EPiC Series in Computer Science

Volume 34, 2015, Pages 188–195

ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems

SMT Encoding of Hybrid Systems in dReal*

Kyungmin Bae[1], Soonho Kong[1], and Sicun Gao[2]

[1] Carnegie Mellon University, PA, USA
[2] Massachusetts Institute of Technology, MA, USA

Abstract

Analysis problems of hybrid systems, involving nonlinear real functions and ordinary differential equations, can be reduced to SMT (satisfiability modulo theories) problems over the real numbers. The dReal solver can automatically check the satisfiability of such SMT formulas up to a given precision $\delta > 0$. This paper explains how bounded model checking problems of hybrid systems are encoded in dReal. In particular, a novel SMT syntax of dReal enables to effectively represent networks of hybrid systems in a modular way. We illustrate SMT encoding in dReal with simple nonlinear hybrid systems.

1 Introduction

An SMT (satisfiability modulo theories) problem is to check the satisfiability of first-order formulas with respect to certain decidable logical theories. Recently, SMT-based techniques, such as [4, 5, 6, 9], have been proposed to automatically analyze a general class of hybrid systems. One advantage of this approach is that we can apply existing state-of-the-art SMT techniques and tools, which are proven to be effective for analyzing discrete systems. Moreover, it is fairly easy to combine different numerical algorithms and decision procedures to analyze the continuous behavior of hybrid systems.

In SMT-based approaches, formal analysis problems of hybrid systems are encoded as SMT formulas. Since hybrid systems usually involve nonlinear real functions and ordinary differential equations (ODEs), the satisfaction problems of these formulas are in general undecidable. But if we take into account robustness properties under numerical perturbations, such problems become decidable up to an arbitrary precision $\delta > 0$ [7, 9]. Suppose that $\delta > 0$, provided by the user, is the bound on numerical errors that is tolerable in the analysis. A *δ-complete decision procedure* for an SMT formula ϕ returns false if ϕ is unsatisfiable, and returns true if its syntactic numerical perturbation of ϕ by bound δ is satisfiable. This is practically very useful since it is not possible to sample exact values of physical parameters in reality.

The dReal tool [8] is an SMT solver to check the satisfiability of logic formulas over the real numbers up to a given precision $\delta > 0$ using δ-complete decision procedures, involving various non-linear real functions, such as polynomials, exponentiation, trigonometric functions, and solutions of Lipschitz-continuous ordinary differential equations (ODEs). dReal is built on

*This work was partially supported by ONR Grant N000141310090, NSF CPS-1330014 and CPS-1446675, and Air Force STTR Grant F14A-T06-0230.

several existing packages to combine SMT and numerical techniques: opensmt [3] for general SMT decision procedures, realpaver [10] for the interval constraint propagation (ICP), and CAPD [1] for computing interval-enclosures of ODEs.

This paper shows SMT encoding for formal analysis of hybrid systems in dReal. To generate such SMT formulas from hybrid system specifications, dReal provide a front-end, called dReach [12], and the user need not know the details of SMT encoding in practice. A number of benchmarks and examples in this paper are available at http://dreal.github.io.

2 Background: Hybrid Automata

In this paper we use hybrid automata [11] as formal models of hybrid systems. Discrete states of a hybrid automaton H are given by a set of control modes Q. Physical states of H are given by a finite set $X = \{x_1, \ldots, x_n\}$ of real-numbered variables. A combined state of H is then a pair $(q, \vec{v})$ of a mode $q \in Q$ and a vector $\vec{v}$ of real numbers. Each mode q has an *invariant condition*, denoted by predicate $inv_q(\vec{x})$, that defines the set of all possible values of X in mode q. Similarly, a set of initial states is expressed by using predicates $init_q(\vec{x})$. The continuous dynamics of a hybrid automaton H is specified by a *flow condition* of the form $\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = flow_q(\vec{x})$ for mode q, expressing a system of ordinary differential equations (ODEs) for the variables X. A discrete transition between two modes q and q' is specified by a *jump condition* of the form $jump_{q,a,q'}(\vec{x}, \vec{x}')$, which can also be identified with action $a \in \Sigma$, given a set of actions Σ.

Example 1 (Periodic Water Tank)**.** *The water level in a tank is periodically controlled by its pump. For each period T, the pump is on if the water level is lower than $L_{\min}$, and the pump is off if the water level is higher than $L_{\max}$. The pump has two control modes $Q = \{m_{\mathtt{on}}, m_{\mathtt{off}}\}$ to denote its status. There are two real-numbered variables $X = \{x, \tau\}$ with x for the water level and τ for its timer. Both modes have the invariant condition $inv(x, \tau) \equiv 0 \leq \tau \leq T$. Initially, the pump is on, $L_{\min} < x < L_{\max}$, and $\tau = 0$. The water level x and the timer τ change according to the nonlinear flow conditions:*

$$\begin{array}{l} \frac{\mathrm{d}x}{\mathrm{d}t} = (p - a\sqrt{2g}\sqrt{x})/A \\ \frac{\mathrm{d}\tau}{\mathrm{d}t} = 1 \end{array} \quad \text{if } q = m_{\mathtt{on}}, \qquad \begin{array}{l} \frac{\mathrm{d}x}{\mathrm{d}t} = -a\sqrt{2g}\sqrt{x}/A \\ \frac{\mathrm{d}\tau}{\mathrm{d}t} = 1 \end{array} \quad \text{if } q = m_{\mathtt{off}},$$

where A, p, a are constants determined by the size of the tank, the power of the pump, and the output of the tank. The jump conditions are given by:

- $jump_{m_{\mathtt{on}},a,m_{\mathtt{on}}}(x, \tau, x', \tau') \equiv \tau = T \wedge x \leq L_{\max} \wedge x' = x \wedge \tau' = 0$,
- $jump_{m_{\mathtt{on}},a,m_{\mathtt{off}}}(x, \tau, x', \tau') \equiv \tau = T \wedge x > L_{\max} \wedge x' = x \wedge \tau' = 0$,
- $jump_{m_{\mathtt{off}},a,m_{\mathtt{on}}}(x, \tau, x', \tau') \equiv \tau = T \wedge x < L_{\min} \wedge x' = x \wedge \tau' = 0$, *and*
- $jump_{m_{\mathtt{off}},a,m_{\mathtt{off}}}(x, \tau, x', \tau') \equiv \tau = T \wedge x \geq L_{\min} \wedge x' = x \wedge \tau' = 0$.

A hybrid system is often composed of a network of smaller hybrid systems, which is specified as a *parallel composition* of hybrid automata. For a parallel composition $H_1 \parallel H_2$ of two hybrid automata H_1 and H_2, its control mode is a pair $(q_1, q_2) \in Q_1 \times Q_2$ of H_1's mode q_1 and H_2's mode q_2. A physical state of $H_1 \parallel H_2$ is given by the set $X_1 \cup X_2$ of the real-numbered variables from both H_1 and H_2. For mode (q_1, q_2), an invariant condition $inv_{(q_1,q_2)}(\vec{x}_1, \vec{x}_2)$ holds iff both H_1's invariant condition $inv^1_{q_1}(\vec{x}_1)$ and H_2's condition $inv^2_{q_2}(\vec{x}_2)$ hold. Likewise, an initial condition $init_{(q_1,q_2)}(\vec{x}_1, \vec{x}_2)$ holds iff both $init^1_{q_1}(\vec{x}_1)$ and $init^2_{q_2}(\vec{x}_2)$ hold.

The continuous interaction between H_1 and H_2 is modeled in their parallel composition $H_1 \parallel H_2$ by using shared variables $X_1 \cap X_2$, and their discrete communication is modeled by

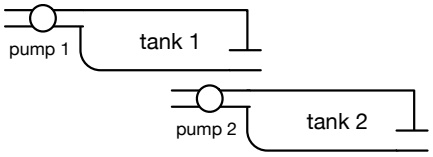

Figure 1: Networked water tanks (output of tank 1 goes to input of tank 2).

using joint synchronous actions. A flow condition of mode (q_1, q_2) is a system of ODEs for $X_1 \cup X_2$ of the form $\frac{\mathrm{d}\vec{x}_1}{\mathrm{d}t} = \mathit{flow}^1_{q_1}(\vec{x}_1) \wedge \frac{\mathrm{d}\vec{x}_2}{\mathrm{d}t} = \mathit{flow}^2_{q_2}(\vec{x}_2)$, where the flow condition of H_1 in mode q_1 is compatible with the flow condition of H_2 in mode q_2 for the shared variables $X_1 \cap X_2$. For modes (q_1, q_2) and (q_1', q_2'), a jump condition $\mathit{jump}_{(q_1,q_2),a,(q_1',q_2')}(\vec{x}_1, \vec{x}_2, \vec{x}_1', \vec{x}_2')$ with action $a \in \Sigma_1 \cup \Sigma_2$ holds iff for $i = 1, 2$, either $\mathit{jump}_{q_i,a,q_i'}(\vec{x}_i, \vec{x}_i')$ holds when a is an action of H_i, or $q_i = q_i'$ and $\vec{x}_i = \vec{x}_i'$ when a is not an action of H_i. Notice that if a is a common action, then H_1 and H_2 must synchronize their transitions with action a.

Example 2 (Networked Water Tanks)**.** *Several water tanks in Example 1 are connected by pipes in sequence as shown in Figure 1 (adapted from [13, 14]). Every tank controller* shares *the same timer variable τ with the flow condition $\frac{\mathrm{d}\tau}{\mathrm{d}t} = 1$. That is, for tank i, $X_i = \{x_i, \tau\}$. The water level of each tank now depends on the levels of the adjacent tanks as well as the pump's mode. Hence, the water level x_i of tank i changes according to the flow condition:*

$$\begin{aligned} \tfrac{\mathrm{d}x_i}{\mathrm{d}t} &= \big((p_i + a\sqrt{2g}\sqrt{x_{i-1}}) - a\sqrt{2g}\sqrt{x_i}\big)/A_i \quad \textit{if } q_i = m_{\mathrm{on}}, \\ \tfrac{\mathrm{d}x_i}{\mathrm{d}t} &= (a\sqrt{2g}\sqrt{x_{i-1}} - a\sqrt{2g}\sqrt{x_i})/A_i \qquad\quad \textit{if } q_i = m_{\mathrm{off}}, \end{aligned}$$

where A_i, p_i, a are determined by the size of the tank, the power of the pump, and the width of the pipe ($x_0 = 0$ for the leftmost tank 1). Each water tank controller has the same period $T \in \mathbb{R}$, and its jump conditions is labeled with the same action a. Therefore, every controller synchronously *performs its discrete transitions with the common action a according to the jump conditions in Example 1 (for each period T, the pump is on if $x_i \leq L_{\min}$, and off if $x_i > L_{\max}$).*

3 SMT Encoding of Hybrid Automata

This section explains how bounded model checking problems of hybrid automata are encoded as SMT problems in dReal. The syntax of SMT formulas in dReal follows version 2 of the SMT-LIB standard [2] with extensions to declare systems of ordinary differential equations and their solutions. We illustrate such SMT encoding with the water tank examples. The dReach front-end [12] automatically generates such formulas from hybrid automata specifications.

Variable Declarations. For bounded model checking of a hybrid automata H with depth N, we need to encode its behavior for N steps of mode changes. A mode of H at the i-th step is expressed as a variable mode_i. A variable time_i is declared to denote the time elapsed within mode_i. For each state variable $y \in X$, we use two sets of variables `y_`i`_0` (0-variables) and `y_`i`_t` (t-variables) to denote the values of x at the beginning and the end of the i-th step, respectively. Such variables are introduced with the `declare-fun` keyword. For example, the two state variables x and τ for the i-th step in Example 1 are declared as follows:

```
(declare-fun mode_i () Real)        (declare-fun time_i () Real)
(declare-fun tau_i_0 () Real)       (declare-fun tau_i_t () Real)
```

```
(declare-fun x_i_0 () Real)          (declare-fun x_i_t () Real)
```

In dReal, we need that each real-numbered variable is bounded for δ-complete decision procedures. For variable y, the `assert` statement `(assert (and (<= a y) (<= y b)))` declares its bound $a \leq y \leq b$. For example, in Example 1:

```
(assert (and (<= 0 mode_i)    (<= mode_i 1)   (<= 0 time_i)   (<= time_i T)
             (<= 0 tau_i_0)   (<= tau_i_0 T) (<= 0 tau_i_t)  (<= tau_i_t T)
             (< 0 x_i_0)      (<= x_i_0 10)   (< 0 x_i_t)     (<= x_i_t 10)))
```

Flow Declarations. In order to declare systems of ODEs, dReal introduces the new keyword `define-ode`. A system of ODEs ($\frac{\mathrm{d}x_1}{\mathrm{d}t} = u_1, \ldots, \frac{\mathrm{d}x_n}{\mathrm{d}t} = u_n$) is defined by using the statement `(define-ode flow ((= d/dt[`x_1`] `u_1`)...(= d/dt[`x_n`] `u_n`)))` with identifier `flow`. For example, the two nonlinear ODE systems in Example 1 are declared by:[1]

```
(declare-fun x () Real)
(declare-fun tau () Real)
(define-ode flow_1 ((= d/dt[x] (/ (- q (* (* a (sqrt (* 2 g))) (sqrt x))) A))
                    (= d/dt[tau] 1)))
(define-ode flow_2 ((= d/dt[x] (/ (* (* -a (sqrt (* 2 g))) (sqrt x)) A))
                    (= d/dt[tau] 1)))
```

Initial Conditions. We define initial conditions as formulas on the initial mode variable $mode_0$, and the 0-variables to denote the values at the beginning of the initial step (that is, for variable y, the variable `y_0_0`). Such conditions are declared using `assert` statements in dReal. For Example 1, when mode $m_{\mathtt{on}}$ is denoted by number 0 and $m_{\mathtt{off}}$ is denoted by number 1:

```
(assert (and (= mode_0 0) (= tau_0_0 0) (< L_min x_0_0) (< x_0_0 L_max)))
```

Jump Conditions. SMT encoding of jump conditions is straightforward. Jump conditions are directly expressed as SMT formulas using `assert` statements and Boolean connectives. Since jumps happen at the end of each step, we define constraints between t-variables of the current step and 0-variables of the next step. For example, the jump conditions of the i-th step in Example 1 are written in dReal as follows, where $j = i + 1$:

```
(assert (or (and (= mode_i 0) (= mode_j 0)
                 (= tau_i_t T) (<= x_i_t L_max) (= x_j_0 x_i_t) (= tau_j_0 0))
            (and (= mode_i 0) (= mode_j 1)
                 (= tau_i_t T) (> x_i_t L_max) (= x_j_0 x_i_t) (= tau_j_0 0))
            (and (= mode_i 1) (= mode_j 0)
                 (= tau_i_t T) (< x_i_t L_min) (= x_j_0 x_i_t) (= tau_j_0 0))
            (and (= mode_i 1) (= mode_j 1)
                 (= tau_i_t T) (>= x_i_t L_min) (= x_j_0 x_i_t) (= tau_j_0 0))))
```

[1] The `sqrt` function is Lipschitz continuous only if its domain is positive. The water level is always positive in our example. dReal reports an exception if the condition cannot be met.

Flow Conditions. Solutions of ODEs are expressed by using the new keyword `integral` in dReal. For example, An integral term $(y_0^t, \ldots, y_n^t) = (y_0^0, \ldots, y_n^0) + \int_0^t \mathit{flow}_i(t)\,\mathrm{d}t$ is written as the statement `(= [y_0_t...y_n_t] (integral 0 t [y_0_0...y_n_0] flow_i))`, where flow_i is declared as a `define-ode` statement. For each i-th step, flow conditions are defined by constraints between 0-variables, t-variables, and time duration of the i-th step according to current modes. For the water tank example in Example 1, given 0-variables $[x_i^0, \tau_i^0]$ at the i-step, if the i-th step's duration is time_i, then the values of the t-variables $[x_i^t, \tau_i^t]$ are defined using `integral` statements according to its mode mode_i as follows:

```
(assert (or (and (= mode_i 0)
                 (= [x_i_t tau_i_t] (integral 0. time_i [x_i_0 tau_i_0] flow_1)))
            (and (= mode_i 1)
                 (= [x_i_t tau_i_t] (integral 0. time_i [x_i_0 tau_i_0] flow_2)))))
```

Invariant Conditions. In a hybrid automaton H, an invariant condition of mode q must be satisfied at any time as long as H's current mode is q. To encode invariant conditions, we need to deal with *universally quantified* formulas over time. Therefore, dReal introduce a new keyword `forall_t`: the statement `(forall_t` n `[0` u`]` $\phi(\vec{x}_t)$`)` declares that $\phi(\vec{x}_t)$ holds for any time $t \in [0, u]$ with the flow condition flow_n (that is, $\forall t \in [0, u].\ (\vec{x}_t = \vec{x}_0 + \int_0^t \mathit{flow}_n\,\mathrm{d}t) \to \phi(\vec{x}_t)$). For example, the invariant condition $0 \le \tau \le T$ for the i-step in Example 1 is written by the `forall_t` keyword as follows:

```
(assert (and (forall_t 1 [0 time_i] (>= tau_i_t 0) (<= tau_i_t T))
             (forall_t 2 [0 time_i] (>= tau_i_t 0) (<= tau_i_t T))))
```

Bounded Model Checking. A safety requirement φ of a hybrid automaton H is expressed using t-variables of the final step, and the reachability goal is given by its negation $\neg\varphi$. For Example 1, the requirement is that the water level x lies between $L_{\min} - \epsilon$ and $L_{\max} + \epsilon$ with a limit $\epsilon > 0$ (that is, $L_{\min} - \epsilon \le x_N^t \le L_{\max} + \epsilon$). The reachability goal is given by its negation:

```
(assert (or (< x_N_t (- L_min ε)) (> x_N_t (+ L_max ε))))
```

The entire formula for bounded model checking with depth N consists of: (i) variable and flow declarations, (ii) initial condition formulas, (iii) flow, jump, and invariant condition formulas for $i = 0, \ldots, N$, and (iv) reachability goal formulas. It begins with the command `(set-logic QF_NRA_ODE)`, and ends with the commands `(check-sat)` `(exit)`. The bound N is iteratively increased from 0, and the verification is performed for every intermediate step.

If the SMT formula is satisfied, then we have a counterexample satisfying the reachability goal, and the safety requirement is violated. Otherwise, the safety requirement is satisfied since there is no counterexample. Because dReal uses δ-complete decision procedures, the result is over-approximated by δ. A counterexample of φ can violate φ with numerical perturbation up to $\delta > 0$ (i.e., a counterexample may be spurious). However, if no counterexample is found, then indeed there exists no counterexample regardless δ.

4 SMT Encoding of Compositions

A parallel composition of hybrid automata can be considered as a single hybrid automaton as mentioned in Section 2, and its bounded model checking problems can be encoded as SMT formulas in the exactly same way. However, the size of the formula can be very big. Consider

a parallel composition $H_1 \parallel \cdots \parallel H_n$ of n hybrid automata $H_1, \ldots, H_n$. If each automaton H_i has k_i modes, then the size of the formula for N-step bounded model checking in the standard encoding is $O(N \cdot \prod_1^n k_i)$. This *formula explosion problem* can make SMT-based analysis of networks of hybrid systems practically infeasible.

One of the reasons is that the `integral` command only accepts *complete* ODE systems. In a parallel composition, each mode of a single automaton corresponds to a (partial) ODE system (where ODEs of some variables can be given in other automata), and a composited mode corresponds to a complete ODE system composed of those partial ODE systems.[2] However, flow conditions cannot be decomposed in general, since variables in ODEs evolve simultaneously over continuous time (on the other hand, jump conditions of parallel compositions can be easily written in a compositional way according to the definition). For this reason, existing SMT techniques use the standard non-compositional encoding for networked hybrid automata.

We have recently introduced two new commends `pintegral` and `connect` in dReal to allow compositional encoding of networks of hybrid systems (and developed an SMT algorithm for the new syntax that will be presented elsewhere). Instead of declaring complete systems of ODEs as Section 3, we only declare several *partial* systems of ODEs by using `define-ode` statements for parallel compositions of hybrid automata. For example, partial ODE systems for two connected water tanks in Example 2 are declared as the following five partial flow decorations:

```
(define-ode flow_1
  ((= d/dt[tau] 1)))
(define-ode flow_2
  ((= d/dt[x1] (/ (- q1 (* (* a (sqrt (* 2 g))) (sqrt x1))) A1))))
(define-ode flow_3
  ((= d/dt[x1] (/ (* (* -a (sqrt (* 2 g))) (sqrt x1)) A1))))
(define-ode flow_4
  ((= d/dt[x2] (/ (+ q2 (* (* a (sqrt (* 2 g))) (- (sqrt x1) (sqrt x2)))) A2))))
(define-ode flow_5
  ((= d/dt[x2] (/ (* (* a (sqrt (* 2 g))) (- (sqrt x1) (sqrt x2))) A2))))
```

Then *parameterized* integral terms $\vec{y}_t = \vec{y}_0 + \int_0^t [h_1(t), \ldots, h_k(t)]\,\mathrm{d}t$ over *flow parameters* $h_1, \ldots, h_k$ are defined by using the keyword `pintegral` with the syntax:

```
(= [y_0_t...y_n_t] (pintegral 0 t [y_0_0...y_n_0] [holder_1...holder_k]))
```

A concrete partial flow $flow_l$ is assigned to a flow parameter $holder_j$ using the keyword `connect` with the syntax `(connect holder_`j `flow_`l`)`. Notice that "complete" assignments to such flow parameters $h_1, \ldots, h_k$ are supposed to give *complete* systems of ODEs.[3]

For example, the i-step flow condition of the two connected water tanks in Example 2 includes only one parameterized integral term over three flow parameters $holder_{a_i}$ for the timer, $holder_{b_i}$ for tank 1, and $holder_{c_i}$ for tank 2 (where $a_i = 3i + 1$, $b_i = 3i + 2$, and $c_i = 3i + 3$):

```
(assert (= [x1_i_t x2_i_t tau_0_t]
   (pintegral 0. time_0 [x1_i_0 x2_i_0 tau_i_0] [holder_a_i holder_b_i holder_c_i])))
```

After that, we separately assign partial flows to those flow parameters according to current modes using the `connect` keyword, where `flow_1` is for the timer τ, `flow_2` and `flow_3` are for tank 1, and `flow_4` and `flow_5` are for tank 2:

[2] For example, in Example 2, the water level x_i of the i-th water tank depends on the water level x_{i-1} of its input water tank, but the flow condition for x_{i-1} is given in the automaton H_{i-1}, not in H_i.

[3] The `forall_t` command is not yet available for flows given by `pintegral` at the moment, but will be implemented soon.

```
(assert (or (and (= mode1_i 0) (connect holder_a_i flow_2))
            (and (= mode1_i 1) (connect holder_a_i flow_3))))
(assert (or (and (= mode2_i 0) (connect holder_b_i flow_4))
            (and (= mode2_i 1) (connect holder_b_i flow_5))))
(assert (connect holder_c_i flow_1))
```

Using this new encoding, the size of the formula for N-step bounded model checking can be $O(N \cdot \sum_1^n k_i)$ for a parallel composition $H_1 \parallel \cdots \parallel H_n$, when each automaton H_i has k_i modes (cf., $O(N \cdot \prod_1^n k_i)$ for the previous encoding). This can in turn greatly improve the performance of SMT-based analysis of networks of hybrid systems. For example, when we consider the two connected water tanks in Example 2, for bound $k = 3$, the new encoding can verify the system in 8 seconds, whereas the old encoding took 22688 seconds, provided that no other heuristics is applied (for details, see http://dreal.github.io/benchmarks/networks/water).

5 Concluding Remarks

We have illustrated SMT encoding for bounded model checking problems of hybrid systems in dReal. In this way, general hybrid systems involving nonlinear real functions and ordinary differential equations can be analyzed by dReal using δ-complete decision procedures. A number of benchmarks and various examples (including the networked water tank example) are available at our tool website http://dreal.github.io. The dReach tool [12] provides a front-end of dReal to automatically generate SMT formulas from hybrid automata specifications. Since dReach currently only supports single hybrid automata, we plan to extend dReach to explicitly support networks of hybrid automata with compositional SMT encoding.

References

[1] CAPD: Computer assisted proofs in dynamical systems. http://capd.ii.uj.edu.pl/index.php.

[2] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In *Proc. SMT*, 2010.

[3] R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich. The OpenSMT solver. In *TACAS*, volume 6015 of *LNCS*. Springer, 2010.

[4] A. Cimatti, S. Mover, and S. Tonetta. SMT-based verification of hybrid systems. In *Proc. AAAI*, 2012.

[5] A. Eggers, M. Fränzle, and C. Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In *Proc. ATVA*, 2008.

[6] M. Fränzle and C. Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[7] S. Gao, J. Avigad, and E. M. Clarke. δ-complete decision procedures for satisfiability over the reals. In *IJCAR*, volume 7364 of *LNCS*, pages 286–300. Springer, 2012.

[8] S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *CADE*, volume 7898 of *LNCS*, pages 208–214. Springer, 2013.

[9] S. Gao, S. Kong, and E. M. Clarke. Satisfiability modulo ODEs. In *FMCAD*, pages 105–112. IEEE, 2013.

[10] L. Granvilliers and F. Benhamou. Algorithm 852: RealPaver: an interval solver using constraint satisfaction techniques. *ACM Trans. Math. Softw.*, 32(1):138–156, 2006.

[11] T. A. Henzinger. *The theory of hybrid automata*. Springer, 2000.

[12] S. Kong, S. Gao, W. Chen, and E. M. Clarke. dReach: δ-reachability analysis for hybrid systems. In *TACAS*, volume 9035 of *LNCS*. Springer, 2015.

[13] S. Kowalewski, O. Stursberg, M. Fritz, H. Graf, I. Hoffmann, J. Preußig, M. Remelhe, S. Simon, and H. Treseler. A case study in tool-aided analysis of discretely controlled continuous systems: the two tanks problem. In *Hybrid Systems V*, pages 163–185. Springer, 1999.

[14] J. Raisch, E. Klein, C. Meder, A. Itigin, and S. OâĂŹYoung. Approximating automata and discrete control for continuous systems âĂŤ two examples from process control. In *Hybrid systems V*, pages 279–303. Springer, 1999.