TLP **17** (5–6): 924–941, 2017. © Cambridge University Press 2017
doi:10.1017/S1471068417000412 First published online 23 August 2017

Representing hybrid automata by action language modulo theories[*]

JOOHYUNG LEE and NIKHIL LONEY
School of Computing, Informatics and Decision Systems Engineering,
Arizona State University, Tempe, AZ, USA
(*e-mail:* `joolee@asu.edu, nloney@asu.edu`)

YUNSONG MENG
Houzz, Inc., Palo Alto, CA, USA
(*e-mail:* `Yunsong.Meng@asu.edu`)

submitted 2 May 2017; revised 20 June 2017; accepted 4 July 2017

Abstract

Both hybrid automata and action languages are formalisms for describing the evolution of dynamic systems. This paper establishes a formal relationship between them. We show how to succinctly represent hybrid automata in an action language which in turn is defined as a high-level notation for answer set programming modulo theories—an extension of answer set programs to the first-order level similar to the way satisfiability modulo theories (SMT) extends propositional satisfiability (SAT). We first show how to represent linear hybrid automata with convex invariants by an action language modulo theories. A further translation into SMT allows for computing them using SMT solvers that support arithmetic over reals. Next, we extend the representation to the general class of non-linear hybrid automata allowing even non-convex invariants. We represent them by an action language modulo ordinary differential equations, which can be compiled into satisfiability modulo ordinary differential equations. We present a prototype system CPLUS2ASPMT based on these translations, which allows for a succinct representation of hybrid transition systems that can be computed effectively by the state-of-the-art SMT solver `dReal`.

KEYWORDS: Answer set programming, Action languages, Hybrid automata.

1 Introduction

Both hybrid automata (Henzinger 1996) and action languages (Gelfond and Lifschitz 1998) are formal models for describing the evolution of dynamic systems. The focus of hybrid automata is to model continuous transitions as well as discrete changes, but, unlike action languages, their discrete components are too simple to represent complex relations among fluents and various properties of actions. On the other hand, transitions described by most action languages are limited to discrete changes only, which hinders action languages from

[*] This work was partially supported by the National Science Foundation under Grants IIS-1319794 and IIS-1526301.

modeling real-time physical systems. One of the exceptions is an enhancement of action language $\mathcal{C}+$ (Lee and Meng 2013), which extends the original, propositional language in the paper by Giunchiglia *et al.* (2004) to the first-order level. The main idea there is to extend the propositional $\mathcal{C}+$ to the first-order level by defining it in terms of answer set programming modulo theories (ASPMT)—a tight integration of answer set programs and satisfiability modulo theories (SMT) to allow SMT-like effective first-order reasoning in answer set programming (ASP).

This paper establishes a formal relationship between hybrid automata and action language $\mathcal{C}+$. We first show how to represent *linear hybrid automata* with *convex invariants* by the first-order $\mathcal{C}+$. A further translation into SMT allows for computing them using state-of-the-art SMT solvers that support arithmetic over reals. However, many practical domains of hybrid systems involve non-linear polynomials, trigonometric functions, and differential equations that cannot be represented by linear hybrid automata. Although solving the formulas with these functions is undecidable in general, Gao *et al.* (2013a) presented a novel approach called a "δ-complete decision procedure" for computing such SMT formulas, which led to the concept of "satisfiability modulo ordinary differential equations (ODEs)[1]." The procedure is implemented in the SMT solver `dReal` (Gao *et al.* 2013b), which is shown to be useful for formalizing the general class of hybrid automata. We embrace the concept into action language $\mathcal{C}+$ by introducing two new abbreviations of causal laws, one for representing the evolution of continuous variables as specified by ODEs and another for describing invariants that the continuous variables must satisfy when they progress. The extension is rather straightforward, thanks to the close relationship between ASPMT and SMT: ASPMT allows for quantified formulas as in SMT, which is essential for expressing non-convex invariants; algorithmic improvements in SMT can be carried over to the ASPMT setting. We show that the general class of hybrid automata containing non-convex invariants can be expressed in the extended $\mathcal{C}+$ modulo ODEs.

The extended $\mathcal{C}+$ allows us to achieve the advantages of both hybrid automata and action languages, where the former provides an effective way to represent continuous changes, and the latter provides an elaboration tolerant way to represent (discrete) transition systems. In other words, the formalism gives us an elaboration tolerant way to represent hybrid transition systems. Unlike hybrid automata, the structured representation of states allows for expressing complex relations between fluents, such as recursive definitions of fluents and indirect effects of actions, and unlike propositional $\mathcal{C}+$, the transitions described by the extended $\mathcal{C}+$ are no longer limited to discrete ones only; the advanced modeling capacity of action languages, such as additive fluents, statically defined fluents, and action attributes, can be achieved in the context of hybrid reasoning.

We implemented a prototype system CPLUS2ASPMT based on these translations, which allows for a succinct representation of hybrid transition systems in language $\mathcal{C}+$ that can be compiled into the input language of `dReal`. We show that the system can be used for reasoning about hybrid transition systems, whereas other action language implementations,

[1] A δ-complete decision procedure for an SMT formula F returns false if F is unsatisfiable, and returns true if its syntactic "numerical perturbation" of F by bound δ is satisfiable, where $\delta > 0$ is number provided by the user to bound on numerical errors. The method is practically useful since it is not possible to sample exact values of physical parameters in reality.

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

such as the Causal Calculator (Giunchiglia *et al.* 2004), CPLUS2ASP (Babb and Lee 2013), and COALA (Gebser *et al.* 2010) cannot.

The paper is organized as follows. In Section 2, we give a review of hybrid automata to set up the terminologies used for the translations. Section 3 presents how to represent the special class of linear hybrid automata with convex invariants by $\mathscr{C}+$ modulo theory of reals. Section 4 introduces two new abbreviations of causal laws that can be used for modeling invariant and flow conditions. Section 5 uses these new constructs to represent the general class of non-linear hybrid automata and shows how to reduce them to the input language of `dReal` leading to the implementation of system CPLUS2ASPMT, a variant of the system CPLUS2ASP.

The proofs of the theorems and the examples of hybrid automata in the input language of CPLUS2ASPMT can be found in the online appendix accompanying the paper at the TPLP archive (Lee *et al.* 2017).

2 Preliminaries

2.1 Review: hybrid automata

We review the definition of hybrid automata (Henzinger 1996; Alur *et al.* 2000), formulated in terms of a logical language by representing arithmetic expressions by many-sorted first-order formulas under background theories, such as QF_NRA (Quantifier-Free Non-linear Real Arithmetic) and QF_NRA_ODE (Quantifier-Free Non-linear Real Arithmetic with Ordinary Differential Equations). By $\mathscr{R}$, we denote the set of all real numbers and by $\mathscr{R}_{\geqslant 0}$ the set of all non-negative real numbers. Let X be a set of real variables. An arithmetic expression over X is an atomic formula constructed using functions and predicates from the signature of the background theory and elements from $\mathscr{R} \cup X$. Let $A(X)$ be an arithmetic expression over X and let x be a tuple of real numbers whose length is the same as the length of X. By $A(x)$, we mean the expression obtained from A by replacing variables in X with the corresponding values in x. For an arithmetic expression with no variables, we say that A *is true* if the expression is evaluated to true in the background theory.

A *Hybrid Automaton* $\mathscr{H}$ consists of the following components:

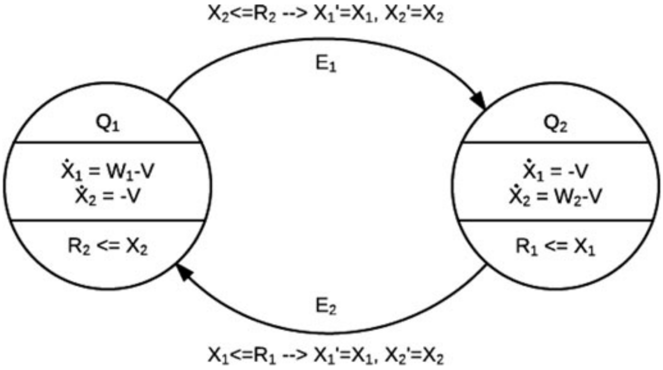
- **Variables:** A finite list of real-valued variables $X = (X_1, \ldots, X_n)$. The number n is called the *dimension* of $\mathscr{H}$. We write $\dot{X}$ for the list $(\dot{X}_1, \ldots, \dot{X}_n)$ of dotted variables, representing first derivatives during a continuous change, and X' for the set $(X'_1, \ldots, X'_n)$ of primed variables, representing the values at the conclusion of the discrete change. $X_0 \subseteq X$ is the set of initial states. We use lowercase letters to denote the values of these variables.
- **Control graph:** A finite directed graph $\langle V, E \rangle$. The vertices are called *control modes*, and the edges are called *control switches*.
- **Initial, invariant, and flow Conditions:** Three vertex labeling functions, Init, Inv, and Flow, that assign to each control mode $v \in V$ three first-order formulas:
  - — $\mathsf{Init}_v(X)$ is a first-order formula whose free variables are from X. The formula constrains the initial condition.
  - — $\mathsf{Inv}_v(X)$ is a first-order formula whose free variables are from X. The formula constrains the value of the continuous part of the state while the mode is v.

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

— $\mathsf{Flow}_v(X, \dot{X})$ is a set of first-order formulas whose free variables are from $X \cup \dot{X}$. The formula constrains the continuous variables and their first derivatives.

- **Events:** A finite set Σ of symbols called *h-event*s and a function, $\mathsf{hevent} : E \rightarrow \Sigma$, that assigns to each edge a unique h-event.
- **Guard:** For each control switch $e \in E$, $\mathsf{Guard}_e(X)$ is a first-order formula whose free variables are from X.
- **Reset:** For each control switch $e \in E$, $\mathsf{Reset}_e(X, X')$ is a first-order formula whose free variables are from $X \cup X'$.

Example 1



The figure shows a hybrid automaton for the Water Tank Example from the lecture note by Lygeros (2004), which consists of two variables $X = (X_1, X_2)$, two h-events E_1 and E_2, and two control modes $V = \{Q_1, Q_2\}$. For example,

- $\mathsf{Flow}_{Q_1}(\dot{X}_1, \dot{X}_2)$ is $\dot{X}_1 = \mathrm{W} - \mathrm{V}_1 \wedge \dot{X}_2 = -\mathrm{V}_2$.
- $\mathsf{Inv}_{Q_1}(X_1, X_2)$ is $X_2 \geqslant \mathrm{R}_2$.
- $\mathsf{Guard}_{(Q_1,Q_2)}(X_1, X_2)$ is $X_2 \leqslant \mathrm{R}_2$.
- $\mathsf{Reset}_{(Q_1,Q_2)}(X_1, X_2, X_1{}', X_2{}')$ is $X_1' = X_1 \wedge X_2' = X_2$.

A *labeled transition system* consists of the following components:

- **State space:** A set Q of states and a subset $Q_0 \subseteq Q$ of initial states.
- **Transition relations:** A set A of labels. For each label $a \in A$, a binary relation $\rightarrow^a$ on the state space Q. Each triple $q \rightarrow^a q'$ is called a *transition*.

The *Hybrid Transition System* T_H of a Hybrid Automaton H is the labeled transition system obtained from H as follows.

- The set Q of *states* is the set of all (v, r) such that $v \in V$, $r \in \mathscr{R}^n$, and $\mathsf{Inv}_v(r)$ is true.
- $(v, r) \in Q_0$ iff both $\mathsf{Init}_v(r)$ and $\mathsf{Inv}_v(r)$ are true.
- The transitions are labeled by members from $A = \Sigma \cup \mathscr{R}_{\geqslant 0}$.
- $(v, r) \rightarrow^\sigma (v', r')$, where $(v, r), (v', r') \in Q$ and σ is an h-event in Σ, is a *transition* if there is an edge $e = (v, v') \in E$ such that (1) $\mathsf{hevent}(e) = \sigma$, (2) the sentence $\mathsf{Guard}_e(r)$ is true, and (3) the sentence $\mathsf{Reset}_e(r, r')$ is true.

- $(v, r) \rightarrow^{\delta} (v, r')$, where $(v, r), (v, r') \in Q$ and δ is a non-negative real, is a *transition* if there is a differentiable function $f : [0, \delta] \rightarrow \mathscr{R}^n$, with the first derivative $\dot{f} : [0, \delta] \rightarrow \mathscr{R}^n$ such that

(1) $f(0) = r$ and $f(\delta) = r'$,

(2) for all real numbers $\epsilon \in [0, \delta]$, $\mathsf{Inv}_v(f(\epsilon))$ is true and, for all real numbers $\epsilon \in (0, \delta)$, $\mathsf{Flow}_v(f(\epsilon), \dot{f}(\epsilon))$ is true. The function f is called the *witness* function for the transition $(v, r) \rightarrow^{\delta} (v, r')$.

2.2 Review: ASPMT and $\mathscr{C}$+

ASPMT (Bartholomew and Lee 2013) is a special case of many-sorted first-order (functional) stable model semantics from the papers by Ferraris *et al.* (2011) and by Bartholomew and Lee (2013) by restricting the background signature to be interpreted in the standard way, in the same way SMT restricts first-order logic.

The syntax of ASPMT is the same as that of SMT. Let σ^{bg} be the (many-sorted) signature of the background theory bg. An interpretation of σ^{bg} is called a *background interpretation* if it satisfies the background theory. For instance, in the theory of reals, we assume that σ^{bg} contains the set $\mathscr{R}$ of symbols for all real numbers, the set of arithmetic functions over real numbers, and the set $\{<, >, \leqslant, \geqslant\}$ of binary predicates over real numbers. Background interpretations interpret these symbols in the standard way.

Let σ be a signature that is disjoint from σ^{bg}. We say that an interpretation I of σ satisfies a sentence F w.r.t. the background theory bg, denoted by $I \models_{bg} F$, if there is a background interpretation J of σ^{bg} that has the same universe as I, and $I \cup J$ satisfies F. Interpretation I is a *stable model* of F relative to a set of function and predicate constants $\mathbf{c}$ (w.r.t. the background theory σ^{bg}) if $I \models_{bg} \mathrm{SM}[F; \mathbf{c}]$ [we refer the reader to the paper by Bartholomew and Lee (2013) for the definition of the SM operator].

In the paper by Lee and Meng (2013), action language $\mathscr{C}$+ was reformulated in terms of ASPMT and was shown to be useful for reasoning about hybrid transition systems. Appendix A (Lee *et al.* 2017) reviews this version of $\mathscr{C}$+.

3 Representing linear hybrid automata with convex invariants by $\mathscr{C}$+ modulo theories

3.1 Representation

Linear hybrid automata (Henzinger 1996) are a special case of hybrid automata where (i) the initial, invariant, flow, guard, and reset conditions are Boolean combinations of linear inequalities, and (ii) the free variables of flow conditions are from $\dot{X}$ only. In this section, we assume that for each $\mathsf{Inv}_v(X)$ from each control mode v, the set of values of X that makes $\mathsf{Inv}_v(X)$ true forms a convex region [2]. For instance, this is the case when $\mathsf{Inv}_v(X)$ is a *conjunction* of linear inequalities.

[2] A set X is *convex* if for any $x_1, x_2 \in X$ and any θ with $0 \leqslant \theta \leqslant 1$, we have $\theta x_1 + (1 - \theta)x_2 \in X$.

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

We show how a linear hybrid automata H can be turned into an action description D_H in $\mathcal{C}+$, and extend this representation to non-linear hybrid automata in the next section. We first define the signature of the action description D_H as follows.

- For each real-valued variable X_i in H, a simple fluent constant X_i of sort $\mathcal{R}$.
- For each control switch $e \in E$ and the corresponding $\mathsf{hevent}(e) \in \Sigma$, a Boolean-valued action constant $\mathsf{hevent}(e)$.
- An action constant *Dur* of sort non-negative reals.
- A Boolean action constant *Wait*.
- A fluent constant *Mode* of sort V (control mode).

The $\mathcal{C}+$ action description D_H consists of the following causal laws. We use lowercase letter x_i for denoting a real-valued variable. Let $X = (X_1, \ldots, X_n)$ and $x = (x_1, \ldots, x_n)$. By $X = x$, we denote the conjunction $(X_1 = x_1) \wedge \cdots \wedge (X_n = x_n)$.

- **Exogenous constants:**

$$\begin{gathered}\textbf{exogenous } X_i \quad (X_i \in X)\\ \textbf{exogenous } \mathsf{hevent}(e)\\ \textbf{exogenous } \mathit{Dur}.\end{gathered}$$

  Intuitively, these causal laws assert that the values of the fluents can be arbitrary. The action constant *Dur* is to record the duration that each transition takes (discrete transitions are assumed to have duration 0).

- **Discrete transitions:** For each control switch, $e = (v_1, v_2) \in E$:

  — **Guard**:

$$\textbf{nonexecutable } \mathsf{hevent}(e) \textbf{ if } \neg \mathrm{Guard}_e(X).$$

  The causal law asserts that an h-event cannot be executed if its guard condition is not satisfied.

  — **Reset**:

$$\textbf{constraint } \mathsf{Reset}_e(x, X) \textbf{ after } X = x \wedge \mathsf{hevent}(e) = \mathrm{TRUE}.$$

  The causal law asserts that if an h-event is executed, the discrete transition sets the new value of fluent X as specified by the reset condition.

  — **Mode and duration:**

$$\begin{array}{ll}\textbf{inertial } \mathit{Mode} = v & (v \in V)\\ \textbf{nonexecutable } \mathsf{hevent}(e) \textbf{ if } \mathit{Mode} \neq v_1 & \\ \mathsf{hevent}(e) \textbf{ causes } \mathit{Mode} = v_2 & \\ \mathsf{hevent}(e) \textbf{ causes } \mathit{Dur} = 0. & \end{array}$$

  The first causal law asserts the commonsense law of inertia on the control mode: the mode does not change when no action affects it. The second causal law asserts an additional constraint for an h-event to be executable (when the state is

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

in the corresponding mode). The third and fourth causal laws set the new control mode and the duration when the h-event occurs.

- **Continuous transitions:**

— **Wait**:

$$\begin{array}{l}\textbf{default } \mathit{Wait}{=}\text{TRUE}\\ \mathsf{hevent}(e) \textbf{ causes } \mathit{Wait}{=}\text{FALSE}.\end{array}$$

Wait is an auxiliary action constant that is true when no h-event is executed, in which case a continuous transition should occur.

— **Flow**: For each control mode $v \in V$ and for each $X_i \in X$,

$$\begin{array}{l}\textbf{constraint } \mathsf{Flow}_v((X-x)/\delta)\\ \qquad\qquad \textbf{after } X=x \wedge \mathit{Mode}=v \wedge \mathit{Dur}=\delta \wedge \mathit{Wait}{=}\text{TRUE} \quad (\delta>0)\\ \textbf{constraint } X=x \textbf{ after } X=x \wedge \mathit{Mode}=v \wedge \mathit{Dur}=0 \wedge \mathit{Wait}{=}\text{TRUE}.\end{array} \tag{3.1}$$

These causal laws assert that when no h-event is executed (i.e., *Wait* is true), the next values of the continuous variables are determined by the flow condition.

— **Invariant**: For each control mode $v \in V$,

$$\textbf{constraint } \mathit{Mode}{=}v \rightarrow \mathsf{Inv}_v(X). \tag{3.2}$$

The causal law asserts that in each state, the invariant condition for the control mode should be true.

It is easy to see from the assumption on the flow condition of linear hybrid automata that the witness function exists and is unique ($f(\epsilon) = x + \frac{x'-x}{\delta}\epsilon$); obviously, it is linear.

Note that (3.2) checks the invariant condition in each state only, not during the transition between the states. This does not affect the correctness because of the assumption that the invariant condition is convex and the flow condition is linear, from which it follows that

$$\forall \epsilon \in [0,\delta](\mathsf{Inv}_v(f(0)) \;\wedge\; \mathsf{Inv}_v(f(\delta)) \rightarrow \mathsf{Inv}_v(f(\epsilon))) \tag{3.3}$$

is true, where f is the witness function.

Figure 1 shows the translation of the Hybrid Automaton in Example 1 into $\mathcal{C}$+.

The following theorem asserts the correctness of the translation. By a *path* we mean a sequence of transitions[3].

Theorem 1

There is a 1:1 correspondence between the paths of the transition system of a hybrid automaton H and the paths of the transition system of the action description D_H.

The proof is immediate from the following two lemmas. First, we state that every path in the labeled transition system of T_H is a path in the transition system described by D_H.

[3] For simplicity of the comparison, as with action descriptions, the theorem does not require that the initial state of a path in the labeled transition system satisfy the initial condition. The condition can be easily added.

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

$q \in \{Q_1, Q_2\}$; t, x_1, x_2 are variables of sort $\mathcal{R}_{\geq 0}$. $\mathrm{W}_1, \mathrm{W}_2, \mathrm{V}$ are fixed real numbers

Simple fluent constants:	Sort:
X_1, X_2	$\mathcal{R}_{\geq 0}$
$Mode$	$\{Q_1, Q_2\}$
Action constants:	Sort:
E_1, E_2, $Wait$	Boolean
Dur	$\mathcal{R}_{\geq 0}$

% Exogenous constants:
exogenous X_1, X_2, E_1, E_2, Dur

% Guard:
nonexecutable E_1 **if** $\neg(X_2 \leq R_2)$ **nonexecutable** E_2 **if** $\neg(X_1 \leq R_1)$

% Reset:
constraint $(X_1, X_2) = (x_1, x_2)$ **after** $(X_1, X_2) = (x_1, x_2) \wedge E_1 = \text{TRUE}$
constraint $(X_1, X_2) = (x_1, x_2)$ **after** $(X_1, X_2) = (x_1, x_2) \wedge E_2 = \text{TRUE}$

% Mode:
nonexecutable E_1 **if** $\neg(Mode = Q_1)$ **nonexecutable** E_2 **if** $\neg(Mode = Q_2)$
E_1 **causes** $Mode = Q_2$ E_2 **causes** $Mode = Q_1$
inertial $Mode = q \quad (q \in \{Q_1, Q_2\}$

% Duration:
E_1 **causes** $Dur = 0$ E_2 **causes** $Dur = 0$

% Wait:
default $Wait = \text{TRUE}$
E_1 **causes** $Wait = \text{FALSE}$ E_1 **causes** $Wait = \text{FALSE}$

% Flow:
constraint $((X_1 – x_1)/t, (X_2 – x_2)/t) = (\mathrm{W}_1 - \mathrm{V}, -\mathrm{V})$
after $(X_1, X_2) = (x_1, x_2) \wedge Mode = Q_1 \wedge Dur = t \wedge t > 0 \wedge Wait = \text{TRUE}$
constraint $((X_1 – x_1)/t, (X_2 – x_2)/t) = (-\mathrm{V}, \mathrm{W}_2 - V)$
after $(X_1, X_2) = (x_1, x_2) \wedge Mode = Q_2 \wedge Dur = t \wedge t > 0 \wedge Wait = \text{TRUE}$
constraint $(X_1, X_2) = (x_1, x_2)$ **after** $(X_1, X_2) = (x_1, x_2) \wedge Mode = q \wedge Dur = 0 \wedge Wait = \text{TRUE} \quad (q \in \{Q_1, Q_2\})$

% Invariant
constraint $Mode = Q_1 \rightarrow X_2 \geq \mathrm{R}_2$
constraint $Mode = Q_2 \rightarrow X_1 \geq \mathrm{R}_1$

Fig. 1. $\mathscr{C}$+ representation of hybrid automaton of water tank.

Lemma 1
For any path

$$p = (v_0, r_0) \xrightarrow{\sigma_0} (v_1, r_1) \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_{m-1}} (v_m, r_m)$$

in the labeled transition system of H, let

$$p' = \langle s_0, a_0, s_1, a_1, \ldots, a_{m-1}, s_m \rangle,$$

where each s_i is an interpretation of fluent constants and each a_i is an interpretation of action constants such that, for $i = 0, \ldots m-1$,

- $s_0 \models_{bg} (Mode, X) = (v_0, r_0)$;
- $s_{i+1} \models_{bg} (Mode, X) = (v_{i+1}, r_{i+1})$;
- if $\sigma_i = \mathsf{hevent}(v_i, v_{i+1})$, then $(Dur)^{a_i} = 0$, $(Wait)^{a_i} = \text{FALSE}$, and, for all $e \in E$, $(\mathsf{hevent}(e))^{a_i} = \text{TRUE}$ iff $e = (v_i, v_{i+1})$;

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

- if $\sigma_i \in \mathscr{R}_{\geqslant 0}$, then $(Dur)^{a_i} = \sigma_i$, $(Wait)^{a_i} = \text{TRUE}$, and, for all $e \in E$, we have $(\mathsf{hevent}(e))^{a_i} = \text{FALSE}$.

Then, p' is a path in the transition system D_H.

Next, we show that every path in the transition system of D_H is a path in the labeled transition system of H.

Lemma 2
For any path

$$q = \langle s_0, a_0, s_1, a_1, \dots, a_{m-1}, s_m \rangle$$

in the transition system of D_H, let

$$q' = (v_0, r_0) \xrightarrow{\sigma_0} (v_1, r_1) \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_{m-1}} (v_m, r_m),$$

where

- $v_i \in V$ and $r_i \in \mathscr{R}^n$ $(i = 0, \dots, m)$ are such that $s_i \models_{bg} (Mode, X) = (v_i, r_i)$;
- σ_i $(i = 0, \dots, m-1)$ is
  - — $\mathsf{hevent}(v_i, v_{i+1})$ if $(\mathsf{hevent}(v_i, v_{i+1}))^{a_i} = \text{TRUE}$;
  - — $(Dur)^{a_i}$ otherwise.

Then, q' is a path in the transition system of T_H.

3.2 Representing non-linear hybrid automata using witness function

Note that formula (3.3) is not necessarily true in general even when $\mathsf{Inv}_v(X)$ is a Boolean combination of linear (in)equalities (e.g., a disjunction over them may yield a non-convex invariant).

Let us assume $\mathsf{Flow}_v(X, \dot{X})$ is the conjunction of formulas of the form $\dot{X}_i = g_i(X)$ for each X_i, where $g_i(X)$ is a Lipschitz continuous function whose variables are from X only[4]. In this case, it is known that the witness function f exists and is unique. This is a common assumption imposed on hybrid automata.

Even when the flow condition is non-linear, as long as we already know the unique witness function satisfies (3.3), the invariant checking can still be done at each state only. In this case, the representation in the previous section works with a minor modification. We modify the **Flow** representation as

- **Flow**: For each $v \in V$ and $X_i \in X$,

  $$\textbf{constraint}\ X_i = f_i(\delta)\ \textbf{after}\ X = x \wedge Mode = v \wedge Dur = \delta \wedge Wait = \text{TRUE}$$

  where $f_i : [0, \delta] \rightarrow \mathscr{R}^n$ is the witness function for X_i such that (i) $f_i(0) = x_i$ and (ii) for all reals $\epsilon \in [0, \delta]$, $\mathsf{Flow}_v(f(\epsilon), \dot{f}(\epsilon))$ is true, where $f = (f_1, \dots, f_n)$.

[4] A function $f : \mathscr{R}^n \rightarrow \mathscr{R}^n$ is called *Lipschitz continuous* if there exists $\lambda > 0$ such that for all $x, x' \in \mathscr{R}^n$,

$$|f(x) - f(x')| \leqslant \lambda |x - x'|.$$

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

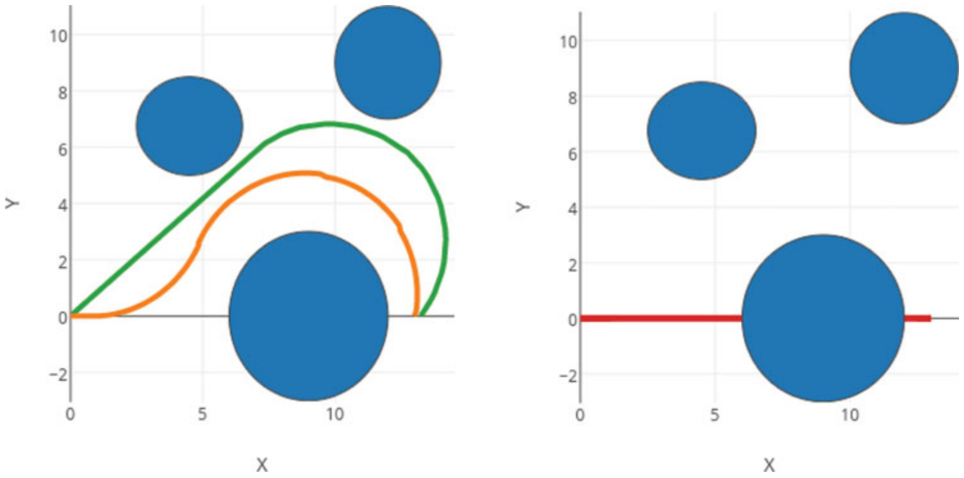
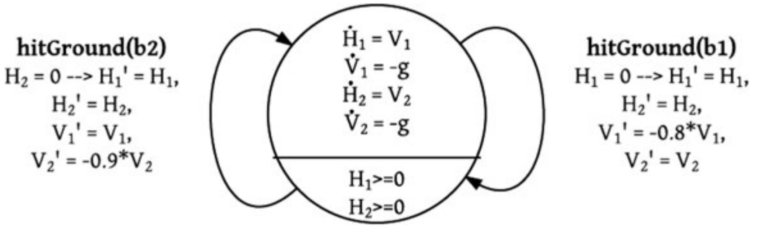


Fig. 2. (a) Feasible plan, (b) infeasible plan.

Example 2

Consider a hybrid automaton for the two bouncing balls with different elasticity.



The **Flow** condition for Ball $b1$ is represented as

$$\textbf{constraint } V_1 = v + (-g) \cdot \delta \textbf{ after } V_1 = v \wedge \mathit{Dur} = \delta \wedge \mathit{Wait} = \text{TRUE}$$
$$\textbf{constraint } H_1 = h + v \cdot \delta - (0.5) \cdot g \cdot \delta \cdot \delta \textbf{ after } H_1 = h \wedge \mathit{Dur} = \delta \wedge \mathit{Wait} = \text{TRUE}.$$

The invariant $(H_1 \geqslant 0, H_2 \geqslant 0)$ is trivial and satisfies equation (3.3). So, it is sufficient to check the invariant using (3.2) at each state only.

However, this method does not ensure that a (non-convex) invariant holds during continuous transitions. For example, consider the problem of a car navigating through the pillars as in Figure 2, where the circles represent pillars that the car has to avoid collision with. Checking the invariants at each discrete time point is not sufficient; it could generate an infeasible plan, such as (b), where the initial position $(0,0)$ and the next position $(13,0)$ satisfy the invariant $(x-9)^2+y^2>9$, but some positions between them, such as $(8,0)$, do not. This is related to the challenge in integrating high-level task planning and low-level motion planning, where plans generated by task planners may often fail in motion planners.

The next section introduces new constructs in $\mathscr{C}+$ to address this issue.

4 New abbreviations of causal laws for expressing continuous evolutions via ODEs

In this section, we introduce two new abbreviations of causal laws to express the continuous evolutions governed by ODEs.

We assume the set σ^{fl} of fluent constants contains a set σ^{diff} of real valued fluent constants $X = (X_1, \ldots, X_n)$ called *differentiable* fluent constants, and an inertial fluent constant *Mode*, which ranges over a finite set of control modes. Intuitively, the values of differentiable fluent constants are governed by some ODEs controlled by each value of *Mode*. We also assume that *Dur* is an exogenous action constant of sort $\mathcal{R}_{\geqslant 0}$.

Below are the two new abbreviations related to ODEs. First, a *rate declaration* is an expression of the form

$$\textbf{derivative of } X_i \textbf{ is } F_i(X) \textbf{ if } Mode = v \tag{4.1}$$

where X_i is a differentiable fluent constant, v is a control mode, and $F_i(X)$ is a fluent formula over $\sigma^{bg} \cup \sigma^{diff}$. We assume that an action description has a unique rate declaration (4.1) for each pair of X_i and v. So, by $d/dt[X_i](v)$ we denote the formula $F_i(X)$ in (4.1). The set of all rate declarations (4.1) for each value v of *Mode* introduces the following causal law:

$$\begin{aligned}\textbf{constraint } (X_1, \ldots, X_n) = (x_1 + y_1, \ldots, x_n + y_n) \textbf{ after } (X_1, \ldots, X_n) = (x_1, \ldots, x_n)\\ \wedge\ (y_1, \ldots, y_n) = \int_0^\delta (d/dt[X_1](v), \ldots, d/dt[X_n](v))dt\\ \wedge\ Mode = v\ \wedge\ Dur = \delta\ \wedge\ Wait = \text{TRUE}\end{aligned} \tag{4.2}$$

where $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are real variables.

Second, an *invariant law* is an expression of the form

$$\textbf{always\_t } F(X) \textbf{ if } Mode = v \tag{4.3}$$

where $F(X)$ is a fluent formula of signature $\sigma^{diff} \cup \sigma^{bg}$.

We expand each invariant law (4.3) into

$$\begin{aligned}&\textbf{constraint } \forall t \forall x\big((0 \leqslant t \leqslant \delta) \wedge\\ &\quad \big(x = \big((x_1, \ldots, x_n) + \int_0^t (d/dt[X_1](v), \ldots, d/dt[X_n](v))dt\big) \rightarrow F(x)\big)\big)\\ &\quad \textbf{after } (X_1, \ldots, X_n) = (x_1, \ldots, x_n) \wedge Mode = v \wedge Dur = \delta \wedge Wait = \text{TRUE}.\end{aligned} \tag{4.4}$$

Notice that the causal law uses the universal quantification to express that all values of X during the continuous transition satisfy the formula $F(X)$.

5 Encoding hybrid transition systems in $\mathcal{C}$+ modulo ODE

5.1 Representation

In this section, we represent the general class of hybrid automata, allowing non-linear hybrid automata with non-convex invariants, in the language of $\mathcal{C}$+ modulo ODE using the new abbreviations introduced in the previous section. As before, we assume derivatives are Lipschitz continuous in order to ensure that the solutions to the ODEs are unique.

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

The translation consists of the same causal laws as those in Section 3, except for those that account for continuous transitions. Each variable in hybrid automata is identified with a differentiable fluent constant. The representations of the flow and the invariant condition are modified as follows.

- **Flow**: We assume that flow conditions are written as a set of $\dot{X}_i = F_i(X)$ for each X_i in σ^{diff} where $F_i(X)$ is a formula whose free variables are from X only, and assume there is only one such formula for each X_i in each mode. For each $v \in V$ and each $X_i \in X$, D_H includes a rate declaration

  $$\textbf{derivative of } X_i \textbf{ is } F_i(X) \textbf{ if } Mode = v$$

  which describes the flow of each differentiable fluent constant X_i for the value of *Mode*.
- **Invariant**: For each $v \in V$, D_H includes an invariant law

  $$\begin{array}{l}\textbf{constraint } Mode{=}v \rightarrow \mathsf{Inv}_v(X)\\ \textbf{always\_t } \mathsf{Inv}_v(X) \textbf{ if } Mode = v\end{array}$$

  The new **always_t** law ensures the invariant is true even during the continuous transition.

The above representation expresses that operative ODEs and invariants are completely determined by the current value of *Mode*. In turn, one can set the value of the mode by possibly complex conditions over fluents and actions.

Theorem 1 and Lemmas 1, 2 remain true even when H is a non-linear hybrid automaton allowing non-convex invariants if we use this version of D_H instead of the previous one.

5.2 Turning in the input language of dReal

Since the new causal laws are abbreviations of basic causal laws, the translation by Lee and Meng (2013) from a $\mathcal{C}$+ description into ASPMT and a further translation into SMT apply to the extension as well. On the other hand, system `dReal` (Gao *et al.* 2013b) has a non-standard ODE extension to SMT-LIB2 standard, which succinctly represents integral and universal quantification over time variables (using `integral` and `forall_t` constructs). In its language, `t`-variables (variables ending with `_t`) have a special meaning. $c\_i\_\mathtt{t}$ is a t-variable between timepoint i and $i+1$ that progresses in accordance with ODE specified by some flow condition and is universally quantified to assert that their values during each transition satisfy the invariant condition for that transition [c.f. (4.4)].

To account for encoding the SMT formula F obtained by the translation into the input language of `dReal`, by $dr(F)$ we denote the set of formulas obtained from F by

- replacing every occurrence of $0\!:\!c$ in F with c_0 if $c \in \sigma^{diff}$;
- replacing every occurrence of $i\!:\!c$ in F with $c\_(i-1)\_\mathtt{t}$ if $c \in \sigma^{diff}$ and $i > 0$;
- replacing every occurrence of $i\!:\!c$ in F with c_i if $c \in \sigma$ and $c \notin \sigma^{diff}$

for every $i \in \{0, \ldots, m-1\}$.

The translations of the causal laws other than (4.1) and (4.3) into ASPMT and then into SMT follows the same one in the paper by Lee and Meng (2013), except that we use $dr(F)$

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

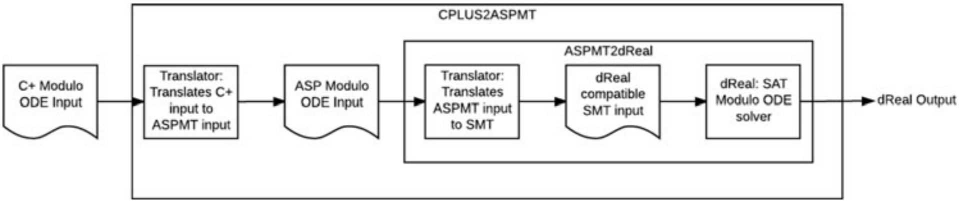

CPLUS2ASPMT
C+ Modulo ODE Input
Translator: Translates C+ input to ASPMT input
ASP Modulo ODE Input
ASPMT2dReal
Translator: Translates ASPMT input to SMT
dReal compatible SMT input
dReal: SAT Modulo ODE solver
dReal Output

Fig. 3. Architecture of system CPLUS2ASPMT.

in place of F. Below we explain how the new causal laws are encoded in the language of `dReal`.

Let θ_v be the list $(d/dt[X_1](v),\dots,d/dt[X_n](v))$ for all differentiable fluent constants $X_1,\dots,X_n$ in σ^{diff}. The set of rate declaration laws (4.1) describes a unique complete set of ODEs θ_v for each value v of *Mode* and can be expressed in the language of `dReal` as

$$\texttt{(define-ode flow\_}v\ ((\texttt{= d/dt}[X_1]\ F_1),\dots,(\texttt{= d/dt}[X_n]\ F_n))).$$

In the language of `dReal`, the integral construct

$$\texttt{(integral (0. }\delta\ \texttt{[}X_1^0\texttt{, ..., }X_n^0\texttt{] flow\_}v\texttt{))}$$

where X_1^0, ..., X_n^0 are initial values of $X_1,\dots,X_n$, represents the list of values

$$(X_1^0,\dots,X_n^0)+\int_0^\delta (d/dt[X_1](v),\dots,d/dt[X_n](v))\,dt.$$

Using the integral construct, causal law (4.2) is turned into the input language of `dReal` as

- if $i=0$,

$$\begin{array}{l}\texttt{(assert (=> (and ((= mode\_0 }v\texttt{) (= wait\_0 true)))}\\ \qquad\qquad\texttt{(= [}X_1\texttt{\_0\_t, ..., }X_n\texttt{\_0\_t]}\\ \qquad\qquad\quad\texttt{(integral (0. dur\_0 [}X_1\texttt{\_0\_0, ... , }X_n\texttt{\_0\_0] flow\_}v\texttt{))))}\end{array}$$

- if $i>1$,

$$\begin{array}{l}\texttt{(assert (=> (and ((= mode\_}i\texttt{\_}v\texttt{) (= wait\_}i\texttt{ true)))}\\ \qquad\texttt{(= [}X_1\texttt{\_}i\texttt{\_t, ..., }X_n\texttt{\_}i\texttt{\_t]}\\ \qquad\quad\texttt{(integral (0. dur\_}i\texttt{ [}X_1\texttt{\_}(i-1)\texttt{\_t, ... , }X_n\texttt{\_}(i-1)\texttt{\_t] flow\_}v\texttt{))))}\end{array}$$

The causal law (4.4), which stands for invariant law (4.3), can be succinctly represented in the language of `dReal` using `forall_t` construct as

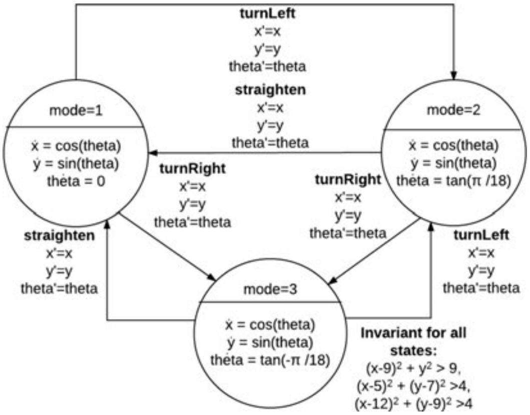
$$\texttt{(assert (forall\_t }v\texttt{ [0 dur\_}i\texttt{] }dr(i:F)\texttt{)).}$$

5.3 *Implementation and example*

We implemented a prototype system CPLUS2ASPMT, which allows us for representing hybrid transition systems in the action language $\mathcal{C}$+. The system supports an extension of $\mathcal{C}$+ by adding constructs for ODE support, then translating into an equivalent ASPMT program and finally translating it into the input language of `dReal`. The architecture of the system is shown in Figure 3. The system CPLUS2ASPMT is available at `http://reasoning.eas.asu.edu/cplus2aspmt`.

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

Example 3
Let us revisit the car example introduced earlier. The car is initially at the origin where $x = 0$ and $y = 0$ and $\theta = 0$. Additionally, there are pillars defined by the equations $(x-9)^2+y^2 \leqslant 9$, $(x-5)^2+(y-7)^2 \leqslant 4$, $(x-12)^2+(y-9)^2 \leqslant 4$. The goal is to find a plan such that the car ends up at $x = 13$ and $y = 0$ without hitting the pillars. The dynamics of the car is as described by Corke (2011).



We show some part of the hybrid automaton representation in the input language of CPLUS2ASPMT[5]. First, fluent constants and action constants are declared as follows:

```
:- constants
x            :: differentiableFluent(real[0..40]);
y            :: differentiableFluent(real[-50..50]);
theta        :: differentiableFluent(real[-50..50]);
straighten, turnLeft, turnRight   :: exogenousAction.
```

(In the ODE support mode, `mode`, `wait`, and `duration` are implicitly declared by the system.)

The derivative of the differentiable fluent constants for mode=2 (movingLeft) is declared as follows:

```
derivative of x is cos(theta) if mode=2.
derivative of y is sin(theta) if mode=2.
derivative of theta is tan(pi/18) if mode=2.
```

The invariants for avoiding the collision with the bottom pillar are represented as follows:

```
constraint x=X & y=Y ->> ((X-9)*(X-9) + Y*Y > 9).
always_t (x=X & y=Y ->> ((X-9)*(X-9) + Y*Y > 9)) if mode=V.
```

The precondition and effects of `turnLeft` action are represented as follows:

```
nonexecutable turnLeft if mode=2.
turnLeft causes mode=2.
turnLeft causes dur=0
```

Figure 4(a) illustrates the trajectory returned by the system when we instruct it to find a plan of length 5 to reach the goal position. For the path of length 3, the system returned the trajectory in Figure 4(b). The system could not find a plan of length 1 because of the **always_t** proposition asserting the invariant during the continuous transition. If we remove the proposition, the system returns the physically unrealizable plan in Figure 2(b).

[5] The complete formalization is given in Appendix C (Lee *et al.* 2017).

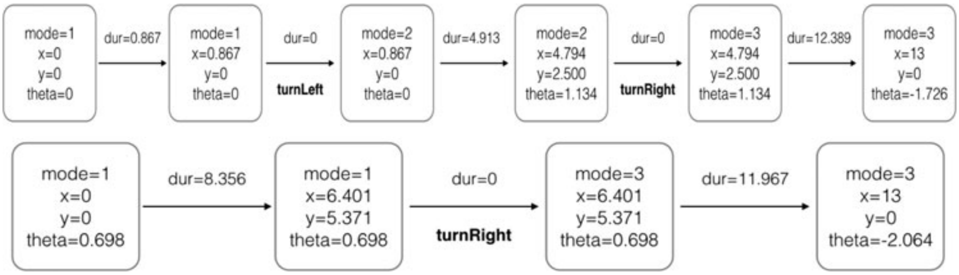


Fig. 4. Output of car example. (a) Top: maxstep = 5. (b) Bottom: maxstep = 3.

6 Related work

Due to space restriction, we list only some of the related work. PDDL+ (Fox and Long 2006) is a planning description language to model mixed discrete and continuous changes. The semantics is defined by mapping primitives of PDDL+ to hybrid automata. Most PDDL+ planners assume that the continuous change is linear, while a recent paper by Bryce *et al.* (2015), closely related to our work, presents an SMT encoding of a PDDL+ description that is able to perform reasoning about non-linear hybrid automata. However, no dedicated translator from PDDL+ to SMT is provided. The fact that both PDDL+ and $\mathcal{C}$+ can be turned into SMT may tell us how the two high-level languages are related to each other, which we leave for future work. In the paper by Bryce *et al.* (2015), the encoding was in the language of dReach with the emphasis on extending dReach with planning-specific heuristics to find a valid and possibly optimized mode path. The heuristic search has not been considered in the work of CPLUS2ASPMT, which makes the system less scalable [see Appendix D (Lee *et al.* 2017) for some experimental result].

SMT solvers have been actively used in formal verification of hybrid systems [e.g., the papers by Cimatti *et al.* (2012) and by Alur (2011)], but mostly focused on linear differential equations. dReal is an exception.

Instead of SMT solvers, constraint ASP solvers may also be used for hybrid automata reasoning. Balduccini *et al.* (2016) shows PDDL+ primitives can be encoded in the language of constraint ASP solvers, and compared its performance with other PDDL+ computing approaches including dReal. On the other hand, unlike our work, the encoding checks continuous invariants at discretized timepoints and no proof of the soundness of the translation is given. Constraint ASP solvers do not support δ-satisfiability checking. Thus, the general method of invariant checking during continuous transitions as in dReal is not yet available there.

Action language $\mathcal{H}$ (Chintabathina *et al.* 2005; Chintabathina and Watson 2012) is another action language that can model hybrid transitions, but its semantics does not describe the hybrid transition systems of the same kind as hybrid automata. Instead of using SMT solvers, an implementation of $\mathcal{H}$ is by a translation into the language $\mathcal{AC}$ (Mellarkod *et al.* 2008), which extends ASP with constraints. Language $\mathcal{H}$ does not provide support for continuous evolution via ODEs and invariant checking during the continuous transition.

ASPMT is also related to HEX programs, which are an extension of answer set programs with external computation sources. HEX programs with numerical external computation

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

have been used for hybrid reasoning in games and robotics (Calimeri *et al.* 2016; Erdem *et al.* 2016).

7 Conclusion

We represented hybrid automata in action language modulo theories. As our action language is based on ASPMT, which in turn is founded on the basis of ASP and SMT, it enjoys the development in SMT solving techniques as well as the expressivity of ASP language. We presented an action language modulo ODE, which lifts the concept of SMT modulo ODE to the action language level.

One strong assumption we imposed is that an action description has to specify *complete* ODEs. This is because existing SMT solving techniques are not yet mature enough to handle composition of partial ODEs. In the paper by Gao *et al.* (2013b), such extension is left for the future work using new commands `pintegral` and `connect`. We expect that it is possible to extend the action language to express partial ODEs in accordance with this extension.

In our representation of hybrid automata in action language $\mathcal{C}+$, we use only a fragment of the action language, which does not use other features, such as additive fluents, statically determined fluents, action attributes, defeasible causal laws. One may write a more elaboration tolerant high-level action description for hybrid domains using these features.

SMT solvers are becoming a key enabling technology in formal verification in hybrid systems. Nonetheless, modeling in the low-level language of SMT is non-trivial. We expect the high-level action languages may facilitate encoding efforts.

Acknowledgements

We are grateful to Sicun Gao and Soonho Kong for their help on running `dReal` and `dReach` systems, to Daniel Bryce for his help with benchmark problems, to Georgios Fainekos, and the anonymous referees for their useful comments.

Supplementary materials

To view supplementary material for this article, please visit https://doi.org/10.1017/S1471068417000412

References

Alur, R. 2011. Formal verification of hybrid systems. In *Proc. of the International Conference on Embedded Software (EMSOFT'11)*, IEEE, 273–278.

Alur, R., Henzinger, T. A., Lafferriere, G. and Pappas, G. J. 2000. Discrete abstractions of hybrid systems. In *Proc. of the IEEE*. 971–984.

Babb, J. and Lee, J. 2013. Cplus2ASP: Computing action language $\mathcal{C}+$ in answer set programming. In *Proc. of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. 122–134.

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

BALDUCCINI, M., MAGAZZENI, D. AND MARATEA, M. 2016. PDDL+ planning via constraint answer set programming. In *Proc. of the Working Notes of the 7th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*.

BARTHOLOMEW, M. AND LEE, J. 2013. Functional stable model semantics and answer set programming modulo theories. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, 718–724.

BRYCE, D., GAO, S., MUSLINER, D. AND GOLDMAN, R. 2015. SMT-based nonlinear PDDL+ planning. In *Proc. of the 29th AAAI Conference on Artificial Intelligence*, 3247–3253.

CALIMERI, F., FINK, M., GERMANO, S., HUMENBERGER, A., IANNI, G., REDL, C., STEPANOVA, D., TUCCI, A. AND WIMMER, A. 2016. Angry-HEX: An artificial player for angry birds based on declarative knowledge bases. *IEEE Transactions on Computational Intelligence and AI in Games 8,* 2, 128–139.

CHINTABATHINA, S., GELFOND, M. AND WATSON, R. 2005. Modeling hybrid domains using process description language[6]. In *Proc. of Workshop on Answer Set Programming: Advances in Theory and Implementation (ASP'05)*.

CHINTABATHINA, S. AND WATSON, R. 2012. A new incarnation of action language $\mathscr{H}$. In *Correct Reasoning*, E. Erdem, J. Lee, Y. Lierler, and D. Pearce, Eds. Lecture Notes in Computer Science, vol. 7265. Springer, 560–575.

CIMATTI, A., MOVER, S. AND TONETTA, S. 2012. SMT-based verification of hybrid systems. In *Proc. of AAAI*, 2100–2105.

CORKE, P. 2011. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, Vol. 73. Springer.

ERDEM, E., PATOGLU, V. AND SCHÜLLER, P. 2016. A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. *AI Communications 29,* 2, 319–349.

FERRARIS, P., LEE, J. AND LIFSCHITZ, V. 2011. Stable models and circumscription. *Artificial Intelligence 175*, 236–263.

FOX, M. AND LONG, D. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research (JAIR) 27*, 235–297.

GAO, S., KONG, S. AND CLARKE, E. 2013a. Satisfiability modulo ODEs. FMCAD, 105–112.

GAO, S., KONG, S. AND CLARKE, E. M. 2013b. dReal: An SMT solver for nonlinear theories over the reals. In *Proc. of International Conference on Automated Deduction*. Springer, Berlin, Heidelberg, 208–214.

GEBSER, M., GROTE, T. AND SCHAUB, T. 2010. Coala: A compiler from action languages to ASP. In *Proc. of European Conference on Logics in Artificial Intelligence (JELIA)*, 360–364.

GELFOND, M. AND LIFSCHITZ, V. 1998. Action languages[7]. *Electronic Transactions on Artificial Intelligence 3*, 195–210.

GIUNCHIGLIA, E., LEE, J., LIFSCHITZ, V., MCCAIN, N. AND TURNER, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence 153*, (1–2), 49–104.

HENZINGER, T. A. 1996. The theory of hybrid automata. In *Proc. of 11th Annual IEEE Symposium on Logic in Computer Science*, 278–292.

LEE, J., LONEY, N. AND MENG, Y. 2017. Online appendix for the paper "representing hybrid automata by action language modulo theories". TPLP Archive.

LEE, J. AND MENG, Y. 2013. Answer set programming modulo theories and reasoning about continuous changes. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, 990–996.

[6] `http://ceur-ws.org/vol-142/page303.pdf`

[7] `http://www.ep.liu.se/ea/cis/1998/016/`

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press

LYGEROS, J. 2004. Lecture notes on hybrid systems. University of Patras, Technical Report.

MELLARKOD, V. S., GELFOND, M. AND ZHANG, Y. 2008. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence 53*, 1–4, 251–287.

https://doi.org/10.1017/S1471068417000412 Published online by Cambridge University Press