SMT-based Nonlinear PDDL+ Planning

Daniel Bryce
SIFT, LLC.
dbryce@sift.net

Sicun Gao
MIT CSAIL
sicung@csail.mit.edu

David Musliner & Robert Goldman
SIFT, LLC.
{musliner, rpgoldman}@sift.net

Abstract

PDDL+ planning involves reasoning about mixed discrete-continuous change over time. Nearly all PDDL+ planners assume that continuous change is linear. We present a new technique that accommodates nonlinear change by encoding problems as nonlinear hybrid systems. Using this encoding, we apply a Satisfiability Modulo Theories (SMT) solver to find plans. We show that it is important to use a novel planning-specific heuristic for variable selection for SMT solving, which is inspired by recent advances in planning as SAT. We show the promising performance of the resulting solver on challenging nonlinear problems.

Introduction

Hybrid planning problems expressed in PDDL+ (Fox and Long 2006) model mixed discrete and continuous change over time. Prior work on PDDL+ plan synthesis assumes that continuous change is linear (Coles et al. 2012; Bogomolov et al. 2014; Coles and Coles 2014; Shin and Davis 2005). We present a Satisfiability Modulo Theories (SMT) encoding of PDDL+ planning that accommodates nonlinear continuous change, and develop novel planning-specific heuristics that build upon advances for SAT planning (Rintanen 2012).

We reduce PDDL+ planning to reachability problems of hybrid systems (Henzinger 1996), which are encoded and solved as first-order logic formulas over the real numbers (we call them $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas (Gao, Avigad, and Clarke 2012)). We use the framework of δ-complete decision procedures over the reals (Gao, Avigad, and Clarke 2012), and apply the dReal SMT solver to decide δ-satisfiability or unsatisfiability of the logic formulas. Showing δ-satisfiability then involves finding bounds on $\vec{x}$ (the continuous state variables) such that the lower and upper bound on each literal has width no more than δ. The dReal solver accomplishes this by first using a DPLL-based SAT solver to find a set of literals that satisfy each Boolean constraint. It then applies an Interval Constraint Propagation (ICP) based branch and prune solver to refine the intervals on each numeric variable constrained by the literals. Because of the relaxation from SMT to δ-SMT, we obtain a "tube" (Fox, Howey, and Long 2006) that may contain a feasible plan up to δ-bounded numerical errors. Fortunately, δ controls the precision of these intervals and can be made very small. Furthermore, if dReal identifies a problem as unsolvable, it is guaranteed to be correct.

SMT solvers have been developed for solving problems in formal verification. Adapting them to planning problems requires us to introduce new solving strategies for SMT solvers. As a main contribution of this work, we describe a novel variable and value selection strategy, which extends similar heuristics in SAT-based planning (Rintanen 2012). Our $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ encoding translates PDDL+ problems to hybrid system problems, which are much more complicated than traditional discrete planning, which only involves SAT-solving. We need to solve SMT formulas that contain modes (discrete states), flows (continuous change in each mode), invariants (constraints in each mode), and guarded jumps between modes with discrete change. The heuristic, which is a variation on the h^1 heuristic (Haslum and Geffner 2000), guides the SAT solver to construct feasible sequences of modes and to remove unreachable modes from the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ encoding. Given a sequence of modes, the jumps between modes correspond to PDDL+ happenings. We also present a new heuristic for ICP that favors early start times for instantaneous actions.

Our approach addresses PDDL+ with instantaneous actions and processes. We accommodate durative actions by first compiling them into instantaneous start and end actions that control a process describing continuous change. We evaluate our solver on both linear and nonlinear variations of the generator and car problems from the literature and a new domain that plans how to dribble a ball subject to nonlinear drag and gravity. We show that dReal is capable of finding non-trivial solutions and that our heuristic improves its scalability.

In the following, we describe dReal, an SMT solver for $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$. We present our encoding of PDDL+ planning as an $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ hybrid system and illustrate the encoding with a simplified version of the car domain. We then discuss our SAT variable selection heuristic and ICP time branching heuristic. We finish by empirically evaluating dReal and illustrate its potential for adapting SAT based planning to handle nonlinear continuous change.

SMT Solving for $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-Formulas

Our work formulates $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ encodings of hybrid systems to express PDDL+ planning. We use the dReal solver to solve (i.e., find satisfying solutions of) these encodings. As part of this work, we extend the dReal solver to use heuristics that incorporate domain-independent knowledge present in the hybrid system specification. In the following, we provide an overview of dReal and how it solves $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ problems.

$\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-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 standard:

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

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 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 unsatisfiable 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. The primary pruning operator enforces hull consistency (Granvilliers and Benhamou 2006) of the box. The constraints that involve integrating an ODE are propagated by performing interval-based integration with the CAPD library.[1] 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 $f(\vec{x}) \geq -\delta$ (or $f(\vec{x}) > -\delta$) is satisfied.

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) 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 instances that are event-free, and contain only instantaneous (non-durative) actions. We also assume a grounded planning instance that is grounded in the conventional manner.

States consist of both discrete variables (propositions $p \in P$) and a vector of numeric variables $\vec{x}$. Actions define a precondition pre over both discrete and numeric variables and effect eff that discretely changes both discrete and numeric variables. Processes define a precondition pre which enables the process when satisfied by the current state and an effect eff that adjusts the rate of change of continuous variables.

[1] http://capd.ii.uj.edu.pl/

Hybrid Automata

We encode hybrid automata reachability problems as a k-step M-delay SMT instance defined with the first-order language $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ over the reals, which allows the use of a wide range of real functions including nonlinear ODEs.

$\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-Representations of Hybrid Automata A hybrid automaton in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-representation is a tuple

$$H = \langle X, Q, \{\mathsf{flow}_q(\vec{x}_0, \vec{x}_t, t) : q \in Q\}, \{\mathsf{inv}_q(\vec{x}) : q \in Q\},$$
$$\{\mathsf{jump}_{q \to q'}(\vec{x}_t, \vec{x}'_0) : q, q' \in Q\}, \{\mathsf{init}_q(\vec{x}) : q \in Q\}\rangle$$

where $X \subseteq \mathbb{R}^n$ for some $n \in \mathbb{N}$, $Q = \{q_1, ..., q_m\}$ is a finite set of modes, and the other components are finite sets of quantifier-free $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas.

Figure 1 lists the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ encoding of a k-step M-delay SMT instance for hybrid system H and reachability property $\mathsf{goal}_{q_G}(\vec{x}^t_k)$. The first line includes the bounded (superscripts) existential quantifiers over continuous variables at the start ($\vec{x}_i$) and end ($\vec{x}^t_i$) of each mode as well as the times of mode jumps t_i (relative to the start of the source mode). The second line states the initial value of each continuous variable, flows, and invariants in the initial mode. The enforce literal is used to simplify encoding mode sequences. The third and fourth lines encode possible mode transitions at each step i, as well as how the destination mode behaves (i.e., its flows and invariants). We note that the set of possible transitions $h_k(Q)$ will include all transitions at each time by default. In the next section, we show how to remove impossible transitions from this set as a by-product of computing reachability heuristics. The final line encodes the goal that must be met in step k.

Encoding Hybrid Planning as a Hybrid Automaton

The hybrid encoding of a PDDL+ planning task defines:

- $X = Fs$, where Fs is the set of ground functions.
- $Q = 2^P \times_{ps \in PS} \mathsf{pre}_X(ps)$, where each mode corresponds to a subset of propositions that are true and a subset of the processes' numeric preconditions that are satisfied. In the following, we use modes $q \in Q$ and the propositions and numeric process conditions corresponding to q interchangibly.

$$\exists^X \vec{x}_0 \exists^X \vec{x}_0^t \cdots \exists^X \vec{x}_k \exists^X \vec{x}_k^t \exists^{[0,M]} t_0 \cdots \exists^{[0,M]} t_k.$$

$$\mathsf{init}_{q_0}(\vec{x}_0) \wedge \mathsf{flow}_{q_0}(\vec{x}_0, \vec{x}_0^t, t_0) \wedge \mathsf{enforce}(q_0, 0) \wedge \forall^{[0,t_0]} t \forall^X \vec{x}\ (\mathsf{flow}_{q_0}(\vec{x}_0, \vec{x}, t) \rightarrow \mathsf{inv}_{q_0}(\vec{x})) \wedge$$

$$\bigwedge_{i=0}^{k-1} \bigvee_{(q,q',i) \in h_k(Q)} \Big(\mathsf{jump}_{q \rightarrow q'}(\vec{x}_i^t, \vec{x}_{i+1}) \wedge \mathsf{flow}_{q'}(\vec{x}_{i+1}, \vec{x}_{i+1}^t, t_{i+1}) \wedge \mathsf{enforce}(q, q', i) \wedge \mathsf{enforce}(q', i+1) \wedge$$

$$\forall^{[0,t_{i+1}]} t \forall^X \vec{x}\ (\mathsf{flow}_{q'}(\vec{x}_{i+1}, \vec{x}, t) \rightarrow \mathsf{inv}_{q'}(\vec{x}))) \Big)$$

$$\wedge\ \mathsf{goal}_{q_G}(\vec{x}_k^t) \wedge \mathsf{enforce}(q_G, k).$$

Figure 1: SMT-encoding of k-step Reachability

- $\mathsf{flow} = \{\mathsf{flow}_q(\vec{x}_0, \vec{x}_t, t) : q \in Q\}$ where $\mathsf{flow}_q(\vec{x}_0, \vec{x}_t, t)$ defines:

$$\bigwedge_{x \in X} \left(x_t = x_0 + \int_0^t \sum_{ps \in Ps(q,x)} \mathsf{eff}^x_{ps}(s) ds \right)$$

  where $Ps(q, x)$ is the set of processes effecting x that have their propositional preconditions satisfied by q and eff^x_{ps} is a process ps's effect upon x.

- $\mathsf{inv} = \{\mathsf{inv}_q(\vec{x}) : q \in Q\}$ where $\mathsf{inv}_q(\vec{x})$ defines:

$$\bigwedge_{ps \in Ps(q)} \bigwedge_{f(\vec{x}) \in \mathsf{pre}_X(ps)} f(\vec{x})$$

  where $Ps(q)$ is the set of all processes whose propositional preconditions are satisfied by q and $\mathsf{pre}_X(ps)$ is the set of numeric preconditions $f(\vec{x})$ of ps.

- $\mathsf{jump} = \{\mathsf{jump}_{q \rightarrow q'}(\vec{x}_t, \vec{x}'_0)\}$ where each jump corresponds to an instantaneous action $a \in As$. If the discrete preconditions of a are satisfied by q, and $q' = q \backslash \mathsf{eff}^-_P(a) \cup \mathsf{eff}^+_P(a)$ (using the respective negative and positive effects), then the jump $\mathsf{jump}^a_{q \rightarrow q'}(\vec{x}_t, \vec{x}'_0)$ corresponding to a defines:

$$\bigwedge_{f(\vec{x}_t) \in \mathsf{pre}_X(a)} f(\vec{x}_t) \wedge \bigwedge_{f(\vec{x}_t, \vec{x}'_0) \in \mathsf{eff}_X(a)} f(\vec{x}_t, \vec{x}'_0) \wedge t \geq \epsilon$$

  where $\mathsf{pre}_X(a)$ is a set of numeric preconditions and $\mathsf{eff}_X(a)$ is a set of discrete numeric effects.

- $\mathsf{init} = \mathsf{init}_{q_0}(\vec{x})$ where q_0 is the mode consistent with the initial state propositions and $\mathsf{init}_{q_0}(\vec{x})$ defines:

$$\bigwedge_{f(\vec{x}_0) \in Init_x} f(\vec{x}_0)$$

  where $Init_x$ is the set of initial function assignments.

We state the goal in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ as $\mathsf{goal}_{q_G}(\vec{x}_k^t)$.

Example: Vehicle Planning Problem

A simplified version of the PDDL+ vehicle problem (Fox and Long 2006) includes the following actions and processes:

```
(:action start
 :precondition ()
 :effect (run))
(:action accel
 :precondition (run)
 :effect (increase a 1))
(:process moving
 :precondition (run)
 :effect (increase v (* #t a)))
(:process drag
 :precondition (and (run) (> v 0))
 :effect (decrease v (* #t (* 0.1 (^ v 2)))))
```

The initial state includes zero propositions indicating the vehicle is not running and two continuous variables indicating its acceleration and velocity:

```
(= a 0) (= v 0)
```

The goal is to increase the velocity by no less than 0.01 units:

```
(>= v 0.01)
```

The $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ encoding of the hybrid system corresponding to this problem defines:

- $X = \{v, a\}$, variables for velocity and acceleration.
- $Q = \{\{\}, \{run\}, \{v > 0\}, \{run, v > 0\}\}$, for states where run is false or true and the velocity is or is not greater than zero.
- $\mathsf{flow}_{\{\}}(\vec{x}_0, \vec{x}_t, t)$ is $(a_t = a_0) \wedge (v_t = v_0)$ $\mathsf{flow}_{\{run\}}(\vec{x}_0, \vec{x}_t, t)$ is $(a_t = a_0) \wedge (v_t = v_0 + \int_0^t (a(s)) ds)$ $\mathsf{flow}_{\{v>0\}}(\vec{x}_0, \vec{x}_t, t)$ is

$$(a_t = a_0) \wedge (v_t = v_0 + \int_0^t -0.1 v(s)^2 ds)$$

  $\mathsf{flow}_{\{run, v>0\}}(\vec{x}_0, \vec{x}_t, t)$ is

$$(a_t = a_0) \wedge (v_t = v_0 + \int_0^t a(s) - 0.1 v(s)^2 ds)$$

  .

- $\mathsf{inv}_{\{v>0\}}$ and $\mathsf{inv}_{\{run, v>0\}}$ is $v > 0$.
- $jump^{start}_{\{\} \rightarrow \{run\}}(\vec{x}_t, \vec{x}'_0)$ is

$$(a'_0 = a_t \wedge v'_0 = v_t \wedge t \geq \epsilon)$$

  $jump^{accel}_{\{run\} \rightarrow \{run\}}(\vec{x}_t, \vec{x}'_0)$ is

$$(a'_0 = a_t + 1 \wedge v'_0 = v_t \wedge t \geq \epsilon)$$

  $jump^{accel}_{\{run\} \rightarrow \{run, v>0\}}(\vec{x}_t, \vec{x}'_0)$ is

$$(a'_0 = a_t \wedge v'_0 = v_t \wedge t \geq \epsilon)$$

- $\mathsf{init}_{\{\}}(\vec{x}_0)$ is $(a_0 = 0) \wedge (v_0 = 0)$
- $\mathsf{goal}_{\{run, v>0\}}(\vec{x}_t)$ is $(v_2^k \geq 0.01) \wedge \mathsf{enforce}(\{run, v > 0\}, 2)$

Heuristics & Encoding Reduction

Modern DPLL-style SAT solvers commonly guide search by branching upon variables appearing most frequently in recent conflict clauses (Moskewicz et al. 2001), such as by using the VSIDS heuristic. Recent work in planning as SAT (Rintanen 2012) shows that planning-specific heuristics improve considerably over problem agnostic variable selection heuristics. The intuition behind the planning as SAT heuristic is to emulate plan space search that performs means-ends reasoning to support each (sub)goal.

We adapt a similar form of means-ends reasoning to hybrid systems as SMT. We select a set of assignments to the $\mathsf{enforce}(q, q', i)$ and $\mathsf{enforce}(q, i)$ literals that encode a feasible k-step path through the modes. By feasible, we mean that the path includes possible jumps, while ignoring the guards and invariants. In selecting the discrete path, the SAT solver can essentially remove all disjunction from the SMT encoding and assign the remaining literals with unit propagation. Without this strategy, the SAT solver assigns the enforce literals in an arbitrary order, only discovering conflicts (due to infeasible discrete paths) after many decisions. While the VSIDS heuristic might be able to learn an ordering of the enforce literals, our approach extracts it from the problem definition and circumvents unnecessary search.

Mode Path Suggestions: Our approach is to maintain a queue $\mathcal{Q}$ of enforce literal assignment suggestions. The SAT solver removes assignments (if present) from the queue and adds them to the current assignment. If the SAT solver backtracks, then we reconstruct $\mathcal{Q}$ with a new set of assignments that are: 1) in agreement with the current DPLL search assignment $\mathcal{D}$, and 2) represent an unexplored discrete path.

We construct a depth first search stack of modes $S = [S_0, \ldots, S_k]$, where each S_i is a list of possible modes at step i. For each mode q in S_i, $0 \leq i < k$ there exists a $\mathsf{jump}_{q \to q'} \in \mathsf{jump}$ where q' is the head of $S_{i+1} = [q'|\_]$. Given S, we define $\mathcal{Q} = \mathcal{Q}_k, \ldots \mathcal{Q}_0$. $\mathcal{Q}_k$ includes $\mathsf{enforce}(q, k)$ for $S_k = [q|\_]$ and $\neg\mathsf{enforce}(q', k)$ for each $q' \in Q, q \neq q'$. Similarly, $\mathcal{Q}_i$, $0 \leq i < k$, includes $\mathsf{enforce}(q, q', i)$ for the $S_i = [q|\_]$, such that $S_{i+1} = [q'|\_]$, and $\neg\mathsf{enforce}(q'', q''', k)$ for each $q'', q''' \in Q, q'' \neq q, q''' \neq q'$. $\mathcal{Q}_0$ is defined similar to $\mathcal{Q}_i$, $0 < i < k$, except that it also includes $\mathsf{enforce}(q, 0)$ and $\neg\mathsf{enforce}(q'', 0)$ literals.

We maintain the search stack S by two algorithms `backtrack` and `extend`. We initialize S with `extend` and then use `backtrack` followed by `extend` to construct a new path that is both in agreement with $\mathcal{D}$ and suggests an unexplored discrete path.

We do not list the pseudocode for `backtrack`, for lack of space and its relative simplicity. The `backtrack` algorithm serves two purposes: to align the stack S with $\mathcal{D}$ and to backtrack from dead ends reached by the `extend` algorithm. Aligning S with $\mathcal{D}$ ensures that $\mathsf{enforce}(q, q', i) \in \mathcal{D}$ iff there exists an $S_i = [q|S_i^{rest}]$ and $S_{i+1} = [q'|S_{i+1}^{rest}]$. Backtracking from dead ends involves i) popping each S_i from stack S until the current S_i has more than one element, and ii) popping the first element of S_i.

The `extend` algorithm (Algorithm 1) expands a current stack S until it includes a mode for each step, or it exhausts all possible mode sequences and fails (meaning the problem is unsatisfiable). Lines 5-9 involve generating the children at step i. Lines 10-17 backtrack if there are no children and return Failure if there are no backtrack points. Lines 18-22 add the children to the stack and sort the children by their forward cost.

Algorithm 1: Extend a choice stack to a complete path.

```
1  extend($S$, $\mathcal{D}$, $h_k(Q)$)
   input  : A partial mode choice stack $S = [S_j, \ldots, S_k]$;
            a set of literal assignments $\mathcal{D}$; and a set of
            allowed transitions $h_k(Q)$.
   output: A complete mode choice stack $S$
2  for i from $j - 1$ to 0 do
3      $S_i \leftarrow []$;
4      //Expand $q'$ at step $i + 1$, generate children at step $i$;
5      for $q \in Q$ do
6          if $(q, q', i) \in h_k(Q)$, $S_{i+1} = [q'|\_]$, and
           $\neg\mathsf{enforce}(q, q', i) \notin D$ then
7              $S_i \leftarrow [q|S_i]$;
8          end
9      end
10     if $|S_i| = 0$ then
11         //$q'$ is dead end;
12         $S \leftarrow$ backtrack$(S, D)$;
13         if $|S| = 0$ then
14             return Failure
15         else
16             $i \leftarrow |S| - 1$
17         end
18     else
19         //prioritize by forward cost from initial state;
20         $S_i \leftarrow \text{sort}_{fc}(S_i)$;
21         $S \leftarrow [S_i|S]$;
22     end
23 end
```

Reachability Heuristic: The forward cost of a mode is the fewest jumps required to reach it from the initial mode:

$$fc(q') = \begin{cases} 0 & : \mathsf{init}_{q'}(\overrightarrow{x}) \\ \min\left(\min\limits_{\mathsf{jump}_{q \to q'} \in \mathsf{jump}} fc(q) + 1, \infty\right) & : \text{otherwise} \end{cases}$$

We define the backward cost of a mode as the fewest jumps required for it to reach a goal mode:

$$bc(q) = \begin{cases} 0 & : \mathsf{goal}_q(\overrightarrow{x}) \\ \min\left(\min\limits_{\mathsf{jump}_{q \to q'} \in \mathsf{jump}} bc(q') + 1, \infty\right) & : \text{otherwise} \end{cases}$$

Using the forward and backward costs, we can prune the set of allowed jumps and define:

$$h_k(Q) = \{(q, q', i) | \mathsf{allow}_k(q, q', i), q, q' \in Q\}$$

where $\mathsf{allow}_k(q, q', i)$ is defined as $fc(q) \leq i \wedge fc(q') \leq i + 1 \wedge bc(q) \leq k - i \wedge bc(q') \leq k - i + 1$.

Encoding Reduction: The forward and backward reachability analyses help prune both the enforce literals suggested by the depth first search and the clauses in Figure 1 (c.f., the disjunction over $h_k(Q)$ in line 3). The heuristic prunes solely based upon the discrete aspects of the hybrid system. It can be extended to consider the continuous aspects as well, for example, by using the intervals propagated by Metric-FF (Hoffmann 2003).

ICP Time Splitting Heuristic: In addition to the SAT heuristic for selecting mode sequences, we also make use of a simple

ICP heuristic for planning. We observe that many actions (e.g., starting the car) can happen as quickly as possible, even though delaying their execution is possible. We force ICP to split time variables t_i in a special manner, but rely upon dReal's existing ICP implementation to select variables to branch. The first time that ICP selects a time variable t_i with interval $[lb(t_i), ub(t_i)]$, we split its interval into the two subintervals $[lb(t_i), lb(t_i) + \epsilon]$ and $[lb(t_i) + \epsilon, ub(t_i)]$. In subsequent splits, we use the default split $[lb(t_i), lb(t_i) + (ub(t_i) - lb(t_i))/2]$ and $[lb(t_i) + (ub(t_i) - lb(t_i))/2, ub(t_i)]$ because the value for t_i must be relatively greater than ϵ otherwise.

Implementation: The mode path selection and time splitting heuristics are implemented by extending existing mechanisms with dReal. dReal is based upon the OpenSMT solver (Bruttomesso et al. 2010), which provides a literal suggestion interface between the SAT solver and each theory solver. In dReal, the theory solver (i.e., ICP) implements the heuristic by reasoning about the hybrid system specification and the current (partial) literal assignment. It communicates these suggestions via the existing interface. The ICP solver wraps the Realpaver (Granvilliers and Benhamou 2006) engine by customizing its algorithms for variable and value selection. The time splitting heuristic is implemented as a custom value-selection/branching heuristic that decides where to split an interval and which side of the split to select first.

Empirical Evaluation

We first show experiments that evaluate dReal with and without the heuristics described in the previous section and for various values of δ. We report results for the minimal step encoding required to solve each instance. In all cases, dReal can detect unsatisfiability of encodings using fewer than the minimal steps in negligible time. We use the generator and car domains from the literature (Bogomolov et al. 2014) and a new domain called Dribble. Next, we compare dReal externally with existing PDDL+ planners on linear problems (using results from (Bogomolov et al. 2014)). The existing planners include SpaceEx (Bogomolov et al. 2014), CoLin (Coles et al. 2012), and UPMurphi (Della Penna et al. 2009). In all experiments comparing with other planners, dReal uses all heuristics and $\delta = 0.1$. We will see that there is still much room for dReal to improve on linear problems by incorporating techniques from leading tools such as SpaceEx.

Domains: The car domain includes only instantenous actions and processes, and is very similar to our running example. The differences are that it includes a decelerate (decel) action and distance variable, it requires that velocity is zero and distance is thirty in the goal, and it scales by imposing increasing limits on the magnitude of the acceleration function. Additional acceleration or deceleration increases the branching factor of the problem. The linear and nonlinear versions of the domain differ in whether they include the nonlinear drag process (as shown in our example).

The dribble domain involves three processes effecting 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) and oppose, which decrease velocity by $f \in \{0, 1, 2, 4\}$ or increase velocity by $-0.9v$. The dribble action has the precondition that velocity is zero, and the oppose action, that the ball position is zero. The initial state places the ball at $x = 1$ with velocity $v = 0$ and the goal is to reach $1.5 \leq x \leq 3.0$.

The generator domain is normally encoded with durative actions for pouring fuel into the generator and running the generator. We translated the problem to use two instantaneous actions and a process for each durative action. Thus, as the number of tanks x increases (in each instance) the number of steps required by dReal's encoding is $2(x+1)$ (the start and end for each tank pouring action and the start and end for running the generator). We also created a nonlinear version that models how fuel pours more quickly over time. The pouring process defines the effects as

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

or

```
(increase ptime (* #t 1))
(increase fuel (* #t (* 0.001 (^ ptime 2))))
```

Internal Comparison: We evaluate how well dReal performs on nonlinear versions of the domains as well as how it is sensitive to the use of heuristics and the choice of δ.

δ	SAT Heur.	ICP Heur.	1	2	3
1	●	●	5.36/2/19	5.36/2/19	5.36/3/19
0.1	●	●	16.62/2/58	16.64/2/58	16.25/3/58
0.01	●	●	15.20/2/66	15.17/2/66	14.81/3/66
1		●	9.19/746/20	447.57/6727/96	27.59/3697/18
0.1		●	17.66/746/59	384.82/6980/254	35.65/3697/57
0.01		●	19.16/746/67	378.19/6980/230	37.17/3697/65
1	●		34.83/2/181	34.54/2/181	34.67/3/181
0.1	●		-	-	-
0.01	●		-	-	-
1			40.45/746/182	453.62/6727/102	58.30/3697/180
0.1			-	-	-
0.01			-	-	-

Table 1: dReal Runtime (s)/SAT Nodes/ICP Nodes results on nonlinear car. "-" indicates a timeout.

SAT Heur.	2	4	6	8	10
●	192.52	33.50	65.16	122.91	224.61
	595.59	57.24	79.16	-	-

Table 2: dReal Runtime (s) results for Dribble with increasing plan lengths $k \in \{2, 4, 6, 8, 10\}$ with and without the SAT mode sequence heuristic. "-" indicates a timeout.

Dom	1	2	3	4	5	6	7	8
Car	16.62	16.64	16.25	16.77	16.56	16.79	17.44	16.6
Gen	12.80	71.63	1696.84	-	-	-	-	-

Table 3: Runtime results (s) on nonlinear generator and car. "-" indicates a timeout.

First, Table 1 lists the runtime, number of SAT decisions (nodes) and number of ICP decisions (nodes) for the nonlinear car domain instances one through three. The table lists results for values of $\delta \in \{1, 0.1, 0.01\}$ and whether dReal uses the SAT variable selection heuristic and the ICP time branching heuristic.

The trends shown in Table 1 are that: i) decreasing δ will always increase the number of ICP nodes and sometimes the SAT nodes, ii) the ICP heuristic has a substantial impact upon the number of ICP nodes, and iii) the SAT heuristic impacts the number of SAT nodes, and holds the number of SAT nodes roughly constant as the instances scale. Improving ICP through better variable selection and constraint propagation will have the most impact on these instances because the SAT heuristic already reduces the number of

SAT nodes to a minimum. This observation applies to the other domains as well. dReal is able to find a feasible sequence of mode transitions quickly, but struggles to show δ-satisfiability when it must tighten the boxes around variables encoding several steps.

Table 2 lists the results for the Dribble domain with and without the SAT mode sequence heuristic. The ICP time splitting heuristic did not change performance in this domain and is omitted from the table. The results illustrate that the SAT heuristic can help reduce total time by considering fewer plans and that the ICP search is effective at verifying a plan satisfies the nonlinear continuous constraints. We note that the instance with $k = 2$ is particularly challenging for ICP because the vertical position is almost exactly 1.5 at the end of the plan and ICP automatically selects a very fine step size for its interval-based integration.

To evaluate scalability, we show in Table 3 the running time on the car and generator domains, using both heuristics and $\delta = 0.1$. (The results for dribble with the same steps have been included in Table 2.) The instances scale by adding the indicated number of acceleration and deceleration steps in car, or by adding the indicated number of tanks to refill the generator. Dribble scales by increasing the number of possible plan steps.

External Comparisons: Table 4 compares dReal with the other planners on linear instances of the generator and car domains. As above, each instance scales the respective number of tanks to fill the generator (where each tank is required) and levels of acceleration/deceleration. We see that dReal is competitive on the car domain, but scales relatively poorly on the generator domain. Upon closer inspection, the reason that dReal scales poorly is that it spends considerable time in its ICP branch and prune search after finding a feasible mode path. We note that during constraint propagation, dReal does not use pruning operators that are optimized for linear functions. Recognizing and exploiting linearity could improve performance considerably. dReal scales well in the car domain because its heuristic uses reachability information to prune unreachable modes. As we show in Table 5, dReal's performance without the heuristic is much worse.

Dom	Planner	1	2	3	4	5	6	7	8
Gen	dReal	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	dReal	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	CoLin	x	x	x	x	x	x	x	x
Car	UPMur	28.44	386.5	-	-	-	-	-	-

Table 4: Runtime results (s) on linear generator and car. "-" indicates a timeout.

Discussion: dReal and SpaceEx find plan tubes, whereas, CoLin and UPMurphi find one or more concrete plans. By concrete, we mean that each action executes at a single, fixed time point. In some restricted cases, dReal and SpaceEx can find concrete plans. In general, constructing a concrete plan from a plan tube is undecidable (Gao, Avigad, and Clarke 2012). One remedy is to make dReal more precise than the plan validator (e.g., VAL (Fox and Long 2003)) by setting δ smaller than the ϵ tolerance used by the validator and then selecting an arbitrary concrete plan consistent with the tube. The best we can hope to attain is a precision greater than the plan executor/validator. In contrast with dReal, SpaceEx is unable to make similar guarantees about its plan tubes because it does not prune the tubes, rather it maximizes its representation of reachable states to improve state subsumption checks prior to plan extraction. Nevertheless, both SpaceEx and dReal might identify plan tubes that do not in fact contain a concrete plan; however, both can reliably prove plan non-existence.

δ	SAT Heur.	ICP Heur.	1	2	3
1	●	●	0.45/2/6	0.48/2/6	0.55/3/6
0.1	●	●	0.55/2/8	0.54/2/8	0.60/3/8
0.01	●	●	0.66/2/10	0.66/2/10	0.73/3/10
1		●	6.82/1302/6	10.83/2286/9	20.71/4196/8
0.1		●	6.76/1302/8	11.81/2286/6	21.12/4196/10
0.01		●	6.96/1302/10	10.93/2286/11	20.96/4196/12
1	●		0.88/2/15	0.91/2/15	1.05/3/15
0.1	●		1.77/2/29	2.07/2/29	2.03/3/29
0.01	●		2.22/2/36	2.77/3/36	2.42/3/36
1			8.78/1302/15	15.40/2286/18	28.15/4196/17
0.1			9.04/1302/29	14.04/2286/27	26.26/4196/31
0.01			10.61/1302/36	16.37/2286/39	25.86/4196/38

Table 5: dReal Runtime (s)/SAT Nodes/ICP Nodes results on linear car.

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 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).

LP-SAT (Shin and Davis 2005) is very similar in spirit to 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 makes use of the hybrid system semantics of PDDL+ in a very direct fashion. LP-SAT more closely resembles classical planning as SAT encodings. Unlike our work, LP-SAT does not incorporate heuristics that are aware the encoding represents a planning problem.

Bogomolov et al. (2014) and Della Penna et al. (2009), like our work, makes 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.

Conclusion

We present a new approach to PDDL+ planning that compiles problems into the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ language. The encoding can be checked for satisfiability by a state of the art SMT solver that supports nonlinear continuous change, which is a first for PDDL+ planning. Due to the nature of the problem, the solver finds plan tubes by solving a δ relaxation of the problem. We have shown that δ is a useful parameter that trades SMT solver effort for solution accuracy.

In addition to showing how to encode PDDL+ planning as a hybrid system in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$, we have presented reachability heuristics that help prune the SMT encoding. These heuristics also inform an SMT variable selection heuristic procedure in the dReal solver. The procedure helps dReal overcome one its main inefficiencies of constructing infeasible sequences of modes unnecessarily. The resulting scale up in performance helps dReal solve challenging PDDL+ benchmarks, putting it on par with the state of the art.

Acknowledgements: This work was supported under ONR Contract XXXX.

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.

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.

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. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Int. Res.* 20(1):61–124.

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

Fox, M.; Howey, R.; and Long, D. 2006. Exploration of the robustness of plans. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, 834. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

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 *IJCAR*, 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.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, 140–149. AAAI.

Henzinger, T. A. 1996. The theory of hybrid automata. In *LICS*, 278–292.

Hoffmann, J. 2003. The metric-ff planning system: Translating ”ignoring delete lists” to numeric state variables. *J. Artif. Intell. Res.(JAIR)* 20:291–341.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, 530–535. ACM.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artif. Intell.* 193:45–86.

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.