δ-Complete Decision Procedures for Satisfiability over the Reals[⋆]

Sicun Gao, Jeremy Avigad, and Edmund M. Clarke

Carnegie Mellon University, Pittsburgh, PA 15213

Abstract. We introduce the notion of "δ-complete decision procedures" for solving SMT problems over the real numbers, with the aim of handling a wide range of nonlinear functions including transcendental functions and solutions of Lipschitz-continuous ODEs. Given an SMT problem φ and a positive rational number δ, a δ-complete decision procedure determines either that φ is unsatisfiable, or that the "δ-weakening" of φ is satisfiable. Here, the δ-weakening of φ is a variant of φ that allows δ-bounded numerical perturbations on φ. We prove the existence of δ-complete decision procedures for bounded SMT over reals with functions mentioned above. For functions in Type 2 complexity class C, under mild assumptions, the bounded δ-SMT problem is in NP^{C}. This stands in sharp contrast to the well-known undecidability results. δ-Complete decision procedures can exploit scalable numerical methods for handling nonlinearity, and we propose to use this notion as an ideal requirement for numerically-driven decision procedures. As a concrete example, we formally analyze the DPLL$\langle$ICP$\rangle$ framework, which integrates Interval Constraint Propagation (ICP) in DPLL(T), and establish necessary and sufficient conditions for its δ-completeness. We discuss practical applications of δ-complete decision procedures for correctness-critical applications including formal verification and theorem proving.

1 Introduction

Given a first-order signature $\mathcal{L}$ and a structure $\mathcal{M}$, the *Satisfiability Modulo Theories* (SMT) problem asks whether a quantifier-free $\mathcal{L}$-formula is satisfiable over $\mathcal{M}$, or equivalently, whether an existential $\mathcal{L}$-sentence is true in $\mathcal{M}$. Solvers for SMT problems have become the key enabling technology in formal verification and related areas. SMT problems over the real numbers are of particular interest, because of their importance in verification and design of hybrid systems, as well as in theorem proving. While efficient algorithms [10] 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 [9] for formulas with nonlinear polynomials have very high complexity [6]. When the sine function is involved, the SMT problem is undecidable, and only partial algorithms can be developed [2,1].

Recently much attention has been given to developing practical solvers that incorporate scalable numerical computations. Examples of numerical algorithms that have been exploited include optimization algorithms [4,27], interval-based algorithms [13,11,12,16], Bernstein polynomials [25], and linearization algorithms [14]. These solvers have shown promising results on various nonlinear benchmarks in terms of scalability.

However, for correctness-critical problems, there is always the concern that numerical errors can result in incorrect answers from numerically-driven solvers. For example, safety problems for hybrid systems can not be decided by numerical methods [28]. The problem is compounded by,

⋆ This research was sponsored by the National Science Foundation grants no. DMS1068829, no. CNS0926181 and no. CNS0931985, the GSRC under contract no. 1041377 (Princeton University), the Semiconductor Research Corporation under contract no. 2005TJ1366, General Motors under contract no. GMCMUCRLNV301, and the Office of Naval Research under award no. N000141010188.

for instance, the difficulty in understanding the effect of floating-point arithmetic in place of exact computation. There are two common ways of addressing these concerns. One is to use exact versions of the numerical algorithms, replacing floating-point operations by exact symbolic arithmetic [25]; the other is to use post-processing (validation) procedures to ensure that only correct results are returned. Both options reduce the full power of numerical algorithms and are usually hard to implement as well. For instance, in the Flyspeck project [18] for the formal proof of the Kepler conjecture, validating the numerical procedures used in the original proof turns out to be the hardest computational part (and unfinished yet). In general, there has been no framework for understanding the actual performance guarantees of numerical algorithms in the context of decision problems.

In this paper we aim to fill this gap by formally establishing the applicability of numerical algorithms in decision procedures, and the correctness guarantees they can actually provide. We do this as follows.

First, we introduce "the δ-SMT problem" over real numbers, to capture what can in fact be *correctly* solved by numerically-driven procedures. Given an SMT formula φ, and any positive rational number δ, the δ-SMT problem asks for one of the following decisions:

– unsat: φ is unsatisfiable.
– δ-sat: The δ-*weakening* of φ is satisfiable.

Here, the δ-weakening of φ is defined as a numerical relaxation of the original formula. For instance, the δ-weakening of $x = 0$ is $|x| \leq \delta$. Note that if a formula is satisfiable, its δ-weakening is always satisfiable. Thus, when a formula is δ-sat, either it is indeed satisfiable, or it is unsatisfiable but a δ-perturbation on its numerical terms would make it satisfiable. The effect of this slight relaxation is striking. In sharp contrast to the undecidability of SMT for any signature extending real arithmetic by sine, we show that the bounded δ-SMT problem for a wide range of nonlinear functions is decidable. In fact, we show that the bounded δ-SMT problem for the theory with exponentiation and trigonometric functions is NP-complete, and PSPACE-complete for theories with Lipschitz-continuous ODEs. We obtain these results using techniques from computable analysis [30,5]. These results serve as the theoretical basis for our investigation of numerically-driven procedures.

Next, if a decision algorithm can solve the δ-SMT problem correctly, we say it is "δ-complete". We propose to use δ-completeness as the ideal correctness requirement on numerically-driven procedures, replacing the conventional notion of complete solvers (which can never be met in this context). This new notion makes it worthwhile to develop formally analyze numerical methods for decision problems and compare their strength, instead of viewing them as partial heuristics. As an example, we study DPLL$\langle$ICP$\rangle$, the integration of Interval Constraint Propagation (ICP) [19] in DPLL(T) [24]. It is a general solving framework for nonlinear formulas and has shown promising results [13,16,12]. We obtain conditions that are sufficient and necessary for the δ-completeness of DPLL$\langle$ICP$\rangle$.

Further, we show the applicability of δ-complete procedures in correctness-critical practical problems. In bounded model checking [7,8], using a δ-complete solver we return one of the following answers: either a system is absolutely safe up to some depth (unsat answers), or it would *become unsafe* under some δ-bounded numerical perturbations (δ-sat answers). Since δ can be made very small, in the latter case the algorithm is essentially detecting robustness problems in the system: If a system would be unsafe under some small perturbations, it can hardly be regarded as safe in practice. Similar guarantees can be given for invariant validation and theorem proving. The conclusion is that, under suitable interpretations, the answers of numerically-driven decision procedures can indeed be relied on in correctness-critical applications, as long as they are δ-complete.

Related Work. Our goal is to provide a formal basis for the promising trend of numerically-driven decision procedures [4,27,13,11,12,16,25,14]. Related attempt can be seen in Ratschan's work [29], in which he investigated the stability of first-order constraints under numerical perturbations. Our approach is, instead, to take numerical perturbations as a given and study its implications in practical applications. Results in this paper are related to our more theoretical results [15] for arbitrarily-quantified sentences, where we do not analyze practical procedures. A preliminary notion of δ-completeness was proposed by us earlier in [16] where only polynomials are considered.

The paper is organized as follows. In Section 2 and 3 we define the bounded δ-SMT problem and establish its decidability and complexity. In Section 4 we formally analyze DPLL$\langle$ICP$\rangle$ and discuss applications in Section 5.

2 SMT with Type 2 Computable Functions

2.1 Basics of Computable Analysis

Real numbers can be encoded as infinite strings, and a computability theory of real functions can be developed with oracle machines that perform operations using function-oracles encoding real numbers. This is the approach developed in Computable Analysis or Type 2 Computability [30,22,5]. We briefly review results of importance to us.

Throughout the paper we use $||\cdot||$ to denote $||\cdot||_\infty$ over $\mathbb{R}^n$ for various n.

Definition 2.1 (Names). *A name of $a \in \mathbb{R}$ is any function $\gamma_a : \mathbb{N} \to \mathbb{Q}$ satisfying that $\forall i \in \mathbb{N}, |\gamma_a(i) - a| < 2^{-i}$. For $\boldsymbol{a} \in \mathbb{R}^n$, $\gamma_{\boldsymbol{a}}(i) = \langle \gamma_{a_1}(i), ..., \gamma_{a_n}(i)\rangle$.*

Thus the name of a real number is a sequence of rational numbers converging to it. For $\boldsymbol{a} \in \mathbb{R}^n$, we write $\Gamma(\boldsymbol{a}) = \{\gamma : \gamma \text{ is a name of } \boldsymbol{a}\}$.

A real function f is computable if there is an oracle Turing machine that can take any argument x of f as a function oracle, and output the value of $f(x)$ up to an arbitrary precision.

Definition 2.2 (Computable Functions). *We say $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ is computable if there exists a function-oracle Turing machine $\mathcal{M}_f$, outputting rational numbers, such that $\forall \boldsymbol{x} \in \mathrm{dom}(f)$ $\forall \gamma_{\boldsymbol{x}} \in \Gamma(\boldsymbol{x})$ $\forall i \in \mathbb{N}, |M_f^{\gamma_{\boldsymbol{x}}}(i) - f(\boldsymbol{x})| < 2^{-i}$.*

In the definition, i specifies the desired error bound on the output of M_f with respect to $f(\boldsymbol{x})$. For any $\boldsymbol{x} \in \mathrm{dom}(f)$, M_f has access to an oracle encoding the name $\gamma_{\boldsymbol{x}}$ of $\boldsymbol{x}$, and output a 2^{-i}-approximation of $f(\boldsymbol{x})$. In other words, the sequence $M_f^{\gamma_{\boldsymbol{x}}}(1), M_f^{\gamma_{\boldsymbol{x}}}(2), ...$ is a name of $f(\boldsymbol{x})$. Intuitively, f is computable if an arbitrarily good approximation of $f(\boldsymbol{x})$ can be obtained using any good enough approximation to any $\boldsymbol{x} \in \mathrm{dom}(f)$. A key property of this notion of computability is that computable functions over reals are continuous [30]. Moreover, over any compact set $D \subseteq \mathbb{R}^n$, computable functions are uniformly continuous with a *computable modulus of continuity* defined as follows.

Definition 2.3 (Uniform Modulus of Continuity). *Let $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ be a function and $D \subseteq \mathrm{dom}(f)$ a compact set. The function $m_f : \mathbb{N} \to \mathbb{N}$ is called a uniform modulus of continuity of f on D if $\forall \boldsymbol{x}, \boldsymbol{y} \in D$, $\forall i \in \mathbb{N}$, $||\boldsymbol{x} - \boldsymbol{y}|| < 2^{-m_f(i)}$ implies $|f(\boldsymbol{x}) - f(\boldsymbol{y})| < 2^{-i}$.*

Proposition 2.1 ([30]). *Let $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ be computable and $D \subseteq \mathrm{dom}(f)$ a compact set. Then f has a computable uniform modulus of continuity over D.*

Intuitively, if a function has a computable uniform modulus of continuity, then fixing any desired error bound 2^{-i} on the outputs, we can compute a *global* precision $2^{-m_f(i)}$ on the inputs from D such that using any $2^{-m_f(i)}$-approximation of any $\boldsymbol{x} \in D$, $f(\boldsymbol{x})$ can be computed within the error bound.

Most common continuous real functions are computable [30]. Addition, multiplication, absolute value, min, max, exp, sin and solutions of Lipschitz-continuous ordinary differential equations are all computable functions. Compositions of computable functions are computable.

Moreover, complexity of real functions can be defined over compact domains.

Definition 2.4 ([23]). *Let $D \subseteq \mathbb{R}^n$ be compact. A real function $f : D \to \mathbb{R}$ is* P*-computable (*PSPACE*-computable), if it is computable by an oracle Turing machine $M_f^{\gamma(\boldsymbol{x})}(i)$ that halts in polynomial-time (polynomial-space) for every $i \in \mathbb{N}$ and every $\boldsymbol{x} \in \mathrm{dom}(f)$.*

We say f is in Type 2 complexity class C if it is C-computable. f is C-complete if it is C-computable and C-hard [22]. If $f : D \to \mathbb{R}$ is C-computable, then it has a C-computable modulus of continuity over D. Polynomials, exp, and sin are all P-computable functions. A recent result [21] established that the complexity of computing solutions of Lipschitz-continuous ODEs over compact domains is a PSPACE-complete problem.

2.2 Bounded SMT over $\mathbb{R}_{\mathcal{F}}$

We now let $\mathcal{F}$ denote an arbitrary collection of Type 2 computable functions. $\mathcal{L}_{\mathcal{F}}$ denotes the first-order signature and $\mathbb{R}_{\mathcal{F}}$ is the standard structure $\langle \mathbb{R}, \mathcal{F} \rangle$. We can then consider the SMT problem over $\mathbb{R}_{\mathcal{F}}$, namely, satisfiability of quantifier-free $\mathcal{L}_{\mathcal{F}}$-formulas over $\mathbb{R}_{\mathcal{F}}$. We consider formulas whose variables take values from bounded intervals. Because of this, it is more convenient to directly write the bounds on existential quantifiers and express bounded SMT problems as Σ_1-sentences with bounded quantifiers.

Definition 2.5 (Bounded Σ_1-Sentences). *A bounded Σ_1-sentence in $\mathcal{L}_F$ is*

$$\varphi : \ \exists^{I_1} x_1 \cdots \exists^{I_n} x_n . \psi(x_1, ..., x_n).$$

- *For all i, $I_i \subseteq \mathbb{R}$ is a bounded (open or closed) interval with rational endpoints.*
- *Each bounded quantifier $\exists^{I_i} x_i . \phi$ denotes $\exists x_i . (x_i \in I_i \wedge \phi)$.*
- *$\psi(x_1, ..., x_n)$ is a quantifier-free $\mathcal{L}_{\mathcal{F}}$-formula, i.e., a Boolean combination of atomic formulas of the form $f(x_1, ..., x_n) \circ 0$, where f is a composition of functions in $\mathcal{F}$ and $\circ \in \{<, \leq, >, \geq, =, \neq\}$.*
- *We write $\mathrm{dom}(\varphi) = I_1 \times \cdots \times I_n$, and require that all the functions occurring in $\psi(\boldsymbol{x})$ are defined everywhere over its closure $\overline{\mathrm{dom}(\varphi)}$.*

We can write a bounded Σ_1-sentence as $\exists^{\boldsymbol{I}} \boldsymbol{x} . \psi(\boldsymbol{x})$ for short.

Lemma 2.1 (Standard Form). *Any bounded Σ_1-sentence φ in $\mathcal{L}_{\mathcal{F}}$ is equivalent over $\mathbb{R}_{\mathcal{F}}$ to a sentence of the following form:*

$$\exists^{I_1} x_1 \cdots \exists^{I_n} x_n \bigwedge_{i=1}^{m} (\bigvee_{j=1}^{k_i} f_{ij}(\boldsymbol{x}) = 0).$$

Proof. Assume that φ is originally $\exists^{\boldsymbol{I}} \boldsymbol{x} \ \bigwedge_{i=1}^{m}(\bigvee_{j=1}^{k_i} g_{ij}(\boldsymbol{x}) \circ 0)$, where $\circ \in \{<, \leq, >, \geq, =, \neq\}$. We apply the following transformations:

1. **(Eliminate $\neq$)** Substitute each atomic formula of the form $g_{ij} \neq 0$ by $g_{ij} < 0 \vee g_{ij} > 0$.

2. **(Eliminate $\leq, <$)** Substitute $g_{ij} \leq 0$ by $-g_{ij} \geq 0$, and $g_{ij} < 0$ by $-g_{ij} > 0$. Now the formula is rewritten to $\exists^{\boldsymbol{I}} \boldsymbol{x}. \bigwedge_{i=1}^{m}(\bigvee_{j=1}^{k_i} g'_{ij}(\boldsymbol{x}) \circ 0)$, where $\circ \in \{>, \geq, =\}$. ($g'_{ij} = -g_{ij}$ if the inequality is reversed; otherwise $g'_{ij} = g_{ij}$.)

3. **(Eliminate $\geq, >$)** Substitute $g'_{ij} \geq 0$ (or $g'_{ij} > 0$) by $g'_{ij} - v_{ij} = 0$, where v_{ij} is a newly introduced variable, and add an innermost bounded existential quantifier $\exists v_{ij} \in I_{v_{ij}}$, where $I_{v_{ij}} = [0, m_{v_{ij}}]$ ($I_v = (0, m_{v_{ij}}]$). Here, $m_{v_{ij}} \in \mathbb{Q}$ is any value greater than the maximum of g'_{ij} over $\overline{\mathrm{dom}(\varphi)}$. Note that such maximum of g'_{ij} always exists over $\overline{\mathrm{dom}(\varphi)}$, since g'_{ij} is continuous on $\overline{\mathrm{dom}(\varphi)}$, which is a compact, and is computable [22].

The formula is now in the form $\exists^{\boldsymbol{I}} \boldsymbol{x} \exists^{\boldsymbol{I_v}} \boldsymbol{v}. \ \bigwedge_{i=1}^{m}(\bigvee_{j=1}^{k_i} f_{ij}(\boldsymbol{x}, \boldsymbol{v}) = 0)$, where $f_{ij} = g'_{ij} - v_{ij}$ if v_{ij} has been introduced in the previous step; otherwise, $f_{ij} = g'_{ij}$. The new formula is in the standard form and equivalent to the original. □

Example 2.1. A standard form of $\exists^{[-1,1]} x \exists^{[-1,1]} y \exists^{[-1,1]} z \ (e^z < x \to y < \sin(x))$ is

$$\exists^{[-1,1]} x \exists^{[-1,1]} y \exists^{[-1,1]} z \exists^{[0,10]} u \exists^{(0,10]} v \ (e^z - x - u = 0) \vee (\sin(x) - y - v = 0).$$

Recall that we allow the interval bounds on variables to be either open or closed. Let $\overline{S}$ and S^o denote the closure and interior of any set S over the reals. Based on our need we can consider the closure or the interior of the domains in a Σ_1-sentence.

Definition 2.6 (Closure and Interior). *Let $\varphi := \exists^{I_1} x_1 \cdots \exists^{I_n} x_n . \psi(\boldsymbol{x})$ be a bounded Σ_1-sentence in $\mathcal{L}_{\mathcal{F}}$, we define the closure and interior of φ as:*

$$\overline{\varphi} := \exists^{\overline{I_1}} x_1 \cdots \exists^{\overline{I_n}} x_n . \psi(\boldsymbol{x}) \qquad \text{(Closure)}$$

$$\varphi^o := \exists^{I_1^o} x_1 \cdots \exists^{I_n^o} x_n . \psi(\boldsymbol{x}) \qquad \text{(Interior)}$$

Proposition 2.2. *For any Σ_1-sentence φ, $\varphi^o \to \varphi$ and $\varphi \to \overline{\varphi}$.*

3 The Bounded δ-SMT Problem

The key for bridging numerical procedures and SMT problems is to introduce syntactic perturbations on Σ_1-sentences in $\mathcal{L}_{\mathcal{F}}$.

Definition 3.1 (δ-Weakening and Perturbations). *Let $\delta \in \mathbb{Q}^+ \cup \{0\}$ be a constant and φ be a Σ_1-sentence in standard form:*

$$\varphi := \exists^{\boldsymbol{I}} \boldsymbol{x}. \bigwedge_{i=1}^{m}(\bigvee_{j=1}^{k_i} f_{ij}(\boldsymbol{x}) = 0).$$

The δ-weakening of φ defined as:

$$\varphi^{\delta} := \exists^{\boldsymbol{I}} \boldsymbol{x}. \bigwedge_{i=1}^{m}(\bigvee_{j=1}^{k} |f_{ij}(\boldsymbol{x})| \leq \delta).$$

Also, a δ-perturbation is a constant vector $\boldsymbol{c} = (c_{11}, ..., c_{mk_m})$, $c_{ij} \in \mathbb{Q}$, satisfying $||\boldsymbol{c}|| \leq \delta$, such that the $\boldsymbol{c}$-perturbed form of φ is given by:

$$\varphi^{\boldsymbol{c}} := \exists^{\boldsymbol{I}} \boldsymbol{x}. \bigwedge_{i=1}^{m} (\bigvee_{j=1}^{k} f_{ij}(\boldsymbol{x}) = c_{ij}).$$

Proposition 3.1. *φ^δ is true iff there exists a δ-perturbation $\boldsymbol{c}$ such that $\varphi^{\boldsymbol{c}}$ is true. In particular, $\boldsymbol{c}$ can be the zero vector, and thus $\varphi \rightarrow \varphi^\delta$.*

We now define the bounded δ-SMT problem. We follow the convention that SMT solvers return sat/unsat, which is equivalent to the corresponding Σ_1-sentence being true/false.

Definition 3.2 (Bounded δ-SMT in $\mathcal{L}_\mathcal{F}$). *Let $\mathcal{F}$ be a finite collection of Type 2 computable functions. Let φ be a bounded Σ_1-sentence in $\mathcal{L}_\mathcal{F}$ in standard form. The* bounded δ-SMT problem *asks for one of the following two decisions on φ:*

- unsat : *φ is false.*
- δ-sat : *φ^δ is true.*

When the two cases overlap, either decision can be returned.

Our main theoretical claim is that the bounded δ-SMT problem is decidable for $\delta \in \mathbb{Q}^+$. This is essentially a special case of our more general results for arbitrarily-quantified $\mathcal{L}_\mathcal{F}$-sentences [15]. However, different from [15], here we defined the standard forms of SMT problems to contain only equalities in the matrix, on which the original proof does not work directly. Also, in [15] we relied on results from computable analysis that are not needed here. We now give a direct proof for the decidability of δ-SMT and analyze its complexity.

Theorem 3.1 (Decidability). *Let $\mathcal{F}$ be a finite collection of Type 2 computable functions and $\delta \in \mathbb{Q}^+$. The bounded δ-SMT problem in $\mathcal{L}_\mathcal{F}$ is decidable.*

Proof. We describe a decision procedure which, given any bounded Σ_1-sentence φ in $\mathcal{L}_\mathcal{F}$ and $\delta \in \mathbb{Q}^+$, decides whether φ is false or φ^δ is true. Assume that φ is in the form of Definition 3.1.

First, we need a uniform bound on all the variables so that a modulus of continuity for each function can be computed. Suppose each x_i is bounded by I_i, whose closure is $\overline{I_i} = [l_i, u_i]$. We write

$$\overline{\varphi} := \exists^{[0,1]} x_1 \cdots \exists^{[0,1]} x_n \bigwedge_{i=1}^{m} (\bigvee_{j=1}^{k_i} f_{ij}(l_1 + (u_1 - l_1)x_1, ..., l_n + (u_n - l_n)x_n) = 0).$$

From now on, $g_{ij} = f_{ij}(l_1 + (u_1 - l_1)x_1, ..., l_n + (u_n - l_n)x_n)$. After the transformation, we have $\mathrm{dom}(\overline{\varphi}) = [0,1] \times \cdots \times [0,1]$, on which each g_{ij} is computable (it is a composition of the finitely many computable functions in $\mathcal{F}$) and has a computable modulus of continuity $m_{g_{ij}}$. We write $\psi(\boldsymbol{x})$ to denote the matrix of φ after the transformation.

Choose $r \in \mathbb{N}$ such that $2^{-r} < \delta/4$. Then for each g_{ij}, we use $m_{g_{ij}}$ to obtain $e_{ij} = m_{g_{ij}}(r)$. Choose $e \in \mathbb{N}$ such that

$$e \geq \max(e_{11}, ..., e_{mk_m}) \quad (1)$$

and write $\varepsilon = 2^{-e}$. We then have

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \overline{\mathrm{dom}(\varphi)}\ (||\boldsymbol{x} - \boldsymbol{y}|| < \varepsilon \to |g_{ij}(\boldsymbol{x}) - g_{ij}(\boldsymbol{y})| < \delta/4). \tag{2}$$

We now consider a finite ε-net of $\overline{\mathrm{dom}(\varphi)}$, i.e., a finite $S_\varepsilon \subseteq \overline{\mathrm{dom}(\varphi)}$, satisfying

$$\forall \boldsymbol{x} \in \overline{\mathrm{dom}(\varphi)}\ \exists \boldsymbol{a} \in S_\varepsilon\ ||\boldsymbol{x} - \boldsymbol{a}|| < \varepsilon. \tag{3}$$

In fact, S_ε can be explicitly defined as

$$S_\varepsilon = \{(a_1, ..., a_n) : a_i = k \cdot \varepsilon,\ \text{where } k \in \mathbb{N}, 0 \leq k \leq 2^e\}. \tag{4}$$

Next, we evaluate the matrix $\psi(\boldsymbol{x})$ on each point in S_ε, as follows. Let $\boldsymbol{a} \in S_\varepsilon$ be arbitrary. For each g_{ij} in ψ, we compute $g_{ij}(\boldsymbol{a})$ up to an error bound of $\delta/8$, and write the result of the evaluation as $\overline{g_{ij}(\boldsymbol{a})}^{\delta/8}$. Then $|g_{ij}(\boldsymbol{a}) - \overline{g_{ij}(\boldsymbol{a})}^{\delta/8}| < \delta/8$. Note $\overline{g_{ij}(\boldsymbol{a})}^{\delta/8}$ is a rational number. We then define

$$\widehat{\psi}(\boldsymbol{x}) := \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} |\overline{g_{ij}(\boldsymbol{x})}^{\delta/8}| < \delta/2. \tag{5}$$

Then for each $\boldsymbol{a}$, evaluating $\widehat{\psi}(\boldsymbol{a})$ only involves comparison of rational numbers and Boolean evaluation, and $\widehat{\psi}(\boldsymbol{a})$ is either true or false. Now, by collecting the value of $\widehat{\psi}$ on every point in S_ε, we have the following two cases.

- Case 1: For some $\boldsymbol{a} \in S_\varepsilon$, $\widehat{\psi}(\boldsymbol{a})$ is true. We show that φ^δ is true. Note that

$$\widehat{\psi}(\boldsymbol{a}) \Rightarrow \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} |\overline{g_{ij}(\boldsymbol{a})}^{\delta/8}| < \delta/2 \Rightarrow \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} |g_{ij}(\boldsymbol{a})| < \delta \cdot 5/8.$$

We need to be careful about $\boldsymbol{a}$, since it is an element in $\overline{\mathrm{dom}(\varphi)}$, not $\mathrm{dom}(\varphi)$. If $\boldsymbol{a} \in \mathrm{dom}(\varphi)$, then φ^δ is true, witnessed by $\boldsymbol{a}$. Otherwise, $\boldsymbol{a} \in \partial(\mathrm{dom}(\varphi))$. Then by continuity of g_{ij}, there exists $\boldsymbol{a}' \in \mathrm{dom}(\varphi)$ such that $\bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} |g_{ij}(\boldsymbol{a}')| < \delta$. (Just let a small enough ball around $\boldsymbol{a}$ intersect $\mathrm{dom}(\varphi)$ at $\boldsymbol{a}'$.) That means φ^δ is also true in this case, witnessed by $\boldsymbol{a}'$.

- Case 2: For every $\boldsymbol{a} \in S_\varepsilon$, $\widehat{\psi}(\boldsymbol{a})$ is false. We show that φ is false. Note that

$$\neg\widehat{\psi}(\boldsymbol{a}) \Rightarrow \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k_i} |\overline{g_{ij}(\boldsymbol{a})}^{\delta/8}| \geq \delta/2 \Rightarrow \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k_i} |g_{ij}(\boldsymbol{a})| \geq \delta \cdot 3/8.$$

Now recall condition (2) and (3). For an arbitrary $\boldsymbol{x} \in \mathrm{dom}(\varphi)$, there exists $\boldsymbol{a} \in S_\varepsilon$ such that $|g_{ij}(\boldsymbol{x}) - g_{ij}(\boldsymbol{a})| < \delta/4$ for every g_{ij}. Consequently, we have $|g_{ij}(\boldsymbol{x})| \geq \delta \cdot 3/8 - \delta/4 = \delta/8$. Thus, $\forall \boldsymbol{x} \in \mathrm{dom}(\varphi), \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k_i} |g_{ij}(\boldsymbol{x})| > 0$. This means $\neg\varphi$ is true, and φ is false.

In all, the procedure that decides either that φ^δ is true, or that φ is false. □

We now analyze the complexity of the δ-SMT problem. The decision procedure given above essentially evaluates the formula on each sample point. Thus, given an oracle for evaluating the functions, we can construct a nondeterministic Turing machine that randomly picks the sample points and decides the formula.

Most of the functions we are interested in (exp, sin, ODEs) are in Type 2 complexity class P or PSPACE. To prove interesting complexity results, a technical restriction is that we need to bound the number of function compositions in a formula, because otherwise evaluating nested polynomial-time functions can be exponential in the number of nesting. Formally we define:

Definition 3.3 (Uniformly Bounded Σ_1-class). *Let $\mathcal{F}$ be a finite set of Type 2 computable functions, and S a class of bounded Σ_1-sentences in $\mathcal{L}_{\mathcal{F}}$. Let $l, u \in \mathbb{Q}$ satisfy $l \leq u$. We say S is uniformly $(l, u, \mathcal{F})$-bounded, if $\forall \varphi \in S$ of the form $\exists^{I_1} x_1 \cdots \exists^{I_n} x_n \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} f_{ij}(\boldsymbol{x}) = 0$,*

- *$\forall 1 \leq i \leq n$, $I_i \subseteq [l, u]$.*
- *Each $f_{ij}(\boldsymbol{x})$ is contained in $\mathcal{F}$.*

Proposition 3.2 ([22]). *Let C be a Type 2 complexity class contained in PSPACE. Then given any compact domain D, a C-computable function has a uniform modulus of continuity over D given by a polynomial function.*

We are now ready to prove the main complexity claim.

Theorem 3.2 (Complexity). *Let $\mathcal{F}$ be a finite set of functions in Type 2 complexity class C, $\mathsf{P} \subseteq \mathsf{C} \subseteq \mathsf{PSPACE}$. The δ-SMT problem for uniformly bounded Σ_1-classes in $\mathcal{L}_{\mathcal{F}}$ is in NP^{C}.*

Proof. We describe a nondeterministic Turing machine with a function oracle of complexity C, that can decide in polynomial-time the δ-SMT problem for a uniformly bounded class.

The function oracle θ we use behaves as follows. Given strings s, t, and d on the query tape, $\theta(s, t, d)$ looks up the function $f_s \in \mathcal{F}$ encoded by s and returns the value of $f_s(\boldsymbol{x}_t)$ up to an error bound of 2^{-d}, where $\boldsymbol{x}_t$ is a rational vector encoded by t taken as the argument of f_s. Since all the functions in $\mathcal{F}$ are in complexity class C, $\theta(s, t, d)$ is a C-oracle.

For any symbol s, we write $len(s)$ to denote its bit-length. For an integer i, we know $len(i) = O(\log(i))$. For a rational number d, which is the ratio of coprime integers p and q, $len(d) = O(len(p) + len(q)) = O(\log(pq))$. For a function f, $len(f)$ is the length of its name. We write $O(\mathsf{poly}(n))$ to denote $\bigcup_k O(n^k)$.

Let φ be the input formula as in Definition 3.1, where each $f_{ij} \in \mathcal{F}$. Suppose φ is in a uniformly $(l, u, \mathcal{F})$-bounded class.

First, we observe that e, defined in (1), can be obtained in time $O(\mathsf{poly}(len(\varphi) + len(\delta)))$, and $e = O(\mathsf{poly}(len(\varphi) + len(\delta)))$ (thus $len(e) = O(len(\varphi) + len(\delta)))$. This can be seen as follows. First, $2^{-r} < \delta$, we know $r = O(\log(\delta)) = O(len(\delta))$. Then for each f_{ij}, we use its uniform modulus of continuity over $[l, u]$, given by a polynomial $m_{f_{ij}}$ (Proposition 3.2), and obtain $e^f_{ij} = m_{f_{ij}}(r)$, in time $O(\mathsf{poly}(len(r)))$ and $e^f_{ij} = O(\mathsf{poly}(r))$. Then we compute e_{ij} for the function g_{ij} by scaling e^f_{ij}, using $e_{ij} = \lceil -\log(2^{-e^f_{ij}} / \max_{1 \leq i \leq n}\{u_i - l_i\}) \rceil$. Thus $e_{ij} = O(e^f_{ij} + \log(\max_i(u_i - l_i))) = O(\mathsf{poly}(len(\delta) + len(\varphi)))$. Finally, let e be the biggest e_{ij}. It is then clear that $e = O(\mathsf{poly}(len(\varphi) + len(\delta)))$, obtainable in polynomial time.

Next, our procedure evaluates the matrix of the formula on each point $\boldsymbol{a} \in S_\varepsilon$. Note from (4) that S_ε is of size exponential in e. Here we exploit the nondeterminism of the machine by randomly picking $0 \leq k \leq 2^e$ on each dimension. Note that since $\log(k) \leq e$, we have $len(k) = O(e) = O(poly(len(\varphi) + len(\delta)))$. Let $\boldsymbol{a} = (a_1, ..., a_n)$ be the randomly picked point in S_ε. Following the above estimate of $len(k)$ and $len(\varepsilon) = O(\log(2^{-e})) = O(e)$, we have $len(\boldsymbol{a}) = O(\mathsf{poly}(len(\varphi) + len(\delta)))$.

Now we evaluate $\widehat{\varphi}(\boldsymbol{a})$. With access to the C-oracle specified above, this can be done in polynomial-time, as follows. For each $g_{ij}(\boldsymbol{a})$, we query the oracle with $\theta(f_{ij}, \boldsymbol{a}_{lu}, \delta/8)$, where $\boldsymbol{a}_{lu}$ is $\boldsymbol{a}$ scaled by $[l_i, u_i]$ on each dimension. This query uses $O(\mathsf{poly}(len(\varphi) + len(\delta)))$-space on the query tape. The oracle then return the value of $\overline{f_{ij}(\boldsymbol{a}_{lu})}^{\delta/8} = \overline{g_{ij}(\boldsymbol{a})}^{\delta/8}$, and since $\mathsf{C} \subseteq \mathsf{PSPACE}$,

$len(\overline{g_{ij}(\boldsymbol{a})}^{\delta/8})$ is polynomial in the input. Next we evaluate each atom by comparing these values obtained from the oracle with $\delta/2$. This uses time $O(\mathsf{poly}(len(\varphi) + len(\delta)))$. Finally, if $\widehat{\psi}(\boldsymbol{x})$ is true, we return δ-sat. Thus the problem is decided in nondeterministic polynomial-time using access to the C-oracle. We can conclude that the δ-SMT problem for a uniformly bounded class is in NP^{C}. □

Remark 3.1. The restriction of a uniformly bounded class of formulas is a technical one. For a class of formulas of interest, we can always choose a rich enough $\mathcal{F}$ that contains the compositions we need, and a loose enough uniform bound on the variables.

We can now obtain a precise characterization of the complexity for δ-SMT problems in signatures of interest. Recall that most common functions, such as polynomials, exp, sin, are all P-computable and Lipschitz-continuous ODEs are PSPACE-complete.

Corollary 3.1. *Let $\mathcal{F}$ be a finite set of P-time computable real functions, such as $\{+, \times, \exp, \sin\}$. The uniformly-bounded δ-SMT problem for $\mathcal{L}_{\mathcal{F}}$ is NP-complete.*

Proof. Since the functions in $\mathcal{F}$ are P-time computable, the δ-SMT problem is in $\mathsf{NP}^{\mathsf{P}} = \mathsf{NP}$. We only need to encode Boolean satisfiability for hardness. We need to be careful that no negations can be used. For any propositional formula $\phi(p_1, ..., p_n)$, substitute p_i by $x_i < 0$ and $\neg p_i$ by $x_i > 1$, and add $(x_i = 0 \vee x_i = 1)$ as a clause to the formula. Add the quantifiers $\exists^{[-1,2]} x_i$ for each x_i. Then for any $\delta < 0.5$, ϕ is satisfiable iff the translation is δ-true, and unsatisfiable iff the translation is false. Note that the cases do not overlap. □

Corollary 3.2. *Let $\mathcal{F}$ be a finite set of Lipschitz-continuous ODEs over compact domains. Then the uniformly-bounded δ-SMT problem in $\mathcal{L}_{\mathcal{F}}$ is in PSPACE, and there exists $\mathcal{L}_{\mathcal{F}}$ such that it is PSPACE-complete.*

Proof. We have $\mathsf{NP}^{\mathsf{PSPACE}} = \mathsf{PSPACE}$. Since some ODEs are PSPACE-complete to solve [21], there exists $\mathcal{L}_{\mathcal{F}}$ for which δ-SMT problem is PSPACE-complete. □

4 δ-Completeness of the DPLL⟨ICP⟩ Framework

We now give a formal analysis of the integration of ICP and DPLL(T) for solving bounded δ-SMT. Our goal is to establish sufficient and necessary conditions under which such an integration is δ-complete.

4.1 Interval Constraint Propagation

The method of Interval Constraint Propagation (ICP) [3] finds solutions of real constraints using a "branch-and-prune" method, combining interval arithmetic and constraint propagation. The idea 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, 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 and we give formal definitions as follows.

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 $\mathbb{IF} = \{[a, b] \subseteq \mathbb{R} : a, b \in \mathbb{F}, a \leq b\}$ denote the set of closed real intervals with floating-point endpoints, and $\mathbb{BF} = \bigcup_{n=1}^{\infty} \mathbb{IF}^n$ the set of* boxes *with these intervals. Let $S \subseteq \mathbb{R}$ be any set of real numbers, the hull of S is written as $\mathrm{Hull}(S) = \bigcap\{I \in \mathbb{IF} : S \subseteq I\}$.*

For $I = [a, b] \in \mathbb{IF}$, we write $|I| = |b - a|$ to denote its size.

Definition 4.2 (Interval Extension (cf. [3])). *Let $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ be a real function. An interval extension operator $\sharp(\cdot)$ maps f to a function $\sharp f :\subseteq \mathbb{BF} \to \mathbb{IF}$, such that $\forall B \in \mathbb{BF} \cap \mathrm{dom}(\sharp f), \{f(\boldsymbol{x}) : \boldsymbol{x} \in B\} \subseteq \sharp f(B)$.*

Example 4.1. The natural extension of $f = 2 \cdot (x + y) \cdot z$ is given by $\sharp f = [2, 2] \cdot (I_x + I_y) \cdot I_z$, where the interval operations are defined as $[a_1, b_1] + [a_2, b_2] = [a_1 + a_2, b_1 + b_2]$ and $[a_1, b_1] \cdot [a_2, b_2] = [\min(a_1a_2, a_1b_2, b_1a_2, b_1b_2), \max(a_1a_2, a_1b_2, b_1a_2, b_1b_2)]$.

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

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

In 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 assumed that there exists some constant $0 < c < 1$ such that $c \cdot |I_i| \leq |I_i'|$ and $c \cdot |I_i| \leq |I_i''|$ for all i.

The key component of the algorithm is the Prune(B, f) operation. A simple example of a pruning operation is as follows.

Example 4.2. Consider $x - y^2 = 0$ with initial intervals $x \in [1, 2]$ and $y \in [0, 4]$. Let $\sharp f(I_x, I_y) = I_x - I_y^2$ be the natural interval extension of the left hand side. Since we know $0 \not\in \sharp f([1, 2], [2, 4])$, we can contract the interval on y from $[0, 4]$ to $[0, 2]$ in one pruning step.

In principle, any operation that contracts the intervals on variables can be seen as pruning. But to ensure correctness, we need formal requirements on the pruning operator in ICP$_\varepsilon$. Basically, the interval extensions of the functions should converge to the true values of the functions, and the pruning operations are "well-defined," as specified below.

Notation 4.1 *For any $f : \mathbb{R}^n \to \mathbb{R}$, we write $Z_f = \{\boldsymbol{a} \in \mathbb{R}^n : f(\boldsymbol{a}) = 0\}$.*

Definition 4.3 (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 $B \in \mathbb{R}^n$, $|\sharp f(B)| \leq \max(c||B||, \delta)$.*

Definition 4.4 (Well-defined Pruning Operators). *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* $\text{Prune}_\sharp :\subseteq \mathbb{BF}\times\mathcal{F}\to\mathbb{BF}$*, such that for any $f\in\mathcal{F}$, $B, B'\in\mathbb{BF}$,*

- *(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)* $B\cap Z_f\subseteq \text{Prune}_\sharp(B,f)$*;*

When $\sharp$ is clear, we simply write Prune. It specifies the following requirements. (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 [3]). We use $\text{Prune}(B, f_1, ..., f_m)$ to denote the iterative application of $\text{Prune}(\cdot, f_i)$ on B for all $1\le i\le m$, until a fixed-point is reached. (Line 4-6 in Algorithm 1.)

Proposition 4.1. *For all i, $Prune(B, f_1, ..., f_m)\subseteq Prune(B, f_i)$.*

It is clear from the description of Algorithm 1 that the following properties hold.

Lemma 4.1. *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 $\forall \boldsymbol{a}\in B_0$, there exists $B\subseteq B_0$ such that $\boldsymbol{a}\in B$ and $\text{Prune}(B, f_1, ..., f_m)=\emptyset$.*

Now we prove the main theorem.

Theorem 4.2 (δ-Completeness of ICP$_\varepsilon$). *Let $\delta\in\mathbb{Q}^+$ be arbitrary. We can find an $\varepsilon\in\mathbb{Q}^+$ such that the* ICP$_\varepsilon$ *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$_\varepsilon$ is well-defined.*

Proof. We consider an arbitrary bounded existential $\mathcal{L}_\mathcal{F}$-sentence containing only conjunctions, written as $\varphi : \exists^{\boldsymbol{I}}\boldsymbol{x}. \bigwedge_{i=1}^m f_i(\boldsymbol{x}) = 0$. Let $B_0 = \boldsymbol{I}$ be the initial bounding box.

Since all the functions in φ are computable over B_0, each f_i has a uniform modulus of continuity over B_0, which we write as m_{f_i}. Choose any $k\in\mathbb{N}$ such that $2^{-k}<\delta$. Then for any $\varepsilon_i < m_{f_i}(k)$, we have

$$\forall \boldsymbol{x},\boldsymbol{y}\in B_0, ||\boldsymbol{x}-\boldsymbol{y}|| < \varepsilon_i \to |f_i(\boldsymbol{x}) - f_i(\boldsymbol{y})| < \delta. \tag{6}$$

We now fix ε to be any positive rational number smaller than $\min(\varepsilon_1, ..., \varepsilon_m)$.

By the previous lemma, we know ICP$_\varepsilon$ terminates and returns either `sat` or `unsat`. We now prove the two directions of the biconditional.

$\Leftarrow$: Suppose the pruning operator in ICP$_\varepsilon$ is well-defined.

Suppose ICP$_\varepsilon$ returns "δ-sat", then by Lemma 4.1, there exist $B, B'\subseteq B_0$ such that $B = \text{Prune}(B', f_1, ..., f_m)$ and $||B'|| < \varepsilon$. Then by the (W2), we know that $0\in\sharp f_i(B_n)$ for every f_i. Now, by the definition of ε, we know from (6) that for every i, $\forall\boldsymbol{a}\in B, |f_i(\boldsymbol{a}) - 0| < \delta$. Namely, any $\boldsymbol{a}\in B$ is a witness for $\varphi^\delta : \exists^{\boldsymbol{I}}\boldsymbol{x}\ |f(\boldsymbol{x})| < \delta$. Thus the δ-weakening of φ is true.

Suppose ICP$_\varepsilon$ returns "unsat". Suppose φ is in fact satisfiable. Then there is a point $\boldsymbol{a}\in B_0$ such that $\psi(\boldsymbol{a})$ is true. However, following Lemma 4.1, $\boldsymbol{a}\in B$ for some $B\subseteq B_0$ and $\text{Prune}(B_0, f_1, ..., f_m) = \emptyset$. However, this contradicts condition (W3) of the pruning operator.

$\Rightarrow$: We only need to show that without any one of the three conditions in Definition 4.4, we can define a pruning operator that fails δ-completeness.

Without (W1), we define a pruning operator that always outputs intervals bigger than ε (such as the initial intervals). Then the procedure never terminates. Note that the other two conditions are trivially satisfied in this case (for any f and B_0 satisfying $0 \in \sharp f(B_0)$). Without (W2), consider the function $f(x) = x^2 + 1$ with $x \in [-1, 1]$. We can define a pruning operator such that $\text{Prune}([-1, 1], f) = [1, 1]$. This operator satisfies the other two conditions. However, the returned result $[1, 1]$ fails δ-completeness for any δ smaller than 2, since $f(1) = 2$. Without (W3), we simply prune any set to $\emptyset$ and always return `unsat`. This violates δ-completeness, which requires that if `unsat` is returned the formula must be indeed unsatisfiable. The other two conditions are also satisfied in this case. □

In practice, pruning operators are defined based on *consistency conditions* from constraint propagation techniques. Many pruning operators are used in practice [3]. Following Theorem 4.2, we only need to prove their well-definedness to ensure δ-completeness. For instance:

Definition 4.5 (Box-consistent Pruning [19]). *We say $\pi_B : \mathbb{BF} \times \mathcal{F} \to \mathbb{BF}$ is box-consistent, if for all $f \in \mathcal{F}$ and $B = I_1 \times \cdots \times I_n \subseteq \text{dom}(f)$, the i-th interval of $\pi_B(B, f)$ is $I_i \cap \text{Hull}\big(\{a_i \in \mathbb{R} : 0 \in \sharp f(I_1, ..., \text{Hull}(\{a_i\}), ..., I_n\}\big)$.*

Proposition 4.2. *The Box-consistent Pruning operator is well-defined.*

4.2 Handling ODEs

In this section we expand our language to consider solutions of the initial value problems (IVP) of Lipschitz-continuous ODEs. Let $t_0, T \in \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$ be a Lipschitz-continuous function, i.e., for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n$, $|g(\boldsymbol{x}_1) - g(\boldsymbol{x}_2)| \leq c||\boldsymbol{x}_1 - \boldsymbol{x}_2||$ for some constant c. Let $t_0, T \in \mathbb{R}$ satisfy $t_0 \leq T$ and $\boldsymbol{y}_0 \in \mathbb{R}^n$. An IVP problem is given by

$$\frac{d\boldsymbol{y}}{dt} = g(\boldsymbol{y}(t)) \text{ and } \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \text{ where } t \in [t_0, T].$$

where $\boldsymbol{y} : [t_0, T] \to \mathbb{R}^n$ is called the *solution* of the IVP. Consider $\boldsymbol{y}(t)$ as $(y_1(t), ..., y_n(t))$, then each component $y_i : [t, T] \to \mathbb{R}$ is a Type 2 computable function, and can appear in some signature $\mathcal{F}$. In fact, we can also regard $\boldsymbol{y}_0$ as an argument of y_i and write $y_i(t_0, \boldsymbol{y}_0)$. This does not change computability properties of y_i, since following the Picard-Lindelöf representation $\boldsymbol{y}(t) = \int_{t_0}^{t} g(\boldsymbol{y}(s))ds + \boldsymbol{y}_0$, $y_i(t)$ is only linearly dependent on $\boldsymbol{y}_0$.

In practice, with an ICP framework, we can exploit interval solvers for IVP problems [26], for pruning intervals on variables that appear in constraints involving ODEs. This direction has received much recent attention [12,11,17,20].

Consider the IVP problem defined above, with $\boldsymbol{y}_0$ contained in a box $B_{t_0} \subseteq \mathbb{R}^n$. Let $t_0 \leq t_1 \leq ... \leq t_m = T$ be a set of points in $[t_0, T]$. An interval-based ODE solver returns a set of boxes $B_{t_1}, ..., B_{t_m}$ such that

$$\forall i \in \{1, ..., m\}, \ [y(t_i; B_{t_0})] = \{\boldsymbol{y}(t) : t_0 \leq t \leq t_i, \boldsymbol{y}_0 \in B_{\boldsymbol{y}_0}\} \subseteq B_{t_i}.$$

Now let $y_i : [t_0, T] \times B_0 \to \mathbb{R}$ be the i-th component of the solution $\boldsymbol{y}$ of an IVP problem. Then interval-based ODE solvers compute interval extensions of y_i. Thus, pruning operators that respect the interval extension computed by interval ODE solvers can be defined. It can be concluded from Theorem 4.2 that ICP_ε is δ-complete for equalities involving ODEs, as long as the pruning operator is well-defined. A simplest strategy is just to prune out any set of points outside the interval extension:

Proposition 4.3 (Simple ODE-Pruning). *Let $y_i(t, \boldsymbol{y}_0)$ be the i-th component function of an IVP problem. Suppose $\sharp y_i$ is computed by an interval ODE solver. Then the pruning operator $\mathrm{Prune}(I, y_i) = I \cap \sharp y_i(I_t, B_{\boldsymbol{y}_0})$ is well-defined.*

4.3 DPLL⟨ICP⟩

Now consider the integration of ICP into the framework of DPLL(T), so that the full δ-SMT problem can be solved. Given a formula φ, a DPLL⟨ICP⟩ solver uses SAT solvers to enumerate solutions to the Boolean abstraction φ^B of the formula, and uses ICP_ε to decide the satisfiability of conjunctions of atomic formulas. DPLL⟨ICP⟩ returns sat when ICP_ε returns sat to some conjunction of theory atoms witnessing the satisfiability of φ^B, and returns unsat when ICP_ε returns unsat on all the solutions to φ^B. Thus, it follows naturally that using a δ-complete theory solver ICP_ε, DPLL⟨ICP⟩ is also δ-complete.

Corollary 4.1 (δ-Completeness of DPLL⟨ICP⟩). *Let $\mathcal{F}$ be a set of real functions. Then the pruning operators in ICP_ε are well-defined for $\mathcal{F}$, if and only if, DPLL⟨ICP⟩ using ICP_ε is δ-complete for bounded Σ_1-sentences in $\mathcal{L}_\mathcal{F}$.*

Proof. Let φ be a bounded SMT problem $\exists^{\boldsymbol{I}} \boldsymbol{x} \bigwedge_i \bigvee_j f_{ij}(\boldsymbol{x}) = 0$. Its Boolean abstraction φ^B is given by $\bigwedge_i \bigvee_j p_{ij}$, where p_{ij} is the propositional abstraction of $f_{ij}(\boldsymbol{x}) = 0$.

Choose ε to satisfy that $\forall \boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{I} |f_{ij}(\boldsymbol{x}) - f_{ij}(\boldsymbol{y})| < \delta$ for all f_{ij} that appear in the φ.

Now, in the DPLL(T) framework, the SAT solver returns an assignment to p_{ij} such that φ^B evaluates to true, then ICP_ε is used for checking the satisfiability of the corresponding conjunction of theory atoms. It is important to note that φ^B does not contain negations.

Suppose the pruning operator in ICP_ε is well-defined. Then ICP_ε is δ-complete. Now, suppose DPLL⟨ICP⟩ returns sat. Then φ^B is true witnessed by a set $\{p_1, ..., p_m\}$ assigned to true, which in turn corresponds to a set $\{f_1(\boldsymbol{x}) = 0, ..., f_m(\boldsymbol{x}) = 0\}$ of the theory atoms. By δ-completeness of ICP_ε, we know that φ^δ is true. On the other hand, suppose φ is decided as unsat. Then either there is no assignment such that φ^B is true, or for each satisfying assigment to φ^B, ICP_ε decides that the corresponding set of theory atoms is not satisfiable. By δ-completeness of ICP_ε, the unsat answers are always correct. In all, DPLL⟨ICP⟩ is also δ-complete.

Suppose the pruning operator in ICP_ε is not well-defined, then DPLL⟨ICP⟩ is simply not δ-complete for conjunctions of theory atoms, and thus not δ-complete for bounded SMT in $\mathcal{L}_\mathcal{F}$. □

5 Applications

δ-Complete solvers return answers that allow one-sided, δ-bounded errors. The framework allows us to easily understand the implications of such errors in practical problems. Indeed, δ-complete solvers can be *directly* used in the following correctness-critical problems.

Bounded Model Checking and Invariant Validation. Let $S = \langle X, \mathsf{Init}, \mathsf{Trans}\rangle$ be a transition system over X, which can by continuous or hybrid. Then given a subset $U \subseteq X$, the bounded model checking problem asks whether

$$\varphi_n := \exists \boldsymbol{x}_0, ..., \boldsymbol{x}_n(\boldsymbol{x}_0 \wedge \bigwedge_{i=0}^{n-1} \mathsf{Trans}(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) \wedge \boldsymbol{x}_n \in U)$$

is true. Here U denotes the "unsafe" values of the system, and we say S is safe up to n if φ_n is false. Thus, using a δ-complete solver for φ_n, we can determine the following: If φ_n is unsat, then S is indeed safe up to n; on the other hand, if φ_n is δ-sat, then either the system is unsafe, or it would be unsafe under a δ-perturbation, and a counterexample is provided by the certificate for δ-sat. This δ can be set by the user based on the intended tolerance of errors of the system. Thus, a δ-complete solver can be directly used.

For invariant validation, a proposed invariant Inv can prove safety if the sentence

$$\varphi := \forall \boldsymbol{x}, \boldsymbol{x}'((\mathsf{Init}(\boldsymbol{x}) \to \mathsf{Inv}(\boldsymbol{x})) \wedge (\mathsf{Inv}(\boldsymbol{x}) \wedge \mathsf{Trans}(\boldsymbol{x}, \boldsymbol{x}') \to \mathsf{Inv}(\boldsymbol{x}')) \wedge \mathsf{Inv}(\boldsymbol{x}) \to \neg(U(\boldsymbol{x})))$$

is true. We then use a δ-complete solver on $\neg\varphi$, which is existential. When unsat is returned, Inv is indeed an inductive invariant proving safety. When δ-sat is returned, either Inv is not an inductive invariant, or under a small numerical perturbation, Inv would violate the inductive conditions.

Theorem Proving. For theorem proving, one-sided errors are not directly useful since no robustness problem is involved. We can still approach a statement φ by making δ-decisions on $\neg\varphi$, and refine δ when needed. Starting from any δ, whenever unsat is returned, φ is proved; when δ-sat, we can try a smaller δ. This reflects the common practice in proving these statements.

6 Conclusion

We introduced the notion of "δ-complete decision procedures" for solving SMT problems over real numbers. Our aim is to provide a general framework for solving a wide range of nonlinear functions including transcendental functions and solutions of Lipschitz-continuous ODEs. δ-Completeness serves as a replacement of the conventional completeness requirement on exact solvers, which is impossible to satisfy in this domain. We proved the existence of δ-complete decision procedures for bounded SMT over reals with Type 2 computable functions and showed the complexity of the problem. We use δ-completeness as the standard correctness requirement on numerically-driven decision procedures, and formally analyzed the solving framework DPLL$\langle$ICP$\rangle$. We proved sufficient and necessary conditions for its δ-completeness. We believe our results serve as a foundation for the development of scalable numerically-driven decision procedures and their application in formal verification and theorem proving.

Acknowledgement

We are grateful for many important suggestions from Lenore Blum, Martin Fränzle, and the anonymous reviewers.

References

1. B. Akbarpour and L. C. Paulson. Metitarski: An automatic theorem prover for real-valued special functions. *J. Autom. Reasoning*, 44(3):175–205, 2010.
2. J. Avigad and H. Friedman. Combining decision procedures for the reals. *Logical Methods in Computer Science*, 2(4), 2006.
3. 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.
4. 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.
5. V. Brattka, P. Hertling, and K. Weihrauch. A tutorial on computable analysis. In S. B. Cooper, B. Löwe, and A. Sorbi, editors, *New Computational Paradigms*, pages 425–491. Springer New York, 2008.
6. C. W. Brown and J. H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *ISSAC-2007*.
7. E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
8. E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 2001.
9. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, pages 134–183, 1975.
10. B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV-2006*.
11. A. Eggers, M. Fränzle, and C. Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In *ATVA*, pages 171–185, 2008.
12. 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.
13. 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.
14. 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.
15. S. Gao, J. Avigad, and E. Clarke. δ-Decidability over the reals. `http://www.cs.cmu.edu/∼sicung`.
16. 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.
17. 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.
18. T. C. Hales. Introduction to the flyspeck project. In T. Coquand, H. Lombardi, and M.-F. Roy, editors, *Mathematics, Algorithms, Proofs*, volume 05021 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.
19. P. V. Hentenryck, D. McAllester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, 1997.
20. 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.
21. A. Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. In *IEEE Conference on Computational Complexity*, pages 149–160. IEEE Computer Society, 2009.
22. K.-I. Ko. *Complexity Theory of Real Functions*. BirkHauser, 1991.
23. K.-I. Ko. On the computational complexity of integral equations. *Ann. Pure Appl. Logic*, 58(3):201–228, 1992.
24. D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. Springer, 2008.
25. C. Muñoz and A. Narkawicz. Formalization of a representation of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 2012. Accepted for publication.
26. N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.
27. 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.
28. A. Platzer and E. M. Clarke. The image computation problem in hybrid systems model checking. In *HSCC*, pages 473–486, 2007.
29. S. Ratschan. Quantified constraints under perturbation. *J. Symb. Comput.*, 33(4):493–505, 2002.
30. K. Weihrauch. *Computable Analysis: An Introduction*. 2000.