Solving Non-Linear Optimization with the Cylindrical Algebraic Covering Method

Valentin Promies[1][0000−0002−3086−9976] and
Erika Ábrahám[1][0000−0002−5647−6134]

RWTH Aachen University, Aachen, Germany
{promies, abraham}@cs.rwth-aachen.de

Abstract. The cylindrical algebraic covering method is an approach for deciding the satisfiability of non-linear real arithmetic formulas. We extend that method for optimization modulo theories (OMT), allowing to find solutions that are optimal w.r.t. a given polynomial objective function. Our approach is complete and detects unbounded objective functions as well as infima/suprema, which the objective function can approach, but never reach. We show how to construct meaningful models even in those special cases and provide an experimental evaluation demonstrating the advantages of our method compared to approaches based on quantifier elimination or incremental linearization.

Keywords: Optimization Modulo Theories · Real Algebra · SMT Solving · Cylindrical Algebraic Decomposition

1 Introduction

Optimization modulo theories (OMT) is the task of minimizing (or maximizing) a given objective function with respect to a given first-order formula and a background theory. It extends *satisfiability modulo theories (SMT)*, which only requires finding *any* solution to a given formula. While SMT solvers are versatile and effective tools for formal reasoning tasks, some problems require or greatly benefit from finding *optimal* solutions, and consequently, there has been active research on OMT in the last years [4,9,13,14].

We are interested in the theory of *non-linear real arithmetic (NRA)*, whose atomic formulas are polynomial constraints over real-valued variables. Already SMT solving (without optimization) for NRA is quite challenging: in practice, all complete methods are currently based on techniques from cylindrical algebraic decomposition [5], which suffers from a doubly exponential worst case complexity. However, significant progress has been made in the last years, allowing to solve many problems efficiently [2,10,11,12]. A notable advance is the *cylindrical algebraic covering* (CAlC) method, which was first developed as a theory backend for CDCL(T) style solving [2] and later extended in [11] to handle Boolean structure and even quantifiers, making it independent of CDCL(T) architectures and allowing to use it for quantifier elimination tasks. Using [2], the CVC5 solver

won the QF_NRA category of the SMT competition [1] in 2021 and 2022, and the SMT-RAT solver used [11] to win the NRA category in 2024.

In the case of OMT, there has been little research targeting NRA problems. The OPTIMATHSAT solver uses incremental linearization [4], which can often find solutions quickly, but it is incomplete. A naive but complete approach is to use quantifier elimination to determine the entire range of the objective function, as the optimum is then the lower (or upper) bound of that range. However, this is inefficient as it computes much more than necessary, and it does not easily provide models at which the optimum is reached.

Probably the most notable result here is [9], which was only published very recently, after we did most of the research for this paper. That approach uses CDCL with the CAlC theory backend as in [2] to enumerate solutions with improving objective value. Importantly, additional information from the CAlC backend is used to 1) derive better lower/upper bounds on the optimum in each iteration and 2) guarantee completeness. While we present similar techniques, there are significant differences and additional contributions, specified below.

There are also numerical optimization methods not stemming from the field of OMT, but these usually cannot detect infima or unboundedness, and they do not provide strong correctness guarantees as they work with floating point arithmetic. Moreover, these methods usually do not support side conditions with Boolean structure. Note that of all mentioned approaches, so far only quantifier elimination can handle quantifiers in the formula.

Contribution. In this paper, we propose an extension of the CAlC method for solving optimization modulo NRA. In contrast to [9], our approach is based on the more general version of CAlC [11] which already proved to be very effective for SMT solving on quantified problems.

Our approach lifts most of the restrictions of other methods mentioned above:

- it can handle arbitrary Boolean structure, including quantifiers,
- it uses exact arithmetic and is not depending on a user-defined precision,
- it is complete and correct, and detects if the objective function is unbounded or if only an infimum is achievable instead of a true minimum.

Moreover, we show how this approach can not only find the optimal objective value, but also provide a model realizing that value. While this is generally not possible if the objective has no true minimum, we present a method for constructing models with an objective value arbitrarily close to the infimum (or to $-\infty$), with little overhead. None of the previous approaches provide this capability, although our result could easily be transferred to the other CAlC-based method [9].

Finally, our approach outperforms other tools ([4,9]) in experiments generated from the QF_NRA set provided by SMT-LIB.

Structure. We first present the necessary background in Section 2, in particular the CAlC method, which we extend for optimization in Section 3. We address model construction in Section 4 and discuss our experiments in Section 5. Finally, Section 6 concludes this paper with an outlook on the next steps.

2 Preliminaries

Non-Linear Real Arithmetic Formulas. Since we are building on [11], the following basic definitions and notations are mostly taken from that paper.

Let $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$ denote the natural (including 0), rational and real numbers, respectively. Given $i \in \mathbb{N}$, $s \in \mathbb{R}^i$, $s' \in \mathbb{R}$, $R \subseteq \mathbb{R}^i$ and $I \subseteq \mathbb{R}$, we denote by $s \times s'$ the point $(s_1, \ldots, s_i, s') \in \mathbb{R}^{i+1}$, by $s \times I$ the set $\{s \times s'' \mid s'' \in I\} \subseteq \mathbb{R}^{i+1}$, and by $R \times I$ the set $\{r \times s'' \mid r \in R, s'' \in I\} \subseteq \mathbb{R}^{i+1}$. A *cell* is a non-empty connected set $R \subseteq \mathbb{R}^i$.

For the rest of this paper, we fix some $n \in \mathbb{N} \setminus \{0\}$ and a collection of ordered real variables $x_1 \prec \ldots \prec x_n$. Let $i \in \{1, \ldots, n\}$. The set of all polynomials with variables $x_1, \ldots, x_i$ and rational coefficients is denoted by $\mathbb{Q}[x_1, \ldots, x_i]$. A (polynomial) *constraint* has the form $p \sim 0$ for some $p \in \mathbb{Q}[x_1, \ldots, x_n]$ and $\sim \in \{<, \leq, =, \neq, \geq, >\}$. An *NRA formula* (or simply, a formula) is a Boolean combination of constraints, using $\neg, \vee, \wedge$, as well as quantifiers $\forall, \exists$.

We assume that all considered formulas are in *prenex normal form (PNF)*, i.e. they have the form

$$\varphi = Q_{k+1} x_{k+1} \ldots Q_n x_n . \overline{\varphi}(x_1, \ldots, x_n)$$

consisting of a quantifier *prefix* with $Q_{k+1}, \ldots, Q_n \in \{\exists, \forall\}$ and a quantifier free formula $\overline{\varphi}$ called *matrix*. Here, $k \in \{0, \ldots, n\}$ indicates the number of *free* variables; if $k = 0$, there are no free variables, and if $k = n$, there are no quantifiers. We sometimes write $\varphi(x_1, \ldots, x_k)$ to indicate the free variables.

Given $s \in \mathbb{R}^i$ and a formula $\varphi(x_1, \ldots, x_k)$, the *(partial) evaluation up to level* i of φ over s is denoted by $\varphi[s]$. That is, constraints of level i or lower evaluate to `True` or `False` according to standard semantics, by substituting s_1 for x_1, s_2 for x_2 etc., and constraints of higher levels evaluate to `Undef`. The semantics is extended for formulas inductively according to the three-value semantics of the logical operators, e.g. $\texttt{Undef} \wedge \texttt{True} = \texttt{Undef}$ and $\texttt{Undef} \wedge \texttt{False} = \texttt{False}$.

A polynomial $p \in \mathbb{Q}[x_1, \ldots, x_i]$ is *sign-invariant* on a set $R \subseteq \mathbb{R}^i$, if it has the same sign (positive, negative, or zero) on every point $r \in R$. A formula φ is *truth-invariant* on R, if it evaluates to the same truth value on each $r \in R$.

Optimization. We are interested in the following problem: given an NRA formula $\varphi(x_1, \ldots, x_k)$ and a polynomial $p \in \mathbb{Q}[x_1, \ldots, x_k]$, find

$$min\langle p \mid \varphi \rangle := \min\{p(s_1, \ldots, s_k) \mid s \in \mathbb{R}^k \wedge \varphi[s] = \texttt{True}\}.$$

Note that we can easily transform maximization problems to minimization by negating the objective function. W.l.o.g. only consider problems of the form $min\langle x_1 \mid \varphi \rangle$; for any problem $min\langle p \mid \varphi \rangle$, we can consider $min\langle x^* \mid \varphi \wedge (x^* = p) \rangle$ instead, with a fresh variable x^* (and we can then rename the variables).

The existence of the desired minimum is only guaranteed if φ is satisfiable and its solution set S is compact (bounded and closed). If φ is unsatisfiable, then we are taking the minimum of an empty set, which is not well-defined. If S is not compact, then also the set $\{p(s_1, \ldots, s_k) \mid s \in \mathbb{R}^k \wedge \varphi[s] = \texttt{True}\} \subset \mathbb{R}$

(the image of S under p) might be unbounded, i.e. p can diverge towards $-\infty$, or it might only have an infimum, towards which p can converge, but which it cannot reach within S. Ideally, an algorithm for solving non-linear optimization recognizes these situations and gives an appropriate output, e.g. "UNSAT" if φ is unsatisfiable, $-\infty$ if the objective is unbounded, or $\ell + \varepsilon$ for an infimum ℓ.

2.1 The Cylindrical Algebraic Covering Method

We now briefly recall the CAlC method for checking the satisfiability of NRA formulas as presented in [11], focusing on the parts needed here. The method is summarized in Algorithm 1, which essentially merges [11, Algorithms 2,3 and 4] into a single algorithm for a more concise presentation.

Algorithm 1: CAlC(s)

Data : Global prefix $Q_{k+1}x_{k+1}\cdots Q_n x_n$ and matrix $\overline{\varphi}$
Input : Sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$
Output : (SAT, C) or (UNSAT, C) where C is an implicit cell of level $i-1$

```
1  𝕀 := ∅                                          // collects the covered cells
2  while ⋃_{C∈𝕀} C.I ≠ ℝ do
3      s_i := sample_outside(𝕀)
4      if φ̄[s × s_i] = False then
5          (f, O) := (UNSAT, get_enclosing_cell(s × s_i))
6      else if φ̄[s × s_i] = True then
7          (f, O) := (SAT, get_enclosing_cell(s × s_i))
8      else
9          (f, O) := CAlC(s × s_i)
10     if (f = SAT ∧ (i ≤ k ∨ Q_i = ∃)) or (f = UNSAT ∧ (i > k ∧ Q_i = ∀)) then
11         C := characterize_cell(s, O)
12         return (f, C)        // early return depending on the quantifier
13     else
14         𝕀 := 𝕀 ∪ {O}
15 C := characterize_covering(s, 𝕀)
16 if i > k ∧ Q_i = ∀ then return (SAT, C)
17 else return (UNSAT, C)
```

Given a formula $\varphi = Q_{k+1}x_{k+1}\ldots Q_n x_n.\overline{\varphi}$, the idea of this method is to construct a satisfying sample point recursively, i.e. level by level. The initial call to CAlC (i.e. level 1) starts with an empty sample $s = ()$, and selects a value $s_1 \in \mathbb{R}$ for x_1. In general, a recursive call CAlC(s) for level i takes as input a partial sample $s \in \mathbb{R}^{i-1}$ for $x_1, \ldots, x_{i-1}$, and it tries to extend that sample with a value s_i for x_i. The call returns SAT if s can be extended to a satisfying assignment of φ, and otherwise UNSAT. In both cases, it also returns information on how the sample can be generalized to a satisfying or falsifying *cell*.

Definition 1 (Implicit Cell [11]). *Let $i \in \mathbb{N} \setminus \{0\}$, $P \subseteq \mathbb{Q}[x_1, \ldots, x_i]$, $s \in \mathbb{R}^i$, and let $I \subseteq \mathbb{R}$ be an interval. Further, let $R \subseteq \mathbb{R}^i$ be the maximal connected subset containing s where all polynomials in P are order-invariant.*[1] *The tuple (P, s, I) is an* implicit cell of level i*, if $I = \{r \in \mathbb{R} \mid (s_1, \ldots, s_{i-1}, r) \in R\}$.*

We call R the cell induced by P and s*. Given an implicit cell $C = (P, s, I)$, we refer to its components by $C.P$, $C.s$ and $C.I$. Moreover, $C.I.l \in \mathbb{R} \cup \{-\infty\}$ and $C.I.u \in \mathbb{R} \cup \{\infty\}$ denote the lower and upper bound of the interval I.*

The polynomials of each implicit cell computed by `CAlC` are derived from the polynomials in φ using CAD techniques, and the respective intervals are used to exclude parts of the search space on each level.

Let us assume first that x_i is not universally quantified. After guessing a value s_i (with `sample_outside`), the recursive call of level i checks whether $\overline{\varphi}[s \times s_i]$ has a definite truth value. If it evaluates to `True`, then a satisfying sample is found, and the call returns `SAT`. If it evaluates to `False`, then the method computes an implicit cell C of level i around $s \times s_i$ (with `get_enclosing_cell`), such that all points in $s \times C.I$ falsify $\overline{\varphi}$, and thus $C.I$ can be excluded from the search. Otherwise, a recursive call `CAlC(`$s \times s_i$`)` for level $i+1$ determines whether $s \times s_i$ can be extended to a satisfying sample. If not, then the recursive call provides an implicit cell (and an interval) to exclude.

This is repeated until either a satisfying sample outside the excluded region is found or until the intervals form a *covering* of the entire real line. In the latter case, s cannot be extended to a solution and thus the current call will return `UNSAT`. Importantly, the call now also returns an implicit cell of level $i - 1$ around s, which is derived from the covering of implicit cells of level i using CAD techniques (with `characterize_covering`). This cell again provides an interval to exclude on the previous level $(i - 1)$, where it might be part of another covering.

If x_i is universally quantified, then the roles of satisfying and falsifying samples are reversed: to infer satisfiability, a covering of *satisfying* intervals has to be computed, and therefore, also satisfying samples (or samples that can be extended to satisfying ones) are always generalized to an implicit cell. In particular, when a call for an existentially quantified (or free) variable returns `SAT`, it also provides a "projection" of the satisfying cell onto level $i - 1$, again using CAD techniques (with `characterize_cell`).

We do not give a detailed explanation of the subroutines `sample_outside`, `get_enclosing_cell`, `characterize_covering`, `characterize_cell`. The important fact for this paper is that all computed implicit cells generalize the current sample in the following sense: if s can (cannot) be extended to a satisfying sample, then the same holds for all points in the induced cell.

The `CAlC` method is correct and complete. That is, the initial call will always terminate with the correct result (`SAT` or `UNSAT`). This is due to the fact each call is guaranteed to cover the real line with a *finite* number of cells (or find a solution before that).

[1] Order-invariance is a strengthening of sign-invariance. For details, see [2].

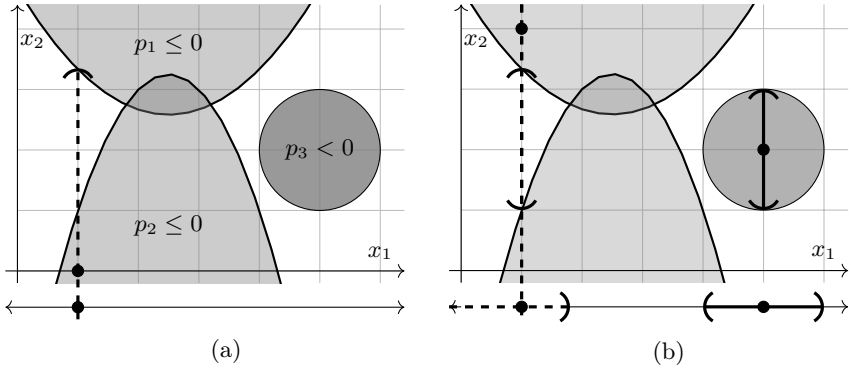

Fig. 1: Illustration of Example 1. The shaded areas indicate the solution set of each constraint; darker shaded areas are the solution set of φ. Satisfiable intervals are indicated with a solid line, unsatisfiable intervals with a dashed line. The additional axis below shows reasoning in the x_1 dimension.

Example 1. Consider the non-linear optimization problem $min\langle x_1 \mid \varphi\rangle$, where $\varphi = (p_1 \leq 0 \wedge p_2 \leq 0) \vee (p_3 < 0)$ and $p_1, p_2, p_3 \in \mathbb{Q}[x_1, x_2]$ are given by

$$x_1^2 - 5x_1 + 14 - 3x_2, \qquad x_1^2 - 5x_1 + 3 + x_2, \qquad (x_1 - 5)^2 + (x_2 - 2)^2 - 1.$$

The satisfying regions of φ and its constraints are depicted in Figure 1. For now, we ignore the objective and use `CAlC` to check the satisfiability of φ.

We start by picking any value for x_1, e.g. $s_1 = 1$. As $\varphi[s_1] =$ `Undef`, we continue with x_2 and choose $s_2 = 0$. Now, $\varphi[(s_1, s_2)] =$ `False` and the interval $(-\infty, 10/3)$ (where $p_1 \leq 0$ is falsified) is excluded for x_2, as depicted in Figure 1a.

Trying $s_2 = 4$ next also gives the unsatisfiable interval $(1, \infty)$ for $p_2 \leq 0$, leading to a *covering* of $\mathbb{R}$ over $s_1 = 1$ (Figure 1b, left). This can be generalized to an unsatisfiable interval $(-\infty, (5 - \sqrt{2})/2)$ for x_1, prompting the algorithm to choose a larger value, e.g. $s_1 = 5$. Now, extending it with $s_2 = 2$ satisfies φ, and the model can also be generalized to satisfying intervals for x_2 and x_1 (Figure 1b, right).

3 Using CAlC for Non-Linear Optimization

We now show how the CAlC method can be adapted for optimization. Our idea is relatively simple: the objective variable x_1 is treated on the bottom level of the method, i.e. it is always assigned first and then CAlC is called recursively as usual. This call determines whether there is a solution of φ with $x_1 = s_1$ and, importantly, it generalizes s_1 to an implicit cell C. If s_1 cannot be extended to a solution of φ, the entire interval $C.I$ can be excluded from the search, as before. Otherwise, s_1 *can* be extended to a solution, and then the same holds for all values in $C.I$. In that case, the lower bound of that interval provides

an upper bound on the desired minimum. If $C.I$ has no lower bound, then the objective function is also unbounded, and thus we can immediately return that the minimum is $-\infty$. If the lower bound l of $C.I$ is strict (i.e. I is left-open), then the new upper bound is only $l+\varepsilon$, since we do not yet know whether $x_1 = l$ is viable. In the subsequent search, we are only interested in values of x_1 that are smaller than the current upper bound, and do not want to pick values above $C.I$. Therefore, we exclude a new interval, which has the same lower bound as $C.I$, but whose upper bound is ∞.

The algorithm stops when the entire real line is covered by satisfying and/or unsatisfying intervals. If all intervals are unsatisfying, then the formula is unsatisfiable, hence `UNSAT` is returned. Otherwise, it returns `SAT` together with the minimum. Our idea is formalized in Algorithm 2.

The correctness and completeness of our approach are straightforward, given the correctness and completeness of CAlC. In the `UNSAT` case, our algorithm behaves exactly like CAlC. In the `SAT` case, we additionally have to ensure that the returned minimum is correct. The way we exclude intervals guarantees that we never choose any value s_1 above the current value of `min`, i.e. it is strictly decreasing throughout the algorithm. When the algorithm terminates, then all possible values for s_1 have been covered and `min` contains the lower bound of the lowest found satisfying interval. The techniques of CAlC then guarantee that all values below are contained in unsatisfying intervals, i.e. there is no solution of φ with any of those objective values.

Algorithm 2: `minimize()`

Data : Global prefix $Q_{k+1}x_{k+1}\cdots Q_n x_n$ and matrix $\overline{\varphi}$.
Output : $(\texttt{SAT}, o)$ or $(\texttt{UNSAT})$.

```
1  min := ∞
2  𝕀 := ∅
3  while ⋃_{I∈𝕀} I ≠ ℝ do
4      s_1 := sample-outside(𝕀)
5      (f, C) := CAlC((s_1))
6      if f = SAT then
7          if C.I = (−∞, u) or C.I = (−∞, u] then
8              return (SAT, −∞)
9          else if C.I = [l, u] or C.I = [l, u) then
10             min := l
11             𝕀 := 𝕀 ∪ {[l, ∞)}
12         else /*(C.I = (l, u) or C.I = (l, u])*/
13             min := l + ε
14             𝕀 := 𝕀 ∪ {(l, ∞)}
15     else
16         𝕀 := 𝕀 ∪ {C.I}
17 if min < ∞ then return (SAT, min)
18 else return UNSAT
```

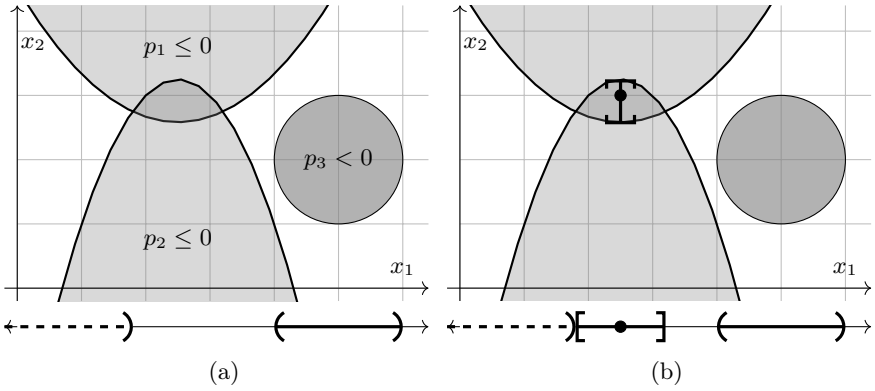

Fig. 2: Illustration of Example 2.

Theorem 1. *Algorithm 2 is correct and complete. That is, let the given input be $\varphi(x_1, \ldots, x_k)$, and let $M := \{s_1 \mid s \in \mathbb{R}^k \wedge s \models \varphi\}$, then Algorithm 2 returns*

- `UNSAT` *iff φ is unsatisfiable,*
- $(\texttt{SAT}, -\infty)$ *iff φ is satisfiable and M has no lower bound (in $\mathbb{R}$),*
- $(\texttt{SAT}, r)$ *for some $r \in \mathbb{R}$ iff φ is satisfiable,* $\min(M)$ *exists and is equal to r,*
- $(\texttt{SAT}, r + \varepsilon)$ *for some $r \in \mathbb{R}$ iff φ is satisfiable,* $\inf(M) = r$*, but $r \notin M$.*

□

Example 2. We revisit Example 1, and now we actually want to minimize x_1. The first steps can be identical to Example 1, i.e. `minimize()` tries out $s_1 = 1$ and $s_1 = 5$, obtaining an unsatisfiable interval $(-\infty, (5 - \sqrt{2})/2)$ and a satisfiable interval $(4, 6)$, as shown in Figure 2a. Now, `minimize()` has to continue and choose a sample below the satisfying interval, e.g. $s_1 = 2.5$. The call `CAlC(`s_1`)` finds the satisfying extension $s_2 = 3$, which can be generalized to an interval for x_2 between the roots of p_2 and p_1. From this characterization, a satisfying interval for x_1 is derived whose bounds correspond to the intersections of the roots of p_1 and p_2, as shown in Figure 2b. These intervals include their bounds since the constraints are weak. Now, all values below the new interval are already covered, and the algorithm can stop with the optimum $(5 - \sqrt{2})/2$.

4 Model Construction

Some applications require not only the optimal value of the objective function, but also a variable assignment (model) at which this value is realized. That is, given $\varphi(x_1, \ldots, x_k)$, we want to find $m \in \mathbb{R}^k$ with $m \models \varphi$ and $m_1 = min\langle x_1 \mid \varphi \rangle$. In this section, we explain how to construct such a model with a little additional bookkeeping. We start with the case of a true minimum, and then extend our technique for infima and the unbounded case.

It is important to note that, while our algorithm already constructs models for φ in the sample s, these models are not necessarily realizing the optimal objective

value. The reason for this counter-intuitive observation is that satisfying samples are generalized to cells, and we only infer the minimal objective value within that cell by using the lower bound of the interval $C.I$ in Algorithm 2, but we do not yet construct a model for it.

4.1 Models for True Minima

When there is a true minimum, i.e. $min\langle x_1 \mid \varphi\rangle = r \in \mathbb{R}$, then one could easily construct a model by calling `CAlC(r)` (fixing x_1 to r) for φ. However, this does more work than necessary, as it might first explore many infeasible branches in the recursion. Instead, our idea is to store the implicit cells corresponding to the levels of the most recent satisfying sample and use them to iteratively derive the desired model.

Whenever a satisfying sample is found, the current call stack induces a stack of implicit cells $C_k, \ldots, C_1$ which generalize the sample on each level. More precisely, for $i \in \{1, \ldots, k\}$, the call of level $i{+}1$ returns the cell C_i, i.e. $s_i \in C_i.I$. The cell C_1 is used to determine the minimum $min\langle x_1 \mid \varphi\rangle = r$ as the lower bound of $C_1.I$, which is also the starting point for our model $m \in \mathbb{R}^k$, i.e. $m_1 := r$. For $i \in \{2, \ldots, k\}$, we then iteratively use s, C_i and $(m_1, \ldots, m_{i-1})$ to determine a value for m_i. For this purpose, we make use of *symbolic intervals*, which consist of *indexed root expressions*.

Definition 2 (Indexed Root Expression, [11]). *Let $i, j \in \mathbb{N} \setminus \{0\}$ and let $p \in \mathbb{Q}[x_1, \ldots, x_i]$. An* indexed root expression *is a function*

$$\mathrm{root}_{p,j} : \mathbb{R}^{i-1} \to \mathbb{R} \cup \{\mathit{undefined}\},$$

s.t. for all $r \in \mathbb{R}^{i-1}$, $\mathrm{root}_{p,j}(r)$ is the j-th real root of the univariate polynomial $p(r, x_i) \in \mathbb{R}[x_i]$ (substituting r_1 for x_1, r_2 for x_2 etc.), or undefined if that root does not exist (i.e. if $p(r, x_i) = 0$ or if it has less than j roots).

Definition 3 (Symbolic Interval). *Let $i \in \mathbb{N}\setminus\{0\}$, $C = (P, s, I)$ be an implicit cell of level i and let $\Xi_l := \{\mathrm{root}_{p,j} \mid p \in P, j \in \mathbb{N}, \mathrm{root}_{p,j}(s_1, \ldots, s_{i-1}) = C.I.l\}$ and analogously $\Xi_u := \{\mathrm{root}_{p,j} \mid p \in P, j \in \mathbb{N}, \mathrm{root}_{p,j}(s_1, \ldots, s_{i-1}) = C.I.u\}$.*

We set $\psi_l := (\xi_l(x_1, \ldots, x_{i-1}) < x_i)$ for some $\xi_l \in \Xi_l$ (or $\psi_l =$ `True` if $\Xi_l = \emptyset$) and $\psi_u := (\xi_u(x_1, \ldots, x_{i-1}) > x_i)$ for some $\xi_u \in \Xi_u$ (or $\psi_u =$ `True` if $\Xi_u = \emptyset$). If the lower bound of $C.I$ is closed (i.e. $C.I.l \in C.I$), then we use $\leq$ instead of $<$ in ψ_l, and we proceed analogously for the upper bound and ψ_u.

The formula $\psi(x_1, \ldots, x_i) := \psi_l \wedge \psi_u$ defines a symbolic interval *on x_i.*

Importantly, Ξ_l and Ξ_u can be computed efficiently and in fact, they are already used implicitly in subroutines of `CAlC`, thus the overhead for deriving a symbolic interval from an implicit cell is negligible. Now, let $\psi_i(x_1, \ldots, x_i)$ be the symbolic interval on x_i derived from C_i. All indexed root expressions in ψ_i can be evaluated on the partial model $(m_1, \ldots, m_{i-1})$ and yield a real value. This way, ψ_i induces a real interval $J_i \subseteq \mathbb{R}$ of suitable values m_i for x_i, and we can choose any such value. Note that the implicit cell $(C_i.P, (m_1, \ldots, m_i), J_i)$

defines the same maximal order-invariant region of $\mathbb{R}^i$ as C_i, i.e. $C_i.I$ and J_i are different slices of the same cell. Therefore, it is guaranteed that our final model $m \in \mathbb{R}^k$ lies in the same satisfying cell as s and is indeed a model.

Example 3. We continue Example 2. As mentioned there, the implicit cell for x_2 is defined by p_1 and p_2, and we can derive the symbolic interval $\psi_2 = (\text{root}_{p_1,1}(x_1) \leq x_2 \wedge x_2 \leq \text{root}_{p_2,1}(x_1))$. This interval is only non-empty if the root of p_1 is below or equal to the root of p_2, which is the case if x_1 satisfies $\psi_1 = (\text{root}_{q,1} \leq x_1 \wedge x_1 \leq \text{root}_{q,2})$, where $q \in \mathbb{Q}[x_1]$ is the *resultant* of p_1 and p_2. We set m_1 to the lower bound $(5 - \sqrt{2})/2$, and evaluating ψ_2 yields $2.75 \leq x_2 \leq 2.75$, leaving only the choice $m_2 = 2.75$. Note that the intervals are not alway point intervals, and then we can choose any value from them.

4.2 Model Templates for Special Cases

The process described above does not work if the objective function is unbounded or if only an infimum is achievable, because we cannot plug symbolic values like $-\infty$ or $r + \varepsilon$ into symbolic intervals to obtain concrete intervals.

Example 4. Consider the formula φ which is given in Figure 3, together with a depiction of its solution set. The satisfiable range of x_1 is $(2, \infty)$, and therefore our approach finds $min\langle x_1 \mid \varphi \rangle = 2 + \varepsilon$. The optimal (and only) satisfying cell is described by $2 < x_1 \wedge (\text{root}_{f,1}(x_1) < x_2 < \text{root}_{g,1}(x_1))$. As x_1 approaches 2 from above, both root expressions in the cell description tend to infinity and at $x_1 = 2$ neither f nor g has a root and thus both expressions are *undefined*.

This means that we cannot provide a "true" model for these special cases. However, we can provide the symbolic intervals $\psi_1, \ldots, \psi_k$ to the user as a convenient description of the optimal truth-invariant cell for φ. In particular, this description can be used as a *model template* to easily compute *nearly-optimal* models: if the optimum is $r + \varepsilon$, then the user may choose some fixed value $\varepsilon > 0$ and

$\varphi : f < 0 \ \wedge \ g > 0 \ \wedge \ h \geq 0$
$f = 4(x_2 - 1)(x_1 - 2) - 1$
$g = (x_2 - 1)(x_1 - 2) - 1$
$h = x_2 - 1$

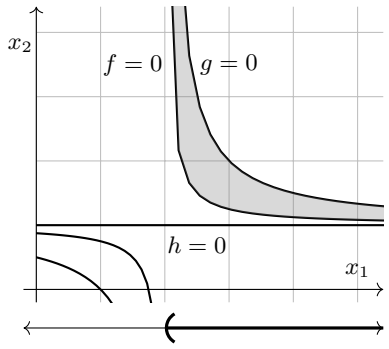

Fig. 3: The formula φ (left) used in Example 4 and its solution set (right, shaded), as well as the optimal interval for x_1 w.r.t. φ.

thus turn the symbolic value into a real one (note that ψ_1 might imply an upper bound on ε). Using this value as m_1, we can then compute a model as before. Similarly, if the optimum is $-\infty$, we can compute models with arbitrarily small values for m_1 (again, ψ_1 might provide an upper bound).

Notably, the model computation is quite cheap and fast compared to the previous call to `CAlC`. This allows to easily and quickly compute nearly-optimal models for different values of ε without having to recompute everything.

Algorithm 3 shows how Algorithm 1 can be adapted to compute and store the symbolic intervals $\psi_1, \ldots, \psi_k$. They can be stored globally and overwritten each time a new (better) satisfying cell is found. The important change is in Line 6. Note that we only need models for the free variables, i.e. if $i \leq k$.

Algorithm 4 shows how to compute a model from the computed symbolic intervals, when the desired objective value m_1 is provided.

Algorithm 3: `CAlC`(s)

Data : Global prefix $Q_{k+1}x_{k+1} \cdots Q_n x_n$, matrix $\overline{\varphi}$, and symbolic intervals $\psi_1, \ldots, \psi_k$
Input : Sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$
Output : $(\texttt{SAT}, C)$ or $(\texttt{UNSAT}, C)$ where C is an implicit cell of level $i-1$

1 $\mathbb{I} := \emptyset$ `// collects the covered cells`
2 **while** $\bigcup_{C \in \mathbb{I}} C.I \neq \mathbb{R}$ **do**
3 $\quad s_i :=$ `sample_outside(`$\mathbb{I}$`)`
4 $\quad$ Determine result (f, O) for $s \times s_i$ (Algorithm 1, lines 4-9)
5 $\quad$ **if** $(f = \texttt{SAT} \wedge (i \leq k \vee Q_i = \exists))$ ***or*** $(f = \texttt{UNSAT} \wedge (i > k \wedge Q_i = \forall))$ **then**
6 $\quad\quad$ **if** $i \leq k$ **then** $\psi_i :=$ `symbolic-interval-from(`O`)`
7 $\quad\quad$ $C :=$ `characterize_cell(`s, O`)`
8 $\quad\quad$ **return** (f, C) `// early return depending on the quantifier`
9 $\quad$ **else**
10 $\quad\quad$ $\mathbb{I} := \mathbb{I} \cup \{O\}$
11 $C :=$ `characterize_covering(`s, $\mathbb{I}$`)`
12 **if** $i > k \wedge Q_i = \forall$ **then return** $(\texttt{SAT}, C)$
13 **else return** $(\texttt{UNSAT}, C)$

Algorithm 4: `compute-model`$(\psi_1, \ldots, \psi_k, m_1)$

Input : Symbolic intervals $\psi_1, \ldots, \psi_k$ for $x_1, \ldots, x_k$ and $m_1 \in \mathbb{R}$ with $m_1 \models \psi_1$
Output : $m = (m_1, \ldots, m_k) \in \mathbb{R}^k$ with $m \models \psi_1 \wedge \ldots \wedge \psi_k$

1 **for** $i = 2, \ldots, k$ **do**
2 $\quad J_i :=$ `real-interval(`$\psi_i(m_1, \ldots, m_{i-1}, x_i)$`)`
3 $\quad$ choose $m_i \in J_i$
4 **return** m

5 Experiments

Tested Solvers. We implemented our approach in the SMT-RAT solver [6], building on its implementation of `CAlC` as in [11]. In our experimental evaluation, we compared the following solvers:

- `CAlC-Opt`: our approach, implemented in SMT-RAT.
- `CAlC-Opt+`: our approach, but using an adapted variable ordering. It is well known that the variable ordering (for the assignments/projections) greatly influences the performance of methods based on cylindrical algebraic decomposition, like CAlC [7,8]. As we enforce that the objective variable is assigned first (and thus projected last), the efficiency of the remaining ordering might be impacted. We address this by sorting the remaining variables into two tiers: first, those occurring in a constraint together with the objective variable and second, those that don't. Within each tier, we use SMT-RAT's standard variable ordering heuristic. The idea is to give more priority on enforcing that the objective function takes on the chosen value, thus potentially finding its range more quickly.
- `CAlC-QE`: determining the optimum via quantifier elimination, by treating all free variables except the objective variable as existentially quantified. The used quantifier elimination is also based on `CAlC`, as in [11].
- `CDCL-OCAC`: an implementation of [9], based on the CVC5 solver.
- `OptiMathSAT`: an incomplete approach using incremental linearization [4].

Benchmark Sets. A severe limitation for our experiments is the lack of suitable and readily available benchmark sets for our targeted kind of optimization problem. The only such set that we found is provided in [4], where it is used to evaluate `OptiMathSAT`, hence we call this set OMS. It contains 752 instances derived from planning tasks and unfortunately, we will see later that it is generally too difficult for our considered tools (similar observations are also found in [4]).

Therefore, we decided to generate two new benchmark sets based on the QF_NRA set provided by SMT-LIB [3], which contains 12154 satisfiability problems with quantifier-free NRA formulas. For each of these problems, we generated two optimization problems with different objective functions. Firstly, we simply choose a random variable as the objective function. Secondly, we choose two random variables x, y and use the polynomial $(x^2 - 1)(y^2 - 1)$ as objective function, which is non-convex and non-concave and thus expected to yield more complex problems. This yields the sets QFNRA and QFNRA-hard, with 12152 and 12145 instances, because there are some instances in the original set with only one or even zero real variables.

Note that our tool can actually handle arbitrary NRA formulas, including quantifiers. However, since the other tools do not have this capability, we do not present experiments with such formulas here.

Testing Setup. We tested all solvers on all benchmarks with a time limit of one minute and a memory limit of 4 GB per instance. All tests were conducted on on identical Intel®Xeon®8468 Sapphire CPUs with 2.1 GHz per core.

Evaluation. The results are summarized in Table 1. Regarding the simple QFNRA benchmarks, both variants of our approach (`CAlC-Opt`, `CAlC-Opt+`) solve at least 200 instances more than the other competitors, and this gap even widens to roughly 700 on the QFNRA-hard benchmarks. Our adapted variable ordering improves the overall performance of our approach: while `CAlC-Opt+` solves fewer unsatisfiable instances than `CAlC-Opt`, it solves significantly more satisfiable instances (and thus more overall).

Table 1: Summary of experimental results. MO indicates memory-outs.

		`CAlC-QE`	`CAlC-Opt`	`CAlC-Opt+`	`CDCL-OCAC`	`OptiMathSAT`
QFNRA	solved	9370	9824	**9847**	9624	7944
	sat	4430	4884	**4923**	4596	2696
	unsat	4940	4940	4924	5028	**5248**
	MO	151	147	145	1	230
QFNRA-hard	solved	8071	9017	**9103**	8301	5800
	sat	3201	4146	**4286**	3409	595
	unsat	4870	4871	4817	4892	**5205**
	MO	154	145	147	0	6
OMS	solved	11	12	11	**148**	114
	sat	0	1	1	0	**5**
	unsat	11	11	10	**148**	109
	MO	24	25	26	0	0

The outcome for the OMS set is different: the `CAlC` based tools solve hardly any instances and `CDCL-OCAC` and `OptiMathSAT` solve at least 100 instances more, which still is not even 20% of the set. Notably, while `CDCL-OCAC` solves the most instances in OMS, it does not solve a single satisfiable instance, and `OptiMathSAT` only solves 5. It is important to note that, while `CDCL-OCAC` employs similar techniques like our approach, it is implemented on top of CVC5, which also uses incremental linearization similar to `OptiMathSAT`. Table 1 shows that `OptiMathSAT` (and hence incremental linearization) excels especially on unsatisfiable instances. It is therefore likely that this causes most of the difference in performance between our approach and `CDCL-OCAC` on the OMS set.

Our approach behaves very similar to `CAlC-QE` on unsatisfiable instances, but solves far more satisfiable instances. This is expected, as both build onto the same `CAlC` implementation, and both have to cover the entire search space to infer unsatisfiability, but `CAlC-Opt` can potentially avoid a lot of work in the satisfiable case.

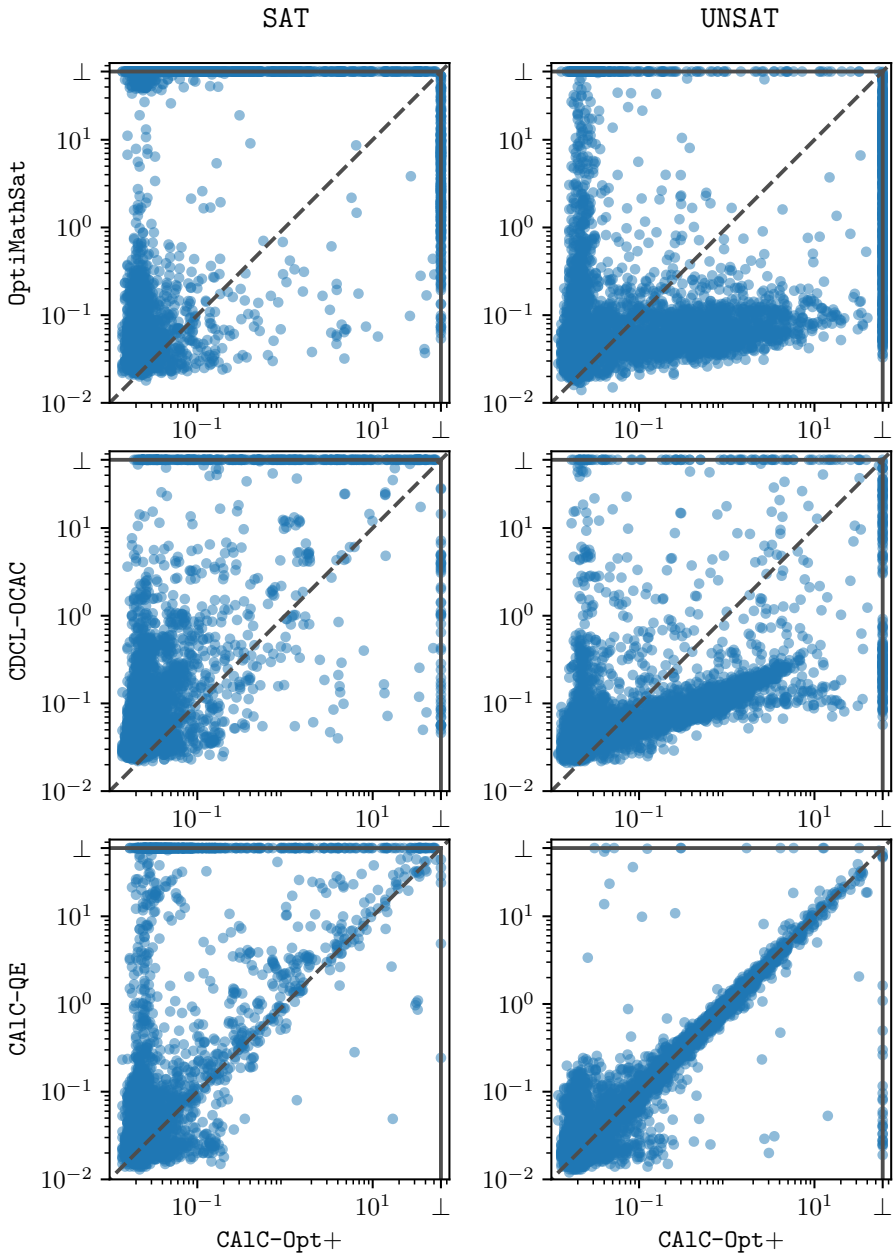

Fig. 4: Comparison of the running times (in seconds) of `CAlC-Opt+` (x-axis) with other tools on the QFNRA set, divided by satisfiable and unsatisfiable instances. Instances unsolved by the respective tool are depicted at the label $\perp$.

The different behaviours for satisfiable and unsatisfiable instances (of the QFNRA set) can also be seen in Figure 4. Interestingly, `CAlC-Opt+` seems to be quite complementary to `OptiMathSAT` and `CDCL-OCAC`: There is a significant number of (both satisfiable or unsatisfiable) instances which one tool can solve but not the other and vice versa; these instances are shown on the lines labelled $\perp$. Moreover, even if both tools (`CAlC-Opt+` and `OptiMathSAT` or `CDCL-OCAC`) solve an instance, one tool is often clearly faster, thus there are few dots around the main diagonal (in contrast to the comparison with `CAlC-QE`).

Validation. We validated the results of our solver by encoding the correctness of each output as satisfiability problems, which we checked with other established SMT solvers, namely z3 [16] and YicesQS [15]. Let $min\langle p \mid \varphi(x_1, \ldots, x_k)\rangle$ be the input optimization problem. If our solver returns `UNSAT`, we simply verify that φ is unsatisfiable. Otherwise, (i.e. our solver returns `SAT` and some optimum O), we have to make more case distinctions.

- If $O = m \in \mathbb{Q}$, verify that $\varphi \wedge p = m$ is satisfiable and that $\varphi \wedge p < m$ is unsatisfiable.
- If $O = m + \varepsilon$ for some $m \in \mathbb{Q}$, verify that $\varphi \wedge p \leq m$ is unsatisfiable and that $\exists t.(m < t \wedge t < u \wedge \forall x_1, \ldots, x_k.(p(x_1, \ldots, x_k) = t \rightarrow \neg\varphi(x_1, \ldots, x_k)))$ is unsatisfiable. The latter encodes that all objective values in the optimal interval ψ_1 (as described in Section 4) admit a solution to φ, and $u \in \mathbb{Q}$ is an upper bound derived from ψ_1.
- If $O = -\infty$, verify that for all objective values below a threshold $u \in \mathbb{Q}$ (again derived from ψ_1), there is a solution of φ, namely by checking that $\exists t.(t < u \wedge \forall x_1, \ldots, x_k.(p(x_1, \ldots, x_k) = t \rightarrow \neg\varphi(x_1, \ldots, x_k)))$ is unsatisfiable.

In the first two cases, m might instead be *irrational*, which means that e.g. $p < m$ would not be a syntactically valid constraint. In that case, m is an *algebraic* number, which is in practice stored as the unique root of a univariate polynomial $q \in \mathbb{Q}[z]$ contained in an interval $I = (I.l, I.u)$ with rational bounds. Thus, we encode m by the formula $q(z) = 0 \wedge I.l < z \wedge z < I.u$ and replace m by the new variable z in the above formulas.

We checked all generated validation formulas with z3 and YicesQS with a time limit of 3 minutes and there were only 267 on which both tools timed out. For all others, the correctness of our results were confirmed by at least one.

6 Conclusions

In this paper, we presented a natural extension of the cylindrical algebraic covering method to tackle optimization problems. Our approach is complete and provides useful models or, for infima and unbounded problems, model templates from which nearly-optimal models can be computed easily. Provided a good implementation of CAlC, our method is easy to implement and outperforms other tools on benchmarks generated from SMT-LIB.

For future work, it would be interesting to see how well our approach can handle real-world applications. While it performed well on our generated benchmarks, which do at least partially stem from SMT applications, we see on the OMS set that complete approaches can quickly reach their limits. Furthermore, investigating variable ordering heuristics further might pay off, since a simple change (yielding `CAlC-Opt+`) already improved the performance. Finally, one could combine our method with e.g. incremental linearization, as we saw that these techniques complement each other quite well.

Data Availability. Our implementation, results, and tools for generating the benchmarks are available at https://doi.org/10.5281/zenodo.15526951.

Acknowledgments. V. Promies was supported by the DFG project SMT-ART (AB 461/9-1). Computing resources were granted under the RWTH project rwth1560.

References

1. International Satisfiability Modulo Theories Competition (SMT-COMP). https://smt-comp.github.io/, accessed: 2025-04-14
2. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. JLAMP **119**, 100633 (2021). https://doi.org/10.1016/J.JLAMP.2020.100633, https://doi.org/10.1016/j.jlamp.2020.100633
3. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard – version 2.0. In: Proc. of SMT '10 (2010), http://theory.stanford.edu/~barrett/pubs/BST10.pdf
4. Bigarella, F., Cimatti, A., Griggio, A., Irfan, A., Jonás, M., Roveri, M., Sebastiani, R., Trentin, P.: Optimization modulo non-linear arithmetic via incremental linearization. In: Konev, B., Reger, G. (eds.) Proc. of FroCoS'21. LNCS, vol. 12941, pp. 213–231. Springer (2021). https://doi.org/10.1007/978-3-030-86205-3_12
5. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975. LNCS, vol. 33, pp. 134–183. Springer (1975). https://doi.org/10.1007/3-540-07407-4_17
6. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In: Proc. of SAT'15. LNCS, vol. 9340, pp. 360–368. Springer (2015). https://doi.org/10.1007/978-3-319-24318-4_26
7. Dolzmann, A., Seidl, A., Sturm, T.: Efficient projection orders for CAD. In: Gutierrez, J. (ed.) ISSAC'04, Santander, Spain, July 4-7, 2004, Proceedings. pp. 111–118. ACM (2004). https://doi.org/10.1145/1005285.1005303, https://doi.org/10.1145/1005285.1005303
8. England, M., Florescu, D.: Comparing machine learning models to choose the variable ordering for cylindrical algebraic decomposition. In: Kaliszyk, C., Brady, E.C., Kohlhase, A., Coen, C.S. (eds.) CICM'19, Prague, Czech Republic, July 8-12, 2019, Proceedings. LNCS, vol. 11617, pp. 93–108. Springer (2019). https://doi.org/10.1007/978-3-030-23250-4_7, https://doi.org/10.1007/978-3-030-23250-4_7

9. Jia, F., Dong, Y., Han, R., Huang, P., Liu, M., Ma, F., Zhang, J.: A complete algorithm for optimization modulo nonlinear real arithmetic. In: Proc. of AAAI'25. pp. 11255–11263 (2025). https://doi.org/10.1609/aaai.v39i11.33224
10. Jovanovic, D., de Moura, L.M.: Solving non-linear arithmetic. In: Proc. of IJCAR'12. LNCS, vol. 7364, pp. 339–354. Springer (2012). https://doi.org/10.1007/978-3-642-31365-3_27
11. Nalbach, J., Kremer, G.: Extensions of the cylindrical algebraic covering method for quantifiers. CoRR **abs/2411.03070** (2024). https://doi.org/10.48550/ARXIV.2411.03070, https://doi.org/10.48550/arXiv.2411.03070
12. Nalbach, J., Ábrahám, E., Specht, P., Brown, C.W., Davenport, J.H., England, M.: Levelwise construction of a single cylindrical algebraic cell. Journal of Symbolic Computation **123**, 102288 (2024). https://doi.org/10.1016/j.jsc.2023.102288
13. Sebastiani, R., Trentin, P.: Optimathsat: A tool for optimization modulo theories. J. Autom. Reason. **64**(3), 423–460 (2020). https://doi.org/10.1007/S10817-018-09508-6, https://doi.org/10.1007/s10817-018-09508-6
14. Tsiskaridze, N., Barrett, C.W., Tinelli, C.: Generalized optimization modulo theories. In: Benzmüller, C., Heule, M.J.H., Schmidt, R.A. (eds.) IJCAR'24, Nancy, France, July 3-6, 2024, Proceedings, Part I. LNCS, vol. 14739, pp. 458–479. Springer (2024). https://doi.org/10.1007/978-3-031-63498-7_27, https://doi.org/10.1007/978-3-031-63498-7_27
15. YicesQS, https://github.com/disteph/yicesQS
16. The z3 Theorem Prover, https://github.com/Z3Prover/z3