Computable Analysis, Hybrid Automata, and Decision Procedures (Extended Thesis Abstract)

Sicun Gao

1 Introduction

1.1 Overview

Cyber-Physical Systems (CPS) are systems that integrate digital computation with the physical environment. They include almost every complex piece of machinery around us, from airplanes to nuclear plants to cardiac pacemakers. In these systems, computation units process information from a physical plant to control it in real-time. Despite their ubiquitous and safety-critical applications, the search for systematic theories and techniques for the analysis and design of these systems has seen limited success. The main difficulty comes from their tight integration of discrete and continuous components, as studied in the area of *hybrid systems*. Formal verification of hybrid systems has appeared fundamentally unattainable. Theoretically, verifying safety of simple hybrid systems is highly undecidable. Practically, the lack of powerful reasoning engines in this domain prevents the use of advanced formal verification techniques. My thesis develops a framework to overcome these barriers.

Theoretical negative results on verification of hybrid systems mostly refer to the limitations of precise and symbolic algorithms. We show that an appropriate relaxation leads to strong decidability and complexity results, both for the decision problems of the underlying logic formulas and the verification problems. The key step is to develop a notion of numerical approximations in the standard decision problems. We define the δ-*decision problem*, where δ is any positive rational number: for a formula φ, we ask whether φ is true or a δ-perturbation of φ is false (or the other way around). Here, a δ-perturbation of a formula is a syntactic variant of it, obtained by modifying some of its constant terms up to δ. The key theorem is that the δ-decision problem is decidable for sentences with bounded quantifiers in first-order theories over the reals with arbitrary Type 2 computable functions, which are essentially "numerically computable" real functions. This notion of numerical computability has been studied extensively in the field of computable analysis, which applies to most well-known continuous functions including transcendental functions and solution functions of differential equations. The δ-decidability results stand in sharp contrast with the well-known undecidability of first-order formulas over the reals with trigonometric functions.

We then show that δ-decidability leads to a new framework for hybrid system verification. We describe hybrid automata using first-order formulas with Type 2 computable functions, which are expressive enough for almost all systems of practical relevance. We define the notion of δ-reachability for hybrid automata using δ-perturbations on first-order formulas. The decidability and complexity results of the logic formulas then transfer to bounded δ-reachability problems of hybrid automata. Again, such results bypass the well-known high undecidability of reachability for simple hybrid systems. In practice, the new framework allows us to exploit the full power of numerical methods in decision problems and formal verification. We define the notion of δ-*complete decision procedures*, which serves as a performance requirement for using numerically-driven procedures in formal verification. We show an analysis of the powerful constraint solving framework *Interval Constraint Propagation* (ICP), and formalize conditions

under which it is δ-complete. We have built a practical tool dReal combining ICP in the DPLL(T) framework. It has successfully solved logic formulas with hundreds of nonlinear differential equations or transcendental functions. This tool is the backend of a hybrid system verification tool dReach, which has efficiently handled real-world nonlinear hybrid system models that are much beyond the reach of other existing tools.

I will sketch the main results in each part of the thesis. Section 2 contains theoretical results on δ-decidability over the reals. Section 3 gives a framework for reachability analysis of hybrid systems based on δ-decidability. Section 4 gives the framework of δ-complete decision procedures as the underlying solving engine. Section 5 introduces the practical tools that implement the framework.

1.2 Related Work

Safety Verification of Hybrid Automata. Hybrid automata are infinite-state systems that combine discrete and continuous parts, it is not surprising that their safety verification problems can be very difficult. Along with the first definition of hybrid automata in [4, 3], it is already shown that the reachability problem for simple classes of hybrid automata, with only constant-rate continuous dynamics, is undecidable. In fact, it can be shown that reachability problems for piecewise-linear systems are already hard enough to encode any problem in the analytic hierarchy [10]. Such very strong negative results seem to indicate that formal verification of realist ic hybrid automata is inherently impossible. Thus, existing approaches have either focused on the simplest decidable classes (timed automata [4]), which can be reduced to discrete systems, or heuristic algorithms for some undecidable classes. We can categorize these approaches based on whether they focus on computing representations of the sets of the reachable states (model-theoretic) or constructing logic proofs of correctness (proof-theoretic).

On the model-theoretic side, *Reachable Set Computation* methods compute reachable states of a hybrid automaton to decide if the safety properties can be falsified. They key challenge is to find suitable representations for the set of reachable states and efficient algorithms for manipulating them. The tool HYTECH was the first model checker to implement symbolic reachability analysis for hybrid systems [5]. It is based on representing the reachable states by polyhedra, where a polyhedron is represented by a conjunction of linear inequalities. The models are restricted to the class of hybrid automata with constant dynamics. The approach is then generalized to handle linear differential equations, as implemented by the tools CheckMate [12] and d/dt [6], using polyhedra to compute *flowpipe* approximations. It is worth noting that the complexity of operations on polyhedra is usually exponential in the number of dimensions, which makes it hard for the reachable set computation to scale. Recent progress involves using Zonotopes and support functions [19, 20], which leads to more efficient algorithms for handling linear systems. On the other hand, the main drawback of the method is the difficulty with handling logic operations in the discrete jumps using the geometric approximations. In general, explicit reachable set computation has been shown possible only for very limited classes of hybrid automata. On the proof-theoretic side, *Deductive Verification* methods uses theorem provers for proving correctness. Such methods are based on identifying inductive invariants, and avoid the iterative calculation of the reachable state sets and is not limited to linear hybrid automata. The tool KEYMAERA [28], based on Differential Dynamic Logic [27, 25, 26], provides support for constructing proofs for correctness by certifying inductive invariants. The proof engine relies on external symbolic algorithms for quantifier elimination in real arithmetic, which puts a constraint on the overall scalability and expressiveness. In most cases such inductive invariants need to be suggested by the user. Automated generation of inductive invariants has remained an active area of research [30, 30].

Decision Procedures over the Reals. While efficient algorithms [14] exist for deciding SMT problems with only linear real arithmetic, practical problems normally contain nonlinear polynomials, transcendental functions, and differential equations. Solving formulas with these functions is inherently intractable. Decision algorithms [13] for formulas with nonlinear polynomials have very high complexity [11]. A more fundamental problem is the lack of

expressiveness: many problems in the intended domains of application cannot even be expressed in the language of real-closed fields. When the sine function is involved, the SMT problem is undecidable, and only partial algorithms can be developed [7, 2]. The symbolic approaches include Cylindrical Decomposition [13], with significant recent improvement [24, **?**], and Gröbner bases [29]. A drawback of symbolic algorithms is that it is restricted to arithmetic, namely polynomial constraints, with the exception of [1]. On the other hand, many practical solvers incorporate scalable numerical computations. Examples of numerical algorithms that have been exploited include optimization algorithms [9, 23], interval-based algorithms [16, 15, 18], Bernstein polynomials [22], and linearization algorithms [17]. In these existing approaches, the numerical algorithms are used as partial heurstics, and there is no framework for comparing their power or formulating correctness guarantees.

2 δ-Decidability over the Reals

Tarski's celebrated result that the first-order theory of real arithmetic is decidable has had a profound impact on automated theorem proving, and has generated much attention in application domains such as formal verification, control theory, and robotics. The hope is that practical problems can be encoded as first-order formulas and automatically solved by decision procedures for the theory. However, in spite of extensive research in optimizing the decision algorithms, there is still a wide gap between the state-of-the-art and the majority of problems in practice. One reason is the procedures' high computational complexity: general quantifier elimination, even restricted to a linear signature, has a doubly exponential lower-bound. A more fundamental problem is the lack of expressiveness: many problems in the intended domains of application cannot even be expressed in the language of real-closed fields. Problems from formal verification and control design can appear much beyond what can be currently solved, because of the use of differential equations, alternating quantifiers, as well as their sheer scale. It is well known that even the set of Σ_1 sentences in a language extending real arithmetic with the sine function is already undecidable. This seems to indicate that developing general logic-based automated methods in these domains is at its core impossible. Our goal is to show that a slight change of perspective provides a completely different, and much more positive, outlook.

It is important to note that the theoretical negative results only refer to the problem of deciding logic formulas *symbolically and precisely*. In this setting, the numerical computability of real functions remains mostly unexploited. This hardly reflects the wide range of solving techniques in practice. For instance, in the Flyspeck project, the nonlinear formulas are proved using various numerical optimization techniques, including linear programming, interval analysis, and Bernstein approximations. In the field of formal verification of real-time systems, a recent trend in developing decision solvers that incorporate numerical methods has also proved very promising. It is natural to ask whether such practices can be theoretically justified in the context of decision problems for first-order theories. Namely, can we give a characterization of the first-order formulas that can be solved using numerically-driven procedures, and if so, bound the complexity of these procedures? Can we formulate a framework for understanding the guarantees that numerically-driven decision procedures can provide? Can we provide general conditions under which a practical verification problem has a satisfactory solution? We answer these questions affirmatively. The key is to shift to a δ-relaxed notion of correctness, which is more closely aligned with the use of numerical procedures.

An informal description of what we can show is as follows. In a very general signature that contains all the aforementioned real functions, there exists an algorithm such that given an arbitrary sentence φ involving only bounded quantifiers, and an arbitrary small numerical parameter δ, one of the following decisions is returned:

- φ is true;
- The "δ-strengthening" of φ is false.

The δ-strengthening of a formula, defined below, is a numerical perturbation which makes it slightly harder for the formula to be true. For example, the strengthening of $\exists x \in I.\ x > 0$, where I is the bound on the quantifier, is

$\exists x \in I.\ x > \delta$. Thus the algorithm reports either that the given formula is true, or that some small perturbation makes it false. These two cases are not mutually exclusive, and in the "grey area" where both cases hold the algorithm is allowed to return either value. We refer to this problem (as well as the dual problem defined below using the δ-weakening of formulas) as the "δ-relaxed decision problem," or simply the "δ-decision problem." The restriction to bounded quantifiers is reasonable, since in practical problems real-valued variables are typically considered within some range.

Here is another way of thinking about our main result. Given a small δ, we can consider the set of first-order sentences with the property that their truth values remain invariant under δ-strengthening (or δ-weakening). Such sentences can be called "δ-robust," in that they do not fall into the "grey area" mentioned in the last paragraph. We believe that, in situations like the Flyspeck project where numerical methods are used, it is implicitly assumed that the relevant assertions have this property. Our algorithm, in particular, decides the truth of bounded δ-robust sentences in a general signature.

Moreover, we show that the δ-decision problems reside in reasonable complexity classes. For instance, if the signature is given by extending arithmetic with exp and sin, the δ-decision problem for bounded Σ_1-sentences is "only" NP-complete. This should be compared with the undecidability of sentences in this class in the ordinary setting. As another example, the δ-decision problem for arbitrarily-quantified bounded sentences with Lipschitz-continuous ordinary differential equations is PSPACE-complete. The fact that this complexity is not higher than that of deciding quantified Boolean formulas is striking.

The "general signature" we mentioned above refer to arbitrary Type 2 computable functions. We now formally state our results. Let $\mathcal{F}$ be any collection of Type 2 computable real functions. First, there exists an algorithm such that given any $\mathcal{L}_{\mathcal{F}}$-sentence φ containing only bounded quantifiers, and any positive rational number δ, decides the δ-relaxed decision problem. Secondly, suppose all the functions in $\mathcal{F}$ are in a Type 2 complexity class C (closed under polynomial-time reduction), then the δ-relaxed decision problem for Σ_n-sentences in $\mathcal{L}_{\mathcal{F}}$ resides in $(\Sigma_n^{\mathsf{P}})^{\mathsf{C}}$. Moreover, the relaxations are necessary. Without either boundedness or δ-relaxation, the general problem would remain undecidable.

2.1 Computable Functions over the Reals

We can encode any real number as an infinite sequence of rational numbers. We write the set of *dyadic rational numbers* as $\mathbb{D} = \{m/2^n : m \in \mathbb{Z}, n \in \mathbb{N}\}$.

Definition 2.1 (Names)**.** *For each real number x, a* **name** *of x is any function $\phi : \mathbb{N} \to \mathbb{D}$ that* **binary-converges** *to x. That is, $\forall n \in \mathbb{N}, |\phi(n) - x| \leq 2^{-n}$. Note that this representation is not unique. We write CF_x to denote the set of all ϕs that binary-converge to x. A real number x is computable, if there exists a computable function $\phi \in \mathsf{CF}(x)$.*

We can compute a real function $f : \mathbb{R} \to \mathbb{R}$ if there is a machine M that, given the representation of any argument $x \in \mathbb{R}$ of the function, computes the representation of its value $f(x)$. Such computation can be realized by function-oracle machines in the following way. The input x is given to M by some $\phi \in \mathsf{CF}_x$ as a function oracle, and the precision 2^{-n} is given as an integer n in unary notation (a string 0^n) as an input to M. M computes $f(x)$ by repeating two steps: first, it decides the precision of the input needed for producing an output of the desirable precision 2^{-n}; second, it queries the function oracle to obtain an approximation of the input and compute the output. Namely, M queries the oracle for $\phi(m)$, which by definition satisfies $|\phi(m) - x| \leq 2^{-m}$ and computes an output $d \in \mathbb{Q}$ with $|d - f(x)| \leq 2^{-n}$, using $\phi(m)$. This is the intuition behind the following formal definition of computable real functions.

Definition 2.2 (Computable Real Functions)**.** *A real function $f : \mathbb{R} \to \mathbb{R}$ is* **computable***, if there is a function-oracle Turing machine M such that for every $x \in \mathbb{R}$ and every $\phi \in \mathsf{CF}_x$, given any $i \in \mathbb{N}$, the machine uses ϕ as an*

oracle, and n as the input, and computes a rational number $M^\phi(i) \in \mathbb{Q}$, such that $|M^\phi(i) - f(x)| \leq 2^{-i}$. In other words, M computes a function ψ that Cauchy-represents $f(x)$. We say a function $f : [a,b] \to \mathbb{R}$ is computable over $[a,b] \subseteq \mathbb{R}$ if the above conditions hold for all $x \in [a,b]$.

A most important property of computable functions is that they must have a computable modulus of continuity. Basically, if a function has a computable uniform modulus of continuity, then fixing any error bound 2^{-i}, we can compute a global error bound $2^{-m_f(i)}$ such that using any $2^{-m_f(i)}$-approximation of x, $f(x)$ can be obtained with error smaller than 2^{-i}. Most standard real functions are Type 2 computable, including: arithmetic, absolute value function, min and max, polynomials with computable coefficients, exponential, trigonometric, square root, and logarithm functions, and solution functions of Lipschitz-continuous ordinary differential equations are Type 2 computable.

Complexity Classes of Type 2 Functions. Let the ordinary complexity classes such as $\mathsf{P}, \mathsf{NP}, \Sigma_k^\mathsf{P}, \mathsf{PSPACE}$ for decision problems be defined in the standard way. Complexity of real functions is usually defined over compact domains. Without loss of generality, we consider functions over $[0,1]$. Intuitively, a real function $f : [0,1] \to \mathbb{R}$ is (uniformly) P-computable (PSPACE-computable), if it is computable by an oracle Turing machine M_f that halts in polynomial-time (polynomial-space) for every $i \in \mathbb{N}$ and every $\vec{x} \in \mathrm{dom}(f)$. Formally, we use the following standard definition:

Definition 2.3. *A real function $f : [0,1]^n \to \mathbb{R}$ is in $\mathsf{P}_{\mathsf{C}[0,1]}$ (resp. $\mathsf{PSPACE}_{\mathsf{C}[0,1]}$), iff there exists a representation (m_f, θ_f) of f such that m_f is a polynomial function, and for any $d \in (\mathbb{D} \cap [0,1])^n$, $e \in \mathbb{D}$, and $i \in \mathbb{N}$, $\theta_f(d,i)$ is computable in time (resp. space) $O((len(d)+i)^k)$ for some constant k.*

Most common real functions reside in $\mathsf{P}_{\mathsf{C}[0,1]}$: absolute value, polynomials, binary max and min, exp, and sin are all in $\mathsf{P}_{\mathsf{C}[0,1]}$. It has been that solutions of Lipschitz-continuous differential equations are $\mathsf{PSPACE}_{\mathsf{C}[0,1]}$-complete.

2.2 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-Formulas

We consider first-order formulas with Type 2 computable functions interpreted over the reals. We write $\mathcal{F}$ to denote an arbitrary collection of symbols representing Type 2 computable functions over $\mathbb{R}^n$ for various n. We always assume that $\mathcal{F}$ contains at least the constant 0, unary negation, addition, and the absolute value. (Constants are seen as constant functions.) Let $\mathcal{L}_\mathcal{F}$ be the signature $\langle \mathcal{F}, > \rangle$. $\mathcal{L}_\mathcal{F}$-formulas are always evaluated in the standard way over the corresponding structure $\mathbb{R}_\mathcal{F} = \langle \mathbb{R}, \mathcal{F}, > \rangle$.

It is not hard to see that we only need to use atomic formulas of the form $t(x_1, ..., x_n) > 0$ or $t(x_1, ..., x_n) \geq 0$, where $t(x_1, ..., x_n)$ are built up from functions in $\mathcal{F}$. This follows from the fact that $t(\vec{x}) = 0$ can be written as $-|t(\vec{x})| \geq 0$, $t(\vec{x}) < 0$ as $-t(\vec{x}) > 0$, and $t(\vec{x}) \leq 0$ as $-t(\vec{x}) \geq 0$. We then take expressions $s < t$ and $s \leq t$ to abbreviate $t - s > 0$ and $t - s \geq 0$, respectively. Moreover, when a formula is in negation normal form, the negations in front of atomic formulas can be eliminated by replacing $\neg t(\vec{x}) > 0$ with $-t(\vec{x}) \geq 0$, and $\neg t(\vec{x}) \geq 0$ with $-t(\vec{x}) > 0$.

In general, to avoid extra preprocessing of formulas, we give an explicit definition of $\mathcal{L}_\mathcal{F}$-formulas as follows.

Definition 2.4 ($\mathcal{L}_\mathcal{F}$-Formulas)**.** *Let $\mathcal{F}$ be a collection of Type 2 functions, which contains at least 0, unary negation -, addition +, and absolute value $|\cdot|$. We define:*

$$t := x \mid f(t(\vec{x})), \text{ where } f \in \mathcal{F}, \text{ possibly constant};$$
$$\varphi := t(\vec{x}) > 0 \mid t(\vec{x}) \geq 0 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x_i \varphi \mid \forall x_i \varphi.$$

In this setting $\neg\varphi$ is regarded as an inductively defined operation which replaces atomic formulas $t > 0$ with $-t \geq 0$, atomic formulas $t \geq 0$ with $-t > 0$, switches $\wedge$ and $\vee$, and switches $\forall$ and $\exists$. Implication $\varphi_1 \to \varphi_2$ is defined as $\neg\varphi_1 \vee \varphi_2$. We use the notation of bounded quantifiers*, defined as $\exists^{[u,v]} x.\varphi =_{df} \exists x.(u \leq x \wedge x \leq v \wedge \varphi)$ and $\forall^{[u,v]} x.\varphi =_{df} \forall x.((u \leq x \wedge x \leq v) \to \varphi)$. We say a sentence is bounded if it only involves bounded quantifiers.*

Expressiveness of $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$. Many standard problems related to ODE systems can be easily generalized and encoded in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$. They motivate the development of decision procedures for the theory. We give encoding for generalizations of Initial Value Problems, Boundary Value Problems, Differential-Algebraic Equations, etc.

2.3 δ-Decidability

We define δ-weakening and δ-strengthening of bounded $\mathcal{L}_\mathcal{F}$-sentences, which explicitly introduce syntactic perturbations in a formula. They are used to formalize the notion of δ-relaxed decision problems for $\mathcal{L}_\mathcal{F}$-sentences.

Definition 2.5 (δ-Variants)**.** *Let $\delta \in \mathbb{Q}^+ \cup \{0\}$, and φ a bounded $\mathcal{L}_\mathcal{F}$-sentence of the form*

$$Q_1^{I_1} x_1 \cdots Q_n^{I_n} x_n . \psi[t_i > 0; t_j \geq 0],$$

where $i \in \{1, ...k\}$ and $j \in \{k+1, ..., j\}$. The δ-strengthening $\varphi^{+\delta}$ of φ is defined to be the result of replacing each atomic formula $t_i > 0$ by $t_i > \delta$ and each atomic formula $t_j \geq 0$ by $t_j \geq \delta$, that is,

$$Q_1^{I_1} x_1 \cdots Q_n^{I_n} x_n . \psi[t_i > \delta; t_j \geq \delta],$$

where $i \in \{1, ...k\}$ and $j \in \{k+1, ..., j\}$. Similarly, the δ-weakening $\varphi^{-\delta}$ of φ is defined to be the result of replacing each atomic formula $t_i > 0$ by $t_i > -\delta$ and each atomic formula $t_j \geq 0$ by $t_j \geq -\delta$, that is,

$$Q_1^{I_1} x_1 \cdots Q_n^{I_n} x_n . \psi[t_i > -\delta; t_j \geq -\delta].$$

We say that a sentence is δ-robust if its truth value remains invariant under δ-weakening, namely, if $\varphi^{-\delta} \to \varphi$. We say φ is robust if it is δ-robust for some $\delta \in \mathbb{Q}^+$.More precisely, we can say that a formula φ is *robust under δ-weakening* if it has this property, and define the analogous notion of being *robust under δ-strengthening*. The two notions have similar properties; for simplicity, we will restrict attention to the first notion below. By Proposition **??**, we always have $\varphi \to \varphi^{-\delta}$, so φ is δ-robust if and only if we have $\varphi \leftrightarrow \varphi^{-\delta}$. Since $\varphi^{-\delta} \to \varphi$ is equivalent to $\neg\varphi^{-\delta} \vee \varphi$, saying that φ is robust is equivalent to saying that either φ is true or $\varphi^{-\delta}$ is false. Intuitively, this means that either φ is true, or "comfortably" false in the sense that no small perturbation makes it true.

Main Theorem. Our main theorem is the following.

Theorem 2.6 (δ-Decidability)**.** *There is an algorithm which, given any bounded $\mathcal{L}_\mathcal{F}$-sentence φ and $\delta \in \mathbb{Q}^+$, correctly returns one of the following two answers:*

- *"*True*": φ is true.*
- *"δ-*False*": $\varphi^{+\delta}$ is false.*

Note that the two cases can overlap. If φ is true and $\varphi^{+\delta}$ is false, then the algorithm is allowed to return either one. The proof idea is that for any formula φ, the strictification of φ is equivalent to the formula $\alpha(\varphi) > 0$. Whether this holds cannot, in general, be determined algorithmically, But given a small δ, we *can* make a choice between the overlapping alternatives $\alpha(\varphi) > 0$ and $\alpha(\varphi) < \delta$, and this is enough to solve the relaxed decision problem. To prove the main theorem, we need the following definitions and lemmas. First, any $\mathcal{F}$ can be extended it as follows.

Definition 2.7 (m-Extension)**.** *Let $\mathcal{F}$ be a collection of computable functions over reals. We define the m-extension of $\mathcal{F}$, written as $\mathcal{F}_m$, to be the closure of $\mathcal{F}$ with the following functions:*

- *Binary min and max:* $\min(\cdot,\cdot), \max(\cdot,\cdot)$*;*

- *Bounded min and max:*

$$\min\{t(\vec{x},\vec{y}) : y_1 \in [u_1, v_1], ..., y_n \in [u_n, v_n]\},$$
$$\max\{t(\vec{x},\vec{y}) : y_1 \in [u_1, v_1], ..., y_n \in [u_n, v_n]\},$$

  where u_i and v_i denote arbitrary $\mathcal{L}_{\mathcal{F}_m}$-terms that do not involve y_i.

It is a standard result in computable analysis that applying minimization and maximization over a bounded interval preserves computability. (This is studied in detail in Chapter 3 of [21].) Thus all functions in $\mathcal{F}_m$ are computable. We can write the bounded min and max as $\min_{\vec{x}\in D}(t(\vec{x},\vec{y}))$ and $\max_{\vec{x}\in D}(t(\vec{x},\vec{y}))$ for short, where $D = [u_1, v_1] \times \cdots \times [u_n, v_n]$. For technical reasons that will become clear in Section 2.4, we interpret $[u, v]$ as $[v, u]$ when $v < u$; one can rule out this interpretation by adding $u \leq v$ as an explicit constraint in the formula. We define a notion that allows us to switch between strict and nonstrict inequalities in the δ-decision problem.

Definition 2.8 (Strictification)**.** *Suppose φ is the formula*

$$\vec{Q}^{\vec{I}}\vec{x}.\psi[t_1 > 0, ..., t_k > 0; t_{k+1} \geq 0, ..., t_m \geq 0].$$

We say φ is strict (resp. nonstrict), if $m = k$ (resp. $k = 0$), i.e., all the inequalities occurring in φ are strict (resp. nonstrict). The strictification of φ *is defined to be*

$$st(\varphi) : \ \vec{Q}^{\vec{I}}\vec{x}.\psi[t_1 > 0, ..., t_k > 0, t_{k+1} > 0, ..., t_m > 0],$$

that is, the result of replacing all the nonstrict inequalities by strict ones. The destrictification of φ *is*

$$de(\varphi) : \ \vec{Q}^{\vec{I}}\vec{x}.\psi[t_1 \geq 0, ..., t_k \geq 0, t_{k+1} \geq 0, ..., t_m \geq 0],$$

this is, the result of replacing all strict inequalities by nonstrict ones.

Note that the bounds on the quantifiers are not changed in the definition. The following fact follows directly from the definition.

Proposition 2.9. *We have*

- *$st(\varphi) \to \varphi$ and $\varphi \to de(\varphi)$.*
- *(Duality) $st(\neg\varphi)$ is equivalent to $\neg de(\varphi)$.*

The key lemma establishes that any bounded $\mathcal{L}_{\mathcal{F}}$-sentence can be expressed as an atomic formula in the extended signature $\mathcal{L}_{\mathcal{F}_m}$.

Lemma 2.10. *Let φ be a bounded $\mathcal{L}_{\mathcal{F}}$-sentence. There is an $\mathcal{L}_{\mathcal{F}_m}$-term $\alpha(\varphi)$ that satisfies:*

- *$de(\varphi) \leftrightarrow \alpha(\varphi) \geq 0$, and $st(\varphi) \leftrightarrow \alpha(\varphi) > 0$;*
- *$de(\varphi^{+\delta}) \leftrightarrow \alpha(\varphi) \geq \delta$, and $st(\varphi^{+\delta}) \leftrightarrow \alpha(\varphi) > \delta$.*

Now, the idea of the proof for the main theorem is as follows. For any formula φ, the strictification of φ is equivalent to the formula $\alpha(\varphi) > 0$. Whether this holds cannot, in general, be determined algorithmically, But given a small δ, we *can* make a choice between the overlapping alternatives $\alpha(\varphi) > 0$ and $\alpha(\varphi) < \delta$, and this is enough to solve the relaxed decision problem.

Corollaries of the main theorem. We have the following corollaries that easily follow from the main theorem.

Corollary 2.11. *There is an algorithm which, given any bounded φ and $\delta \in \mathbb{Q}^+$, correctly returns one of the following two answers:*

- *"δ-True": $\varphi^{-\delta}$ is true.*
- *"False": φ is false.*

Corollary 2.12 (Robustness implies decidability)**.** *There is an algorithm that, given $\delta \in \mathbb{Q}^+$ and a bounded δ-robust φ, decides whether φ is true or false.*

Corollary 2.13. *Let L be a class of bounded $\mathcal{L}_{\mathcal{F}}$-sentences. Suppose it is undecidable whether an arbitrary sentence in L is true. Then it is undecidable, given any $\delta \in \mathbb{Q}^+$, whether an arbitrary bounded $\mathcal{L}$-sentence is δ-robust.*

This can be contrasted with the simple fact that if $\mathbb{R}_{\mathcal{F}}$ has a decidable theory, then it is decidable whether any bounded $\mathcal{L}_{\mathcal{F}}$-sentence is robust, since the conditions can be expressed by just another bounded $\mathcal{L}_{\mathcal{F}}$-sentence.

We can contrast the above results with the following negative results, to show that both the boundedness and δ-relaxation are necessary for decidability. We allow the signature $\mathcal{L}_{\mathcal{F}}$ to be arbitrary Type 2 computable functions, then without either boundedness or robustness, $\mathcal{L}_{\mathcal{F}}$-sentences are undecidable.

Proposition 2.14. *There exists $\mathcal{F}$ such that it is undecidable whether an arbitrary quantifier-free sentence (and thus trivially bounded) in $\mathcal{L}_{\mathcal{F}}$ is true.*

The proof of this proposition involves adding countably many constant symbols to the language, one for each a_i. Alternatively, it is not hard to define a single computable function $g : \mathbb{Q} \to \mathbb{R}$ such that for each $i \in \mathbb{N}$, $g(i) = a_i$, by interpolating outputs linearly for inputs between integer values.

Proposition 2.15. *There exists $\mathcal{F}$ such that it is undecidable whether an arbitrary δ-robust quantifier-free $\mathcal{L}_{\mathcal{F}}$-sentence is true.*

2.4 Complexity Results

In this section we consider the complexity of the δ-decision problem for signatures of interest. In the proof of the main theorem, we have established a reduction from the δ-decision problems of $\mathcal{L}_{\mathcal{F}}$ to computing the value of $\mathcal{L}_{\mathcal{F}_m}$-terms with alternations of min and max. The complexity of computing such terms can be exactly characterized by the min-max hierarchy over computable functions, as defined in [21].

First, we need the definition of $\Sigma_{\mathsf{k},\mathsf{C}[0,1]}$-functions.

Definition 2.16 ([21])**.** *For $k \geq 0$, we say a real function $f : [0,1] \to \mathbb{R}$ is in $\Sigma_{\mathsf{k},\mathsf{C}[0,1]}$ (resp. $\Pi_{\mathsf{k},\mathsf{C}[0,1]}$) if there exists a representation (m_f, θ_f) of f, such that*

1. *The modulus function $m_f : \mathbb{N} \to \mathbb{N}$ is a polynomial, and*
2. *for all $d \in \mathbb{D} \cap [0,1]$ and all $i \in \mathbb{N}$, $|\theta_f(d,n) - f(d)| \leq 2^{-i}$, and the set $A_{\theta_f} = \{\langle d, e, 0^i \rangle : e \leq \theta_f(d,i)\}$ is in Σ_{k} (resp. Π_{k}). (0^i denotes the string of i zeros.)*

Remark 2.17. *Note that using membership queries to A_ψ, we can easily (in polynomial-time) determine the value of $\psi(d,i)$. Thus by replacing the third condition with P or PSPACE, we obtain the definition of $\mathsf{P}_{\mathsf{C}[0,1]}$ and $\mathsf{PSPACE}_{\mathsf{C}[0,1]}$. It is also clear that $\Sigma_{0,\mathsf{C}[0,1]} = \Pi_{0,\mathsf{C}[0,1]} = \mathsf{P}_{\mathsf{C}[0,1]}$.*

The key result as shown by Ko [21] is that, if $f(x,y)$ is in $\mathsf{P}_{\mathsf{C}[0,1]}$, then $\max_{x\in[0,1]} f(x,y)$ is in $\mathsf{NP}_{\mathsf{C}[0,1]}$:

Proposition 2.18 ([21]). *Let $f:[0,1]^n\to\mathbb{R}$ be a real function in $\mathsf{P}_{\mathsf{C}[0,1]}$. Define $g:[0,1]^{m_0}\to\mathbb{R}$ as*

$$g(\vec{x}_0)=\max_{\vec{x}_1\in[0,1]^{m_1}}\min_{\vec{x}_2\in[0,1]^{m_2}}\cdots\underset{\vec{x}_k\in[0,1]^{m_k}}{opt} f(\vec{x}_0,\vec{x}_1,...,\vec{x}_k)$$

where opt is min *if k is even and* max *if k is odd, and $\sum_{i=0}^k m_i=n$. We then have $g\in\Sigma_{\mathsf{k},\mathsf{C}[0,1]}$.*

Following the definition of $\Sigma_{\mathsf{k},\mathsf{C}[0,1]}$-classes, it is straightforward to obtain the decision version of this result, and also to relativize to complexity classes other than $\mathsf{P}_{\mathsf{C}[0,1]}$.

Lemma 2.19. *Suppose $f:[0,1]^n\to\mathbb{R}$ is in complexity class C with a polynomial modulus function. Define $g:[0,1]^{m_0}\to\mathbb{R}$ as*

$$g(\vec{x}_0)=\max_{\vec{x}_1\in[0,1]^{m_1}}\min_{\vec{x}_2\in[0,1]^{m_2}}\cdots\underset{\vec{x}_k\in[0,1]^{m_k}}{opt} f(\vec{x}_0,\vec{x}_1,...,\vec{x}_k)$$

where opt is min *if k is even and* min *if k is odd, and $\sum_{i=0}^k m_i=n$. Then there exists a representation of g, (m_g,θ_g), such that the following problem is in $(\Sigma_{\mathsf{k}}^{\mathsf{P}})^{\mathsf{C}}$: given any $d,e\in\mathbb{D}$ and $i\in\mathbb{N}$, decide if $\theta_g(d,i)\geq e$.*

Now we are ready to state the complexity results for the δ-decision problems.

Theorem 2.20. *Let $\mathcal{F}$ be a class of computable functions. Let S be a class of $\mathcal{L}_{\mathcal{F}}$-sentences, such that for any φ in S, the terms in $\varphi_{[0,1]}$ are computable in complexity class C where $\mathsf{P}_{\mathsf{C}[0,1]}\subseteq\mathsf{C}\subseteq\mathsf{PSPACE}_{\mathsf{C}[0,1]}$. Then, for any $\delta\in\mathbb{Q}^+$, the δ-decision problem for bounded Σ_n-sentences in S is in $(\Sigma_{\mathsf{n}}^{\mathsf{P}})^{\mathsf{C}}$.*

As corollaries, we now prove completeness results for signatures of interest.

Corollary 2.21. *Let $\mathcal{F}$ be a set of P-computable functions (which, for instance, includes* exp *and* sin*). The δ-decision problem bounded Σ_n-sentences in $\mathcal{L}_{\mathcal{F}}$ is $\Sigma_{\mathsf{n}}^{\mathsf{P}}$-complete.*

Corollary 2.22. *Suppose $\mathcal{F}$ consists of Lipschitz-continuous ODEs over compact domains. The δ-decision problem for bounded $\mathcal{L}_{\mathcal{F}}$-sentences is PSPACE-complete.*

2.5 δ-Complete Decision Procedures

We now focus on bounded existential formulas. They form the class of problems often referred to as the SMT problems, whose detailed algorithms will be studied in Part III. This class of problems is a special case of the general δ-decidability results we have obtained in the previous chapters. Yet, it is worthwhile to give alternative proofs in this special case which do not rely much results in computable analysis and will also be useful for understanding practical algorithms. We have defined δ-strengthening and δ-weakening of general first-order sentences. We put formulas in normal forms that contains only inequalties. Yet in the case of Σ_1-sentences, it will be shown that using equalities is closer to the analysis of practical algorithms. Thus, we give a special definition for formulas using equalities in the normal forms.

Definition 2.23 (δ-Weakening of Σ_1-sentences). *Let $\delta\in\mathbb{Q}^+\cup\{0\}$ be a constant and φ be a Σ_1-sentence in standard form: $\varphi:=\exists^{\vec{I}}\vec{x}.\bigwedge_{i=1}^m(\bigvee_{j=1}^{k_i}f_{ij}(\vec{x})=0)$. The δ-weakening of φ defined as $\varphi^\delta:=\exists^{\vec{I}}\vec{x}.\bigwedge_{i=1}^m(\bigvee_{j=1}^{k}|f_{ij}(\vec{x})|\leq\delta)$. Also, a δ-perturbation is a constant vector $\vec{c}=(c_{11},...,c_{mk_m})$, $c_{ij}\in\mathbb{R}$, satisfying $||\vec{c}||\leq\delta$, such that the $\vec{c}$-perturbed form of φ is given by: $\varphi^{\vec{c}}:=\exists^{\vec{I}}\vec{x}.\bigwedge_{i=1}^m(\bigvee_{j=1}^{k}f_{ij}(\vec{x})=c_{ij})$.*

Definition 2.24 (Bounded δ-SMT in $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$). *Let $\mathcal{F}$ be a finite collection of Type 2 computable functions. Let φ be a bounded Σ_1-sentence in $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$ in standard form. The* bounded δ-SMT problem *asks for one of the following either* unsat *: φ is false, or δ-*sat *: φ^δ is true. When the two cases overlap, either decision can be returned.*

Definition 2.25 (δ-Completeness). *We say a procedure is δ-complete if it solves the δ-SMT problem correctly.*

3 Formal Verification of Hybrid Automata

There are two main difficulties in formal verification of hybrid systems. Theoretically, it is well known that the safety verification problem for hybrid systems with very simple dynamics is highly undecidable. Consequently, a unified framework for solving the reachability problem seems impossible, especially for nonlinear hybrid systems. Some decidability results exist for restricted classes of nonlinear systems but they do not apply to the majority of realistic systems. Practically, formal analysis of hybrid systems requires both numerical computation and logical reasoning. A core requirement is the availability of efficient solving techniques for handling logical combinations of assertions over the real numbers. To model physical laws and control systems, the logic formulas in question have to contain various nonlinear real functions. The logical decision problem for such formulas is extremely hard. The lack of efficient solving engines limits the scalability of existing verification techniques for hybrid systems. Advanced verification techniques, which are successful in other domains such as hardware and software verification, are hard to apply on hybrid systems.

We show that the framework of δ-decisions bring a new perspective on the problem. In practice, hybrid systems interact with the physical world and it is impossible to avoid slight perturbations; they may come from lack of precision in the sensors, errors in floating-point computation, physical disturbance in the environment, etc. Note that such robustness problems can not be discovered by solving the standard reachability problem. Thus, we argue that the notion of δ-reachability is not weaker, and in fact better suits the need of safety verification in practice.

3.1 Hybrid Automata and $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-Representations

Hybrid automata are finite automata combined with dynamical systems. We first show that $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas can concisely represent hybrid automata.

Definition 3.1 ($\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-Representations of Hybrid Automata)**.** *A hybrid automaton in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-representation is a tuple*

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

where $X \subseteq \mathbb{R}^n$ for some $n \in \mathbb{N}$, $Q = \{q_1, ..., q_m\}$ is a finite set of modes, and the other components are finite sets of quantifier-free $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas. We can write $H = \langle X, Q, \mathsf{flow}, \mathsf{jump}, \mathsf{inv}, \mathsf{init}\rangle$.

Intuitively, the flow formulas specify the rules for the continuous dynamics in each mode. The jump conditions specify when an automaton *may* switch to another mode. The invariants (when violated) specify when an automaton *must* switch to another mode. The init conditions specify the initial states of the system. We say a hybrid automaton H has a *computable representation*, if H has an $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-representation, and all functions in $\mathcal{F}$ are Type 2 computable. From now on we will only consider hybrid automata that have computable representations.

We have not restricted the form of the formulas for defining hybrid automata. This makes the definition more general than necessary. For instance, the flow should be a continuous mapping from $\vec{x}_0$ and t to $\vec{x}_t$ (and thus a conjunction of equations of the form $\vec{x}_t = f(\vec{x}_0, t)$), instead of arbitrary formulas. Different classes of hybrid systems can be defined by refining this definition. Again, $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ representations can contain almost all functions needed in describing hybrid systems, including nonlinear ODEs that have no analytic expressions. Most of the hybrid systems studied in the existing literature can be defined by restricting the signature $\mathcal{F}$, for instance:

Example 3.2 (Linear and Polynomial Hybrid Automata)**.** *Let $\mathcal{F}^{\text{lin}} = \{+\} \cup \mathbb{Q}$ and $\mathcal{F}^{\text{poly}} = \{\times\} \cup \mathcal{F}^{\text{poly}}$ (Rational numbers are considered as 0-ary functions.) We say a hybrid automaton H is a* linear hybrid automaton *if it has an $\mathcal{L}_{\mathbb{R}_{\mathcal{F}^{\text{lin}}}}$-representation, and a* polynomial hybrid automaton *if it has an $\mathcal{L}_{\mathbb{R}_{\mathcal{F}^{\text{poly}}}}$-representation.*

Trajectories of hybrid systems combine continuous flows and discrete jumps. This motivates the use of a hybrid time domain, with which we can keep track of both the discrete changes and the duration of each continuous flow. A hybrid time domain is a sequence of closed intervals on the real line, and a hybrid trajectory is a mapping from the time domain to the Euclidean space.

Definition 3.3 (Hybrid Time Domains and Hybrid Trajectories)**.** *A* hybrid time domain *is a subset of* $\mathbb{N} \times \mathbb{R}$ *of the form* $T_m = \{(i,t) : i < m \text{ and } t \in [t_i, t_i'] \text{ or } [t_i, +\infty)\}$, *where* $m \in \mathbb{N} \cup \{+\infty\}$, $\{t_i\}_{i=0}^m$ *is an increasing sequence in* $\mathbb{R}^+$, $t_0 = 0$, *and* $t_i' = t_{i+1}$. *When* $X \subseteq \mathbb{R}^n$ *is an Euclidean space and* T_m *a hybrid time domain, a* hybrid trajectory *is a* continuous *mapping* $\xi : T_m \to X$. *We can write the time domain* T_m *of* ξ *as* $T(\xi)$.

We can now define trajectories of hybrid automata. To link hybrid trajectories with automata, we need a labeling function $\sigma_{\xi,H}(i)$ that maps each step i in the hybrid trajectory to an appropriate discrete mode in H, and make sure that the $\mathsf{flow}, \mathsf{jump}, \mathsf{inv}, \mathsf{init}$ conditions are satisfied.

Definition 3.4 (Trajectories of Hybrid Automata)**.** *Let* H *be a hybrid automaton,* T_m *a hybrid domain, and* $\xi : T_m \to X$ *a hybrid trajectory. We say that* ξ *is* a trajectory of H of discrete depth m, *written as* $\xi \in [\![H]\!]$, *if there exists a* labeling function $\sigma_{\xi,H} : \mathbb{N} \to Q$ *such that:*

- *For some* $q \in Q$, $\sigma_{\xi,H}(0) = q$ *and* $\mathbb{R}_{\mathcal{F}} \models \mathsf{init}_q(\xi(0,0))$.
- *For any* $(i,t) \in T_m$, $\mathbb{R}_{\mathcal{F}} \models \mathsf{inv}_{\sigma_{\xi,H}(i)}(\xi(i,t))$.
- *For any* $(i,t) \in T_m$*: When* $i = 0$, $\mathbb{R}_{\mathcal{F}} \models \mathsf{flow}_{q_0}(\xi(0,0), \xi(0,t), t)$. *When* $i = k+1$, *where* $0 < k+1 < m$,

$$\mathbb{R}_{\mathcal{F}} \models \mathsf{flow}_{\sigma_\xi^H(k+1)}(\xi(k+1, t_{k+1}), \xi(k+1, t), (t - t_{k+1})), \text{and}$$
$$\mathbb{R}_{\mathcal{F}} \models \mathsf{jump}_{\sigma_{\xi,H}(k) \to \sigma_{\xi,H}(k+1)}(\xi(k, t_k'), \xi(k+1, t_{k+1})).$$

The definition is straightforward. In each mode, the system flows continuously following the dynamics defined by flow_q. Note that $(t - t_k)$ is the actual duration in the k-th mode. When a switch between two modes is performed, it is required that $\xi(k+1, t_{k+1})$ is updated from the exit value $\xi(k, t_k')$ in the previous mode, following the jump conditions.

Note that we gave no restriction on the formulas that can be used for describing hybrid automata in Definition 3.1. A minimal requirement is that the flow predicates should define continuous trajectories over time, namely:

Definition 3.5 (Well-Defined Flow Predicates)**.** *Let* $\mathsf{flow}(\vec{x}, \vec{y}, t)$ *be a flow predicate for a hybrid automaton* H. *We say the flow predicate is* well-defined, *if for all tuples* $(\vec{a}, \vec{b}, \tau) \in X(H) \times X(H) \times \mathbb{R}^{\geq 0}$ *such that* $\mathbb{R} \models \mathsf{flow}(\vec{a}, \vec{b}, \tau)$, *there exists a continuous function* $\eta : [0, \tau] \to X$ *such that* $\eta(0) = \vec{a}$, $\eta(\tau) = \vec{b}$, *and for all* $t' \in [0, \tau]$, *we have* $\mathbb{R} \models \mathsf{flow}(\vec{a}, \eta(t), t)$. *We say* H *is* well-defined *if all its flow predicates are well-defined.*

This definition requires that we can always construct a trajectory from the end points and the initial points that satisfy a flow predicate. Flows that are defined using differential equations, differential inclusions, and explicit continuous mappings all satisfy this condition. Thus, from now on our discussion of hybrid automata assume their well-definedness.

3.2 Reachability

Formally, the reachability problem for hybrid automata defined as follows.

Definition 3.6 (Reachability)**.** *Let H be an n-dimensional hybrid automaton, and U a subset of its state space $Q \times X$. We say U* is reachable by *H, if there exists $\xi \in [\![H]\!]$,such that there exists $(i,t) \in T(\xi)$ satisfying $(\sigma_\xi^H(i), \xi(i,t)) \in U$.*

The bounded reachability problem for hybrid systems is defined by restricting the continuous time duration to a bounded interval, and the number of discrete transitions to a finite number.

Definition 3.7 (Bounded Reachability)**.** *Let H be an n-dimensional hybrid automaton, whose continuous state space X is a bounded subset of $\mathbb{R}^n$. Let U be a subset of its state space. Set $k \in \mathbb{N}$ and $M \in \mathbb{R}^{\geq 0}$. The (k, M)-bounded reachability problem asks whether there exists $\xi \in [\![H]\!]$ such that there exists $(i,t) \in T(\xi)$ with $i \leq k$, $t = \sum_{i=0}^{k} t_i$ where $t_i \leq M$, and $(\sigma_\xi(i), \xi(i,t)) \in U$.*

By "step", we mean the number of discrete jumps. We say H can reach U in k steps, if there exists $\xi \in [\![H]\!]$ that contains k discrete jumps, which consists of $k+1$ pieces of continuous flows in the corresponding discrete modes.

3.3 Encoding Bounded Reachability in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$

We now define the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-encoding of bounded reachability. We need to define a set of auxiliary formulas that will be important for ensuring that a particular mode is picked at a certain step.

Definition 3.8. *Let $Q = \{q_1, ..., q_m\}$ be a set of modes. For any $q \in Q$, and $i \in \mathbb{N}$, use b_q^i to represent a Boolean variable. We now define $\mathsf{enforce}_Q(q,i) = b_q^i \wedge \bigwedge_{p \in Q \setminus \{q\}} \neg b_p^i$ and $\mathsf{enforce}_Q(q,q',i) = b_q^i \wedge \neg b_{q'}^{i+1} \wedge \bigwedge_{p \in Q \setminus \{q\}} \neg b_p^i \wedge \bigwedge_{p' \in Q \setminus \{q'\}} \neg b_{p'}^{i+1}$.*

We say a hybrid automaton H is *invariant-free* if $\mathsf{inv}_q(H) = \top$ for every $q \in Q(H)$. We use $\mathsf{unsafe} = \{\mathsf{unsafe}_q : q \in Q\}$ as the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-representation of an unsafe region in the state space of H. We can write $[\![\mathsf{unsafe}]\!] = \bigcup_{q \in Q} [\![\mathsf{unsafe}_q]\!] \times \{q\}$. We define the following formula that checks whether an unsafe region is reachable after exactly k steps of discrete transition in a hybrid system.

Definition 3.9 (k-Step Reachability, Invariant-Free Case)**.** *Suppose H is invariant-free, and U a subset of its state space represented by* unsafe*. The $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula $\mathsf{Reach}_{H,U}(k,M)$ is defined as:*

$$
\begin{aligned}
& \exists^X \vec{x}_0 \exists^X \vec{x}_0^t \cdots \exists^X \vec{x}_k \exists^X \vec{x}_k^t \exists^{[0,M]} t_0 \cdots \exists^{[0,M]} t_k. \\
& \bigvee_{q \in Q} \Big(\mathsf{init}_q(\vec{x}_0) \wedge \mathsf{flow}_q(\vec{x}_0, \vec{x}_0^t, t_0) \wedge \mathsf{enforce}(q,0) \Big) \\
\wedge \quad & \bigwedge_{i=0}^{k-1} \Bigg(\bigvee_{q,q' \in Q} \Big(\mathsf{jump}_{q \to q'}(\vec{x}_i^t, \vec{x}_{i+1}) \wedge \mathsf{enforce}(q,q',i) \wedge \mathsf{flow}_{q'}(\vec{x}_{i+1}, \vec{x}_{i+1}^t, t_{i+1}) \wedge \mathsf{enforce}(q', i+1) \Big) \Bigg) \\
\wedge \quad & \bigvee_{q \in Q} \mathsf{unsafe}_q(\vec{x}_k^t).
\end{aligned}
$$

Intuitively, the trajectories start with some initial state satisfying $\mathsf{init}_q(\vec{x}_0)$ for some q. In each step, it follows $\mathsf{flow}_q(\vec{x}_i, \vec{x}_i^t, t)$ and makes a continuous flow from $\vec{x}_i$ to $\vec{x}_i^t$ after time t. When H makes a jump from mode q' to q, it resets variables following $\mathsf{jump}_{q' \to q}(\vec{x}_k^t, \vec{x}_{k+1})$. The auxiliary $\mathsf{enforce}$ formulas ensure that picking $\mathsf{jump}_{q \to q'}$ in the i-the step enforces picking flow_q' in the $(i+1)$-th step.

The case of nontrivial invariants and nondeterministic flows require more quantifiers. When the invariants are not trivial, we need to ensure that for all the time points along a continuous flow, the invariant condition holds. Thus, we need to universally quantify over time. For deterministic flows, we have:

Definition 3.10 (k-Step Reachability, Nontrivial Invariant and Deterministic Flow)**.** *Suppose H contains invariants and only deterministic flow , and U a subset of its state space represented by* unsafe*. In this case, the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula $\mathsf{Reach}_{H,U}(k,M)$ is defined as:*

$$
\begin{aligned}
& \exists^X \vec{x}_0 \exists^X \vec{x}_0^t \cdots \exists^X \vec{x}_k \exists^X \vec{x}_k^t \exists^{[0,M]} t_0 \cdots \exists^{[0,M]} t_k. \\
& \bigvee_{q \in Q} \Big(\mathsf{init}_q(\vec{x}_0) \wedge \mathsf{flow}_q(\vec{x}_0, \vec{x}_0^t, t_0) \wedge \mathsf{enforce}(q,0) \wedge \forall^{[0,t_0]} t \forall^X \vec{x} \; (\mathsf{flow}_q(\vec{x}_0, \vec{x}, t) \rightarrow \mathsf{inv}_q(\vec{x})) \Big) \\
\wedge \quad & \bigwedge_{i=0}^{k-1} \Bigg(\bigvee_{q,q' \in Q} \Big(\mathsf{jump}_{q \rightarrow q'}(\vec{x}_i^t, \vec{x}_{i+1}) \wedge \mathsf{flow}_{q'}(\vec{x}_{i+1}, \vec{x}_{i+1}^t, t_{i+1}) \wedge \mathsf{enforce}(q, q', i) \\
& \qquad \wedge \mathsf{enforce}(q', i+1) \wedge \forall^{[0,t_{i+1}]} t \forall^X \vec{x} \; (\mathsf{flow}_{q'}(\vec{x}_{i+1}, \vec{x}, t) \rightarrow \mathsf{inv}_{q'}(\vec{x})) \Big) \Bigg) \\
\wedge \quad & \bigvee_{q \in Q} (\mathsf{unsafe}_q(\vec{x}_k^t) \wedge \mathsf{enforce}(q,k)).
\end{aligned}
$$

The extra universal quantifier for each continuous flow expresses the requirement that for all the time points between the initial and ending time point ($t \in [0, t_i + 1]$) in a flow, the continuous variables $\vec{x}$ must take values that satisfy the invariant conditions $\mathsf{inv}_q(\vec{x})$. For the case of nondeterministic flows, one needs to quantify over possible choices for each time point, and the details are in the thesis.

3.4 δ-Complete Analysis of Bounded Reachability

We now define the δ-complete analysis problem and prove its decidability.

Definition 3.11 (δ-Weakening of Hybrid Automata)**.** *Let $\delta \in \mathbb{Q}^+ \cup \{0\}$ be arbitrary. Let H be a hybrid automaton in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-representation. The* δ-weakening *of H is $H^\delta = \langle X, Q, \mathsf{flow}^\delta, \mathsf{jump}^\delta, \mathsf{inv}^\delta, \mathsf{init}^\delta \rangle$ which is obtained by weakening all formulas in the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-representations of H.*

Definition 3.12 (Bounded δ-Reachability)**.** *Let H be a hybrid system and U a subset of its state space. Suppose U is represented by the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula* unsafe*. Let $k \in \mathbb{N}$ and $M \in \mathbb{R}^+$. The δ-complete analysis for (k,M)-bounded reachability problem asks for one of the following answers:*

- *(k,m)-Safety: H does not reach $[\![\mathsf{unsafe}]\!]$ within the (k,M)-bound.*
- *δ-Unsafety: H^δ reaches $[\![\mathsf{unsafe}^\delta]\!]$ within the (k,M)-bound.*

It is straightforward to transfer |*delta*-decidability of the formulas to the decidability of bounded δ-reachability.

Lemma 3.13. *Let $\delta \in \mathbb{Q}^+ \cup \{0\}$ be arbitrary. Suppose H is a well-defined hybrid automaton with strictly-imposed invariants. Let U a subset of the state space of H, represented by the set* unsafe *of $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas. Let $\mathsf{Reach}_{H,U}(k,M)$ be the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula encoding (k,M)-bounded reachability of H with respect to U. We always have that $\mathbb{R} \models (\mathsf{Reach}_{H,U}(k,M))^\delta$ iff there exists a trajectory $\xi \in [\![H^\delta]\!]$ such that for some $(k,t) \in T(\xi)$, where $0 \le t \le M$, $(\xi(k,t), \sigma_\xi(k)) \in [\![\mathsf{unsafe}^\delta]\!]$.*

Theorem 3.14 (Decidability)**.** *Let $\delta \in \mathbb{Q}^+$ be arbitrary. There exists an algorithm such that, for any bounded well-defined hybrid automaton $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-represented by H, and any unsafe region U $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-represented by* unsafe*, correctly performs δ-complete analysis for (k,M)-bounded reachability for H, for any $k \in \mathbb{N}, M \in \mathbb{R}^+$.*

Theorem 3.15 (Complexity)**.** *Suppose all the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-terms in the description of H and U are in complexity class* C*. Then deciding the (k, M)-bounded δ-reachability problem is in*

- NP^{C} *for an invariant-free H;*
- $(\Sigma_2^P)^{\mathsf{C}}$ *for an H with nontrivial invariants and deterministic flows;*
- $(\Sigma_3^P)^{\mathsf{C}}$ *for an H with nontrivial invariants and nondeterministic flows.*

Corollary 3.16. *For linear and polynomial hybrid automata, δ-complete bounded reachability analysis ranges from being* NP*-complete to Σ_2^{P}-complete for the three cases. For hybrid automata that can be $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-represented with whose $\mathcal{F}$ contains the set of ODEs defined* P*-computable right-hand side functions, the problem is* PSPACE*-complete.*

The results come from the fact that the complexity of polynomials is in P, and the set of ODEs in questions are PSPACE-complete. The complexity results indicate that the worst-case running time of the analysis is exponential in all the input parameters. In particular, the worst-case running time grows exponentially with the δ and the size of the domains. We need to use efficient decision procedures to manage this complexity.

4 Practical δ-Complete Decision Procedures

We describe practical algorithms for computing δ-decisions of SMT problems in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$. The new framework allows us to exploit the full power of numerical methods in decision problems and formal verification. We show that δ-completeness serve as a reasonable performance requirement for numerically-driven procedures to be used in formal verification. We show an analysis of the powerful constraint solving framework Interval Constraint Propagation (ICP), obtaining conditions under which it is δ-complete. In particular, we develop decision procedures for formulas that contain ODEs. 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). We also give a proof calculus that can be used to validate the correctness of unsat results given by DPLL(ICP). Such proofs ensure the correctness of the unsat without numerical errors.

4.1 The DPLL$\langle$ICP$\rangle$ Framework

Current SMT solvers are mostly built on the DPLL(T) framework. An SMT problem is a quantifier-free first order formula φ with atomic formulas specified by some theory T. The DPLL(T) approach requires the use of a SAT solver and a theory solver (T-solver). The SAT solver is first applied on the Boolean abstraction φ^B of the formula φ, which is the propositional formula with all the theory atoms replaced by propositional variables. If φ^B is Boolean-satisfiable, the SAT solver should return a satisfying assignment for all the Boolean variables. Since the Boolean variables correspond to theory atoms, we need to further check whether the Boolean assignment is consistent with the theory T. This task calls for the use of a T-solver, which takes a set of theory atoms as input and checks whether the set is consistent. If the set is consistent, the original formula φ is satisfiable; otherwise, the corresponding Boolean assignment is spurious, and the SAT solver looks for a different satisfying assignment. The process is iterated until a real solution is found, or all the Boolean solutions have been exhausted and the formula is determined unsatisfiable.

The main utility of the T-solver, which decides whether a set of theory atoms is consistent, is provided by the Check() procedure. A partial check procedure, named Assert(), is also used alongside the SAT solver, which takes

each incomplete set of theory atoms asserted by the SAT solver and returns whether it already contains an evident conflict. The Assert() procedure should work cheaply and incrementally. Usually, Check() is supposed to be sound and complete while Assert() only has to be sound. We call both Assert() and Check() the checking procedures of the theory solver. To achieve efficiency, it is not enough to have sound and complete checking procedures only. When a set of asserted theory atoms is determined to be inconsistent, explanations for such inconsistency should be provided, so that a clause can be learned for refining the search space. When conflicts occur, the SAT solver needs to backtrack, and the T-solver should also provide methods for efficient backtracking on the theory atoms.

Interval Constraint Propagation. The method of Interval Constraint Propagation (ICP) [8] 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 4.1 (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 $\mathbb{IF} = \{[a, b] \subseteq \mathbb{R} : a, b \in \mathbb{F}, a \leq b\}$ 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: $\mathsf{Hull}(S) = \bigcap\{B \in \mathbb{BF} : S \subseteq B\}$.*

Definition 4.2 (Interval Extension [8])**.** *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 $\{f(\vec{x}) : \vec{x} \in B\} \subseteq \sharp f(B)$.*

The idea of interval constraint propagation is to use 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. A high-level description of the decision version of ICP is given in Algorithm 1. In the Algorithm 1, 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 $I_i \subseteq I_i' \cup I_i''$. To ensure termination, it is required that there exists some constant $c \in (0, 1)$ such that $|I_i'| \geq c \cdot |I_i|$ and $|I_i''| \geq c \cdot |I_i|$. The key component of the ICP algorithm is the Prune(B, f) operation in Algorithm 1.

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

$S \leftarrow B_0$
while $S \neq \emptyset$ **do**
  $B \leftarrow S$.pop()
  while $\exists 1 \leq i \leq m, B \neq_\delta$ Prune(B, f_i) **do**
    $B \leftarrow$ Prune(B, f_i)
  end while
  if $B \neq \emptyset$ **then**
    if $\exists 1 \leq i \leq n, |\sharp f_i(B)| \geq \delta$ **then**
      $\{B_1, B_2\} \leftarrow$ Branch(B, i)
      S.push$(\{B_1, B_2\})$
    else
      return sat
    end if
  end if
end while
return unsat

4.2 δ-Completeness of DPLL$\langle$ICP$\rangle$

We formulate the conditions under with the DPLL$\langle$ICP$\rangle$ algorithm is δ-complete. The key component of the ICP algorithm is the Prune(B, f) operation. In principle, any operation that contracts the intervals on variables can be seen as pruning. However, for correctness we need several formal requirements on the pruning operator in ICP.

Definition 4.3 (Well-defined Pruning Operators)**.** *Let $\mathcal{F}$ be a collection of real functions, and $\sharp$ be an interval extension operator on $\mathcal{F}$. A* well-defined (equality) pruning operator *with respect to $\sharp$ is a partial function* $\text{Prune}_\sharp :\subseteq \mathbb{BF} \times \mathcal{F} \to \mathbb{BF}$*, such that $\forall f \in \mathcal{F}$, $B, B' \in \mathbb{BF}$,*

- *(W1)* $\text{Prune}_\sharp(B, f) \subseteq B$*;*
- *(W2) If* $(\text{Prune}_\sharp(B, f)) \neq \emptyset$*, then* $0 \in \sharp f(\text{Prune}_\sharp(B, f))$*.*
- *(W3)* $B \cap Z_f \subseteq \text{Prune}_\sharp(B, f)$*;*

Intuitively, (W1) requires contraction, so that the algorithm always makes progress: branching always decreases the size of boxes, and pruning never increases them. (W2) requires that the result of a pruning is always a reasonable box that may contain a zero. Otherwise B should have been pruned out. (W3) ensures that the real solutions are never discarded in pruning (called "completeness" in [8]). It is clear from the description of Algorithm 1 that the following properties hold.

Lemma 4.4. *Algorithm 1 always terminates. If it returns* sat *then there exists nonempty boxes $B, B' \subseteq B_0$, such that $||B|| < \varepsilon$ and $B = \text{Prune}(B', f_1, ..., f_m)$. If it returns* unsat *then $\forall \vec{a} \in B_0$, there exists $B \subseteq B_0$ such that $\vec{a} \in B$ and $\text{Prune}(B, f_1, ..., f_m) = \emptyset$.*

Theorem 4.5 (δ-Completeness of ICP)**.** *Let $\delta \in \mathbb{Q}^+$ be arbitrary. The* ICP_ε *algorithm is δ-complete for conjunctive Σ_1-sentences in $\mathcal{L}_\mathcal{F}$ (where* sat *is interpreted as δ-*sat*) if and only if the pruning operator in ICP_ε is well-defined.*

4.3 Handling Differential Equations

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. To define the pruning operators, we need to use the interval extensions of the solution functions of ODE systems. Let $D \subseteq \mathbb{R}^n$ be compact and $g_i : D \to \mathbb{R}$ be n Lipschitz-continuous functions. Given the first-order autonomous ODE system $\frac{d\vec{y}}{dt} = \vec{g}(\vec{y}(t, \vec{x}_0))$ and $\vec{y}(0, \vec{x}_0) = \vec{x}_0$ where $\vec{x}_0 \in D$, we write $y_i : [0, T] \times D \to \mathbb{R}$ to represent the i-th solution function of the ODE system. The 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 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 $\{x_t \in \mathbb{R} : x_t = y_i(t, \vec{x}_0), \vec{x}_0 \in B_{\vec{x}_0}, t \in I_t\} \subseteq \sharp y_i(I_t, B_{\vec{x}_0})$. We define the pruning operators based on the interval extensions of the ODE solution functions. 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. For instance, forward pruning can be defined as follows. Gven interval assignments on $\vec{x}_0$ and t, we compute a refinement of the interval assignments on $\vec{x}_t$. Let $\vec{y} : [0, T] \times D \to \mathbb{R}^n$ 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. We define the forward-pruning operator as:

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

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.

Proposition 4.6. *The three ODE pruning operators are well-defined.*

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.

4.4 From Decisions to Proofs

We focus on the proof the unsatisfiability of conjunctions of theory atoms in the DPLL(T) framework, i.e., formulas of the form $\exists^{I_1}x_1 \cdots \exists^{I_n}x_n.\ \bigwedge_{i=1}^{m} f_i(x_1, ..., x_n) \sim 0$ where $\sim \in \{=, \neq, >, \geq, <, \leq\}$. It is clear that once such proofs are obtained, the proof of unsatisfiability of Boolean combinations of the theory atoms can be obtained, by simply plugging them in the high level resolution proof. Also, it is important to note that the ICP algorithm solves *systems* of constraints, and it regards the conjunction $\bigwedge_{i=1}^{m} f_i(x_1, ..., x_n) \sim 0$ as one constraint $c(x_1, ..., x_n)$. Consequently, our task is now reduced to obtaining proofs for the validity of formulas of the form $\forall x_1 \cdots \forall x_n.(x_1 \in I_1 \wedge \cdots \wedge x_n \in I_n) \rightarrow \neg c(\vec{x})$, from the failure of ICP search for a solution to the original SMT formula $\exists \vec{x}.\vec{x} \in \vec{I} \wedge c(\vec{x})$. We construct a simple first-order proof calculus $\mathbb{D}_A$, and show how to transform ICP runs into proofs in the system. The main fact is:

Proposition 4.7. *For every ICP run ending with a contradiction, the tree construction procedure produces a valid natural deduction tree for the negation of the input formula in $\mathbb{D}_A$. The size of the proofs is linear in the computation steps.*

Note that once the proof tree is constructed, the details of the ICP algorithm itself no longer matters.

5 Implementation and Experiments

In this chapter, we present our tools dReal and dReach that implement the algorithms described in the preceding chapters. We show experimental results on formula benchmarks and nonlinear hybrid systems models that are beyond the scope of existing tools. dReal is an SMT solver for formulas over the reals that can handle various nonlinear elementrary functions in the framework of δ-complete decision procedures. It returns unsat or δ-sat on input formulas, where δ can be specified by the user. When the answer is unsat, dReal produces a proof of unsatisfiability. When the result is δ-sat, it provides a solution such that a δ-perturbed form of the input formula is satisfied. The tool is built based on OpenSMT for DPLL(T) framework, RealPaver for ICP, and CAPD for reliable integration of ODEs. dReach is a bounded model checker for hybrid systems. Both tools are open-source, available at http://dreal.cs.cmu.edu. dReal has scaled on benchmarks that contain hundreds of nonlinear ODEs or transcendental functions. The tools have performed verification on various highly nonlinear hybrid system models that arise in practical applications. Details are given in the thesis.

6 Summary

This thesis developed a new framework for the formal verification of CPS. We start with a new way of defining core decision problem of the logic formulas used in verification. We show that such requiring solving the logic formulas symbolically and precisely is an overkill for reasoning about hybrid systems. In fact, an appropriate relaxation of the exact decision problems leads to strong decidability and complexity results, both for the decision problems of the logic formulas and the reachability problems of hybrid automata, that suggest a very different and positive outlook

for the field. The key step is to develop a concept of numerical approximations in the standard decision problems. We do this by defining the notion of δ-perturbations on logic formulas, where δ is any positive rational number. A δ-perturbation on a formula is a syntactic variant of it. Using this notion, we are able to define the *δ-decision problems*: for a formula φ, we ask whether φ is true or a δ-perturbation of φ is false (or the other way around). The key theorem is that the δ-decision problem is decidable for sentences with bounded quantifiers in first-order theories over the reals with any "numerically computable" real functions. Such a notion of numerical computability has been studied extensively in the field of computable analysis, and all common continuous functions are computable. This stands in sharp contrast with the undecidability results for the first-order theory containing trignometric functions.

Next, we show that δ-decidability leads to a new perspective on hybrid system verification. First, we can describe hybrid automata using arbitrary first-order formulas with computable functions. This leads to general definitions that are broad enough to express almost all systems of practical relevance. We then show that a notion of δ-reachability can be defined for hybrid automata using exactly the δ-perturbations on first-order formulas. This leads to positive decidability and complexity results for solving bounded δ-reachability through bounded model checking, relying on algorithms that solve the δ-decision problems. We have similar positive results for inductive invariant validation problems. In all, δ-decisions allow us to circumvent mmuch of the theoretical difficulty in the field.

In practice, the new framework allows us to exploit the full power of numerical methods in decision problems and formal verification. We show that δ-completeness serve as a reasonable performance requirement for numerically-driven procedures to be used in formal verification. We show an analysis of the powerful constraint solving framework Interval Constraint Propogation (ICP), obtaining conditions under which it is δ-complete. We have consequently built a practical tool dReal combining ICP in the DPLL(T) framework, which can solve formulas in a δ-complete way for a very general nonlinear signature that inclueds transcendental functions and ODEs. This tool is the backend of a hybrid system verification tool dReach, which solved challenging nonlinear hybrid automata models beyond the reach of existing tools.

References

[1] B. Akbarpour and L. C. Paulson. Metitarski: An automatic prover for the elementary functions. In *AISC/MKM/Calculemus*, pages 217–231, 2008.

[2] B. Akbarpour and L. C. Paulson. Metitarski: An automatic theorem prover for real-valued special functions. *J. Autom. Reasoning*, 44(3):175–205, 2010.

[3] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1992.

[4] R. Alur and D. L. Dill. The theory of timed automata. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer, 1991.

[5] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Software Eng.*, 22(3):181–201, 1996.

[6] E. Asarin, T. Dang, O. Maler, and O. Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In *HSCC*, pages 20–31, 2000.

[7] J. Avigad and H. Friedman. Combining decision procedures for the reals. *Logical Methods in Computer Science*, 2(4), 2006.

[8] 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.

[9] C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell, and A. Rubio. Solving non-linear polynomial arithmetic via sat modulo linear arithmetic. In *CADE*, pages 294–305, 2009.

[10] O. Bournez. Achilles and the tortoise climbing up the hyper-arithmetical hierarchy. *Theor. Comput. Sci.*, 210(1):21–71, 1999.

[11] C. W. Brown and J. H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *ISSAC-2007*.

[12] A. Chutinan and B. H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *HSCC*, pages 76–90, 1999.

[13] G. E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, pages 134–183, 1975.

[14] B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV-2006*.

[15] A. Eggers, M. Fränzle, and C. Herde. Sat modulo ode: A direct sat approach to hybrid systems. In *ATVA*, pages 171–185, 2008.

[16] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT*, 1(3-4):209–236, 2007.

[17] M. K. Ganai and F. Ivančić. Efficient decision procedure for non-linear arithmetic constraints using cordic. In *Formal Methods in Computer Aided Design (FMCAD)*, 2009.

[18] S. Gao, M. Ganai, F. Ivancic, A. Gupta, S. Sankaranarayanan, and E. Clarke. Integrating icp and lra solvers for deciding nonlinear real arithmetic. In *FMCAD*, 2010.

[19] A. Girard, C. L. Guernic, and O. Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In J. P. Hespanha and A. Tiwari, editors, *HSCC*, volume 3927 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2006.

[20] C. L. Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 540–554. Springer, 2009.

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

[22] C. Munoz and A. Narkawicz. Formalization of an efficient representation of bernstein polynomials and applications to global optimization. http://shemesh.larc.nasa.gov/people/cam/Bernstein/.

[23] P. Nuzzo, A. Puggelli, S. A. Seshia, and A. L. Sangiovanni-Vincentelli. Calcs: Smt solving for non-linear convex constraints. In R. Bloem and N. Sharygina, editors, *FMCAD*, pages 71–79. IEEE, 2010.

[24] G. O. Passmore and P. B. Jackson. Combined decision techniques for the existential theory of the reals. In *Calculemus/MKM*, pages 122–137, 2009.

[25] A. Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010.

[26] A. Platzer. Differential dynamic logics. *KI*, 24(1):75–77, 2010.

[27] A. Platzer. Logics of dynamical systems. In *LICS*, pages 13–24, 2012.

[28] A. Platzer and J.-D. Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2008.

[29] A. Platzer, J.-D. Quesel, and P. Rümmer. Real world verification. In *CADE*, pages 485–501, 2009.

[30] S. Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In K. H. Johansson and W. Yi, editors, *HSCC*, pages 221–230. ACM ACM, 2010.