Satisfiability Modulo ODEs

Sicun Gao
Computer Science Department
Carnegie Mellon University
Pittsburgh, USA
Email: sicung@cs.cmu.edu

Soonho Kong
Computer Science Department
Carnegie Mellon University
Pittsburgh, USA
Email: soonhok@cs.cmu.edu

Edmund M. Clarke
Computer Science Department
Carnegie Mellon University
Pittsburgh, USA
Email: emc@cs.cmu.edu

Abstract—We study SMT problems over the reals containing ordinary differential equations,. They are important for formal verification of realistic hybrid systems and embedded software. We develop δ-complete algorithms for SMT formulas that are purely existentially quantified, as well as $\exists\forall$-formulas whose universal quantification is restricted to the time variables. We demonstrate scalability of the algorithms, as implemented in our open-source solver **dReal**, on SMT benchmarks with several hundred nonlinear ODEs and variables.

I. INTRODUCTION

Hybrid systems tightly combine finite automata and continuous dynamics. In most cases, the continuous components are specified by ordinary differential equations (ODEs). Thus, formal verification of general hybrid systems requires reasoning about logic formulas over the reals that contain ODE constraints. This problem is considered very difficult and has not been investigated in the context of decision procedures until recently [7], [8], [16]. It is believed that current techniques are not powerful enough to handle formulas that arise from formal verification of realistic hybrid systems, which typically contain many nonlinear ODEs and other constraints.

Since the first-order theory over the reals with trigonometric functions is already undecidable, solving formulas with general ODEs seems inherently impossible. We have resolved much of this theoretical difficulty by proposing the study of δ-*complete* decision procedures for such formulas [10]. An algorithm is δ-complete for a set of SMT formulas, where δ is an arbitrary positive rational number, if it correctly decides whether a formula is unsatisfiable or δ-satisfiable. Here, a formula is δ-satisfiable if, under some δ-*perturbations*, a syntactic variant of the original formula is satisfiable [9]. We have shown that δ-complete decision procedures are suitable for various formal verification tasks [9], [10]. We have also proved that δ-complete decision procedures exist for SMT problems over the reals with Lipschitz-continuous ODEs. Such results serve as a theoretical foundation for developing practical decision procedures for the SMT problem.

In this paper we study practical δ-complete algorithms for SMT formulas over the reals with ODEs. We show that such algorithms can be made powerful enough to scale to realistic benchmark formulas with several hundred nonlinear ODEs.

We develop decision procedures for the problem following a standard DPLL(ICP) framework, which relies on constraint solving algorithms as studied in Interval Constraint Propagation (ICP) [2]. In this framework, for any ODE system we can consider its solution function $\vec{x}_t = \vec{f}(t, \vec{x}_0)$ as a constraint between the initial variables $\vec{x}_0$, time variable t, and the final state variables $\vec{x}_t$. We define pruning operators that take interval assignments on $\vec{x}_0$, t, and $\vec{x}_t$ as inputs, and output refined interval assignments on these variables. We formally prove that the proposed algorithms are δ-complete. Beyond standard SMT problems where all variables are existentially quantified, we also study $\exists\forall$-formulas under the restriction that the universal quantifications are limited to the time variables (we call them $\exists\forall^t$-formulas). Such formulas have been an obstacle in SMT-based verification of hybrid systems [4], [5].

In brief, this paper makes the following contributions:

• We formalize the SMT problem over the reals with general Lipschitz-contiunous ODEs, and illustrate its expressiveness by encoding various standard problems concerning ODEs: initial and boundary value problems, parameter synthesis problems, differential algebraic equations, and bounded model checking of hybrid systems. In some cases, $\exists\forall^t$-formulas are needed.

• We propose algorithms for solving SMT with ODEs, using ODE constraints to design pruning operators in a branch-and-prune framework. We handle both standard SMT problems with only existentially quantified variables, as well as $\exists\forall^t$-formulas. We prove that the algorithms are δ-complete.

• We demonstrate the scalability of the algorithms, as implemented in our open-source solver **dReal** [11], on realistic benchmarks encoding formal verification problems for several nonlinear hybrid systems.

Related Work. Solving real constraints with ODEs has a wide range of applications, and much previous work exists for classes with special structures in different paradigms [6], [13], [18]. Recently [12] proposed a more general constraint solving framework, focusing on the formulation of the problem in the standard CP framework. On the SMT solving side, several authors have considered logical combinations of ODE constraints and proposed partial decision procedures [7], [8], [16]. We aim to extend and formalize existing algorithms for a general SMT theory with ODES, and formally prove that they can be made δ-complete. In terms of practical performance, the proposed algorithms are made scalable to various benchmarks that contain hundreds of nonlinear ODEs and variables.

The paper is organized as follows. In Section II, we define the SMT problem with ODEs and show how it can encode various standard problems with ODEs. In Section III, we propose algorithms in the DPLL(ICP) framework for solving fully existentially quantified formulas as well as $\exists\forall^t$ formulas. In Section IV we show experimental results.

II. SMT OVER THE REALS WITH ODEs

A. A First-Order Signature with Computable Real Functions

As studied in Computable Analysis [19], [17], we can encode real numbers as infinite strings, and develop a computability theory of real functions using Turing machines that perform operations using oracles encoding real numbers. We briefly review definitions and results of importance to us. Throughout the paper we use $|| \cdot ||$ to denote the max norm $|| \cdot ||_\infty$ over $\mathbb{R}^n$ for various n. First, a *name* of a real number is a sequence of rational numbers converging to it:

Definition 1 (Names). *A name of $a \in \mathbb{R}$ is any function $\gamma_a :$ $\mathbb{N} \to \mathbb{Q}$ that satisfies: for any $i \in \mathbb{N}$, $|\gamma_a(i) - a| < 2^{-i}$. For $\vec{a} \in \mathbb{R}^n$, $\gamma_{\vec{a}}(i) = \langle \gamma_{a_1}(i), ..., \gamma_{a_n}(i) \rangle$. We write the set of all possible names for $\vec{a}$ as $\Gamma(\vec{a})$.*

Next, a real function f is *computable* if there is a Turing machine that can use any argument x of f as an oracle, and compute the value of $f(x)$ up to an arbitrary precision 2^{-i}, where $i \in \mathbb{N}$. Formally:

Definition 2 (Computable Functions). *We say $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ is computable if there exists an oracle Turing machine $\mathcal{M}_f$ such that for any $\vec{x} \in dom(f)$, any name $\gamma_{\vec{x}}$ of $\vec{x}$, and any $i \in \mathbb{N}$, the machine uses $\gamma_{\vec{x}}$ as an oracle and i as an input to compute a rational number $M_f^{\gamma_{\vec{x}}}(i)$ satisfying $|M_f^{\gamma_{\vec{x}}}(i) - f(\vec{x})| < 2^{-i}$.*

The definition requires that for any $\vec{x} \in dom(f)$, with access to an arbitrary oracle encoding the name $\gamma_{\vec{x}}$ of $\vec{x}$, M_f outputs a 2^{-i}-approximation of $f(\vec{x})$. In other words, the sequence $M_f^{\gamma_{\vec{x}}}(1), M_f^{\gamma_{\vec{x}}}(2), ...$ is a name of $f(\vec{x})$. Intuitively, f is computable if an arbitrarily good approximation of $f(\vec{x})$ can be obtained using any good enough approximation to any $\vec{x} \in dom(f)$. Most common continuous real functions are computable [19]. Addition, multiplication, absolute value, min, max, exp, sin. Compositions of computable functions are computable. In particular, solution functions of Lipschitz-continuous ordinary differential equations are computable, as we explain next.

B. Solution Functions of ODEs

We now show that the framework of computable functions allows us to consider solution functions of ODE systems.

Notation 3. *We use $\vec{x} = \vec{y}$ between n-dimensional vectors to denote the system of equations $x_i = y_i$ for $1 \leq i \leq n$.*

Let $D \subseteq \mathbb{R}^n$ be compact and $g_i : D \to \mathbb{R}$ be n Lipschitz-continuous functions, which means that for some constant $c_i \in \mathbb{R}^+$ ($1 \leq i \leq n$), for all $\vec{x}_1, \vec{x}_2 \in D$,

$$|g_i(\vec{x}_1) - g_i(\vec{x}_2)| \leq c_i ||\vec{x}_1 - \vec{x}_2||.$$

Let t be a variable over $\mathbb{R}$. We consider the first-order autonomous ODE system

$$\frac{d\vec{y}}{dt} = \vec{g}(\vec{y}(t, \vec{x}_0)) \text{ and } \vec{y}(0, \vec{x}_0) = \vec{x}_0 \qquad (1)$$

where $\vec{x}_0 \in D$. Here, each

$$y_i : \mathbb{R} \times D \to \mathbb{R} \qquad (2)$$

is called the i-th solution function of the ODE system (1). A key result in computable analysis is that these solution functions are computable, in the sense of Definition 2:

Proposition 4 ([17]). *The solution functions $\vec{y}$ in the form of (2) of the ODE system (1) are computable over $\mathbb{R} \times D$.*

To see why this is true, recall that for any $t \in \mathbb{R}$ and $\vec{x}_0 \in D$, the value of the solution function follows the Picard-Lindelöf form:

$$\vec{y}(t, \vec{x}_0) = \int_0^t \vec{g}(\vec{y}(s, \vec{x}_0))ds + \vec{x}_0.$$

Approximations of the right-hand side of the equation can be computed by finite sums, theoretically up to an arbitrary precision.

C. SMT Problems and δ-Complete Decision Procedures

We now let $\mathcal{F}$ denote an arbitrary collection of computable real functions, which can naturally contain solution functions of ODE systems in the form of (2). Let $\mathcal{L}_\mathcal{F}$ denote the first-order signature $\langle \mathcal{F}, < \rangle$, where constants are seen as 0-ary functions in $\mathcal{F}$. Let $\mathbb{R}_\mathcal{F}$ be the structure $\langle \mathbb{R}, \mathcal{F}^\mathbb{R}, <^\mathbb{R} \rangle$ that interprets $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas in the standard way. We focus on formulas whose variables take values from bounded domains, which can be defined using bounded quantifiers:

Definition 5 (Bounded Quantifiers). *The bounded quantifiers $\exists^{[u,v]}$ and $\forall^{[u,v]}$ are defined as*

$$\exists^{[u,v]}x.\varphi =_{df} \exists x.(u \leq x \wedge x \leq v \wedge \varphi),$$
$$\forall^{[u,v]}x.\varphi =_{df} \forall x.((u \leq x \wedge x \leq v) \to \varphi),$$

where u and v denote $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ terms, whose variables only contain free variables in φ excluding x. It is easy to check that $\exists^{[u,v]}x.\varphi \leftrightarrow \neg\forall^{[u,v]}x.\neg\varphi$.

The key definition in our framework is δ-*variants* of first-order formulas:

Definition 6 (δ-Variants). *Let $\delta \in \mathbb{Q}^+ \cup \{0\}$, and φ a bounded $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-sentence of the standard form*

$$\varphi : \ Q_1^{I_1}x_1 \cdots Q_n^{I_n}x_n \ \psi[t_i(\vec{x}) > 0; t_j(\vec{x}) \geq 0],$$

where $i \in \{1, ...k\}$ and $j \in \{k+1, ..., m\}$. Note that negations are represented by sign changes on the terms. The δ-weakening φ^δ of φ is defined as the result of replacing each atom $t_i > 0$ by $t_i > -\delta$ and $t_j \geq 0$ by $t_j \geq -\delta$. That is,

$$\varphi^{-\delta} : \ Q_1^{I_1}x_1 \cdots Q_n^{I_n}x_n \ \psi[t_i(\vec{x}) > -\delta; t_j(\vec{x}) \geq -\delta].$$

The SMT problem is standardly defined as deciding satisfiability of quantifier-free formulas, which is equivalent to deciding the truth value of fully existentially quantified sentences. We will also consider formulas that are partially universally quantified. Thus, we consider both Σ_1 and Σ_2 formulas here.

Definition 7 (Bounded Σ_1- and Σ_2-SMT Problems). *A Σ_1-SMT problem is a formula of the form*

$$\exists^{I_1}x_1 \cdots \exists^{I_n}x_n.\varphi(\vec{x})$$

and a Σ_2-SMT problem is of the form

$$\exists^{I_1}x_1 \cdots \exists^{I_n}x_n \forall^{I_{n+1}}x_{n+1} \cdots \forall^{I_m}x_m.\varphi(\vec{x}).$$

In both cases $\varphi(\vec{x})$ is a quantifier-free $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-formula.

Definition 8 (δ-Completeness [9]). *Let S be a set of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$ formulas, and $\delta \in \mathbb{Q}^+$. We say a decision procedure A is δ-complete for S, if for any $\varphi \in S$, A correctly returns one of the following answers*

- *φ is false;*
- *$\varphi^{-\delta}$ is true.*

If the two cases overlap, either one is correct.

We have proved in [10] that δ-complete decision procedures exists for arbitrary bounded $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-sentences. In particular, there exists δ-complete decision procedures for the bounded Σ_1 and Σ_2 SMT problems. This serves as the theoretical foundation as well as a correctness requirement for the practical algorithms that we will develop in the following sections.

D. SMT Encoding of Standard Problems with ODEs

In this section, we list several standard problems related to ODE systems and show that they can be easily encoded and generalized through SMT formulas. They motivate the development of decision procedures for the theory.

Remark 9. *In all the following cases, solutions to the standard problems are obtained from witnesses for the existentially quantified variables in the SMT formulas.*

Remark 10. *In the definitions below, when the solution functions $\vec{y}$ of ODE systems are written as part of a formula, no analytic forms are needed. They are functions included in the signature $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$.*

Generalized Initial Value Problems. Given an ODE system, the standard initial value problem asks for a solution of the variables at certain time, given a complete assignment to the initial conditions of the system. In the form of SMT formulas, we easily allow the initial conditions to be constrained by arbitrary quantifier-free $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-formulas:

Definition 11 (Generalized IVP). *Let $X \subseteq \mathbb{R}^n$ be a compact domain, $T \in \mathbb{R}^+$, and $\vec{y} : [0,T] \times X \to X$ be the computable solution functions of an ODE system. Let $t \in [0,T]$ be an arbitrary constant that represents a time point of interest. The generalized IVP problem is defined by formulas of the form:*

$$\exists^X x_0 \exists^X \vec{x}. \ \varphi(\vec{x}_0) \wedge \vec{x} = \vec{y}(t, \vec{x}_0),$$

where φ is a quantifier-free $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-formula constraining the initial states $\vec{x}_0$, and $\vec{x}$ is the needed value for time point t.

Generalized Boundary Value Problems. Given an ODE system, the standard boundary value problem is concerned with computing the computable solution function when part of the variables are assigned values at the beginning of flow, and part of the variables as assigned values at the end of the flow. A generalized version as encoded by SMT formulas is:

Definition 12 (Generalized BVP). *Let $X \subseteq \mathbb{R}^n$ be a compact domain, $T \in \mathbb{R}^+$, and $\vec{y} : [0,T] \times X \to X$ be the solution functions of an ODE system. Let $t, t' \in [0,T]$ be two time points of interest. The generalized BVP problem is:*

$$\exists^X x_0 \exists^X \vec{x}_t \exists^X \vec{x}. \varphi(\vec{x}_0, \vec{x}_t, t) \wedge \vec{x}_t = \vec{y}(t, \vec{x}_0) \wedge \vec{x} = \vec{y}(t', \vec{x}_0)$$

where φ is a quantifier-free $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-formula that specifies the boundary conditions. Note that $\vec{x}$ is the value that we are interested in solving in the chosen time point t'.

Data-Fitting and Parameter Synthesis. The data fitting problem is the following. Suppose an ODE system has part of its parameters unspecified. Given a sequence of data $(t_1, \vec{a}_1), ..., (t_k, \vec{a}_k)$, we need to find the values of the missing parameters of the original ODE system. More formally:

Definition 13 (Data-Fitting Problems). *Let $X \subseteq \mathbb{R}^n$ and $P \subseteq \mathbb{R}^m$ be compact domains, $T \in \mathbb{R}^+$, and $\vec{y}(\vec{p}) : [0,T] \times X \to X$ be the solution functions of an ODE system, where $\vec{p} \in P$ be a vector of parameters. Let $(t_1, \vec{a}_1), ..., (t_k, \vec{a}_k)$ be a sequence of pairs in $[0,T] \times X$. The data-fitting problem is defined by:*

$$\exists^P \vec{p} \exists^X x_0. \ \varphi(\vec{x}_0) \wedge \vec{a}_1 = \vec{y}(\vec{p}, t_1, \vec{x}_0) \wedge \cdots \wedge \vec{a}_k = \vec{y}(\vec{p}, t_k, \vec{x}_0),$$

where a quantifier-free φ constraints the initial states $\vec{x}_0$.

Differential Algebraic Equations. DAE problems combine ODEs and algebraic constraints:

$$\frac{d\vec{y}}{dt} = \vec{g}(\vec{y}(t, \vec{y}_0), \vec{z}) \tag{3}$$

$$0 = \vec{h}(\vec{y}, \vec{z}, t) \tag{4}$$

where $\vec{y}, \vec{y}_0 \in \mathbb{R}^n$, $\vec{z} \in \mathbb{R}^m$. To express the problem in $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$, we need to use extra universal quantification to ensure that the algebraic relations hold throughout the time duration. Again, we can also generalize the equation in (4) to an arbitrary quantifier-free $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$-formula. The problem is encoded as:

Definition 14 (DAE Problems). *Let $X \subseteq \mathbb{R}^n$ be a compact domain, $T \in \mathbb{R}^+$, and $\vec{y} : [0,T] \times X \times X \to X$ be the computable solution functions of the ODE system in (3) parameterized by $\vec{z}$. Let h be defined by (4). Let $t \in [0,T]$ be a time point of interest. A DAE problem is defined by the following formula:*

$$\exists^X \vec{x}_0 \exists^X \vec{x} \exists^Z \vec{z} \forall^{[0,t]} t'.$$
$$\varphi(\vec{x}_0) \wedge \vec{x} = \vec{y}(t, \vec{x}_0, \vec{z}) \wedge h(\vec{y}(\vec{x}_0, t'), \vec{z}, t') = 0$$

where a quantifier-free φ specifies the initial conditions for $\vec{y}$, and $\vec{x}$ is the needed value at time point t.

Bounded Model Checking of Hybrid Systems. Bounded model checking problems for hybrid systems can be naturally encoded as SMT formulas with ODEs [7], [8], [16], [4], [5]. We consider a simple hybrid system to show an example. Let H be an n-dimensional 2-mode hybrid system. In mode 1, the flow of the system follows an ODE system whose solution function is $\vec{y}_1(t, \vec{x}_0)$, and in mode 2, it follows another solution function $\vec{y}_2(t, \vec{x}_0)$. The jump condition from mode 1 to mode 2 is specified by $\mathsf{jump}(\vec{x}, \vec{x}')$. The invariants are specified by $\mathsf{inv}_i(\vec{x})$ and for mode i. Let $\mathsf{unsafe}(\vec{x})$ denote an unsafe region. Let the continuous variables be bounded in X and time be bounded in $[0,T]$. Now, if H starts from mode 1 with initial states satisfying $\mathsf{init}(\vec{x})$, it can reach the unsafe region after one discrete jump from mode 1 to mode 2, iff the following formula is true:

$$\exists^X \vec{x}_1 \exists^X \vec{x}_1^t \ \exists^X \vec{x}_2 \exists^X \vec{x}_2^t \ \exists^{[0,T]} t_1 \exists^{[0,T]} t_2 \ \forall^{[0,t_1]} t_1' \forall^{[0,t_2]} t_2'.$$
$$\mathsf{init}(\vec{x}_1) \wedge \vec{x}_1^t = \vec{y}_1(t_1, \vec{x}_1) \wedge \mathsf{inv}_1(\vec{y}_1(t_1', \vec{x}_1)) \wedge \mathsf{jump}(\vec{x}_1^t, \vec{x}_2)$$
$$\wedge \vec{x}_2^t = \vec{y}_2(t_2, \vec{x}_2) \wedge \mathsf{inv}_2(\vec{y}_2(t_2', \vec{x}_2)) \wedge \mathsf{unsafe}(\vec{x}_2^t).$$

The encoding can be explained as follows. For each mode, we use two variable vectors $\vec{x}_i$ and $\vec{x}_i^t$ to represent the continous flows. $\vec{x}_i$ denote the starting values of a flow, and $\vec{x}_i^t$ denotes the final values. In mode 1, the flow starts with some values in the initial states, specified by $\mathsf{init}(\vec{x}_1)$. Then, we follow the continuous dynamics in mode 1, so that $\vec{x}_1^t$ denotes the final value $\vec{x}_1^t = \vec{y}(t_1, \vec{x}_1)$. Then the system follows the jumping condition and resets the variables from $\vec{x}_1^t$ to $\vec{x}_2$ as specified by $\mathsf{jump}(\vec{x}_1^t, \vec{x}_2)$. After that, the system follows the flow in mode 2. In the end, we check if the final state $\vec{x}_2^t$ in mode 2 satisfies the unsafe predicate, $\mathsf{unsafe}(\vec{x}_2)$.

III. ALGORITHMS

A. The ICP framework

The method of Interval Constraint Propagation (ICP) [2] finds solutions of real constraints using a "branch-and-prune" method that performs constraint propagation of interval assignments on real variables. The intervals are represented by floating-point end-points. Only over-approximations of the function values are used, which are defined by interval extensions of real functions.

Definition 15 (Floating-Point Intervals and Hulls)**.** *Let* $\mathbb{F}$ *denote the finite set of all floating point numbers with symbols* $-\infty$ *and* $+\infty$ *under the conventional order* $<$. *Let*

$$\mathbb{IF} = \{[a,b] \subseteq \mathbb{R} : a, b \in \mathbb{F}, a \le b\} \text{ and } \mathbb{BF} = \bigcup_{n=1}^{\infty} \mathbb{IF}^n$$

denote the set of closed real intervals with floating-point endpoints, and the set of boxes *with these intervals, respectively. When* $S \subseteq \mathbb{R}^n$ *is a set of real numbers, the hull of* S *is:*

$$\mathrm{Hull}(S) = \bigcap\{B \in \mathbb{BF} : S \subseteq B\}.$$

Definition 16 (Interval Extension [2])**.** *Suppose* $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ *is a real function. An interval extension operator* $\sharp(\cdot)$ *maps* f *to a function* $\sharp f :\subseteq \mathbb{BF} \to \mathbb{IF}$, *such that for any* $B \in dom(\sharp f)$, *it is always true that* $\{f(\vec{x}) : \vec{x} \in B\} \subseteq \sharp f(B)$.

Algorithm 1 $\mathrm{ICP}(f_1, ..., f_m, B_0 = I_1^0 \times \cdots \times I_n^0, \delta)$

1: $S \leftarrow B_0$
2: **while** $S \ne \emptyset$ **do**
3:      $B \leftarrow S.\mathrm{pop}()$
4:      **while** $\exists 1 \le i \le m, B \ne_\delta \mathrm{Prune}(B, f_i)$ **do**
5:          $B \leftarrow \mathrm{Prune}(B, f_i)$
6:      **end while**
7:      **if** $B \ne \emptyset$ **then**
8:          **if** $\exists 1 \le i \le n, |\sharp f_i(B)| \ge \delta$ **then**
9:              $\{B_1, B_2\} \leftarrow \mathrm{Branch}(B, i)$
10:             $S.\mathrm{push}(\{B_1, B_2\})$
11:          **else**
12:             **return** sat
13:          **end if**
14:      **end if**
15: **end while**
16: **return** unsat

ICP uses interval extensions of functions to "prune" out sets of points that are not in the solution set, and "branch"

on intervals when such pruning can not be done, until a small enough box that may contain a solution is found. A high-level description of the decision version of ICP is given in Algorithm 1. In Algorithm 1, $\mathrm{Branch}(B, i)$ is an operator that returns two smaller boxes $B' = I_1 \times \cdots \times I_i' \times \cdots \times I_n$ and $B'' = I_1 \times \cdots \times I_i'' \times \cdots \times I_n$, where $I_i \subseteq I_i' \cup I_i''$. The key component of the algorithm is the $\mathrm{Prune}(B, f)$ operation. Any operation that contracts the intervals on variables can be seen as pruning, but for correctness we need formal requirements on the pruning operator in ICP. Basically, we need to require that the interval extensions of the functions converge to the true values of the functions, and that the pruning operations are well-defined, as specified below.

Definition 17 (δ-Regular Interval Extensions)**.** *We say an interval extension* $\sharp f$ *of* $f : \mathbb{R}^n \to \mathbb{R}$ *is* δ-regular, *if for some constant* $c \in \mathbb{R}$, *for any* $B \in \mathbb{R}^n$, $|\sharp f(B)| \le \max(c||B||, \delta)$.

Definition 18 (Well-defined Pruning Operators [9])**.** *Let* $\mathcal{F}$ *be a collection of real functions, and* $\sharp$ *be a* δ-regular interval *extension operator on* $\mathcal{F}$. *A well-defined (equality) pruning operator with respect to* $\sharp$ *is a partial function* $\mathrm{Prune}_\sharp :\subseteq \mathbb{BF} \times \mathcal{F} \to \mathbb{BF}$, *such that for any* $f \in \mathcal{F}$, $B, B' \in \mathbb{BF}$,

1) $\mathrm{Prune}_\sharp(B, f) \subseteq B$;
2)   *If* $\mathrm{Prune}_\sharp(B, f) \ne \emptyset$, *then* $0 \in \sharp f(\mathrm{Prune}_\sharp(B, f))$;
3) $B \cap \{\vec{a} \in \mathbb{R}^n : f(\vec{a}) = 0\} \subseteq \mathrm{Prune}_\sharp(B, f)$.

When $\sharp$ is clear, we simply write Prune. The rules can be explained as follows. (W1) ensures that the algorithm always makes progress. (W2) ensures that the result of a pruning is always a reasonable box that may contain a zero, and otherwise B is pruned out. (W3) ensures that the real solutions are never discarded. We proved the following theorem in [9]:

Theorem 19. *Algorithm 1 is* δ-complete *if the pruning operators are well-defined.*

B. ODE Pruning in an ICP Framework

We now study the algorithms for SMT formulas with ODEs. The key is to design the appropriate pruning operators for the solution functions of ODE systems. The pruning operations here strengthen and formalize the ones proposed in [7], [8], [12], such that δ-completeness can be proved.

We recall some notations first. Let $D \subseteq \mathbb{R}^n$ be compact and $g_i : D \to D$ be n Lipschitz-continuous functions. Given the first-order autonomous ODE system

$$\frac{d\vec{y}}{dt} = \vec{g}(\vec{y}(t, \vec{x}_0)) \text{ and } \vec{y}(0, \vec{x}_0) = \vec{x}_0 \quad (5)$$

where $\vec{x}_0 \in D$, we write

$$y_i : [0, T] \times D \to D_i$$

to represent the i-th solution function of the ODE system. The δ-regular interval extension of y_i is an interval function

$$\sharp y_i : (\mathbb{IF} \cap [0, T]) \times (\mathbb{BF} \cap D) \to \mathbb{IF}$$

such that for a constant $c \in \mathbb{R}$, for any time domain $I_t \subseteq \mathbb{IF} \cap [0, T]$ and any box of initial values $B_{\vec{x}_0} \subseteq \mathbb{BF} \cap D$, we have

$$\{x_t \in \mathbb{R} : x_t = y_i(t, \vec{x}_0), \vec{x}_0 \in B_{\vec{x}_0}, t \in I_t\} \subseteq \sharp y_i(I_t, B_{\vec{x}_0})$$

and

$$|\sharp y_i(I_t, B_{\vec{x}_0})| \leq \max(c \cdot ||I_t \times B_{\vec{x}_0}||, \delta).$$

We will also need the notion of the *reverse* of the ODE system (5), as defined by

$$\frac{d\vec{y}_-}{dt} = \vec{g}_-(\vec{y}_-(t, \vec{x}_t)) \text{ and } \vec{y}(0, \vec{x}_t) = \vec{x}_t. \quad (6)$$

Here, $\vec{g}_-$ is defined as $-\vec{g}$, the vector of functions consisting of the negation of each function in $\vec{g}$, which is equivalent to reversing time in the flow defined by the ODE system. That is, for $\vec{x}_0, \vec{x}_t \in D, t \in \mathbb{R}$, we always have

$$\vec{x}_t = \vec{y}(t, \vec{x}_0) \text{ iff } \vec{x}_0 = \vec{y}_-(t, \vec{x}_t). \quad (7)$$

Naturally, we write $\sharp(y_-)_i$ to denote the δ-regular interval extension of the i-th component of $\vec{y}_-$.

Algorithm 2 ODEPruning($\sharp\vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t$)

1: **repeat**
2:     $B'_{\vec{x}_t} \leftarrow \text{Prune}_{\text{fwd}}(\sharp\vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t)$
3:     $I'_t \leftarrow \text{Prune}_{\text{time}}(\sharp\vec{y}, B_{\vec{x}_0}, B'_{\vec{x}_t}, I_t)$
4:     $B'_{\vec{x}_0} \leftarrow \text{Prune}_{\text{bwd}}(\sharp\vec{y}, B_{\vec{x}_0}, B'_{\vec{x}_t}, I'_t)$
5: **until** $B_{\vec{x}_0} = B'_{\vec{x}_0} \wedge B_{\vec{x}_t} = B'_{\vec{x}_t} \wedge I_t = I'_t$
6: **return** $(B'_{\vec{x}_0}, B'_{\vec{x}_t}, I'_t)$

The relation between the initial variables $\vec{x}_0$, the time duration t, and the flow variables $\vec{x}_t$ is specified by the constraint $\vec{x}_t = \vec{y}(t, \vec{x}_0)$. Given the interval assignment on any two of $\vec{x}_0$, $\vec{x}_t$, and t, we can use the constraint to obtain a refined interval assignment to the third variable vector. Thus, we can define three pruning operators as follows.

Remark 20. *The precise definitions of the pruning operators should map the interval assignments on all variables to new assignments on all variables. For notational simplicity, in the pruning operators below we only list the assignments that are actually changed between inputs and outputs. For instance, the forward pruning operator only changes the values on $B_{\vec{x}_t}$.*

Forward Pruning. Given interval assignments on $\vec{x}_0$ and t, we compute a refinement of the interval assignments on $\vec{x}_t$. Figure 1 depicts the forward pruning operation. Formally, we define the following operator:

Definition 21 (Forward Pruning). *Let $\vec{y} : [0, T] \times D \to D$ be the solution functions of an ODE system. Let $B_{\vec{x}_0}$, $B_{\vec{x}_t}$, and I_t be interval assignments on the variables $\vec{x}_0$, $\vec{x}_t$, and t. We define the forward-pruning operator as:*

$$\text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y}) = \text{Hull}\left(B_{\vec{x}_t} \cap \sharp\vec{y}(I_t, B_{\vec{x}_0})\right).$$

Backward Pruning. Given interval assignments on $\vec{x}_t$ and t, we can compute a refinement of the interval assignments on $\vec{x}_0$ using the reverse of the solution function. Figure 2 depicts backward pruning. Formally, we define the following operator:

Definition 22 (Backward Pruning). *Let $\vec{y} : [0, T] \times D \to D$ be the solution functions of an ODE system, and let $\vec{y}_-$ be the reverse of $\vec{y}$. Let $B_{\vec{x}_0}$, $B_{\vec{x}_t}$, and I_t be interval assignments on the variables $\vec{x}_0$, $\vec{x}_t$, and t. We define the backward-pruning*
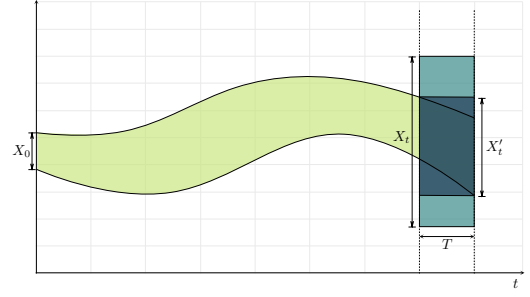

Fig. 1: Forward Pruning. X_0, X_t, t represents the current interval assignments, and X'_t is the refined interval assignment on $\vec{x}_t$ after pruning.

Algorithm 3 Prune$_{\text{fwd}}(\sharp\vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t)$

1: $B'_{\vec{x}_t} \leftarrow \phi$
2: $I_{\Delta t} \leftarrow [I_t^l, I_t^l + \varepsilon]$
3: **while** $I_{\Delta t}^u < I_t^u$ **do**
4:     $B'_{\vec{x}_t} \leftarrow \text{Hull}(B'_{\vec{x}_t} \cup \sharp\vec{y}(I_{\Delta t}, B_{\vec{x}_0}))$
5:     $I_{\Delta t} \leftarrow I_{\Delta t} + \varepsilon$
6: **end while**
7: **return** $B_{\vec{x}_t} \cap B'_{\vec{x}_t}$

---

operator as:

$$\text{Prune}_{\text{bwd}}(B_{\vec{x}_0}, \vec{y}) = \text{Hull}\left(B_{\vec{x}_0} \cap \sharp\vec{y}_-(I_t, B_{\vec{x}_t})\right).$$

Time-Domain Pruning. Given interval assignments on $\vec{x}_0$ and $\vec{x}_t$, we can also refine the interval assignment on t by pruning out the time intervals that do not contain any $\vec{x}_t$ that is consistent with the current interval assignments on $\vec{x}_t$. Figure 3 depicts time-domain pruning. Formally, we define the following operator:

Definition 23 (Time-Domain Pruning). *Let $\vec{y} : [0, T] \times D \to D$ be the solution functions of an ODE system. Let $B_{\vec{x}_0}$, $B_{\vec{x}_t}$, I_t be interval assignments on the variables $\vec{x}_0$, $\vec{x}_t$, and t. We define the time-domain pruning operator as:*

$$\text{Prune}_{\text{time}}(I_t, \vec{y}) = \text{Hull}\left(I_t \cap \{I : \sharp\vec{y}(I, B_{\vec{x}_0}) \cap B_{\vec{x}_t} \neq \emptyset\}\right).$$
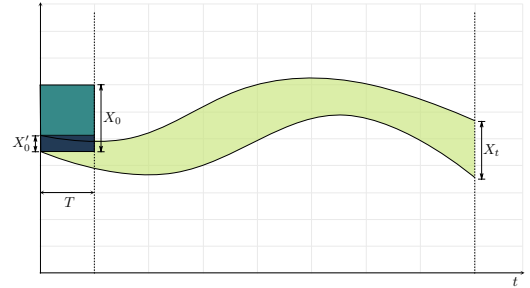

Fig. 2: Backward Pruning. X_0, X_t, t represents the current interval assignments, and X'_0 is the refined interval assignment on $\vec{x}_0$ after pruning.
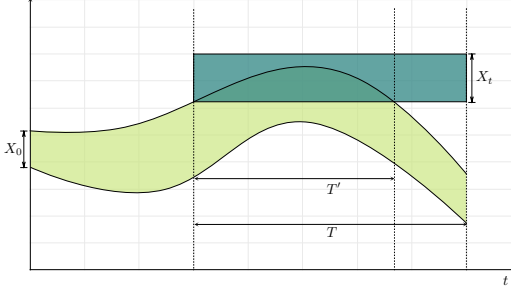
Fig. 3: Time-Domain Pruning. X_0, X_t, t represents the current interval assignments, and T' is the refined interval assignment on t after pruning.

Overall, the pruning algorithm on based on ODE constraints iteratively applies the three pruning operators until a fixed point on the interval assignments is reached.

Algorithm 4 $\text{Prune}_{\text{bwd}}(\sharp \vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t)$

1: $B'_{\vec{x}_0} \leftarrow \phi$
2: $I_{\Delta t} \leftarrow [I_t^l, I_t^l + \varepsilon]$
3: **while** $I_{\Delta t}^u < I_t^u$ **do**
4:      $B'_{\vec{x}_0} \leftarrow \text{Hull}(B'_{\vec{x}_0} \cup \sharp \vec{y}_-(I_{\Delta t}, B_{\vec{x}_t}))$
5:      $I_{\Delta t} \leftarrow I_{\Delta t} + \varepsilon$
6: **end while**
7: **return** $B_{\vec{x}_0} \cap B'_{\vec{x}_0}$

We show the more detailed steps in the three pruning operations in Algorithm 2, 3, 4, and 5.

Algorithm 5 $\text{Prune}_{\text{time}}(\sharp \vec{y}, B_{\vec{x}_0}, B_{\vec{x}_t}, I_t)$

1: $I'_t \leftarrow \phi$
2: $I_{\Delta t} \leftarrow [I_t^l, I_t^l + \varepsilon]$
3: **while** $I_{\Delta t}^u < I_t^u$ **do**
4:      $B'_{\vec{x}_t} \leftarrow \sharp \vec{y}(I_{\Delta t}, B_{\vec{x}_0})$
5:      **if** $B'_{\vec{x}_t} \cap B_{\vec{x}_t} \neq \phi$ **then**
6:          $I'_t = \text{Hull}(I'_t \cup I_{\Delta t})$
7:      **else**
8:          $I_{\Delta t} \leftarrow I_{\Delta t} + \varepsilon$
9:      **end if**
10: **end while**
11: **return** I'_t

Theorem 24. *The three pruning operators are well-defined.*

Proof: We prove that the forward pruning operator is well-defined, and the proofs for the other two operators are similar. Note that the definitions of well-defined pruning are formulated for equality constraints compared to 0. Here we use the function $f = \vec{y}(t, \vec{x}_0) - \vec{x}_t$ in the pruning operator. (Strictly speaking f is a function vector that evaluates to $\vec{0}$ on points satisfying the ODE flow. Here for notational simplicity we just write f as a single-valued function and compare with the scalar 0.)

First, (W1) is satisfied because of the simple fact that for any boxes $B_1, B_2 \in \mathbb{BF}$, we have $\text{Hull}(B_1 \cap B_2) \subseteq B_1$.

Next, suppose $0 \notin \sharp f(\text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y}) - B_{\vec{x}_t})$. Then there does not exist any $\vec{a}_t \in \mathbb{R}^n$ that satisfies both $\vec{a}_t \in B_{\vec{x}_t}$ and $\vec{a}_t \in \text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y})$. Since at the same time

$$\text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y}) = \text{Hull}\left(B_{\vec{x}_t} \cap \sharp \vec{y}(I_t, B_{\vec{x}_0})\right) \subseteq B_{\vec{x}_t},$$

this requires that $\text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y}) = \emptyset$. Consequently (W2) is satisfied.

Third, note that $\sharp \vec{y}(I_t, B_{\vec{x}_0})$ is an interval extension of $\vec{y}$. Thus, for any $\vec{a}_t \in \mathbb{R}^n$ such that $\vec{y}(t, \vec{x}_0)$ for some $t \in I_t$ and $\vec{x}_0 \in B_{\vec{x}_0}$, we have $\vec{a}_t \in \sharp \vec{y}(I_t, B_{\vec{x}_0})$. Following the definition of the pruning operator, we have $\vec{a}_t \in \text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, \vec{y})$. Thus, $B_{\vec{x}_t} \cap Z_f \subseteq \text{Prune}_{\text{fwd}}(B_{\vec{x}_t}, f)$ and (W3) holds. ∎

C. $\exists \forall^t$-Formulas and Low-Order Approximations

For $\exists \forall$-formulas, if the universal quantification is only over the time variables, we can follow the trajectory and prune away the assignment on $\vec{x}_0$, $\vec{x}_t$, and t that violates the constraints on the universally quantified time variable. In fact, although the extra quantification complicates the problem, the universal constraints improve the power of the pruning operations.

Here we focus on problems with one ODE system, which can be easily generalized. Let $\vec{y}$ denote the solution functions of an ODE system, we consider an $\exists \forall^t$-formula of the form

$$\exists^X \vec{x}_0 \exists^X \vec{x}_t \exists^{[0,T]} t \forall^{[0,t]} t'. \ \vec{x}_t = \vec{y}(t, \vec{x}_0) \wedge \varphi(\vec{y}(t', \vec{x}_0)) \qquad (8)$$

Note that the problems encoded as Σ_2-SMT formulas as listed in Section II-D are all of this form.

We consider $\varphi(\vec{y}(t', \vec{x}_0))$ as a special constraint on the $\vec{x}_0$ and t variables. Using this constraint, we can further refine the three pruning operators as follows.

Definition 25 (Pruning Refined by $\forall^t$-Constraints). *Let $\vec{y}$: $[0, T] \times D \rightarrow \mathbb{R}^n$ be the solution functions of an ODE system. Let $B_{\vec{x}_0}$, $B_{\vec{x}_t}$, and I_t be interval assignments on the variables $\vec{x}_0$, $\vec{x}_t$, and t. Let $\varphi(\vec{y}(t', \vec{x}_0))$ be a constraint on the universally quantified time variable, as in (8). We first define*

$$\sharp \varphi(I_t, B_{\vec{x}_0}) = \text{Hull}(\{\vec{a} \in \mathbb{R}^n : \vec{a} = \vec{y}(t, \vec{x}_0), t \in I_t, \\ \vec{x}_0 \in B_{\vec{x}_0}, \text{ and } \varphi(\vec{a}) \text{ is true.}\})$$

and define $\sharp \varphi_-$ by replacing $\vec{y}$ with $\vec{y}_-$ in the definition above. The forward pruning operator with φ, written as $\text{Prune}_{\text{fwd}}^\varphi(B_{\vec{x}_t}, \vec{y})$, is defined as

$$\text{Hull}\left(B_{\vec{x}_t} \cap \sharp \vec{y}(I_t, B_{\vec{x}_0}) \cap \sharp \varphi(I_t, B_{\vec{x}_0})\right)$$

Backward pruning $\text{Prune}_{\text{bwd}}^\varphi(B_{\vec{x}_0}, \vec{y})$ is defined as

$$\text{Hull}\left(B_{\vec{x}_0} \cap \sharp \vec{y}_-(I_t, B_{\vec{x}_t}) \cap \sharp \varphi_-(I_t, B_{\vec{x}_t})\right).$$

Time-domain pruning $\text{Prune}_{\text{time}}^\varphi(I_t, \vec{y})$ is defined as

$$\text{Hull}\left(I_t \cap \{I : \sharp \vec{y}(I, B_{\vec{x}_0}) \cap B_{\vec{x}_t} \cap \sharp \varphi(I_t, B_{\vec{x}_0}) \neq \emptyset\}\right).$$

In general, $\sharp \varphi$ can be computed by a recursive call to DPLL(ICP), by solving the Σ_1-SMT problem $\varphi(\vec{x})$. In many practical applications, φ is of some simple form such as $\vec{a} \leq \vec{x}_t \leq \vec{b}$, in which case simple pruning is shown in Figure 4. Another useful heuristic in ODE pruning is to bound the range
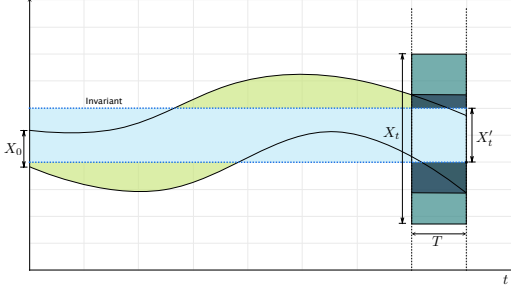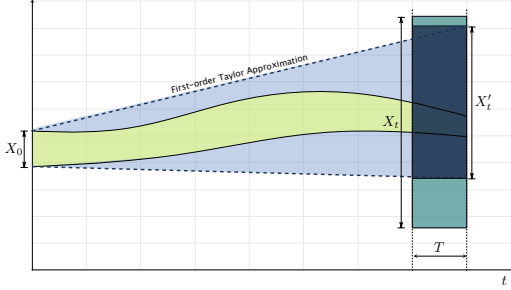
Fig. 4: Pruning with $\forall^t$-Constraints

Fig. 5: Pruning with Low-Order Taylor Approximations

P	#M	#D	#O	#V	delta	R	Time(s)	Trace
AF	4	3	20	44	0.001	S	43.10	90K
AF	8	7	40	88	0.001	S	698.86	20M
AF	8	23	120	246	0.001	S	4528.13	59M
AF	8	31	160	352	0.001	S	8485.99	78M
AF	8	47	240	528	0.001	S	15740.41	117M
AF	8	55	280	616	0.001	S	19989.59	137M
CT	2	2	15	36	0.005	S	345.84	3.1M
CT	2	2	15	36	0.002	S	362.84	3.1M
EO	3	2	18	42	0.01	S	52.93	998K
EO	3	2	18	42	0.001	S	57.67	847K
EO	3	11	72	168	0.01	U	7.75	–
BB	2	10	22	66	0.01	S	0.25	123K
BB	2	20	42	126	0.01	S	0.57	171K
BB	2	20	42	126	0.001	S	2.21	168K
BB	2	40	82	246	0.01	U	0.27	—–
BB	2	40	82	246	0.001	U	0.26	—–
D1	3	2	9	24	0.1	S	30.84	72K
DU	3	2	6	16	0.1	U	0.04	–

TABLE I: #M = Number of modes in the hybrid system, #D = Unrolling depth, #O = Number of ODEs in the unrolled formula, #V = Number of variables in the unrolled formula, R = Bounded Model Checking Result (delta-SAT/UNSAT) Time = CPU time (s), Trace = Size of the ODE trajectory, AF = Atrial Filbrilation, CT = Cancer Treatment, EO = Electronic Oscillator, BB = Bouncing Ball with Drag, D1,DU = Decay Model.

of the derivatives for a vector space specified by $\vec{g}$. Suppose for any time $t \in [0, T]$, the derivatives $\vec{g}$ are bounded in $[\vec{l_g}, \vec{u_g}]$. Then by the Picard-Lindelöf representation, we have

$$\vec{x}_t = \int_0^t \vec{g}(\vec{y}(s, \vec{y}_0))ds + \vec{y}_0 \in [0, T] \cdot [\vec{l_g}, \vec{u_g}] + B_{\vec{x}_0}$$

We can use this formula to perform preliminary pruning on $\vec{x}_t$, which is especially efficient when combined with $\forall^t$-constraints. Figure 5 illustrates this pruning method.

IV. Experiments

Our tool dReal implements the procedures we studied for solving SMT formulas with ODEs. It is built on several existing packages, including opensmt [3] for the general DPLL(T) framework, realpaver [14] for ICP, and CAPD [1] for computing interval-enclosures of ODEs. The tool is open-source at http://dreal.cs.cmu.edu. All benchmarks and data shown here are also available on the tool website.

All experiments were conducted on a machine with a 3.4GHz octa-core Intel Core i7-2600 processor and 16GB RAM, running 64-bit Ubuntu 12.04LTS. Table I is a summary of the running time of the tool on various SMT formulas generated from bounded model checking hybrid systems. The formulas typically contain a large number of variables and nonlinear ODEs.

The AF model as we show in Table I is obtained from [15]. It is a precise model of atrial fibrillation, a serious cardiac disorder. The continuous dynamics in the model concerns four
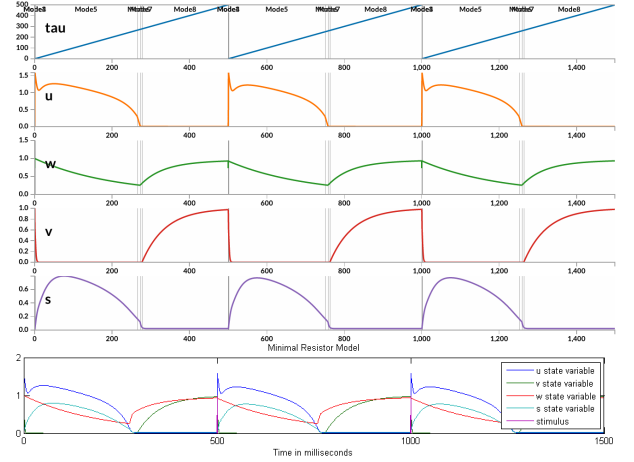

Fig. 6: Above: Witness for the AF model at depth 23 and 1500 time units. Below: Experimental simulation data.

state variables and the ODEs are highly nonlinear, such as:

$$\frac{du}{dt} = e + (u - \theta_v)(u_u - u)vg_{fi} + wsg_{si} - g_{so}(u)$$
$$\frac{ds}{dt} = \frac{g_{s2}}{(1 + e^{-2k(u-us)})} - g_{s2}s$$
$$\frac{dv}{dt} = -g_v^+ \cdot v \qquad \frac{dw}{dt} = -g_w^+ \cdot w$$

The exponential term on the right-hand side of the ODE is the sigmoid function, which often appears in modelling biological switches. On this model, our tool is able to perform a depth-55 unrolling, and solve the generated logic formula. Such a formula contains 280 nonlinear ODEs of the type shown here,

with 616 variables. The computed trace from dReal suggests a witness of the reachability property that can be confirmed by experimental simulation. Figure 6 shows the comparison between the trace computed from bounded model checking and the actual experimental simulation trace.
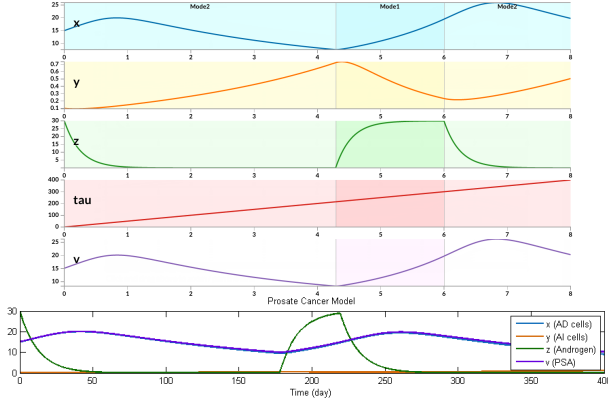

Fig. 7: Above: Witness computed for the CT model at depth 3 and 500 time units. Below: Experimental simulation data.

The CT model represents a prostate cancer treatment model that contains nonlinear ODEs such as following:

$$\frac{dx}{dt} = (\alpha_x(k_1 + (1-k_1)\frac{z}{z+k_2}$$
$$-\beta_x((1-k_3)\frac{z}{z+k_4}+k_3)) - m_1(1-\frac{z}{z_0}))x + c_1 x$$
$$\frac{dy}{dt} = m_1(1-\frac{z}{z_0})x + (\alpha_y(1-d\frac{z}{z_0})-\beta_y)y + c_2 y$$
$$\frac{dz}{dt} = \frac{-z}{\tau} + c_3 z$$
$$\frac{dv}{dt} = (\alpha_x(k_1 + (1-k_1)\frac{z}{z+k_2} - \beta_x(k_3 + (1-k_3)\frac{z}{z+k_4}))$$
$$-m_1(1-\frac{z}{z_0}))x + c_1 x + m_1(1-\frac{z}{z_0})x$$
$$+(\alpha_y(1-d\frac{z}{z_0})-\beta_y)y + c_2 y$$

The EO model represents an electronic oscillator model that contains nonlinear ODEs such as the following:

$$\frac{dx}{dt} = -ax \cdot sin(\omega_1 \cdot \tau)$$
$$\frac{dy}{dt} = -ay \cdot sin((\omega_1 + c_1) \cdot \tau) \cdot sin(\omega_2) \cdot 2$$
$$\frac{dz}{dt} = -az \cdot sin((\omega_2 + c_2) \cdot \tau) \cdot cos(\omega_1) \cdot 2$$
$$\frac{\omega_1}{dt} = -c_3 \cdot \omega_1 \quad \frac{\omega_2}{dt} = -c_4 \cdot \omega_2 \quad \frac{d\tau}{dt} = 1$$

The other models are standard simple nonlinear models (for instance, bouncing ball with nonlinear friction), on which our tool has no difficulty in solving.

V. CONCLUSION

In this paper we have studied SMT problems over the real numbers with ODE constraints. We have developed δ-complete algorithms in the DPLL(ICP) framework, for both the standard SMT formulas that are purely existentially quantified, as well

as $\exists\forall$-formulas whose universal quantification is restricted to the time variables. We have demonstrated the scalability of our approach on nonlinear SMT benchmarks. We believe that the proposed decision procedures can scale on nonlinear problems and can serve as the underlying engine for formal verification of realistic hybrid systems and embedded software.

REFERENCES

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

[2] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 16. Elsevier, 2006.

[3] R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich. The opensmt solver. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 150–153. Springer, 2010.

[4] A. Cimatti, S. Mover, and S. Tonetta. A quantifier-free SMT encoding of non-linear hybrid automata. In *FMCAD*, pages 187–195, 2012.

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

[6] J. Cruz and P. Barahona. Constraint satisfaction differential problems. In *CP*, pages 259–273, 2003.

[7] A. Eggers, M. Fränzle, and C. Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In *ATVA*, pages 171–185, 2008.

[8] A. Eggers, N. Ramdani, N. Nedialkov, and M. Fränzle. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In G. Barthe, A. Pardo, and G. Schneider, editors, *SEFM*, volume 7041 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2011.

[9] S. Gao, J. Avigad, and E. M. Clarke. Delta-complete decision procedures for satisfiability over the reals. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 2012.

[10] S. Gao, J. Avigad, and E. M. Clarke. Delta-decidability over the reals. In *LICS*, pages 305–314, 2012.

[11] S. Gao, S. Kong, and E. Clarke. dReal: An SMT solver for nonlinear theories over the reals. *CADE*, 2013.

[12] A. Goldsztejn, O. Mullier, D. Eveillard, and H. Hosobe. Including ordinary differential equations based constraints in the standard cp framework. In D. Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 221–235. Springer, 2010.

[13] L. Granvilliers. Parameter estimation using interval computations. *SIAM J. Sci. Comput.*, 26(2):591–612, Feb. 2005.

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

[15] R. Grosu, G. Batt, F. H. Fenton, J. Glimm, C. L. Guernic, S. A. Smolka, and E. Bartocci. From cardiac cells to genetic regulatory networks. In *CAV*, pages 396–411, 2011.

[16] D. Ishii, K. Ueda, and H. Hosobe. An interval-based sat modulo ode solver for model checking nonlinear hybrid systems. *STTT*, 13(5):449–461, 2011.

[17] K.-I. Ko. *Complexity Theory of Real Functions*. BirkHauser, 1991.

[18] Y. Lin and M. A. Stadtherr. Guaranteed state and parameter estimation for nonlinear continuous-time systems with bounded-error measurements. *Industrial and Engineering Chemistry Research*, pages 7198–7207, 2007.

[19] K. Weihrauch. *Computable Analysis: An Introduction*. 2000.