Mobile Netw Appl (2016) 21:35–52
DOI 10.1007/s11036-015-0671-7

SMT-Based Symbolic Encoding and Formal Analysis of HML Models

Huixing Fang[1] · Huibiao Zhu[1] · Jifeng He[1]

Published online: 25 January 2016

Abstract Hybrid system is a dynamic system that involves continuous, discrete behaviors, and the interactions between continuous physical components and discrete controllers. In this paper a hybrid modeling language (called HML) for hybrid systems is extended with templates to achieve code reuse. For the formal analysis of the corresponding hybrid system models in this modeling language, these models are translated into SMT (satisfiability modulo theories) formulas as the input to an SMT solver dReal which retains the capability of bounded reachability analysis for nonlinear hybrid systems. Moreover, dReal can produce data for potential traces of hybrid systems, thus it can be employed to simulate on hybrid systems. In this paper the simulation and reachability analysis are integrated in a prototype tool (open source). We present a case study for an inverted pendulum with PID (Proportional-Integral-Derivative) controllers and a rod reactor system for temperature control, both are verified to demonstrate the efficiency of the prototype tool. We conclude that, this modeling language is capable of modeling and verification of hybrid systems based on simulation and bounded reachability analysis.

Keywords Hybrid systems · SMT · Simulation · Verification · Hybrid modeling language · dReal

✉ Huibiao Zhu
hbzhu@sei.ecnu.edu.cn

[1] Shanghai Key Laboratory of Trustworthy Computing, School of Computer Science and Software Engineering, East China Normal University, Shanghai, China

1 Introduction

Hybrid system consists of discrete actions (control logics) and continuous behaviours (flows). In general, hybrid systems are modeled as finite state machines with differential equations representing continuous flows, and difference equations for control logics. Automata, process algebras and state-charts are the three most popular notations for the modeling of hybrid systems.

Nowadays, various modeling languages exist: *Hybrid Automata* [1], *Hybrid CSP* [23, 42], *Simulink/Stateflow*, *HyPA* [11] and *Hybrid Programs* [38]. In this paper, the hybrid (relational) modeling language (called **HML** in this paper) which was proposed by He [24] is utilized, where the complex interaction between components in a hybrid system is synchronized by signals. Minor modifications for the syntax of the language are made and extended with primitive types and constraints for specifying the bounds of variables on hybrid systems.

Following the reusability principles in software engineering, code reuse focuses on reducing redundancy by using assets that exist in proper forms within the system development process. To make the models of hybrid systems much more readable and maintainable, a syntax structure called *template* is proposed to achieve code reuse. The template in the language takes formal parameters and a template body which may contain differential equations, traditional assignments or conditional statements. Therefore, the code reuse in the language is adequate for applications in coarse-grained (e.g., blocks of statements), and also in fine-grained (e.g., just one differential equation, or some assignments for control logic implementation).

In this paper, the model of the language is translated into SMT formulas, and an SMT solver **dReal** [17] is employed for formal analysis. The SMT solver **dReal** solves SMT problems over the reals with non-linear functions, e.g., trigonometric functions, exponential functions, logarithmic functions. It is based on OpenSMT [7] for DPLL(T) framework [36], and RealPaver [19] for interval constraint satisfaction. The result of the solver is UNSAT or δ-SAT on input formulas, where δ is a positive rational number denoting a numerical error bound. In addition, **dReal** produces certificates, i.e., the proofs of unsatisfiability or solutions when the input formulas are satisfiable. Based on the certificates, the simulation and reachability analysis can be implemented for hybrid systems. The translation and formal analysis are integrated in our open source tool, the corresponding source code is available at GitHub.[1]

Moreover, a case study is proposed for simulation and reachability analysis of an inverted pendulum with non-linear dynamics. The inverted pendulum is controlled by a PID (*Proportional-Integral-Derivative* [2]) controller. PID controllers are feedback controllers used in various industrial control systems. It contains three tuning parameters, the proportional gain K_p, integral gain K_i, and derivative gain K_d. During the simulation, these parameters of the controller are revealed incrementally. After that, the parameters are used to implement a PID controller for discrete control of the pendulum with a sampling time of 0.08s. The reachability analysis is used to check the stability of the pendulum under the controller.

In addition, a rod reactor system with linear dynamics is modeled and verified in this paper. The object of the system is to control the temperature of the reactor by two cooling rods. The differential equations for the cooling dynamics of these two rods contain identical form in equation and termination conditions except for different values of parameters, template is employed to implement code-reuse in the **HML** model for this case.

The remainder of this paper is organized as follows. In Section 2,the related work is presented. In Section 3, the syntax of the revised modeling language for hybrid systems is proposed. In Section 4, some preliminary definitions relate to **dReal** are presented. The translation of the model for **HML** into **dReal** is introduced in Section 5. In Section 6, the case study of inverted pendulum with PID controllers is proposed, the simulation and reachability analysis are elaborated in detail. Section 7 is a case of rod reactor for the control of the reactor's temperature by two cooling rods, the reuse of template is presented in the **HML** model of the system, and the shutdown property is verified in our prototype tool. Finally, conclusions are given in Section 8.

[1] https://github.com/fanghuixing/HML. The main task of this tool is translating **HML** models into SMT formulas which can be checked w.r.t. the satisfiability of properties based on **dReal**. The syntax of **HML** was expressed and encoded based on Terence Parr's tool ANTLRv4. The checking result was stored in a JSON file which consists of the states of the corresponding hybrid system. The JSON file was analyzed and filtered, then sent to JfreeChart for graphical demonstration of system states and behaviors

2 Related work

Phase transition systems, abbreviated as PTS [33], extend discrete transition systems with continuous actives for the modeling of hybrid systems, and the specification for hybrid behavior is based on Statecharts [22]. In PTS, the behavior of a hybrid system can be considered as a sequence of continuous and discrete phases that are interacted. The continuous phase involves continuous evolution of variables, while the discrete phase contains some (finite) discrete transitions.

In 1992, Rajeev Alur et al. introduced hybrid automata [1, 25] for modeling and specifying of hybrid systems. The authors also discussed the reachability problem of hybrid automata that is undecidable in general.

Nevertheless, the authors of [1] invented two semidecision procedures for checking piecewise-linear hybrid automata with respect to safety properties. The details for the decidability on hybrid automata can be found in [27]. The first formal verification tool for linear hybrid automata is HyTech [26].

Hybrid I/O automata [31, 32] are hybrid automata except for the division of external actions as input and output actions, and the variables in hybrid I/O automata are also partitioned into input and output variables. PHAVer [12, 13] is a tool proposed by Goran Frehse et al., which is capable of formal verification of hybrid I/O automata with affine continuous dynamics (e.g., $\dot{v} = P \cdot v + q$, where elements of P and q are intervals) based on over-approximation and partition for reachable states.

SpaceEx [14] is a verification tool also presented by Goran Frehse et al. for scalable reachability analysis of linear hybrid systems (about 100 variables) with piecewise affine, non-deterministic dynamics. SpaceEx employs polyhedra and support function [20] for over-approximation of reachable states.

For non-linear hybrid systems, Flow* [9] adopts Taylor model [5, 6] arithmetics for approximations of non-linear dynamics. As discussed in [10], Flow* is more suitable than dReach [30] on hard non-linear dynamics, while dReach is better for moderate dynamics.

For the modeling of hybrid systems, there are notations from process algebra, for instance, HyPA (*hybrid process algebra*, [11]) is an extension of ACP (*algebra of communicating processes*, [3]), with the disrupt operator, continuous

flow-clauses, and re-initializations for discrete transitions. Hybrid CSP [23, 42] extends CSP (*communicating sequential processes*, [28]) with continuous variables, differential equations, and events etc.

In addition, Modelica [15] and Matlab Simulink/Stateflow [34, 35] are modeling languages that can be applied for the simulation of hybrid systems. But the specification in Modelica are ambiguous, for example, whether an event is synchronous or not, is disagreed among various interpretations.

The Simulink/Stateflow modeling language is popular in industry for large-scale and complex systems, but lacks (built-in) formal verification support in the Matlab toolset, let alone the verification of hybrid systems under model perturbations. Moreover, the statechart also cannot be reused as a template in Simulink/Stateflow for hybrid systems.

In this paper, the prototype tool is based on **dReal**, it can cope with model perturbations for the verification of hybrid systems. Moreover, in our prototype tool, the reachability states of hybrid systems can also be viewed in a graphical way as in the toolset of Matlab Simulink.

3 Modeling language for hybrid systems

In this section, the syntax of the hybrid modeling language (**HML**) is presented, the semantics of the language is explained in an informal way, and the usage of **HML** is illustrated by a simple example. The details for the hybrid modeling language **HML** are presented as follows:

$$
\begin{aligned}
AP &::= \texttt{skip} \mid v = e \mid !s \mid \texttt{wait}(e)\\
EQ &::= R(v,\dot{v}) \mid R(v,\dot{v})\ \texttt{init}\ v_0 \mid EQ \mid\mid EQ\\
P &::= AP \mid P;\, P \mid \{P \mid\mid P\} \mid (P \langle b \rangle P)\\
&\quad \mid EQ\ \texttt{until}\ g \mid \texttt{when}\{\, G\,\}\\
&\quad \mid \texttt{while}\ (b)\{\, P\,\} \mid Id(es)\\
g &::= \epsilon \mid @(s) \mid b \mid g\ \langle\texttt{and}\rangle\ g \mid g\ \langle\texttt{or}\rangle\ g\\
b &::= \texttt{true} \mid \texttt{false} \mid v \circ c \mid \sim b\\
&\quad \mid b\ \texttt{and}\ b \mid b\ \texttt{or}\ b\\
G &::= (g\ \texttt{then}\ P) \mid G, G
\end{aligned}
$$

where, $\circ \in \{>=, >, ==, <, <=\}$, unary logical operator '$\sim$' is used as negation ($\neg$), 'b `and` b' represents conjunction ($\wedge$) of boolean expressions, and 'b `or` b' is disjunction ($\vee$) of boolean conditions. The syntax is explained as follows:

- `skip` is the empty command that terminates immediately;
- The atomic process '$!s$' emits signal s, then terminates immediately;
- The assignment statement '$v = e$' has the traditional form for variable v and expression e, for example ($v = v + 1$) is an assignment that increases v by 1;
- The process '`wait`(e)' suspends the current running process for e time units, e is an expression representing a nonnegative rational number;
- The relation predicate $R(v, \dot{v})$ defines a flow constraint, i.e., the differential equation for variable v and the first-order derivative of v over time. For example, $\dot{v} = 1.2$ and $\dot{v} = \sin(v) + \sqrt{|\ v\ |} + 1$ are two different relation predicates for variables v and $\dot{v}$; In **HML**, (`dot` v) is used to denote the first order derivative of v over time, i.e., $\dot{v}$;
- '$R(v, \dot{v})$ `init` v_0' declares a flow constraint with v_0 the initial value for v;
- The parallel equation '$EQ_1 \mid\mid EQ_2$' specifies the simultaneous differential equations;
- The process '$P_1; P_2$' denotes the sequential composition of two processes;
- The parallel composition of processes is represented by '$\{P_1 \mid\mid P_2\}$';
- The conditional choice between processes P and Q with boolean expression b is '$(P \langle b \rangle Q)$'. If b is true, it behaves like P, and Q when b is evaluated to false;
- A continuous flow consists of differential equations with exit-condition that denotes the termination condition is declared by 'EQ `until` g'. The exit-condition g can be conditional guards, signals or their combinations;
- The when-choice process '`when`{G}' activates P_i (in G) when the corresponding guard g_i is valid. If all guards are invalid, the when-choice process will wait until a valid one exists. For instance, the when-choice process '`when`{(g_1 `then` P_1), (g_2 `then` P_2)}' will wait for one of the guards g_1 and g_2 to be valid;
- The while-loop '`while` (b){P}' has the traditional form, where b is a boolean condition expression, P is the loop body;
- $Id(es)$ is an instantiation of a template, where es denotes the values passing into the corresponding template. A template in this language is used for encapsulation of statements (differential equations or control logics) that can be reused during the modeling of hybrid systems;
- For the guard g, the empty guard ϵ is valid immediately when it is evaluated. The signal guard '$@(s)$' is valid when signal s is present. The boolean guard b denotes a boolean expression. g_1 $\langle$`and`$\rangle$ g_2 is valid iff both g_1 and g_2 are valid, and g $\langle$`or`$\rangle$ g when one of the guard is valid;
- (g `then` P) is a guarded choice that is related to the when-choice process. If g is valid then P is the subsequent behavior, otherwise the process is waiting; if

'G_1, G_2' is related to the when-choice process, the process will wait until at least one guard is valid.

For different types of guards, the difference between signal and boolean expression is: signal is instantaneous while boolean expression can be valid for some durations.

For example, let v be a variable representing the velocity of a car, boolean expression $v \geq 0$ is valid after the car starts to run, and the boolean expression would be invalid until the car runs in the opposite direction with respect to the original one. However, in **HML**, the waiting process for the guard should terminate whenever the guard is valid.

In addition, four types for variables in the hybrid modeling language are specified:

$$Type ::= \texttt{Signal} \mid \texttt{boolean} \mid \texttt{int} \mid \texttt{float}$$

where, keyword `Signal` is used to declare signals, `boolean` for boolean variables, `int` for variables that take the values of integer numbers, and `float` for real numbers. Moreover, some extra syntax structures are given below, i.e., the bounded constraints, template and main function:

$$Constraint ::= v \ \texttt{in} \ [l, h] \tag{1}$$

$$Template ::= \texttt{Template} \ Id(f_s)\{ P \} \tag{2}$$

$$Main ::= \texttt{Main} \{ P \} \tag{3}$$

where, syntax (1) is used for specifying constraint conditions of the variables in an **HML** model. Symbol v stands for a variable (with type `int` or `float`), l for the minimum value and h for the maximum value that v can take, i.e., $l \leq v \leq h$. The second syntax (2) is about the template declaration. The identifier Id is the name of the template, and f_s stands for the list of formal parameters, P is the template body.

The template structure is employed to declare one block of statements that can be reused in **HML** models with proper parameters when it was instantiated. The third one (3) is used for the main function declaration that specifies the entry point of an **HML** model.

Example 1 Let x be a variable, s be a signal, the following code represents a simple **HML** model:

```
1 Signal s;    //declares signal s
2 float x=0;   //defines variable x with initial value 0
3
4 x in [0,1]; //specifies the boundaries of x
5
6 Main {
7   {
8       {dot  x = 1  init  0  until  @(s)} || {wait(1); !s}
9   }
10 }
```

The block statement on the left of || is a differential equation, it denotes the flow condition of variable v, that is $\dot{x} = 1$ with initial value of x be 0, and the continuous flow shall terminate at the moment when the input signal s is present. The right block statement waits for 1 unit of time then emits the signal s. As a result, the value of x is 1 when the system of the model terminates.

4 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-Formulas and dReal

Some definitions are reviewed following [16, 18]. Let $\mathcal{F}$ be a set of computable real functions which are called Type 2 computable functions in [40] (see Appendix A). These functions are real functions that can be approximated numerically.

In this paper, suppose that all the functions used for modeling are Type 2 computable, such as solution functions of Lipschitz-continuous ordinary differential equations, exponentiation, trigonometric functions and polynomials.

Definition 1 (**Lipschitz-Continuous Functions**) Let $V \subseteq \mathbb{R}^n$ be compact, function $f : V \to \mathbb{R}$ is Lipschitz-continuous on V if there exists a constant $k \in \mathbb{R}^+$ such that for all $v, v' \in V$,

$$|f(v) - f(v')| \leq k\|v - v'\|.$$

where, $\|\cdot\|$ is the max norm over $\mathbb{R}^n$.

Consider the initial value problem of ordinary differential equation:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}(t, \mathbf{v}_0)), \quad \mathbf{y}(0, \mathbf{v}_0) = \mathbf{v}_0 \in V,$$

where, $\mathbf{f} = \langle f_1, ..., f_n \rangle$, $V \subseteq \mathbb{R}^n$ is compact. The sufficient condition for the equation to have a unique solution $\mathbf{y}$ is that f_i is Lipschitz-continuous for $1 \leq i \leq n$. The ordinary differential equation satisfying the sufficient condition is called Lipschitz-continuous. The solution functions $\mathbf{y}_i : \mathbb{R} \times V \to \mathbb{R}$ are computable [29] over $\mathbb{R} \times V$ if the ODE is Lipschitz-continuous.

In the SMT solver **dReal**, hybrid systems are represented with $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas. It is defined as follows.

Definition 2 ($\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-**Formulas** [16]) Let $\mathcal{F}$ be a set of Type 2 computable real functions (constants are 0-ary functions). It is defined as follows:

$$t ::= v \mid f(t(\mathbf{v})), where \ f \in \mathcal{F};$$

$$\phi ::= t(\mathbf{v}) > 0 \mid t(\mathbf{v}) \geq 0 \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists x_i.\phi \mid \forall x_i.\phi,$$

where, v represents variable or constant, $\mathbf{v} = (v_1, v_2, ..., v_n)$, t denotes $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ terms. The negation $\neg\phi$ can be implemented by the equivalence relation:

$$\neg(t > 0) \equiv -t \geq 0, \qquad \neg\exists x_i.\phi \equiv \forall x_i.\neg\phi,$$
$$\neg(t \geq 0) \equiv -t > 0, \qquad \neg\forall x_i.\phi \equiv \exists x_i.\neg\phi.$$

And, the implication $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$.

The atomic formula $(t(\mathbf{v}) = 0)$ can be replaced by the formula $-|t(\mathbf{v})| \geq 0$. Also, for atomic formula $t(\mathbf{v}) < 0 \equiv -t(\mathbf{v}) > 0$, and $t(\mathbf{v}) \leq 0 \equiv -t(\mathbf{v}) \geq 0$. Moreover, $s < t$ can be written as $(t - s) > 0$, it is similar for the relation $\leq$.

As this paper focus on formulas with bounded variables, the bounded quantifiers definition is presented as follow:

Definition 3 (**Bounded Quantifiers** [16]) The bounded existential quantifier $\exists^I$ and bounded universal quantifier $\forall^I$ are defined:

$$\exists^I v.\phi =_{def} \exists v.(v \in I \wedge \phi);$$
$$\forall^I v.\phi =_{def} \forall v.(v \in I \rightarrow \phi),$$

where, the interval $I = [l, h]$, l and h denote $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ terms, which only contain free variables in ϕ excluding v.

Moreover, a bounded $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-sentence is of the form:

$$Q_1^{I_1} v_1 \cdots Q_n^{I_n} v_n.\phi(v_1, ..., v_n),$$

in which $Q_i^{I_i}$ represents bounded quantifier, and $\phi(v_1, ..., v_n)$ denotes quantifier-free $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula.

The SMT problems that **dReal** had solved are deciding the truth value of fully existentially quantified sentences (Σ_1-SMT problem) and the formulas that are partially universally quantified (Σ_2-SMT problem), i.e., a Σ_1-SMT problem is a formula as $\exists^{I_1} v_1 \cdots \exists^{I_n} v_n.\phi(v_1, ..., v_n)$, while a Σ_2-SMT problem is $\exists^{I_1} v_1 \cdots \exists^{I_n} v_n \forall^{I_{n+1}} v_{n+1} \cdots \forall^{I_m} v_m.\phi(v_1, ..., v_m)$.

SMT solver **dReal** can deal with first-order formulas (bounded quantifiers) with computable non-linear real functions in the framework of δ-complete decision procedures [16]. In this framework the δ-$variant$ of first-order formulas is defined as follows.

Definition 4 (δ-**Variant**) Let $\mathbb{Q}^+$ be the set of positive rational numbers, $\delta \in \mathbb{Q}^+ \cup \{0\}$, and $\varphi =_{def} Q_1^{I_1} v_1 \cdots Q_m^{I_m} v_m.\phi[t_i(\mathbf{v}) > 0; t_j(\mathbf{v}) \geq 0]$ for $i \in \{1, ..., k\}$ and $j \in \{k + 1, ..., m\}$. The δ-weakening formula φ^δ is defined as:

$$Q_1^{I_1} v_1 \cdots Q_m^{I_m} v_m.\phi[t_i(\mathbf{v}) > -\delta; t_j(\mathbf{v}) \geq -\delta].$$

φ^δ is obtained from φ with the replacing of atomic formula $(t_i > 0)$ by $(t_i > -\delta)$, and $(t_i \geq 0)$ by $(t_i \geq -\delta)$.

For any φ **dReal** returns one of the following answers [17]:

- UNSAT : formula φ is unsatisfiable;
- δ-SAT : formula φ^δ is satisfiable.

Based on the result, the conclusion of whether a property with hybrid system model in SMT-formulas is satisfiable (or reachable for the corresponding states) with numerical perturbation δ or the proof of unsatisfiability for the property without perturbation can be made.

5 Translation into dReal

In this section, the translation on **HML** models is proposed, which is from **HML** model to SMT [4] formulas as the input for **dReal**. The first part focus on the translation (unrolling) of sequential models, after that the parallel model unrolling is presented.

5.1 Sequential model unrolling

Each variable in the formulas after unrolling is accompanied by the unrolling depth and initial/final flags. For a maximum unrolling depth of n, the flags are $0 \leq depth \leq n$, $flag \in \{\rho, \tau\}$, where ρ is used to denote the initial value and τ for the final (or intermediate) value of the corresponding variable for discrete (or continuous) behaviors. For instance, $v_{0,\rho}$ represents the initial value of v at the depth of 0. For sequential model unrolling, five types of discrete action statements are considered as follows.

Assignment For assignment statement $(\mathbf{x} = \mathbf{e}(\mathbf{x}))$, in general, the SMT formulas are in prefix notation, but for simplicity, in this paper the formula is presented in infix notation. Suppose that the current depth is m, $0 \leq m \leq n$, we can encode the assignment as formula:

$$\mathbf{x}_{m,\tau} = \mathbf{e}(\mathbf{x}_{m,\rho}),$$

where, $\mathbf{e}(\mathbf{x}_{m,\rho})$ is the expression of $\mathbf{e}(\mathbf{x})$ with the replacing of $\mathbf{x}$ by initial values $\mathbf{x}_{m,\rho}$. For example, the assignment $x = 1 + x$ for variable x is represented as $x_{m,\tau} = 1 + x_{m,\rho}$. For assignment $x = x + y + 1$, it can be translated into $x_{m,\tau} = x_{m,\rho} + y_{m,\rho} + 1$, and in the form of SMT formula: $(= x_{m,\tau} (+ x_{m,\rho} (+ y_{m,\rho} 1)))$.

Signal Emitting The emitting of signal $(!s)$ is implemented as a discrete assignment. The fact of signal emitting is the recording of time instant when the signal is present. In the unrolling, the signals are regarded as variables with the type of `float`. The global time of an **HML** model is represented by variable $global$. Thus, the formula is

$$s_m = global_{m,\rho},$$

where, $global_{m,\rho}$ represents the time point when signal s is emitted. The time instant is recored by the variable s_m (without the suffix τ) as signals do not emitted during a continuous flow.

Sequential Composition For sequential composition of two discrete actions $(P_1; P_2)$, suppose the formula for P_1 has the form $\mathcal{F}_1(\mathbf{x}^1_{m,\tau}, \mathbf{x}^1_{m,\rho}) \wedge (\mathbf{x}^1_{m,\tau} = \mathbf{e}_1(\mathbf{x}^1_{m,\rho}))$ and the formula for P_2 is $\mathcal{F}_2(\mathbf{x}^2_{m,\tau}, \mathbf{x}^2_{m,\rho}) \wedge (\mathbf{x}^2_{m,\tau} = \mathbf{e}_2(\mathbf{x}^2_{m,\rho}))$, then the formula is

$$\mathcal{F}_1(\mathbf{e}_1(\mathbf{x}_{m,\rho}), \mathbf{x}_7 7) \wedge \mathcal{F}_2(\mathbf{x}_{m,\tau}, \mathbf{e}_1(\mathbf{x}_{m,\rho})) \wedge (\mathbf{x}_{m,\tau} = \mathbf{e}_2(\mathbf{e}_1(\mathbf{x}_{m,\rho}))),$$

where, $\mathbf{x}_{m,\rho}$ denotes the initial value for $\mathbf{x}$ of $(P_1; P_2)$. The superscripts (1 and 2) are just used to distinguish the initial/final values of P_1 and P_2. Thus, $\mathbf{x}^1_{m,\rho} = \mathbf{x}_{m,\rho}$, $\mathbf{x}^2_{m,\rho} = \mathbf{x}^1_{m,\tau}$, and $\mathbf{x}^2_{m,\tau} = \mathbf{x}_{m,\tau}$. In the following, the formula $\mathbf{x}_{m,\tau} = \mathbf{e}_2(\mathbf{e}_1(\mathbf{x}_{m,\rho}))$ is used to denote the result of $[\mathbf{x}^1_{m,\tau} = \mathbf{e}_1(\mathbf{x}^1_{m,\rho}); \mathbf{x}^2_{m,\tau} = \mathbf{e}_2(\mathbf{x}^2_{m,\rho})]$ for sequential operator $(;)$.

Conditional Choice The choice between two discrete actions with condition $(P_1 \langle b \rangle P_2)$,

$$[b(\mathbf{x}_{m,\rho}) \wedge \mathbf{x}_{m,\tau} = \mathbf{e}_1(\mathbf{x}_{m,\rho})] \vee [\neg b(\mathbf{x}_{m,\rho}) \wedge \mathbf{x}_{m,\tau} = \mathbf{e}_2(\mathbf{x}_{m,\rho})],$$

where, $b(\mathbf{x}_{\mathbf{m},\rho})$ represents the boolean value of condition b. And, $\mathbf{e}_1$ and $\mathbf{e}_2$ are related to P_1 and P_2, respectively.

While-Loop For a while-loop statement with discrete body (`while` $(b)\{\ P\ \}$), let i denote the number of iterations of the loop, i is a natural number. If $i = 0$, the representation of the loop is

$$LP_0 =_{def} \neg b(\mathbf{x}_{m,\rho}),$$

for $i = 1$,

$$LP_1 =_{def} [b(\mathbf{x}_{m,\rho}) \wedge \mathbf{x}_{m,\tau} = \mathbf{e}(\mathbf{x}_{m,\rho})] \wedge [\neg b(\mathbf{x}_{m,\tau})],$$

for $i = 2$,

$$LP_2 =_{def} [b(\mathbf{x}_{m,\rho}) \wedge \mathbf{x}_{m,\tau} = \mathbf{e}(\mathbf{e}(\mathbf{x}_{m,\rho}))] \wedge [\neg b(\mathbf{x}_{m,\tau})],$$

in which, $\mathbf{e}$ represents the transformation of loop body P, thus, in general, if the current depth is m, $LP_i =_{def} [b(\mathbf{x}_{m,\rho}) \wedge \mathbf{x}_{m,\tau} = \mathbf{e}^i(\mathbf{x}_{m,\rho})] \wedge [\neg b(\mathbf{x}_{m,\tau})]$. The notation $\mathbf{e}^i$ denotes the application of $\mathbf{e}$ for i times with the initial value $\mathbf{x}_{m,\rho}$. As the number of iterations can be infinite, in this paper, only the type of While-Loop program that its loop body must contain continuous behaviors is considered in this paper. Note that the maximum unrolling depth is bounded in the model translation and verification.

For continuous part, it focus on three types of statements as follows.

Waiting For the waiting statement `wait`$(e(\mathbf{x}))$, where e is an expression specifying the units of time, in the unrolling, the waiting statement is translated into a differential equation for variable $clock$ with derivative 1, that is, the value of $clock$ is the integral over time in the interval $[0, time_m]$, denoted by

$$[\langle clock_{m,\tau} \rangle] = (\texttt{integral}\ [0, time_m]\ clock_{m,\rho}\ flow),$$

where, $[\langle \cdot \rangle]$ is used to indicate the values of variables during the continuous flow. The differential equation $flow =_{def} (\frac{d[clock]}{d[t]} = 1)$, and the initial value $clock_{m,\rho} = 0$ with constraint $\forall t \in [0, time_m].\neg(clock_{m,\tau} \geq e(\mathbf{x}_{m,\rho}))$ before the termination of the waiting. This special form of formulas for differential equations with extra constraints can be processed by **dReal** to specify the values for continuous variables. In addition, to specify the termination of the flow, the formulas are appended with the constraint $clock_{m,\tau} \geq e(\mathbf{x}_{m,\rho})$ without bounded universal quantifier.

Differential Equation For a differential equation of the form $(\dot{v} = e(v)$ `init` v_0 `until` $g(\mathbf{x}))$, the formula is

$$[\langle v_{m,\tau} \rangle] = (\texttt{integral}\ [0, time_m]\ v_{m,\rho}\ flow),$$

where, $v_{m,\rho} = v_0$, and $flow =_{def} (\frac{d[v]}{d[t]} = e(v))$. Also, the corresponding constraint is $\forall t \in [0, time_m].(\neg g(\mathbf{x})_{m,\tau})$. Note that, for different types of g, the forms of constraints are different. Two main types are listed here.

- For signal guard $@(s)$, $\neg g(\mathbf{x}_{m,\tau})$ has the form $\neg(s_m \geq global_{m,\tau})$. If the signal is emitted before a continuous flow, then $(s_m \geq global_{m,\tau})$ is valid, the continuous flow will terminate immediately;
- For boolean condition b, $\neg g(\mathbf{x}_{m,\tau})$ has the form $\neg b(\mathbf{x}_{m,\tau})$.

For simultaneous differential equations, the corresponding formulas of flows and constraints can appended with conjunction operators.

When-Choice For when-choice `when`$\{(g$ `then` $P)\}$, continuous flows are employed as previous.

$$[\langle clock_{m,\tau} \rangle] = (\texttt{integral}\ [0, time_m]\ clock_{m,\rho}\ flow),$$

The differential equation $flow =_{def} (\frac{d[clock]}{d[t]} = 1)$. Again, the corresponding constraint is

$$\forall t \in [0, time_m].(\neg g(\mathbf{x})_{m,\tau}).$$

The body P is used for the next depth $(m + 1)$ unrolling.

Let $\mathcal{F}_D$ and $\mathcal{F}_C$ denote formulas for discrete and continuous actions, respectively, a renaming operation for $\mathcal{F}_D$ is need to combine the resulted formulas. The operation links

the values in current depth to the values in previous depth. In general, for variable v, the renaming is implemented by changing the variable $v_{m,\rho}$ to $v_{m-1,\tau}$, and $v_{m,\tau}$ to $v_{m,\rho}$. And, initial values are used when $m = 0$.

For different choices (with conditions) in an **HML** model, it is needed to consider different paths of the corresponding system. The amount of paths may grow exponentially as the increase of the unrolling depth. A dynamic merging algorithm is proposed to compress paths.

Algorithm 1 Dynamic merging of paths.

Require: The number of paths, $n \geq 2$, the maximum unrolling depth $d \geq 0$; The set of n paths, $P_s = \{\bigwedge_{j=0}^{m}\langle D_j, C_j\rangle_i \mid 0 \leq i \leq n, and\ m \leq d\}$.

Ensure: The set of paths after merging, P_s';

1: Picking and deleting one path p from P_s, i.e., $P_s := P_s - \{p\}$;
2: Iterating all paths in P_s, if one path $q \in P_s$ can be merged with p, remove it from P_s, i.e., $P_s := P_s - \{q\}$ and $p := \mathbf{merge}(p, q)$; put p into P_s';
3: Checking the size of P_s, if $\mathbf{size}(P_s) = 1$, then remove the path from P_s to P_s', then goto 4; otherwise if $\mathbf{size}(P_s) \geq 2$ goto 1;
4: **return** P_s'.

In Algorithm 1, the input size of paths is ≥ 2, each unrolling formula of one path at specific depth consists of the discrete part D and continuous part C, separately. The method **size** calculates the number of elements of one set. In addition, **merge** takes two paths, and returns one path for the merged result of paths. The checking of two paths (p_1 and p_2) that whether they can be merged is decided according to the following rules:

- Same length: $p_1.m \equiv p_2.m$, suppose that m is the length of one path;
- Same continuous formula: Each pair of unrolling formulas at the same depth has the same continuous formula.

Base on these rules, the merging is implemented on the discrete unrolling parts of formulas. And the merging is considered for each unrolling depth before the maximum depth is reached.

5.2 Parallel model unrolling

For **HML** models, only two types of dynamic behaviors are modeled for hybrid systems, the discrete action D and continuous flow C. The parallel composition of two process relates to these two types of dynamic behaviors. Thus, in general, three forms of parallel compositions are described as follows.

Discrete-Continuous For discrete action paralleled with continuous flow, translation rule is presented as follows:

$$\frac{D \parallel C}{D\ ;\ C}$$

In this rule, the discrete action is executed first, and then the continuous flow.

Example 2 Let discrete action $D =_{def} \{x = 1 + x\}$, and continuous flow $C =_{def} \{\dot{y} = 1\ \texttt{init}\ 0\ \texttt{until}\ (y >= 1)\}$, then $\{D \parallel C\}$ can be encoded as follows:

$$\begin{aligned} &x_{m,\rho} = 1 + x_{m-1,\tau} \wedge y_{m,\rho} = 0\ \wedge \\ &clock_{m,\rho} = 0 \wedge [\langle clock_{m,\tau}, x_{m,\tau}, y_{m,\tau}\rangle] \\ &= (\texttt{integral}\ [0, time_m]\ clock_{m,\rho}\ x_{m,\rho}\ y_{m,\rho}\ flow), \end{aligned}$$

where, m is the current depth of unrolling, thus $m - 1$ is the previous depth, $flow =_{def} (\frac{d[clock]}{d[t]} = 1 \wedge \frac{d[x]}{d[t]} = 0 \wedge \frac{d[y]}{d[t]} = 1)$. This formula indicates that the initial values of x and y for the continuous flow are assigned to $1 + x_{m-1,\tau}$ and $y_{m-1,\tau}$, respectively. Moreover, the following constraint $\forall t \in [0, time_m].\neg(y_{m,\tau} \geq 1)$ is required, which specifies the condition needs to be satisfied before the flow terminates.

Discrete-Discrete (interleaving) For the parallel composition of two discrete actions (D_1 and D_2),

$$\frac{D_1 \parallel D_2}{(D_1\ ;\ D_2)\ \vee\ (D_2\ ;\ D_1)}$$

The order of the execution of D_1 and D_2 can be nondeterministic (represented by the disjunction operator $\vee$).

Continuous-Continuous The parallel of two continuous flows (C_1 and C_2, all continuous behaviors can be translated into flows),

$$\frac{C_1 \parallel C_2}{\begin{array}{l}(\sim_{1,2})\ \texttt{until}\ g_{1,2}\ ; \\ (g_1 \wedge g_2 \wedge \texttt{skip}) \vee (g_1 \wedge \neg g_2 \wedge C_2) \vee (\neg g_1 \wedge g_2 \wedge C_1)\end{array}}$$

in which, $(\sim_{1,2})$ is the simultaneous differential equations that are retrieved separately from C_1 and C_2. The new guard $g_{1,2}$ is the merged guard from C_1 and C_2, $g_{1,2} = g_1 \vee g_2$. The formulas connected by $(\vee)$ behind semicolon (;) represent three possibilities: two guards are valid at the same time then C_1 and C_2 terminate simultaneously; g_1 is valid but g_2 is invalid, thus C_1 is terminated, C_2 has to continue; likewise, the third possibility, g_2 is valid but g_1 is invalid.

Example 3 ($C_1 \parallel C_2$) Let the first continuous flow C_1 be

$$C_1 =_{def} \{(\dot{p} = v\ \texttt{init}\ 0 \parallel \dot{v} = 0.5\ \texttt{init}\ 0)\ \texttt{until}\ (p \geq 100)\},$$

the second flow is $C_2 =_{def} \{\texttt{wait(2)}\}$. The result of parallel composition is

$$[\langle clock_{m,\tau}, p_{m,\tau}, v_{m,\tau}\rangle]$$
$$= (\texttt{integral}\ [0, time_m]\ clock_{m,\rho}\ p_{m,\rho}\ v_{m,\rho}\ flow),$$

where, $flow =_{def} (\frac{d[clock]}{d[t]} = 1 \wedge \frac{d[p]}{d[t]} = v \wedge \frac{d[v]}{d[t]} = 0.5)$. Moreover, the following constraint is needed:

$$\forall t \in [0, time_m].\neg(clock_{m,\tau} \geq 2)$$
$$\wedge\ \forall t \in [0, time_m].\neg(p_{m,\tau} \geq 100).$$

The initial values $clock_{m,\rho} = 0$, $p_{m,\rho} = 0$ and $v_{m,\rho} = 0$. Based on the parallel unrolling rule, the formulas after $(;)$ can be processed separately and combined with $(\vee)$.

In addition, there exists one special encoding of parallel **HML** model when it contains `wait` statements. For each depth of our unrolling, the special variable *clock* is set to 0 as the initial value of time for continuous behaviors. The guard of `wait` statement needs to be revised properly when the waiting process is not terminated in one depth of unrolling as discussed in the following example.

Example 4 Assume that the current depth is m, and we have $C_1 =_{def} \{wait(1)\}$, $C_2 =_{def} \{\dot{x} = 1\ \texttt{init}\ 0\ \texttt{until}\ (x >= 0.5)\}$, then we can encode the $\{C_1 \parallel C_2\}$ as follows:

$$clock_{m,\rho} = 0 \wedge x_{m,\rho} = 0 \wedge [\langle clock_{m,\tau}, x_{m,\tau}\rangle] =$$
$$(\texttt{integral}\ [0, time_m]\ clock_{m,\rho}\ x_{m,\rho}\ flow_1) \wedge$$
$$\forall t \in [0, time_m].\neg(clock_{m,\tau} \geq 1) \wedge$$
$$\forall t \in [0, time_m].\neg(x_{m,\tau} \geq 0.5)$$
$$\wedge$$
$$clock_{m+1,\rho} = 0 \wedge x_{m+1,\rho} =$$
$$x_{m,\tau} \wedge [\langle clock_{m+1,\tau}, x_{m+1,\tau}\rangle] =$$
$$(\texttt{integral}\ [0, time_{m+1}]\ clock_{m+1,\rho}\ x_{m+1,\rho}\ flow_2) \wedge$$
$$\forall t \in [0, time_{m+1}].\underline{\neg(clock_{m+1,\tau} \geq (1 - clock_{m,\tau}))},$$

where $flow_1 =_{def} (\frac{d[clock]}{d[t]} = 1 \wedge \frac{d[x]}{d[t]} = 1)$, and $flow_2 =_{def} (\frac{d[clock]}{d[t]} = 1 \wedge \frac{d[x]}{d[t]} = 0)$. The underscored formula respects the revised guard for `wait` statement. In this case, C_2 is terminated before C_1, thus the waiting process has to move forward for the remaining waiting time. This kind of revision in our prototype tool is constructed based on the dynamic checking satisfiability of guards with **dReal**.

5.3 Dynamic translation for HML models

In parallel unrolling, if there are n discrete actions from n parallel processes, their parallel composition would produce $n!$ different executing orders. It is challenging to tackle the total orders as n increases, for instance, the amount of orders is 3,628,800 when $n = 10$. In this section, a procedure is presented, it implements the dynamic translation for **HML** models based on **dReal**. This translation is the base for the simulation of **HML** models.

The Algorithm 2 presents the generic procedure of dynamic translation of **HML** models. The discrete actions are stored in the set S_d. The guards for processes P_i are checked by **dReal** during the translation, the continuous actions whose guards are unsatisfiable would be added to the set S_c.

At the end, the result is the conjunction of formulas returned from the unrolling of discrete actions in S_d and the continuous actions in S_c. Various orders can be processed for the unrolling of S_d, but for the simulation in this paper, the sequential order that the actions are added is respected.

Note that, the conditions in Conditional-Choice and While-Loop statements are also dynamic checked by **dReal** during the translation, but here the details in Algorithm 2 are omitted. Program **unroll** is based on the unrolling approach presented in Sections 5.1 and 5.2 of Section 5.

Algorithm 2 Generic Dynamic Translation for One Depth of **HML** Model

Require: The number of processes $n \geq 2$, and the set of n processes $\{P_i \mid 1 \leq i \leq n\}$
Ensure: SMT formula for one depth of unrolling, $\mathcal{F}_u$

```
1:  for i := 1 ... n do
2:      while true do
3:          //pick the current action of P_i
4:          act := P_i.getNextAct();
5:          if act is null then
6:              break;
7:          end if
8:          if act is a discrete action then
9:              //S_d is the set of discrete actions, initialized as ∅
10:             S_d := S_d ∪ {act};
11:         else
12:             if the guard of act is unsatisfiable then
13:                 //S_c is the set of continuous actions
14:                 S_c := S_c ∪ {act}; break;
15:             end if
16:         end if
17:     end while
18: end for
19: // unrolling of discrete and continuous actions
20: F_u := unroll(S_d) ∧ unroll(S_c).
```

6 Case study-1 : inverted pendulum

In this section, an inverted pendulum on a small cart is modeled in **HML**. In this case study, the pendulum is controlled by a PID controller. The formal model, simulation and reachability analysis are provided in detail.

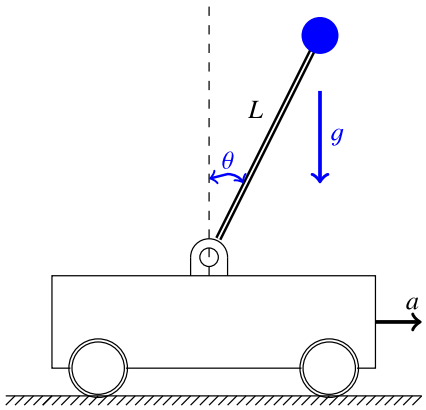


Fig. 1 Inverted pendulum

The simulation results are visualized with a data viewer of our prototype tool. The inverted pendulum is illustrated in Fig. 1, the pendulum can rotate around the pivot, the cart can move left or right, where,

- L: the length of the pendulum, 1 meter;
- θ: the angular position of the pendulum from upright position;
- a: the acceleration of the cart;
- g: the gravity acceleration, 9.8 m/s^2.

The relation of these variables in the inverted pendulum can be specified with the non-linear differential equation:

$$L\frac{d^2\theta(t)}{dt^2} = g \cdot \sin[\theta(t)] - a(t) \cdot \cos[\theta(t)],$$

where, t represents time. The control for inverted pendulum can be implemented by PID [8] or fuzzy-logic controllers [37, 41].

Figure 2 depicts a PID controller in a feedback loop. PID controllers are widely used in industrial control systems. The controller involves three correcting terms: the proportional, the integral and derivative values, denoted by P, I, and D, respectively.

The term P depends on the current control error, I on the integral of the error, and D is a prediction of future errors, based on the derivative (rate) of the error. These terms are summed to calculate the output of the PID controller. As

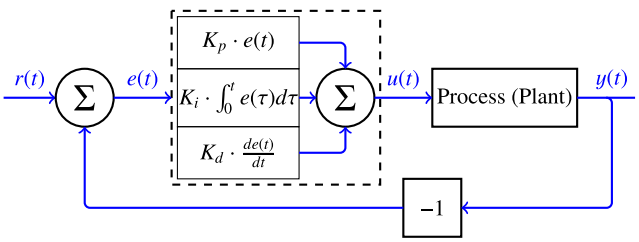


Fig. 2 Block diagram of PID controller

illustrated in Fig. 2, the control output is:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt},$$

where $e(t)$ denotes the control error. The parameters K_p, K_i and K_d are proportional, integral and derivative gains, respectively [2]. The output of the process (plant) is called process variable, denoted by $y(t)$. The desired value of the process variable is called *set point* and is denoted by $r(t)$.

In this case study of inverted pendulum, $r(t) = 0$ which means that it is required to control the pendulum to be vertical, i.e., the angular position θ be 0. As a result, the control error can be specified by $\theta(t)$.

In Fig. 1, the angular position is controlled through the acceleration of the cart, therefore $u(t) = a(t)$. The corresponding models in **HML** are presented in Section 6.1 for continuous control and Section 6.2 for discrete control, respectively.

6.1 Continuous PID control : simulation

The **HML** model with parameters K_i and K_d are equal to 0 is provided in Fig. 3. The parameter K_p for proportional term is 80. The initial position is 0.2 (rad). In this scenario, the acceleration of the cart $a(t) = K_p\theta(t)$. The constants (`g` and `kp`) are indicated by the keyword `final` at line 1.

The constraints are specified for variables in the model at lines 4∼8. The template `Pendulum` is instantiated with parameters `position` and `velocity`. Therefore, variable `theta` in template `Pendulum` is related to variable `position` for the angular position of the pendulum and `omega` is related to variable `velocity` for angular velocity. The simulation results for angular position and velocity are illustrated in Figs. 4 and 5, respectively.

As depicted in the simulation result, both position and velocity are oscillating. But, the control object for this case is the stable angular position of the pendulum, that is approaching the vertical line in the upward direction as quickly as possible.

In the following, the value of K_d is increased, which results in the angle to be 0. Figure 6 is the simulation with $K_d = 5$, where the position is damping to 0 after $2s$. As the value of K_d increasing, the damping rate is faster.

In Fig. 7, the position is decreasing to 0 after $0.6s$. Unfortunately, it is not better when increasing the value of K_d much more. Figure 8 is an example with $K_d = 30$, in which the time for the stable control is also about $2s$.

As a result, the decreasing of the time consumed for the control of the pendulum position (expected to be stable at 0), is not continuable when the value of K_d increases.

At this point, based on the simulation, the preferable parameters ($K_p = 80$, $K_i = 0$ and $K_d = 15$) are revealed that can be applied for the discrete control in Section 6.2.

Fig. 3 The **HML** model for simulation of inverted pendulum (the concrete encoded SMT formulas are presented in Appendix B). The first-order derivative of variable v over time is declared by `dot v`

```
1  final float g=9.8, kp=80;
2  float position=0.2, velocity=0, clock=0, global=0;
3
4  position in [-10, 10];
5  velocity in [-40, 40];
6  time in [0, 10];
7  clock in [0, 10];
8  global in [0, 10];
9
10 Template  Pendulum(float theta, float omega){
11   (dot  theta = omega )
12    ||
13   (dot  omega = g*sin(theta) - (kp*theta)*cos(theta) )
14    until  (false)
15 }
16
17 Main {
18       Pendulum(position, velocity)
19 }
```

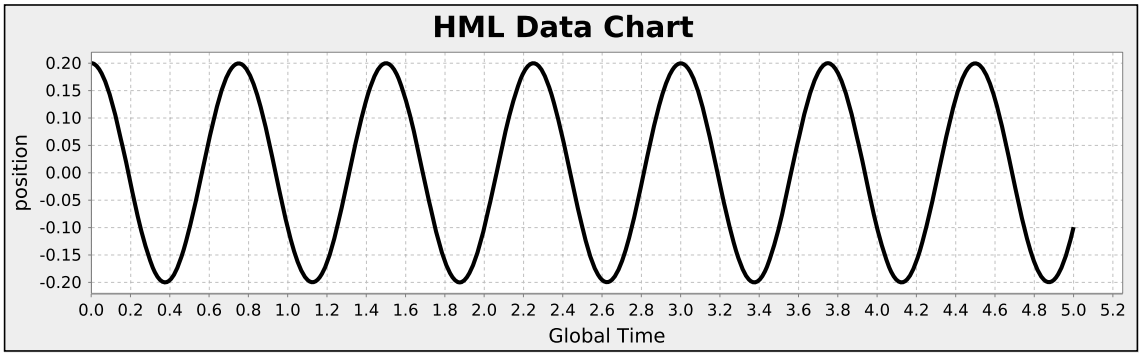


Fig. 4 Angular position with $K_p = 80$, $K_i = 0$, and $K_d = 0$. Simulation time is $5s$

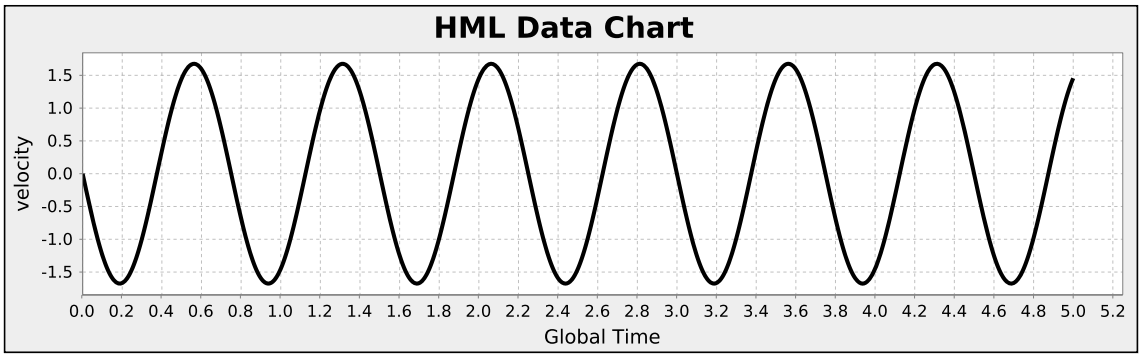


Fig. 5 Angular velocity with $K_p = 80$, $K_i = 0$, and $K_d = 0$. Simulation time is $5s$

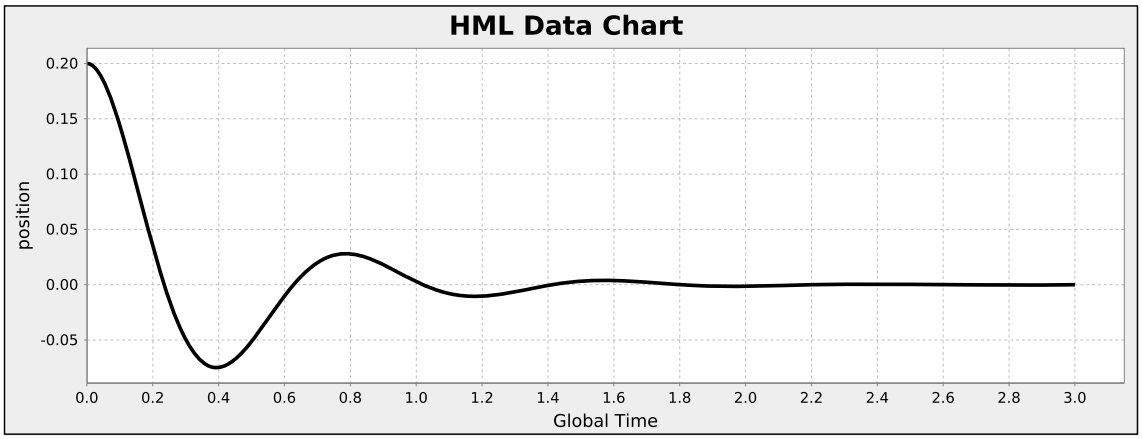


Fig. 6 Angular position with $K_p = 80$, $K_i = 0$, and $K_d = 5$. Simulation time is $3s$

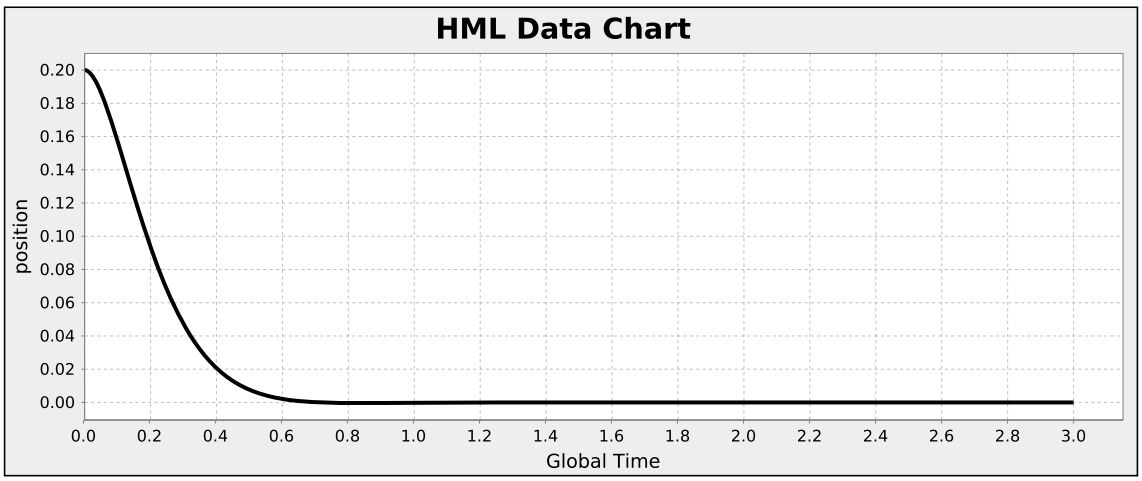


Fig. 7 Angular position with $K_p = 80$, $K_i = 0$, and $K_d = 15$. Simulation time is $3s$

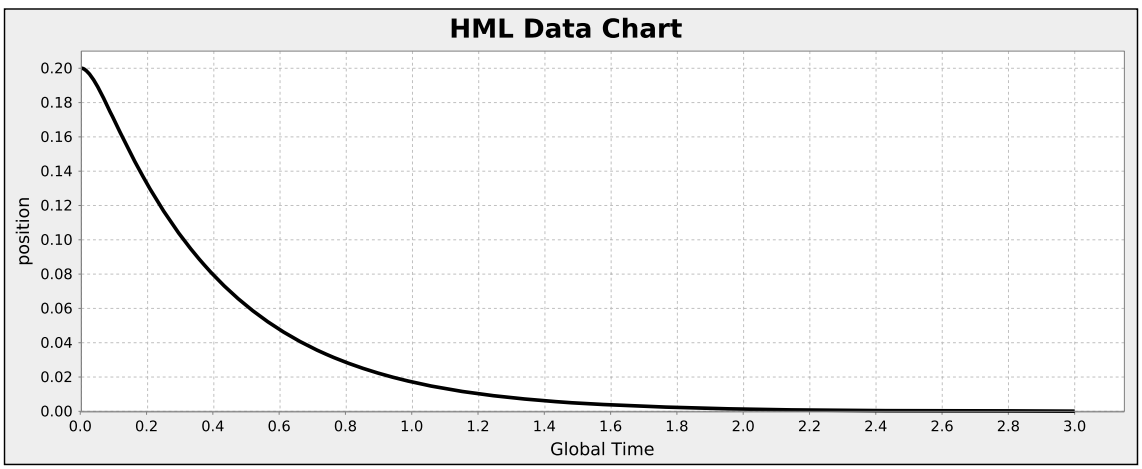


Fig. 8 Angular position with $K_p = 80$, $K_i = 0$, and $K_d = 30$. Simulation time is $3s$

```
1  Template  Control(float a, float theta, float omega){
2      while (true) {
3          a = kp*theta  +  kd*omega;
4          wait(0.08)
5      }
6  }
7
8  Template  Pendulum(float theta, float omega, float a){
9     (dot  a = 0)
10     ||
11    (dot  theta = omega )
12     ||
13    (dot  omega = g*sin(theta) - a*cos(theta) )
14     until  (false)
15 }
16
17 Main {
18        {
19                Pendulum(position, velocity, acc)
20                ||
21                Control(acc, position, velocity)
22        }
23 }
```

Fig. 9 The **HML** model for discrete control of inverted pendulum. Variable definitions and constraints are omitted, `acc` is the acceleration of the cart

6.2 Discrete PID control : reachability analysis

The **HML** model for discrete control of inverted pendulum is provided in Fig. 9 with $K_p = 80$, $K_i = 0$, and $K_d = 15$, where K_i is removed in the model. The sampling time is $0.08s$, i.e., the acceleration is calculated for every $0.08s$. With the maximum unrolling depth of 11, consider the following properties:

1. The position and velocity can be in the range of $-0.01 \sim +0.01$:

$$(position \in [-0.01, 0.01]) \wedge (velocity \in [-0.01, 0.01]).$$

The verification result is SAT (i.e., δ-SAT), meaning that the set of states represented by this property is reachable for this model with depth 11. The position is illustrated in Fig. 10.

2. The position and velocity are not outside of the interval $[-0.01, 0.01]$:

$$position \notin [-0.01, 0.01] \vee velocity \notin [-0.01, 0.01].$$

The verification result is UNSAT (i.e., unsatisfiable), thus the corresponding states are unreachable, then it is known that the position and velocity are stable within

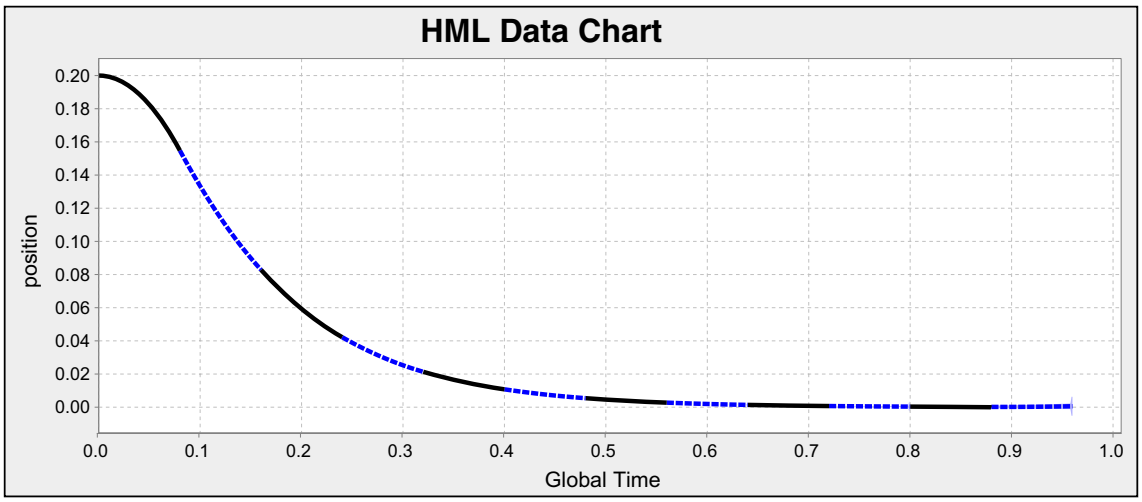


Fig. 10 Angular position under discrete PID control. Sampling time is 0.08s, simulation time is 0.96s. The curve for each depth is solid-black or dotted-blue

Fig. 11 The **HML** model for rod reactor. `final` is a keyword for declaration of constant value

```
final int KNR=50, KRF=56, KRS=60; //constants
float tp=510, timer1=20, timer2=20; //variables
... //other variables and bounded constraints
Template  RodIn(int k){
  (dot  tp = 0.1*tp - k) || (dot  timer1 = 1)
   || (dot  timer2 = 1)  until  (tp<=510)
}

Template  RodOut(int k){
  (dot  tp = 0.1*tp - k) || (dot  timer1 = 1)
   || (dot  timer2 = 1)  until  (tp>=550
   and (timer1>=20 or timer2>=20))
}

Main {
  while (true){
        RodOut(KNR);    //No rods are in the reactor
        RodIn(KRF);     //Use the first cooling rod
        timer1=0;       //Reset the first timer
        RodOut(KNR);    //No rods are in the reactor
        RodIn(KRS);     //Use the Second cooling rod
        timer2=0        //Reset the second timer
  }
}
```

the interval $[-0.01, 0.01]$. Moreover, as the result illustrated in Fig. 10, the position may be stable after 0.6s. Thus, the reachability analysis is taken for the model with depth 8, the result is also UNSAT.

The first property indicates the position and velocity can be stable. The second one emphasizes that the angular position is stable without perturbation for the inverted pendulum at depth 11.

The experiment environment for this case study is a machine with 3.2GHz Quad-Core Intel Core i5-4460 processor, 24GB RAM, and 64-bit Ubuntu 14.10 (Utopic). The time consumed for the verification of continuous control is less than 2 minutes, but the time for the discrete control is about 2 hours and 10 minutes with 29.8 GB data for reachable states.

7 Case Study-2 : rod reactor

This case consists of a (nuclear) reactor tank and two cooling rods (called rod-1 and rod-2). The goal in this case is to control the temperature of the reactor tank in a certain range by inserting in or pulling out the cooling rods [39]. The rods have to be used alternatively because they cannot be in the reactor tank for a long time.

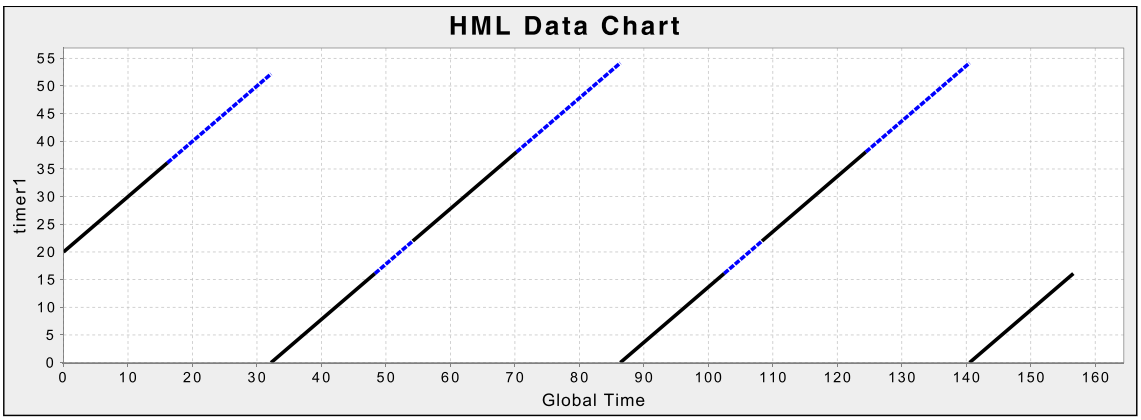


Fig. 12 The value of the timer for rod-1

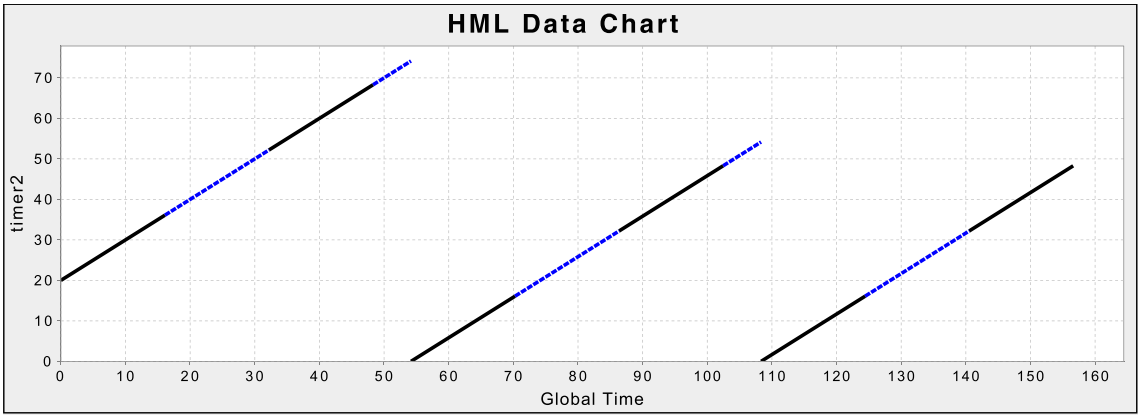


Fig. 13 The value of the timer for rod-2

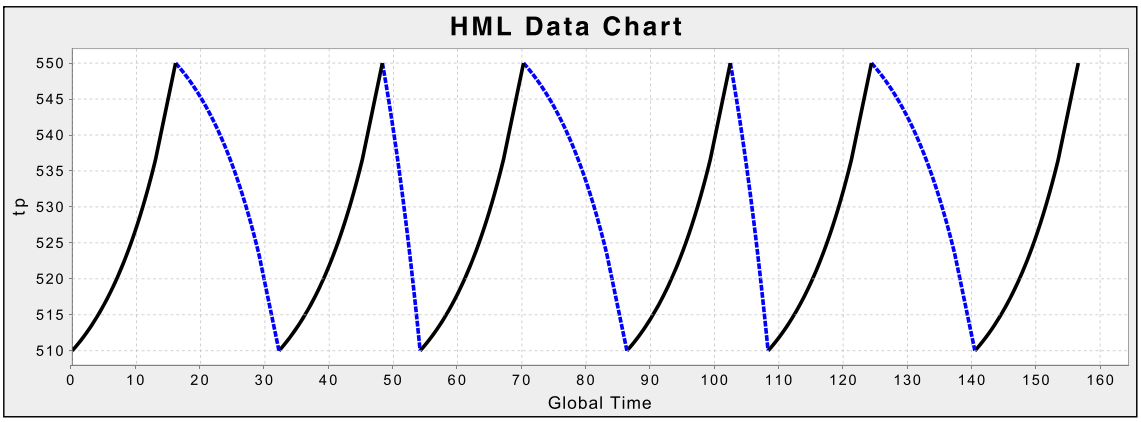


Fig. 14 The temperature of the reactor tank

Fig. 15 The temperature under one running path, KNR $\in$ [40, 50]

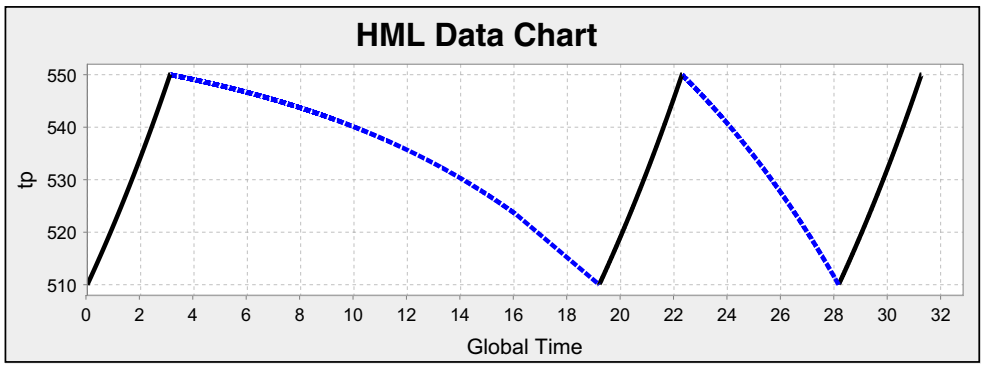
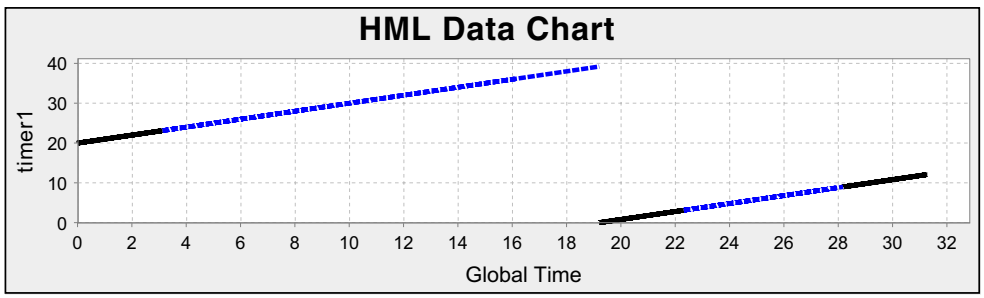


Fig. 16 The timer of rod-1 under one running path, KNR $\in$ [40, 50]



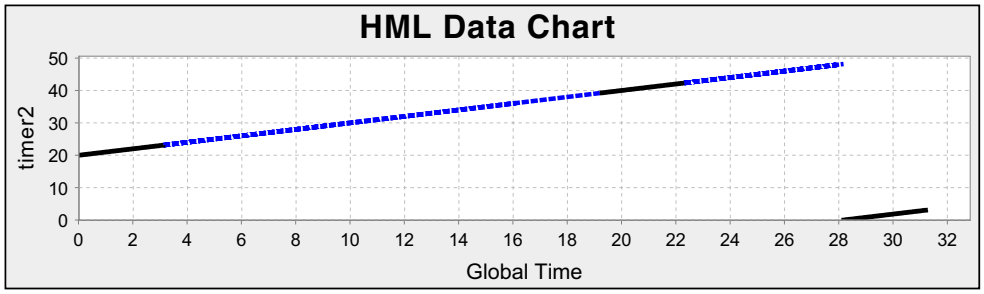


Fig. 17 The timer of rod-2 under one running path, KNR ∈ [40, 50]

It is needed to shutdown the reactor if no cooling rods are available when the temperature is critical (high). The reactor contains two kinds of procedures, that are,

- No rods are in the tank. The temperature of the reactor is increasing over time.
- One cooling rod is positioned in the tank. The reactor's temperature is decreasing.

The following differential equations specify the dynamics:

$$\dot{tp} = \frac{1}{10} \cdot tp - 50 \tag{4}$$

$$\dot{tp} = \frac{1}{10} \cdot tp - 56 \tag{5}$$

$$\dot{tp} = \frac{1}{10} \cdot tp - 60 \tag{6}$$

where, tp represents the temperature of the tank. Equation 4 is employed for the situation when no rods are in the tank. Equations 5 and 6 are used for the cooling procedures of two rods, respectively. These two rods distinguished in their cooling dynamics.

In addition, two clocks $timer1$ and $timer2$ are needed for the measurement of the time elapsed since the last use of the rod-1 and rod-2, respectively.

The **HML** model for the rod reactor is presented in Fig. 11. The templates `RodIn` and `RodOut` specify the dynamics of the reactor when a rod is in the tank or not. Obviously, the template `RodIn` is reused for the cooling dynamics of rod-1 and rod-2 as they can be initialized with two different parameters `KRF` and `KRS`, respectively.

Initially, the temperature of the tank is 510 °C, and the two timers of rod-1 and rod-2 are 20 s. To check whether there exists the shutdown scenario for the reactor, consider the properties as follows,

- $(tp \geq 550) \wedge ((timer1 \geq 20) \vee (timer2 \geq 20))$,
- $(tp \geq 550) \wedge (timer1 < 20) \wedge (timer2 < 20)$,

where, 550 is the value of critical temperature of the reactor tank, and 20 is the minimum time for a rod to be available for the cooling task. The verification results for the above two properties in the prototype tool with the maximum unrolling depth of 10 for the model are SAT and UNSAT, respectively. The reachable states for the system are illustrated in Fig. 14 for the temperature of the reactor tank, and the timers of two rods in Figs. 12 and 13. Therefore, the reactor would not be shutdown in this case.

However, consider the following differential equation which specifies the dynamics of temperature when no rods are in the reactor tank,

$$\dot{tp} = \frac{1}{10} \cdot tp - \texttt{KNR}, \text{ where } \texttt{KNR} \in [40, 50],$$

inwhich, `KNR` is a bounded parameter evaluated in the closed interval [40, 50]. In **HML**, `KNR` can be declared as a `float` variable with an interval be its initial value, this can be specified as follows:

```
float KNR = [40.0, 50.0];
```

Thus, the value of `KNR` is non-deterministic (within [40, 50]) in the new **HML** model of rod reactor. Also, whether the reactor would be shutdown in this new case can be checked.

Figures 15, 16 and 17 depict the reachable states of one running path of the reactor for the new case with `KNR` ∈ [40, 50] and the unrolling depth is 4. These three figures indicate that there are no rods available (both `timer1` and `timer2` take values that are less than 20) when the temperature reaches 550 °C at the end point of the time. For this case, the reactor has to shut down before the running time reaches 32 (s).

8 Conclusion

In this paper, a modeling language **HML** has been extended with types, constraints and templates to better fulfill the modeling of hybrid systems. The **HML** model has been translated into SMT formulas which are acceptable by the SMT solver **dReal** for bounded reachability analysis of non-linear hybrid systems. Moreover, the simulation and

reachability analysis have been integrated in a prototype tool, and a case study for an inverted pendulum has been presented, also a rod reactor system for the control of the temperature of the reactor tank has been verified. The utility of this work has been checked, the language **HML** is capable of modeling and verification for (linear and nonlinear) hybrid systems based on simulation and bounded reachability analysis.

Future works may include: (1) introducing probability features in **HML** for modeling of stochastic behaviors in stochastic hybrid systems; (2) reachability analysis stochastic **HML** models with continuous-time Markov decision processes [21].

Acknowledgments This work was partly supported by the Danish National Research Foundation and the National Natural Science Foundation of China (Grant No. 61361136002) for the Danish-Chinese Center for Cyber Physical Systems. It was also supported by National Natural Science Foundation of China (Grant No. 61321064), Shanghai Collaborative Innovation Center of Trustworthy Software for Internet of Things (No. ZF1213) and Shanghai Minhang Talent Project.

Appendix A: Definitions Related to Computable Functions

Some basic definitions that are related to this paper are reviewed here. For more details, see [29, 40].

Definition 5 (**Names**) A name of $v \in \mathbb{R}$ is any function $\gamma_v : \mathbb{N} \to \mathbb{D}$ satisfying

$$\forall i \in \mathbb{N},\ |\gamma_v(i) - v| < 2^{-i},$$

where, $\mathbb{D}$ is the set of all dyadic rationals (numbers of the form $\frac{\varphi}{2^\psi}$ for an integer φ and natural number ψ).

For multi-dimensional name of $\mathbf{v} \in \mathbb{R}^n$, $\gamma_{\mathbf{v}} = \langle \gamma_{\mathbf{v}_1}, ..., \gamma_{\mathbf{v}_n} \rangle$. The name of v is a sequence of dyadic rationals converging to it.

Definition 6 (**Computable Reals**) A real number $v \in \mathbb{R}$ is computable if there exists a name γ_v of v that is a computable function.

A real function is computable if its value can be approximated for arbitrary precision by a function-oracle Turing machine.

Definition 7 (Computable Functions) Let $V \subseteq \mathbb{R}^n$, function $f : V \to \mathbb{R}$ is computable if there exists a function-oracle Turing machine **F-OTM**$_f$ calculating a rational number $\mathcal{Q}_f^{\gamma_{\mathbf{v}}}(i)$ for $i \in \mathbb{N}$ and $\gamma_{\mathbf{v}}$ satisfying $|\mathcal{Q}_f^{\gamma_{\mathbf{v}}}(i) - f(\mathbf{v})| < 2^{-i}$.

A function-oracle Turing machine **F-OTM** is an ordinary Turing machine except that **F-OTM** have an additional tape for query and two additional states (query and answer states, respectively).

When **F-OTM** enters the query state, the oracle γ replaces the current string v by $\gamma(v)$ in the query tape, then the tape head returns to the first cell of the query tape, and the state of the machine is reset to the answer state.

For "types" of functions, informally, integer and rational numbers are Type-0 objects, a real number can be considered as a Type-1 function that maps a Type-0 object to type-0 object. Type-2 functions are those functions that map from Type-1 functions to Type-1 (or Type-0) functions. Thus, a function as $f : \mathbb{R} \to \mathbb{R}$ is a Type-2 function.

Appendix B: Sample SMT Formulas

Here, we present one sample SMT formulas that are encoded (with maximum unrolling depth 0) for our **HML** model. In the following list of SMT formulas, there are some important parts should be explained:

- **Logic.** In line 1, `set-logic` is a keyword of SMT standard language. `(set-logic L)` tells the solver (**dReal**) what logic `L` is being used for satisfiability checking of the SMT formulas. Here, `QF_NRA_ODE` is the logic implemented in **dReal** for non-linear differential equations.
- **Variable.** `(declare-fun position () Real)` declares a variable named `position` with data type `Real`.
- **Differential equation.** `(define-ode flow_n F)` defines a differential equation, number `n` is an index for this equation, `F` is the formula representing the concrete form of the equation, for example line 16 defines a simultaneous differential equation. E.g., SMT formula `(= d/dt[clock] 1)` can be considered as a differential equation $\frac{\mathtt{dclock}}{\mathtt{dt}} = 1$ for variable `clock`.
- **Assert.** The assert command `(assert F)` instructs the SMT solver to assume that the formula `F` is true.
- **Check.** The check command `(check-sat)` tells the SMT solver to do the checking of the satisfiability of the SMT formulas.
- **Property.** The properties that users want to check can be added as normal assert commands.
- **Exit.** The exit command `(exit)` returns success and the SMT solver **dReal** terminates.

```
(set-logic QF_NRA_ODE)
(declare-fun position () Real)
(declare-fun velocity () Real)
(declare-fun clock () Real)
(declare-fun global () Real)
(declare-fun position_0_0 () Real)
(declare-fun position_0_t () Real)
(declare-fun velocity_0_0 () Real)
(declare-fun velocity_0_t () Real)
(declare-fun clock_0_0 () Real)
(declare-fun clock_0_t () Real)
(declare-fun global_0_0 () Real)
(declare-fun global_0_t () Real)
(declare-fun time_0 () Real)
(declare-fun mode_0 () Int)
(define-ode flow_1 ((= d/dt[position] velocity) (=
      d/dt[velocity] (- (* 9.8 position) (* (* 80
      position) (cos position)))(= d/dt[clock] 1)(=
       d/dt[global] 1)))
(assert (<= -10 position_0_0))
(assert (<= position_0_0 10))
(assert (<= -10 position_0_t))
(assert (<= position_0_t 10))
(assert (<= -40 velocity_0_0))
(assert (<= velocity_0_0 40))
(assert (<= -40 velocity_0_t))
(assert (<= velocity_0_t 40))
(assert (<= 0 time_0))
(assert (<= time_0 10))
(assert (<= 0 clock_0_0))
(assert (<= clock_0_0 10))
(assert (<= 0 clock_0_t))
(assert (<= clock_0_t 10))
(assert (<= 0 global_0_0))
(assert (<= global_0_0 10))
(assert (<= 0 global_0_t))
(assert (<= global_0_t 10))
(assert (<= 1 mode_0))
(assert (<= mode_0 1))
(assert (and (= mode_0 1) (= global_0_0 0) (=
      position_0_0 0.2) (= velocity_0_0 0) (=
      clock_0_0 0) (= [position_0_t velocity_0_t
      clock_0_t global_0_t]  (integral 0. time_0 [
      position_0_0 velocity_0_0 clock_0_0
      global_0_0]  flow_1)) (forall_t 1 [0 time_0]
      (not false))(= mode_0 1) (= global_0_t 5)))
(check-sat)
(exit)
```

References

1. Alur R, Courcoubetis C, Henzinger TA, Ho PH (1993) Hybrid Automata: An Algorithmic Approach to the Specification and Analysis of Hybrid Systems. In: Hybrid Systems, LNCS, vol 736. Springer, pp 209–229. doi:10.1007/3-540-57318-6_30
2. Åström KJ, Hägglund T (2006) Advanced PID control. ISA-The Instrumentation, Systems, and Automation Society, Research Triangle Park, NC 27709
3. Baeten JCM, Weijland WP (1990) Process Algebra. Cambridge University Press
4. Barrett C, Stump A, Tinelli C (2010) The SMT-LIB Standard: Version 2.0. Tech. rep., Department of Computer Science, The University of Iowa, available at www.SMT-LIB.org
5. Berz M (1999) Modern Map Methods in Particle Beam Physics. ADV IMAG ELECT PHYS, vol 108. Elsevier. doi:10.1016/S1076-5670(08)70222-2
6. Berz M, Makino K (1998) Verified Integration of ODEs and Flows Using Differential Algebraic Methods on High-Order Taylor Models. Reliab Comput 4(4):361–369
7. Bruttomesso R, Pek E, Sharygina N, Tsitovich A (2010) The OpenSMT Solver. In: Proceedings of TACAS, LNCS, vol 6015. Springer, Berlin, pp 150–153
8. Chang WD, Shih SP (2010) PID Controller Design of Nonlinear Systems Using an Improved Particle Swarm Optimization Approach. Commun Nonlinear Sci 15(11):3632–3639
9. Chen X, Ábrahám E, Sankaranarayanan S (2013) Flow*: An Analyzer for Non-linear Hybrid Systems. In: Proceedings of CAV, LNCS, vol 8044. Springer, pp 258–263
10. Chen X, Schupp S, Makhlouf I, Ábrahám E, Frehse G, Kowalewski S (2015) A Benchmark Suite for Hybrid Systems Reachability Analysis. In: NASA Formal Methods, LNCS, vol 9058. Springer, pp 408–414
11. Cuijpers PJL, Reniers MA (2005) Hybrid Process Algebra. J Logic Algebr Progr 62(2):191–245
12. Frehse G (2008) PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. Int J Softw Tools Technol Transfer 10(3):263–279
13. Frehse G, Han Z, Krogh B (2004) Assume-Guarantee Reasoning for Hybrid I/O-Automata by Over-Approximation of Continuous Interaction. In: Proceedings of CDC. IEEE, pp 479–484
14. Frehse G, Le Guernic C, Donzé A, Cotton S, Ray R, Lebeltel O, Ripado R, Girard A, Dang T, Maler O (2011) SpaceEx: Scalable Verification of Hybrid Systems. In: Proceedings of CAV, LNCS, vol 6806. Springer, pp 379–395
15. Fritzson P, Engelson V (1998) Modelica–A Unified Object-Oriented Language for System Modeling and Simulation. In: Proceedings of ECOOP, LNCS, vol 1445. Springer, pp 67–90
16. Gao S, Avigad J, Clarke EM (2012) Delta-Decidability Over the Reals. In: Proceedings of LICS. IEEE, pp 305–314
17. Gao S, Kong S, Clarke EM (2013a) dReal: An SMT Solver for Nonlinear Theories Over the Reals. In: Proceedings of CADE. Springer, pp 208–214
18. Gao S, Kong S, Clarke EM (2013b) Satisfiability Modulo ODEs. In: Proceedings of FMCAD. IEEE, pp 105–112
19. Granvilliers L, Benhamou F (2006) Algorithm 852: RealPaver: An Interval Solver Using Constraint Satisfaction Techniques. ACM T Math Software 32(1):138–156
20. Guernic CL, Girard A (2010) Reachability Analysis of Linear Systems Using Support Functions. Nonlinear Analysis: Hybrid Systems 4(2):250–262
21. Guo X, Hernndez-Lerma O (2009) Continuous-time markov decision processes. Stochastic Modelling and Applied Probability, vol 62. Springer, Berlin, pp 9–18
22. Harel D (1987) Statecharts: A Visual Formalism for Complex Systems. Sci Comput Program 8(87):231–274
23. He J (1994) From CSP to Hybrid Systems. In: A Classical Mind, Essays in Honour of C.A.R. Hoare, Prentice Hall International, pp 171–189
24. He J (2013) Hybrid Relation Calculus. In: Proceedings of ICECCS. IEEE, p 2
25. Henzinger TA (1996) The Theory of Hybrid Automata. In: Proceedings of LICS. IEEE, pp 278–292
26. Henzinger TA, Ho PH, Wong-Toi H (1997) HyTech : A Model Checker for Hybrid Systems. Int J Softw Tools Technol Transfer 1(1–2):110–122
27. Henzinger TA, Kopke PW, Puri A, Varaiya P (1998) What's Decidable about Hybrid Automata? Journal of Computer and System Sciences 57:94–124
28. Hoare CAR (1985) Communicating Sequential Processes. Prentice Hall

29. Ko KI (1991) Complexity Theory of Real Functions. Birkhauser Boston Inc., Cambridge
30. Kong S, Gao S, Chen W, Clarke E (2015) dReach: Delta-Reachability Analysis for Hybrid Systems. In: Proceedings of TACAS, Springer-Verlag, LNCS, vol 9035, pp 200–205
31. Lynch N, Segala R, Vaandrager F, Weinberg H (1996) Hybrid I/O Automata. In: Hybrid Systems III, LNCS, vol 1066. Springer, pp 496–510
32. Lynch N, Segala R, Vaandrager F (2001) Hybrid I/O Automata Revisited. In: Proceedings of HSCC, LNCS, vol 2034. Springer, pp 403–417
33. Maler O, Manna Z, Pnueli A (1992) From Timed to Hybrid Systems. In: Real-Time: Theory in Practice, LNCS, vol 600. Springer, Berlin, pp 447–484
34. MathWorks (2015a) Simulink
35. MathWorks (2015b) Stateflow
36. Nieuwenhuis R, Oliveras A, Tinelli C (2006) Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). J ACM 53(6):937–977
37. Phan PA, Gale TJ (2008) Direct Adaptive Fuzzy Control with A Self-Structuring Algorithm. Fuzzy Set Syst 159(8):871–899
38. Platzer A (2010) Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics. Springer
39. Von Mohrenschildt M (2001) Symbolic Verification of Hybrid Systems: An Algebraic Approach. Eur J Control 7(5):541–556
40. Weihrauch K (2000) Computable Analysis: An Introduction. Springer
41. Yi J, Yubazaki N (2000) Stabilization Fuzzy Control of Inverted Pendulum Systems. Artif Intell Eng 14(2):153–163
42. Zhou C, Wang J, Ravn AP (1996) A Formal Description of Hybrid Systems. In: Hybrid Systems III, LNCS, vol 1066. Springer, pp 511–530