SMT Encoding of Planning for Hybrid Systems

Michael Cashmore Daniele Magazzeni Parisa Zehtabi
Kings College London, London, WC2R 2LS
firstname.lastname@kcl.ac.uk

Abstract

Planning in hybrid systems is important for dealing with real-world applications. PDDL+ supports this representation of domains with mixed discrete and continuous dynamics, and supports *events* and *processes* modelling exogenous change. Motivated by numerous SAT-based planning approaches, we propose an approach to PDDL+ planning through SMT, describing an SMT encoding that captures all the features of the PDDL+ problem as published by Fox and Long. The encoding can be applied on domains with nonlinear continuous change. We describe the encoding in detail and provide in-depth examples. We apply this encoding in a simple planning algorithm, demonstrating excellent results on a set of benchmark problems. We further extend the encoding to include planning with control parameters. The extended encoding allows the definition of actions to include infinite domain parameters, called control parameters. We provide an example problem with control parameters and show how it can be encoded and solved in SMT.

1. Introduction

Planning for hybrid systems is an important area in planning, mainly motivated by the need to deal with real-world applications. Hybrid systems are systems described by discrete as-well-as continuous variables. Many real world problems involve hybrid systems, subject to (continuous) physical effects and controlled by (discrete) digital equipment. PDDL+ [20] is the extension of Planning Domain Definition Language (PDDL) designed to model hybrid systems, pushing forward the use of planning in real-world domains.

Examples of real-world domains include *unit commitment*, which is a fundamental problem in power systems engineering. Unit commitment is the problem of deciding which generating units should be switched on, and when to switch them on, in order to efficiently meet anticipated demand. It has traditionally been solved as a Mixed Integer Programming (MIP) problem, however Campion et al. [7] investigate the benefits of using planning over the current established methods. Another planning application involving hybrid systems is investigated by Fox et al. [21], in which improving the efficiency of multiple battery usage has been modelled as planning problem. Vallati et al. [52] focus on using mixed discrete-continuous planning to deal with unexpected circumstances in *urban traffic control*.

Many efforts are being made to extend current planning systems and modelling languages to support such domains. Penna et al. highlight the importance of hybrid systems in the real world and present the planner UPMURPHI [16], a universal planner that is capable of reasoning with mixed discrete/continuous domains while respecting the semantics of PDDL+. A number of other approaches have been proposed that can handle *subsets* of PDDL+, as described in Section 8.2.

In this paper we propose a new approach for PDDL+ planning that can handle the whole set of PDDL+ features and respects the full PDDL+ semantics. We propose a Satisfiability Modulo Theory (SMT) encoding of PDDL+ problems based on Boolean Satisfiability Problem (SAT) encodings of classical planning problems [31, 45]. Planning as SAT has been effective for classical planning problems (e.g. the planner SATPlan [33]) and through carefully devised preprocessing and encoding can be applied effectively to temporal planning problems (e.g. the planner ITSAT [42]). In contrast to the state-based search approach to planning, these planners perform a search on encodings of a plan trace, iteratively deepening the trace's horizon. We build on this previous work, using the additional expressive power of SMT to approach PDDL+ planning.

The proposed encoding allows for teh easy moelling of *Control Parameters* [48], which is an interesting emerging area in planning. Using control parameters allows for the modelling of actions with infinite do-

main parameters (e.g. real numbers). This is a challenging proposition for state-based search planners, as the branching factor becomes similarly infinite. We show that for some domains, by extending PDDL+ with control parameters actually improves the performance of our SMT-based planner, allowing it to scale to problems that without control parameters cannot be solved in a reasonable time.

The paper is structured as follows. In the next section we provide the background about hybrid systems, PDDL+, planning as SAT, and SMT. In Section 3, we introduce a working example that highlights PDDL+ features. We describe our encoding of PDDL+ into SMT in Section 4, followed by an in-depth example of our encoding using the working example in Section 5. In Section 6 we show how the encoding is extended to describe control parameters, and provide another in-depth example of the encoding. The approach is then evaluated on a set of benchmark problems in Section 7. In Section 8 we give an overview of related works and conclude in Section 9.

2. Background

In this section, we introduce the theoretical foundations of hybrid systems, namely the hybrid automata. We then give a brief introduction to PDDL2.1 and then explain how it is extended to describe the hybrid automata in PDDL+. After this we introduce planning as Boolean Satisfiability and then SMT.

2.1 Hybrid Systems

A hybrid system is a system where there are both continuous variables and discrete logical modes of operation. It represents a powerful model to describe the dynamic behaviour of modern engineering artefacts. Hybrid systems frequently occur in practice, e.g., in robotics or embedded systems. Dealing with hybrid systems is becoming more and more an important challenge, as many real-world scenarios feature a mixture of discrete and continuous behaviours. Some example applications include coordination of activities of a planetary lander, oil refinery management, autonomous vehicles, chemical plant management [15], planning for smart grids [7], and battery management [21]. Such scenarios motivate the need to reason with mixed discrete-continuous domains.

The theory of hybrid automata, introduced by Henzinger [24], represents a well-defined formalism for describing hybrid systems. Intuitively, hybrid automata are finite state automata extended with continuous variables that evolve over time. More formally, we have the following:

Definition 1 (Hybrid Automaton). A *hybrid automaton* is a tuple $\mathcal{H} = (Loc, Var, Init, Flow, Trans, I)$, where

- Loc is a finite set of locations,
- $Var = \{x_1, \ldots, x_n\}$ is a set of real-valued variables,
- $Init(\ell) \subseteq \mathbb{R}^n$ is the set of initial values for $x_1, \ldots, x_n$ for all locations ℓ.
- For each location ℓ, $Flow(\ell)$ is a relation over the variables in Var and their derivatives of the form:

$$\dot{x}(t) = Ax(t) + u(t), u(t) \in \mathcal{U},$$

  where $x(t) \in \mathbb{R}^n$, A is a real-valued $n \times n$ matrix and $\mathcal{U} \subseteq \mathbb{R}^n$ is a closed and bounded convex set.
- $Trans$ is a set of discrete transitions. A discrete transition $t \in Trans$ is defined as a tuple (ℓ, g, ξ, ℓ') where ℓ and ℓ' are the source and the target locations, respectively, g is the guard of t (given as a linear constraint), and ξ is the update of t (given by an affine mapping).
- $I(\ell) \subseteq \mathbb{R}^n$ is an invariant for all locations ℓ.

An example is the hybrid automaton for a thermostat shown in Figure 1. Here, the temperature is represented by the continuous variable x. In the discrete location corresponding to the heater being off, the

temperature falls according to the flow condition $\dot{x} = -0.1x$, while, when the heater is on, the temperature increases according to the flow condition $\dot{x} = 5 - 0.1x$. The discrete transitions state that the heater *may* be switched on when the temperature falls below 19 degrees, and switched off when the temperature is greater than 21 degrees. Finally, the invariants state that the heater can be on (off) *only* if the temperature is not greater than 22 degrees (not less than 18 degrees).

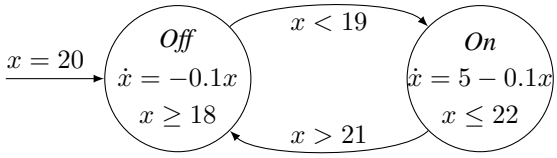


Figure 1: Thermostat hybrid automaton

2.2 PDDL2.1 and PDDL+ Planning

Application of planners in real world problems has challenged the limitations of PDDL and increased the necessity of dealing with time and resources. This lead to the development of PDDL2.1 [19] as the extension of PDDL able to model temporal domains. PDDL2.1 has been extended to PDDL+ [20] to enable the modelling of mixed discrete-continuous domains. In this section we first we discuss the semantics of PDDL2.1 and later describe the extensions that have been added as PDDL+.

Definition 2 (PDDL2.1 Planning). A PDDL2.1 planning problem is a tuple $\Pi := \{P,V,A,I,G\}$, where P is a set of propositions; V is a vector of real variables, called fluents; both are manipulated by A, a set of durative and instantaneous actions. $I(P,V)$ is a function over $P \cup V$ which describes the *initial state* of the problem and $G(P,V)$ is a function that describes the *goal condition*.

A durative action $a \in A$ is described as a tuple:

$$a := \{pre_a, \mathit{eff}_a, dur_a\}$$

where $pre_a(P,V)$ is a function over $P \cup V$ that represents the action's preconditions – conditions that must hold for the action to be applied. Similarly, eff_a represents the action's effects, and dur_a is a duration constraint, a conjunction of numeric constraints corresponding to the duration of the action a.

A single condition is either a single proposition $p \in P$, its negation, or a numeric constraint over V. A precondition is a conjunction of zero or more single conditions. The precondition of an action pre_a can be separated into three disjoint subsets:

$$pre_{\vdash a}, pre_{\leftrightarrow a}, pre_{\dashv a} \subseteq pre_a$$

These represent the conditions that must hold at the start of the action, throughout its execution, and at the end of the action, respectively.

Action effects are described by seven subsets:

$$\begin{array}{l} \mathit{eff}^{+}_{\vdash a}, \mathit{eff}^{-}_{\vdash a}, \mathit{eff}^{num}_{\vdash a}, \\ \mathit{eff}^{+}_{\dashv a}, \mathit{eff}^{-}_{\dashv a}, \mathit{eff}^{num}_{\dashv a}, \\ \mathit{eff}_{\leftrightarrow a} \end{array}$$

The first six are the instantaneous effect of adding or removing propositions, or instantaneous numeric effects. These are bound to the start or end of the action. For example, $\mathit{eff}^{+}_{\vdash a}$ denotes the propositions added at the start of the action. In the semantics of PDDL2.1, the values of such instantaneous effects can be exploited to support other actions only after a small amount of time ε [19]. This is referred to as *epsilon separation*. The final set, $\mathit{eff}_{\leftrightarrow a}$ is a conjunction of *continuous* numeric effects $\mathit{eff}_{\leftrightarrow}$, which are applied continuously

while the action is executing. As a special case, *instantaneous* actions have duration 0, have only one set of preconditions pre_a; and three sets of effects eff_a^+, eff_a^-, and eff_a^{num}.

Actions can only be applied together at the same time if they are not mutually exclusive (mutex). Actions $a1$ and $a2$ can be applied simultaneously if:

$$pre_{a1} \cap (eff_{a2}^+ \cup eff_{a2}^- \cup eff_{a2}^{num}) = \emptyset$$
$$eff_{a1}^+ \cap eff_{a2}^- = eff_{a2}^+ \cap eff_{a1}^- = \emptyset$$
$$\{v1 \in eff_{a1}^{num}\} \cap \{v2 \in eff_{a2}^{num}\} = \emptyset$$

PDDL+ is an extension of PDDL2.1, based on hybrid automata semantics. PDDL+ extends PDDL2.1 to support the modelling of exogenous events, reflecting changes that are initiated by the environment. These are introduced by the new constructs of *processes* and *events*.

Definition 3 (PDDL+ Planning). A PDDL+ planning problem is a tuple $\Pi+ := \langle P,V,A,Ps,E,I,G \rangle$, in which P is a set of propositions; V is a vector of real variables, called fluents; and A is a set of durative and instantaneous actions. Ps is a set of processes, and E a set of events. $I(P,V)$ and $G(P,V)$ represent the initial state and goal condition respectively.

The elements P,V,A,I, and G are as in Definition 2. As an analogue, an event $e \in E$ is akin to an instantaneous action: if an event's preconditions pre_e are satisfied, it occurs, yielding the event's instantaneous effects. Similarly, processes are akin to durative actions which apply continuous effects over a period of time.

Note that continuous processes are triggered as soon as their precondition becomes true, and in this sense they *must* be triggered. Exogenous events follow the same semantics. The rationale behind this is that processes and events are used to model changes that are initiated by changes in the world, therefore they are not under the direct control of the executive and are triggered immediately (see Bogomolov et al. [5] for more details). This is a critical distinction between processes/events and actions. The process/event will automatically occur as soon as its precondition is satisfied; whereas an action will only happen if chosen to be executed by the planner.

Furthermore, effects become instantaneously available to events, without the epsilon separation. This means, if one event $e1$ is triggered, with effects that satisfy another event $e2$ and trigger a process $p1$, then $e1$, $e2$, and the start of $p1$ all happen at exactly the same time-point. It is due to this behaviour that PDDL+ semantics place a bound on the number of cascading (parallel) events. According to PDDL+ semantics, a causal chain of events cannot include a cycle, and a single event cannot be triggered more than once in a single instant [18].

Given the PDDL+ semantics for modelling hybrid systems, finding an efficient planner that can handle events and processes is the next crucial step. A number of methods have been proposed to handle mixed discrete and continuous domains. We review these in Section 8.

2.3 Planning as Satisfiability

The problem of determining SAT is the problem of finding an assignment of truth values to variables in order to make a set of propositional formulae true. The problem stated as: given ϕ a Boolean expression with variables $X : \{x_1, x_2, ..., x_n\}$, is it true that there exists an assignment to variables X such that ϕ is satisfied? In other words:

$$\exists\, x_1, x_2, ..., x_n \quad \phi?$$

For example,

$$\phi := (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (x_3 \vee x_2) \wedge \neg x_3$$

is false, since there are no assignments to the three variables x_1, x_2, x_3 that will satisfy ϕ. A SAT solver solves the decision problem and if the expression is satisfiable, often returns an example satisfying assignment. Kautz and Selman [30] formalised propositional planning as a Boolean Satisfiability (SAT) problem in the following way.

First, the classical planning problem for a fixed horizon n is modelled by defining n copies of the Boolean variables which describe the state and actions, and unrolling n times the transition relation that holds between states. This gives a set of variables (X) and constraints between them (ϕ). A satisfying assignment to the variables corresponds to a plan trace.

In more detail, the set of variables X comprises one Boolean variable for each $a \in A$, and each $p \in P$ at each time-step $t_0, \ldots, t_n$. In propositional planning, there are no numeric variables $v \in V$. Constraints are added to ϕ, asserting:

- The initial state holds at time-step 0, and goals at time-step n.
- If an action is true, then its preconditions hold in that time-step.
- If an action is true, then its effects hold in the following time-step.
- Two actions that are mutex cannot be applied together in the same time-step.

To solve the original planning problem without a fixed time horizon, an initial encoding with horizon n_0 is solved, and if found, the satisfying assignment is converted into a corresponding plan. Otherwise, the horizon is incremented and the process is repeated [32].

A significant number of works have been devoted to formalising planning problems using propositional logic, and improving those encodings [46, 10, 47, 45, 8], which we discuss in Section 8.

2.4 Satisfiability Modulo Theory

Satisfiability Modulo Theory (SMT) [2] is the problem of deciding the satisfiability of a first-order formula expressed in a given theory. An SMT problem is given as a set of variables and a set of constraints over those variables. In contrast to the SAT problem, the variables are not restricted to Boolean values, but depend upon a *theory*, and the constraints are expressed with respect to a background *logic*. The theory and logic are critical elements of an SMT problem. Theories exist for Boolean propositions, finite and infinite trees, lists, bitvectors, arrays, integers, and reals. SMT takes advantage of the fact that some of them already have efficient solvers in order to reason more efficiently. In our work we use the theory of real numbers and the logic of *quantifier-free non-linear real arithmetic*.

For example, an SMT problem in *quantifier-free linear real arithmetic* might be:

$$\exists u,v,x,y,z \quad \phi?$$

where

$$\phi := (x + 3 <= 2u) \vee (v + 4 >= y) \vee (x + y + z >= 2)$$

Problems in SMT can be expressed in the SMT-LIB standard syntax [1]. Figure 2 illustrates the example problem above in the standard sytax. The encoding we describe in this paper is produced in this standard format, and can be read many SMT solvers, including Z3 [14], which we use to solve our problems in Section 7.

3. Working Example

For the purpose of clarity and ease, we have chosen a simple *Free Fall* problem as our working example and we use this problem to indicate different concepts explained throughout this paper.

In the first part of this section we explain the *Free Fall* problem, and later we describe the hybrid automaton corresponding to this problem. Finally, we provide the PDDL+ model of our problem and map the hybrid automaton to the PDDL+ model. We will then present the SMT encoding of this problem in Section 5.1.

```
( set-logic QF_LRA )
( declare-fun u () Real)
( declare-fun v () Real)
( declare-fun x () Real)
( declare-fun y () Real)
( declare-fun z () Real)
( assert
    (or
    ( <= (+ x 3) (* 2 u ) )
    ( >= (+ v 4) y )
    ( >= (+ x y z ) 2)
    )
)
```

Figure 2: SMT problem in the SMT-LIB standard syntax.

3.1 Free Fall Problem

Our working example considers a ball that falls vertically under the influence of gravity (g) and bounces when it touches the floor. The goal is that a robotic hand catches the ball. Specific conditions will be defined for the robot which determines when to catch the ball (i.e. at a specific time or position, or after a specific number of bounces), as shown in Figure 3. It is important to note that Figure 3 shows the real case of a bouncing ball, however in our working example the motion is idealised as other forces and their effects (i.e. air resistance) are assumed to be negligible (so the ball will bounce an infinite number of times and every time reach to its initial height). The ball can be released with an initial velocity of zero ($v_0 = 0$) or it can have an initial velocity of v_0.

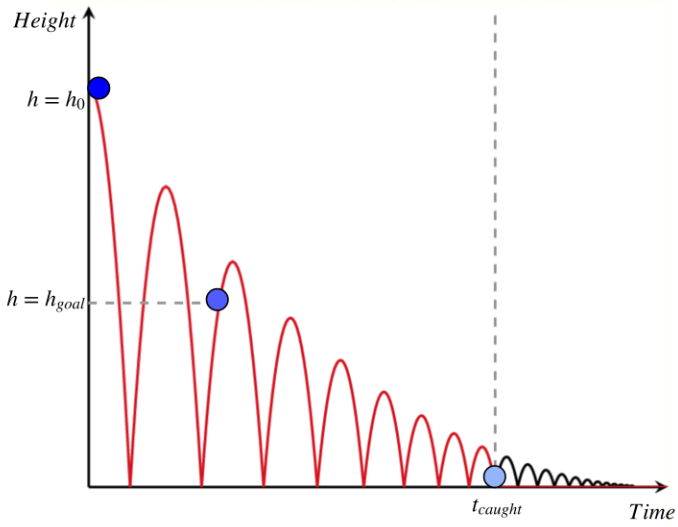


Figure 3: Working example (Free Fall problem): indicates a ball that is released vertically from the initial height of h_0 and bounces as soon as it reaches the floor. The diagram shows two different goal conditions we can specify for the robot to catch the ball; at a specific height (h_{goal}) or at a specific time after releasing the ball (t_{caught}).

3.2 Hybrid Automaton: Free Fall Problem

The hybrid automaton of the Free Fall problem is shown in Figure 4. In this problem the height and velocity of the ball are represented by the continuous variables h and v, respectively. The acceleration of the ball due to gravity is represented by the variable a.

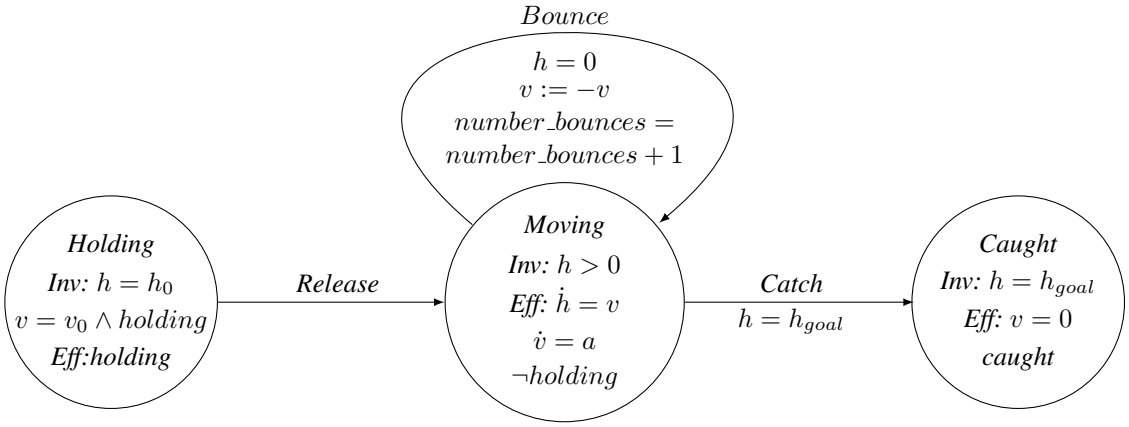


Figure 4: Free Fall hybrid automaton

Considering the dynamics of the problem, we have three discrete locations. The initial state which indicates our first discrete mode: the ball is held at the height h_0 and the velocity $v = v_0$ (this state is shown by the left circle in Figure 4). As soon as the ball is released, a discrete transition moves the automaton to the next discrete location representing the moving state (this state is shown by the middle circle in Figure 4). In this state, the height of the ball changes according the velocity $\dot{h} = v$, and the velocity changes according to acceleration $\dot{v} = a$. As soon as the height of the ball becomes zero ($h = 0$), the ball bounces. A discrete transition negates the velocity.

The robot can catch the ball as soon as the conditions of the goal state are satisfied. In Figure 4 this condition is that the ball is at a specific height $h = h_{goal}$. At this point, the hybrid automaton will move to the next discrete state which represents the state where the ball has been caught by the robot (this state is shown by the right circle in Figure 4).

3.3 PDDL+ Model of the Free Fall Problem

In this section we describe how the hybrid automaton of the Free Fall problem can be written in PDDL+. A planning model represented in PDDL+ consists of the domain model and the problem instance. The domain model lists the action schema and the possible processes and events. The problem instance describes the objects, the initial state, and the goal state. The domain corresponding to the hybrid automaton in Figure 4 is shown in Figure 5, while the problem instance is shown in Figure 6. In this problem we include a single ball, *ball1*, and define that the robot should catch the ball when the height of the ball is 5.

Grounding the problem by applying the objects from the problem instance to the predicates of the domain forms the propositional variables $(holding\ ball1)$ and $(caught\ ball1)$. These variables describe a discrete state-space of 4 states:

- The initial state in which $(holding\ ball1)$ is true and $(caught\ ball1)$ corresponds to the *Holding* location of the hybrid automaton.
- The state in which both variables are false is reached after applying the action $(release\ ball1)$. This state corresponds to the *Moving* location of the hybrid automaton. In this discrete state, the process $(moving\ ball1)$ asserts the continuous change on the numeric variables.

```
(define (domain dropping_ball)
(:requirements :fluents :typing :negative-preconditions)

(:types ball)
(:predicates (holding ?b - ball) (caught ?b - ball))
(:functions
    (velocity ?b - ball)
    (height ?b - ball)
    (catch_height)
    (number_bounces ?b - ball)
    (catch_bounce)
)

(:action release
 :parameters (?b - ball)
 :precondition (and (holding ?b) (= (velocity ?b) 0))
 :effect (and (not (holding ?b)))
)

(:process moving
 :parameters (?b - ball)
 :precondition (and (not (holding ?b)) (>= (height ?b) 0))
 :effect (and
    (increase (velocity ?b) (* #t -9.8) )
    (increase (height ?b) (* #t (velocity ?b))))
 )

(:event bounce
 :parameters (?b - ball)
 :precondition (and (<= (height ?b) 0.000001))
 :effect (and
    (increase (number_of_bounces ?b) 1)
    (assign (velocity ?b) (* -1 (velocity ?b))))
)

(:action catch
 :parameters (?b - ball)
 :precondition (and
    (>= (height ?b) (catch_height))
    (<= (height ?b) (+ (catch_height) 0.1)
    (>= (number_bounces ?b) (catch_bounce)))
 :effect (and
    (holding ?b)
    (assign (velocity ?b) 0)
    (caught ?b))
))
```

Figure 5: Simplified PDDL+ domain description for Free Fall.

- The state where both variables are true, is reached by applying the action $(catch\ ball1)$. This state corresponds to the *Caught* location in the hybrid automaton. The goal holds in this state.
- The state in which $(holding\ ball1)$ is false and $(caught\ ball1)$ is true is not reachable from the initial state without first reaching the goal, and does not appear in the hybrid automaton.

```
(define (problem 1_ball)
(:domain dropping_ball)
(:objects ball1 - ball)

(:init
    (holding ball1)
    (= (velocity ball1) 0)
    (= (height ball1) 10)
    (= (catch_height) 5)
    (= (number_bounces ball1) 0)
    (= (catch_bounce) 1)
)

(:goal
    (caught ball1)
))
```

Figure 6: Simplified PDDL+ Free Fall problem file

The continuous change of the location *Moving* is asserted by the process $(moving\ ball1)$, which is active while its preconditions, $(not(holding\ ball1))$ and $(>=\ (height\ ball1)\ 0)$, are satisfied. The event $(bounce\ ball1)$ has no effect upon the discrete state, instead affecting only the numeric variables and also increases the $(number_bounces)$. This event corresponds to the self-looping transition *Bounce* in the hybrid automaton. As explained previously, an event is triggered as soon as its preconditions are satisfied. In this case, the event will be triggered as soon as the ball has reached the floor. This precondition is shown as $(<=\ (height\ ?b)\ 0.000001)$.

4. Encoding PDDL+ Domains in SMT

In this section we describe how we encode a PDDL+ domain into SMT. First we describe the basic elements of the encoding, the happening and later we expand the encoding for the whole planning problem.

4.1 Encoding of a single happening

Our encoding is based on the notion of *happening*, which is used to capture the discrete change in the state at a given time point due to the effects of actions, processes, or events. Namely, each happening encodes the causal chain of events, processes and instantaneous actions which might occur simultaneously at a given time point. We have defined a bound B as the length of the causal cascading instantaneous events and we split durative actions as two instantaneous actions, representing the start and end of the action, and one process representing the continuous numeric effects and invariant.

Definition 4 (Happening). A happening is the tuple $x := \{P, V, E, Ps, A\}$, where:

- $P = \{P_0, \ldots, P_{B+1}\}$ are the propositional state variables at happening x, in $B + 1$ layers;
- $V = \{V_0, \ldots, V_{B+1}\}$ are the real state variables at happening x, also in $B + 1$ layers;
- $E = \{E_0, \ldots, E_B\}$ represents the chain of events triggered at happening x. The chain is of length B;
- Ps represents the set of processes active over the next interval;
- A is the set of actions applied at the happening.

Each P_i is a set of all the propositions, i.e.:

$$P_i = \{p_i, \forall p \in P\}, i \in \{0, \ldots, B + 1\}$$

Similarly we have the following for the V_i and E_i:

$$V_i = \{v_i, \forall v \in V\}, \quad i \in \{0, \ldots, B+1\}$$
$$E_i = \{e_i, \forall e \in E\}, \quad i \in \{0, \ldots, B\}$$

An example of happening is shown in Figure 7. The chain of events forms a set of layers at each happening. Each circle in Figure 7 represents a layer. The number of event steps in each happening is bounded to B. The last layer indicates the final value of the state variables after considering the effects of the cascading events. If the causal chain of events is longer than this bound, then a valid plan will not be found.

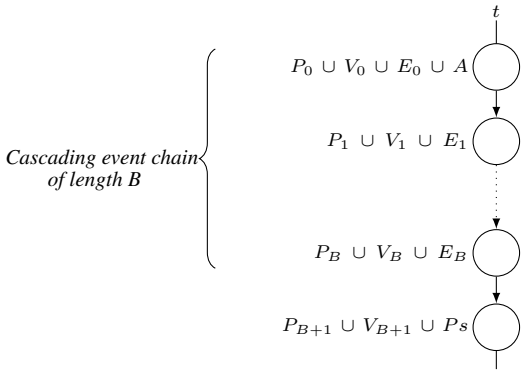


Figure 7: A single happening occurs at time t, and includes several sets of state variables. These sets describe a causal chain of instantaneous events.

Following from Definition 4, happening x_t is encoded by the SMT variables:

$$x_t := \left\langle \begin{array}{l} time, \\ P_{0,t}, \ldots, P_{B+1,t}, \\ V_{0,t}, \ldots, V_{B+1,t}, \\ E_{0,t}, \ldots, E_{B,t}, \\ Ps_t, A_t, \\ flow_{V_t}, dur_{Ps_t} \end{array} \right\rangle$$

The happenings in our encoding either describe a moment of discrete change - which corresponds to the discrete transition *Trans* of the hybrid automata - or a point in time between moments of discrete change in which the derivative is equal to zero for some numeric continuous change. The latter case is to ensure that invariant conditions hold, avoiding the case described later in Figure 11. Between happenings there is only continuous numeric change (*Flow*). The key difference between a hybrid automaton and the SMT encoding is that multiple actions can be performed in a single happening in parallel, meaning that while the hybrid automaton is exponential in the size of the PDDL+ description, our encoding will be linear.

The set $flow_{V_t} := \{flow_{v_t} | \forall v_t \in V_t\}$ is a set of numeric expressions that represent the change in value of each numeric variable v from this time point to the next. The variables $dur_{Ps_t} := \{dur_{ps_t} | \forall ps_t \in Ps_t\}$ represents the remaining duration of each process. dur_{ps} and is constrained to be positive if and only if the process is currently executing.

The constraints within a happening are shown in Figure 8.

- Proposition and real variable support

  These constraints ensure that the value of propositions (H1-H4) and real variables (H5-H6) remains consistent from $P_{0,t} \cup V_{0,t}$ to $P_{B+1,t} \cup V_{B+1,t}$. Constraints (H1) and (H2) enforce the correct values after considering the add and delete effects of the initial events ($E_{0,t}$) and actions (A_t). Similarly,

Proposition and real variable support

H1. $\bigwedge_{p\in P} p_{1,t} \rightarrow (p_{0,t} \vee \bigvee_{e\in E|p\in eff_e^+} e_{0,t} \vee \bigvee_{a_t\in A|p\in eff_a^+} a_t)$

H2. $\bigwedge_{p\in P} \neg p_{1,t} \rightarrow (\neg p_{0,t} \vee \bigvee_{e\in E|p\in eff_e^-} e_{0,t} \vee \bigvee_{a\in A|p\in eff_a^-} a_t)$

H3. $\bigwedge_{i=1}^{B} \bigwedge_{p\in P} p_{i+1,t} \rightarrow (p_{i,t} \vee \bigvee_{e\in E|p\in eff_e^+} e_{i,t})$

H4. $\bigwedge_{i=1}^{B} \bigwedge_{p\in P} \neg p_{i+1,t} \rightarrow (\neg p_{i,t} \vee \bigvee_{e\in E|p\in eff_e^-} e_{i,t})$

H5. $\bigwedge_{v\in V}(\bigwedge_{a\in A|v\in eff_a^{num}} \neg a_t \wedge \bigwedge_{e\in E|v\in eff_e^{num}} \neg e_{0,t}) \rightarrow (v_{i+1,t} = v_{i,t})$

H6. $\bigwedge_{i=1}^{B} \bigwedge_{v\in V}(\bigwedge_{e\in E|v\in eff_e^{num}} \neg e_{i,t}) \rightarrow (v_{i+1,t} = v_{i,t})$

Event preconditions and effects

H7. $\bigwedge_{i=0}^{B} \bigwedge_{e\in E} e_{i,t} \leftrightarrow pre_e(P_i \cup V_i)$

H8. $\bigwedge_{i=0}^{B} \bigwedge_{e\in E} e_{i,t} \rightarrow eff_e(P_{i+1} \cup V_{i+1})$

Action preconditions and effects

H9. $\bigwedge_{a\in A} a_t \rightarrow pre_a(P_{0,t} \cup V_{0,t})$

H10. $\bigwedge_{a\in A} a_t \rightarrow eff_a(P_{1,t} \cup V_{1,t})$

Process triggering

H11. $\bigwedge_{ps\in Ps} ps_t \leftrightarrow pre_{ps}(P_{B+1,t} \cup V_{B+1,t})$

H12. $\bigwedge_{ps\in Ps} dur_{ps_t} >= 0$

H13. $\bigwedge_{ps\in Ps} ps_t \leftrightarrow (dur_{ps_t} > 0)$

Action mutexes

H14. $\bigwedge_{a\in A} \bigwedge_{a'\in A|a\nparallel a'} (\neg a_t \vee \neg a'_t)$

Figure 8: Reduction of a PDDL+ happening to SMT.

constraints (H3) and (H4) enforce the propositional add and delete effects throughout the instantaneous chain of events. (H5) enforces the numeric effects of the initial events and actions. (H6) enforces the numeric effects of the remaining events. Note that after the first layer at each happening there are only events, and no actions.

- Event preconditions and effects

  This set of constraints enforce that an event is triggered if and only if its precondition holds (H7) and that if an event is triggered, its effects are present in the next layer of that happening (H8).

- Action preconditions and effects

  Similar to (H6-H7), these constraints ensure for actions A_t that their preconditions must hold in $P_{0,t} \cup V_{0,t}$ (H9) and their effects are enforced in $P_{1,t} \cup V_{1,t}$ (H10).

- Process triggering

This group of constraints enforce that a process is active if and only if its preconditions are satisfied in set $P_{B+1,t} \cup V_{B+1,t}$ (H11). It also enforces that the real variable dur_{ps_t} for each process is greater than or equal to zero (H12), and strictly greater that zero if and only if the process is active at the end of the causal chain of events (H13). These constraints will be used to ensure that a process cannot finish outside of a happening.

- Action mutexes

  This set of binary constraints enforces that no two mutex actions can be applied simultaneously. For each pair of mutex actions, denoted $a_t \nparallel a'_t$, at least one must be false (H14).

4.2 Encoding of a Plan

Having defined happenings, a PDDL+ problem $\Pi+$ can be encoded as a bounded number of happenings $X := \{x_1, \ldots, x_n\}$, such that any proof for the SMT formula represents the trace of a valid plan for $\Pi+$. The *plan* corresponding to that trace is the set of action assignments $A_1 \cup \ldots \cup A_n$.[1]

Therefore, the existence of a plan for a *PDDL+ planning problem* $\Pi+$ *with bound* n is proved by building the SMT formula $\Pi+_n$ in the theory of quantifier-free (nonlinear) real arithmetic with n copies of the set of happening variables $X = \{x_1, \ldots, x_n\}$ for $n \geq 1$. A *plan for* $\Pi+$ is the assignment to the action variables in any proof of $\Pi+_n$. The encoding is illustrated by Figure 9.

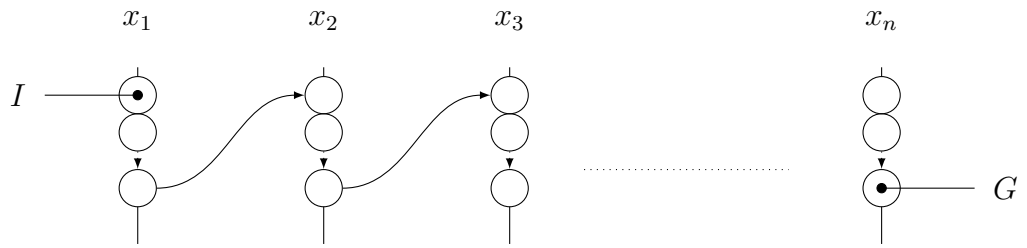


Figure 9: A plan is found by building a formula with n copies of the set of variables x_t for $t = 1 \ldots n$. Each Happening models discrete change. Between happenings there is only continuous numeric change. The initial state is modelled in t_1, and the goal constraints are added to t_n.

The constraints for a happening in Figure 8 are copied for each happening $x_1 \ldots x_n$. Additional constraints in the SMT formula $\Pi+_n$ are shown in Figure 10 and explained below.

Instance description

P1. $I(P_{0,1} \cup V_{0,1})$

P2. $G(P_{B+1,n} \cup V_{B+1,n})$

P3. $time_0 = 0$

P4. $\bigwedge_{i=2}^{n} time_i \geq time_{i-1} + \varepsilon$

Proposition support

P5. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} p_{0,i} \rightarrow p_{B+1,i-1}$

P6. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} \neg p_{0,i} \rightarrow \neg p_{B+1,i-1}$

Invariants

P7. $\bigwedge_{i=2}^{n} \bigwedge_{ps \in Ps} ps_{i-1} \rightarrow dur_{ps_i} = dur_{ps_{i-1}} - time_i + time_{i-1}$

P8. $\bigwedge_{i=1}^{n-1} \bigwedge_{ps \in Ps} ps_i \leftrightarrow pre_{\leftrightarrow ps_i}$

P9. $\bigwedge_{i=1}^{n-1} \bigwedge_{e \in E} \neg pre_{\leftrightarrow e_i}$

Continuous change on real variables

P10. $\bigwedge_{i=1}^{n-1} \bigwedge_{v \in V} flow_{v_i} = \int_{time_i}^{time_{i+1}} \sum_{ps \in Ps} eff_{\leftrightarrow ps}^{num}(V_i) dt$

P11. $\bigwedge_{i=2}^{n} \bigwedge_{v \in V} (v_{0,i} = v_{B+1,i-1} + flow_{v,i-1})$

Figure 10: Reduction of PDDL+ planning problem $\Pi+$ to SMT.

1. As processes and events do not appear in a PDDL+ plan.

- Instance description

  These constraints enforce the initial state to hold in the first happening (P1), and that the goal is achieved in the final happening (P2). Following PDDL2.1 and PDDL+ semantics, they constrain the timing of happenings to enforce epsilon separation (P3-P4).

- Proposition support

  These constraints ensure that the discrete state variables do not change between happenings (P5-P6).

- Invariants

  These constraints ensure that the continuous numeric change between happenings does not violate any invariant constraints. First (P7) ensures that if a process is active in the previous happening, its duration is decreased by the time between happenings. This constraint, in combination with constraints (H12-H13) of Figure 8, ensure that a process cannot end between happenings. A process can remain active over intervals spanning multiple happenings.

  Constraint (P8) enforces the invariant of the process. If a process is active, then the precondition of the process is active over the whole interval between happenings, and if the process is not active, then its preconditions are false over the whole interval. For constraints over real valued variables, this is done by checking the value either side of the interval. For nonlinear change, this is not sufficient (as shown in Figure 11). It is necessary to include the following additional constraint:

$$ps_{t_j} \rightarrow \bigwedge_{a_{t_j}=1}^{A_{t_j}} \left((\frac{d^a f}{dt^a})_j (\frac{d^a f}{dt^a})_{j+1} >= 0 \right);$$

  where f is the numeric, non-constant part of the invariant, and where the $(A+1)$th derivative of f is identically zero. This ensures that the derivatives of the function do not cross zero over the interval, thus a fluctuating value of f cannot violate the invariant condition between t_j and t_{j+1}.

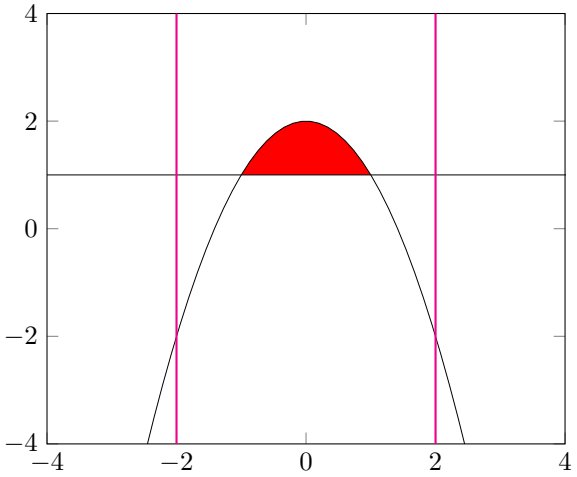


Figure 11: The plot illustrates continuous non-linear numeric change between two time points, indicated by vertical lines. While considering the non-linear change, we need to check whether the invariant condition of a process is satisfied throughout the interval. If the invariant asserts that the function should remain below the horizontal line, the figure shows that it is not sufficient to check the value of the function only at the time points.

Constraint (P9) similarly ensures that an event is not triggered during an interval.

- Continuous change on real variables

  These constraints enforce continuous change over the interval (P10), applying the result to each real variable (P11). In order to calculate the change, the indefinite integral of the process' effects upon the variable must be computed. This is done automatically using the computer algebra systems (CAS) SymPy [50] and Piranha [3].

  Note that in this approach, the integration and differentiation required for P8, P9, and P10 are performed outside the solver, during the encoding. Hence the integration is done only once for each domain.

4.3 SMTPlan Architecture

In this section we describe the architecture of SMTPlan. As shown in Figure 12, SMTPlan encodes the PDDL+ domain with an initial number of happenings n. For each problem, assuming that the initial and goal state are different, we need at least two happenings to encode our problem. Then the encoding is passed to the SMT solver to check if it is satisfiable or not. If the encoding is satisfiable, the satisfying assignment found by the SMT solver is converted into the corresponding plan. Otherwise, the number of happenings is increased and the process is repeated.

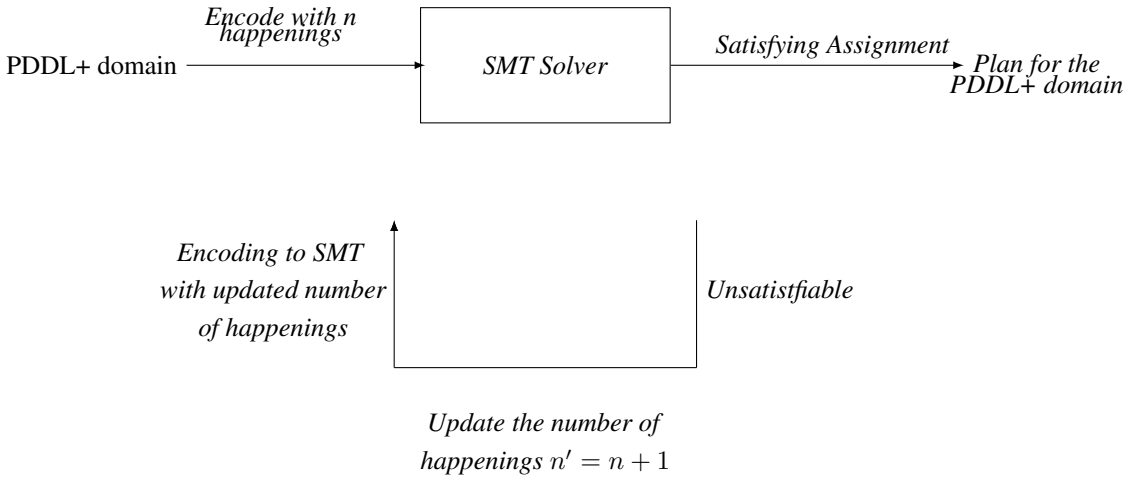


Figure 12: An overview of a SMT encoding process for finding a plan for a planning problem. replace image completely

5. Examples

In this section we describe the encoding of two different problems in SMT. The first one will be the Free Fall problem that we have described earlier as our working example and the second one will be the Generator Problem.

5.1 Free Fall

We consider the working example introduced earlier: *Free Fall* problem. We release the ball from the height of 10 meters from the ground (initial velocity is zero). The goal is to catch the ball when the height of the ball is 5 meters. The domain and problem files are shown in Figures 5 and 6, respectively.

In order to describe the SMT encoding, first we define the SMT variables used in our encoding, for a single happening x_t. Since the number of events in our domain is equal to one, the bound of cascading events in our problem is fixed as one, so we just have $E_{0,t}$. This means the number of internal layers at each

happening is equal to 2. Below we show the constraints on one happening (x_1), then the constraints describing the transition relation in an encoding with four happenings $x_1, \ldots, x_3$.

$$x_t := \left\langle \begin{array}{ll} time & : Real \\ holding_{0,t}, holding_{1,t}, caught_{0,t}, caught_{1,t} & : Bool \\ height_{0,t}, height_{1,t}, velocity_{0,t}, velocity_{1,t}, number\_bounces_{0,t}, number\_bounces_{1,t} & : Real \\ bounce_{0,t} & : Bool \\ moving_t & : Bool \\ release_t, catch_t & : Bool \\ moving_{dur,t} & : Real \end{array} \right\rangle$$

As we can see in section 4.1, the first group of constraints are related to *proposition and real variable support*. The following constraints show $H1$ and $H2$:

$$\begin{array}{l} holding_{1,1} \rightarrow (holding_{0,1} \vee catch_1) \\ caught_{1,1} \rightarrow (caught_{0,1} \vee catch_1) \\ \neg holding_{1,1} \rightarrow (\neg holding_{0,1} \vee release_1) \\ \neg caught_{1,1} \rightarrow \neg caught_{0,1} \end{array}$$

Constraint $H5$ is modelled as following:

$$\begin{array}{l} (\neg bounce_{0,1} \wedge \neg catch_1) \rightarrow (velocity_{1,1} = velocity_{0,1}) \\ \neg bounce_{0,1} \rightarrow (number\_bounces_{1,1} = number\_bounces_{0,1}) \\ height_{0,1} = height_{1,1} \\ \neg moving_1 \rightarrow (height_{1,1} = height_{0,1}) \end{array}$$

Since the bound on the number of cascading events is 1, there are no constraints required for $H3$,$H4$ and $H5$. The next group of constraints are *event preconditions and effects*, which are encoded as following (corresponding to $H7$ and $H8$):

$$\begin{array}{l} bounce_{0,1} = (height_{1,0} <= 1/1000000) \\ bounce_{0,1} \rightarrow (velocity_{1,1} = -velocity_{1,0}) \\ bounce_{0,1} \rightarrow (number\_bounces_{1,1} = number\_bounces_{0,1} + 1) \end{array}$$

Similar to the events, the next group of constraints are *actions preconditions and effects* which correspond to $H9$ and $H10$.

$$\begin{array}{l} release_1 \rightarrow holding_{0,1} \\ release_1 \rightarrow ((velocity_{0,1} <= 0) \wedge (velocity_{0,1} >= 0)) \\ catch_1 \rightarrow holding_{1,1} \\ catch_1 \rightarrow caught_{1,1} \\ catch_1 \rightarrow (height_{0,1} >= 5) \\ catch_1 \rightarrow (height_{0,1} <= 5.01) \\ catch_1 \rightarrow (velocity_{0,1} <= 0)) \\ release_1 \rightarrow \neg holding_{1,1} \\ catch_1 \rightarrow (velocity_{1,1} = 0) \\ catch_1 \rightarrow (number\_bounces >= 1) \end{array}$$

The constraints related to the *process triggering* ($H11 - H13$) are encoded as follows:

$$\begin{array}{l} moving_1 = \neg holding_{1,1} \wedge (height_{1,1} >= 0) \\ moving_1 = (moving_{dur,1} > 0) \\ moving_{dur,1} >= 0 \end{array}$$

Since the two actions that we have are not mutex, the constraint related to that ($H14$) is not needed. The next series of constraints encode the transition of one happening to the next happening. We start with *Instance description* constraints of the problem ($P1 - P4$). First the initial state of the problem (corresponding to $P1$):

$$holding_{0,1}$$
$$\neg caught_{0,1}$$
$$velocity_{0,1} = 0$$
$$height_{0,1} = 10$$
$$number\_bounces_{0,1} = 0$$

The goal state (corresponding to $P2$):

$$caught_{1,3}$$

The timings of the happenings (corresponding to $P3$ and $P4$):

$$time_1 = 0$$
$$time_2 >= time_1 + 0.001$$
$$time_3 >= time_2 + 0.001$$

The next group of constraints is *Proposition support* (corresponding to $P5$ and $P6$).

$$holding_{0,2} \rightarrow holding_{1,1}$$
$$caught_{0,2} \rightarrow caught_{1,1}$$
$$\neg holding_{0,2} \rightarrow \neg holding_{1,1}$$
$$\neg caught_{0,2} \rightarrow \neg caughtcaught_{1,1}$$

Invariants are the next group of constraints (corresponding to $P7 - P9$).

$$moving_1 \rightarrow (moving_{dur,2} = moving_{dur,1} + time_1 - time_2)$$
$$moving_1 \rightarrow (height_{0,2} * height_{1,1} >= 0)$$

The last group of constraints ensures the *continuous change on the real variables*. The kinematic equations of the free fall problem are the following:

$$v = at + v_0$$
$$h = 1/2at^2 + v_0t + h_0$$

Based on these equations, we can encode the height and the velocity of the ball as follows:

$$moving_1 \rightarrow (velocity_{0,2} = -9.8 * (time_2 - time_1) + velocity_{1,1})$$
$$moving_1 \rightarrow (height_{0,2} = -4.9 * (time_2 - time_1)^2 + velocity_{1,1} * (time_2 - time_1) + height_{1,1})$$

After encoding the problem with a initial number of happenings, the SMTPlan passes the encoding to an SMT solver. The SMT solver tries to find assignments for the variables in order to make the whole set of constraints to be satisfied. In the case that the solver can not find assignments that can satisfy the encoding, the SMTPlan increases the number of happenings and then encodes the problem again. This process continues until the solver can find a satisfying assignments for the variables. In this problem the SMTPlan found satisfying assignments for SMT variables with three happenings.

Table 1 shows the SMT variables and the values assigned to them, corresponding to a valid plan trace. The corresponding plan generated by SMTPlan is shown in Figure 13.

5.2 Simple Generator

In this section we describe the encoding of a PDDL+ benchmark, the *generator* problem. The domain and the problem files are shown in Figures 14 and 15, respectively. The initial state asserts that there is one generator;

```
0.0      : release      [0.0]
1.850446 : catch        [0.0]
```

Figure 13: Plan for the Free Fall problem.

x_1	x_2	x_3
$time_1 := 0$	$time_2 := 1.428571$	$time_3 := 1.850446$
$holding_{0,1} := 1$ $holding_{1,1} := 0$ $caught_{0,1} := 0$ $caught_{1,1} := 0$	$holding_{0,2} := 0$ $holding_{1,2} := 0$ $caught_{0,2} := 0$ $caught_{1,2} := 0$	$holding_{0,3} := 0$ $holding_{1,3} := 1$ $caught_{0,3} := 0$ $caught_{1,3} := 1$
$height_{0,1} := 10$ $height_{1,1} := 10$ $velocity_{0,1} := 0$ $velocity_{1,1} := 0$ $number\_bounces_{0,1} := 0$ $number\_bounces_{1,1} := 0$	$height_{0,2} := 0.000001$ $height_{1,2} := 0.000001$ $velocity_{0,2} := -13.999999$ $velocity_{1,2} := 13.999999$ $number\_bounces_{0,2} := 0$ $number\_bounces_{1,2} := 1$	$height_{0,3} := 5.034156$ $height_{1,3} := 5.034156$ $velocity_{0,3} := 9.865624$ $velocity_{1,3} := 0$ $number\_bounces_{0,3} := 1$ $number\_bounces_{1,3} := 1$
$bounce_{0,1} := 0$	$bounce_{0,2} := 1$	$bounce_{0,3} := 0$
$moving_1 := 1$	$moving_2 := 1$	$moving_3 := 0$
$\mathbf{release_1 := 1}$ $catch_1 := 0$ $moving_{sta,1} := 1$ $moving_{end,1} := 0$	$release_2 := 1$ $catch_2 := 0$ $moving_{sta,1} := 0$ $moving_{end,1} := 0$	$release_3 := 0$ $\mathbf{catch_3 := 1}$ $moving_{sta,1} := 0$ $moving_{end,1} := 1$
$moving_{dur,1} := 1.850446$	$moving_{dur,2} := 0.421875$	$moving_{dur,3} := 0$

Table 1: A plan (the assignment to the SMT variables) for the free fall problem domain. True assignments to action and event variables are shown in bold. Fix the assignments

the generator's capacity is C; and initial fuel level is F. The goal state is that the generator has been run: `(generator-ran)` for a given amount of fuel.

We will show the encoding of two happenings x_1 and x_2 – the minimum number of steps required to solve this problem. As there are no events in the domain, we will impose a bound $B = 0$ on the number of cascading events. Therefore each happening is the set:

$$x_i := \left\langle \begin{array}{ll} time & : Real \\ gen\_ran_{0,t}, gen\_ran_{1,t} & : Bool \\ fuelLevel_{0,t}, fuelLevel_{1,t}, capacity_{0,t}, capacity_{1,t} & : Real \\ generate_{sta,t}, generate_{end,t} & : Bool \\ generate_t & : Bool \\ generate_{dur,t} & : Real \end{array} \right\rangle$$

```
(define (domain simple_generator)
(:requirements
    :fluents :durative-actions
    :duration-inequalities :adl
    :typing)

(:types generator)

(:predicates
    (generator-ran)

(:functions
    (fuelLevel ?g - generator)
    (capacity ?g - generator))

(:durative-action generate
 :parameters (?g - generator)
 :duration (= ?duration 1000)
 :condition (over all (>= (fuelLevel ?g) 0))
 :effect (and
    (decrease (fuelLevel ?g) (* #t 1))
    (at end (gen-ran)))))
```

Figure 14: Simplified PDDL+ generator domain file

```
(define (problem simple_generator)
(:domain simple_generator)
(: objects gen - generator)
(:initial
    (= (fuelLevel gen) 1020)
    (= (capacity gen) 1060))

(:goal
    (gen-ran)))
```

Figure 15: Simplified PDDL+ generator problem file

The instance description enforces the initial state, the goal condition, and that the second happening is at least ε after the first (P1-P4).

$$\begin{aligned}
&\neg gen_ran_{0,1}\\
&fuelLevel_{0,1} = 1020\\
&capacity_{0,1} = 1060\\
&gen_ran_{1,2}\\
&time_1 = 0\\
&time_2 >= time_1 + 0.001
\end{aligned}$$

Proposition support constraints (between happenings) ensure that discrete state stays constant during the interval between happenings (P5-P6):

$$gen\_ran_{1,1} = gen\_ran_{0,2}$$

Invariant constraints ensure that the durative action's overall condition holds over an interval in which its associated process is active, and that the durative action's duration is properly updated across the interval

(P7-P9):

$$generate_1 \rightarrow (fuelLevel_{0,1} >= 0)$$
$$generate_2 \rightarrow (fuelLevel_{0,2} >= 0)$$
$$generate_1 \rightarrow (generate_{dur,2} = generate_{dur,1} + time_1 - time_2)$$

The continuous change over real variables is defined and enforced by the $flow$ variables (P10-P11):

$$fuelLevel_{0,2} = fuelLevel_{1,1} + \int_{time_1}^{time_2}(-1.0)dt$$
$$\neg generate_1 \rightarrow (fuelLevel_{0,2} = fuelLevel_{1,1})$$
$$capacity_{0,2} = capacity_{0,1}$$

Proposition support constraints (within happenings) and action preconditions and effects, describe the discrete changes that occurs within a happening (H1-H10):

$$gen\_ran_{1,1} \rightarrow (gen\_ran_{0,1} \vee generate_{end,1})$$
$$\neg gen\_ran_{1,1} \rightarrow \neg gen\_ran_{0,1}$$
$$gen\_ran_{1,2} \rightarrow (gen\_ran_{0,2} \vee generate_{end,2})$$
$$\neg gen\_ran_{1,2} \rightarrow \neg gen\_ran_{0,2}$$

$$fuelLevel_{0,1} = fuelLevel_{1,1}$$
$$fuelLevel_{0,2} = fuelLevel_{1,2}$$

$$generate_{sta,1} \rightarrow (fuelLevel_{0,1} >= 0)$$
$$generate_{sta,2} \rightarrow (fuelLevel_{0,2} >= 0)$$
$$generate_{end,2} \rightarrow (fuelLevel_{0,2} >= 0)$$

$$generate_{sta,1} \rightarrow (generate_{dur,1} = 1000.0)$$
$$generate_{sta,2} \rightarrow (generate_{dur,1} = 1000.0)$$
$$\neg generate_{end,1}$$
$$generate_{end,2} \rightarrow generate_1$$
$$generate_{end,2} \rightarrow (generate_{dur,2} = 0.0)$$
$$generate_{end,2} \rightarrow gen\_ran_{1,2}$$

Process triggering constraints work together to ensure that the durative action begins and ends within a happening (H11-H13):

$$generate_1 = (generate_{dur,1} > 0)$$
$$\neg generate_1 = generate_{dur,1} = 0$$
$$generate_2 = (generate_{dur,2} > 0)$$
$$\neg generate_2 = generate_{dur,2} = 0$$

Finally, action mutexes are included (H14). In our encoding, we make the starts and ends of durative actions mutually exclusive, eg. for $i = \{1,2\}$:

$$\neg gen\_start_i \vee \neg gen\_end_i$$

The resulting plan is shown by assignment to variables in Table 2. Also, the plan corresponding to the variables presented in Table 2 is shown in Figure 16.

```
0.0      : generate      [1000.0]
```

Figure 16: Plan for the generator problem.

x_1	x_2
$t_1 := 0$	$t_2 := 1000$
$gen\_ran_{0,1} := 0$ $gen\_ran_{1,1} := 1$	$gen\_ran_{0,2} := 1$ $gen\_ran_{1,2} := 0$
$fuelLevel_{0,1} := 1020.0$ $fuelLevel_{1,1} := 1020.0$ $capacity_{0,1} := 1060.0$ $capacity_{1,1} := 1060.0$	$fuelLevel_{0,2} := 20.0$ $fuelLevel_{1,2} := 20.0$ $capacity_{0,2} := 1060.0$ $capacity_{1,2} := 1060.0$
$generate_1 := 1$	$generate_2 := 0$
$generate_{sta,1} := 1$ $generate_{end,1} := 0$	$generate_{sta,2} := 0$ $generate_{end,2} := 1$
$generate_{dur,1} := 1000.0$	$generate_{dur,2} := 0$

Table 2: The SMT variables and the assignments found by SMTPlan for the simple generator domain, as assignment to the variables. Boolean variables that are true are assigned as 1 and the ones are false are 0.

6. Planning with Control Parameters as SMT

PDDL is known as an expressive modelling language, however it enforces a significant limitation on the modelling of durative actions, restricting the action parameters to choose their values from finite domains. The parameter domains for actions is specified in the problem description, enumerated as a list of objects. Using this enumeration, the action schemes can be grounded, producing a finite set of grounded actions. The only exception to finite domain parameters, introduced in PDDL2.1, is the duration of durative actions. The duration of a durative action can be chosen by the planner. This makes possible the modelling of duration dependent effects. Without the flexible duration parameter, the state space is always locally finite, which is to say that only finitely many states are reachable from a given initial state, using plans bounded by a given finite length. The addition of numeric state variables to the classical propositional language allows for the construction of an infinite state space [48].

Savas et al. [48] introduces new set of numeric action parameters called *control parameters*: a generalised version of the duration parameter, which allows the planner to choose their values from an infinite domain. The control parameters can appear both in the conditions and the effects of a durative action. Control parameters as described by Savas et al. are constrained with an upper bound and a lower bound in the domain file. The main example used by Savas et al. is a domain which includes the action of withdrawing cash from an ATM (referred to as the *cashpoint* example). Choosing the value of the withdrawal is a numeric control parameter in this action.

In this section we briefly describe how PDDL is extended to include control parameters, and later we show the extension to our SMT encoding for the domains with control parameters. We use the *cashpoint* domain as a running example throughout the section.

6.1 Cashpoint Problem

In the cashpoint problem, suppose that we want to go to a pub. Figure 17 shows the initial state and goal of the cashpoint problem. Initially we are at home and have £2 in our pocket. The aim of the problem is to be at the pub with £20 in our pocket and have purchased snacks on the way to the pub.

The amount of money that the we withdraw from the cash machine is defined as a control parameter whose value is chosen by the planner. The proposed PDDL domain with control parameters is an extended version of PDDL2.1 in which durative actions can include an additional field for control parameters.

Figure 18 illustrates the domain for this problem. The action $(withdraw\_money)$ is a durative action with the control parameter $(?cash - number)$. In this action, the control parameter is used in the effects of the action and not in the conditions. As the effect of this action, the numeric variable $(inpocket\ ?z)$ where $?z$ is the type of currency, is increased by the value of the control parameter. Similar to that the numeric variable

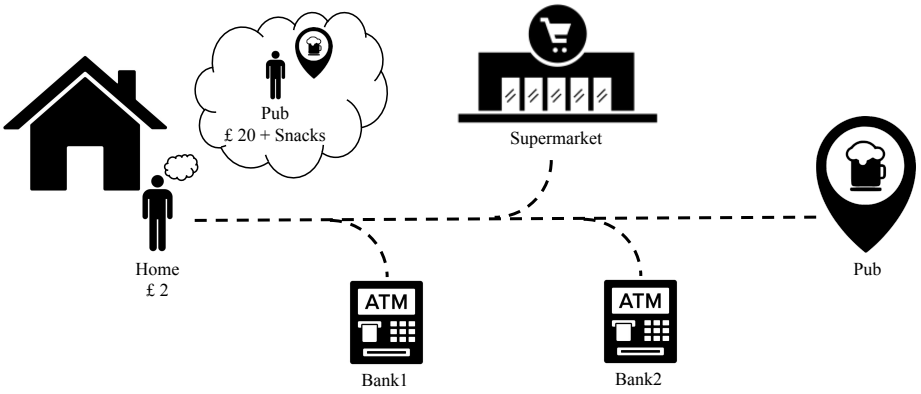


Figure 17: Initial state and goal of the cashpoint problem. The person is initially at home with £2 and the goal is to be at the pub with snacks and £20.

$(maxwithdraw\ \ ?b\ \ ?z)$ where $?b$ is the location that we withdraw money from (bank1 or bank2), has been decreased by the same value.

Figure 19 shows an example problem file for the cashpoint domain. In the problem file for this example there are five locations. Two banks, each of which has a maximum amount of £200 to withdraw, a supermarket where snacks can be purchased, the pub, and the home. Later, we use this example to show the encoding of control parameters.

6.2 Encoding of PDDL Domains with Control Parameters

In this section we describe in detail how we extend our previous encoding described in Section 4 to handle actions with control parameters. First, we discuss the modifications over the definition of a single happening by introducing control parameters and later we extend these changes to the constraints between happenings.

Following Savas et al. [48], control parameters are only defined for durative actions. An action with a control parameter, $a \in A$, is defined as:

$$a := \langle pre_a, eff_a, dur_a, cparam_a \rangle$$

where $cparam_a$ represents a finite set of numeric control parameters, where each $d^a \in cparam_a$ has a domain $dom(d^a)$. In order to use the control parameters in our encoding, we add a new set of variables to the encoding of a happening explained in Section 4.1:

$$D_t := \{d_t^a \in cparam_a, \forall a \in A\}$$

where each $d_t^a \in \mathbb{R}$ describes the value of a control parameter. The encoding of a happening at time t is therefore defined as the tuple:

$$x_t := \left\langle \begin{array}{c} time, P_{0,t}, \ldots, P_{B,t}, P_{B+1,t}\, V_{0,t}, \ldots, V_{B,t}, V_{B+1,t} \\ D_t, E_{0,t}, \ldots, E_{B,t}, Ps_t, A_t, flow_{V_t}, dur_{Ps_t} \end{array} \right\rangle$$

By introducing the control parameter variables, the constraints over a happening are also extended in the following way. As explained before, the control parameters are bounded in the domain file. First, we define two sets of constant values associated with each control parameter as the upper bounds and lower bounds. The members of set U_d are constants which define the upper bounds of control parameters at each happening (since these upper bounds will not change throughout different happenings, we do not need to define different upper bounds for each happening). Similar to the upper bounds, we define a set of lower bounds as L_d for the

```
(:durative-action withdraw_money
:parameters (?b - location ?z - currency)
:control (?cash - number)
:duration (= ?duration 2)
:condition (and (over all (at ?b))
                (at start (>= ?cash 5))
                (at start (<= ?cash 100))
                (at start (>= (maxwithdraw ?b ?z) 0 ))
                (at end (>= (maxwithdraw ?b ?z) 0))
                (at start (available))
                (at start (canwithdraw_money ?b))
)
:effect (and
                (at start (not (available)))
                (at start (increase (inpocket ?z) ?cash))
                (at start (decrease (maxwithdraw ?b ?z) ?cash))
                (at end (available))
        ))


(:durative-action buy_snacks
:parameters (?a - location ?z - currency)
:duration (= ?duration 1)
:condition (and     (at start (at ?a))
                    (over all (at ?a))
                    (at start (available))
                    (at start (>= (inpocket ?z) 5))
                    (at end (>= (inpocket ?z) 0))
                    (at start (canbuy ?a ?z)) )
:effect (and    (at start (decrease (inpocket ?z) 5))
                (at start (not (available)))
                (at end (gotsnacks))
                (at end (available))
        ))

(:durative-action check_pocket
:parameters (?z - currency)
:duration (<= ?duration 0.5)
:condition (and     (at start (>= (inpocket ?z) 100) )
                    (at start (available))
                    (at end (>= (inpocket ?z) 0) )  )
:effect (and
                    (at start (not (available)))
                    (at end (have_enough ?z))
                    (at start (decrease (inpocket ?z) 100))
                    (at end (available)) ) )

)
```

Figure 18: Main actions of the cashpoint domain

control parameters. Figure 20 shows the set of constraints within a happening in the case of having control parameters in the domain. The changes are shown in bold compared to Figure 8.

Within a happening, the structure of most of the constraints remain the same. However we need to add the constraints associated with the lower and upper bounds of control parameters (shown in equations $H15$ and $H16$). These constraints state that when the action is applied, the control parameter is bounded by its constant upper and lower bounds $u_d \in U_d$ and $l_d \in L_d$, respectively.

Moreover, control parameters can appear in the conditions or effect of a durative action. Therefore $H9$ and $H10$ are also affected by control parameters. These constraints model the conditions and effects of an action, which can include expressions involving control parameter variables. The control parameter variables $d_t \in D_t$ can be used directly in these constraints.

Adding control parameters to durative actions also affects the encoding of the constraints between the happenings explained in section 4.2. Figure 21 shows these equations. The changes compared to Figure 10 are shown in bold.

The only constraint added to the set of encodings regarding the constraints between the happenings is control parameter support constraint ($P12$). This constraint ensures that the value of a control parameter

```
   (:objects pub supermarket home - location
           bank1 bank2 - location
           pounds - currency)

   (:init (at  home)
        (canbuy supermarket pounds)
        (canwithdraw_money bank1)
  (available)
        (= (maxwithdraw bank1 pounds) 200)
        (= (maxwithdraw bank2 pounds) 200)
        (= (inpocket pounds) 2)
 )

 (:goal (and (have_enough pounds) (gotsnacks) (at pub) (>= (inpocket pounds) 20) ))
)
```

Figure 19: An example problem file for the cashpoint domain.

stays constant during the execution of the action. In P12, $dur_{ps_i^a}$ refers to the duration variable of the action a containing the control parameter.

6.3 Encoding of the cashpoint problem

Considering the domain shown above, we introduce a variable cp_cash_t for each happening as the control parameter associated with the durative action. In this domain we only have one durative action with control parameter:

$$(withdraw\_money\ \ bank1\ \ pounds)$$

The numeric variables associated with this action $(maxwithdraw\ \ bank1\ \ pounds)$ and $(inpocket\ \ pounds)$ are affected by the control parameter variable. Constraint $H10$ is modelled as the following constraint, which illustrates the effect of the control parameters on the numeric variables.

$$\begin{gathered}(withdraw\_money\, bank1\, pounds)_{sta,t} \rightarrow \\ (maxwithdraw\, bank1\, pounds)_{1,t} = ((maxwithdraw\, bank1\, pounds)_{0,t} - cp\_cash_0) \\ \wedge \\ (withdraw\_money\, bank1\, pounds)_{sta,t} \rightarrow \\ (inpocket\, pounds)_{1,t} = ((inpocket\, pounds)_{0,t} + cp\_cash_0)\end{gathered}$$

The upper bound u_d is 100 and the lower bound l_d is 5. Using these bounds we can write the equations $H15$ and $H16$ as :

$$\begin{gathered}(withdraw\_money\, bank1\, pounds)_{sta,t} \rightarrow (cp\_cash_0) >= 5 \\ (withdraw\_money\, bank1\, pounds)_{sta,t} \rightarrow (cp\_cash_0) <= 25\end{gathered}$$

The plan found by the SMTPlan for this domain and problem file is shown in Figure 22. The reason that the planner suggests to withdraw £99.5, is that the action $check\_pocket$ has a precondition $(inpocket\ \ ?z) >= 100$ and the initial value for $(inpocket\ \ pounds)$ is 2.

Table 3 shows part of a satisfying assignment to an encoding of a problem in the cashpoint domain. The assignment corresponds to the plan shown in Figure 22, and displays only the second and third happenings. The second happening occurs 7 time units after starting the plan, and the duration of interval between the two happenings 2 time units.

7. Experimental Evaluation

In this section we present our experimental results in four parts.

1. In Section 7.1 we compare the performance of SMTPlan against other PDDL+ planners on PDDL+ benchmarks for hybrid systems domains. We aim to show that in problems with non-linear continuous change, and a small number of happenings, SMTPlan performs very well.

Proposition and real variable support

H1. $\bigwedge_{p_{0,t} \in P} p_{1,t} \rightarrow (p_{0,t} \vee \bigvee_{e_{0,t} | p_{0,t} \in eff^+_{e_{0,t}}} e_{0,t} \vee \bigvee_{a_t | p_{0,t} \in eff^+_{a_t}} a_t)$

H2. $\bigwedge_{p_{i,t} \in P} \neg p_{1,t} \rightarrow (\neg p_{0,t} \vee \bigvee_{e_{i,t} | p_{i,t} \in eff^-_{e_{i,t}}} e_{0,t} \vee \bigvee_{a_t | p_{i,t} \in eff^-_{a_t}} a_t)$

H3. $\bigwedge_{i=1}^{B+1} \bigwedge_{p_{i,t} \in P} p_{i+1,t} \rightarrow (p_{i,t} \vee \bigvee_{e_{i,t} | p_{i,t} \in eff^+_{e_{i,t}}} e_{i,t})$

H4. $\bigwedge_{i=1}^{B+1} \bigwedge_{p_{i,t} \in P} \neg p_{i+1,t} \rightarrow (\neg p_{i,t} \vee \bigvee_{e_{i,t} | p_{i,t} \in eff^-_{e_{i,t}}} e_{i,t})$

H5. $\bigwedge_{v_{i,t} \in V} (\bigwedge_{a_t | v_{i,t} \in eff^{num}_{a_t}} \neg a_t \wedge \bigwedge_{e_{i,t} | v_{i,t} \in eff^{num}_{e_{i,t}}} \neg e_{0,t}) \rightarrow (v_{i+1,t} = v_{i,t})$

H6. $\bigwedge_{i=1}^{B} \bigwedge_{v_{i,t} \in V} (\bigwedge_{e_{i,t} | v_{i,t} \in eff^{num}_{e_{i,t}}} \neg e_{i,t}) \rightarrow (v_{i+1,t} = v_{i,t})$

Event preconditions and effects

H7. $\bigwedge_{i=0}^{B} \bigwedge_{e_{i,t} \in E} e_{i,t} \leftrightarrow pre_{e_{i,t}}(P_i \cup V_i)$

H8. $\bigwedge_{i=0}^{B} \bigwedge_{e_{i,t} \in E} e_{i,t} \rightarrow eff_{e_{i,t}}(P_{i+1} \cup V_{i+1})$

Action preconditions and effects

H9. $\boldsymbol{\bigwedge_{a_t \in A} a_t \rightarrow pre_{a_t}(P_{0,t} \cup V_{0,t})}$

H10. $\boldsymbol{\bigwedge_{a_t \in A} a_t \rightarrow eff_{a_t}(P_{1,t} \cup V_{1,t})}$

Process triggering

H11. $\bigwedge_{ps_t \in Ps} ps_t \leftrightarrow pre_{ps_t}(P_{B+1,t} \cup V_{B+1,t})$

H12. $\bigwedge_{ps_t \in Ps} dur_{ps_t} >= 0$

H13. $\bigwedge_{ps_t \in Ps} ps_t \leftrightarrow (dur_{ps_t} > 0)$

Action mutexes

H14. $\bigwedge_{a_t \in A} \bigwedge_{a'_t \in A | a_t \nparallel a'_t} (\neg a_t \vee \neg a'_t)$

Bounds on control parameters

H15. $\boldsymbol{\bigwedge_{d^a_t \in D_t} a_t \rightarrow d^a_t <= u_d}$

H16. $\boldsymbol{\bigwedge_{d^a_t \in D_t} a_t \rightarrow d^a_t >= l_d}$

Figure 20: Reduction of a PDDL+ happening with control parameters to SMT.

2. In Section 7.2 we investigate the scalability of SMTPlan in PDDL+ domains in more detail.

3. In Section 7.3 we compare SMTPlan with temporal planners from the International Planning Competition (IPC) on a set of temporal planning benchmarks used in the IPC. In the experiment we explore the limitations inherent in the Satisfiability approach in terms of scalability with respect to the size of the discrete state-space.

4. Finally, in Section 7.4 we evaluate the effect of control parameters on our encoding, comparing the effectiveness of SMTPlan on two versions of the same domain, with and without control parameters.

Instance description

P1. $I(P_{0,1} \cup V_{0,1})$

P2. $G(P_{B+1,n} \cup V_{B+1,n})$

P3. $time_0 = 0$

P4. $\bigwedge_{i=2}^{n} time_i \geq time_{i-1} + \varepsilon$

Proposition support

P5. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} p_{0,i} \rightarrow p_{B+1,i-1}$

P6. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} \neg p_{0,i} \rightarrow \neg p_{B+1,i-1}$

Invariants

P7. $\bigwedge_{i=2}^{n} \bigwedge_{ps_i \in Ps_i} ps_{i-1} \rightarrow dur_{ps_i} = dur_{ps_{i-1}} - time_i + time_{i-1}$

P8. $\bigwedge_{i=1}^{n-1} \bigwedge_{ps_i \in Ps_i} ps_i \leftrightarrow pre_{\leftrightarrow ps_i}$

P9. $\bigwedge_{i=1}^{n-1} \bigwedge_{e \in E} \neg pre_{\leftrightarrow e_i}$

Continuous change on real variables

P10. $\bigwedge_{i=1}^{n-1} \bigwedge_{v \in V} (flow_v)_i = \int_{time_i}^{time_{i+1}} \sum_{ps \in Ps} (\mathit{eff}^{num}_{\leftrightarrow ps}[v])_i dt$

P11. $\bigwedge_{i=2}^{n} \bigwedge_{v \in V} (v_{0,i} = v_{B+1,i-1} + (flow_v)_{i-1})$

Control parameter support

P12. $\boldsymbol{\bigwedge_{i=1}^{n-1} \bigwedge_{a_t \in A} dur_{ps_i^a} \rightarrow d_i = d_{i+1}}$

Figure 21: Reduction of PDDL+ planning problem with control parameters II+ to SMT.

```
0.0: (goto home bank1) [5.0]
7.0: (withdraw_money bank1 pounds) [2.0] (?cash [24.0])
11.0: (withdraw_money bank1 pounds) [2.0] (?cash [99.5])
15.0: (check_pocket pounds) [0.25]
17.25: (goto bank1 supermarket) [5.0]
24.25: (buy_snacks supermarket pounds) [1.0]
27.25: (goto supermarket pub) [5.0]
```

Figure 22: The plan found by the SMTPlan for the cashpoint domain

7.1 PDDL+ Benchmarks

We use our encoding to solve PDDL+ planning problems with a parallel iterative deepening technique, widely used in SAT-based planning approaches [37, 46]. The top-level algorithm encodes and solves n SMT encod-

x_2	x_3
$t_2 := 7$	$t_3 := 9$
$available_{0,2} := 1$ $available_{1,2} := 0$	$available_{0,3} := 0$ $available_{1,3} := 1$
$maxwithdraw_{0,2} := 200$ $maxwithdraw_{1,2} := 176$ $inpocket_{0,2} := 0$ $inpocket_{1,2} := 24$	$maxwithdraw_{0,3} := 176$ $maxwithdraw_{1,3} := 176$ $inpocket_{0,3} := 24$ $inpocket_{1,3} := 24$
$withdraw\_money_1 := 1$	$withdraw\_money_2 := 0$
$withdraw\_money_{sta,2} := 1$ $withdraw\_money_{end,2} := 0$	$withdraw\_money_{sta,2} := 0$ $withdraw\_money_{end,2} := 1$
$withdraw\_money_{dur,1} := 2$	$withdraw\_money_{dur,2} := 0$

Table 3: Assignment to numeric and propositional variables associated with the $withdraw\_money$ action in the cashpoint domain. Boolean variables that are true are assigned as 1 and the ones are false are 0.

ings simultaneously, solving the planning problem for horizon lengths 1,2,3,4,...,n. In this case, horizon length corresponds to number of happenings. If a formula is found satisfiable, then a plan has been found and the planner terminates. If a formula is found unsatisfiable, then an encoding is made for the next shortest horizon length, so that there are always n SMT instances being solved. The SMT solver we use is z3 [14].

We compare our approach (called SMTPlan) against existing PDDL+ planner UPMurphi [16], and with dReach [6], using the SMT solver dReal [23], on domains both with and without events[2]. We use the *generator* and *car* domains [5]. The experiments were run using 8GB of RAM and a 30 minute timeout. All test domains and problems are available at: *kcl-planning.github.io/SMTPlan*.

The generator domain is a PDDL+ benchmark problem that revolves around refueling a diesel-powered generator, which has to run for a given duration without overflowing or running dry. To test scalability the number of tanks is increased while decreasing the initial fuel level.

We consider four versions of this domain: linear, simplified-nonlinear (the same used in [6]), nonlinear with events, and the Torricelli nonlinear [25]. Note that the latter version uses the Torricelli's Law (which is too complex for dReach), and hence the fuel level in a refueling tank (V_{fuel}) is calculated by:

$$V_{fuel} = (-kt_r + \sqrt{V_{init}})^2 \qquad t_r \in \left[0, \frac{\sqrt{V_{init}}}{k}\right] \tag{1}$$

where V_{init} is the initial volume of fuel in the tank, k is the fuel flow constant (which depends on gravity, size of the drain hole, and the cross-section of the tank), and t_r is the time of refueling (bounded by the fuel level and the flow constant). An example of plan found by SMTPlan for the Torricelli nonlinear generator (Fuel level 960, Generator capacity 990) is shown in Figure 23:

```
0.0: generate     [1000.0]
959.0: refuel_tank1 [10.0]
959.0: refuel_tank2 [10.0]
```

Figure 23: Plan for a problem in the Toricelli Generator domain.

The car domain is another PDDL+ benchmark [20] where a vehicle has to cover a given distance and have a zero velocity at the end, and the actions available are accelerate and decelerate that increments or decrements by 1 the current velocity, respectively. To test scalability, the bound on maximum acceleration/deceleration is increased.

Our results for solvable instances are reported in Table 4. On both linear and nonlinear domains, SMTPlan outperforms all other planners in time to solve and in number of instances solved. In all domains, SMTPlan scales very well. For these domains, the number of happenings required is small, thus the minimal SMT encoding required to solve the problem is also small. The iterative deepening algorithm is able to reach a satisfiable encoding, and produce a plan very quickly. The offline computation with SymPy is required only once per domain, and in all cases required 0.3 seconds or less.

dReach also performs iterative deepening, but performs more poorly. This is due to the semantics of dReach; in the dReach domain and problem description, each mode of continuous change must be explicitly defined, and the number of modes increases exponentially with the number of processes and durative actions (eg. the files for 1, 2, 3 and 4 tanks problems are respectively 91, 328, 1350, 5762 lines long). Furthermore, the bound is not on the number of happenings, but on the number of mode changes, which does not allow for parallel execution of actions.

Moreover, dReach does not perform integration and differentiation outside of the solver, during encoding. Instead it relies upon the more expressive logic of the internal SMT solver, dReal, at the cost of extra computation time. In addition, they are unable to use SMT solvers other than dReal.

We also compare our encoding directly against dReach as reported in prior work [6]: reporting times to solve only the encoding of a minimal step plan for each instance (table 6). These are not the times required to

2. We considered domains without events as we wanted to show the comparison with dReach which does not handle domains including events

solve a PDDL+ instance, but a direct comparison of encodings on satisfiable problems. We find the encodings exhibit similar performance in the car domain. However, we find the SMTPlan encoding scales far better on the generator problem, as discussed above. Moreover, the SMTPlan encoding does not require the advanced features of dReal, and can be solved more quickly using z3.

Our results for unsolvable instances are shown in table 5. SMTPlan and dReach can only prove unsolvability up to an upper bound on the number of happenings. Here we prove plan non-existence for domains which have a tight deadline, and where each ground action can only be applied a finite number of times. We also include SpaceEx that can be used to prove plan-non existence for the generator linear domain [5]. We observe that both totally ordered planning approaches perform well proving unsolvability in the car domain. There are few choices of symbolic plan in this domain, leaving only the timing of the happenings and numeric constraints to be solved. Both SMTPlan and dReach solve these constraints very quickly. However, for PDDL+ problems in general, without deadlines and with repeatable actions, proving unsolvability is difficult through totally ordered planning with iterative deepening.

Domain	Tool	1	2	3	4	5	6	7	8
Generator linear	SMTPlan	0.02	0.03	0.02	0.01	0.02	0.02	0.02	0.02
	dReach	2.87	-	-	-	-	-	-	-
	UPMurphi	0.2	18.2	402.34	-	-	-	-	-
Generator nonlinear	SMTPlan	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
	dReach	5.16	-	-	-	-	-	-	-
	UPMurphi	63.16	-	-	-	-	-	-	-
Generator nonlin. events	SMTPlan	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.05
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	658.18	-	-	-	-	-	-	-
Generator Torricelli	SMTPlan	0.03	0.03	0.15	0.92	0.04	0.05	0.09	0.50
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	63.16	-	-	-	-	-	-	-
Car	SMTPlan	0.02	0.02	0.02	0.02	0.02	0.02	0.01	0.02
	dReach	1.30	1.41	1.48	1.53	1.47	1.54	1.40	1.53
	UPMurphi	28.44	386.5	-	-	-	-	-	-

Table 4: Results in seconds for solvable instances. Instance numbers correspond to number of tanks (generator) and number of acceleration steps (car). Abbrev.: -: tool still running after 30 minutes, '.': tool ran out or memory, x: tool cannot handle the problem.

7.2 Free Fall and Generator

Furthermore we designed two new sets of experiments. The first experiment evaluates the effects of having multiple objects in the domain file.

For this mean we have expanded the *Free Fall* problem and considered problem instances where we have more than one ball in our problem file.

We considered two scenarios for these problem instances. In both scenario we increase the number of balls in the problem file. We create cases that include 25,50 ,100 and 200 balls. In the first scenario we try to find a plan for a set of problem instances while the goal state is to catch just the first ball. As we can see in Figure 24, as we increase the number of balls in the problem file, the time needed to solve the problem increases linearly. However, by increasing the number of balls, the search space grows exponentially.

Domain	Tool	1	2	3	4	5	6	7	8
Generator linear	SMTPlan	0.01	0.02	0.16	2.84	390.86	-	-	-
	dReach	2.57	189.94	-	-	-	-	-	-
	UPMurphi	0.90	29.42	-	-	-	-	-	-
Generator nonlinear	SMTPlan	0.01	1.95	33.48	-	-	-	-	-
	dReach	2.43	212.43	-	-	-	-	-	-
	UPMurphi	-	-	-	-	-	-	-	-
Generator nonlin. events	SMTPlan	0.02	18.58	21.83	-	-	-	-	-
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	-	-	-	-	-	-	-	-
Generator Toricelli	SMTPlan	0.03	2.06	19.57	-	-	-	-	-
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	-	-	-	-	-	-	-	-
Car	SMTPlan	0.68	0.02	0.00	0.00	0.00	0.00	0.00	0.01
	dReach	0.67	0.50	0.62	0.45	0.58	0.57	0.49	0.65
	UPMurphi	36.01	445.23	-	-	-	-	-	-

Table 5: Results in seconds for unsolvable instances. Instance numbers correspond to number of tanks (generator) and number of acceleration steps (car). Abbrev.: -: tool still running after 30 minutes, x: tool cannot handle the problem.

Domain	Tool	1	2	3	4	5	6	7	8
Generator linear	SMTPlan	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02
	dReach	2.73	13.47	104.61	695.70	-	-	-	-
Generator nonlinear	SMTPlan	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	dReach	10.42	1685.35	-	-	-	-	-	-
Car	SMTPlan	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	dReach	0.77	0.76	0.76	0.76	0.76	0.76	0.77	0.76

Table 6: Results in seconds for minimal step encoding required to solve each instance.

SMTPlan focuses on variables related to the objects from the problem file that are involved in the goal state and tries to find the assignments for them. In other word, it also propagates the search branches regarding the variables that are representing other objects that are not in the goal state.

In the second scenario we continued the experiments by changing the goal state to catch all the balls in the problem file (we considered all the balls have the same initial height and we want to catch all of them at a certain height). Figure 24 indicates the results of catching all the balls in the cases we used before. As it is shown the trend of the total time needed to solve the problem is linear. All the plans obtained from SMTPlan suggest to release all the balls together and also catch all of them at the same time point. This is resulted from the fact that SMTPlan tries to solve the problem with minimum number of happenings. Similar to the previous case, the search space grows exponentially, however this growth affects our solving time linearly as the SMTPlan tries to solve the problem with minimum number of happenings [3].

3. As it is mentioned in Section 4.3, SMTPlan starts to find a plan with an initial number of happening which can be defined for it (otherwise it starts with two happening. If it can not find a plan it increases the number of happenings and this increase can be by

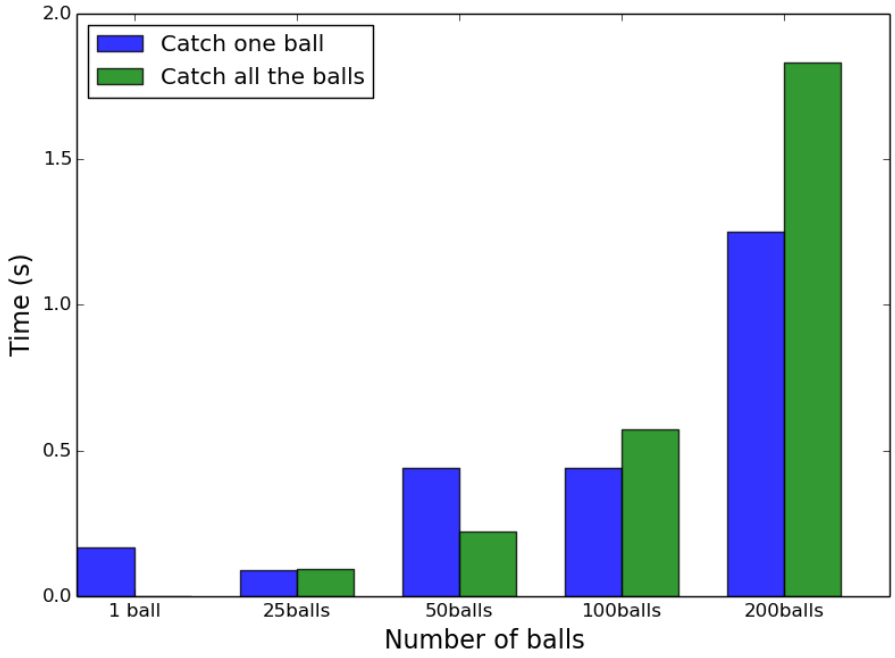


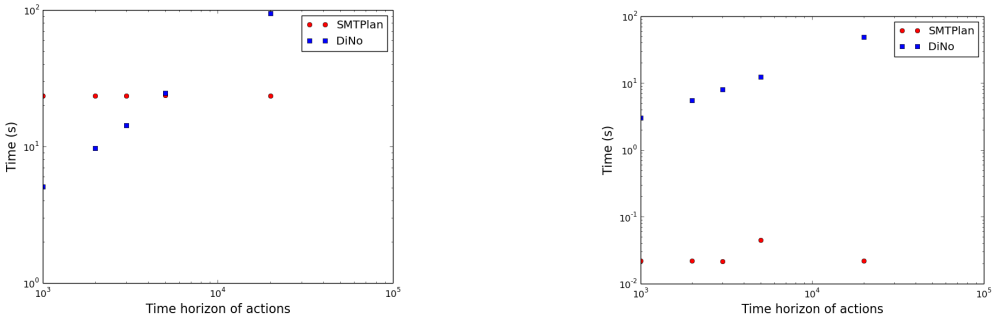
Figure 24: The bar chart compares the time taken for SMTPlan to solve the free fall problem. The blue bars indicate the scenarios that we have 1,25,50,100 and 200 balls and the aim of the problem is to just catch the first call. The green bars represents the scenarios that we have 25,50,100 and 200 balls and the goal is to catch all of them.

We also tested the performance of the SMTPlan when we have domains with actions that have longer time horizon. For this mean we used two different generator domains and generated a number of problems when the capacity of the generator and the running time of it increase gradually. In the initial problem the running time of the generator is $1{,}000$ seconds, however we tested problems where the generator has running times of$2{,}000$, $3{,}000$, $5{,}000$ and $20{,}000$ time units. As we wanted to test the effects of the time horizon on the planner, we also changed the initial amount of the fuel in the tank, so the number of times that we need to apply the re-fuelling action remains the same. In other words, just the length of the re-fuelling process has been extended. As you can see in Figures 25a and 25b, we have done the test both using SMTPlan and DiNo. SMTPlan has shown that the time to find a plan is not related to the time horizon of the actions (as we increase the time horizon the time needed to to find the plan for SMTPln remains the same). However, as we increase the time horizon, DiNo needs longer time to solve the problem. This can be explained by the fact that, DiNo needs to discretised the time and as we have a longer time horizon the search space grows and the planner needs more time to find the plan. On the other hand for the SMTPlan, since the number of the happenings are the same, increasing the time horizon won't change anything for the SMTPlan and for this reason it is able to find the plan in the same time.

7.3 Temporal Domains

For the sake of completeness, we also tested the SMTPlan on the temporal domains that are used in the temporal track at ICAPS 2018. The results are shown in Table 7. We compared the SMTPlan with the other

one or a step size that can be defined for the planner. If we run the SMTPlan with the initial number of happenings of two and define that step size as one, the planner finds a planner with the minimum number of happenings, if the problem is solvable.



(a) (b)

Figure 25: Generator domain with (a)events/ (b)linear continuous changes with different time horizons

planners participated in this track. These planners are CP4TP [22], TFLAP [38], TemPorAl [27], PopCorn [17] and OPTIC [13].

The time given to each planner was 30 minutes and 8GB of memory. In total 10 domains have been chosen and for each domain 10 problem instances have been selected. Each number in the table indicates the number of problem instances that each planner has managed to solve. At each raw, for each domain, the maximum number of problems that is solved is shown in bold.

		Planners					
Domain	N	SMTPlan	CP4TP	TFLAP	TemPorAl	PopCorn	OPTIC
Airport	10	1	9	9	**10**	3	3
Cushing	10	**10**	**10**	3	0	1	**10**
Floortile	10	0	2	3	**10**	0	0
Map Analyser	10		**10**	8	**10**	0	0
Parking	10	0	**10**	**10**	**10**	4	8
Quantum	10	1	**10**	8	**10**	5	8
Road Traffic	10	0	7	0	**10**	0	0
Sokoban	10	0	**6**	4	**6**	1	1
Trucks-time-strips	10	0	**10**	**10**	**10**	9	**10**
Total	100	12	74	55	**76**	23	40

Table 7: Results in the number of problems that each planner could solve for the domains from ICAPS 2018 temporal tracks

7.4 Domains with Control Parameters

In this section we have designed a set of experiments to show how introducing control parameters affects the performance of SMTPlan. We used the *cashpoint* domain for this experiments. We did the same experiments using two different types of the same domain; one domain including control parameter variables and the other one without. For both domains we considered a set of problem instances that get more difficult gradually. We increased the difficulty of the problems in two ways: first, by adding new objects in the problem file; and second, by considering scenarios in which we have more actions in the domain file and in order to reach the goal we need to apply these new actions.

First we create a set of problem instances. The goal of all these problem files is to have a specific amount of money, which is indicated by the variable $(inpocket\ \ pounds)$ in the goal state. We have started from $(= (inpocket\ \ pounds)10)$ and with increments of 10 unites and it reaches to $(= (inpocket\ \ pounds)100)$.

Moreover, in order to make more difficult problems, the number of objects in each problem file is increased. We first start the problems with just one bank, then increase them to 6 banks. For all the experiments in this section, we have used these set of benchmarks.

We also added more propositions to the goal state, which leads to plans with more actions. The goal of the first scenario is just to withdraw money. Then, we gradually make increase the difficulty of the domain by adding new actions. The next considered scenario is a domain in which we need to go from home to bank and then withdraw money. For the third case we considered that we need to go to the pub at the end and in the last scenario we need to go the supermarket and get some snacks as well.

The result for these experiments are shown in 26. The y axis shows the results of the experiments for domains without control parameters and the x axis is for domains with control parameters. The time given to the solver for these set of experiments was 3600 seconds. As we can see on the graph, the time to solve the problems without control parameters are much longer compare to the domains with control parameters.

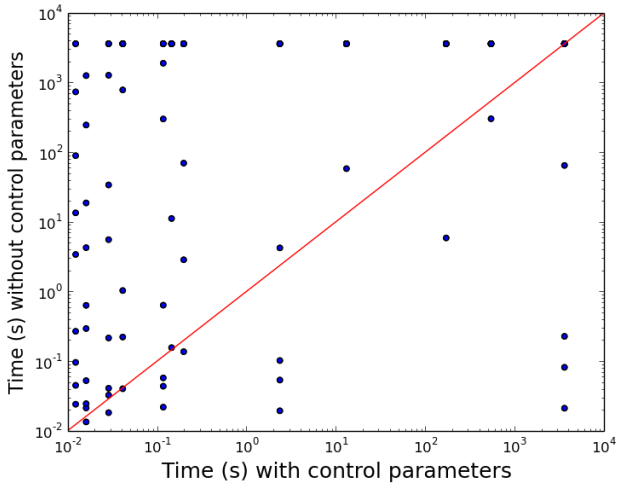


Figure 26: The results of running SMTPlan for cashpoint example. The experiment compares the time taken to solve the cashpoint problems using two domains: (a) the domain without having control parameter variables (b) same domain with durative action which have control parameters

Moreover, the number of problems that domain's with control parameters can solve is compared in Table 8.

Domain	Total number of problems	Planner	
		SMTPlan with control parameters	SMTPlan without control parameters
Domain0	60	50	39
Domain1	40	30	7
Domain2	30	20	4
Domain3	20	10	1

Table 8: Comparison of number of domains that can be solved by SMTPlan that has control parameters and without control parameters

8. Related Work

In this section we discuss the related work. First in section 8.1 we describe the preceding work in planning as satisfiability, and then in section 8.2 we place our work with resepect to other approaches to planning in hybrid domains.

8.1 Planning as Satisfiability

Planning as Satisfiability was pioneered by Kautz and Selman, beginning with a translation from Planning into propositional satisfiablity (SAT) [30], and the planner SATPlan [33]. Since then there have been many contributions improving the effectiveness of planning as SAT, including alternate encodings of the state or transition relation; the embedding of additional planning-specific knowledge such as heuristic evaluation; or planning-specific improvements to the SAT solver. Many of these ideas are orthogonal to the choice of representation between SAT and SMT. There are relatively few approaches to planning as SMT. The DREACH planner plans a subset of PDDL+ using the DREAL SMT solver for ODEs, and TM-LPSAT uses an encoding of propositional and numeric variables with linear constraints solved by the LPSAT engine. However, neither of these approaches produce SMT problems that can be solved by general SMT solvers.

Lifted causal encodings, first introduced by Kautz et al. [29], inspired by the lifted version of the SNLP causal link planner of McAllester and Rosenblitt [36] differ from the state-based encodings in that there is no proper notion of a state. The lifted encoding encodes a propositional planning problem as lifted SAT, which is then reduced to SAT. The encoding includes an assignment of action to plan steps, and the a valid causal ordering between plan steps. While it proved less effective in propositional planning with SAT, this work is applicable using the first-order expressions of SMT in a partially-ordered happening-based encoding.

Rintanen et al. have greatly advanced the state-in-the-art for planning as satisfiability, including new semantics for plan steps [46]; embedding planning heuristics in the SAT solver [44], such as the ”helpful actions” heuristic in the planner MADAGASCAR [45], and other top-level search strategies that improve over the iterative deepening used by SATPlan and SMTPlan [43]. A shared purpose of these advancements is to help the SAT solver scale to the large discrete search space present in most planning problems. These approaches are orthogonal to the choice of encoding formalism (SAT or SMT), and can be applied directly in SMTPlan.

The planner ITSAT [42] is a SAT-based planner for non-numeric temporal planning problems. The planner extends the step semantics introduced by Rintanen et al. [46] to temporally abstract the problem without losing the ability to express concurrent activities.

8.2 Planning in Hybrid Domains

Various techniques and tools have been proposed to deal with hybrid domains. ZENO [39] is a planner which can handle actions occurring over extended intervals of time. ZENO is able to reason about goals with deadlines, piece-wise linear continuous change, external events and to a limited extent, simultaneous actions.

More recent approaches in this direction have been proposed by Bologomov et al. [5], where the close relationship between hybrid planning domains and hybrid automata is explored, and [6] where hybrid domains are handled using SMT. DREACH [6], a planner for hybrid systems uses the dReal solver [23], a non-linear SMT solver that uses its own theory of ODEs. Input is provided in the language of DREACH as opposed to PDDL+, and hybrid problems have to be manually encoded. This language can only handle a restricted subset of the language features contained in PDDL+. In particular, it cannot handle exogenous events.

More similar to the approach of SMTPlan is the planner TM-LPSAT [49]. The planner is able to solve problems with atomic and durative actions, processes, events, and linear change. TM-LPSAT uses a happening-based encoding, containing propositional and numeric variables and linear constraints. This is solved by the LPSAT constraint engine. While SMTPlan is able to handle polynomial non-linear change, TM-LPSAT is restricted to continuous linear change; the LPSAT solver requires only linear constraints, and the encoding does not account for the zero-crossing problem introduced by non-linear change.

Many works have been proposed in the model checking and control communities to handle hybrid systems [11, 9, 12, 51, 35], including sampling-based planners [28, 34]. Another related direction is *falsification* of hybrid systems [41] (i.e., guiding the search towards the error states, that can be easily cast as a planning problem). However, while all these works aim to address a similar problem, they cannot be used to handle PDDL+ models. Bogomolov et al. [5, 4] are working towards a formal translation between PDDL+ and standard hybrid automata, but so far only an over-approximation has been defined, which allows the use of those tools only for proving plan non-existence.

To date, the only viable approach to general PDDL+ planning is via discretisation. UPMURPHI [16], which implements the discretise-and-validate approach, is able to deal with the full range of PDDL+ features. Discretise and validate works by first discretising time into small steps, solving the problem, and validating the result against the original continuous domain. If the plan is not valid with respect to the continuous semantics, then a finer discretisation is generated and the process iterates. However, UPMURPHIperforms blind search, which limits its scalability.

More recent work in this direction is the planner, DINO. Similar to UPMURPHI, DINO is also based on the discretise-and-validate approach and uses the novel Staged Relaxed Planning Graph+ (SRPG+) heuristic [40] to help cope with the scalability issues faced by UPMURPHI.

Considering the related works mentioned above, the SMT encoding is able to capture all features of PDDL+ and works by directly translating standard PDDL+ domain and problem files. Furthermore, it correctly captures the *must* semantics of PDDL+ (which constrains how processes and events interact with each other and with actions). Also, SMTPlan models the precise semantics of ε-separation of effects and action preconditions [20].

9. Conclusion

In this paper we presented a new approach for PDDL+ planning that can handle the whole set of PDDL+ features and respects Fox and Longs semantics. We proposed an SMT encoding of PDDL+ domains that correctly captures the must semantics of PDDL+ which constrains how processes and events interact with each other and with actions. The encoding is general and can be used with any SMT solver in the theory of quantifier-free nonlinear arithmetic.

Experimental results show that the approach dramatically outperforms existing work in finding plans for solvable problems, and it is efficient also in proving plan-non-existence.

References

[1] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[2] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[3] Francesco Biscani, Isuru Fernando, Dario Izzo, Sumith Kulal, 7ofNine, Ondej ertk, and Jean-Paul Pelteret. bluescarni/piranha: piranha 0.11, May 2018.

[4] Sergiy Bogomolov, Daniele Magazzeni, Stefano Minopoli, and Martin Wehrle. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *Proceedings of ICAPS*, 2015.

[5] Sergiy Bogomolov, Daniele Magazzeni, Andreas Podelski, and Martin Wehrle. Planning as model checking in hybrid domains. In *Proceedings of AAAI*, 2014.

[6] Daniel Bryce, Sicun Gao, David J. Musliner, and Robert P. Goldman. SMT-based nonlinear PDDL+ planning. In *Proceedings of AAAI*, 2015.

[7] Joshua Campion, Chris Dent, Maria Fox, Derek Long, and Daniele Magazzeni. Challenge: Modelling unit commitment as a planning problem. In *Proceedings of ICAPS*, 2013.

[8] Michael Cashmore, Maria Fox, and Enrico Giunchiglia. Planning as quantified boolean formulae. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, 2012.

[9] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv symbolic model checker. In *Proceedings of CAV*, 2014.

[10] Yixin Chen, Zhao Xing, and Weixiong Zhang. Long-distance mutual exclusion for propositional planning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1840–1845, 2007.

[11] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. HyComp: An SMT-based model checker for hybrid systems. In *Proceedings of ETAPS*, 2015.

[12] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. SMT-based verification of hybrid systems. In *Proceedings of AAAI*, 2012.

[13] Andrew Coles and Amanda Coles. Optic. IPC 2018 - Temporal Tracks.

[14] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proceedings of TACAS*, 2008.

[15] Giuseppe Della Penna, Benedetto Intrigila, Daniele Magazzeni, and Fabio Mercorio. A PDDL+ benchmark problem: The batch chemical plant. In *Proceedings of ICAPS*, 2010.

[16] Giuseppe Della Penna, Daniele Magazzeni, and Fabio Mercorio. A universal planning system for hybrid domains. *Applied Intelligence*, 36(4):932–959, 2012.

[17] Maria Fox Derek Long Daniele Magazzeni Emre Savas, Stefan Edelkamp. Temporal-numeric planning with infinite domain action parameters. IPC 2018 - Temporal Tracks.

[18] Maria Fox and Derek Long. Pddl+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, volume 4, page 34, 2002.

[19] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Res. (JAIR)*, 20:61–124, 2003.

[20] Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *Jorunal of Artificial Intelligence Research (JAIR)*, 27:235–297, 2006.

[21] Maria Fox, Derek Long, and Daniele Magazzeni. Automatic construction of efficient multiple battery usage policies. In *Proceedings of IJCAI*, 2011.

[22] Daniel Furelos-Blanco and Anders Jonsson. Cp4tp: A classical planning for temporal planning portfolio. IPC 2018 - Temporal Tracks.

[23] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. Delta-complete decision procedures for satisfiability over the reals. In *Proceedings of IJCAR*, 2012.

[24] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of IEEE Symposium on Logic in Computer Science*, 1996.

[25] Richard Howey and Derek Long. VAL's progress: The automatic validation tool for PDDL2.1 used in the international planning competition. In *Proc. of ICAPS Workshop on IPC*, 2003.

[26] Richard Howey, Derek Long, and Maria Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, pages 294–301, 2004.

[27] Lukas Chrpa Toms de la Rosa Isabel Cenamor, Mauro Vallati and Fernando Fernndez. Temporal: Temporal portfolio algorithm. IPC 2018 - Temporal Tracks.

[28] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth J. Teller. Anytime motion planning using the RRT. In *Proceedings of ICRA*, 2011.

[29] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proceedings of the 5th International Conference of Principles on Knowledge Representation and Reasoning (KR'96)*, pages 374–384, 1996.

[30] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of ECAI*, 1992.

[31] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic and stochastic search. In *Proceedings of AAAI*, 1996.

[32] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 318–325, 1999.

[33] Henry A. Kautz, Bart Selman, and Jörg Hoffmann. SatPlan: Planning as satisfiability. In *Abstracts of the 5th International Planning Competition*, 2006.

[34] Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. A sampling-based strategy planner for nondeterministic hybrid systems. In *Proceedings of ICRA*, 2014.

[35] Matthew R. Maly, Morteza Lahijanian, Lydia E. Kavraki, Hadas Kress-Gazit, and Moshe Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *Proceedings of HSCC*, 2013.

[36] David Mcallester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI'91)*, volume 2, pages 634–639, 1991.

[37] Hidetomo Nabeshima, Koji Iwanuma, and Katsumi Inoue. Effective sat planning by speculative computation. In *AI 2002: Advances in Artificial Intelligence*, volume 2557. Springer, 2002.

[38] Eva Onaindia Oscar Sapena, Eliseo Marzal. Tflap: a temporal forward partial-order planner. IPC 2018 - Temporal Tracks.

[39] J. Scott Penberthy and Daniel S. Weld. Temporal planning with continuous change. In *Proceedings of AAAI*, 1994.

[40] Wiktor Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. Heuristic planning for hybrid systems. In *Conference on Artificial Intelligence, AAAI*, pages 4254–4255. AAAI Press, 2016.

[41] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Falsification of LTL safety properties in hybrid systems. *STTT*, 15(4):305–320, 2013.

[42] Masood Feyzbakhsh Rankooh and Gholamreza Ghassem-Sani. Itsat: An efficient sat-based temporal planner. *J. Artif. Intell. Res.*, 53:541–632, 2015.

[43] Jussi Rintanen. Evaluation strategies for planning as satisfiability. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 682–687, 2004.

[44] Jussi Rintanen. Heuristics for planning with SAT. In *Proceedings of the 16th international conference on Principles and Practice of Constraint Programming (CP'10)*, pages 414–428, 2010.

[45] Jussi Rintanen. Madagascar: Efficient planning with SAT. In *the 7th International Planning Competition*, 2010.

[46] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170:1031–1080, 2006.

[47] Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. SAT-based parallel planning using a split representation of actions. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 2009.

[48] Emre Savas, Maria Fox, Derek Long, and Daniele Magazzeni. Planning using actions with control parameters. In *ECAI*, pages 1185–1193, 2016.

[49] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artificial Intelligence*, 166(1), 2005.

[50] SymPy. Website, 2013. http://www.sympy.org/.

[51] Paulo Tabuada, George J. Pappas, and Pedro U. Lima. Composing abstractions of hybrid systems. In *Proceedings of HSCC*, 2002.

[52] Mauro Vallati, Daniele Magazzeni, Bart De Schutter, Lukas Chrpa, and Thomas Leo Mccluskey. Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach. In *Proceedings of AAAI*, 2016.