Cylindrical Algebraic Coverings for Quantifiers⋆

Gereon Kremer[1,*,†], Jasper Nalbach[2,*,†]

[1]Stanford University, Stanford, USA
[2]RWTH Aachen University, Aachen, Germany

Abstract

The *cylindrical algebraic coverings* method was originally proposed to decide the satisfiability of a set of *nonlinear real arithmetic* constraints. We reformulate and extend the cylindrical algebraic coverings method to allow for checking the validity of arbitrary nonlinear arithmetic formulas, adding support for both quantifiers and arbitrary Boolean structure. Furthermore, we also propose a variant to perform *quantifier elimination* on such formulas.

Keywords
Nonlinear Arithmetic, Cylindrical Algebraic Coverings, Quantifier Elimination

1. Introduction

Nonlinear real arithmetic is the first-order theory whose atoms are polynomial constraints over real variables. We consider three fundamental problems that deal with formulas from this theory: *satisfiability*, *validity* and *quantifier elimination*. *Satisfiability* is concerned with the existential fragment (or equivalently the quantifier-free fragment) of this theory: given a purely existentially quantified formula (or a quantifier-free formula) it decides whether an assignment to the formula's variables exists such that the formula evaluates to `True`. In contrast to this, *validity* considers fully quantified formulas and checks whether they are equivalent to `True` or `False`. Finally, *quantifier elimination* deals with formulas that have both free variables (*parameters*) and quantified variables, and constructs equivalent quantifier-free formulas over the parameters.

The *cylindrical algebraic decomposition* [1] method is the only complete procedure for solving all these questions for nonlinear real arithmetic that is used in practice, despite its doubly exponential worst-case complexity that severely limits the scalability of the method. For the satisfiability problem of conjunctions of constraints, motivated by the application in satisfiability modulo theories solving, the *cylindrical algebraic coverings* method [2] has been developed based on cylindrical algebraic decomposition. Although it retains the doubly exponential complexity, its performance is significantly better in practice [2, 3] while its implementation requires only a simple bookkeeping data structure. Furthermore, it closer resembles human reasoning and is more accessible to proof production [4, 5].

Contribution. We propose a novel reformulation and extension of the cylindrical algebraic coverings method that goes beyond the satisfiability problem of conjunctions and also allows to solve arbitrary quantified formulas as well as quantifier elimination queries. We first consider validity where all variables are explicitly quantified, either existentially or universally, in Section 3, and then expand to the *quantifier elimination* problem in Section 4.

SC^2 2022: The 7th International Workshop on Satisfiability Checking and Symbolic Computation, August 12, 2022, Haifa, Israel
⋆Jasper Nalbach was supported by the DFG RTG 2236 *UnRAVeL*.

*Corresponding author.
†These authors contributed equally.
gkremer@cs.stanford.edu (G. Kremer); nalbach@cs.rwth-aachen.de (J. Nalbach)
0000-0002-0393-5739 (G. Kremer); 0000-0002-2641-1380 (J. Nalbach)

2. Preliminaries

We assume every formula φ to be a first-order formula over nonlinear real arithmetic with polynomial constraints defined in variables $x_1, \dots, x_n \in \mathbb{R}$. Furthermore, we expect φ to be transformed to prenex normal form, i.e., consisting of a *prefix* of quantifiers and a quantifier-free formula called the *matrix* $\overline{\varphi}$:

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

If $k \neq 0$, φ has free variables that are not explicitly quantified. These can be considered to be implicitly quantified existentially, much like we do for satisfiability modulo theories queries in general. We assume that in Section 3 and actually solve $\exists x_1 \cdots \exists x_k Q_{k+1}x_{k+1} \cdots Q_n x_n.\ \overline{\varphi}(x_1 \dots x_n)$. Alternatively, these free variables can be understood as *parameters* to make the input a quantifier elimination problem, as we do in Section 4.

We use standard notation for arithmetic and assume an ordering on the variables $x_1 \prec \cdots \prec x_n$. The highest variable occurring in a polynomial or constraint is called their *main variable*. For further details, we refer to [2]. Given a (partial) sample point $s \in \mathbb{R}^k$ we denote the extension of s to $(s_1, \dots s_{k+1}) \in \mathbb{R}^{k+1}$ by $s \times s_{k+1}$ and the *(partial) evaluation up to level k* of φ or $\overline{\varphi}$ over s by $\varphi[s]$ or $\overline{\varphi}[s]$, respectively: constraints with main variable at most x_k evaluate to `True` or `False` according to standard semantics, otherwise they evaluate to `Undef` (i.e., under this partial evaluation $x_1 \cdot x_2 > 1$ evaluates to `Undef` at $x_1 = 0$); the semantics are extended for formulas accordingly.

We denote the constraints that occur in a formula φ by $\texttt{constraints}(\varphi)$. To select only those with main variable x_i, we write $\texttt{constraints}_i(\varphi)$.

Cylindrical Algebraic Coverings. We briefly present the idea behind the cylindrical algebraic coverings method for checking the existential fragment of nonlinear real arithmetic and refer to [2] for more details. The fundamental idea is to recursively construct a (partial) sample point and collect intervals that represent unsatisfiable regions above this sample point. When a sample point can not be extended because these intervals form a covering of the real line in the next dimension, the covering is projected into the previous dimension to refute the current sample point. We then backtrack and choose a different value for the sample point in the highest dimension. Eventually, either a full sample point is constructed and we return SAT, or an unsatisfiable covering is constructed in the first dimension and we return UNSAT. In contrast to cells from cylindrical algebraic decomposition, intervals do not form a decomposition as they may overlap.

The algorithm starts by constructing unsatisfiable intervals for x_1 based on univariate constraints and then tries so select a value s_1 for the variable x_1 outside of these intervals. If such a value exists, the method is called recursively with the partial sample point (s_1). After substituting $x_1 = s_1$, the constraints with main variable x_2 become univariate and thus suitable for identifying unsatisfiable intervals for x_2. This process is continued recursively until either all constraints are satisfied (and we return SAT) or for some x_i no suitable value exists. In the latter case, the set of unsatisfiable intervals covers the whole real line and forms a covering. This covering is generalized by projecting it to dimension $i - 1$. The idea is to use projection tools borrowed from cylindrical algebraic decomposition with some improvements: as we only need to characterize this covering and not a decomposition, only a subset of the full projection is needed. Using the current sample point, an interval for the variable x_{i-1} with respect to the projection result can be computed which is added to the set of unsatisfiable intervals for x_{i-1}, possibly taking part in an unsatisfiable covering for x_{i-1}. Unless we find a full satisfying sample point we eventually obtain an unsatisfiable covering for the first variable x_1 and return UNSAT.

Algebraic Intervals. We generalize intervals (over a partial sample point) by attaching algebraic information in the form of sets of polynomials whose order-invariance characterizes satisfiability-invariant regions of a formula in multidimensional space. We call them *algebraic intervals* and represent them as a tuple $I = (I_\ell, I_u, I_L, I_U, I_{P_i}, I_\perp)$. (I_ℓ, I_u) is the (numeric) interval over an $(i-1)$-dimensional sample point, I_L and I_U are sets of the polynomials with main variable x_i which vanish at $(s_1, \dots, s_{i-1}, I_\ell)$ and $(s_1, \dots, s_{i-1}, I_u)$, respectively, I_{P_i} is a set of polynomials with main variable x_i which should be order-invariant in the constructed interval and $I_\perp$ is a set of lower-level polynomials which need to be order-invariant in the underlying cell as well. See [2] for more details.

Example 1. *Consider the polynomials* $P = \{x_2+1, x_2-1, x_1^1+x_2^2-2, x_1-1\}$ *and the sample point* $s = (0,0)$*. Then the algebraic interval* $(-1, 1, \{x_2+1\}, \{x_2-1\}, \{x_2+1, x_2-1, x_1^1+x_2^2-2\}, \{x_1-1\})$ *represents the interval* $(-1,1)$ *for* x_2 *around* $x_2 = 0$ *over the partial sample point* $x_1 = 0$*. The lower and upper bounds are defined by* x_2+1 *and* x_2-1*, respectively. The polynomials in* P *are order-invariant in the represented interval, the polynomials of* P *defining zeros of* x_2 *are* $\{x_2+1, x_2-1, x_1^1+x_2^2-2\}$*, while* x_1-1 *is of lower level and thus stored separately.*

Implicants. An *implicant* ψ of a formula φ is usually understood to be a "simpler" formula that implies φ, or formally $\psi \Rightarrow \phi \wedge \texttt{constraints}(\psi) \subseteq \texttt{constraints}(\varphi)$. We adapt this concept as follows. Let $s \in \mathbb{R}^i$ be a (partial) sample point. If $\varphi[s] = \texttt{True}$, then ψ is an *implicant of* φ *with respect to* s if

$$\psi[s] = \texttt{True} \wedge (\psi \Rightarrow \varphi) \wedge \texttt{constraints}(\psi) \subseteq \texttt{constraints}_i(\varphi).$$

Otherwise, if $\varphi[s] = \texttt{False}$, then ψ is an *implicant of* φ *with respect to* s if

$$\psi[s] = \texttt{True} \wedge (\psi \Rightarrow \neg\varphi) \wedge \texttt{constraints}(\psi) \subseteq \texttt{constraints}_i(\varphi).$$

We call ψ a *prime implicant* of φ if $\texttt{constraints}(\psi)$ is minimal among all implicants of φ.

Example 2. *Let* $\varphi = x^2 > 0 \wedge (x < 2 \vee x > 4)$*. Note that* $\varphi(1) = \texttt{True}$*,* $\varphi(3) = \texttt{False}$ *and* $\varphi(0) = \texttt{False}$*.* $x^2 > 0 \wedge x < 2$ *is a prime implicant of* φ *w.r.t.* 1*.* $\neg(x < 2 \vee x > 4)$ *is a prime implicant of* φ *w.r.t.* 3*. Both* $\neg(x^2 > 0)$ *and* $\neg(x^2 > 0 \wedge x > 4)$ *are implicants of* φ *w.r.t.* 0*, but only the first is a prime implicant.*

Algorithm 1: `user_call()`

Data: Global prefix $Q_1 x_1 \cdots Q_n x_n$ and matrix $\overline{\varphi}$.
Output: Either SAT or UNSAT

$(f, O) := \texttt{recurse}(())$ // Algorithm 2
return f

Algorithm 2: `recurse(`s`)`

Data: Global prefix $Q_1 x_1 \cdots Q_n x_n$ and matrix $\overline{\varphi}$.
Input: Sample point $s = (s_1, \dots, s_{i-1}) \in \mathbb{R}^{i-1}$ such that $\overline{\varphi}[s] \neq \texttt{False}$.
Output: (SAT, I) or (UNSAT, I) where $s \times I$ can or can not be extended to a model for any $s_i \in I$. In both cases, the algebraic information attached to I describes how s can be generalized.

if $Q_i = \exists$ **then return** `exists(`s`)` // Algorithm 3
else return `forall(`s`)` // Algorithm 4

Algorithm 3: `exists`(s)

Data: Global prefix $Q_1 x_1 \cdots Q_n x_n$ and matrix $\overline{\varphi}$.
Input : Sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$ such that $\overline{\varphi}[s] \neq$ `False`.
Output: see Algorithm 2

$\mathbb{I}_{unsat} := \emptyset$ // [2, Algorithm 3]
while $\bigcup_{I \in \mathbb{I}_{unsat}} I \neq \mathbb{R}$ **do**
  $s_i :=$ `sample_outside`($\mathbb{I}_{unsat}$)
  if $\overline{\varphi}[s \times s_i] =$ `False` **then**
    $(f, O) := ($UNSAT, `get_enclosing_interval`(s, s_i)$)$ // Algorithm 5
  else if $\overline{\varphi}[s \times s_i] =$ `True` **then**
    $(f, O) := ($SAT, `get_enclosing_interval`(s, s_i)$)$ // Algorithm 5
  else it holds $i < n$
    $(f, O) :=$ `recurse`($s \times s_i$) // Algorithm 2, recursive call
  if $f =$ SAT **then**
    $R :=$ `characterize_interval`(s, O) // Algorithm 6
    $I :=$ `interval_from_characterization`($(s_1, \ldots, s_{i-2})$, s_{i-1}, R) // [2, Algorithm 5]
    return (SAT, I)
  else if $f =$ UNSAT **then**
    $\mathbb{I}_{unsat} := \mathbb{I}_{unsat} \cup \{O\}$
$R :=$ `characterize_covering`(s, $\mathbb{I}_{unsat}$) // Algorithm 7
$I :=$ `interval_from_characterization`($(s_1, \ldots, s_{i-2})$, s_{i-1}, R) // [2, Algorithm 5]
return (UNSAT, I)

3. Quantified Problems

We first describe how the cylindrical algebraic coverings method can be adapted for problems where all variables are quantified. Our presentation stays very close to [2], and we reuse utility methods when possible.

One of the most notable changes is the interface of the main method. In [2], `get_unsat_cover` always returns a witness, either for satisfiability (a *model*) or for unsatisfiability (an *unsatisfiable covering*, possibly over a partial sample point). In our counterparts Algorithms 3 and 4, we instead always return a satisfiability-invariant interval in the dimension of the caller, which provides for a common interface for both existentially and universally quantified variables. In particular, we move the computation of the characterization from the caller to the callee.

This introduces a technical problem for the first dimension ($i = 1$), as we refer to s_{i-1} in the arguments to `interval_from_characterization` which does not exist. This is to be expected, as the returned interval would live in the "zero-th dimension". To simplify the presentation, we assume that a special placeholder value is returned instead of an actual interval. Algorithm 2 only returns SAT or UNSAT and no longer exposes a model or an unsatisfiable covering. In an actual implementation, this information is easily accessible.

Algorithm 1 is the interface to the recursive Algorithm 2, calling it with an empty sample point and extracting the main return value. Algorithm 2 checks the current quantifier and calls out to Algorithm 3 or Algorithm 4 accordingly.

Algorithm 3 is mostly equivalent to [2, Algorithm 2] and incorporates the following changes: instead of returning a model, we obtain a feasible interval around the model and return the algebraic interval that can directly be used for a satisfiable covering in dimension $i - 1$; instead of computing an algebraic interval from the covering obtained from the (UNSAT) recursive call, we use the result as it is and return the appropriate algebraic interval instead of a covering.

Algorithm 4 is analogous to Algorithm 3 that is used if the current variable is universally quantified. The two procedures are almost identical: while Algorithm 3 collects unsatisfiable intervals and returns

Algorithm 4: `forall(`s`)`

Data: Global prefix $Q_1 x_1 \cdots Q_n x_n$ and matrix $\overline{\varphi}$.
Input : Sample point $s = (s_1, \dots, s_{i-1}) \in \mathbb{R}^{i-1}$ such that $\overline{\varphi}[s] \neq$ `False`.
Output : see Algorithm 2

$\mathbb{I}_{sat} := \emptyset$
while $\bigcup_{I \in \mathbb{I}_{sat}} I \neq \mathbb{R}$ **do**
  $s_i :=$ `sample_outside(`$\mathbb{I}_{sat}$`)`
  if $\overline{\varphi}[s \times s_i] =$ `False` **then**
    $(f, O) := ($UNSAT, `get_enclosing_interval(`s, s_i`)`$)$ // Algorithm 5
  else if $\overline{\varphi}[s \times s_i] =$ `True` **then**
    $(f, O) := ($SAT, `get_enclosing_interval(`s, s_i`)`$)$ // Algorithm 5
  else it holds $i < n$
    $(f, O) :=$ `recurse(`$s \times s_i$`)` // Algorithm 2, recursive call
  if $f =$ SAT **then**
    $\mathbb{I}_{sat} := \mathbb{I}_{sat} \cup \{O\}$
  else if $f =$ UNSAT **then**
    $R :=$ `characterize_interval(`s, O`)` // Algorithm 6
    $I :=$ `interval_from_characterization(`$(s_1, \dots, s_{i-2})$, s_{i-1}, R`)` // [2, Algorithm 5]
    return $($UNSAT$, I)$
$R :=$ `characterize_covering(`s, $\mathbb{I}_{sat}$`)` // Algorithm 7
$I :=$ `interval_from_characterization(`$(s_1, \dots, s_{i-2})$, s_{i-1}, R`)` // [2, Algorithm 5]
return $($SAT$, I)$

Algorithm 5: `get_enclosing_interval(`s, s_i`)`

Data: Global matrix $\overline{\varphi}$.
Input : Sample point $s \in \mathbb{R}^{i-1}$ and $s_i \in \mathbb{R}$ such that $\overline{\varphi}[s \times s_i] \in \{$`False`, `True`$\}$.
Output : A satisfiability-invariant algebraic interval I around s_i over s.

$P :=$ `implicant_polynomials(`$\overline{\varphi}$, $s \times s_i$`)`
Replace P by its irreducible factors
$I :=$ `interval_from_characterization(`s, s_i, P`)` // [2, Algorithm 5]
return I

early when it finds a satisfiable interval, Algorithm 4 collects satisfiable intervals and returns early when it finds an unsatisfiable interval. Note that we call out to `characterize_covering` for both satisfiable and unsatisfiable coverings in the very same way.

Algorithm 5 computes an interval around the given sample point that is satisfiability-invariant with respect to $\overline{\varphi}$. It first obtains the set of relevant polynomials by calling `implicant_polynomials` and then uses [2, Algorithm 5] to construct the interval that is being returned. The helper function `implicant_polynomials` is expected to return the polynomials of an implicant of $\overline{\varphi}$ with respect to $s \times s_i$. This might include polynomials with main variable x_i or lower, effectively providing for a proper characterization of the interval not only in variable x_i, but also for lower variables. Further, if $\overline{\varphi}[s \times s_i] =$ `False` and $\overline{\varphi}$ is a simple conjunction, it is easy to obtain a *prime implicant* as the negation of a single conflicting constraint in `constraints(`$\overline{\varphi}$`)`. Calling it in a loop as done in Algorithm 3 is thus a direct generalization of `get_unsat_intervals` from [2]. If $\overline{\varphi}[s \times s_i] =$ `True` and $\overline{\varphi}$ is a simple conjunction and non-redundant (i.e. no sub-formula of $\overline{\varphi}$ implies $\overline{\varphi}$), then $\overline{\varphi}$ itself is the only prime implicant.

Algorithm 6 and Algorithm 7 replace [2, Algorithm 4] and compute the characterizations for a single interval and a covering, respectively. They contain no changes, except generalizing their input and output descriptions to any coverings, either satisfiable or unsatisfiable.

Algorithm 6: `characterize_interval`$(s,\ I)$

Input : Sample point $s \in \mathbb{R}^i$ and a single interval I over s in dimension $i+1$.
Output : Polynomials $R \subseteq \mathbb{Q}[x_1, \ldots, x_i]$ characterizing a satisfiability-invariant region around s.

Extract $\ell = I_\ell, u = I_u, L = I_L, U = I_U, P_{i+1} = I_{P_{i+1}}, P_\perp = I_{P_\perp}$
$R := P_\perp \cup \mathtt{disc}(P_{i+1}) \cup \{\mathtt{required\_coefficients}(p) \mid p \in P_{i+1}\}$
$R := R \cup \{\mathtt{res}(p,\ q) \mid p \in L, q \in P_{i+1}, q(s \times \alpha) = 0 \text{ for some } \alpha \leq l\}$
$R := R \cup \{\mathtt{res}(p,\ q) \mid p \in U, q \in P_{i+1}, q(s \times \alpha) = 0 \text{ for some } \alpha \geq u\}$
Replace R by its irreducible factors
return R

Algorithm 7: `characterize_covering`$(s,\ \mathbb{I})$

Input : Sample point $s \in \mathbb{R}^i$ and a covering of algebraic intervals $\mathbb{I}$ over s in dimension $i+1$.
Output : Polynomials $R \subseteq \mathbb{Q}[x_1, \ldots, x_i]$ characterizing a satisfiability-invariant region around s.

$\mathbb{I} :=$ `compute_cover`$(\mathbb{I})$ // [2, Section 4.4.1]
$R := \bigcup_{I \in \mathbb{I}}$ `characterize_interval`$(s,\ I)$ // Algorithm 6
for $j \in \{1, \ldots, |\mathbb{I}| - 1\}$ **do**
  $R := R \cup \{\mathtt{res}(p,\ q) \mid p \in U_j, q \in L_{j+1}\}$
Replace R by its irreducible factors
return R

3.1. Notes on CAD projection

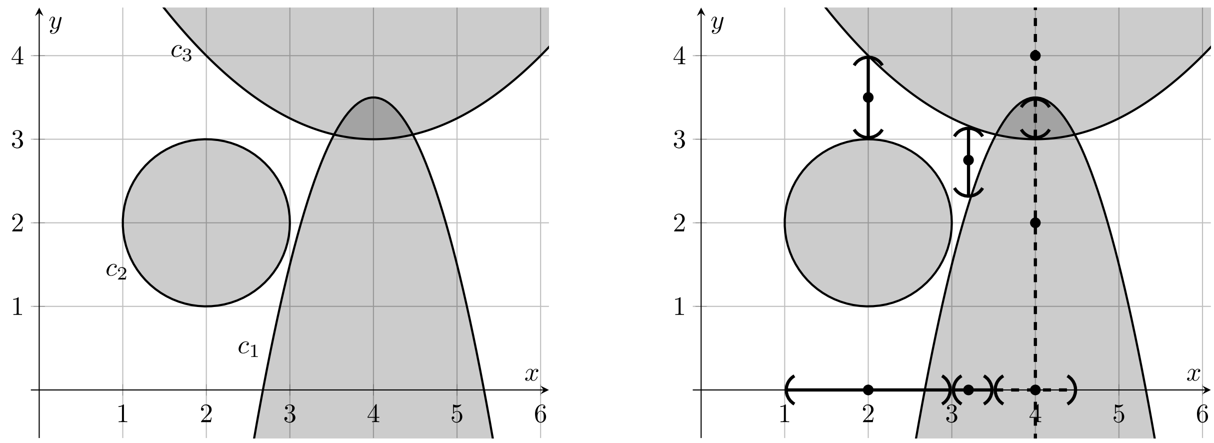
Note that we changed how we normalize the polynomial sets after projection. While [2] assumes "standard CAD simplifications", we explicitly use the set of its irreducible factors in Algorithm 5, Algorithm 6, and Algorithm 7 to satisfy the requirements of the projection operator that is used in Algorithm 6. Simply using an irreducible square-free basis, the common standard formulation for CAD projection, is not quite sufficient for cylindrical algebraic coverings: we eventually compute resultants of polynomials that come from different local projection sets, i.e. from different basis'. If carefully executed, these sets can be made "pairwise square-free", as mentioned in [6, Section 2.1]. Fully factoring all polynomials is more robust and probably even more efficient in practice, if the implementation at hand has this capability.

Depending on the implementation of the `required_coefficients()` subroutine and which semantics for the root isolation is used, the projection operator would usually be either McCallum's or Lazard's projection operator; this choice is already discussed in [2, Section 4.4.6] and possibly changes the properties of the algebraic intervals which we define based on order-invariance.

3.2. Example

As wide parts of the algorithm are taken from the cylindrical algebraic covering method, we again refer to [2] for more intuition of unsatisfiable coverings. In this example, we illustrate how both satisfiable and unsatisfiable regions are characterized for an existentially quantified variable and how coverings of satisfying regions are computed for a universally quantified variable. We consider the following formula with constraints c_1, c_2 and c_3 that are depicted in Figure 1a:

$$\varphi := \forall x_1.\ \exists x_2.\ c_1 : x_2 > 3.5 - 2(x_1-4)^2 \wedge c_2 : (x_1-2)^2 + (x_2-2)^2 - 1 > 0 \wedge c_3 : x_2 < 3 + 0.25(x_1-4)^2$$



(a) Graphs of the constraints. The gray areas depict the conflicting regions of the constraints.

(b) The satisfiable intervals are indicated with a solid line, the unsatisfiable intervals with a dashed line.

Figure 1: Illustration of the example.

We start with the first variable being universally quantified:

forall($s = ()$**)** We start covering the real line with satisfiable intervals by sampling values for x_1 (Lines 1 and 2). We then sample any value outside the excluded intervals (in this case, we can pick any value); for illustrational purposes (as for all samples in this example), we chose 2 (Line 3). As $\overline{\varphi}$ does not evaluate to a value yet, we call the algorithm with the current partial sample to handle the next variable (Line 9).

exists($s = (2)$**)** We start covering the real line with unsatisfiable intervals (Lines 1 and 2). We sample $x_2 = 3.5$ (Line 3) and find a satisfying sample. Now, we generalize to the feasible interval around $(2, 3.5)$ as depicted in Figure 1b, which is bounded from below by c_1 and from above by c_3 (Line 7). Its projection is the satisfiable interval $(1, 3)$ for x_1 that we return (Lines 11 and 12).

We store the received satisfying interval (Line 11). As there exist samples outside the set of satisfying intervals (Line 2), we pick the next value 3.2 for x_1 (Line 3):

exists($s = (3.2)$**)** We sample $x_2 = 2.75$ and find a satisfying sample. We generalize to the feasible interval bounded by c_1 and c_3. Note that in the projection of the feasible interval, we take all constraints into account (as all constraints are part of the implicant), even if they do not have a real root at $x = 3.2$ – here, the discriminant of c_2 is added to the projection ensuring that no root of c_2 crosses the feasible interval. The resulting projection is the satisfiable interval $(3, \underline{3.5})$ for x. (The underlined value is an approximation).

Similarly, the reveived interval is stored and we proceed with the sample 4 for x_1:

exists($s = (4)$**)** We sample $x_2 = 4$ (Line 3) to obtain the unsatisfiable interval $(3, \infty)$ (Line 5) which we store in the set of unsatisfying intervals (Line 15). As this set does not cover the whole real line yet (Line 2), we sample $x_2 = 2$ (Line 3) to obtain the unsatisfiable interval $(-\infty, 3.5)$ (Line 5), which is again stored (Line 15). The intervals cover the real line for x_2 (Line 2), as depicted dashed in Figure 1b. We return the unsatisfiable interval $(\underline{3.5}, \underline{4.5})$ for x_1 which is the projection of the generalization of the covering (Lines 16 and 17).

Algorithm 8: `user_call_qe()`

Data: Global prefix $Q_{k+1}x_{k+1}\cdots Q_n x_n$ and matrix $\overline{\varphi}$.
Output: A disjunction of satisfiable regions of $Q_{k+1}x_{k+1}\cdots Q_n x_n.\overline{\varphi}$.

1 **if** $k = 0$ **then return** `recurse(())` // Algorithm 2
2 **else return** `parameter(())` // Algorithm 9

Algorithm 9: `parameter(`s`)`

Data: Global prefix $Q_{k+1}x_{k+1}\cdots Q_n x_n$ and matrix $\overline{\varphi}$.
Input : Sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$ such that $\overline{\varphi}[s] \neq$ `False`.
Output: (ψ, I) where ψ characterizes all satisfying regions over s within $s \times I$.

1 $\mathbb{I} = \emptyset$
2 $\psi :=$ `False`
3 **while** $\bigcup_{I \in \mathbb{I}} I \neq \mathbb{R}$ **do**
4   $s_i :=$ `sample_outside(`$\mathbb{I}$`)`
5   **if** $\overline{\varphi}[s \times s_i] =$ `False` **then**
6     $(T, O) := ($`False`, `get_enclosing_interval(`s, s_i`)`$)$ // Algorithm 5
7   **else if** $\overline{\varphi}[s \times s_i] =$ `True` **then**
8     $(T, O) := ($`True`, `get_enclosing_interval(`s, s_i`)`$)$ // Algorithm 5
9   **else if** $i < k$ **then**
10     $(T, O) :=$ `parameter(`$s \times s_i$`)` // recursive call
11   **else** it holds $k \leq i < n$
12     $(f, O) :=$ `recurse(`$s \times s_i$`)` // Algorithm 2, recursive call
13   **if** $f =$ SAT **then** $T :=$ `True`
14   **else** $T :=$ `False`
15   $\mathbb{I} := \mathbb{I} \cup \{O\}$
16   $\psi := \psi \vee ($`indexed_root_formula(`O`)` $\wedge T)$
17 $R :=$ `characterize_covering(`s, $\mathbb{I}$`)` // Algorithm 7
18 $I :=$ `interval_from_characterization(`$(s_1, \ldots, s_{i-2})$, s_{i-1}, R`)` // [2, Algorithm 5]
19 **return** (ψ, I)

As a recursive call returned an unsatisfiable interval, the algorithm terminates here by returning UNSAT (Line 15).

4. Quantifier Elimination

For extending the method for quantifier elimination, we could follow a NuCAD [7] like approach: we could "guess" a sample point for all parameters at once, check the satisfiability of the formula using the method above and construct a cell around the sample point. We would iterate this by guessing sample points outside the already constructed cells until no such sample points exist. Finally, we would obtain a list of cells which are either satisfying or unsatisfying.

We propose an alternative approach in Algorithms 8 and 9 which builds upon the cylindrical algebraic coverings method. The idea is to consider the parameters first, and treating them similar to existentially quantified variables with a few differences: instead of returning as soon as we find a satisfiable interval, we collect both satisfiable and unsatisfiable intervals until the whole real line is covered by either satisfiable or unsatisfiable intervals and return a characterization of this covering. This ensures that all satisfiable regions of the parameter space are enumerated. Simultaneously, a symbolic description of the satisfiable regions in the parameters is constructed as a formula and returned.

For the latter, we employ the concept of *indexed root expressions* [8]: an indexed root expression is a function $\mathrm{root}[p, j] : \mathbb{R}^i \to \mathbb{R} \cup \{\text{undefined}\}$ where $p \in \mathbb{R}[x_1, \ldots, x_{i+1}]$ and $j \in \mathbb{N}_{>0}$; for all $r \in \mathbb{R}^i$, $\mathrm{root}[p, j](r)$ is the j-th real root of the univariate polynomial $p(r, x_{i+1}) \in \mathbb{R}[x_{i+1}]$

(or undefined if this root does not exist). We use constraints over indexed root expressions to describe intervals symbolically: for an algebraic interval I in main variable x_i, we define the formula $\texttt{indexed\_root\_formula}(I) = \bigwedge_{p \in I_L} \text{root}[p, j_{p,\ell}] < x_i \wedge x_i < \bigwedge_{p \in I_U} \text{root}[p, j_{p,u}]$ where $j_{p,\ell}$ and $j_{p,u}$ are chosen such that $\text{root}[p, j_{p,\ell}](s_1, \ldots, s_{i-1}) = I_\ell$ and $\text{root}[p, j_{p,u}](s_1, \ldots, s_{i-1}) = I_u$, respectively. While indexed root expressions are an extension to regular nonlinear real arithmetic, equivalent "pure" nonlinear real arithmetic formulas can be constructed with reasonable effort [8].

5. Conclusion

We have proposed an extension of the cylindrical algebraic coverings approach that is suitable to solve the validity problem for quantified formulas as well as quantifier elimination queries for partially quantified formulas. This significantly extends the applicability of the cylindrical algebraic coverings method to problems that were reserved to regular cylindrical algebraic decomposition so far. At the same time it is backwards compatible in the sense that it can produce the same results as the original method from [2], given that appropriate heuristics are used.

We look forward to see how implementations of this approach fare in practice compared to the techniques for these problems that are in use today. Given the pleasant practical experience with cylindrical algebraic coverings in solving regular satisfiability modulo theories queries – one of the reasons cvc5 won on the QF_NRA logic in the SMT-COMP 2021 over alternative approaches – we are optimistic it can bring significant improvements, all while still allowing for a fairly easy implementation.

References

[1] G. E. Collins, Quantifier elimination for real closed fields by cylindrical algebraic decomposition, in: Automata Theory and Formal Languages, volume 33 of *LNCS*, Springer, 1975, pp. 134–183. doi:10.1007/3-540-07407-4_17.

[2] E. Ábrahám, J. H. Davenport, M. England, G. Kremer, Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings, Journal of Logical and Algebraic Methods in Programming 119 (2021). doi:10.1016/j.jlamp.2020.100633.

[3] G. Kremer, E. Ábrahám, M. England, J. H. Davenport, On the implementation of cylindrical algebraic coverings for satisfiability modulo theories solving, in: Symbolic and Numeric Algorithms for Scientific Computing, 2021. doi:10.1109/SYNASC54541.2021.00018.

[4] E. Abrahám, J. H. Davenport, M. England, G. Kremer, Proving unsat in SMT: The case of quantifier free non-linear real arithmetic, arXiv preprint arXiv:2108.05320 (2021). doi:10.48550/arXiv.2108.05320.

[5] E. Ábrahám, J. H. Davenport, M. England, G. Kremer, Z. Tonks, New opportunities for the formal proof of computational real geometry?, in: Practical Aspects of Automated Reasoning and Satisfiability Checking and Symbolic Computation Workshop, volume 2752 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 178–188. URL: http://ceur-ws.org/Vol-2752/paper13.pdf.

[6] G. Kremer, A. Reynolds, C. Barrett, C. Tinelli, Cooperating techniques for solving nonlinear real arithmetic in the cvc5 SMT solver (system description), in: Automated Reasoning, volume 13385 of *LNAI*, Springer, 2022, pp. 95–105. doi:10.1007/978-3-031-10769-6_7.

[7] C. W. Brown, Open non-uniform cylindrical algebraic decompositions, in: International Symposium on Symbolic and Algebraic Computation, 2015, pp. 85–92. doi:10.1145/2755996.2756654.

[8] C. W. Brown, Solution formula construction for truth invariant CAD's, Ph.D. thesis, University of Delaware, 1999.