\langle s,a,d\rangle \in \pi} \mathbf{F}\ (s_a = s \wedge \overline{a} \wedge \delta_a = d)$. In particular, the formula is unsatisfiable if and only if π is cannot be extended to a solution plan for $\mathcal{P}$, otherwise any satisfying trace for the formula witnesses an execution in which at least all the action prescribed by π are executed, and possibly other actions are used to complete the plan. We can construct the complete plan analogously to the previous query.

Plan Validation. We can reduce the validity checking of a given a plan $\pi \doteq \{\langle s_1, a_1, d_1\rangle, \cdots \langle s_n, a_n, d_n\rangle\}$, to an LTL_{RA} satisfiability problem by considering $\epsilon^d_{\mathcal{P}} \wedge \epsilon^p_{\mathcal{P}} \wedge \phi^p_{\mathcal{P}} \wedge \chi_\pi \wedge \xi_\pi$ where, $\xi_\pi \doteq \bigwedge_{a\in A} \mathbf{G}(\overline{a} \rightarrow \bigvee_{\langle s,a,d\rangle\in\pi}(s_a = s \wedge \delta_a = d))$. If the formula is satisfiable, then the plan is valid and each satisfying trace yields a witness of the plan execution, otherwise the plan is invalid. The intuition is that we are performing a plan completion check with the additional constraints that no actions can be added by the planner.

Domain specific. Apart for the aforementioned structural queries, the user can also specify domain specific properties directly in LTL_{RA}, and check them against the encoding, following an approach similar to (Cimatti et al. 2012).

For example, it is possible to check that a particular value v for a fluent f needs to be eventually achieved in any valid plan (i.e. $f = v$ is a landmark) by checking the formula: $\epsilon^d_{\mathcal{P}} \wedge \epsilon^p_{\mathcal{P}} \wedge \phi^p_{\mathcal{P}} \rightarrow \mathbf{F}\ (\overline{f} = v)$.

In addition, it is also possible to inspect traces satisfying a property, leveraging symbolic simulation provided by some model-checkers.

Discussion and Conclusions

In this paper we tackled the problem of validation for temporal planning. We presented a semantic characterization of the temporal fragment of the ANML language, a provably correct encoding into LTL_{RA}, and we defined a number of plan and domain validation queries.

The approach is made practical by leveraging efficient, modern infinite-state model-checkers. We implemented a prototype of the encoder and experimented with several ANML and PDDL problems. We use the NUXMV (Cavada et al. 2014) model-checker to check the satisfiability (or the validity) of the produced formulae. Our implementation is available at `https://es.fbk.eu/people/amicheli/resources/aaai17`. The tool is able to quickly handle all the ANML handcrafted domains we tested on the various queries, and to prove action executability for all the domains in the 2014 IPC (Vallati et al. 2015). The current approach falls short when confronted with planning problems over real-sized domains from the IPC. This is to be expected, due to the huge size of the problems resulting from grounding, and to the lack of heuristics, that reduces planning based on LTL_{RA} encoding to an iterative-deepening search.

There are several directions for future work. The first one is to optimize our encoder and empirically evaluate it on several benchmark problems to provide a coherent tool that automatizes the interaction between the encoder and the model-checker. Many optimizations are possible: for example, the duration is often a constant, and need not be encoded as a variable; furthermore, the initial state (i.e. the TILs specified at time 0) can be enforced in LTL_{RA} instead of treating it as a general-case TIL. We remark, however, that the primary purpose of the proposed techniques is not plan generation, but rather to support the elimination of modeling flaws in temporal domains.

Other extensions include the modeling of continuous change and resources, based on the use of a more expressive logic over hybrid traces such as $HRELTL$ (Cimatti, Roveri, and Tonetta 2015). Finally, a fundamental direction is the analysis of domains including actions with uncontrollable durations and non-deterministic effects.

References

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability modulo theories. In *Handbook of Satisfiability*. IOS Press. 825–885.

Behrmann, G.; David, A.; Larsen, K. G.; Hakansson, J.; Petterson, P.; Yi, W.; and Hendriks, M. 2006. Uppaal 4.0. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems*, QEST '06, 125–126.

Bensalem, S.; Ganesh, V.; Lakhnech, Y.; Muoz, C.; Owre, S.; Ruess, H.; Rushby, J.; Rusu, V.; Saidi, H.; Shankar, N.; Singerman, E.; and Tiwari, A. 2000. An overview of SAL. In *Fifth NASA Langley Formal Methods Workshop*, 187–196.

Bensalem, S.; Havelund, K.; and Orlandini, A. 2014. Verification and validation meet planning and scheduling. *STTT* 16(1):1–12.

Calvanese, D.; De Giacomo, G.; and Vardi, M. Y. 2002. Reasoning about actions and planning in LTL action theories. In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, 593–602.

Cavada, R.; Cimatti, A.; Dorigatti, M.; Griggio, A.; Mariotti, A.; Micheli, A.; Mover, S.; Roveri, M.; and Tonetta, S. 2014. The nuxmv symbolic model checker. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, 334–342.

Cerrito, S., and Mayer, M. C. 1998. Using linear temporal logic to model and solve planning problems. In *AIMSA*, 141–152.

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2010. Validation and verification issues in a timeline-based planning system. *Knowledge Eng. Review* 25(3):299–318.

Cimatti, A.; Roveri, M.; Susi, A.; and Tonetta, S. 2012. Validation of requirements for hybrid systems: A formal approach. *ACM Trans. Softw. Eng. Methodol.* 21(4):22.

Cimatti, A.; Griggio, A.; Mover, S.; and Tonetta, S. 2014. Verifying LTL properties of hybrid systems with k-liveness. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, 424–440.

Cimatti, A.; Roveri, M.; and Tonetta, S. 2015. HRELTL: A temporal logic for hybrid systems. *Inf. Comput.* 245:54–71.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artif. Intell.* 173(1):1–44.

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

Howey, R.; Long, D.; and Fox, M. 2004. VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), 15-17 November 2004, Boca Raton, FL, USA*, 294–301.

Jilani, R.; Crampton, A.; Kitchin, D. E.; and Vallati, M. 2014. Automated knowledge engineering tools in planning: State-of-the-art and future challenges. In *Knowledge Engineering for Planning and Scheduling (KEPS) - part of ICAPS 2014*.

Khatib, L.; Muscettola, N.; and Havelund, K. 2000. Verification of plan models using UPPAAL. In *Formal Approaches to Agent-Based Systems, First International Workshop, FAABS 2000 Greenbelt, MD, USA, April 5-7, 2000, Revised Papers*, 114–122.

Khatib, L.; Muscettola, N.; and Havelund, K. 2001. Mapping temporal planning constraints into timed automata. In *Eigth International Symposium on Temporal Representation and Reasoning, TIME-01, Civdale del Friuli, Italy, June 14-16, 2001*, 21–27.

Long, D.; Fox, M.; and Howey, R. 2009. Planning domains and plans: validation, verification and analysis. In *Verification and Validation of Planning and Scheduling Systems Workshop*.

Mayer, M. C.; Limongelli, C.; Orlandini, A.; and Poggioni, V. 2007. Linear temporal logic as an executable semantics for planning languages. *Journal of Logic, Language and Information* 16(1):63–89.

Orlandini, A.; Bernardi, G.; Cesta, A.; and Finzi, A. 2014. Planning meets verification and validation in a knowledge engineering environment. *Intelligenza Artificiale* 8(1):87–100.

Penix, J.; Pecheur, C.; and Havelund, K. 1998. Using model checking to validate ai planner domain models. In *In the Proceedings of the 23rd Annual Software Engineering Workshop*.

Pnueli, A. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, 46–57.

Raimondi, F.; Pecheur, C.; and Brat, G. 2009. Pdver, a tool to verify pddl planning domains. In *Verification and Validation of Planning and Scheduling Systems Workshop*.

Siminiceanu, R.; Butler, R. W.; and Muñoz, C. A. 2008. Experimental evaluation of a planning language suitable for formal verification. In *Model Checking and Artificial Intelligence, 5th International Workshop, MoChArt 2008, Patras, Greece, July 21, 2008. Revised Selected and Invited Papers*, 132–146.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowledge Eng. Review* 22(2):117–134.

Smith, D.; Frank, J.; and Cushing, W. 2008. The ANML language. In *ICAPS Poster session*.

Vallati, M.; Chrpa, L.; Grzes, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine* 36(3):90–98.