A Complete Algorithm for Optimization Modulo Nonlinear Real Arithmetic

Fuqi Jia[1, 3], **Yuhang Dong**[2, 3], **Rui Han**[1, 3], **Pei Huang**[4], **Minghao Liu**[5], **Feifei Ma**[2, 3 *], **Jian Zhang**[1, 3 *]

[1]SKLCS and Key Laboratory of System Software, ISCAS, Beijing, China
[2]Laboratory of Parallel Software and Computational Science, ISCAS, Beijing, China
[3]University of Chinese Academy of Sciences, Beijing, China
[4] Stanford University, Stanford, USA
[5] University of Oxford, Oxford, UK
{jiafq, maff, zj}@ios.ac.cn

Abstract

Optimization Modulo Nonlinear Real Arithmetic, abbreviated as OMT(NRA), generally focuses on optimizing a given objective subject to quantifier-free Boolean combinations of primitive constraints, including Boolean variables, polynomial equations, and inequalities. It is widely applicable in areas like program verification, analysis, planning, and so on. The existing solver, OptiMathSAT, officially supporting OMT(NRA), employs an incomplete algorithm. We present a sound and complete algorithm, Optimization Cylindrical Algebraic Covering (OCAC), integrated within the Conflict-Driven Clause Learning (CDCL) framework, specifically tailored for OMT(NRA) problems. We establish the correctness and termination of CDCL(OCAC) and explore alternative approaches using cylindrical algebraic decomposition (CAD) and first-order formulations. Our work includes the development of the first complete OMT solver for NRA, demonstrating significant performance improvements. In benchmarks generated from SMT-LIB instances, our algorithm finds the optimum value in about 150% more instances compared to the current leading solver, OptiMathSAT.

Supplementary — https://github.com/fuqi-jia/AAAI25

Introduction

Satisfiability Modulo Theories (SMT) (Kroening and Strichman 2016; Barrett et al. 2021) is a problem of checking the satisfiability of a first-order logic formula under specific theories (Barrett, Dill, and Levitt 1998; Dutertre and de Moura 2006; Jovanovic and de Moura 2012; Liang et al. 2016; Jia et al. 2023b; Zhang, Li, and Cai 2024). Many real-world problems involve not only determining satisfiability but also optimizing a given objective function. It leads to the emergence of more challenging and practical problems named Optimization Modulo Theories (OMT) (Nieuwenhuis and Oliveras 2006; Ma, Yan, and Zhang 2012; Bjørner, Phan, and Fleckenstein 2015; Sebastiani and Trentin 2020). An OMT formula generally combines constraints from a particular theory with an objective function for optimization. The development of OMT solvers spans a variety of theories, such as linear real / integer arithmetic (Sebastiani and Tomasi 2012; Bjørner, Phan, and Fleckenstein 2015; Sebastiani and Trentin 2015; He et al. 2024), bit vectors (Nadel and Ryvchin 2016; Trentin and Sebastiani 2021), and floating point arithmetic (Trentin and Sebastiani 2019). It has numerous applications such as program verification (Liu et al. 2017; Karpenkov 2017; Ratschan 2017), system safety analysis (Bertolissi, dos Santos, and Ranise 2018; Paoletti et al. 2019; Wang et al. 2021; Erata et al. 2023), software analysis and testing (Zhang 2000; Zhang, Ma, and Zhang 2012; Zhang et al. 2014; Henry et al. 2014; Karpenkov, Friedberger, and Beyer 2016; Jiang et al. 2017; Yao et al. 2021), planning (Roselli, Bengtsson, and Åkesson 2018; Yan et al. 2019; Leofante et al. 2019; Marchetto et al. 2021; Jin et al. 2021; Leofante 2023) and machine learning (Teso, Sebastiani, and Passerini 2017; Sivaraman et al. 2020; Huang et al. 2022, 2024). This paper focuses on OMT(NRA) problems.

OMT(NRA) has not been thoroughly investigated. The only solver officially supporting OMT(NRA) is OptiMathSAT (Bigarella et al. 2021), which can handle challenging instances. However, it uses an incomplete algorithm. The algorithm is based on incremental linearization (Cimatti et al. 2018) to calculate an approximate optimum within the linear real arithmetic theory. Subsequently, it refines the objective by performing multiple iterations of SMT solving with boundary constraints to pinpoint the optimum. The limitation is that it can only partially handle optimums that are irrational numbers, infinitely close to a given value ($c \pm \epsilon$ as $\epsilon \to 0$) or infinitely large or small ($\pm\infty$). In practice, it cannot solve the first case and only partly handles the others.

Example 1. *Consider an OMT(NRA) formula,*

$$\min x + y, \quad s.t.\ x^2 + y^2 = 1,$$

the objective $x + y$ has a minimum value of $-\sqrt{2}$.

For the simple OMT instance as in Example 1, OptiMathSAT (Bigarella et al. 2021) and Z3 [1] (Li et al. 2014; Bjørner, Phan, and Fleckenstein 2015) cannot find the optimum in 2 hours. Each solution provided by the SMT solver incrementally approximates the final result of $-\sqrt{2}$. Due to the density of rational numbers, the solver is persistently close to

*Corresponding authors.
Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]Z3 does not officially support OMT(NRA) formulas (some related issues: issue 2247, issue 4872, issue 5264, issue 5339), but sometimes can give correct answers.

the value but cannot precisely achieve the optimum. Similar situations arise when the solution is infinitely close to a number ($c \pm \epsilon, \epsilon \to 0$) or is infinitely large or small ($\pm\infty$). Since these are not specific numbers, no model can accurately represent these cases. The solver can only approximate values within each iteration.

A recent work, General Optimization Modulo Theories (GOMT) (Tsiskaridze, Barrett, and Tinelli 2024), presents a generalization of the OMT problem and introduces a theory-agnostic calculus that unifies single- and multi-objective optimization within a single framework. An OMT(NRA) or NRA optimization solver can be a subroutine of the general GOMT calculus.

This paper proposes a sound and complete algorithm, CDCL(OCAC), for OMT(NRA). It integrates the Conflict-Driven Clause Learning (CDCL) framework (Ganzinger et al. 2004; Kroening and Strichman 2016; Barrett et al. 2021) and the Optimization Cylindrical Algebraic Covering (OCAC) algorithm we proposed. The CDCL(T) framework systematically explores all paths in the search tree. Meanwhile, the OCAC algorithm focuses on finding the optimum within a specific path.

OCAC first converts the objective term into a variable. Then it uses the projection operator to obtain the projected polynomial set, which defines the boundary of satisfiable regions. It can represent the optimum through a real root derived from the polynomial set. Furthermore, an open boundary signifies that the optimum is infinitely close to a certain number ($c \pm \epsilon, \epsilon \to 0$) or infinitely small or large ($\pm\infty$). The characteristics of projection operators ensure the correctness of our algorithm and guarantee the termination.

We also investigate several variants of the algorithms. Cylindrical Algebraic Decomposition (CAD) (Collins 1975; Arnon, Collins, and McCallum 1984) is an algorithm that decomposes space into sign-invariant regions. It can be adapted for optimization (Nan et al. 2017; Wolfram 2024). We introduce a first-order formulation for the OMT(NRA) problem. These variants, along with OptiMathSAT (Bigarella et al. 2021), serve as baselines for comparison with CDCL(OCAC), enabling a comprehensive evaluation.

We summarize our contributions as follows:

- Development of the first complete OMT solver for NRA: We present OCAC and CDCL(OCAC), sound and complete algorithms for solving OMT(NRA) formulas, along with proofs of correctness and termination.
- Investigation of Variants: We explore two additional solving algorithm variants, using CAD and a first-order formulation approach.
- Evaluation: We integrated this algorithm into CVC5 (Barbosa et al. 2022) and conducted empirical evaluations demonstrating that our algorithm successfully solved numerous instances beyond the capability of the leading OMT solver, OptiMathSAT.

Preliminary

In this section, we describe the OMT(NRA) formulation and the mathematical foundations of developing a complete solving algorithm for SMT(NRA), i.e., cell and projection operators. Following this, we briefly discuss the CAD and CAC algorithms. Since maximization and minimization are interconvertible, we focus solely on minimization in the subsequent content. One can refer to (Kremer 2020) for a more comprehensive set of definitions of CAD.

Nonlinear Real Arithmetic

The set of natural numbers, the set of rational numbers, and the set of real numbers are denoted $\mathbb{N}$, $\mathbb{Q}$, and $\mathbb{R}$, respectively. The syntax of a general SMT formula over nonlinear real arithmetic (NRA) is as follows:

$$p := x \mid c \mid p + p \mid p \cdot p \qquad \text{(polynomial)}$$
$$\phi := b \mid p \geq 0 \mid p = 0 \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \qquad \text{(formula)}$$

In this context, x represents a variable for real numbers, c denotes a constant, and b stands for a Boolean variable. A polynomial atom can be expressed as $p \odot 0$, where p is a polynomial, and $\odot$ belongs to the set $\{>, \geq, =, <, \leq\}$. Polynomial atoms also function as Boolean variables. This notation is derived from the aforementioned syntax. For example, the expression $p < 0$ can be rewritten as $\neg(p \geq 0)$. ϕ is a general SMT(NRA) formula. An NRA-interpretation $\mathcal{I}$ provides an assignment to Boolean and real variables of ϕ. If ϕ is satisfiable, then $\phi^{\mathcal{I}} = \top$. We divide it into two parts: $\mathcal{I}_{\mathcal{B}}$ and $\mathcal{I}_{\mathcal{R}}$, which means the interpretation of Boolean variables and real variables. In the CDCL(T) framework (Ganzinger et al. 2004; Kroening and Strichman 2016; Barrett et al. 2021), a *branch* is a sequence of decisions (assignments) and implications, where $\psi = \phi^{\mathcal{I}_{\mathcal{B}}}$ represents the remaining formula after partial assignment of Boolean variables. The formula ψ is a conjunction of polynomial atoms. Then the *branch* ψ is then extended by invoking the NRA-solver to handle the interpretation of real variables $\mathcal{I}_{\mathcal{R}}$. The set of polynomials in formula ψ is denoted by $\mathcal{P}(\psi)$.

CAD Algorithm

The cell defines a sign-invariant region where the sign of every polynomial in the set remains unchanged. The expression $p(s)$ signifies the value obtained by substituting s into the polynomial p. The *sign* function displays the sign of the number derived from a polynomial with assignment s.

Definition 1 (Cell). *Given a set of polynomials $P = \{p_1, \cdots, p_m\} \subseteq \mathbb{Q}[x_1, \cdots, x_n]$ and $s \in \mathbb{R}^n$, a cell $\mathcal{C}(P, s)$ is a non-empty connected subset of $\mathbb{R}^n$ that is sign-invariant for P and contains s. Specifically, for all $i \in \{1, \cdots, m\}$,*

$$\forall s' \in \mathcal{C}(P, s), sign(p_i(s)) = sign(p_i(s')).$$

The projection operator can construct a polynomial set that defines the boundary of the cell. The extension expression $(s, s^i) = ((s^1, \cdots, s^{i-1}), s^i) = (s^1, \cdots, s^{i-1}, s^i)$ represents the extension of a sample point from $\mathbb{R}^{i-1}$ to $\mathbb{R}^i$.

We extend $\mathbb{R}$ to $\overline{\mathbb{R}}$, where $\overline{\mathbb{R}} := \mathbb{R} \cup \{\infty\} \cup \{\epsilon\}$ and $\epsilon \to 0$. A strict partial order $<_{\overline{\mathbb{R}} \times \overline{\mathbb{R}}}$ can be defined, with an obvious interpretation. For example, $-\infty$ is smaller than any other number, and $c - \epsilon < c, \epsilon \to 0$. For simplicity, we use $<$ to denote this relationship in the following context.

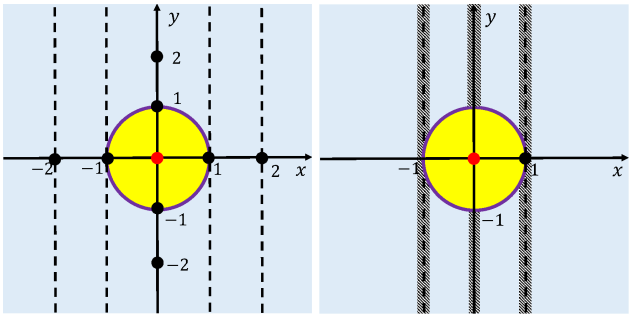


Figure 1: 1a and 1b are the exhibitive results of Example 2 and 3, respectively.

We define $\mathcal{C}(P,s)_{x_i}$ as an interval $\{\mathcal{L},\mathcal{U}\}$ over x_i, where $\mathcal{L},\mathcal{U} \in \overline{\mathbb{R}}$ are the lower bound and the upper bound, i.e.,

$$\forall r \in \mathbb{R}.(\mathcal{L} < r < \mathcal{U} \rightarrow \overline{s} \in \mathcal{C}(P,s)),$$

where $\overline{s} := (s^1, \cdots, s^{i-1}, r, s^{i+1}, \cdots, s^n)$, indicating that s_i is replaced by r.

A CAD (Collins 1975; Arnon, Collins, and McCallum 1984) is a decomposition algorithm for a set of polynomials in $\mathbb{R}^n, n \in \mathbb{N}$ space resulting in a finite number of *cells*. It encompasses two phases: *projection* and *lift*, exemplified in Example 2. It begins by reducing a polynomial set $P \subseteq \mathbb{Q}[x_1, \cdots, x_n]$ through projection operator, sequentially eliminating variables $x_n, x_{n-1}, \cdots, x_2$ to yield sets $P^{(1)}, \cdots, P^{(n-1)}$, each with progressively fewer variables. During the lift phase, the root isolation algorithm segments $\mathbb{R}$ into cells using the roots of the univariate polynomial set $P^{(n-1)}$. For instance, if there are k roots, CAD produces $2k+1$ samples for x_1, chosen from the roots and the intervening intervals. It plugs each sample for $(x_1, \cdots, x_i)$ into the polynomial set $P^{(n-i-1)}$, transforming it into different univariate polynomial sets. Then it isolates the roots and constructs new samples for $(x_1, \cdots, x_i, x_{i+1})$. Repeated $n-1$ times, CAD yields sign-invariant cells marked by n-dimensional sample points. The projection operator used in CAD is defined as follows:.

Definition 2 (CAD Projection Operator)**.** *Given a set of polynomials $P = \{p_1, \cdots, p_m\} \subseteq \mathbb{Q}[x_1, \cdots, x_i]$, the projection operator $proj_{dec}$ is a function that maps P and x_i to a polynomial set $P' \subseteq \mathbb{Q}[x_1, \cdots, x_{i-1}]$ such that for any $s \in \mathbb{R}^{i-1}$, there exists $\{s'_1, \cdots, s'_l\} \subseteq \mathbb{R}$ such that*

$$\mathcal{C}(P',s) \times \mathbb{R} = \bigcup_{j=1}^{l} \mathcal{C}(P,(s,s'_j)).$$

We define $proj^i_{dec}(P)$ as the i-th projection, that is, $proj^i_{dec}(P) = P^{(n-i)}$.

Example 2 (CAD)**.** *Consider the set of polynomials $\{x^2 + y^2 - 1\}$ from Example 1 and the variable order is $x \prec y$. CAD projects y first and results in a polynomial set $\{x^2 - 1\}$. The polynomial set has two roots by root isolation, that is, $x = -1$ and $x = 1$. Then $\mathbb{R}$ will be divided into five segments: $\{x < -1, x = -1, -1 < x < 1, x = 1, x > 1\}$. We sample $\{x = -2, x = -1, x = 0, x = 1, x = 2\}$, which results in five sets of polynomials by* lift*: $\{y^2 + 3\}$, $\{y^2\}$, $\{y^2 - 1\}$, $\{y^2\}$, and $\{y^2 + 3\}$. Let us take the third set of polynomials as an example, which has two roots, i.e., $\{-1, 1\}$. Then $\mathbb{R}$ will be divided into five segmentations: $\{y < -1, y = -1, -1 < y < 1, y = 1, y > 1\}$. Sample points in the segmentations can represent sign-invariant regions. As shown in Figure 1a, the red point $(0,0)$ can represent the sign-invariant region of $\{x^2 + y^2 - 1 < 0\}$.*

CAC Algorithm

The Cylindrical Algebraic Covering Algorithm (Ábrahám et al. 2021; Bär et al. 2023) is a variant of CAD, adjusted to the context of SMT. This adaptation reduces the number of projection polynomials needed within a specific projection. If CAC determines that x_{i+1} cannot be assigned due to the union of unsatisfiable intervals encompassing $\mathbb{R}$, it backtracks to the previous assignment of x_i after only collecting the conflicting polynomial set P. CAC generates unsatisfiable intervals for x_i by analyzing P and resamples for x_i. This iteration continues until it ends with $\top$ or $\bot$. The projection operator used in CAC is defined as follows:

Definition 3 (Covering Projection Operator (Bär et al. 2023))**.** *Given m polynomial sets $P^1, \cdots, P^m$, where each $P^i \subseteq \mathbb{Q}[x_1, \cdots, x_n], i \in \{1, \cdots, m\}$, a point $s \in \mathbb{R}^{n-1}$, a set of m real values $S = \{s'_1, \cdots, s'_m\} \subseteq \mathbb{R}$, and m cells such that*

$$\{s\} \times \mathbb{R} \subseteq \bigcup_{j=1}^{m} \mathcal{C}(P^j, (s, s'_j)).$$

The covering projection operator $proj_{cov}$ is a function that maps $P^1, \cdots, P^m, s, S$ to a polynomial set P' such that

$$\mathcal{C}(P',s) \times \mathbb{R} \subseteq \bigcup_{j=1}^{m} \mathcal{C}(P^j, (s, s'_j)).$$

Example 3 (CAC)**.** *Consider a new polynomial atom from Example 1: $x^2 + y^2 < 1$. Assume that CAC first assigns $x = 1$, resulting in $y^2 < 0$, equivalent to $\bot$. Any value for y will result in a conflict. Then CAC returns with the characterizing polynomial set $\{x^2 - 1\}$ from $\{x^2 + y^2 - 1\}$. CAC identifies an unsatisfiable region $\{-1, 1\} \times \mathbb{R}$ due to the same conflict reason as $x = 1$. Then excludes this interval and then samples for x from $\mathbb{R} - \{-1, 1\}$. Assume CAC assigns $x = 0$, and then it will result in a polynomial set $\{y^2 < 1\}$. The unsatisfiable interval for y is $(-\infty, -1]$ and $[1, +\infty)$. Figure 1b depicts the slashed parts as the eliminated unsatisfiable regions. Sampling $y = 0$ will result in assignment $(0,0)$, which satisfies the set of polynomial atoms.*

GOMT over Nonlinear Real Arithmetic

Definition 4 (GOMT problem (Tsiskaridze, Barrett, and Tinelli 2024))**.** *A General Optimization Modulo Theories problem is a tuple $\mathcal{GO} := (t, \prec, \phi)$, where,*

- *t, a Σ-term of some sort σ, is an objective term to optimize;*

- $\prec$ *is a strict partial order definable in* $\mathcal{T}$*, whose defining formula has two free variables, each of sort* σ*, and*
- ϕ *is a* Σ*-formula.*

Example 1 is a simple example of $\mathcal{GO}(x+y, <, x^2+y^2 = 1)$. For any GOMT problem $\mathcal{GO}$ and $\mathcal{T}$-interpretations $\mathcal{I}$ and $\mathcal{I}'$, we say that:

- $\mathcal{I}$ is $\mathcal{GO}$*-consistent* if $\mathcal{I} \models \phi$;
- $\mathcal{I}$ $\mathcal{GO}$*-dominates* $\mathcal{I}'$, denoted by $\mathcal{I} <_{\mathcal{GO}} \mathcal{I}'$, if $\mathcal{I}$ and $\mathcal{I}'$ are $\mathcal{GO}$-consistent and $t^{\mathcal{I}} \prec t^{\mathcal{I}'}$;
- $\mathcal{I}$ is a $\mathcal{GO}$-solution if $\mathcal{I}$ is $\mathcal{GO}$-consistent and no $\mathcal{T}$-interpretation $\mathcal{GO}$-dominates $\mathcal{I}$.

The GOMT is a theory-agnostic problem. Users can define different kinds of strict partial order or use built-in ones.

Sometimes, finding the $\mathcal{GO}$-solution will be a *Zeno-style* infinite chain of increasingly better solutions (Tsiskaridze, Barrett, and Tinelli 2024; Sebastiani and Tomasi 2012; Sebastiani and Trentin 2015). If we incrementally invoke an SMT solver and prune the obtained objective value c via $t < c$, there are two types of *Zeno-style* infinite chain:

- Infinitesimal case: The optimum is $c + \epsilon$, where c is a constant and ϵ is infinitesimal.
- Unbounded case: The optimum does not exist; in other words, it is $-\infty$.

In order to cover these two special cases, we define the GOMT problem in NRA.

Definition 5 (GOMT over NRA)**.** *A* $\mathcal{GO}_{\mathcal{NRA}}$ *is a tuple* $\mathcal{GO}_{\mathcal{NRA}} := (t, < \phi)$*, where,*

- t *is the polynomial objective term to optimize;*
- $<$ *is a strict partial order definable in NRA, whose formula defines over* $\overline{\mathbb{R}} \times \overline{\mathbb{R}}$*,*
- ϕ *is an SMT(NRA) formula.*

Note that it may not have an exact interpretation $\mathcal{I}$ due to infinitesimal and unbounded cases.

The Optimization Procedure

This section presents an overview of the optimization procedure and then describes the main algorithm, Optimization Cylindrical Algebraic Covering (OCAC). We also show illustrative cases that demonstrate the execution of the algorithm, CDCL(OCAC).

Overview

This procedure relies on an extension of the CDCL(T) framework. The CDCL(T) framework introduces a *branch* ψ that is a conjunction of polynomial atoms. It serves as input for OCAC. OCAC returns UNSAT or SAT with optimum and a cutting lemma. The cutting lemma prunes the searched space for completeness. The algorithm transforms the objective term into a variable x_t. We can define the *OMT branch formula* as $\psi \wedge t = x_t$. Then OCAC establishes a variable order that grants priority to x_t. Following the CAC procedure, the values are assigned to the variables in the predetermined order, leading to a full assignment or a conflict. If OCAC achieves a full assignment, it constructs a cell that encapsulates the corresponding point defined by the full assignment. By analyzing the boundary of the cell, the OCAC obtains a temporary optimum for x_t in the branch. If OCAC encounters a conflict, it resolves the conflict like CAC, that is, finding and pruning a cover of samples within the conflict. Each case narrows the unexplored space for x_t. Following several iterations of these processes, OCAC thoroughly investigates $\mathbb{R}$, the domain of x_t. Ultimately, OCAC determines the optimum value of the *branch* or determines that the *branch* is unsatisfiable. This process repeats until the CDCL(T) framework exhausts all possible branches. Finally, the CDCL(T) framework outputs the optimum among all satisfiable branches.

Algorithm 1: OCAC

Input: $\psi \wedge t = x_t$: The *OMT branch formula*, with n variables.
Output: g, v, l: A flag that $\psi \wedge t = x_t$ is satisfiable; The optimum value; The cutting lemma.

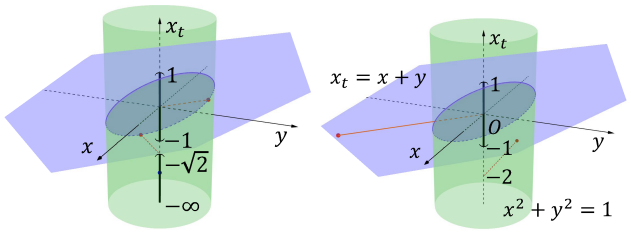
```
1:  𝕀 := ∅, g := ⊥, v := None
2:  while ⋃_{I∈𝕀} I ≠ ℝ do
3:      s_t := Sample_Objective_Value(𝕀)
4:      (T, O) := Solve_Internal(ψ ∧ t = x_t ∧ x_t = s_t)
5:      if T = ⊤ then
6:          g := ⊤, v := Analyze_Cell(O)
7:          O := O ∪ [s_t, +∞)
8:      end if
9:      𝕀 := 𝕀 ∪ {O}
10: end while
11: return (g, v, Lemma(v))
```

Optimization Cylindrical Algebraic Covering

OCAC Algorithm 1 iteratively samples the values for the objective s_t and exhausts the search space for x_t. It accepts an *OMT branch formula*. Then, it samples values outside the infeasible intervals for x_t, focusing on an optimizing direction. It prioritizes the smallest values, starting from the leftmost unsearched interval (line 3). It solves the problem $\psi \wedge t = x_t \wedge x_t = s_t$ and returns a satisfiability flag T along with a characterization interval O of x_t (line 4). Upon finding a full assignment (line 5), it identifies the optimum at the left boundary of the characterization interval O (line 6). The search space prunes the cell $\mathcal{C}(P(\psi), s)_{x_t}$ and $[s_t, +\infty)$ (line 7). After exhausting all search spaces for x_t, OCAC generates a cutting lemma l for CDCL(T) (line 11). For example, if the optimum of the current *branch* is $v = c$, the lemma will be $x_t < c$. It forces the algorithm to find better optimum values when processing other branches.

Solve_Internal It checks the satisfiability of the formula $\psi(\boldsymbol{x}) \wedge t(\boldsymbol{x}) = x_t \wedge x_t = s_t$ by iteratively assigning values to the current variable until all variables are considered (Kremer and Nalbach 2022). Then it constructs a characterization interval O for x_t using projection operators and root isolation algorithm. The intermediate solving process outlines the search space of x_t in an *OMT branch formula*.

Analyze_Cell It examines the characterization interval to locate the optimum. This interval, I, is derived from a cell



(a) (b)

Figure 2: 2a and 2b indicate the process of CDCL(OCAC) with x_t, y, x for Example 4 and Example 5, respectively. The dark parts in the axis-x_t are the pruned intervals via satisfiability or unsatisfiability characterization intervals.

and can manifest itself as an open interval or a point interval. The interval that includes s_t satisfies $\psi \wedge t = x_t \wedge x_t = s_t$, which means that any value within I can lead to the full assignment for the same reasons. The correctness is guaranteed by Theorem 2. OCAC can determine the optimum for the current branch as stated in Corollary 1.

We can define a specific GOMT problem to formalize the process and results of OCAC.

Definition 6 (GOMT in OCAC)**.** *A $\mathcal{GO}_{\mathcal{OCAC}}$ is a tuple $\mathcal{GO}_{\mathcal{OCAC}} := (x_t, \prec_{Cell}, \psi)$, where,*

- *x_t, a variable with $x_t = t$, is the objective variable where t is the polynomial objective term to optimize;*
- *$\prec_{Cell}$ is a strict partial order definable in NRA, whose formula defines over $\mathbb{R} \times \mathbb{R}$, and $x_t^{\mathcal{I}} \prec_{Cell} x_t^{\mathcal{I}'}$ is equivalent to:*

$$\mathcal{L}(\mathcal{C}(\mathcal{P}(\psi), \mathcal{I}_{\mathcal{R}})_{x_t}) < \mathcal{L}(\mathcal{C}(\mathcal{P}(\psi), \mathcal{I}'_{\mathcal{R}})_{x_t}),$$

- *ψ is a conjunction of polynomial atoms.*

Given an $\mathcal{GO}_{\mathcal{OCAC}}$-solution $\mathcal{I}$, we can determine the optimum v_t, expressed as $v_t := \mathcal{L}(\mathcal{C}(\mathcal{P}(\psi), \mathcal{I}_{\mathcal{R}})_{x_t})$. After the CDCL(T) framework traverses all *branches* ψ of the SMT formula ϕ, the minimum value of v_t is identified as the optimum of t in $\mathcal{GO}(t, <, \phi)$.

Examples

This section concentrates on how CDCL(OCAC) executes specific instances with variable order (x_t, y, x). Note that the order is projection variable order; for assigning, the order is reversed, from x_t to x. The instances are simple, with only one branch, but can intuitively show the process.

Example 4. *Continuing with Example 1, the algorithm finds two full assignments and approaches the optimum on the second attempt, as shown in Figure 2a. Initially, it first samples $x_t \mapsto 0$ and finds a full assignment $(x, y, x_t) = (-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0)$ satisfying the constraints. Then it searches for the satisfiable interval (cell) for $x_t \mapsto 0$. The values in the satisfiable interval of x_t can also be satisfiable for the same reason as in $x_t \mapsto 0$, yielding $(-1, 1)$. It also excludes $[0, +\infty)$ because the minimum cannot be there, finally discarding $(-1, +\infty)$. Next, it samples $x_t \mapsto -2$ and fails to complete the assignment. Then, based on CAC, it identifies $(-\infty, -\sqrt{2})$ as the reason why x_t cannot be extended to a full assignment, leading to the pruning of $(-\infty, -\sqrt{2})$. The remaining interval potentially containing the optimum is $[-\sqrt{2}, -1]$. The algorithm first samples the minimal, that is, $x_t \mapsto -\sqrt{2}$, achieves a full assignment, and excludes $[-\sqrt{2}, +\infty)$. Consequently, no further value can be assigned to x_t, prompting a return to the outer CDCL framework with the cutting lemma $x_t < -\sqrt{2} \wedge \neg(x^2 + y^2 = 1)$. The outer CDCL framework ends with an optimum of $-\sqrt{2}$.*

Example 5. *We slightly change the OMT formula in Example 4 to $\mathcal{GO}(z_{x+y}, \prec_{Cell}, x^2 + y^2 \geq 1)$. The algorithm finds two full assignments and shows that the formula does not have an optimum, that is, $x + y \mapsto -\infty$, as shown in Figure 2b. Similarly to Example 4, it initially samples $x_t \mapsto 0$ and prunes $(-1, +\infty)$. Then it samples $x_t \mapsto -2$ and obtains the full assignment. Upon analyzing the cell $\mathcal{C}(\{x^2+y^2, x+y-x_t\}, (-2, 0, -2))$, it finds $x+y \mapsto -\infty$. The cutting lemma will be $\bot$, which directly terminates the outer CDCL framework with $x + y \mapsto -\infty$.*

The Termination and Correctness

We begin by introducing the correctness of the CAC algorithm, as guaranteed by Theorem 1 (Ábrahám et al. 2021; Bär et al. 2023).

Theorem 1. *Let ψ be a conjunction of polynomial atoms with $x_1, \cdots, x_n, S = \{s'_1, \cdots, s'_m\} \subseteq \mathbb{R}, P^1, \cdots, P^m \subseteq \mathbb{Q}[x_1, \cdots, x_i]$ and $s \in \mathbb{R}^{i-1}$ for $1 < i \leq n$. If $\{s\} \times \mathbb{R} \subseteq \bigcup_{j=1}^{m} \mathcal{C}(P^j, (s, s'_j))$ and for $1 \leq j \leq m, \mathcal{C}(P^j, (s, s'_j))$ is unsatisfiable for ψ, then $\mathcal{C}(proj_{cov}(P^1, \cdots, P^m, s, S), s)$ is unsatisfiable for ψ.*

In practice, CAC excludes cells that cannot extend to a full assignment, guaranteeing finite-step termination by either exhausting the entire space or finding a full assignment.

Theorem 2. *Given an OMT branch formula $\psi \wedge t = x_t$, P denotes the set of polynomials in $\psi \wedge t = x_t$. If $\psi \wedge t = x_t$ is satisfiable, that is, there exists a complete assignment $s = (s_t, s_1, \cdots, s_n)$ that satisfies $\psi \wedge t = x_t$, then $\forall \gamma_o \in \mathcal{C}(proj^n_{dec}(P), s)_{x_t}, \psi \wedge t = \gamma_o$ is satisfiable.*

In the OCAC algorithm, once a full assignment is found, a satisfiable interval for x_t can be constructed based on Theorem 2. Within this interval, every value leads to a full assignment, indicating that the optimum lies on the boundary.

Theorem 3 (Termination of OCAC)**.** *Given an OMT branch formula $\psi \wedge t = x_t$, OCAC terminates.*

Proof sketch. For the case of x_t, we can deduce the following: If the current assignment is satisfiable, the corresponding space can be pruned according to Theorem 2. On the other hand, if it cannot extend to a full assignment, the space can be eliminated according to the correctness of the *covering projection operator*. The process is finite, which ensures that $\mathbb{R}$ is systematically excluded, leading to termination.

Theorem 4 (Correctness of OCAC)**.** *Given an OMT branch formula $\psi \wedge t = x_t$, if $\psi \wedge t = x_t$ is unsatisfiable, OCAC returns UNSAT; otherwise, OCAC can find the optimum.*

Proof sketch. Since the unsatisfiable intervals of Theorem 1 and the satisfiable intervals of Theorem 2 are disjoint, the optimum can be identified from the lower bound of the leftmost satisfiable interval.

Building on Theorem 2 and Theorem 4, we can formulate a corollary to decide the optimum for x_t.

Corollary 1. *Given a conjunction of polynomial atoms ψ, let I_o of x_t represent the leftmost satisfiable interval for minimization, which can be characterized by three cases:*

- $I_o = (-\infty, u)$ *implies* $\min(x_t) = -\infty$*;*
- $I_o = (l, u)$ *implies* $\min(x_t) = l + \epsilon$*;*
- $I_o = [l, l]$ *implies* $\min(x_t) = l$*.*

Theorem 5 (Termination and Correctness)**.** *For a given instance $\mathcal{GO}_{\mathcal{NRA}}(t, <, \phi)$, the CDCL(OCAC) algorithm is guaranteed to terminate and produce correct results.*

Proving the theorem is a straightforward task. The outer CDCL(T) framework systematically explores every branch of ϕ, with OCAC terminated, and finds the optimum or proves the unsatisfiability on each call. The cutting lemma guarantees an improved optimum.

The complexity of CDCL(OCAC) stems from the need to explore all branches along with the OCAC algorithm. In the worst-case scenario, the OCAC may have to traverse all cells to find the optimum within a single branch, resulting in doubly exponential complexity relative to the number of variables (Collins 1975; Arnon, Collins, and McCallum 1984; Jia et al. 2023a). However, in practice, the covering projection operator can more effectively eliminate larger unsatisfiable intervals for x_t than the cell projection operator, helping to identify cells that contain the optimum.

Baselines

This section explores several candidate baselines for solving OMT(NRA) problems, featuring variants derived from CAD and first-order formulation. We briefly review the techniques used in OptiMathSAT. In addition, the reader can refer to Chapter 7.6 of Kremer (2020), which details a variant developed from MCSAT (Jovanovic and de Moura 2012).

CAD-Based Variant

This variant employs the CAD algorithm for optimization (Wolfram 2024). It constructs $proj^n_{dec}(\mathcal{P}(\psi))$, where ψ denotes a *branch* of the SMT(NRA) formula. Using the root isolation algorithm, the algorithm generates a list of candidate intervals for x_t. The process starts by selecting a sample point from the leftmost interval and lifting the partial assignment until it reaches a satisfiable assignment. If it fails, the process moves progressively toward the rightmost interval of x_t, continuing until a full assignment is achieved. If the search exhausts without success, the branch is unsatisfiable. The left boundary of the first interval where x_t achieves a full assignment is identified as the optimum. Although this variant works with the CDCL(T) framework, it is resource-intensive, particularly due to the demanding *lift* technique (Strzebonski 2006; Iwane et al. 2013).

First-Order Formulation

A previous encoding of first-order logic (FOL) for OMT problems can be found in (Kong, Solar-Lezama, and Gao 2018; Yao et al. 2021):

$$\psi(\boldsymbol{x}) \wedge \forall \boldsymbol{y}.(\psi(\boldsymbol{y}) \rightarrow t(\boldsymbol{x}) \leq t(\boldsymbol{y})).$$

Essentially, this formulation is appropriate when a model exists. If *Zeno-style* infinite chain exists, the encoding will report $\perp$. For example, if $t \mapsto -\infty$, the algorithm can identify a valid model for any sufficiently large negative value of t. Applying the constraint leads to an unsatisfiable result which does not differentiate between these two cases and the unsatisfiable OMT problem.

We can formulate these cases using the definitions. If the optimum v is a constant, we have,

$$\textit{fixity} \rightarrow (\psi(\boldsymbol{x}) \wedge t(\boldsymbol{x}) = v \wedge \forall \boldsymbol{y}.(\psi(\boldsymbol{y}) \rightarrow t(\boldsymbol{y}) \geq v)).$$

If it is infinitesimal case, i.e., $v = c + \epsilon, \epsilon \rightarrow 0$, we have,

$$\begin{aligned}\textit{infinitesimal} \rightarrow (&\forall \boldsymbol{y}.(\psi(\boldsymbol{y}) \rightarrow t(\boldsymbol{y}) > v) \wedge \\ &\forall \epsilon. \exists \boldsymbol{z}(\epsilon > 0 \rightarrow \psi(\boldsymbol{z}) \wedge t(\boldsymbol{z}) < v + \epsilon)).\end{aligned}$$

If it is unbounded case, i.e., $v \rightarrow -\infty$, we have,

$$\textit{unboundedness} \rightarrow (\forall M. \exists \boldsymbol{y}.(\psi(\boldsymbol{y}) \wedge t(\boldsymbol{y}) < M)).$$

An OMT(NRA) problem can be classified into one of three categories, i.e.,

$$\textit{fixity} \vee \textit{infinitesimal} \vee \textit{unboundedness}.$$

This encoding can also verify the results by integrating the derived optimum into the first-order formulation.

OptiMathSAT

OptiMathSAT is the only solver that officially supports solving OMT(NRA) formulas (Sebastiani and Trentin 2020), but it is not open source. OptiMathSAT (Bigarella et al. 2021), based on MathSAT (Cimatti et al. 2013), employs incremental linearization, converting nonlinear atoms into linear atoms with uninterpreted functions and determining the optimum value. The value then serves as the lower bound of the optimum for the OMT(NRA) formula. The algorithm refines the optimum through multiple iterations of SMT solving. It can partially solve instances of the infinitesimal case ($c + \epsilon, \epsilon \rightarrow 0$), or the unbounded case ($-\infty$), when the linear optimum coincides with or is very close to the nonlinear optimum. It has different strategies to approach the optimum: ***Bin*** (binary search) and ***Lin*** (linear search). The binary search computes the pivot $p = \frac{l+u}{2}$ from the lower bound (l) and the upper bound (u) and adds the constraint $x_t < p$. Linear search adds an incremental lemma $x_t < v_o$, to prune the satisfiable assignment found v_o to x_t.

Empirical Evaluation

We implemented OCAC in CVC5 1.0.4, utilizing the Lazard projection operator (Lazard 1994; Kremer and Brandt 2021) provided by CoCoALib (Abbott and Bigatti 2010) and LibPoly (Jovanovic and Dutertre 2017).

The experimental instances are generated from satisfiable SMT(QF_NRA) benchmarks. For variety, we consider five

	#(RAN)	#(RAN $+\epsilon$)	#($\mathbb{Q}$)	#($\mathbb{Q}+\epsilon$)	#(∞)	#SAT	#UNSAT
CDCL(CAD)	246	551	802	2990	1129	5718	4568
FOL	304	610	**1101**	3545	1165	6725	4392
OptiMathSAT(Bin)	0	0	943	1870	353	3166	**5040**
OptiMathSAT(Lin)	0	0	928	1819	336	3083	**5040**
CDCL(OCAC) (Ours)	**369**	**981**	1084	**4248**	**1250**	**7932**	5019

Table 1: Performance on the number of solved instances, including 10000 satisfiable and 5532 unsatisfiable ones.

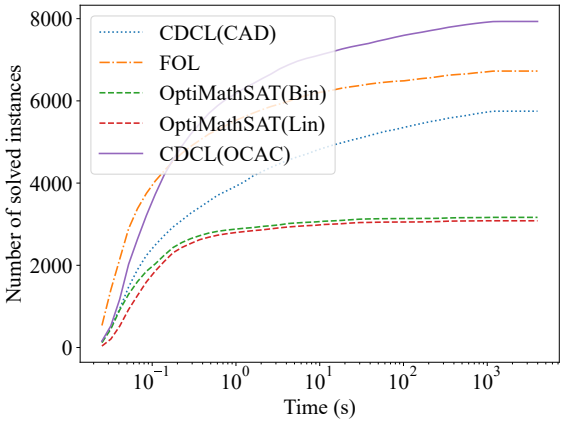


Figure 3: Performance on satisfiable instances over time.

types of minimization objective functions: x, $x+y$, x^2+y^2, xy, and $xy+z$, where x, y, and z are randomly selected from the set of declared variables of the original SMT instances. We randomly select 10000 instances. We also gather all unsatisfiable instances by adding a random declared variable as the objective, a total of 5532 instances.

All experiments are done with an Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz and 256G RAM within Ubuntu 20.04.6 LTS. The timeout for each solver that executes each instance is 1200 seconds. We use Z3 4.13.0 (de Moura and Bjørner 2008) for the first-order formulation and to verify the optimum correctness with a timeout of 1200 s. We provide additional experiments and results in the supplementary material, including the situation of yicesQS (Bonacina, Graham-Lengrand, and Vauthier 2023), CVC5 (Barbosa et al. 2022), and dReal (Gao, Kong, and Clarke 2013).

Results

We aim to compare the performance of CDCL(OCAC) with baseline methods. Table 1 displays the performance of different solvers in the number of solved instances. The notations #(RAN), #(RAN + ϵ), #($\mathbb{Q}$), #($\mathbb{Q}+\epsilon$), #(∞), represent the number of solved instances by result type, with RAN indicating a real algebraic number optimum, and the others following similarly. #SAT and #UNSAT are total numbers of solved instances. CDCL(OCAC) outperforms other methods in finding optimums in most categories of satisfiable instances. It solves 150.5% and 157.3% more instances compared to OptiMathSAT(Bin/Lin), respectively. CDCL(CAD) and FOL perform well in satisfiable instances, with FOL excelling particularly in #($\mathbb{Q}$) case. OptiMathSAT is the best at solving unsatisfiable instances, while CDCL(OCAC) also demonstrates competitive performance. OptiMathSAT cannot solve any instances of #(RAN) or #(RAN + ϵ), but can provide an approximate optimum within a specified tolerance for 42 instances categorized as #($\mathbb{Q}$) or #($\mathbb{Q}+\epsilon$). Figure 3 displays the performance of different solvers over time. Initially, FOL performs faster within approximately the first 0.2 seconds, but CDCL(OCAC) outperforms it afterwards. Compared to other baseline solvers, CDCL(OCAC) consistently shows higher efficiency throughout the entire duration. Overall, CDCL(OCAC) generally demonstrates better solving ability and speed compared to other baselines.

Limitations

One challenge is the lack of standardized benchmarks for OMT. Consequently, we depend on instances randomly generated from SMT(QF_NRA) instances with a uniform distribution. We have successfully solved instances with up to 29 variables, covering 92.01% of the original satisfiable SMT(QF_NRA) instances. and a more scalable algorithm is required to solve larger instances.

Conclusion and Future Work

In this paper, we propose a sound and complete OMT(NRA) algorithm. It is the first complete OMT solver for NRA. We prove the correctness and termination of the algorithm and investigate some variants of the OMT(NRA) algorithm, using CAD and first-order formulation. The empirical results show that our algorithm can find all types of optimums. OCAC complements incomplete algorithms in certain cases. For future work, it is desirable to use and evaluate the solver in more applications. In addition, we can improve the efficiency by incorporating incomplete algorithms and expand the versatility by integrating it into the GOMT framework.

Acknowledgments

This research has been supported by the National Natural Science Foundation of China (NSFC) under grants No.62132020 and No.61972384. The authors thank Clark Barrett and Nestan Tsiskaridze for introducing the concept of Generalized Optimization Modulo Theories and addressing our questions about it. We also sincerely thank the anonymous reviewers and editors for their valuable feedback and suggestions.

References

Abbott, J.; and Bigatti, A. M. 2010. CoCoALib: A C++ Library for Computations in Commutative Algebra... and Beyond. In *ICMS*, volume 6327 of *LNCS*, 73–76.

Ábrahám, E.; Davenport, J. H.; England, M.; and Kremer, G. 2021. Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. *JLAMP*, 119: 100633.

Arnon, D. S.; Collins, G. E.; and McCallum, S. 1984. Cylindrical Algebraic Decomposition I: The Basic Algorithm. *SICOMP*, 13(4): 865–877.

Bär, P.; Nalbach, J.; Ábrahám, E.; and Brown, C. W. 2023. Exploiting Strict Constraints in the Cylindrical Algebraic Covering. In *SMT*, volume 3429 of *CEUR Workshop Proceedings*, 33–45.

Barbosa, H.; Barrett, C.; Brain, M.; Kremer, G.; Lachnitt, H.; Mann, M.; Mohamed, A.; Mohamed, M.; Niemetz, A.; Nötzli, A.; et al. 2022. cvc5: A versatile and industrial-strength SMT solver. In *TACAS*, 415–442.

Barrett, C. W.; Dill, D. L.; and Levitt, J. R. 1998. A Decision Procedure for Bit-Vector Arithmetic. In *DAC*, 522–527.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2021. Satisfiability Modulo Theories. In *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 1267–1329. IOS Press.

Bertolissi, C.; dos Santos, D. R.; and Ranise, S. 2018. Solving Multi-Objective Workflow Satisfiability Problems with Optimization Modulo Theories Techniques. In *SACMAT*, 117–128.

Bigarella, F.; Cimatti, A.; Griggio, A.; Irfan, A.; Jonás, M.; Roveri, M.; Sebastiani, R.; and Trentin, P. 2021. Optimization Modulo Non-linear Arithmetic via Incremental Linearization. In *FroCoS*, volume 12941 of *LNCS*, 213–231.

Bjørner, N. S.; Phan, A.; and Fleckenstein, L. 2015. νZ - An Optimizing SMT Solver. In *TACAS*, volume 9035 of *LNCS*, 194–199.

Bonacina, M. P.; Graham-Lengrand, S.; and Vauthier, C. 2023. QSMA: A New Algorithm for Quantified Satisfiability Modulo Theory and Assignment. In *CADE*, volume 14132 of *LNCS*, 78–95.

Cimatti, A.; Griggio, A.; Irfan, A.; Roveri, M.; and Sebastiani, R. 2018. Incremental Linearization for Satisfiability and Verification Modulo Nonlinear Arithmetic and Transcendental Functions. *TOCL*, 19(3): 19:1–19:52.

Cimatti, A.; Griggio, A.; Schaafsma, B. J.; and Sebastiani, R. 2013. The MathSAT5 SMT Solver. In *TACAS*, volume 7795 of *LNCS*, 93–107.

Collins, G. E. 1975. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata theory and formal languages*, 134–183. Springer.

de Moura, L. M.; and Bjørner, N. S. 2008. Z3: An Efficient SMT Solver. In *TACAS*, volume 4963 of *LNCS*, 337–340.

Dutertre, B.; and de Moura, L. M. 2006. A Fast Linear-Arithmetic Solver for DPLL(T). In *CAV*, volume 4144 of *LNCS*, 81–94.

Erata, F.; Piskac, R.; Mateu, V.; and Szefer, J. 2023. Towards Automated Detection of Single-Trace Side-Channel Vulnerabilities in Constant-Time Cryptographic Code. In *EuroS&P*, 687–706.

Ganzinger, H.; Hagen, G.; Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2004. DPLL(T): Fast Decision Procedures. In *CAV*, volume 3114 of *LNCS*, 175–188.

Gao, S.; Kong, S.; and Clarke, E. M. 2013. dReal: An SMT Solver for Nonlinear Theories over the Reals. In *CADE*, volume 7898 of *LNCS*, 208–214.

He, X.; Li, B.; Zhao, M.; and Cai, S. 2024. A Local Search Algorithm for MaxSMT(LIA). In *FM*, volume 14933 of *LNCS*, 55–72.

Henry, J.; Asavoae, M.; Monniaux, D.; and Maiza, C. 2014. How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics. In *LCTES*, 43–52.

Huang, P.; Wu, H.; Yang, Y.; Daukantas, I.; Wu, M.; Zhang, Y.; and Barrett, C. W. 2024. Towards Efficient Verification of Quantized Neural Networks. In *AAAI*, 21152–21160.

Huang, P.; Yang, Y.; Liu, M.; Jia, F.; Ma, F.; and Zhang, J. 2022. ϵ-weakened robustness of deep neural networks. In *ISSTA*, 126–138.

Iwane, H.; Yanami, H.; Anai, H.; and Yokoyama, K. 2013. An effective implementation of symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. *TOCS*, 479: 43–69.

Jia, F.; Dong, Y.; Liu, M.; Huang, P.; Ma, F.; and Zhang, J. 2023a. Suggesting Variable Order for Cylindrical Algebraic Decomposition via Reinforcement Learning. In *NeurIPS*.

Jia, F.; Han, R.; Huang, P.; Liu, M.; Ma, F.; and Zhang, J. 2023b. Improving Bit-Blasting for Nonlinear Integer Constraints. In *ISSTA*, 14–25.

Jiang, J.; Chen, L.; Wu, X.; and Wang, J. 2017. Block-Wise Abstract Interpretation by Combining Abstract Domains with SMT. In *VMCAI*, volume 10145 of *LNCS*, 310–329.

Jin, X.; Xia, C.; Guan, N.; and Zeng, P. 2021. Joint Algorithm of Message Fragmentation and No-Wait Scheduling for Time-Sensitive Networks. *IEEE CAA J. Autom. Sinica*, 8(2): 478–490.

Jovanovic, D.; and de Moura, L. M. 2012. Solving Nonlinear Arithmetic. In *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, 339–354.

Jovanovic, D.; and Dutertre, B. 2017. LibPoly: A Library for Reasoning about Polynomials. In *CAV*, volume 1889 of *CEUR Workshop Proceedings*, 28–39.

Karpenkov, E. G. 2017. *Finding inductive invariants using satisfiability modulo theories and convex optimization.* Ph.D. thesis, Grenoble Alpes University, France.

Karpenkov, E. G.; Friedberger, K.; and Beyer, D. 2016. JavaSMT: A Unified Interface for SMT Solvers in Java. In *VSTTE*, volume 9971 of *LNCS*, 139–148.

Kong, S.; Solar-Lezama, A.; and Gao, S. 2018. Delta-Decision Procedures for Exists-Forall Problems over the Reals. In *CAV*, volume 10982 of *LNCS*, 219–235.

Kremer, G. 2020. *Cylindrical algebraic decomposition for nonlinear arithmetic problems*. Ph.D. thesis, RWTH Aachen University, Germany.

Kremer, G.; and Brandt, J. 2021. Implementing arithmetic over algebraic numbers A tutorial for Lazard's lifting scheme in CAD. In *SYNASC*, 4–10.

Kremer, G.; and Nalbach, J. 2022. Cylindrical Algebraic Coverings for Quantifiers (short paper). In *IJCAR*, volume 3458 of *CEUR Workshop Proceedings*, 1–9.

Kroening, D.; and Strichman, O. 2016. *Decision Procedures - An Algorithmic Point of View, Second Edition*. Texts in Theoretical Computer Science. An EATCS Series. Springer. ISBN 978-3-662-50496-3.

Lazard, D. 1994. An improved projection for cylindrical algebraic decomposition. In *Algebraic geometry and its applications*, 467–476.

Leofante, F. 2023. OMTPlan: A Tool for Optimal Planning Modulo Theories. *JSAT*, 14(1): 17–23.

Leofante, F.; Ábrahám, E.; Niemueller, T.; Lakemeyer, G.; and Tacchella, A. 2019. Integrated Synthesis and Execution of Optimal Plans for Multi-Robot Systems in Logistics. *Inf. Syst. Frontiers*, 21(1): 87–107.

Li, Y.; Albarghouthi, A.; Kincaid, Z.; Gurfinkel, A.; and Chechik, M. 2014. Symbolic optimization with SMT solvers. In *POPL 2014*, 607–618.

Liang, T.; Reynolds, A.; Tsiskaridze, N.; Tinelli, C.; Barrett, C. W.; and Deters, M. 2016. An efficient SMT solver for string constraints. *Formal Methods Syst. Des.*, 48(3): 206–234.

Liu, T.; Tyszberowicz, S. S.; Beckert, B.; and Taghdiri, M. 2017. Computing Exact Loop Bounds for Bounded Program Verification. In *SETTA*, volume 10606 of *LNCS*, 147–163.

Ma, F.; Yan, J.; and Zhang, J. 2012. Solving Generalized Optimization Problems Subject to SMT Constraints. In *FAW-AAIM*, volume 7285 of *LNCS*, 247–258.

Marchetto, G.; Sisto, R.; Valenza, F.; Yusupov, J.; and Ksentini, A. 2021. A Formal Approach to Verify Connectivity and Optimize VNF Placement in Industrial Networks. *IEEE Trans. Ind. Informatics*, 17(2): 1515–1525.

Nadel, A.; and Ryvchin, V. 2016. Bit-Vector Optimization. In *TACAS*, volume 9636 of *LNCS*, 851–867.

Nan, M. S.; Bogdan, C.; Grecea, D.; and Mamara, N. L. 2017. Exact global optimization. *International Multidisciplinary Scientific GeoConference: SGEM*, 17: 303–309.

Nieuwenhuis, R.; and Oliveras, A. 2006. On SAT Modulo Theories and Optimization Problems. In *SAT*, volume 4121 of *LNCS*, 156–169.

Paoletti, N.; Jiang, Z.; Islam, M. A.; Abbas, H.; Mangharam, R.; Lin, S.; Gruber, Z.; and Smolka, S. A. 2019. Synthesizing stealthy reprogramming attacks on cardiac devices. In *ICCPS*, 13–22.

Ratschan, S. 2017. Simulation Based Computation of Certificates for Safety of Dynamical Systems. In *FORMATS*, volume 10419 of *LNCS*, 303–317.

Roselli, S. F.; Bengtsson, K.; and Åkesson, K. 2018. SMT Solvers for Job-Shop Scheduling Problems: Models Comparison and Performance Evaluation. In *CASE*, 547–552.

Sebastiani, R.; and Tomasi, S. 2012. Optimization in SMT with ($\mathbb{Q}$) Cost Functions. In *IJCAR*, 484–498.

Sebastiani, R.; and Trentin, P. 2015. Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions. In *TACAS*, volume 9035 of *LNCS*, 335–349.

Sebastiani, R.; and Trentin, P. 2020. OptiMathSAT: A Tool for Optimization Modulo Theories. *JAR*, 64(3): 423–460.

Sivaraman, A.; Farnadi, G.; Millstein, T. D.; and den Broeck, G. V. 2020. Counterexample-Guided Learning of Monotonic Neural Networks. In *NeurIPS*.

Strzebonski, A. W. 2006. Cylindrical Algebraic Decomposition using validated numerics. *J. Symb. Comput.*, 41(9): 1021–1038.

Teso, S.; Sebastiani, R.; and Passerini, A. 2017. Structured learning modulo theories. *AIJ*, 244: 166–187.

Trentin, P.; and Sebastiani, R. 2019. Optimization Modulo the Theory of Floating-Point Numbers. In *CADE*, volume 11716 of *LNCS*, 550–567.

Trentin, P.; and Sebastiani, R. 2021. Optimization Modulo the Theories of Signed Bit-Vectors and Floating-Point Numbers. *JAR*, 65(7): 1071–1096.

Tsiskaridze, N.; Barrett, C. W.; and Tinelli, C. 2024. Generalized Optimization Modulo Theories. In *IJCAR*, volume 14739 of *LNCS*, 458–479.

Wang, Q.; Chen, M.; Xue, B.; Zhan, N.; and Katoen, J. 2021. Synthesizing Invariant Barrier Certificates via Difference-of-Convex Programming. In *CAV*, volume 12759 of *LNCS*, 443–466.

Wolfram. 2024. Exact Global Optimization. https://reference.wolfram.com/language/tutorial/ConstrainedOptimizationExact.html.en. Section: Optimization by Cylindrical Algebraic Decomposition, Accessed: August 15, 2024.

Yan, R.; Cai, A.; Gao, H.; Ma, F.; and Yan, J. 2019. SMT-based Multi-objective Optimization for Scheduling of MP-SoC Applications. In *TASE*, 160–167.

Yao, P.; Shi, Q.; Huang, H.; and Zhang, C. 2021. Program analysis via efficient symbolic abstraction. *PACMPL*, 5(OOPSLA): 1–32.

Zhang, J. 2000. Specification Analysis and Test Data Generation by Solving Boolean Combinations of Numeric Constraints. In *APAQS*, 267–274.

Zhang, J.; Ma, F.; and Zhang, Z. 2012. Faulty Interaction Identification via Constraint Solving and Optimization. In *SAT*, volume 7317 of *LNCS*, 186–199.

Zhang, X.; Li, B.; and Cai, S. 2024. Deep Combination of CDCL(T) and Local Search for Satisfiability Modulo Non-Linear Integer Arithmetic Theory. In *ICSE*, 125:1–125:13.

Zhang, Z.; Yan, J.; Zhao, Y.; and Zhang, J. 2014. Generating combinatorial test suite using combinatorial optimization. *J. Syst. Softw.*, 98: 191–207.