Delta-Decision Procedures for Exists-Forall Problems over the Reals

Soonho Kong[1], Armando Solar-Lezama[2], and Sicun Gao[3]

[1] Toyota Research Institute
soonho.kong@tri.global
[2] Massachusetts Institute of Technology, USA
asolar@csail.mit.edu
[3] University of California, San Diego, USA
sicung@ucsd.edu

Abstract. We propose δ-complete decision procedures for solving satisfiability of nonlinear SMT problems over real numbers that contain universal quantification and a wide range of nonlinear functions. The methods combine interval constraint propagation, counterexample-guided synthesis, and numerical optimization. In particular, we show how to handle the interleaving of numerical and symbolic computation to ensure delta-completeness in quantified reasoning. We demonstrate that the proposed algorithms can handle various challenging global optimization and control synthesis problems that are beyond the reach of existing solvers.

1 Introduction

Much progress has been made in the framework of delta-decision procedures for solving nonlinear Satisfiability Modulo Theories (SMT) problems over real numbers [1,2]. Delta-decision procedures allow one-sided bounded numerical errors, which is a practically useful relaxation that significantly reduces the computational complexity of the problems. With such relaxation, SMT problems with hundreds of variables and highly nonlinear constraints (such as differential equations) have been solved in practical applications [3]. Existing work in this direction has focused on satisfiability of quantifier-free SMT problems. Going one level up, SMT problems with both free and universally quantified variables, which correspond to $\exists\forall$-formulas over the reals, are much more expressive. For instance, such formulas can encode the search for robust control laws in highly nonlinear dynamical systems, a central problem in robotics. Non-convex, multi-objective, and disjunctive optimization problems can all be encoded as $\exists\forall$-formulas, through the natural definition of "finding some x such that for all other x', x is better than x' with respect to certain constraints." Many other examples from various areas are listed in [4].

Counterexample-Guided Inductive Synthesis (CEGIS) [5] is a framework for program synthesis that can be applied to solve generic exists-forall problems. The idea is to break the process of solving $\exists\forall$-formulas into a loop between *synthesis* and *verification*. The synthesis procedure finds solutions to the existentially

quantified variables and gives the solutions to the verifier to see if they can be validated, or falsified by *counterexamples*. The counterexamples are then used as learned constraints for the synthesis procedure to find new solutions. This method has been shown effective for many challenging problems, frequently generating more optimized programs than the best manual implementations [5].

A direct application of CEGIS to decision problems over real numbers, however, suffers from several problems. CEGIS is complete in finite domains because it can explicitly enumerate solutions, which can not be done in continuous domains. Also, CEGIS ensures progress by avoiding duplication of solutions, while due to numerical sensitivity, precise control over real numbers is difficult. In this paper we propose methods that bypass such difficulties.

We propose an integration of the CEGIS method in the branch-and-prune framework as a generic algorithm for solving nonlinear $\exists\forall$-formulas over real numbers and prove that the algorithm is δ-complete. We achieve this goal by using CEGIS-based methods for turning universally-quantified constraints into pruning operators, which is then used in the branch-and-prune framework for the search for solutions on the existentially-quantified variables. In doing so, we take special care to ensure correct handling of numerical errors in the computation, so that δ-completeness can be established for the whole procedure.

The paper is organized as follows. We first review the background, and then present the details of the main algorithm in Section 3. We then give a rigorous proof of the δ-completeness of the procedure in Section 4. We demonstrated the effectiveness of the procedures on various global optimization and Lyapunov function synthesis problems in Section 5.

Related Work. Quantified formulas in real arithmetic can be solved using symbolic quantifier elimination (using cylindrical decomposition [6]), which is known to have impractically high complexity (double exponential [7]), and can not handle problems with transcendental functions. State-of-the-art SMT solvers such as CVC4 [8] and Z3 [9] provide quantifier support [10,11,12,13] but they are limited to decidable fragments of first-order logic. Optimization Modulo Theories (OMT) is a new field that focuses on solving a restricted form of quantified reasoning [14,15,16], focusing on linear formulas. Generic approaches to solving exists-forall problems such as [17] are generally based on CEGIS framework, and not intended to achieve completeness. Solving quantified constraints has been explored in the constraint solving community [18]. In general, existing work has not proposed algorithms that intend to achieve any notion of completeness for quantified problems in nonlinear theories over the reals.

2 Preliminaries

2.1 Delta-Decisions and CNF$^{\forall}$-Formulas

We consider first-order formulas over real numbers that can contain arbitrary nonlinear functions that can be numerically approximated, such as polynomials, exponential, and trignometric functions. Theoretically, such functions are called

Type-2 computable functions [19]. We write this language as $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$, formally defined as:

Definition 1 (The $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ Language). *Let $\mathcal{F}$ be the set of Type-2 computable functions. We define $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ to be the following first-order language:*

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

Remark 1. Negations are not needed as part of the base syntax, as it can be defined through arithmetic: $\neg(t > 0)$ is simply $-t \geq 0$. Similarly, an equality $t = 0$ is just $t \geq 0 \wedge -t \geq 0$. In this way we can put the formulas in normal forms that are easy to manipulate.

We will focus on the $\exists\forall$-formulas in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ in this paper. Decision problems for such formulas are equivalent to satisfiability of SMT with universally quantified variables, whose free variables are implicitly existentially quantified.

It is clear that, when the quantifier-free part of an $\exists\forall$ formula is in Conjunctive Normal Form (CNF), we can always push the universal quantifiers inside each conjunct, since universal quantification commute with conjunctions. Thus the decision problem for any $\exists\forall$-formula is equivalent to the satisfiability of formulas in the following normal form:

Definition 2 ($\text{CNF}^\forall$ Formulas in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$). *We say an $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula φ is in the $\text{CNF}^\forall$, if it is of the form*

$$\varphi(\boldsymbol{x}) := \bigwedge_{i=0}^{m} \left(\forall \boldsymbol{y} (\bigvee_{j=0}^{k_i} c_{ij}(\boldsymbol{x}, \boldsymbol{y})) \right) \tag{1}$$

where c_{ij} are atomic constraints. Each universally quantified conjunct of the formula, i.e.,

$$\forall \boldsymbol{y} (\bigvee_{j=0}^{k_i} c_{ij}(\boldsymbol{x}, \boldsymbol{y}))$$

is called as a **$\forall$-clause**. *Note that $\forall$-clauses only contain disjunctions and no nested conjunctions. If all the $\forall$-clauses are vacuous, we say $\varphi(\boldsymbol{x})$ is a* ground SMT *formula.*

The algorithms described in this paper will assume that an input formula is in $\text{CNF}^\forall$ form. We can now define the *δ-satisfiability* problems for $\text{CNF}^\forall$-formulas.

Definition 3 (Delta-Weakening/Strengthening). *Let $\delta \in \mathbb{Q}^+$ be arbitrary. Consider an arbitrary $\text{CNF}^\forall$-formula of the form*

$$\varphi(\boldsymbol{x}) := \bigwedge_{i=0}^{m} \left(\forall \boldsymbol{y} (\bigvee_{j=0}^{k_i} f_{ij}(\boldsymbol{x}, \boldsymbol{y}) \circ 0) \right)$$

where $\circ \in \{>, \geq\}$. *We define the* δ*-weakening of* $\varphi(\boldsymbol{x})$ *to be:*

$$\varphi^{-\delta}(\boldsymbol{x}) := \bigwedge_{i=0}^{m} \Big(\forall \boldsymbol{y}(\bigvee_{j=0}^{k_i} f_{ij}(\boldsymbol{x}, \boldsymbol{y}) \geq -\delta)\Big).$$

Namely, we weaken the right-hand sides of all atomic formulas from 0 *to* $-\delta$. *Note how the difference between strict and nonstrict inequality becomes unimportant in the* δ*-weakening. We also define its dual, the* δ*-strengthening of* $\varphi(\boldsymbol{x})$*:*

$$\varphi^{+\delta}(\boldsymbol{x}) := \bigwedge_{i=0}^{m} \Big(\forall \boldsymbol{y}(\bigvee_{j=0}^{k_i} f_{ij}(\boldsymbol{x}, \boldsymbol{y}) \geq +\delta)\Big).$$

Since the formulas in the normal form no longer contain negations, the relaxation on the atomic formulas is implied by the original formula (and thus weaker), as was easily shown in [1].

Proposition 1. *For any* φ *and* $\delta \in \mathbb{Q}^+$, $\varphi^{-\delta}$ *is logically weaker, in the sense that* $\varphi \to \varphi^{-\delta}$ *is always true, but not vice versa.*

Example 1. Consider the formula

$$\forall y\ f(x, y) = 0.$$

It is equivalent to the $\text{CNF}^{\forall}$-formula

$$(\forall y(-f(x, y) \geq 0) \wedge \forall y(f(x, y) \geq 0))$$

whose δ-weakening is of the form

$$(\forall y(-f(x, y) \geq -\delta) \wedge \forall y(f(x, y) \geq -\delta))$$

which is logically equivalent to

$$\forall y(\|f(x, y)\| \leq \delta).$$

We see that the weakening of $f(x, y) = 0$ by $\|f(x, y)\| \leq \delta$ defines a natural relaxation.

Definition 4 (Delta-Completeness). *Let* $\delta \in \mathbb{Q}^+$ *be arbitrary. We say an algorithm is* δ*-complete for* $\exists\forall$*-formulas in* $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$, *if for any input* $\text{CNF}^{\forall}$*-formula* φ, *it always terminates and returns one of the following answers correctly:*

- **unsat***:* φ *is unsatisfiable.*
- δ**-sat***:* $\varphi^{-\delta}$ *is satisfiable.*

When the two cases overlap, it can return either answer.

Algorithm 1 Branch-and-Prune

```
1: function SOLVE(f(x) = 0, B_x, δ)
2:     S ← {B_x}
3:     while S ≠ ∅ do
4:         B ← S.pop()
5:         B' ← FixedPoint(λB.B ∩ Prune(B, f(x) = 0), B)
6:         if B' ≠ ∅ then
7:             if ‖f(B')‖ > δ then
8:                 {B_1, B_2} ← Branch(B')
9:                 S.push({B_1, B_2})
10:            else
11:                return δ-sat
12:            end if
13:        end if
14:    end while
15:    return unsat
16: end function
```

2.2 The Branch-and-Prune Framework

A practical algorithm that has been shown to be δ-complete for ground SMT formulas is the *branch-and-prune* method developed for interval constraint propagation [20]. A description of the algorithm in the simple case of an equality constraint is in Algorithm 1.

The procedure combines *pruning* and *branching* operations. Let $\mathcal{B}$ be the set of all boxes (each variable assigned to an interval), and $\mathcal{C}$ a set of constraints in the language. FixedPoint(g, B) is a procedure computing a fixedpoint of a function $g : \mathcal{B} \to \mathcal{B}$ with an initial input B. A pruning operation $\mathsf{Prune} : \mathcal{B} \times \mathcal{C} \to \mathcal{B}$ takes a box $B \in \mathcal{B}$ and a constraint as input, and returns an ideally smaller box $B' \in \mathcal{B}$ (Line 5) that is guaranteed to still keep all solutions for all constraints if there is any. When such pruning operations do not make progress, the Branch procedure picks a variable, divides its interval by halves, and creates two sub-problems B_1 and B_2 (Line 8). The procedure terminates if either all boxes have been pruned to be empty (Line 15), or if a small box whose maximum width is smaller than a given threshold δ has been found (Line 11). In [2], it has been proved that Algorithm 1 is δ-complete iff the pruning operators satisfy certain conditions for being *well-defined* (Definition 5).

3 Algorithm

The core idea of our algorithm for solving $\text{CNF}^{\forall}$-formulas is as follows. We view the universally quantified constraints as a special type of pruning operators, which can be used to reduce possible values for the free variables based on their consistency with the universally-quantified variables. We then use these

special ∀-pruning operators in an overall branch-and-prune framework to solve the full formula in a δ-complete way. A special technical difficulty for ensuring δ-completeness is to control numerical errors in the recursive search for counterexamples, which we solve using *double-sided error control*. We also improve quality of counterexamples using local-optimization algorithms in the ∀-pruning operations, which we call *locally-optimized counterexamples*.

In the following sections we describe these steps in detail. For notational simplicity we will omit vector symbols and assume all variable names can directly refer to vectors of variables.

3.1 ∀-Clauses as Pruning Operators

Consider an arbitrary CNF$^\forall$-formula[4]

$$\varphi(x) := \bigwedge_{i=0}^{m} \Big(\forall y(\bigvee_{j=0}^{k_i} f_{ij}(x,y) \geq 0)\Big).$$

It is a conjunction of ∀-clauses as defined in Definition 2. Consequently, we only need to define pruning operators for ∀-clauses so that they can be used in a standard branch-and-prune framework. The full algorithm for such pruning operation is described in Algorithm 2.

Algorithm 2 ∀-Clause Pruning

function PRUNE(B_x, B_y, $\forall y \bigvee_{i=0}^{k} f_i(x,y) \geq 0$, δ', ε, δ)
  repeat
    $B_x^{\text{prev}} \leftarrow B_x$
    $\psi \leftarrow \bigwedge_i f_i(x,y) < 0$
    $\psi^{+\varepsilon} \leftarrow \mathsf{Strengthen}(\psi, \varepsilon)$
    $b \leftarrow \mathsf{Solve}(y, \psi^{+\varepsilon}, \delta')$ ▷ $0 < \delta' < \varepsilon < \delta$ should hold.
    if $b = \emptyset$ **then**
      return B_x ▷ No counterexample found, stop pruning.
    end if
    for $i \in \{0, ..., k\}$ **do**
      $B_i \leftarrow B_x \cap \mathsf{Prune}\Big(B_x, f_i(x,b) \geq 0\Big)$
    end for
    $B_x \leftarrow \bigsqcup_{i=0}^{k} B_i$
  until $B_x \neq B_x^{\text{prev}}$
  return B_x
end function

[4] Note that without loss of generality we only use nonstrict inequality here, since in the context of δ-decisions the distinction between strict and nonstrict inequalities as not important, as explained in Definition 3.

In Algorithm 2, the basic idea is to use special y values that witness the *negation* of the original constraint to prune the box assignment on x. The two core steps are as follows.

1. Counterexample generation (Line 4 to 9). The query for a counterexample ψ is defined as the negation of the quantifier-free part of the constraint (Line 4). The method $\mathsf{Solve}(y, \psi, \delta)$ means to obtain a solution for the variables y δ-satisfying the logic formula ψ. When such a solution is found, we have a counterexample that can falsify the $\forall$-clause on some choice of x. Then we use this counterexample to prune on the domain of x, which is currently B_x. The strengthening operation on ψ (Line 5), as well as the choices of ε and δ', will be explained in the next subsection.
2. Pruning on x (Line 10 to 13). In the counterexample generation step, we have obtained a counterexample b. The pruning operation then uses this value to prune on the current box domain B_x. Here we need to be careful about the logical operations. For each constraint, we need to take the intersection of the pruned results on the counterexample point (Line 11). Then since the original clause contains the disjunction of all constraints, we need to take the box-hull ($\bigsqcup$) of the pruned results (Line 13).

We can now put the pruning operators defined for all $\forall$-clauses in the overall branch-and-prune framework shown in Algorithm 1.

The pruning algorithms are inspired by the CEGIS loop, but are different in multiple ways. First, we never explicitly compute any candidate solution for the free variables. Instead, we only prune on their domain boxes. This ensures that the size of domain box decreases (together with branching operations), and the algorithm terminates. Secondly, we do not explicitly maintain a collection of constraints. Each time the pruning operation works on previous box – i.e., the learning is done on the model level instead of constraint level. On the other hand, being unable to maintain arbitrary Boolean combinations of constraints requires us to be more sensitive to the type of Boolean operations needed in the pruning results, which is different from the CEGIS approach that treats solvers as black boxes.

3.2 Double-Sided Error Control

To ensure the correctness of Algorithm 2, it is necessary to avoid spurious counterexamples which do *not* satisfy the negation of the quantified part in a $\forall$-clause. We illustrate this condition by consider a *wrong* derivation of Algorithm 2 where we do not have the strengthening operation on Line 5 and try to find a counterexample by directly executing $b \leftarrow \text{Solve}(y, \psi = \bigwedge_{i=0}^{k} f_i(x, y) < 0, \delta)$. Note that the counterexample query ψ can be highly nonlinear in general and not included in a decidable fragment. As a result, it must employ a delta-decision procedure (i.e. Solve with $\delta' \in \mathbb{Q}^+$) to find a counterexample. A consequence of relying on a delta-decision procedure in the counterexample generation step is

that we may obtain a spurious counterexample b such that for some $x = a$:

$$\bigwedge_{i=0}^{k} f_i(a,b) \leq \delta \quad \text{instead of} \quad \bigwedge_{i=0}^{k} f_i(a,b) < 0.$$

Consequently the following pruning operations fail to reduce their input boxes because a spurious counterexample does not witness any inconsistencies between x and y. As a result, the fixedpoint loop in this $\forall$-Clause pruning algorithm will be terminated immediately after the first iteration. This makes the outermost branch-and-prune framework (Algorithm 1), which employs this pruning algorithm, solely rely on branching operations. It can claim that the problem is δ-satisfiable while providing an arbitrary box B as a model which is small enough ($\|B\| \leq \delta$) but does not include a δ-solution.

To avoid spurious counterexamples, we directly strengthen the counterexample query with $\varepsilon \in \mathbb{Q}^+$ to have

$$\psi^{+\varepsilon} := \bigwedge_{i=0}^{k} f_i(a,b) \leq -\varepsilon.$$

Then we choose a weakening parameter $\delta' \in \mathbb{Q}$ in solving the strengthened formula. By analyzing the two possible outcomes of this counterexample search, we show the constraints on δ', ε, and δ which guarantee the correctness of Algorithm 2:

- **δ'-sat case**: We have a and b such that $\bigwedge_{i=0}^{k} f_i(a,b) \leq -\varepsilon + \delta'$. For $y = b$ to be a valid counterexample, we need $-\varepsilon + \delta' < 0$. That is, we have

$$\delta' < \varepsilon. \tag{2}$$

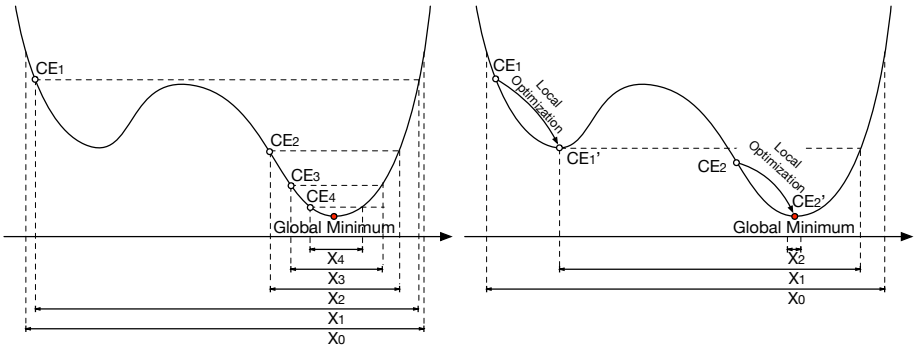
  In other words, the strengthening factor ε should be greater than the weakening parameter δ' in the counterexample search step.
- **unsat case**: By checking the absence of counterexamples, it proved that $\forall y \bigvee_{i=0}^{k} f_i(x,y) \geq -\varepsilon$ for all $x \in B_x$. Recall that we want to show that $\forall y \bigvee_{i=0}^{k} f_i(x,y) \geq -\delta$ holds for some $x = a$ when Algorithm 1 uses this pruning algorithm and returns δ-sat. To ensure this property, we need the following constraint on ε and δ:

$$\varepsilon < \delta. \tag{3}$$

3.3 Locally-Optimized Counterexamples

The performance of the pruning algorithm for $\text{CNF}^{\forall}$-formulas depends on the quality of the counterexamples found during the search.

Figure 1a illustrates this point by visualizing a pruning process for an unconstrained minimization problem, $\exists x \in X_0 \forall y \in X_0 f(x) \leq f(y)$. As it finds a series of counterexamples CE_1, CE_2, CE_3, and CE_4, the pruning algorithms



(a) Without local optimization. (b) With local optimization.

Fig. 1: Illustrations of the pruning algorithm for $\text{CNF}^{\forall}$-formula with and without using local optimization.

uses those counterexamples to contract the interval assignment on X from X_0 to X_1, X_2, X_3, and X_4 in sequence. In the search for a counterexample (Line 6 of Algorithm 2), it solves the strengthened query, $f(x) > f(y) + \delta$. Note that the query only requires a counterexample $y = b$ to be δ-away from a candidate x while it is clear that the further a counterexample is away from candidates, the more effective the pruning algorithm is.

Based on this observation, we present a way to improve the performance of the pruning algorithm for $\text{CNF}^{\forall}$-formulas. After we obtain a counterexample b, we locally-optimize it with the counterexample query ψ so that it "further violates" the constraints. Figure 1b illustrates this idea. The algorithm first finds a counterexample CE_1 then refines it to CE_1' by using a local-optimization algorithm (similarly, $\text{CE}_2 \rightarrow \text{CE}_2'$). Clearly, this refined counterexample gives a stronger pruning power than the original one. This refinement process can also help the performance of the algorithm by reducing the number of total iterations in the fixedpoint loop.

The suggested method is based on the assumption that local-optimization techniques are cheaper than finding a global counterexample using interval propagation techniques. In our experiments, we observed that this assumption holds practically. We will report the details in Section 5.

4 δ-Completeness

We now prove that the proposed algorithm is δ-complete for arbitrary $\text{CNF}^{\forall}$ formulas in $\mathcal{L}_{\mathbb{R}_{\mathcal{F}}}$. In the work of [2], δ-completeness has been proved for branch-and-prune for ground SMT problems, under the assumption that the pruning operators are *well-defined*. Thus, the key for our proof here is to show that the $\forall$-pruning operators satisfy the conditions of well-definedness.

The notion of a well-defined pruning operator is defined in [2] as follows.

Definition 5. *Let ϕ be a constraint, and $\mathcal{B}$ be the set of all boxes in $\mathbb{R}^n$. A pruning operator is a function* $\mathsf{Prune} : \mathcal{B} \times \mathcal{C} \to \mathcal{B}$. *We say such a pruning operator is well-defined, if for any $B \in \mathcal{B}$, the following conditions are true:*

1. $\mathsf{Prune}(B, \phi) \subseteq B$.
2. $B \cap \{a \in \mathbb{R}^n : \phi(a) \text{ is true.}\} \subseteq \mathsf{Prune}(B, \phi)$.
3. *Write* $\mathsf{Prune}(B, \phi) = B'$. *There exists a constant $c \in \mathbb{Q}^+$, such that, if $B' \neq \emptyset$ and $\|B'\| < \varepsilon$ for some $\varepsilon \in \mathbb{Q}^+$, then for all $a \in B'$, $\phi^{-c\varepsilon}(a)$ is true.*

We will explain the intuition behind these requirements in the next proof, which aims to establish that Algorithm 2 defines a well-defined pruning operator.

Lemma 1 (Well-definedness of $\forall$-Pruning). *Consider an arbitrary $\forall$-clause in the generic form*

$$c(x) := \forall y \Big(f_1(x, y) \geq 0 \vee ... \vee f_k(x, y) \geq 0 \Big).$$

Suppose the pruning operators for $f_1 \geq 0, ..., f_k \geq 0$ are well-defined, then the $\forall$-pruning operation for $c(x)$ as described in Algorithm 2 is well-defined.

Proof. We prove that the pruning operator defined by Algorithm 2 satisfies the three conditions in Definition 5. Let $B_0, ..., B_k$ be a sequence of boxes, where B_0 is the input box B_x and B_k is the returned box B, which is possibly empty.

The first condition requires that the pruning operation for $c(x)$ is reductive. That is, we want to show that $B_x \subseteq B_x^{\text{prev}}$ holds in Algorithm 2. If it does not find a counterexample (Line 8), we have $B_x = B_x^{\text{prev}}$. So the condition holds trivially. Consider the case where it finds a counterexample b. The pruned box B_x is obtained through box-hull of all the B_i boxes (Line 13), which are results of pruning on B_x^{prev} using ordinary constraints of the form $f_i(x, b) \geq 0$ (Line 11), for a counterexample b. Following the assumption that the pruning operators are well-defined for each ordinary constraint f_i used in the algorithm, we know that $B_i \subseteq B_x^{\text{prev}}$ holds as a loop invariant for the loop from Line 10 to Line 12. Thus, taking the box-hull of all the B_i, we obtain B_x that is still a subset of B_x^{prev}.

The second condition requires that the pruning operation does not eliminate real solutions. Again, by the assumption that the pruning operation on Line 11 does not lose any valid assignment on x that makes the $\forall$-clause true. In fact, since y is universally quantified, any choice of assignment $y = b$ will preserve solution on x as long as the ordinary pruning operator is well-defined. Thus, this condition is easily satisfied.

The third condition is the most nontrivial to establish. It ensures that when the pruning operator does not prune a box to the empty set, then the box should not be "way off", and in fact, should contain points that satisfy an appropriate relaxation of the constraint. We can say this is a notion of "faithfulness" of the pruning operator. For constraints defined by simple continuous functions, this can be typically guaranteed by the modulus of continuity of the function (Lipschitz constants as a special case). Now, in the case of $\forall$-clause pruning, we need to prove that the faithfulness of the ordinary pruning operators that are

used translates to the faithfulness of the $\forall$-clause pruning results. First of all, this condition would not hold, if we do not have the strengthening operation when searching for counterexamples (Line 5). As is shown in Example 1, because of the weakening that δ-decisions introduce when searching for a counterexample, we may obtain a *spurious counterexample* that does not have pruning power. In other words, if we keep using a wrong counterexample that already satisfies the condition, then we are not able to rule out wrong assignments on x. Now, since we have introduced ε-strengthening at the counterexample search, we know that b obtained on Line 6 is a true counterexample. Thus, for some $x = a$, $f_i(a, b) < 0$ for every i. By assumption, the ordinary pruning operation using b on Line 11 guarantees faithfulness. That is, suppose the pruned result B_i is not empty and $\|B_i\| \leq \varepsilon$, then there exists constant c_i such that $f_i(x, b) \geq -c_i\varepsilon$ is true. Thus, we can take the $c = \min_i c_i$ as the constant for the pruning operator defined by the full clause, and conclude that the disjunction $\bigvee_{i=0}^{k} f_i(x, y) < -c\varepsilon$ holds for $\|B_x\| \leq \varepsilon$.

Using the lemma, we follow the results in [2], and conclude that the branch-and-prune method in Algorithm 1 is delta-complete:

Theorem 1 (δ-Completeness). *For any $\delta \in \mathbb{Q}^+$, using the proposed $\forall$-pruning operators defined in Algorithm 2 in the branch-and-prune framework described in Algorithm 1 is δ-complete for the class of CNF$^\forall$-formulas in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$, assuming that the pruning operators for all the base functions are well-defined.*

Proof. Following Theorem 4.2 (δ-Completeness of ICP$_\varepsilon$) in [2], a branch-and-prune algorithm is δ-complete iff the pruning operators in the algorithm are all well-defined. Following Lemma 1, Algorithm 2 always defines well-defined pruning operators, assuming the pruning operators for the base functions are well-defined. Consequently, Algorithm 2 and Algorithm 1 together define a delta-complete decision procedure for CNF$^\forall$-problems in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$.

5 Evaluation

Implementation We implemented the algorithms on top of dReal [21], an open-source delta-SMT framework. We used IBEX-lib [22] for interval constraints pruning and CLP [23] for linear programming. For local optimization, we used NLopt [24]. In particular, we used SLSQP (Sequential Least-Squares Quadratic Programming) local-optimization algorithm [25] for differentiable constraints and COBYLA (Constrained Optimization BY Linear Approximations) local-optimization algorithm [26] for non-differentiable constraints. The prototype solver is able to handle $\exists\forall$-formulas that involve most standard elementary functions, including power, exp, log, $\sqrt{\cdot}$, trigonometric functions (sin, cos, tan), inverse trigonometric functions (arcsin, arccos, arctan), hyperbolic functions (sinh, cosh, tanh), etc.

Experiment environment All experiments were ran on a 2017 Macbook Pro with 2.9 GHz Intel Core i7 and 16 GB RAM running MacOS 10.13.4. All code and benchmarks are available at `https://github.com/dreal/CAV18`.

Name	Solution			Time (sec)		
	Global	No L-Opt.	L-Opt.	No L-Opt.	L-Opt.	Speed Up
Ackley 2D	0.00000	0.00000	0.00000	0.0579	0.0047	12.32
Ackley 4D	0.00000	0.00005	0.00000	8.2256	0.1930	42.62
Aluffi Pentini	-0.35230	-0.35231	-0.35239	0.0321	0.1868	0.17
Beale	0.00000	0.00003	0.00000	0.0317	0.0615	0.52
Bohachevsky1	0.00000	0.00006	0.00000	0.0094	0.0020	4.70
Booth	0.00000	0.00006	0.00000	0.5035	0.0020	251.75
Brent	0.00000	0.00006	0.00000	0.0095	0.0017	5.59
Bukin6	0.00000	0.00003	0.00003	0.0093	0.0083	1.12
Cross in Tray	-2.06261	-2.06254	-2.06260	0.5669	0.1623	3.49
Easom	-1.00000	-1.00000	-1.00000	0.0061	0.0030	2.03
EggHolder	-959.64070	-959.64030	-959.64031	0.0446	0.0211	2.11
Holder Table2	-19.20850	-19.20846	-19.20845	52.9152	41.7004	1.27
Levi N13	0.00000	0.00000	0.00000	0.1383	0.0034	40.68
Ripple 1	-2.20000	-2.20000	-2.20000	0.0059	0.0065	0.91
Schaffer F6	0.00000	0.00004	0.00000	0.0531	0.0056	9.48
Testtube Holder	-10.87230	-10.87227	-10.87230	0.0636	0.0035	18.17
Trefethen	-3.30687	-3.30681	-3.30685	3.0689	1.4916	2.06
W Wavy	0.00000	0.00000	0.00000	0.1234	0.0138	8.94
Zettl	-0.00379	-0.00375	-0.00379	0.0070	0.0069	1.01
Rosenbrock Cubic	0.00000	0.00005	0.00002	0.0045	0.0036	1.25
Rosenbrock Disk	0.00000	0.00002	0.00000	0.0036	0.0028	1.29
Mishra Bird	-106.76454	-106.76449	-106.76451	1.8496	0.9122	2.03
Townsend	-2.02399	-2.02385	-2.02390	2.6216	0.5817	4.51
Simionescu	-0.07262	-0.07199	-0.07200	0.0064	0.0048	1.33

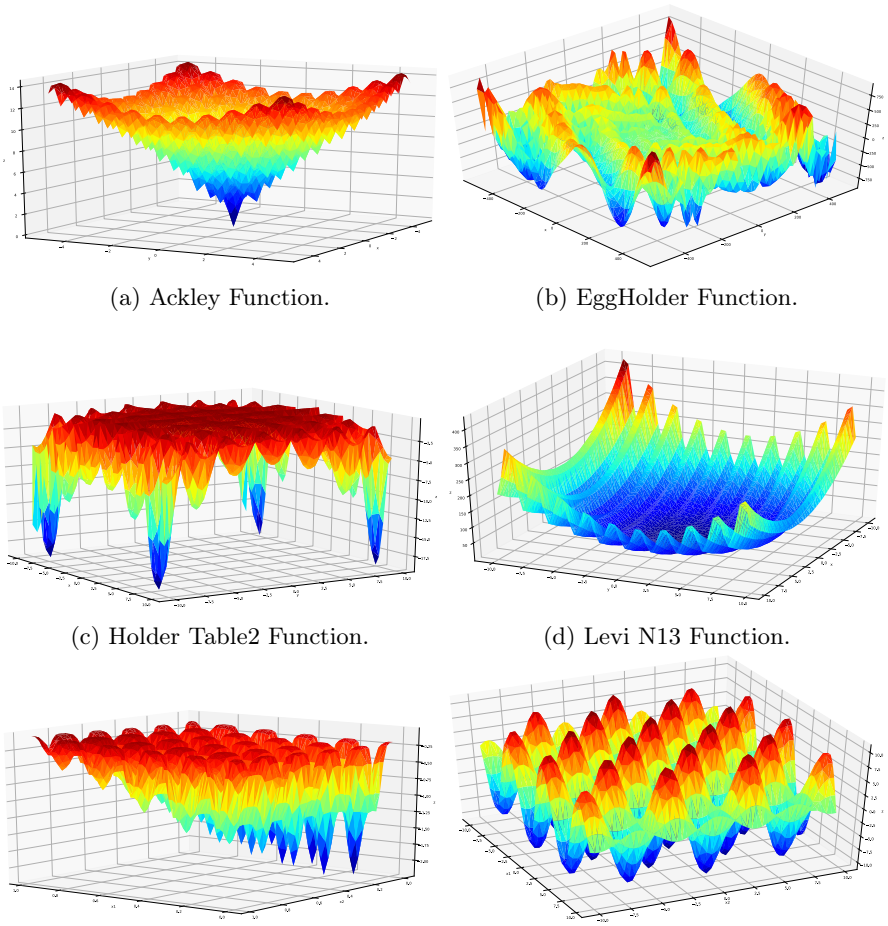
Table 1: Experimental results for nonlinear global optimization problems: The first 19 problems (Ackley 2D – Zettl) are unconstrained optimization problems and the last five problems (Rosenbrock Cubic – Simionescu) are constrained optimization problems. We ran our prototype solver over those instances with and without local-optimization option ("L-Opt." and "No L-Opt." columns) and compared the results. We chose $\delta = 0.0001$ for all instances.

Parameters In the experiments, we chose the strengthening parameter $\epsilon = 0.99\delta$ and the weakening parameter in the counterexample search $\delta' = 0.98\delta$. In each call to NLopt, we used 1e−6 for both of absolute and relative tolerances on function value, 1e−3 seconds for a timeout, and 100 for the maximum number of evaluations. These values are used as stopping criteria in NLopt.

5.1 Nonlinear Global Optimization

We encoded a range of highly nonlinear ∃∀-problems from constrained and unconstrained optimization literature [27,28]. Note that the standard optimization problem

$$\min f(x) \text{ s.t. } \varphi(x), \quad x \in \mathbb{R}^n,$$



(a) Ackley Function. (b) EggHolder Function.

(c) Holder Table2 Function. (d) Levi N13 Function.

(e) Ripple 1 Function. (f) Testtube Holder Function.

Fig. 2: Nonlinear Global Optimization Examples.

can be encoded as the logic formula:

$$\varphi(x) \wedge \forall y\Big(\varphi(y) \rightarrow f(x) \leq f(y)\Big).$$

As plotted in Figure 2, these optimization problems are non-trivial: they are highly non-convex problems that are designed to test global optimization or genetic programming algorithms. Many such functions have a large number of local minima. For example, Ripple 1 Function [27]

$$f(x_1, x_2) = \sum_{i=1}^{2} -e^{-2(\log 2)\left(\frac{x_1-0.1}{0.8}\right)^2}(\sin^6(5\pi x_i) + 0.1\cos^2(500\pi x_i))$$

defined in $x_i \in [0, 1]$ has 252004 local minima with the global minima $f(0.1, 0.1) = -2.2$. As a result, local-optimization algorithms such as gradient-descent would not work for these problems for itself. By encoding them as $\exists\forall$-problems, we can perform guaranteed global optimization on these problems.

Table 1 provides a summary of the experiment results. First, it shows that we can find minimum values which are close to the known global solutions. Second, it shows that enabling the local-optimization technique speeds up the solving process significantly for 20 instances out of 23 instances.

5.2 Synthesizing Lyapunov Function for Dynamical System

We show that the proposed algorithm is able to synthesize Lyapunov functions for nonlinear dynamic systems described by a set of ODEs:

$$\dot{\boldsymbol{x}}(t) = f_i(\boldsymbol{x}(t)), \quad \forall \boldsymbol{x}(t) \in X_i.$$

Our approach is different from a recent related-work [29] where they used dReal only to verify a candidate function which was found by a simulation-guided algorithm. In contrast, we want to do both of search and verify steps by solving a single $\exists\forall$-formula. Note that to verify a Lyapunov candidate function $v : X \to \mathbb{R}^+$, we need to show that the function v satisfies the following conditions:

$$\begin{aligned} \forall \boldsymbol{x} \in X \setminus \mathbf{0}\ v(\boldsymbol{x})(\mathbf{0}) = 0 \\ \forall \boldsymbol{x} \in X\ \nabla v(\boldsymbol{x}(t))^T \cdot f_i(\boldsymbol{x}(t)) \leq 0. \end{aligned}$$

We assume that a Lyapunov function is a polynomial of some fixed degrees over $\boldsymbol{x}$, that is, $v(\boldsymbol{x}) = \boldsymbol{z}^T \boldsymbol{P} \boldsymbol{z}$ where $\boldsymbol{z}$ is a vector of monomials over $\boldsymbol{x}$ and P is a symmetric matrix. Then, we can encode this synthesis problem into the $\exists\forall$-formula:

$$\begin{aligned} \exists \boldsymbol{P}\ [&(v(\boldsymbol{x}) = (\boldsymbol{z}^T \boldsymbol{P} \boldsymbol{z})) \wedge \\ &(\forall \boldsymbol{x} \in X \setminus \mathbf{0}\ v(\boldsymbol{x})(\mathbf{0}) = 0) \wedge \\ &(\forall \boldsymbol{x} \in X\ \nabla v(\boldsymbol{x}(t))^T \cdot f_i(\boldsymbol{x}(t)) \leq 0)] \end{aligned}$$

In the following sections, we show that we can handle two examples in [29].

Normalized Pendulum Given a standard pendulum system with normalized parameters

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\sin(x_1) - x_2 \end{bmatrix}$$

and a quadratic template for a Lyapunov function $v(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{P} \boldsymbol{x} = c_1 x_1 x_2 + c_2 x_1^2 + c_3 x_2^2$, we can encode this synthesis problem into the following $\exists\forall$-formula:

$$\begin{aligned} \exists c_1 c_2 c_3\ \forall x_1 x_2\ [&((50 c_3 x_1 x_2 + 50 x_1^2 c_1 + 50 x_2^2 c_2 > 0.5) \wedge \\ &(100 c_1 x_1 x_2 + 50 x_2 c_3 + (-x_2 - \sin(x_1)(50 x_1 c_3 + 100 x_2 c_2)) < -0.5)) \vee \\ &\neg((0.01 \leq x_1^2 + x_2^2) \wedge (x_1^2 + x_2^2 \leq 1))] \end{aligned}$$

Our prototype solver takes 44.184 seconds to synthesize the following function as a solution to the problem for the bound $\|\boldsymbol{x}\| \in [0.1, 1.0]$ and $c_i \in [0.1, 100]$ using $\delta = 0.05$:

$$v = 40.6843x_1x_2 + 35.6870x_1^2 + 84.3906x_2^2.$$

Damped Mathieu System Mathieu dynamics are time-varying and defined by the following ODEs:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}x_2\\ -x_2 - (2+\sin(t))x_1\end{bmatrix}.$$

Using a quadratic template for a Lyapunov function $v(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x} = c_1x_1x_2 + c_2x_1^2 + c_3x_2^2$, we can encode this synthesis problem into the following $\exists\forall$-formula:

$$\begin{aligned}\exists c_1c_2c_3\ \forall x_1x_2t\ [&(50x_1x_2c_2 + 50x_1^2c_1 + 50x_2^2c_3 > 0)\wedge\\ &(100c_1x_1x_2 + 50x_2c_2 + (-x_2 - x_1(2+\sin(t)))(50x_1c_2 + 100x_2c_3) < 0)\\ &\vee \neg((0.01 \le x_1^2 + x_2^2) \wedge (0.1 \le t) \wedge (t \le 1) \wedge (x_1^2 + x_2^2 \le 1))]\end{aligned}$$

Our prototype solver takes 26.533 seconds to synthesize the following function as a solution to the problem for the bound $\|\boldsymbol{x}\| \in [0.1, 1.0]$, $t \in [0.1, 1.0]$, and $c_i \in [45, 98]$ using $\delta = 0.05$:

$$V = 54.6950x_1x_2 + 90.2849x_1^2 + 50.5376x_2^2.$$

6 Conclusion

We have described delta-decision procedures for solving exists-forall formulas in the first-order theory over the reals with computable real functions. These formulas can encode a wide range of hard practical problems such as general constrained optimization and nonlinear control synthesis. We use a branch-and-prune framework, and design special pruning operators for universally-quantified constraints such that the procedures can be proved to be delta-complete, where suitable control of numerical errors is crucial. We demonstrated the effectiveness of the procedures on various global optimization and Lyapunov function synthesis problems.

References

1. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: LICS. (2012) 305–314
2. Gao, S., Avigad, J., Clarke, E.M.: Delta-complete decision procedures for satisfiability over the reals. In Gramlich, B., Miller, D., Sattler, U., eds.: IJCAR. Volume 7364 of Lecture Notes in Computer Science., Springer (2012) 286–300

3. Kong, S., Gao, S., Chen, W., Clarke, E.: dReach: δ-reachability analysis for hybrid systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer (2015) 200–205
4. Ratschan, S.: Applications of quantified constraint solving over the reals-bibliography. arXiv preprint arXiv:1205.5571 (2012)
5. Solar-Lezama, A.: Program synthesis by sketching. University of California, Berkeley (2008)
6. Collins, G.E.: Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Automata Theory and Formal Languages. (1975) 134–183
7. Brown, C.W., Davenport, J.H.: The complexity of quantifier elimination and cylindrical algebraic decomposition. In: ISSAC-2007
8. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: Cvc4. In: Proceedings of the 23rd International Conference on Computer Aided Verification. CAV'11, Berlin, Heidelberg, Springer-Verlag (2011) 171–177
9. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. TACAS'08/ETAPS'08, Berlin, Heidelberg, Springer-Verlag (2008) 337–340
10. Moura, L., Bjørner, N.: Efficient e-matching for SMT solvers. In: Proceedings of the 21st International Conference on Automated Deduction: Automated Deduction. CADE-21, Berlin, Heidelberg, Springer-Verlag (2007) 183–198
11. Bjørner, N., Phan, A.D., Fleckenstein, L.: νz - an optimizing SMT solver. In Baier, C., Tinelli, C., eds.: Tools and Algorithms for the Construction and Analysis of Systems, Berlin, Heidelberg, Springer Berlin Heidelberg (2015) 194–199
12. Ge, Y., Barrett, C., Tinelli, C.: Solving quantified verification conditions using satisfiability modulo theories. In Pfenning, F., ed.: Proceedings of the 21st International Conference on Automated Deduction (CADE-21), Bremen, Germany. Volume 4603 of Lecture Notes in Computer Science., Springer (2007) 167–182
13. Reynolds, A., Deters, M., Kuncak, V., Tinelli, C., Barrett, C.W.: Counterexample-guided quantifier instantiation for synthesis in SMT. In Kroening, D., Pasareanu, C.S., eds.: Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II. Volume 9207 of Lecture Notes in Computer Science., Springer (2015) 198–216
14. Nieuwenhuis, R., Oliveras, A. In: On SAT Modulo Theories and Optimization Problems. Springer Berlin Heidelberg, Berlin, Heidelberg (2006) 156–169
15. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability modulo the theory of costs: Foundations and applications. In: Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. TACAS'10, Berlin, Heidelberg, Springer-Verlag (2010) 99–113
16. Sebastiani, R., Tomasi, S.: Optimization in SMT with LA(Q) cost functions. In: Proceedings of the 6th International Joint Conference on Automated Reasoning. IJCAR'12, Berlin, Heidelberg, Springer-Verlag (2012) 484–498
17. Dutertre, B.: Solving exists/forall problems with yices. In: Workshop on Satisfiability Modulo Theories. (2015)
18. Nightingale, P. In: Consistency for Quantified Constraint Satisfaction Problems. Springer Berlin Heidelberg, Berlin, Heidelberg (2005) 792–796
19. Weihrauch, K.: Computable Analysis: An Introduction. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2000)

20. Benhamou, F., Granvilliers, L.: Continuous and interval constraints. In Rossi, F., van Beek, P., Walsh, T., eds.: Handbook of Constraint Programming. Elsevier (2006)
21. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: CADE. (2013) 208–214
22. Trombettoni, G., Araya, I., Neveu, B., Chabert, G.: Inner regions and interval linearizations for global optimization. In Burgard, W., Roth, D., eds.: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011, AAAI Press (2011)
23. Lougee-Heimer, R.: The common optimization interface for operations research: Promoting open-source software in the operations research community. IBM J. Res. Dev. **47**(1) (January 2003) 57–66
24. Johnson, S.G.: The NLopt nonlinear-optimization package. (2011)
25. Kraft, D.: Algorithm 733: Tomp–fortran modules for optimal control calculations. ACM Trans. Math. Softw. **20**(3) (September 1994) 262–281
26. Powell, M.: Direct search algorithms for optimization calculations. Acta numerica **7** (1998) 287–336
27. Jamil, M., Yang, X.S.: A literature survey of benchmark functions for global optimisation problems. International Journal of Mathematical Modelling and Numerical Optimisation **4**(2) (2013) 150–194
28. Wikipedia contributors: Test functions for optimization — Wikipedia, The Free Encyclopedia (2017)
29. Kapinski, J., Deshmukh, J.V., Sankaranarayanan, S., Arechiga, N.: Simulation-guided lyapunov analysis for hybrid dynamical systems. In: HSCC'14, Berlin, Germany, April 15-17, 2014. (2014) 133–142