The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence
Planning for Hybrid Systems: Technical Report WS-16-12

A Happening-Based Encoding for Nonlinear PDDL+ Planning

Daniel Bryce
dbryce@sift.net
SIFT, LLC.

Abstract

Hybrid planning with nonlinear continuous change is a significant challenge for existing planners. Prior works limit their scope to linear change or base their formalisms in model checking frameworks with inherent limitations. We address nonlinear PDDL+ planning with a new encoding in first order logic over real valued functions. Our planner, `PluReal`, translates PDDL+ to this logical encoding and applies the dReal Satisfiability Modulo Theories (SMT) solver to construct plans. Unlike prior work that uses dReal in the hybrid system model checking tradition, `PluReal` is based in the planning as satisfiability (SAT) heritage. Adopting the SAT approach helps lift several unnatural restrictions that are imposed by the translation through hybrid systems and leads to improved scalability even without SMT solver variable selection heuristics.

Introduction

Most contemporary work on hybrid planning focuses on linear continuous change. Those that allow nonlinear continuous change, dReach (Bryce et al. 2015) and UPMurphi (Della Penna et al. 2009), suffer from the state explosion problem because they flatten the discrete aspects of the problems. These works, and a majority of hybrid planners, transform hybrid planning into model checking hybrid automata. As we describe, there are a number of hybrid planning constructs that are difficult to express as hybrid automata and motivate us to seek another formalism.

We revisit the encodings of TM-LPSAT (Shin and Davis 2005) in the context of Satisfiability Modulo Theories (SMT) and apply the dReal SMT solver. Our resulting planner, Planning Using Reals (`PluReal`), handles the full PDDL+ language, including processes, events, durative actions, nonlinear change, and timed initial literals. The primary difference from the TM-LPSAT encoding stems from using a nested universal quantification over time to handle "zero-crossing" happenings. Zero-crossings refer to cases where an exogenous process or event becomes dis/enabled and the encoding must ensure that a happening occurs at that point. Where Shin and Davis (2005) use an arguably elegant/complicated set of constraints that assumes linear change, we use a simple invariant constraint that applies to general nonlinear change.

Figure 1 illustrates how `PluReal` and our prior work (Bryce et al. 2015) with dReach encode and solve PDDL+ problems with dReal. Our prior work encodes PDDL+ problems as a hybrid automata, and then applies dReach to construct a counter-example corresponding to a plan. The critical limitation of dReach is that it reasons with hybrid automata, and our prior encoding of PDDL+ is susceptible to state explosion – the hybrid automaton associates each logical state with a mode. In contrast, `PluReal` avoids the state explosion problem with its encoding. Following the planning as SAT tradition, `PluReal` encodes discrete state fluents as Boolean variables.

Aside from factoring the discrete part of the state, the new encoding largely resembles dReach's SMT-based encodings of hybrid automata. Assignments to discrete variables correspond to modes, and discrete changes occurring at happenings correspond to jumps, and continuous change between happenings correspond to continuous change while occupying a mode. The encoding also consolidates multiple simultaneous discrete or continuous changes due to actions, processes, or events to specify the equivalent of a hybrid automaton's flows and jump guards and updates. Unlike in hybrid automata where "must" semantics (Bogomolov et al. 2015) for event or processes transitions are difficult to encode, our encoding facilitates these easily and naturally. Moreover, we are not aware of any hybrid automata approaches to hybrid planning that handle timed initial literals.

We find that the new happening-based encoding is not only smaller than the dReach encoding, but more able to exploit parallelism in the PDDL+ problems. The primary difference is that we factor concurrency so that it can occur at the same happening (if possible) rather than over multiple hybrid automaton jumps that do not advance time. We show significant scale-up in the generator problem, where it is possible to simultaneously run a generator and refill it with multiple tanks. `PluReal` also performs competitively with dReach in the dribble domain, but is less efficient in the car domain. We attribute the lower scalability in these domains to the lack of a reachability heuristic (as used by dReach), and how parallelism cannot be capitalized upon to the same degree as the generator problem. We are encouraged by the strong performance without a heuristic, and mention avenues for incorporating heuristics into `PluReal`.

In the following, we provide a brief introduction to the PDDL+ planning model, an overview of our encoding through an example based in the car domain, a description of the encoding, an empirical evaluation, related work, and conclusion.

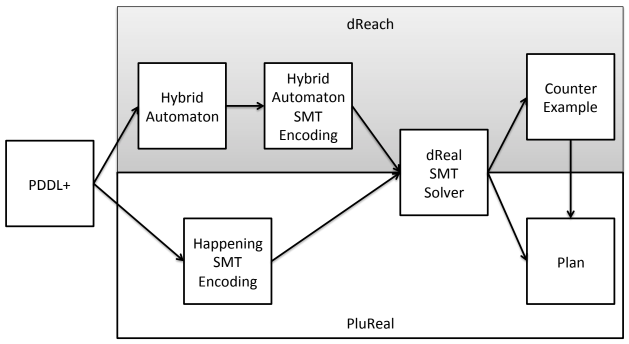

Figure 1: `PluReal` and dReach encode and solve instances with the dReal SMT solver.

Happenings:	$t_0 = 0$	$t_1 = 0$	$t_2 = 6$
Reals:	$a_0^{\vdash} = 0$ $a_0^{\dashv} = 0$ $v_0^{\vdash} = 0$ $v_0^{\dashv} = 0$ $d_0^{\vdash} = 0$ $d_0^{\dashv} = 0$	$a_1^{\vdash} = 1$ $a_1^{\dashv} = 1$ $v_1^{\vdash} = 0$ $v_1^{\dashv} = 6$ $d_1^{\vdash} = 0$ $d_1^{\dashv} = 18$	$a_2^{\vdash} = 0$ $v_2^{\vdash} = 6$ $d_2^{\vdash} = 18$
Booleans:	$running_0$ $\neg in - zone_0$	$running_1$ $\neg in - zone_1$	$running_2$ $in - zone_2$
Processes:	$moving_0$	$moving_1$	$moving_2$
Events:	$\neg explode_0$	$\neg explode_1$	$\neg explode_2$
Atomic Actions:	$\neg accel_0$ $\neg decel_0$	$accel_1$ $\neg decel_1$	$\neg accel_2$ $decel_2$
Durative Actions:	$\neg song_0^s$ $\neg song_0^c$ $\neg song_0^e$	$song_1^s$ $\neg song_1^c$ $\neg song_1^e$	$\neg song_2^s$ $\neg song_2^c$ $song_2^e$

Figure 2: Example solution for variation of the car domain.

Background

Hybrid Planning

Following (Bogomolov et al. 2014), a PDDL+ planning instance is a pair $I = (Dom, Prob)$ where $Dom = (Fs, Rs, As, Es, Ps, arity)$ is a tuple comprising a finite set of function symbols Fs, a finite set of relation symbols Rs, a finite set of durative and atomic actions As, a finite set of events Es, a finite set of processes Ps, and a function arity mapping all symbols in $Fs \cup Rs$ to their respective arities. The triple $Prob = (Os, Init, G)$ comprises a set of domain objects Os, the initial state $Init$, and the goal specification G. In the following, we restrict our focus to grounded planning instances that are grounded in the conventional manner. We discuss each element of the planning task description as we describe the encoding.

First-order Theories of the Reals

Our work formulates $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ encodings, where $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ represents the first-order signature over the reals with the set $\mathcal{F}$ of computable real functions. We use the dReal solver to solve (i.e., find satisfying solutions of, or lack thereof) these encodings. In the following, we provide an overview of $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$.

$\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-Formulas $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas are first-order formulas over real numbers, whose signature allows an arbitrary collection $\mathcal{F}$ of Type 2 computable real functions (Gao, Avigad, and Clarke 2012). The syntax is:

$$t := c \mid x \mid f(t(\vec{x}));$$
$$\varphi := t(\vec{x}) > 0 \mid t(\vec{x}) \geq 0 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x_i \varphi \mid \forall x_i \varphi.$$

A function is Type 2 computable if it can be algorithmically evaluated up to an arbitrary numerical accuracy. All common continuous real functions are Type 2 computable.

dReal

dReal checks whether an $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ formula is δ-satisfiable (a decidable problem) by combining a SAT solver (Eén and Sörensson 2004) with an ICP solver (Granvilliers and Benhamou 2006). dReal employs the DPLL(T) framework (Bruttomesso et al. 2010) for SMT. It first solves the Boolean constraints to find a satisfying set of literals of the form $(t(\vec{x}) \geq 0)$ or $\neg(t(\vec{x}) \geq 0)$. This conjunctive set of literals imposes a set of numeric constraints that are solved using ICP. If successful, dReal finishes, and otherwise, the ICP solver returns a set of literals that explain inconsistency. The inconsistent literals become a conflict clause that can be used by the SAT solver. If the SAT solver cannot find a satisfying set of literals, then it returns with an unsatisfiability result.

The ICP solver uses the branch and prune (Van Hentenryck, McAllester, and Kapur 1997) algorithm to refine a set of intervals over the continuous variables (called a box). Each branch splits the interval of a single continuous variable, creating two boxes. Pruning operators propagate the constraints to shrink the boxes. ICP continues to branch and prune boxes until it finds a box that is δ-satisfiable or establishes that no such box exists (i.e., the constraints are inconsistent). A box is δ-satisfiable when for any vector of values $\vec{x}$ represented by the box, each constraint $f(\vec{x}) \geq -\delta$ is satisfied.

Overview of the Happening Encoding

We explain our encoding by first describing how we encode a solution. Figure 2 illustrates the variable assignments that correspond to the plan:

$$0.0 : song[6.0]$$
$$0.0 : accel[0.0]$$
$$6.0 : decel[0.0]$$

for a variation of the PDDL+ car domain (where the start times are noted before the colon and durations are in parentheses). For the sake of this example, we extend the car domain with the durative action $song$ because the car domain (Fox and Long 2006) does not include durative actions.

The happening variables indicate the initial state is at time 0.0 (t_0), the plan starts at 0.0 (t_1), and the $song$ action ends and the final action occurs at 6.0 (t_2). The real valued variables hold values just after ($\vdash$ superscript) and just prior to ($\dashv$ superscript) each happening. For example, between happenings one and two, the acceleration variable a is 1.0, the velocity v changes continuously from 0.0 to 6.0, and the distance d changes from 0.0 to 18.0. The Boolean variables hold their values between happenings, and only change over happenings. For example, $in-zone$ changes from false to true over happening 2 (as an effect of the $song$ action).

The $moving$ process is active between each happening because its condition $running$ is true. It defines the rate at which the velocity and distance change. The $explode$ event does not occur at any happening because its precondition is never satisfied. The $accel$ action occurs at happening 1 (increasing a discretely by 1) and the $decel$ action occurs at happening 2 (decreasing a discretely by 1). The durative action $song$ may start s, continue c, or end e at any happening. In this plan, it starts at happening 1 and ends at happening 2. In longer plans it may also continue over a happening if the happening occurs prior to its duration being exhausted.

In the following section, we detail the constraints necessary to ensure that assignments to these variables represent valid plans.

PDDL+ to SMT2 Encoding

The translation of a PDDL+ instance I with k time points (happenings) involves finding a satisfying assignment to the logical formula:

$$\begin{aligned}
&\exists \vec{F}_{0:k} \exists \vec{A}_{1:k} \exists \vec{E}_{1:k} \exists \vec{PS}_{0:k} \exists \vec{til} \exists t_{0:k} \exists \vec{\Delta}_{1:k} \exists \vec{\Gamma}_{0:k} \exists \vec{dur}_{1:k}.\\
&\mathsf{init}(\vec{F}_{0:k}, \vec{til}, t_{0:k}) \wedge\\
&\left(\bigwedge_{i=1}^{k} \mathsf{trans}(\vec{F}_{i:k}, \vec{A}_{i:k}, \vec{E}_{i:k}, \vec{PS}_{i:k}, t_{i:k}, \vec{dur}_i, \vec{\Gamma}_i, \vec{\Delta}_i, \vec{til})\right) \wedge\\
&(t_0 = 0) \wedge \left(\bigwedge_{i=0}^{k} t_i \leq t_{i+1}\right) \wedge \mathsf{goal}(\vec{F}_{k+1})
\end{aligned}$$

where the clauses encode the initial state, transition relation, happening order, and goal. The existentially quantified variables refer to time-stamped instances of the variables described in the previous section. The abbreviation of the form $\exists \vec{X}_{0:k}$, denotes $\exists \vec{X}_0 \exists \vec{X}_1 \ldots \exists \vec{X}_k$. Each $\exists \vec{X}_i$ denotes $\exists x_i \ldots \exists x'_i$, where x_i is a Boolean, integer, or real variable.

More specifically, $\vec{F}_i$ denotes all time-stamped fluents of the form $f_i^{\vdash}$ and $f_i^{\dashv}$ for real values just after happening i and just before happening $i+1$, and f_i for Boolean values persisting between happenings i and $i+1$. The action variables $\vec{A}_i$ include a_i to indicate if an atomic action occurs at i, and a_i^s, a_i^c, and a_i^e to denote that a durative action respectively starts at, continues over, or ends at happening i. The event variables $\vec{E}_i$ include Boolean event variables e_i that indicate whether an event occurs at happening i. The process variables $\vec{PS}_i$ include Boolean process variables ps_i to indicate if a process is active between happenings i and $i+1$. Each til variable in $\vec{til}$ denotes the happening at which a timed initial literal assignment occurs. Each variable t_i denotes the absolute time at which happening i occurs. The $\Delta_i(a, f)$ variables denote the value by which an action a increases the value of fluent f at happening i. The $\Gamma_i(a)$ variables are 0-1 integers that indicate whether a durative action or process is active between happenings i and $i+1$. The $dur_i(a)$ variables denote the duration of an instance of a durative action a starting at happening i.

Initial State: The clause $\mathsf{init}(\vec{F}_{0:k}, \vec{til}, t_{0:k})$ encodes both initial fluent values and any timed initial literals. It defines:

$$\left(\bigwedge_{I(f,0)\in I_0} f_0^* = I(f,0)\right) \wedge \left(\bigwedge_{I(f,t)\in I_{til}} \mathsf{til}(f, t, F_{0:k}, t_{0:k})\right) \tag{1}$$

where f_0^* denotes $f_0^{\vdash}$ if f is real and f_0 if f is Boolean, I_{til} is a set of timed initial literals and I_0 is a set of initial state assignments.

Each timed initial literal is a fluent assignment $I(f, t)$, where $t > 0$. We associate a unique integer variable til with each timed initial literal to denote the happening at which it occurs. The clause $\mathsf{til}(f, t, F_{0:k}, t_{0:k})$ defines:

$$(1 \leq til \leq k) \wedge \left(\bigvee_{i=1}^{k} (til = i) \rightarrow (f_i^* = I(f,t)) \wedge (t_i = t)\right) \tag{2}$$

where f_i^* is $f_i^{\vdash}$ if f is a real variable, and f_i if it is a Boolean variable. The clause states that the timed initial literal must occur at some happening and if it occurs at happening i then the corresponding fluent is assigned appropriately and the time specified by the happening is equal to the time of the timed initial literal.

Transition Relation: The transition relation states for each happening i which discrete changes can or must occur and what continuous changes occur between i and $i+1$. The clause $\mathsf{trans}(\vec{F}_{i:k}, \vec{A}_{i:k}, \vec{E}_{i:k}, \vec{PS}_{i:k}, t_{i:k}, \vec{dur}_i, \vec{\Gamma}_i, \vec{\Delta}_i)$ defines:

$$\begin{aligned}
&\mathsf{acts}(\vec{A}_i, \vec{F}_{i-1:i}) \wedge \mathsf{duracts}(\vec{A}_{i:k}, \vec{F}_{i-1:i}, \vec{dur}_i) \wedge\\
&\mathsf{events}(\vec{E}_i, \vec{F}_{i-1:i}) \wedge \mathsf{procs}(\vec{PS}_i, \vec{F}_i) \wedge\\
&\mathsf{concur}(\vec{A}_i, \vec{E}_i, \vec{til}, \vec{PS}_i, \vec{F}_i) \wedge \mathsf{frame}(\vec{A}_i, \vec{E}_i, \vec{til}, \vec{F}_i) \wedge\\
&\mathsf{mutex}(\vec{A}_i, \vec{E}_i)
\end{aligned} \tag{3}$$

where acts, $\mathsf{duracts}$, events, and procs define the semantics of the respective elements, concur defines how simultaneous concurrent change is reconciled, frame defines the frame axioms, and mutex defines mutexes.

Atomic Actions: For each happening i, the clause $\mathsf{acts}(\vec{A}_i, \vec{F}_{i-1:i})$ encodes the following:

$$\bigwedge_{a\in As_{atomic}} \mathsf{pre}(a_i, \vec{F}_{i-1}) \wedge \mathsf{eff}(a_i, \vec{F}_{i-1:i}) \tag{4}$$

where As_{atomic} is the set of ground atomic actions. The precondition clause $\mathsf{pre}(a_i, \vec{F}_{i-1})$ defines:

$$a_i \rightarrow \mathsf{pre}_{i-1}^{\dashv}(a) \tag{5}$$

where $\mathsf{pre}^{\dashv}_{i-1}(a)$ is the clause obtained by substituting each fluent in $\mathsf{pre}(a)$ by f_{i-1} if Boolean and $f^{\dashv}_{i-1}$ if real. The effect clause $\mathsf{eff}(a_i, \vec{F}_{i-1:i})$ defines:

$$a_i \rightarrow \left(\bigwedge_{(f:=\alpha)\in \mathsf{eff}(a)} (f^*_i = \alpha^{\dashv}_{i-1}) \right) \wedge \tag{6}$$

$$\left(\bigwedge_{(f+=\alpha)\in \mathsf{eff}(a)} (\Delta_i(a,f) = \alpha^{\dashv}_{i-1}) \right) \tag{7}$$

where each assignment $f := \alpha$ assigns the value of $f^{\vdash}_i$ if f is real or f_i if Boolean (denoted by f^*_i), and each increment effect $f+=\alpha$ assigns the increment variable $\Delta_i(a,f)$. We describe below how the increment variables are aggregated to express simultaneous increases from multiple concurrent actions. We also convert decrease effects into increase effects without loss of generality. The value $\alpha^{\dashv}_{i-1}$ assigned to these variables is the value of the expression α evaluated just prior to happening i (i.e., by substituting the fluents f in α by $f^{\dashv}_{i-1}$ or f_i, as appropriate).

Events: Events resemble atomic actions, except that they must occur if their preconditions are satisfied. The $\mathsf{events}(\vec{E}_i, \vec{F}_{i-1:i})$ clause defines:

$$\bigwedge_{e\in Es} \mathsf{immed}(e_i, \vec{F}_{i-1}) \wedge\ \mathsf{pre}(e_i, \vec{F}_{i-1}) \wedge \mathsf{eff}(e_i, \vec{F}_{i-1:i}) \tag{8}$$

where Es is the set of ground events. The $\mathsf{immed}(e_i, \vec{F}_{i-1})$ clause asserts that if the event is enabled by a change at happening at $i-1$, then happening i must occur immediately:

$$\mathsf{pre}^{\vdash}_{i-1}(e) \rightarrow (t_i = t_{i-1}) \tag{9}$$

where $\mathsf{pre}^{\vdash}_{i-1}(e)$ is the formula obtained by replacing fluents in $\mathsf{pre}(e)$ by their time indexed fluents.

An event must occur at happening i iff its precondition is satisfied by a change at happening $i-1$ or it became satisfied between $i-1$ and i. The precondition clause $\mathsf{pre}(e_i, \vec{F}_{i-1})$ defines:

$$e_i \leftrightarrow \left(\mathsf{pre}^{\vdash}_{i-1}(e)\right) \vee \left(\mathsf{pre}^{\dashv}_{i-1}(e) \wedge \forall t \in [t_{i-1}, t_i).\neg \mathsf{pre}_t(e)\right) \tag{10}$$

so that the precondition must be satisfied just after happening $i-1$ or just before i. We note that the immed and pre clauses solve the zero-crossing problem for events by ensuring that a happening occurs at the first time that the event condition is satisfied. The above clause also ensures the "must" semantics for events in that it uses a bi-implication, as opposed to the similar clause for actions the uses an implication for "may" semantics. The effect clause $\mathsf{eff}(e_i, \vec{F}_{i-1:i})$ defines:

$$e_i \rightarrow \left(\bigwedge_{(f:=\alpha)\in \mathsf{eff}(e)} (f^*_i = \alpha^{\dashv}_{i-1}) \right) \wedge$$

$$\left(\bigwedge_{(f+=\alpha)\in \mathsf{eff}(e)} (\Delta_i(e,f) = \alpha^{\dashv}_{i-1}) \right) \tag{11}$$

where each assignment $f := \alpha$ assigns the value of $f^{\vdash}_i$ if f is real or f_i if Boolean (denoted by f^*_i), and each increment effect $f+=\alpha$ assigns the increment variable $\Delta_i(e,f)$.

Durative Actions: The clause $\mathsf{duracts}(\vec{A}_{i:k}, \vec{F}_{i-1:k})$ defines:

$$\bigwedge_{a\in As_{dur}} \mathsf{dur}(a_i, \vec{A}_{i:k}, \vec{dur}_i, t_{i:k}, \vec{F}_{i:k}) \wedge$$
$$\mathsf{coher}(a_i, \vec{A}_{i:k}, \vec{dur}_i, t_{i:k}) \wedge$$
$$\mathsf{pre}^s(a_i, \vec{F}_{i-1}) \wedge \mathsf{pre}^e(a_i, \vec{F}_{i-1}) \wedge \mathsf{pre}^o(a_i, \vec{F}_{i-1}) \wedge$$
$$\mathsf{eff}^s(a_i, \vec{F}_{i-1:i}) \wedge \mathsf{eff}^e(a_i, \vec{F}_{i-1:i}) \wedge \mathsf{eff}^o(a_i, \vec{F}_{i-1:i}) \tag{12}$$

where As_{dur} is the set of ground durative actions. A durative action a, if executed, starts at happening i, continues over happenings $i+1, \ldots j-1$, and ends at j. The action's duration $dur_i(a)$ is either evaluated just prior to its occurrence, defining $\mathsf{dur}(a_i, \vec{A}_{i:k}, \vec{dur}_i, t_{i:k}, \vec{F}_{i:k})$ as:

$$a^s_i \rightarrow (dur_i(a) = dur^{\dashv}_{i-1}(a)) \tag{13}$$

or just prior to its end, defining $\mathsf{dur}(a_i, \vec{A}_{i:k}, \vec{dur}_i, t_{i:k}, \vec{F}_{i:k})$ as:

$$\bigwedge_{j\in[i+1,k]} a^s_i \wedge a^c_{i+1} \wedge \ldots \wedge a^c_{j-1} \wedge a^e_j \rightarrow$$
$$(dur_i(a) = dur^{\dashv}_{j-1}(a)) \tag{14}$$

For action coherence, the $\mathsf{coher}(a_i, \vec{A}_{i:k}, \vec{dur}_i, t_{i:k})$ clause defines that the duration is equivalent to the difference of the start and end time point, and that the starting, ending, and continuation of actions at happenings are well formed:

$$\left(\bigwedge_{j\in[i+1,k]} a^s_i \wedge a^c_{i+1} \wedge \ldots \wedge a^c_{j-1} \wedge a^e_j \rightarrow (t_j - t_i = dur_i(a)) \right) \wedge$$
$$\left(a^s_i \rightarrow a^c_{i+1} \vee a^e_{i+1}\right) \wedge$$
$$\left(a^e_i \rightarrow a^c_{i-1} \vee a^s_{i-1}\right) \wedge$$
$$\left(a^c_i \rightarrow a^c_{i+1} \vee a^e_{i+1}\right) \wedge$$
$$\left(a^c_i \rightarrow a^c_{i-1} \vee a^s_{i-1}\right)$$

The at-start precondition clause $\mathsf{pre}^s(a_i, \vec{F}_{i-1})$ defines:

$$a^s_i \rightarrow s\mathsf{pre}^{\dashv}_{i-1}(a) \tag{15}$$

where $s\mathsf{pre}^{\dashv}_{i-1}(a)$ is a conjunction of at-start preconditions whose values are evaluated just prior to happening i. The at-end precondition clause $\mathsf{pre}^e(a_i, \vec{F}_{i-1})$ defines:

$$a^e_i \rightarrow e\mathsf{pre}^{\dashv}_{i-1}(a) \tag{16}$$

where $e\mathsf{pre}^{\dashv}_{i-1}(a)$ is a conjunction of at-end preconditions whose values are evaluated just prior to happening i. The overall preconditions clause $\mathsf{pre}^o(a_i, \vec{F}_{i-1})$ defines:

$$a^s_i \vee a^c_i \rightarrow \forall t \in [t_i, t_{i+1}].o\mathsf{pre}_t(a) \tag{17}$$

where $o\mathsf{pre}_t(a)$ is a conjunction of overall preconditions.

The at-start effect clause $\mathsf{eff}^s(a_i, \vec{F}_{i-1:i})$ defines:

$$a_i^s \rightarrow \left(\bigwedge_{(f:=\alpha)\in s\mathsf{eff}(a)} (f_i^* = \alpha_{i-1}^{\dashv}) \right) \wedge \quad (18)$$

$$\left(\bigwedge_{(f+=\alpha)\in s\mathsf{eff}(a)} (\Delta_i(a,f) = \alpha_{i-1}^{\dashv}) \right) \quad (19)$$

where $s\mathsf{eff}(a)$ are the set of start effects. The terms in the clause resemble those appearing in the atomic action effect clause. The at-end effect clause $\mathsf{eff}^e(a_i, \vec{F}_{i-1:i})$ is defined similarly, but substitutes the at-end effects $e\mathsf{eff}(a)$ for the at-start effects and a_i^e for a_i^s. The overall effect clause $\mathsf{eff}^s(a_i, \vec{F}_{i-1:i})$ defines when the continuous effects are active because the action has just started or is ongoing:

$$\big((a_i^s \vee a_i^c) \wedge (a_{i+1}^c \vee a_{i+1}^e) \leftrightarrow (\Gamma_i(a) = 1)\big) \wedge$$
$$\big(\neg((a_i^s \vee a_i^c) \wedge (a_{i+1}^c \vee a_{i+1}^e)) \leftrightarrow (\Gamma_i(a) = 0)\big) \quad (20)$$

where $\Gamma_i(a)$ is an indicator function that is used below to determine whether a is active and contributes to the simultaneous continuous change of real fluents it effects.

Processes: The $\mathsf{procs}(\vec{P}S_i, \vec{F}_{i-1:i})$ clause states the conditions under which each process is (in)active and how each process effects the real fluents. It defines:

$$\bigwedge_{ps\in Ps} \mathsf{pre}(ps_i, \vec{F}_i) \wedge \mathsf{eff}(ps_i, \vec{F}_i) \quad (21)$$

The $\mathsf{pre}(ps_i, \vec{F}_i)$ clause defines:

$$(ps_i \leftrightarrow \forall t \in [t_i, t_{i+1}].\mathsf{pre}_t(ps)) \wedge$$
$$(\neg ps_i \leftrightarrow \forall t \in [t_i, t_{i+1}].\neg\mathsf{pre}_t(ps)) \quad (22)$$

so that between happenings, processes are active iff their preconditions are satisfied and are inactive iff their preconditions are not satisfied (i.e., the "must" semantics). Similar to events, the universally quantified time variable ensures that the processes dis/enable on happenings (i.e., zero-crossings). The $\mathsf{eff}(ps_i, \vec{F}_i)$ clause defines:

$$(ps_i \leftrightarrow (\Gamma_i(ps) = 1)) \wedge$$
$$(\neg ps_i \leftrightarrow (\Gamma_i(ps) = 0)) \quad (23)$$

whether the process is active or not, and the $\Gamma_i(ps)$ indicator function helps define whether it contributes to the continuous change of real fluents.

Concurrent Modifications: The $\mathsf{concur}(\vec{A}_i, \vec{E}_i, \vec{til}, \vec{P}S_i, F_i)$ clause defines how to aggregate multiple simultaneous discrete or continuous changes to each fluent:

$$\bigwedge_f \mathsf{concur}^{disc}(\vec{A}_i, \vec{E}_i, \vec{til}, \vec{P}S_i, f_i) \wedge \mathsf{concur}^{cont}(\vec{A}_i, \vec{P}S_i, f_i) \quad (24)$$

The $\mathsf{concur}^{disc}(\vec{A}_i, \vec{E}_i, \vec{til}, \vec{P}S_i, f_i)$ clause defines:

$$\neg\mathsf{assigners}_i(f) \rightarrow \left(f_i^{\vdash} = f_{i-1}^{\dashv} + \sum_{a\in A(f)} \Delta_i(a,f) \right) \quad (25)$$

where $A(f)$ denotes the actions and events that increase f, and $\mathsf{assigners}_i(f)$ defines:

$$\left(\bigvee_{til\in til(f)} (til = i) \right) \vee \left(\bigvee_{e\in E(f)} e_i \right) \quad (26)$$

where $til(f)$ denotes the set of timed initial literals that assign f, and let $E(f)$ denote the actions and events that assign f. In order to simplify the notation, we assume that the sets $E(f)$ and $A(f)$ include durative actions and we designate appropriately the literals a_i^s or a_i^e for each durative action that increase f at the start or end, respectively. If no assignments are made to f at happening i, its value just after happening i is the sum of all increase effects.

The $\mathsf{concur}^{cont}(\vec{A}_i, \vec{E}_i, \vec{til}, \vec{P}S_i, f_i)$ defines:

$$f_i^{\dashv} = f_i^{\vdash} + \int_{t_i}^{t_{i+1}} \sum_{a\in A(f)} \Gamma_i(a)\mathsf{eff}^f(a)(s)d(s) \quad (27)$$

where $A(f)$ denotes the durative actions and processes that effect f continuously, and $\mathsf{eff}^f(a)$ denotes the differential $d[f]/dt$ defined by a.

Frame Axioms: The $\mathsf{frame}(\vec{A}_i, \vec{E}_i, \vec{til}, \vec{F}_i)$ clause defines:

$$\neg\mathsf{effectors}_i(f) \rightarrow (f_i^{\vdash} = f_i^{\dashv}) \quad (28)$$

where $\mathsf{effectors}_i(f)$ is defined by:

$$\left(\bigvee_{til\in til(f)} til \neq i \right) \wedge \left(\bigvee_{a\in A(f)} a_i \right) \quad (29)$$

where $A(f)$ are the action or events effecting f discretely and $til(f)$ are the timed initial literals effecting f.

Interference: The $\mathsf{mutex}(\vec{A}_i, \vec{E}_i)$ clause defines:

$$\bigwedge_{\substack{a,a'\in As\cup Es: \\ interfere(a,a')}} a_i \rightarrow \neg a_i' \quad (30)$$

where $interfere(a, a')$ uses the standard PDDL 2.1 rules for interference. With a slight abuse of notation, we treat durative actions as two actions, a start and an end, that can interfere with other actions and events.

Goal: The goal clause $\mathsf{goal}(\vec{F}_{k+1}^{\dashv})$ is an expression over the fluents that must hold at a special happening $k+1$, where no other actions or events may be active or occur, stated as:

$$G_{k+1}^{\dashv} \quad (31)$$

where $G_{k+1}^{\dashv}$ substitutes each fluent in the goal expression by a time stamped version of the fluent variable. The goal can also be thought of as an atomic action occurring at happening $k+1$ and whose preconditions (the goal) must be satisfied just prior to its execution (as denoted by the "$\dashv$" superscript). This allows the planner to decide what time the goal is satisfied following the last normal happening k.

Plan Evaluation

dReal can be used to decide whether a δ-satisfiable solution exists for the encoding described in the previous section. If no such δ-satisfiable solution exists, then dReal will report unsatisfiability. A δ-satisfiable solution is an assignment to Boolean variables and an interval for each real variable. The solution encodes a plan tube, a set of possible plans. Realizing a plan involves selecting a start time and duration for each action from the plan tube. In the results reported in the following section, we report the time to encode and find a solution, but not to extract a plan.

We do not extract a plan from the plan tube for two reasons. First, we can reduce δ so that the plan tube is arbitrarily precise. While reducing δ does not guarantee that the plan tube reduces to a single plan, it can lead to intervals for action start times and durations that are smaller than the precision attainable by a finite-precision plan executor. Second, the plan tube is identical, in principle, to the plan tube created by VAL while validating plans. VAL defines a set of intervals around the each action start time and duration and then uses Monte Carlo sampling from the intervals to check whether a sufficient number of plan perturbations are valid. Validating a δ plan tube is identical to that of a plan, with the exception that we provide the plan tube directly. Because VAL grows the plan tube symmetrically around a plan, one possible plan extraction strategy is to select the center point of the plan tube – in this case, the VAL generated plan tube matches the δ plan tube.

Empirical Evaluation

We compare `PluReal` with dReach and existing planners, including SpaceEx (Bogomolov et al. 2014), CoLin (Coles et al. 2012), and UPMurphi (Della Penna et al. 2009). We use the generator and car instances from the literature (Bogomolov et al. 2014) and the Dribble domain (Bryce et al. 2015). We compare on linear and nonlinear versions of generator and car, but only a nonlinear version of Dribble.

Domains: The car domain includes only instantaneous actions and processes. The actions are to start or stop the car, and accelerate or decelerate. The moving process models one-dimensional kinematics (distance as a function of velocity and velocity as a function of acceleration) and the wind-resistance process models the drag effect upon velocity. Additional actions for additional acceleration or deceleration increments increases the branching factor of the problem. The linear and nonlinear versions of the domain differ in whether they include the nonlinear wind-resistance process.

The generator domain includes two durative actions: generate, and refuel. The generate action has a duration of 1000 time units and consumes fuel at a linear rate. Its at-end effect satisfies the goal. Its overall condition requires that the fuel level is non-negative. The instances scale in the number of tanks required to refuel the generator so that its overall condition is satisfied. The refuel actions increase the fuel level in the generator continuously, by a linear rate (in the linear version) or a nonlinear rate (in the nonlinear version). For example, the refuel action defines the effects linearly as

```
(increase (fuel ?g) (* #t 2))
```

or nonlinearly

```
(increase (ptime ?t) (* #t 1))
(increase (fuel ?g)
          (* #t (* 0.1 (* (ptime ?t)
                          (ptime ?t)))))
```

The dribble domain involves a process effecting the location of a ball, upward velocity (v) decreasing due to gravity ($-g$) and drag ($-0.1v^2$), and vertical position (x) increasing with velocity (v). The available actions are dribble(f) which decrease velocity by $f \in \{0, 1, 2, 4\}$. The dribble actions have the precondition that velocity is zero. The bounce event increases velocity by $-0.9v$ and has the condition that the ball position x is zero. The initial state places the ball at $x = 1$ with velocity $v = 0$ and the goal is to reach $1.5 \leq x \leq 3.0$.

Results: Table 1 lists results for nonlinear problems, and Table 2, linear problems. The tables list the domain, planner, and runtimes in seconds for several instances. The `PluReal` and dReach results were run on the same machine with a 2.6 GHz Intel Core i7 and 8GB RAM. The results for other planners are reproduced from prior work (Bogomolov et al. 2014). CoLin is only capable of addressing the generator problem, so we omit it from the car results. The SpaceEx results are the time to show that the goal is reachable in an over-approximation of the hybrid state space (i.e., possible plan existence), and not to generate/extract a plan. Similarly, the `PluReal` and dReach results are the total time (including translation and encoding) to show that there exists a δ-satisfiable solution for an SMT instance, where $\delta = 0.1$. That is, the plan encoded by the δ-satisfiable solution is correct up to δ perturbation of the real variables.

We compare only `PluReal` and dReach on the nonlinear instances. We see that `PluReal` excels in the generator domain compared to dReach, is competitive in the dribble domain, and is not able to scale well in the car domain. The same trends hold in the linear problem instances. The reason that `PluReal` performs well in generator, is that it can find a solution in a three happening ($k = 3$) encoding by parallelizing the refuel actions with the generate action. The competing approaches cannot exploit parallelism in this way because they involve encoding each action as a series of transitions in a hybrid automaton. While the transitions may not advance time, they essentially serialize the decisions for parallel actions. While `PluReal` must also serialize its decisions within the SMT solver variable assignments, the happening-based encoding is more compact (and hence uses less variables) because it specifies action choice variables once per happening, and not once per hybrid automata transition. `PluReal` performs comparatively worse when problems require serial actions, as in the car and dribble domains. dReach performs better in these domains because it employs a reachability heuristic, where `PluReal` does not.

Related Work

While PDDL+ has been an accepted language for planning with continuous change for nearly a decade, very few planners have been able to handle its expressivity. Planners either assume that all continuous change is linear (Shin and

Dom	Planner	1	2	3	4	5	6	7	8
Gen	`PluReal`	0.84	0.78	0.90	1.17	1.25	1.44	1.81	1.57
Gen	dReach	12.80	71.63	1696.84	-	-	-	-	-
Dribble	`PluReal`	25.49	45.07	106.38	439.09	2030.05	n.a.	n.a.	n.a.
Dribble	dReach	192.52	33.50	65.16	122.91	224.61	n.a.	n.a.	n.a.
Car	`PluReal`	-	-	-	-	-	-	-	-
Car	dReach	16.62	16.64	16.25	16.77	16.56	16.79	17.44	16.6

Table 1: Runtime results (s) on nonlinear generator and car.

Dom	Planner	1	2	3	4	5	6	7	8
Gen	`PluReal`	0.34	0.38	0.54	0.83	1.34	2.72	4.68	9.01
Gen	dReach	3.07	15.6	134.71	1699.87	-	-	-	-
Gen	SpaceEx	0.01	0.03	0.07	0.1	0.19	0.28	0.45	0.65
Gen	CoLin	0.01	0.09	0.2	2.52	32.62	600.58	-	-
Gen	UPMur	0.2	18.2	402.34	-	-	-	-	-
Car	`PluReal`	362.05	1015.22	-	-	-	-	-	-
Car	dReach	1.07	1.17	1.16	1.22	1.23	1.29	1.26	1.21
Car	SpaceEx	0.01	0.01	0.01	0.03	0.04	0.05	0.06	0.07
Car	UPMur	28.44	386.5	-	-	-	-	-	-

Table 2: Runtime results (s) on linear generator and car. “-” indicates a timeout.

Davis 2005; Coles and Coles 2014; Bogomolov et al. 2014; Coles et al. 2012) or handle nonlinear change by discretization (Della Penna et al. 2009).

Unlike UPMurphi, dReal does not discretize to deal with nonlinear continuous change. While dReal uses ICP search to split real intervals and its constraint propagation may reduce an interval to a single value, it does not produce a finite set of values from an interval in a lossy manner. dReal differs from UPMurphi in that it reasons about continuous change as intervals, and not as sets of values.

TM-LPSAT (Shin and Davis 2005) is the basis for our work because it uses a SAT solver to solve Boolean constraints and an LP solver to solve continuous (linear) constraints. The nature of the encodings is somewhat different in that our encoding handles zero-crossing events with universal quantification and can thus address problems with nonlinear change.

Bryce et al. (2015), Bogomolov et al. (2014) and Della Penna et al. (2009), make use of the planning as model checking paradigm. Unlike our work, Bogomolov et al. encode a network of linear hybrid automata and handle durative actions and events. Bogomolov et al. use the SpaceEx model checker (Frehse et al. 2011), which performs a symbolic search over the hybrid automata.

Coles and Coles (2014) and Coles et al. (2012) approach PDDL+ from the perspective of heuristic state space search. Coles and Coles exploit piecewise linear representations of continuous change to derive powerful pruning conditions for forward heuristic search. It is possible to combine this work with our happening-based encoding to manage the constraints describing a single partial plan, rather than the entire plan space. That is, it is conceivable to replace the LP constraints and associated solver with dReal.

Conclusion

We present a new approach to PDDL+ planning that compiles problems into the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ language, and is based in the planning as satisfiability tradition. The encoding is especially compact for problems with parallelism, as is typical of most SAT-based approaches to planning. By phrasing PDDL+ planning in SMT, we open future work on applying many of the useful SAT planning techniques developed for classical planning. Most notably, we hope to develop variable selection heuristics to prioritize the search and reachability analysis for computing mutexes and filtering unreachable actions and variables.

Acknowledgments: This work was supported under ONR contract N00014-13-1-0090.

References

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as model checking in hybrid domains. In Brodley, C. E., and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2228–2234. AAAI Press.

Bogomolov, S.; Magazzeni, D.; Minopoli, S.; and Wehrle, M. 2015. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In Brafman, R. I.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, 42–46. AAAI Press.

Bruttomesso, R.; Pek, E.; Sharygina, N.; and Tsitovich, A. 2010. The opensmt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 150–153.

Bryce, D.; Gao, S.; Musliner, D. J.; and Goldman, R. P. 2015. Smt-based nonlinear PDDL+ planning. In Bonet, B., and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 3247–3253. AAAI Press.

Coles, A. J., and Coles, A. I. 2014. PDDL+ planning with events and linear processes. In Chien, S.; Do, M. B.; Fern, A.; and Ruml, W., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*. AAAI.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2012. Colin: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44:1–96.

Della Penna, G.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. Upmurphi: A tool for universal planning on pddl+ problems. In *ICAPS*.

Eén, N., and Sörensson, N. 2004. An extensible sat-solver. In *Theory and applications of satisfiability testing*, 502–518. Springer.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res.(JAIR)* 27:235–297.

Frehse, G.; Le Guernic, C.; Donzé, A.; Cotton, S.; Ray, R.; Lebeltel, O.; Ripado, R.; Girard, A.; Dang, T.; and Maler, O. 2011. Spaceex: Scalable verification of hybrid systems. In *Computer Aided Verification*, 379–395. Springer.

Gao, S.; Avigad, J.; and Clarke, E. M. 2012. Delta-complete decision procedures for satisfiability over the reals. In *IJ-CAR*, 286–300.

Granvilliers, L., and Benhamou, F. 2006. Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques. *ACM Transactions on Mathematical Software (TOMS)* 32(1):138–156.

Shin, J., and Davis, E. 2005. Processes and continuous change in a sat-based planner. *Artif. Intell.* 166(1-2):194–253.

Van Hentenryck, P.; McAllester, D.; and Kapur, D. 1997. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis* 34(2):797–827.