Diese Arbeit wurde vorgelegt am LuFG Theorie hybrider Systeme

BACHELORARBEIT

TRAVERSAL HEURISTICS FOR CYLINDRICAL ALGEBRAIC COVERINGS

Alexej Kolbin

Communicated by
Prof. Dr. Erika Ábrahám

Prüfer:
Prof. Dr. Erika Ábrahám
Prof. Dr. Jürgen Giesel

Zusätzlicher Berater:
Jasper Nalbach

Aachen, 13.02.2025

Abstract

This thesis presents different traversal heuristics for the *cylindrical algebraic coverings* algorithm, a specialized adaption from the *cylindrical algebraic decomposition* method for deciding the satisfiability of a set of constraints, consisting of multivariate polynomials with rational coefficients over the reals (*non-linear real arithmetic*).

The focus of the heuristic will lie on avoiding irrational samples during the search, with the premise to improve the running time at the cost of a higher memory usage. Preferring rational samples could simplify further computations and reduce the number of projections needed. But to find a solution, it could require to revisit earlier cached paths with irrational samples. Furthermore, we will show the result of different heuristic settings and evaluate them at the end.

Contents

1 Introduction **7**

2 Preliminaries **9**
2.1 Problem Definition 9
2.2 Cylindrical Algebraic Decomposition 9
2.2.1 Projection and Lifting 10
2.3 Cylindrical Algebraic Covering 12
2.3.1 Algebraic Intervals 13
2.3.2 Characterization 14
2.3.3 Completeness 14
2.3.4 Example 14
2.4 Irrational Samples 16

3 Heuristic Implementation **17**
3.1 Sample Decision Method 19
3.2 Unknown Intervals 20
3.3 Termination 20
3.4 Overlapping UNSAT and UNKNOWN intervals 21
3.5 Updated example using the heuristic 23

4 Evaluation **25**
4.1 SMT-LIB Benchmark Results 25
4.1.1 Irrational real roots during search 31
4.1.2 Revisiting unknown intervals 31

5 Conclusion **35**
5.1 Future work 35

Bibliography **37**

Chapter 1

Introduction

Satisfiability modulo theories (SMT) deals with the satisfiability of mathematical formula for a variety of logic theories. It can be seen as generalization of the classical propositional satisfaction problem *SAT*. Depending on the theory, some problems are NP-hard and some are not decidable. Still, applications exist in a variety of domains. Some of these domains include chemistry [EEG+15], geometric theorem proving [Stu17], and the verification of properties for probabilistic pushdown automata [WK23].

Quantifier-free non-linear real arithmetic is the theory which will be the focus of this thesis. It is a first-order theory with multivariate polynomial constraints over real variables and rational coefficients. It has no quantifiers, or equivalently just existential quantifiers. In addition we will restrict ourselves to conjunctions of polynomial constraints. Tarski's method of *quantifier-elimination* for elementary algebra is a proof that this theory can be decided [Tar98]. First it shows that, for real-algebraic formulae with quantifiers, there exists a semantically equivalent quantifier-free formula. In addition, there exists a procedure for the quantifier-free formula to evaluate it to be true or false. Our set of polynomial constraints can be expressed and decided this way. One caveat is the feasibility of this procedure, due to the non-elementary time complexity [Fef06].

In 1975, Collins published a new method called *Cylindrical Algebraic Decomposition* with a theoretical doubly exponential upper bound complexity [Col]. The idea is to decompose the space $\mathbb{R}^n$ for n variables into disjoint sign-invariant subsets, called cells. All points in a sign-invariant cell evaluate the polynomial of a constraint to the same sign. With such a decomposition, one can sample one point from each cell and evaluate the input formula to find either a satisfying point or conclude unsatisfiability. The decomposition is achieved in two steps. It starts with the projection and ends with lifting. During the projection, one variable for a polynomial with n variables is eliminated, such that a decomposition for $\mathbb{R}^n$ can be constructed from a decomposition in $\mathbb{R}^{n-1}$. The desired property for latter cells, made possible by the projection, is called *cylindrical*. This is repeated until the space is 1-dimensional and decomposed through root isolation of the univariate polynomial. The lifting step constructs the higher dimensional cells incrementally, until the $\mathbb{R}^n$ space is decomposed.

This complete method is the foundation of several adaptations and improved versions. One of the first improvements is partial CAD, where cells are incrementally constructed and refined to avoid a full decomposition [CH91].

The idea of further improvements is to avoid constructing sign-invariant cells and decide satisfiability with more relaxed cell properties. Some examples for these adaptations are NLSAT [JDM13], NuCAD [Bro15], and *Cylindrical Algebraic Coverings* (CAlC) [ÁDEK21].

This thesis implements a heuristic for cylindrical algebraic coverings. The original paper did also focus only on quantifier-free nonlinear real arithmetic, but the algorithm got extended to alternating quantifiers and arbitrary Boolean structures [KN22].

The heuristic presented here changes the behaviour of the sample point construction and tries to prefer sample points without irrational values. The basic idea is to avoid irrational values for the construction of cells and benefit from faster computations. Depending on the problem space, irrational values are required to decide the satisfiability. In this case the heuristic will incrementally increase the amount of irrational values allowed in a sample point. The influence of different initially allowed irrational values and different incrementation strategies will also be shown here.

First, we will look at the prerequisites, the cylindrical algebraic covering method and the role of irrational samples during the algorithm in Chapter 2. Then we will present the heuristic and implementation with some examples in Chapter 3. Chapter 4 shows the performance and evaluation of our new heuristic. In the end, in Chapter 5 we conclude a result of the heuristic and discuss possible future work.

Chapter 2

Preliminaries

As mentioned, we will focus on conjunctions of polynomial constraints over the reals, also called *non-linear real arithmetic (NRA)*. An extension could be done similarly to the works in [KN22].

2.1 Problem Definition

Our formulae φ to solve for satisfiability is defined as a conjunction of multivariate-polynomial constraints P, such that $\varphi = \bigwedge_{c \in P} c$.

Definition 2.1.1 (Multivariate-Polynomial Constraints)**.** *We allow the following inequalities and equalities in our constraint set P:*

$$p(x_1, \ldots, x_n) < 0$$

$$p(x_1, \ldots, x_n) \leq 0$$

$$p(x_1, \ldots, x_n) = 0$$

$$p(x_1, \ldots, x_n) \neq 0$$

$$p(x_1, \ldots, x_n) \geq 0$$

$$p(x_1, \ldots, x_n) > 0$$

where $p \in \mathbb{Q}[x_1, \ldots, x_n]$ is a multivariate-polynomial with variables $x_1, \ldots, x_n$.

We say φ is satisfied, if there exists a witness point $s \in \mathbb{R}^n$, which can be mapped to our variables $x_1, \ldots, x_n$, such that every constraint in φ is true.

2.2 Cylindrical Algebraic Decomposition

Because the cylindrical algebraic covering (CAlC) algorithm is based upon techniques from cylindrical algebraic decomposition (CAD), we will swiftly introduce the common definitions and properties, which will help with further definitions in Section 2.3.

Definition 2.2.1 (Cell)**.** *A* cell *is a non-empty connected subset of $\mathbb{R}^n$.*

Definition 2.2.2 (*UNSAT* Cell)**.** *A cell C is called UNSAT exactly when all points in C evaluate some constraint in φ to* `False`*.*

The CAD algorithm decomposes the solution space $\mathbb{R}^n$ into finitely many disjoint sign-invariant cells, that can be sampled and checked individually for satisfiability. The complexity of the CAD algorithm is polynomial in the degree and number of polynomials, but doubly exponential in the number of variables [BPR06]. It is still a significant improvement over the first complete method by Tarski.

These cells are constructed in CAD through the projection and lifting phases and form a decomposition [CH91].

Definition 2.2.3 (Decomposition [ÁDEK21])**.** *A* decomposition *of R^n is a finite set of cells $D = \{C_1, \ldots, C_h\}$, which are pair-wise disjoint and form a partition $R^n = \bigcup_{i=1}^{h} C_i$.*

One important property, which remains in the CAlC method is the cylindrical order of cells. We will go into the difference between a cylindrical decomposition and a cylindrical covering later on.

Definition 2.2.4 (Cylindrical Decomposition [ÁDEK21])**.** *A decomposition D of $\mathbb{R}^n$ is* cylindrical *over a decomposition D' of $\mathbb{R}^m$, where $0 < m < n$, if every pair of projected cells of $\mathbb{R}^n$ onto $\mathbb{R}^m$ are either disjoint or identical.*

With the cylindrical property, the projected cells form a structure of aligned stacked cells. The underlying decomposition is defined by a set of polynomials which is computed by projections. The required data structure will be shown in Section 2.3.1

2.2.1 Projection and Lifting

We will introduce the projecting and lifting phases, due CAlCs adaption of CADs McCallum projections. Given a set of polynomials P, the McCallum projection operator $Proj(P)$ consists of the discriminants of polynomials $disc(p)$, pair-wise resultants $res(p_1, p_2)$ and leading coefficients $lcoef(p)$ [McC98].

First we take a look at the following example of polynomials $P_1 = \{p_1, p_2\}$ with variable order $x \prec y$.

$$P_1 = \{x^2 + (y^2 - 2)^2 - 2.25 \\ x - y + 2\}$$

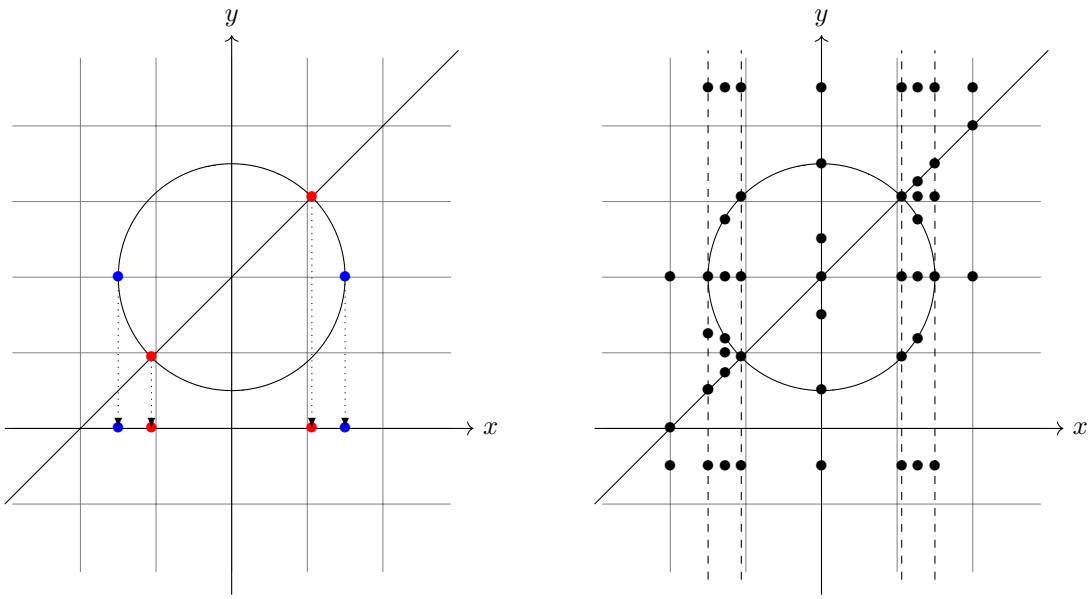
The discriminant of the circle p_1 in y is $9 - 4x^2$ with roots ± 1.5 and the roots of the resultant between both polynomials equal to $(\frac{-3}{2\sqrt{2}}, 2 - \frac{3}{2\sqrt{2}})$ and $(\frac{3}{2\sqrt{2}}, 2 + \frac{3}{2\sqrt{2}})$.

$$\begin{aligned} Proj(P_1) = \{&9 - 4x^2, 1, && disc(p_1), lcoef(p_1) \\ &1, -1, && disc(p_2), lcoef(p_2) \\ &2x^2 - 2.25 && res(p_1, p_2)\} \end{aligned}$$

The resulting polynomials with degree 0 ($disc(p_1), lcoef(p_1), lcoef(p_2)$) have no roots and can be ignored in this example. In Figure 2.1 (a) the real roots are indicated as red and blue points. It depicts the elimination of the variable y, by projecting the points to the x-axis.

The lifting phase can begin right after the first projection, because P_1 contains only two variables. Otherwise, the projection operator would be applied again on our resulting polynomial set. In traditional CAD we will take a sample of every root of $Proj(P_1)$ and every (possibly open) interval between roots. Every sample is then lifted, which means that it gets evaluated in our polynomials in P_1. Here we have to sample also within the (possibly open) intervals between the evaluated points, to get a final sample for each cell.

As seen in Fig. 2.1 (b), every black point represents the final sample of a CAD cell. Then each sample can be evaluated in the initial polynomials and checked for consistency of some input constraints. There are already 48 cells in this relatively small example, which will get exponentially worse for more variables. Improvements can be achieved, by reducing the required projected polynomials $Proj(P_1)$, which does happen in CAlC [ÁDEK21].



(a) Projection phase. The blue points indicate the roots of the discriminant. The red points show the common roots of the resultant between both polynomials.

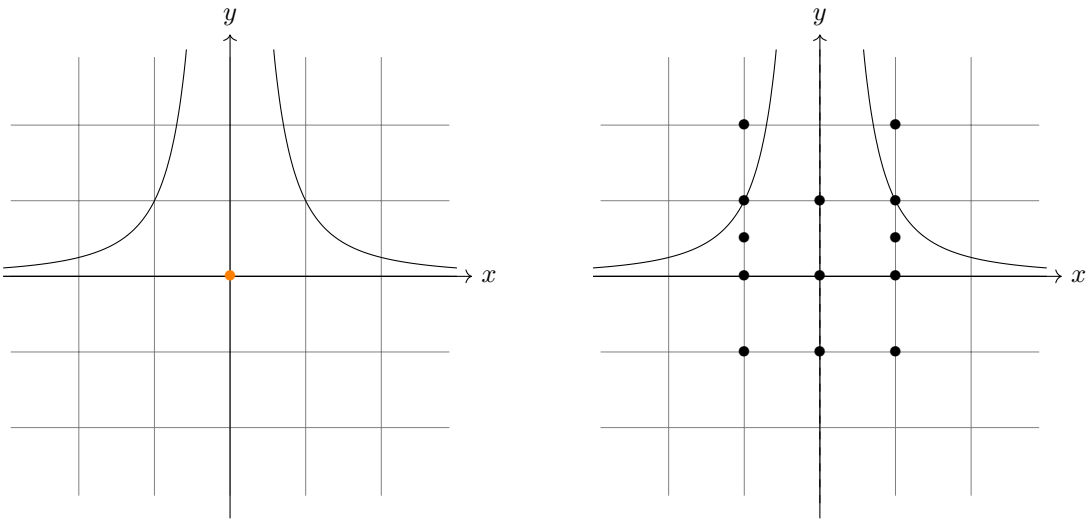
(b) Lifting phase. Dashed lines indicate the cylindrical alignment of cells. Each black dot represents a sample from a cell.

Figure 2.1: Projection and lifting phase for the polynomials in P_1

In the previous example the leading coefficients do not have a real root. Still, they are required for polynomials with asymptotes. For the second example, we look at one polynomial in $P_2 = \{p_1\} = \{y - x^2y^2 = 0\}$, which has the projection $Proj(P_2)$.

$$Proj(P_2) = \{1, x^2, \qquad disc(p_1), lcoef(p_1)\}$$

The only real root is $x = 0$ for $lcoef(p_1) = x^2$ and for the lifting the samples for x are $\{-1, 0, 1\}$, which results in 13 cells as seen in Fig. 2.1.



(a) Projection phase. The orange point indicates the root of the leading coefficient x^2.

(b) Lifting phase. Each black dot represent a cell.

Figure 2.2: Projection and lifting phase for the polynomials in P_2

2.3 Cylindrical Algebraic Covering

Cylindrical Algebraic Covering is a complete algorithm to decide the satisfiability of our *NRA* formulae. The complexity of this algorithm is not analyzed in detail, but provides improved running times on practical examples [ÁDEK21].

The cylindrical algebraic coverings method can be coarsely described as swapping the projection and lifting steps in CAD. It will incrementally construct a sample point and in case of unsatisfiability, generalizes the point to a cylindrical cell using CAD projection methods. These cells can overlap and be larger and fewer than in CAD, but still keep their cylindrical ordering.

Definition 2.3.1 (Covering [ÁDEK21])**.** *A covering of* $\mathbb{R}^n$ *is a finite set* $D = \{C_1, \ldots, C_h\}$ *of cells, where* $\mathbb{R}^n = \bigcup_{C_i \in D} C_i$ *holds. D is called UNSAT if all cells are UNSAT.*

Definition 2.3.2 (Cylindrical Covering [ÁDEK21])**.** *A covering D of* $\mathbb{R}^n$ *is* cylindrical *over a covering* D' *of* $\mathbb{R}^m$ *if every projected cell of D onto* $\mathbb{R}^m$ *is a cell of* D' *and* $0 < m < n$*.*

Due to the cylindrical ordering *UNSAT* cells can be excluded within the theory solver, unlike NLSAT which uses the Boolean solver instead [ÁDEK21]. The algorithm proceeds constructing sample points outside of these *UNSAT* cells. In case of unsatisfiability, they can cover the problem space $\mathbb{R}^n$ completely. [ÁDEK21].

The benefits of this idea are, that it is not necessary to decompose the problem space fully, like in CAD. Less and larger *UNSAT* cells could result in faster running times.

The implementation in this thesis will be based on the extended version of CAlC for arbitrary quantified formulae [KN22], although we restrict ourself just to existential

quantifiers equivalent to quantifier-free formulas.

We will explain the Algorithm 1 used for our heuristic in detail.

Generally, it tries to construct a sample point $s = (s_1, \ldots, s_n)$, by recursively choosing a value per variable. It maintains for each variable a set of unsat intervals. It starts with the empty sample point $s = ()$ and samples a value s_i outside of an existing *UNSAT* interval. We write $s \times s_i = (s_1, \ldots, s_i)$ to indicate the extension of a sample s by s_i.

Data: Quantifier-free formula φ
Input : Sample point $s = (s_1, \ldots, s_{i-1}) \in \mathbb{R}^{i-1}$.
Output: Either (SAT, I) or (UNSAT, I) where $s \times I$ can or can not be extended to a model for any $s_i \in I$

$\mathbb{I}_{unsat} := \emptyset$
while $\bigcup_{I \in \mathbb{I}} I \neq \mathbb{R}$ **do**
  $s_i :=$ `sample_outside`$(\mathbb{I}_{unsat})$
  if $\overline{\varphi}[s \times s_i] =$ `False` **then**
    $(f, O) := (\mathit{UNSAT}, $`get_enclosing_interval(s, `$s_i$`)`$)$
  else if $\overline{\varphi}[s \times s_i] =$ `True` **then**
    $(f, O) := (\mathit{SAT}, $`get_enclosing_interval(s, `$s_i$`)`$)$
  else
    (f, O) := exists$(s \times s_i)$
  if $f = \mathit{SAT}$ **then**
    return $(\mathit{SAT}, [s \times s_i, s \times s_i])$
  else if $f = \mathit{UNSAT}$ **then**
    $\mathbb{I}_{unsat} := \mathbb{I}_{unsat} \cup \{O\}$
$R :=$ `characterize_covering`$(s, \mathbb{I}_{\mathit{UNSAT}})$
$I :=$ `interval_from_characterization`$((s_1, \ldots, s_{i-2}), s_{i-1}, R)$
return $(\text{UNSAT}, \mathbb{I}_{unsat})$

Algorithm 1: `exists(`s`)` [KN22]

The sample s_i is determined by a sample heuristic in `sample_outside`, which tries to find a value outside of unsat intervals $\mathbb{I}_{\mathit{UNSAT}}$. If $s \times s_i$ violates a constraint, then a generalizing *UNSAT* interval around s_i is determined based on the polynomials of the conflicting constraints (c. `get_enclosing_intervals`). Relevant polynomials from implicants based on the conflicting constraint will be derived and used to characterize an *UNSAT* interval (c. `interval_from_characterization`). In case of conjunctions, we can simply negate the conflicting constraints and get the corresponding polynomials for characterization.

If $s \times s_i$ is a satisfying solution, the algorithm returns recursively *SAT* with the witness point s.

Otherwise the method gets called recursively with a new $s = (s_1, \ldots, s_i)$. In the case unsat intervals cover $\mathbb{R}$, the method returns *UNSAT* with an interval characterized by polynomials from these intervals (c. `characterize_covering`).

2.3.1 Algebraic Intervals

To be able to determine a generalized interval from unsat intervals, these intervals contain additional algebraic information besides the algebraic bounds (I_ℓ, I_u) and its

sample point $(s_1, \ldots, s_{i-1})$. If the bounds are defined and not $-\infty$ or $+\infty$, then they are represented by derived roots from polynomials and are, as such, symbolic.

Additionally, they contain the set of polynomials I_L, I_U, which define the bounds, such that their highest appearing variable (main variable) x_i disappears, if evaluated at the bounds. Also they contain multivariate polynomials I_P with the main variables x_i and $I_\perp$ with a main variable smaller than x_i. A characterization of a covering can be computed with this set of polynomials [ÁDEK21].

2.3.2 Characterization

A characterization is a set of polynomials for a sample $s \in \mathbb{R}^{i-1}$, which characterise why an extension $s \times s_i$ is unsatisfiable. If the extension violates directly a constraint, then the characterization consists of the negation of violated constraints.

Else if there exists no extension outside an unsat interval, then a covering characterization for s_{i-1} is determined from unsat intervals based upon the CAD projection (`characterize_covering`). Different to the classical CAD projection, the projected polynomials contain only a subset from the original projection operator required for the unsatisfiability of an interval. Fewer projections can have a large impact on the performance due to the exponential growth with each variable. Only the relevant discriminants and resultants for the unsat interval are computed. Besides leading coefficients, some lesser coefficients are also required [ÁDEK21].

The intervals for the covering characterization require a special ordering, where contained intervals have to be excluded. Otherwise they can exclude satisfiable regions (c. Section 4.5.1 [ÁDEK21]).

With these characterization polynomials a generalized unsat interval can be derived for a sample point $(s_1, \ldots, s_i)$ (c. `interval_from_characterization`). While the characterization shows similarity to the CAD projection, the interval construction can be compared to the lifting phase. The smallest possible interval around s_i is determined based on the real roots of the characterization.

Then, the underlying cell from this interval can be larger than a typical CAD cell from the sample [ÁDEK21].

2.3.3 Completeness

The reason for the completeness of CAlC is similar to CAD, in the sense that there exist finitely many combinations of constraint polynomials from which the finite amount of cells can be constructed, which leads to an UNSAT covering or a witness point. Due to the incompleteness of the McCallum projection, on which the generalization is based upon, some characterizations would be invalid. These can happen, if a polynomial gets nullified through a sample of lower dimension. But these cases can be safely detected [ÁDEK21].

2.3.4 Example

We assume variable-order $x \prec y$ and start guessing a sample initially at zero.

$$\varphi = (x^2 - 1 = 0) \wedge (y^2 + x - 1 = 0) \tag{2.1}$$

The first constraint evaluates to $-1 = 0$ and is inconsistent. We negate it into the implicant $-1 \neq 0$. This results into two "real roots" -1 and 1, which is our

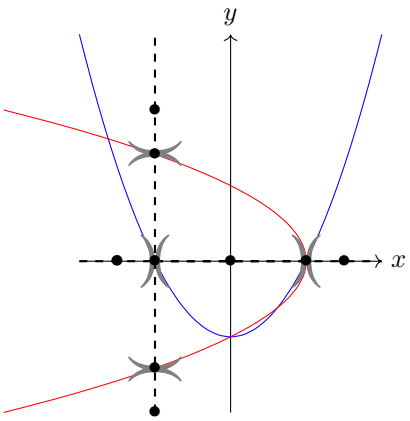


Figure 2.3: Our polynomials $x^2 - 1$ (blue) and $y^2 + x - 1$ (red) with unsat intervals (black)

characterization. For the construction of an unsat interval, the closest root below the sample $x = 0$ has to be chosen for the lower bound I_ℓ. This is $I_\ell = -1$. Similarly, the upper bound is the smallest root above the sample, which is $I_u = 1$ and results in the interval $(-1, 1)$. If our sample heuristic picks the next closest integer of our lower bound, then we get the sample $x = -2$. The same implicant characterization leads now to the unsat interval $(-\infty, -1)$. Analogously for $x = 2$ we get $(1, \infty)$. This leads us to sample $x = -1$ and after a similar process for y, we get a witness point $s = (-1, -\sqrt{2})$. As we can see in Fig. 2.3, there exists a witness point with rational values $(1, 0)$.

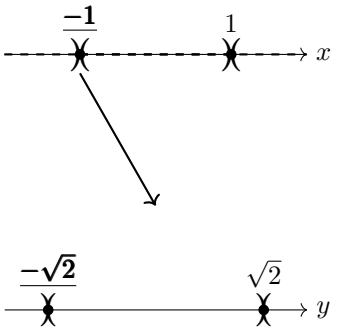


Figure 2.4: Linear search tree for are Example 2.3.4. Black intervals mark unsat intervals. The selected samples are underlined.

Search Tree

We will introduce a notation for the search tree of samples. Our heuristic will navigate through it in a different manner than the original CAlC algorithm, which follows a depth-first approach. A node in the tree resembles a sample s. We visualize the unsat intervals within the node, but do not show each derivation one-by-one. It's children

will be extensions of $s \times s_i$ with a sampled $s_i \in \mathbb{R}$. The level i of a node is the current dimension of a sample $s = (s_1, \ldots, s_i)$.

The search starts at the root with an empty sample $s = ()$. Like in Fig. 2.4 it would be a simple linear tree, but this will change with our heuristic.

2.4 Irrational Samples

Some solutions can consist of irrational sample values, which have to be represented in an exact way. An approximate representation by a close rational value would not be correct and not fit into the workings of CAlC or CAD.

The representation consists of a defining univariate polynomial with an interval, which has rational bounds. Within the interval, there is exactly one real root of the accompanying polynomial. This interval can be determined via real root isolation, which happens inside of `interval_from_characterization` (Algorithm 5 [ÁDEK21]) to determine all real roots required.

The appearance of irrational samples can happen only with weak inequalities or equalities, as they result from intersections of polynomials. The reduction of irrational samples during our search path could lead to reduced real root isolations, which saves expensive computations. Therefore, the heuristic in Chapter 3 will try to avoid them.

Chapter 3

Heuristic Implementation

The idea of the heuristic is to avoid irrational samples during the search to avoid real root isolations, which could improve the running time during characterization.

The original algorithm has a sample heuristic, which describes how to choose a sample outside from unsat intervals. It already favors rational samples over irrational ones. However we can run into a situation, where there is no other choice than an irrational sample for a certain assignment. The strategy is to try all other search paths without any irrational sample and deviate from the depth-first-search approach. This requires to fallback to other samples in previous assignments, which will be done by returning a new *UNKNOWN* flag with a corresponding interval. Unknown intervals are handled, at first, like unsat intervals, where a sample is chosen outside of both sets of intervals.

If none of the other paths lead to *SAT* or *UNSAT*, then the global result will be *UNKNOWN*. Then we retry the search and allow one additional irrational sample along a search path. Our allowed irrational samples per search will be called h_{max}. During a retry, we restore known intervals for assignments to avoid unnecessary recomputation.

The retries can be repeated, until we allow as much irrational samples as our variable count n. The result is then not necessary the same as with the original algorithm, since we will prefer visited paths with rational samples first (c. Algorithm 4). But if we start our search with $h_{max} = n$, then the algorithm should be equivalent to the original one.

Algorithm 2 is the adaptation with our new heuristic, based on Algorithm 1. Our input has three additional parameters. We get the set of unknown and unsat intervals, in case we revisit an assignment. This is also required for termination, aside from the avoidance of recomputation. The h_{max} parameter indicates how many irrational samples can be chosen.

Otherwise, the coarse flow remains close to the original. If we get an unsat covering, then we will return *UNSAT*.

Our choice for a sample s_i can have three different outcomes, as described in Section 3.1. Initially it will behave similar to the original, until we enter the first situation, where we can choose just between irrational values. In this case, we get our first unknown interval returned. We add it to the set $\mathbb{I}_{UNKNOWN}$ or update our existing unknown interval accordingly. Otherwise, if *UNKNOWN* is returned at the user call function (Algorithm 3), it needs to increase h_{max} and recall our algorithm

Input : Current sample s, initial unknown and unsat interval $\mathbb{I}_{UNSAT}, \mathbb{I}_{UNKNOWN}$ and the number of allowed irrational samples left h_{max}

Output: Either (SAT, I) or (UNSAT, I), where $s \times I$ can or can not be extended to a model for any $s_i \in I$. Additionally $(\text{UNKNOWN}, I)$ with the unknown interval I, if no solution with h_{max} irrational samples was found.

while $\bigcup_{I \in \mathbb{I}_{UNSAT}} I \neq \mathbb{R}$ **do**
  $(decision, s_i, I_{unknown}) := decide\_sample(\mathbb{I}_{UNSAT}, \mathbb{I}_{UNKNOWN}, h_{max})$
  if $decision \neq$ `RETURN_UNKNOWN` **then**
    if $\overline{\varphi}[s \times s_i] =$ `False` **then**
      $(f, O) := (UNSAT, enclosing\_interval(s))$
    else if $\overline{\varphi}[s \times s_i] =$ `True` **then**
      $(f, O) := (SAT, enclosing\_interval(s))$
    else
      $h_{next} := h_{max}$
      if $is\_irrational(s_i)$ **then**
        $h_{next} := h_{max} - 1$
      if $decision =$ `INSIDE_UNKNOWN` **then**
        $(\_, \mathbb{J}_{\text{UNSAT}}, \mathbb{J}_{\text{UNKNOWN}}, \_) := I_{unknown}$
        $(f, O) :=$ `modified_exists`$(s \times s_i, \mathbb{J}_{\text{UNSAT}}, \mathbb{J}_{\text{UNKNOWN}}, h_{next})$
      else
        $(f, O) :=$ `modified_exists`$(s \times s_i, \emptyset, \emptyset, h_{next})$
  else
    $gaps :=$ `determine_gap_intervals`$(\mathbb{I}_{UNSAT} \cup \mathbb{I}_{UNKNOWN})$
    R := `characterize_covering`$(\mathbb{I}_{UNSAT} \cup \mathbb{I}_{UNKNOWN} \cup gaps)$
    I := `interval_from_characterization`(R)
    $h_{min} :=$ `min_h`$(\mathbb{I}_{UNKNOWN})$
    return *(UNKNOWN, (I,* $\mathbb{I}_{UNSAT}$*,* $\mathbb{I}_{UNKNOWN}$*,* h_{min}*))*
  if $f = SAT$ **then**
    return *(SAT, O)*
  else if $f = UNSAT$ **then**
    $\mathbb{I}_{UNSAT} := \mathbb{I}_{UNSAT} \cup O$
    `update_unknowns`$(\mathbb{I}_{UNKNOWN}, \mathbb{I}_{UNSAT})$
  else
    if $decision =$ `INSIDE_UNKNOWN` **then**
      `// Keep/Update the new unknown interval`
      $\mathbb{I}_{UNKNOWN} := \mathbb{I}_{UNKNOWN} \setminus I_{unknown}$
    $\mathbb{I}_{UNKNOWN} := \mathbb{I}_{UNKNOWN} \cup O$
    `update_unknowns`$(\mathbb{I}_{UNKNOWN}, \mathbb{I}_{UNSAT})$
$R :=$ `characterize_covering`$(s, \mathbb{I}_{UNSAT})$
$I :=$ `interval_from_characterization`$((s_1, \ldots, s_{i-2}), s_{i-1}, R)$
return $(\text{UNSAT}, \mathbb{I}_{UNSAT})$

Algorithm 2: `modified_exists`$(s, \mathbb{I}_{UNSAT}, \mathbb{I}_{UNKNOWN}, h_{max})$

with our stored interval computations. There can be different incremental strategies for h_{max} and initial values, but we will assume in this chapter an initial value of zero and single increments. Other variants will be shown in Chapter 4.

Input : $h_{initial}$ initial amount of allowed irrational samples
Output: Either (SAT, I) or (UNSAT, I), where $s \times I$ can or can not be extended to a model for any $s_i \in I$.

$h_{next} := h_{initial}$
$\mathbb{I}_{UNSAT} = \emptyset$
$\mathbb{I}_{UNKNOWN} = \emptyset$
do
  (f, O) := `modified_exists`((), $\mathbb{I}_{UNSAT}, \mathbb{I}_{UNKNOWN}, h_{next}$)
  if f = *UNKNOWN* **then**
    (_, $\mathbb{I}_{UNSAT}$, $\mathbb{I}_{UNKNOWN}$, _) := O
    h_{next} := `increase_strategy`(h_{next})
while f = *UNKNOWN*;
return *(f, O)*

Algorithm 3: `user_call`($h_{initial}$)

3.1 Sample Decision Method

The sample decision in Algorithm 4 prefers rational values, if possible. If it has to choose an irrational sample, then it prefers samples within unknown intervals over new samples outside of unsat and unknown intervals. To sample within an unknown interval and respect our h_{max} limit, we also need to consider, whether the chosen sample is irrational, in addition to the intervals h_{min}. If both is not possible, then we cannot choose a sample with respect to our h_{max} limit and the parent method has to return *UNKNOWN* with an unknown interval.

Input : Set of intervals $\mathbb{I}_{UNSAT}$, $\mathbb{I}_{UNKNOWN}$ and the amount of irrational samples h_{max} allowed to sample.
Output: Tuple (*decision*, *sample*, $I_{unknown}$), where *decision* can either be `OUTSIDE`, `INSIDE_UNKNOWN`, `RETURN_UNKNOWN`. Only in case of `INSIDE_UNKNOWN` the $I_{unknown}$ has a value.

s_{out} := determine_sample_outside($\mathbb{I}_{UNSAT}$, $\mathbb{I}_{UNKNOWN}$)
if s_{out} *exists* ***and*** *is_irrational*(s_{out}) **then**
  return *(`OUTSIDE`, s_{out}, ())*
else
  ($s_{unknown}$, $I_{unknown}$) := determine_sample_inside($\mathbb{I}_{UNSAT}$, $\mathbb{I}_{UNKNOWN}$)
  (h_{min}, _, _, _) := $I_{unknown}$
  if $s_{unknown}$ *exists* ***and*** *((*
    is_irrational($s_{unknown}$) ***and*** $h_{max} \geq h_{min} + 1$*)* ***or***
    (!is_irrational($s_{unknown}$) ***and*** $h_{max} \geq h_{min}$*))* **then**
    $I_{unknown}$.*sample* := $s_{unknown}$
    return *(`INSIDE_UNKNOWN`, $s_{unknown}$, $I_{unknown}$)*
  else if s_{out} *exists* ***and*** $h_{max} > 0$ **then**
    return *(`OUTSIDE`, s_{out}, ())*
  else
    return *(`UNKNOWN`, (), ())*

Algorithm 4: `decide_sample`($\mathbb{I}_{UNSAT}, \mathbb{I}_{UNKNOWN}, h_{max}$)

3.2 Unknown Intervals

Unknown intervals need to store additional information besides their algebraic interval I, such as the computed unknown intervals *stored_unknown*, and unsat intervals *stored_unsat*. While sampling the chosen unknown sample *sample* $\in I$ is additionally stored.

To avoid early re-entrance and infinite loops, we also need to store the minimum required irrational samples h_{min} for search paths originating the underlying unknown cell. This can be precomputed at the return of the interval, which will also update an existing unknown interval. The computation of h_{min} has to consider whether the unknown sample *sample* is itself irrational as seen in Algorithm 5.

The characterization of an unknown interval is similar to CAlC, except that we use both unsat and unknown intervals for the covering. Due to the gaps of irrational values, which the heuristic tries to avoid, the covering is not complete. The polynomials, which define the boundaries of the gaps, are added to get a full covering.

3.3 Termination

With the introduction of unknown intervals, the algorithm could risk to repeatedly reenter the surface of nested unknown intervals.

We look at the following situation, where a solution to a problem requires two irrational values for variables x_k, x_{k+1},x_{k+2}. We allow one irrational sample $h_{max} = 1$ and assume to sample a rational value for x_k within an existing unknown interval.

Our next sample x_{k+1} has to be irrational and h_{max} becomes zero.

Further on, we assume, x_{k+2} also has to be irrational, but due to $h_{max} = 0$, we must return *UNKNOWN* to x_{k+1}, which also returns *UNKNOWN* to x_k.

If h_{min} would still be 1 for x_k, then we would resample from the interval and loop infinitely. Therefore unknown intervals need to accumulate the required irrational samples from nested unknown intervals into h_{min} as seen in Algorithm 5.

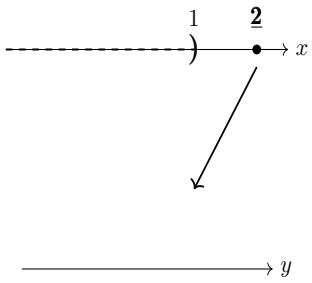
An example of this situation with variable order $x \prec y \prec z$ is given with the following constraints:

$$P_3 = \{x > 1, y^2 - x = 0, z^2 - y = 0\}.$$

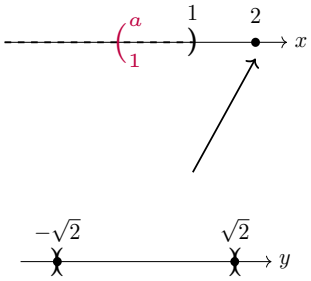
The first *UNKNOWN* result does not cause a problem and returns an unknown interval a, which forms with the unsat interval a covering at the lowest level. The *user_call* function has to increase h_{max} by one and the unknown interval can be sampled (c. Section 3.3).

After choosing $y = -\sqrt{2}$, which leads to an unsat interval $(-\infty, 0)$, the last possible sample for y is selected (c. Fig. 3.2). The result is a nested unknown interval b with $h^b_{min} = 1$. According to the sample procedure, the irrational sample $y = -\sqrt{2}$ with h^b_{min} would violate our $h_{max} = 1$ constraint and another *UNKNOWN* result is returned.

Unknown interval a is updated to $h^a_{min} = 2$, and again, Algorithm 4 returns *UNKNOWN*. The result arrives at the *user_call* function, which increases $h_{max} = 2$, and recalls our algorithm. A solution can be found with two irrational samples and the program terminates.



(a) Expansion to a new sample $x = 2$



(b) *UNKNOWN* result due to missing rational samples for y. The new unknown interval a with bounds $(0, \infty)$ has its $h_{min} = 1$, which is indicated by the subscript.

Figure 3.1: Search tree on first *UNKNOWN* result for P_3 with $h_{max} = 0$.

Input : Set of unknown intervals $\mathbb{I}_{UNKNOWN}$
Output: An integer, which indicates the minimum amount of required irrational samples upon entering an unknown interval in $\mathbb{I}_{UNKNOWN}$

```
59 if 𝕀_UNKNOWN = ∅ then
60 |  return 1
61 else
62 |  heuristic := ∞
   |   for I in 𝕀_UNKNOWN do
63 |  |  (new_h, _, _, _) := I
64 |  |  if is_irrational(I.sample) then
65 |  |  |  new_h := new_h + 1
66 |  |  if new_h < heuristic then
67 |  |  |  heuristic := new_h
68 |  return heuristic
```

Algorithm 5: `min_h` ($\mathbb{I}_{UNKNOWN}$)

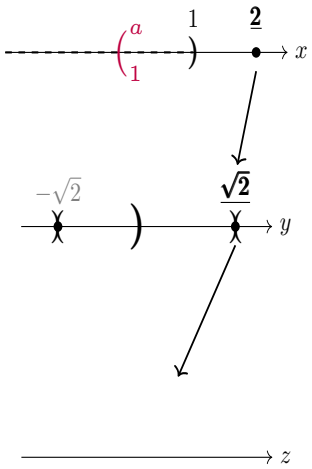
3.4 Overlapping UNSAT and UNKNOWN intervals

As seen in Fig. 3.2 there is an overlap between an unsat and unknown interval a. This did not cause any problems in the given example, but it could happen that an unknown sample gets invalidated within an unsat interval.

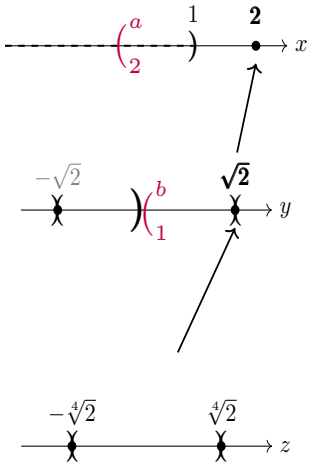
The implementation calls *update_unknowns* after each change in unsat/unknown intervals to check this case.

If an unknown interval is a subset of an unsat interval, then it gets removed, because our unknown cell is part of a known unsat cell.

If an unknown interval overlaps an unsat interval and was sampled with a value which lies now within an unsat interval, then we need to discard our stored intervals and recompute h_{min} recursively, as it could have changed.



(a) After $y = -\sqrt{2}$ leads to an unsat interval $y = \sqrt{2}$ is expanded.



(b) *UNKNOWN* result due to missing rational samples for z with a new unknown interval b with bounds $(0, \infty)$ and $h^b_{min} = 2$. Leads to another *UNKNOWN* result with updated $h^a_{min} = 2$.

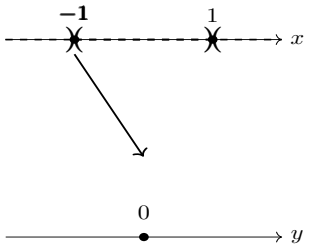
Figure 3.2: Search tree with nested unknown intervals for P_3 with $h_{max} = 1$.

3.5 Updated example using the heuristic

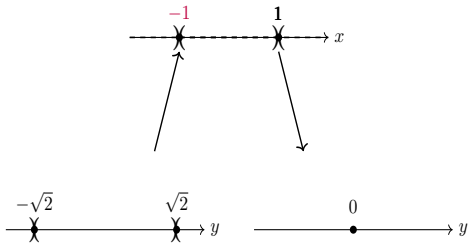
We take a look at our previous Example 2.3.4 and the changes with our heuristic.

It is possible to find a solution with only rational samples instead of irrational ones, where we choose another sample for x (c. Fig. 3.4). Our variable order is still $x \prec y$ and our sample heuristic chooses the smaller value, if possible.

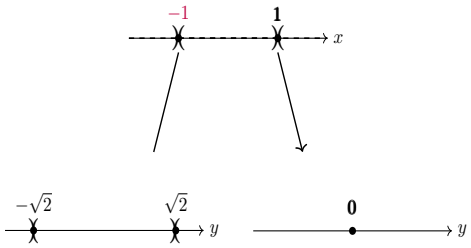
For the first variable x, we still get three unsat intervals, which leave two common boundaries -1 and 1. If $x = -1$ is chosen by the sample heuristic, the second constraint will be equivalent to $y^2 = 2$, which has only irrational solutions $\pm\sqrt{2}$, as seen in Fig. 3.3. In this case we get the unknown interval returned $(-\infty, 1)$. This leads to the only possible assignment $x = 1$ and a rational solution for $y = 0$.



(a) We choose the same sample as the original algorithm.



(b) Upon discovery of irrational samples *UNKNOWN* is returned and $x = 1$ gets chosen.



(c) We find a rational solution at cost of a search in breadth.

Figure 3.3: Three states of the search tree in chronological order, which is now affected by the new heuristic.

If such situations occur early in our search, then the further search could benefit from having computationally cheaper projections with less irrational numbers.

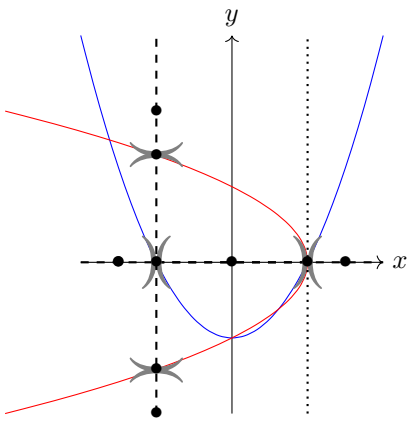


Figure 3.4: The dashed line indicates the original sample choice and the dotted line the new choice of the heuristic with a rational solution.

Chapter 4

Evaluation

We will compare the performance of our heuristic with the original CAlC algorithm. The implementation is incorporated in the SMT solver SMT-RAT [CKJ+15], based on the `CoveringNG` implementation.

Multiple variations of our heuristic will be evaluated. It can be adjusted via two parameters. First, the initial number of allowed irrational samples $h_{initial}$ for the start value of h_{max}. Our initial values can be either *zero*, one *quarter* of the variable count, or the *full* sample can consist of irrational values.

Secondly, the increment strategy h_{inc}, which specifies how many irrational values should be allowed after one global *UNKNOWN* result. It can be incremental *inc*, which allows one additional irrational value during the next search. Alternatively it can be *instant*, which allows every irrational value again.

The idea of the *quarter* configuration is to start the breadth-first search further away from the root with the goal to search within a smaller breadth. The average ratio of irrational values in a solution respective to the number of variables in the original algorithm is approximately one half. One quarter was chosen to avoid being too close to the original traversal behaviour.

We will use the results of the original algorithm as a baseline, by setting our parameters to $h_{initial} = full$ and $h_{inc} = instant$. If we allow initially a full assignment of irrational samples, then the search does not backtrack through unknown cells, and behaves like the original CAlC algorithm.

Generally, we would expect for *zero-inc* the fewest irrational values occurring during the search and within witness points for satisfying problems and the largest amount for *full-instant*.

4.1 SMT-LIB Benchmark Results

We will use the latest SMT-LIB benchmark [PSB+24] to assess our performance. The benchmarks were performed on the CLAIX-2023 cluster of the RWTH Aachen University. A process timeout of thirty seconds and limit of 4GB memory was imposed for all variants.

As we can see in Fig. 4.1, our heuristic has in total less solved instances than the original algorithm. The number of solved problems increases linearly with the amount of initially allowed irrational values and faster increment strategies. The running time does also not significantly differ. The overall decline for *zero-inc* the given benchmark

$h_{initial}$	h_{inc}	SAT		$UNSAT$		$Total$	
zero	*inc*	5070	0.18	4939	0.50	10009	82.35%
zero	*instant*	5103	0.22	4944	0.52	10047	82.66%
quarter	*inc*	5113	0.19	4949	0.53	10062	82.8%
quarter	*instant*	5113	0.20	4950	0.53	10063	82.8%
full	*instant*	5124	0.20	4952	0.55	10076	82.9%

Figure 4.1: Amount of solved problem instances for the SMT-LIB quantifier-free non-linear-arithmetic benchmark on CLAIX-2023 (30s timeout, 4GB memory, per instance)

affects 70 instances out of the total set of around 12000. The *quarter* configurations perform just marginally worse compared to the original.

First we will compare the occurrences of different irrational values for each configurations. In Fig. 4.4 and Fig. 4.3 we can observe the amount of irrational values within witness points on SAT instances for each configuration and the occurrences of irrational real roots.

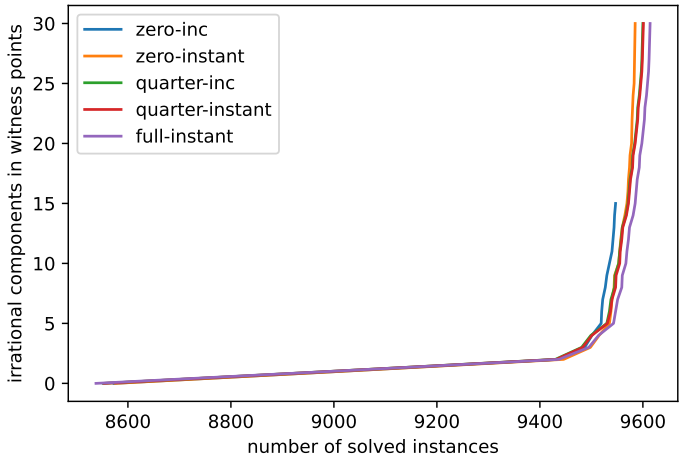


Figure 4.2: Irrational components in witness points for SAT instances.

We can see the similar behaviour of *quarter* configurations to the original in both metrics. It is notable that the *zero-instant* and the *quarter* configurations solve more instances in time with the similar amount of irrational real roots compared to *zero-inc*. This indicates that the strict breadth-first-search for rationals by *zero-inc* leads in total to more overhead than the hybrid approach of switching back to a depth-first-search later on.

The *zero-inc* configuration finds only at most 15 irrational components within witnesspoints for most problem instances compared to up to 30 irrational components in other configurations. One explanation could be that the heuristic prefers rational samples and in most cases, a solution with fewer irrational values can be found.

This can be seen in Fig. 4.4 (a), where each point indicates a solved SAT problem

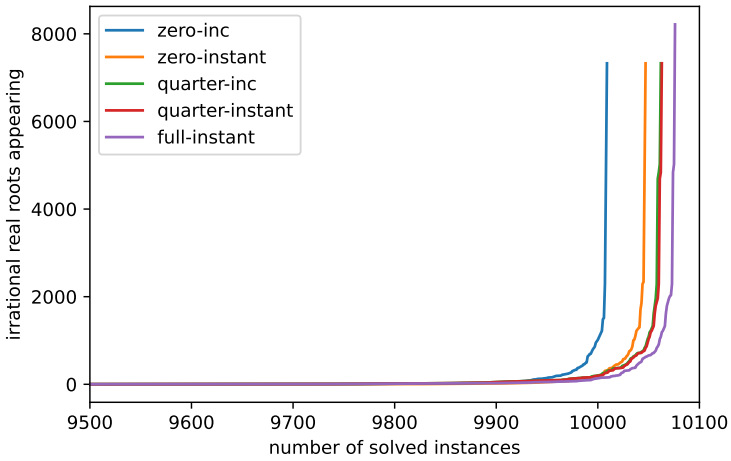


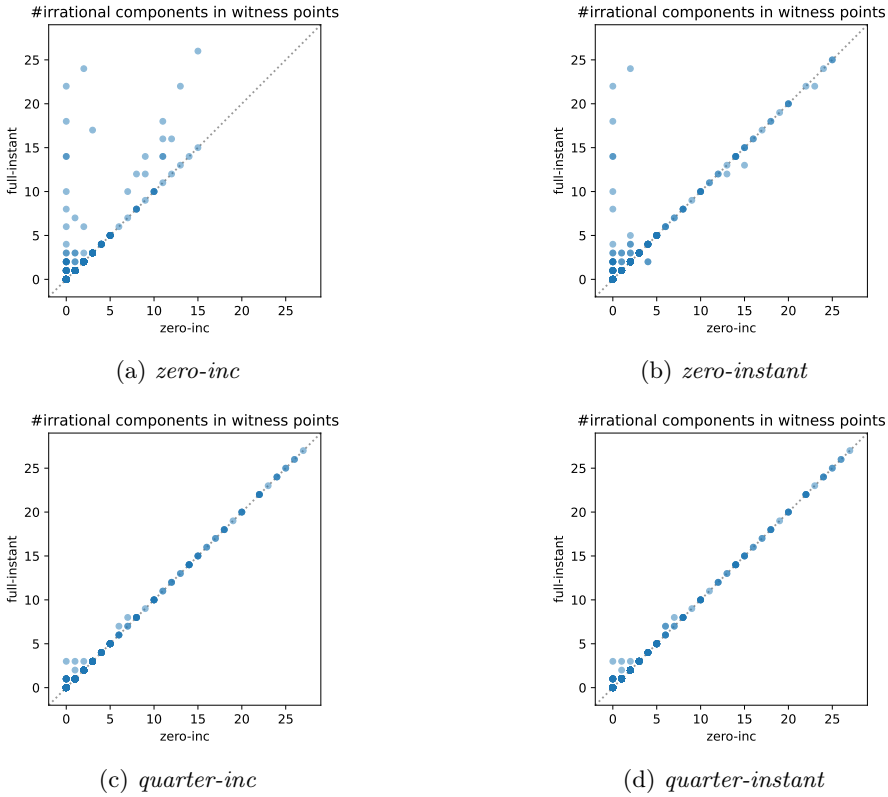
Figure 4.3: Occurrence of irrational real roots.

by the original and a heuristic configuration, and is positioned based on the amount of irrational values for each witness point. The diagonal line indicates instances with the same amount of irrational values. Points above the diagonal line indicate, that the configuration has found a solution with less irrational values than the original configuration. For the *zero-inc* configuration, there is no instance, where the original configuration has a solution with less irrational values, such that there are no points below the diagonal.

This is not the case for *zero-instant*, which has few of these instances below the diagonal line. Even though all restrictions on irrational values are removed after one global *UNKNOWN* result, our sample decision method prefers unknown intervals over the first possible irrational sample, which leads to a different traversal than the original algorithm and can possibly result in a witness point with more irrational values.

The *quarter* configurations show approximately a diagonal line, which indicates close results to the original behaviour. The increment strategy seems to have no significant impact here. Only a few instances with a low amount of irrational values in *full-instant* witness points, have less irrational values with the heuristic. The minimal amount of irrational values in any witness point in larger problem instances is likely less than a quarter of number of variables.

An additional explanation, why the *zero-inc* does not find more than 15 irrational values could be that, if an instance requires at least 16 irrational values, then the heuristic configuration will likely time out beforehand, due to the additional time needed for the breadth-search with 15 levels. Such breadth-searches can result in a multitude of formula evaluations, which happen each time a sample is extended, and therefore implicate the number of samples during the search. This can be seen in Fig. 4.5, where timed out instances by the heuristic configuration and solved by the original are indicated as red points. Instances the other way around are marked green and instances which are solved by both configurations are blue.



(a) *zero-inc* (b) *zero-instant*

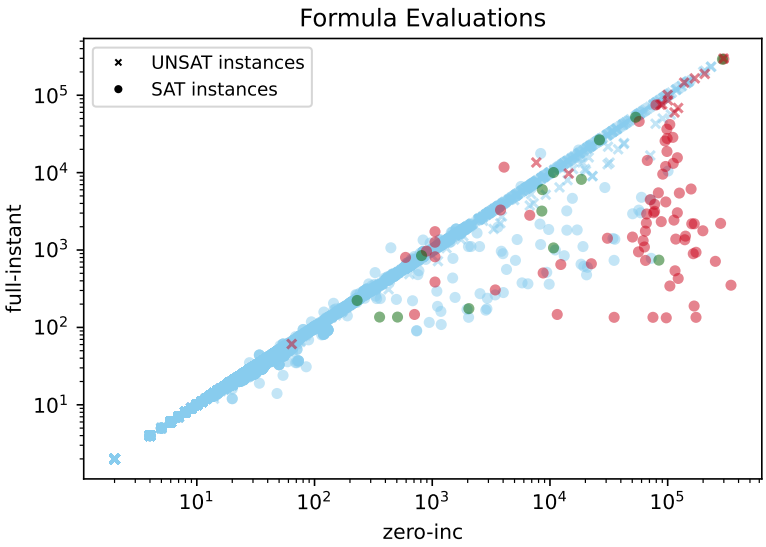
(c) *quarter-inc* (d) *quarter-instant*

Figure 4.4: Total number of irrational values in a witness point for *SAT* instances for each heuristic configuration compared with the original algorithm.

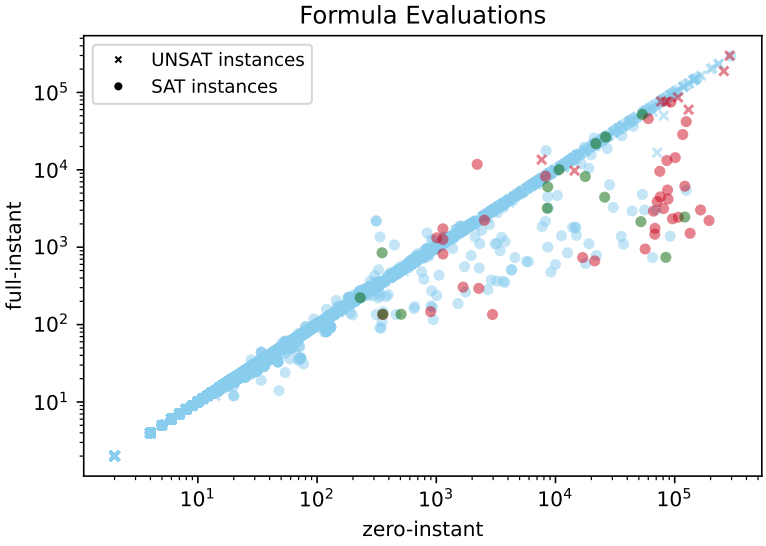
For *zero-inc* a red cluster is recognizable beneath the diagonal line, which shows, that the heuristic has significantly more formula evaluations than the original. The cluster appears to form only at a certain larger amount of formula evaluations and consists mostly of unsolved *SAT* instances. For configurations with less strict initial values and increment strategies the red clusters get smaller and the plot approximates a diagonal line, which indicate an approximation to a similar behaviour of the original algorithm. The unsolved *UNSAT* instances by the heuristic appear close to the diagonal. The original algorithm solves them also close to the timeout, as seen in Fig. 4.7, such that these instances can be explained as a combination of noise regarding the timeout measurement and slight search overhead.

Regarding memory usage (c. Fig. 4.6) the heuristic uses, as expected, in most cases more memory than the original algorithm. Especially timed out *SAT* instances, which are solved in time by the original algorithm, show a higher usage, which suggest a higher amount of samples and stored unknown intervals during the search. For the few cases, where a witness point for *SAT* instances is found quicker, the memory usage is also lower.

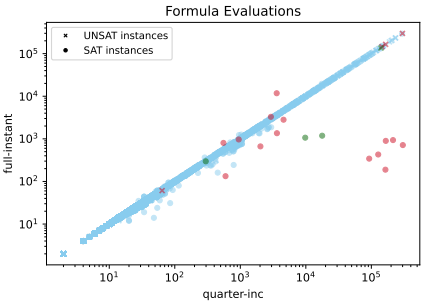
Inspecting the running times, we have some noise regarding precise time measurement, especially for low values, which shows a large cluster in the lower left area. There are some outliers with worse running times with our heuristic, especially for *SAT* instances.



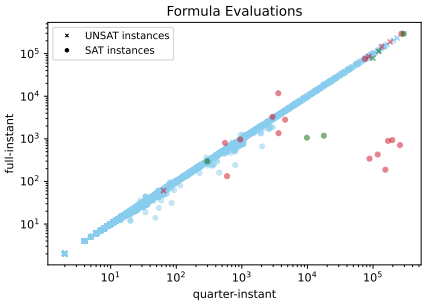
(a) *zero-inc*



(b) *zero-instant*



(c) *quarter-inc*



(d) *quarter-instant*

Figure 4.5: Total number of formula evaluations per instance for each heuristic configuration compared against the original algorithm.

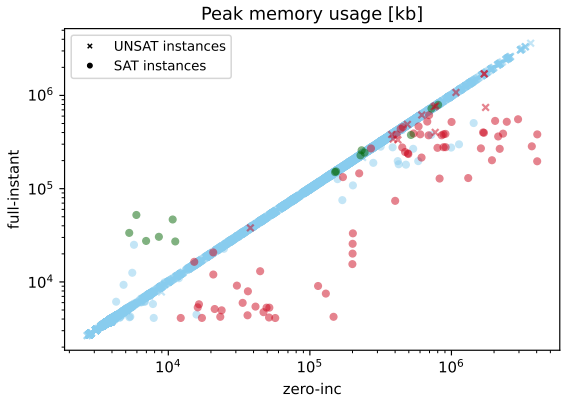
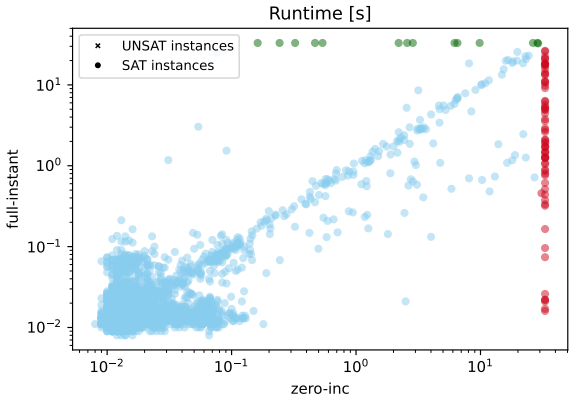
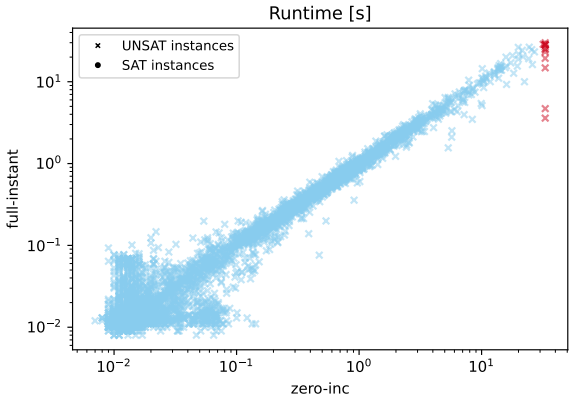


Figure 4.6: Peak memory usage per instance for *zero-inc* and *full-instant*.



(a) *SAT* instances



(b) *UNSAT* instances

Figure 4.7: Running times per instance for *zero-inc* in comparision to *full-instant*.

4.1.1 Irrational real roots during search

We have seen that for *SAT* instances the heuristic has an effect on the specific result with a visible reduction in irrationals, but it is not clear whether the heuristic has a significant impact during the search regarding the avoidance of irrational real roots. If we look at Fig. 4.8 for the appearance of real roots during the search, then we can not see any pattern or cluster, but rather scattered instances. The heuristic produces a higher variance in occurred irrational real roots compared to the original. But both the increased and decreased amount in irrational real roots correlate with a time out by the heuristic. In both cases it seems to be, that the increased need for formula evaluations is the bottleneck for our heuristic, which can not be compensated with less irrational samples. Even for timed out *SAT* instances by the original our heuristic has more irrational real roots during the search and just seems to be fortunate with the search path. For *UNSAT* instances the heuristic has on average slightly more irrational real roots occurring. Like before, *quarter* configurations show a lower variance and show in most cases a similar amount of irrational real roots. In instances with fewer irrational values in witness points with *zero-inc*, the heuristic did also encounter more irrational real roots than the original algorithm.

zero-instant has in some cases fewer irrational real roots than *zero-inc*, because it tries not to search all paths with the least amount of irrational samples, but rather prefers to search within a smaller subset of these paths. This narrowed search seems to reduce the total amount of irrationals more effectively.

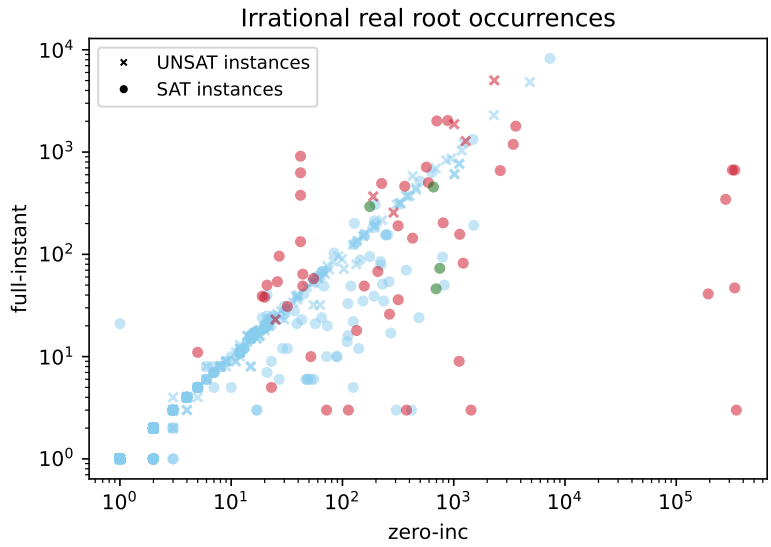
But generally, the extended search done by the heuristic does not consistently lead to less irrational samples.

4.1.2 Revisiting unknown intervals

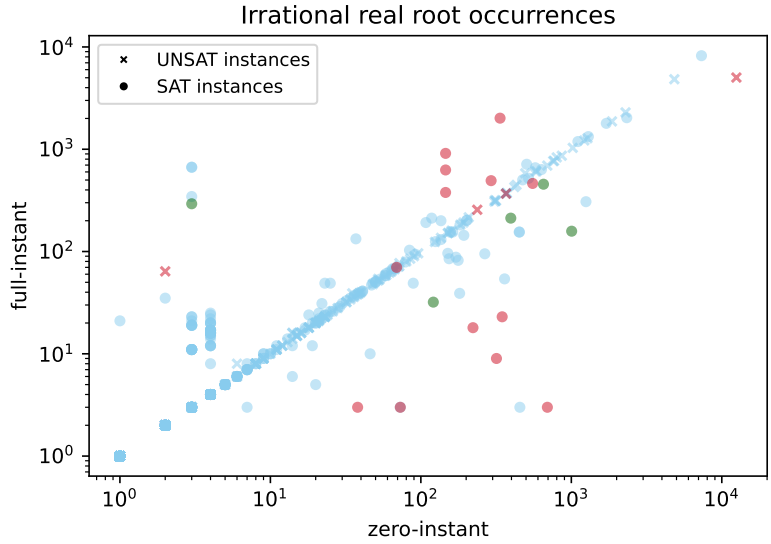
Ideally the heuristic would not have to revisit unknown intervals, if possible. Otherwise it could have directly searched further during the first visit. This is for *SAT* instances often not the case, where around half the times in all configurations an unknown interval has to be revisited.

In Fig. 4.9 we can see for *zero-instant*, that after the first global unknown result, there are less unknown intervals and the revisit ratio is smaller. Also for *UNSAT* instances there happen to be fewer revisits on average than for *SAT* instances. An interesting observation is the timeout behaviour for *zero-inc* and *zero-instant*, where no unknown interval is revisited on average. This hints at a large search space for rational samples, which results in a lot of computation time. For *quarter* configurations unknown intervals are often revisited on timed out instances.

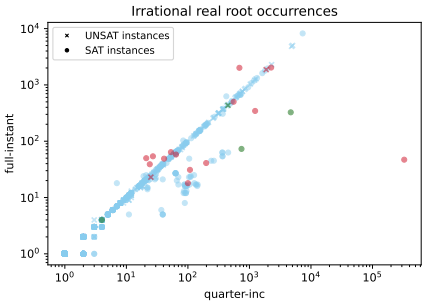
In most instances *UNKNOWN* intervals do not overlap with *UNSAT* intervals and within the around 300 problem instances, they show no strong correlation with the result of our heuristic.



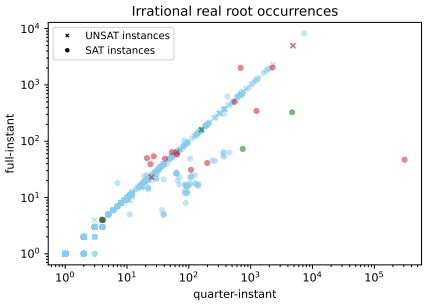
(a) *zero-inc*



(b) *zero-instant*

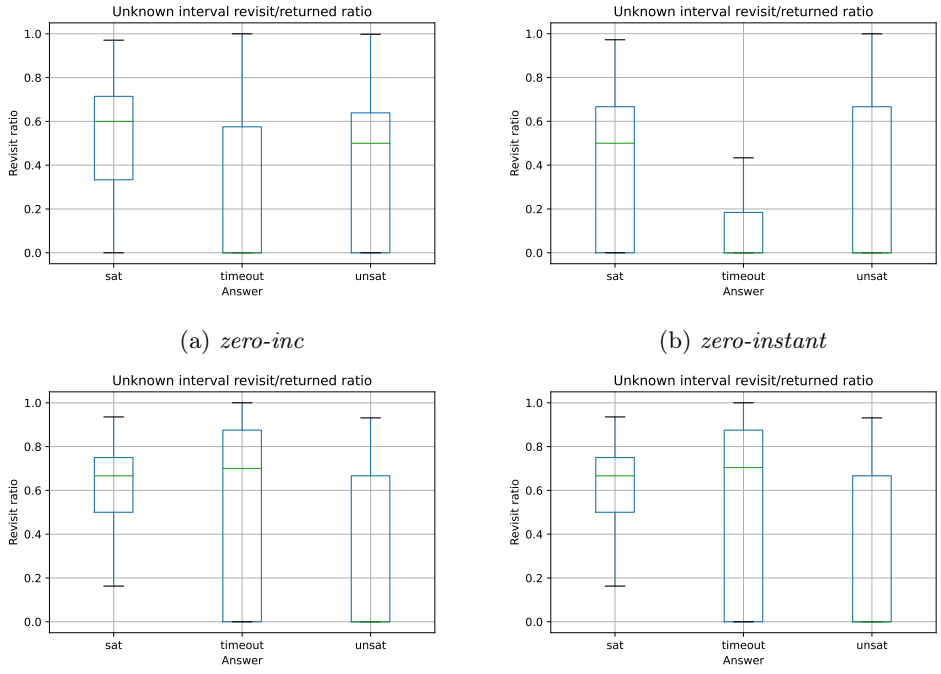


(c) *quarter-inc*



(d) *quarter-instant*

Figure 4.8: Total number of irrational real roots appearing for each configuration in comparision with *full-instant*.



(a) *zero-inc*

(b) *zero-instant*

(c) *quarter-inc*

(d) *quarter-instant*

Figure 4.9: Unknown intervals revisited in relation to unknown intervals returned and grouped by the answers of each heuristic configuration.

Chapter 5

Conclusion

The premise of the heuristic to avoid computationally expensive irrational real roots and adjusting the traversal strategy to search in breadth, fails due to the large search space. Some form of limit to these large searches and returning to the depth-first-search again (c. *zero-instant*, *quarter-inc*, *quarter-instant*) has a positive impact, but does not reach the original performance. Only for some specific problem instances the heuristic finds a witness point faster, but seems to be more fortunate with the search paths, rather than being impacted by a reduced amount of irrational samples during the search. Avoiding irrational samples does not lead to consistently less irrational real roots encountered during the search.

In summary the avoidance of irrational values leads generally to a more expensive search with additional samples and formula evaluations, and has the consequence of longer computations.

5.1 Future work

Besides the presented heuristic configurations, there could be different initial values explored (e.g a variety of constants could be tried), and different increment strategies, like exponential growth in allowed irrational values.

Alternatively, an iterative traversal technique, compatible with this heuristic, could be implemented and explored. The current recursive approach restricts us to navigate the search tree neighbour-by-neighbour, where we can not jump directly to a previously encountered sample. On each level it could store every possible sample and expand the one, which a heuristic would value most desirable. But this would come at the cost of an even higher memory footprint, which could require some form of limit to the encountered samples. Such limit could also benefit our heuristic approach, in a way that the search in breadth is somewhat contained.

Our heuristic would benefit by avoiding to backtrack and revisit the search tree to the next minimal unknown interval, which can be in another part of the search tree, and jump directly to it. It has to be evaluated, whether this practical improvement will have a significant impact. Otherwise there could be other heuristics applied.

Acknowledgements

Computations were performed with computing resources granted by RWTH Aachen University under project *thes1859*.

Bibliography

[ÁDEK21] Erika Ábrahám, James H Davenport, Matthew England, and Gereon 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*, 119:100633, 2021.

[BPR06] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[Bro15] Christopher W Brown. Open non-uniform cylindrical algebraic decompositions. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 85–92, 2015.

[CH91] George E Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, 1991.

[CKJ+15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. Smt-rat: an open source c++ toolbox for strategic and parallel smt solving. In *Theory and Applications of Satisfiability Testing–SAT 2015: 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings 18*, pages 360–368. Springer, 2015.

[Col] George E Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20–23, 1975*.

[EEG+15] Hassan Errami, Markus Eiswirth, Dima Grigoriev, Werner M. Seiler, Thomas Sturm, and Andreas Weber. Detection of hopf bifurcations in chemical reaction networks using convex coordinates. *Journal of Computational Physics*, 291:279–302, 2015.

[Fef06] Solomon Feferman. Tarski's influence on computer science. *Logical Methods in Computer Science*, 2, 2006.

[JDM13] Dejan Jovanović and Leonardo De Moura. Solving non-linear arithmetic. *ACM Communications in Computer Algebra*, 46(3/4):104–105, 2013.

[KN22] Gereon Kremer and Jasper Nalbach. Cylindrical algebraic coverings for quantifiers. In *Proceedings of the 7th International Workshop on Satisfiability Checking and Symbolic Computation (SC^2 2022)*, 2022.

[McC98] Scott McCallum. An improved projection operation for cylindrical algebraic decomposition. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 242–268. Springer, 1998.

[PSB+24] Mathias Preiner, Hans-Jörg Schurr, Clark Barrett, Pascal Fontaine, Aina Niemetz, and Cesare Tinelli. Smt-lib release 2024 (non-incremental benchmarks), April 2024.

[Stu17] Thomas Sturm. A survey of some methods for real quantifier elimination, decision, and satisfiability and their applications. *Mathematics in Computer Science*, 11(3):483–502, Dec 2017.

[Tar98] Alfred Tarski. A decision method for elementary algebra and geometry. In *Quantifier elimination and cylindrical algebraic decomposition*, pages 24–84. Springer, 1998.

[WK23] Tobias Winkler and Joost-Pieter Katoen. On certificates, expected runtimes, and termination in probabilistic pushdown automata. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2023.