Practical Approximate Quantifier Elimination for Non-linear Real Arithmetic

S. Akshay[1], Supratik Chakraborty[1], Amir Kafshdar Goharshady[2(✉)], R. Govind[3], Harshit Jitendra Motwani[2], and Sai Teja Varanasi[1]

[1] IIT Bombay, Mumbai, India
{akshayss,supratik,200050152}@cse.iitb.ac.in
[2] HKUST, Hong Kong, China
{goharshady,csemotwani}@ust.hk
[3] Uppsala University, Uppsala, Sweden
govind.rajanbabu@it.uu.se

Abstract. Quantifier Elimination (QE) concerns finding a quantifier-free formula that is semantically equivalent to a quantified formula in a given logic. For the theory of non-linear arithmetic over reals (NRA), QE is known to be computationally challenging. In this paper, we show how QE over NRA can be solved approximately and efficiently in practice using a Boolean combination of constraints in the linear arithmetic over reals (LRA). Our approach works by approximating the solution space of a set of NRA constraints when all real variables are bounded. It combines adaptive dynamic gridding with application of Handelman's Theorem to obtain the approximation efficiently via a sequence of linear programs (LP). We provide rigorous approximation guarantees, and also proofs of soundness and completeness (under mild assumptions) of our algorithm. Interestingly, our work allows us to bootstrap on earlier work (viz. [38]) and solve quantified SMT problems over a combination of NRA and other theories, that are beyond the reach of state-of-the-art solvers. We have implemented our approach in a preprocessor for Z3 called POQER. Our experiments show that POQER+Z3EG outperforms state-of-the-art SMT solvers on non-trivial problems, adapted from a suite of benchmarks.

1 Introduction

Given a first-order logic formula with quantifiers, quantifier elimination (or QE) requires us to find a quantifier-free formula that is semantically equivalent to the given quantified formula. Not every first-order theory admits QE; however, several important ones do, and QE for several such theories are implemented in modern Satisfiability Modulo Theories (SMT) solvers (viz. [1,9,27,31,36]). QE in combinations of first-order theories is particularly challenging, and algorithms that achieve this for some theories used in practical applications have been reported in earlier works (e.g. [11,38,51]). However, QE (even approximate versions) in combinations of theories including non-linear real arithmetic (NRA)

A. Platzer et al. (Eds.): FM 2024, LNCS 14933, pp. 111–130, 2025.
https://doi.org/10.1007/978-3-031-71162-6_6

has proved more difficult. This is not surprising since QE over NRA is computationally challenging by itself [29]. In this paper, we add to the repertoire of practically efficient techniques for reasoning about NRA constraints by showing how NRA constraints over bounded variables can be approximated efficiently using a Boolean combination of real interval constraints. This yields a practical algorithm for approximately solving QE over NRA, and also allows us to bootstrap on existing QE techniques that work well for combinations of LRA and other theories (viz. [38]) to solve QE in combinations of theories including NRA.

At the heart of our approach lies a practically efficient technique for approximating a Boolean combination of polynomial inequalities over bounded reals with a Boolean combination of real interval constraints. This immediately yields a practically efficient approximate QE algorithm for NRA. This problem is also popularly called QE over reals (henceforth called QER). QER is a central problem in computer algebra and real algebraic geometry, with many practical applications, including control system design [35,42,46], program verification [14,50,62,64,66], analysis of hybrid systems [7,79] and robot motion planning [53,56,77]. The study of QER has a long and storied history. Tarski first showed the decidability of QER in [71]. By the Tarski-Seidenberg theorem, the projection of a semi-algebraic set (i.e. solutions of a Boolean combination of polynomial inequalities) is always semi-algebraic [67,71]. Hence, it suffices to eliminate existentially quantified variables from a conjunction of polynomial inequalities. A landmark result in this area was the development of the *cylindrical algebraic decomposition (CAD)* algorithm by Collins [28] in 1975. Over the past half century, CAD has remained one of the most important algorithms for QER, although several improvements have been proposed over the years. An excellent, albeit dated, survey of these algorithms can be found in [16,29], while more recent works have been reported in [3,12,26,49,52,57,58,65,68]. The book by Basu, Pollack and Roy [10] is a definitive treatise on exact algorithms for QER and related problems. Over the years, practical scalability concerns have also motivated researchers to investigate versions of QER for special cases [32,47,48,54,59,63,75,76]. Advances resulting from these efforts have been implemented in state-of-the-art tools, including open-source academic tools such as QEPCAD [13,30], REDLOG [34], SMT-RAT [31] and SageMath [72], as well as commercial tools such as Mathematica [44,68,69] and Maple [25,45].

The verification community has long been interested in QER, thanks to its many applications in problems related to automated reasoning. For example, QER for polynomial equalities and disequalities has been used to compute strongest post- and weakest pre-conditions of programs [14,62], to compute abstract transformers for program statements [60], and for inductive assertion and program invariant generation [50,66]. In hybrid systems verification, reach set computation has been shown to reduce to QER [7,79]. In [78], QER has been used to find parametric optimal strategies for Markov decision processes. Quantifier elimination in mixed theories including the theory of linear real arithmetic (LRA) has been reported in several earlier works (see e.g. [11,38,51]). For example, [11] gives model based projection techniques for several combinations

of theories and [38] gives e-graph based techniques for similar combinations. However, quantifier elimination (even approximate versions) in combinations of theories including NRA has remained elusive in practice, primarily because of the high-degree polynomials that result in general from QER.

Our algorithm provides strong guarantees of approximation and allows the user to trade off precision for performance. It builds upon the well-known theorem of Handelman [41] which characterizes positive polynomials over polytopes. This theorem has previously been used in developing static analysis methods for termination and runtime analysis [17–19,43], cost analysis [15,21,23,24,70,74], invariant generation [20], reachability [8,73] and LTL verification [22], as well as program synthesis [4,39]. See [6] for a comparison between the current work and [4]. Most of these approaches are template-based and use Handelman's theorem to solve for unknown variables in their templates. In contrast, our approach is gridding-based and uses techniques similar to PROPhESY [33] but combines them with Handelman-based reasoning. The primary workhorse we use at the backend is a linear-programming (LP) solver, with occasional invocations of an SMT solver. This allows our method to scale well on many non-trivial examples. Our primary contributions are as follows:

1. We formalize two notions of approximation for QER, called ϵ-*approximation* and (ϵ, δ)-*approximation*, that are motivated by practical applications and introduce union of (adaptively sized) hyperrectangles as a knowledge representation form for approximate QER. This allows us to compute ϵ- and (ϵ, δ)-approximations of QER, for every $\epsilon, \delta > 0$, efficiently in practice.
2. We present an approach to over- and under-approximate NRA constraints with a Boolean combination of LRA constraints, where each dimension is bounded. Specifically, we use Handelman's Theorem in combination with dynamic adaptive gridding to reduce the approximation problem to multiple linear programming (LP) instances, that are then discharged by a state-of-the-art LP solver.
3. We prove the soundness of our algorithm, and its completeness under two different settings. Assuming access to a sound and complete satisfiability oracle for polynomial inequalities (in practice, an SMT solver), we show that our algorithm produces an ϵ-approximation of QER. Without access to the above oracle, and relying only on linear programming, we can obtain (ϵ, δ)-approximations of QER. Our notions of approximation for the original semi-algebraic set are closely related to those of [37]. Due to the special format of our approximation as a union of hyperrectangles, we obtain approximations of the projection set easily. Our approach extends the results of [55] which directly approximate the projection.

4. We apply this new algorithm to show how QE over theories involving Nonlinear Real Arithmetic (NRA) can be reduced to QE over LRA and other theories, thereby making it possible to solve problems beyond the reach of state-of-the-art solvers.
5. We show the practical effectiveness of our algorithm through two sets of experiments with POQER – a tool that implements our algorithm. First, a comparison with state-of-the-art tools shows that POQER significantly outperforms available open-source tools that perform exact QER, even with small values of ϵ and δ. Comparison with Mathematica, a commercial tool, shows that our tool almost always generates solutions (unions of hyperrectangles) that are easier to process subsequently than solutions generated by Mathematica. Second, we demonstrate how POQER can find approximate solutions for NRA+ADT benchmarks well beyond the reach of state-of-the-art SMT-solvers like Z3 and Z3EG.

2 Algorithm

In this section, we start by formalizing our quantifier elimination problem as computing a projection $\pi(S)$ of a semialgebraic set S. We then present the concept of ϵ-inflations to overapproximate semialgebraic sets, in our case the projection $\pi(S)$, to a desired level ϵ of precision. This is followed by our algorithm which computes an ϵ-approximation of $\pi(S)$.

2.1 Problem Definition

Input Format. We are given a positive real number ϵ and a finite set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ of real-valued variables partitioned into two sets $\mathcal{V}_1$ and $\mathcal{V}_2$. Throughout this paper, we use the standard vector notation for valuations to variables and assume that $\mathcal{V}_1$ comes before $\mathcal{V}_2$ lexicographically. Our input also contains a formula φ from the grammar below:

$$\begin{array}{rl} \varphi := \ell \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi & \text{formulas} \\ \ell := f \geq 0 \mid f > 0 & \text{literals} \\ f \in \mathbb{R}[\mathcal{V}] & \text{polynomials} \end{array}$$

The input formula φ naturally defines the semialgebraic set

$$S := SAT(\varphi) := \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \models \varphi\}.$$

We assume the set S is bounded, i.e. there is a positive real number B given in the input such that for all $\mathbf{x} \in S$, we have $||\mathbf{x}|| < B$.

Projection. Given a set $S \subseteq \mathbb{R}^n$, its projection $\pi(S)$ onto $\mathcal{V}_1$ is defined as

$$\pi(S) := \{\mathbf{x_1} \in \mathbb{R}^{|\mathcal{V}_1|} \mid \exists \mathbf{x_2} \in \mathbb{R}^{|\mathcal{V}_2|} \ \ (\mathbf{x_1}, \mathbf{x_2}) \in S\}.$$

Our goal is to approximate $\pi(S)$. We now formalize this.

ϵ-inflations and ϵ-approximations. Given $\epsilon > 0$ and a set $T \subseteq \mathbb{R}^n$, we define the ϵ-*inflation* of T as

$$\mathcal{I}_\epsilon(T) := \{\mathbf{x} \in \mathbb{R}^n \mid \exists \mathbf{x}' \in T \;\; ||\mathbf{x} - \mathbf{x}'|| < \epsilon\}.$$

In other words, $\mathcal{I}_\epsilon(T)$ consists of all the points in T as well as points that are within a distance ϵ to T. We say $O \subseteq \mathbb{R}^n$ is an ϵ-*approximation* of T iff $T \subseteq O \subseteq \mathcal{I}_\epsilon(T)$. Intuitively, an ϵ-approximation includes everything in the original set T and may also include some extra points, but these points are guaranteed to be within ϵ distance to the boundary of T. In this work, we use the Euclidean norm, but our results are independent of the distance metric used and can be straightforwardly extended to other norms.

Output. Our algorithm outputs an ϵ-approximation of $\pi(S)$.

Example. Figure 1 shows a semi-algebraic set in black and its ϵ-inflation in red.

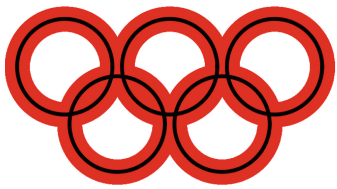

Fig. 1. A semi-algebraic set S (black) and its ϵ-inflation (red) (Color figure online)

Hyperrectangles. A *hyperrectangle* $H \subseteq \mathbb{R}^n$ is the set of points that satisfy the inequalities

$$\psi_H := \begin{cases} \alpha_1 \leq v_1 \leq \beta_1 \\ \alpha_2 \leq v_2 \leq \beta_2 \\ \vdots \\ \alpha_n \leq v_n \leq \beta_n \end{cases}$$

where the α_i and β_i's are real constants and we have $\beta_i > \alpha_i$ for every $1 \leq i \leq n$.

Literal Complements. Let ℓ be a literal. We define its complement $\overline{\ell}$ as follows:

$$\overline{\ell} := \begin{cases} -f > 0 & \ell = (f \geq 0) \\ -f \geq 0 & \ell = (f > 0) \end{cases}$$

It is easy to see that $\overline{\ell} \equiv \neg\ell$.

Ternary Evaluation. Let φ be a Boolean formula and L the set of literals appearing in φ. Consider a function $\theta : L \to \{0, 1, ?\}$ that assigns a truth value to each literal. Here, ? models uncertainty. Based on the function θ, we define the evaluation of φ recursively as follows:

$$[|\ell|]_\theta = \theta(\ell) \qquad [|\varphi_1 \vee \varphi_2|]_\theta = \begin{cases} 1 & [|\varphi_1|]_\theta = 1 \vee [|\varphi_2|]_\theta = 1 \\ 0 & [|\varphi_1|]_\theta = 0 \wedge [|\varphi_2|]_\theta = 0 \\ ? & \text{otherwise} \end{cases}$$

$$[|\neg\varphi|]_\theta = \begin{cases} ? & [|\varphi|]_\theta = ? \\ \neg[|\varphi|]_\theta & \text{otherwise} \end{cases} \qquad [|\varphi_1 \wedge \varphi_2|]_\theta = \begin{cases} 1 & [|\varphi_1|]_\theta = 1 \wedge [|\varphi_2|]_\theta = 1 \\ 0 & [|\varphi_1|]_\theta = 0 \vee [|\varphi_2|]_\theta = 0 \\ ? & \text{otherwise} \end{cases}$$

Informally, we are going to use this kind of evaluation when we want to check whether a given φ holds over all points in a set (1), none of the points in the set (0) or potentially some of them (?). We say we are *uncertain* about φ when $[|\varphi|]_\theta = ?$.

2.2 Our Overapproximation Algorithm

Oracles. Our algorithm is modular and relies on two oracles:

- *Implication Oracle:* Given a hyperrectangle $H \subseteq \mathbb{R}^n$ and a literal ℓ, this oracle checks whether ℓ holds at every point in H. Equivalently, as ψ_H is the formula defining H, it checks whether $\forall \mathbf{x} \in \mathbb{R}^n \quad \psi_H \Rightarrow \ell$.
- *Satisfiability Oracle:* This oracle decides whether a given semialgebraic set is non-empty, i.e., it checks the satisfiability of a given formula φ.

We say that an oracle is *sound* if whenever it returns true, the implication (resp. satisfiability) holds. Conversely, an oracle is *complete* if whenever the implication (resp. satisfiability) holds, it returns true.

In this section, we provide the main procedure of our algorithm, assuming that the two oracles above are available. In Sect. 2.3, we will provide an LP-based implication oracle. Thus, calls to the implication oracle are relatively cheap in practice. In contrast, we rely on SMT solvers as satisfiability oracles. Thus, for practical scalability, our approach calls this oracle as late as possible and only in ϵ-diameter subsets of $\mathbb{R}^n$. Finally, in Sect. 2.4 we show that our overapproximation remains sound even in the absence of a satisfiability oracle but can only provide a weaker guarantee of approximation quality.

Our Algorithm. We are now ready to present our algorithm that finds an ϵ-approximation of $\pi(S)$. See [6] for a discussion of the intuition. Our algorithm consists of three steps and is provided in Algorithm 1.

Step 1. Literal Extraction. In the first step, our algorithm generates a set L consisting of all literals ℓ that appear in the formula φ. This is done by a standard parsing of φ.

Step 2. Dynamic Gridding. In this step, our initial goal is to produce an ϵ-approximation O of S itself, rather than its projection. Given that S is bounded, we can apply the idea of gridding. However, we do this in a dynamic and recursive manner, creating smaller grid cells only when necessary. We keep a set A of hyperrectangles whose union forms our answer. Initially $A = \emptyset$. We start with a hyperrectangle H_0 which covers all of S as the initial grid cell. For example, we can set $H_0 = \{(x_1, x_2, \ldots, x_n) \mid -B \leq x_i \leq B\}$. When processing each grid cell H, our algorithm does the following:

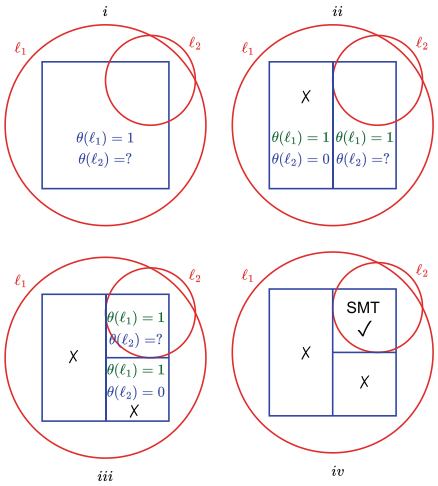

Fig. 2. An example of our gridding algorithm with memoization

(a) For every literal $\ell \in L$, use the implication oracle to decide whether ℓ holds at every point in H, i.e. check $\forall \mathbf{x} \in \mathbb{R}^n \quad \psi_H \Rightarrow \ell$.

(b) For every literal $\ell \in L$, use implication oracle to decide whether its complement $\overline{\ell}$ holds at every point in H, i.e. query the oracle for $\forall \mathbf{x} \in \mathbb{R}^n \;\; \psi_H \Rightarrow \overline{\ell}$.
(c) Create a ternary valuation $\theta : L \to \{0, 1, ?\}$ in which $\theta(\ell) = 1$ if the check in (a) passes, $\theta(\ell) = 0$ if the check in (b) passes and otherwise $\theta(\ell) =?$. Use this valuation to evaluate $[|\varphi|]_\theta$, thus deciding whether φ holds at every point in H.
(d) If $[|\varphi|]_\theta = 1$, then add the grid cell H to the answer A. Conversely, if $[|\varphi|]_\theta = 0$, exclude H from A. The only remaining case is if we are uncertain about φ. We break this down into two further cases:
  (i) If the diameter of H is more than ϵ, cut H into two halves H' and H'' by bisecting its longest edge. Apply the algorithm recursively on both.
  (ii) If the diameter of H is at most ϵ, then use the satisfiability oracle on $\psi_H \wedge \varphi$. This will tell us whether there exists at least one point in H that satisfies φ. If such a point exists, include H in A. Else, exclude H.

Memoization. If one of the checks in (a) or (b) above succeed, then the corresponding (complement) literal holds at every point in H. Thus, if the algorithm later divides H in (d), we do not need to check the same literals again in H' and H''. Hence, our algorithm memoizes the set of literals that are known to hold or not hold at every point in H. This is shown as L_1 and L_0 in the pseudocode.

Example. Figure 2 shows a simple example of our dynamic gridding. Our goal is to approximate the intersection $\varphi = \ell_1 \wedge \ell_2$ of the two red circles, i.e. each circle corresponds to a literal ℓ_i. A grid cell is shown in blue in part (i). Initially, our algorithm finds out that ℓ_1 holds at every point in the cell, but ℓ_2 is uncertain. Thus, in part (ii), we divide our cell in two. At this point we already know that ℓ_1 holds in both halves. This is memoized (shown in green) and not recomputed. In the left half, ℓ_2 does not hold at any point. Thus, we have $\theta(\ell_2) = 0$ and exclude this half from the solution. In part (iii), we cut the right half in two. The bottom part is excluded from the solution since no point in it satisfies ℓ_2. In the top right part, ℓ_1 is known to hold everywhere (memoized) and ℓ_2 is uncertain. However, at this point the diameter of the cell is less than ϵ. Thus, our approach makes an SMT call in part (iv) and realizes that there is a point in this cell that satisfies φ. Hence, the top right cell is included in the answer.

Step 3. Projection. Let $O = \bigcup_{H \in A} H$. We will prove further below that O is an ϵ-approximation of S. However, we would like an ϵ-approximation of $\pi(S)$. In this step, the algorithm computes $\pi(O) = \bigcup_{H \in A} \pi(H)$ and outputs it as the answer. We note that projecting each hyperrectangle $H \in A$ is a simple matter of dropping some constraints. Specifically, we have:

$$\psi_H = \begin{cases} \alpha_1 \le v_1 \le \beta_1 \\ \vdots \\ \alpha_n \le v_n \le \beta_n \end{cases} \quad \Rightarrow \quad \psi_{\pi(H)} = \begin{cases} \alpha_1 \le v_1 \le \beta_1 \\ \vdots \\ \alpha_{|\mathcal{V}_1|} \le v_{|\mathcal{V}_1|} \le \beta_{|\mathcal{V}_1|} \end{cases}.$$

Theorem 1 (Correctness, Proof in [6]). *Assume that we have a sound implication oracle and a sound and complete satisfiability oracle. Given*

Algorithm 1. POQER

```
1:  A ← ∅
2:  L ← ∅
3:  procedure Main(φ, ϵ, n, V, V1, V2, B)
4:      L ← literals in φ                                          ▷ Step 1
5:      ψ_{H0} ← ⋀_{i=1}^{n} −B ≤ v_i ≤ B
6:      Grid(H0, ∅, ∅, φ, ϵ, n, V)                                 ▷ Step 2
7:      X ← ∅
8:      for all H ∈ A do                                           ▷ Step 3
9:          X ← X ∪ Project(H, V1)
10:     return X
11: procedure Grid(H, L0, L1, φ, ϵ, n, V)
12:     θ ← ∅
13:     for all ℓ ∈ L do
14:         if ℓ ∈ L1 ∨ ImplicationOracle(H, ℓ, n, V) then         ▷ Step 2 (a)
15:             θ[ℓ] ← 1
16:             L1 = L1 ∪ {ℓ}                                      ▷ Memoization
17:         else if ℓ ∈ L0 ∨ ImplicationOracle(H, ℓ̄, n, V) then    ▷ Step 2 (b)
18:             θ[ℓ] ← 0
19:             L0 = L0 ∪ {ℓ}                                      ▷ Memoization
20:         else
21:             θ[ℓ] ←?
22:     if [|φ|]_θ = 1 then                                        ▷ Step 2 (d), Ternary Evaluation
23:         A ← A ∪ {H}                                            ▷ Adding H to the overapproximation
24:     else if [|φ|]_θ =? then
25:         if Diameter(H) ≥ ϵ then
26:             H′, H′′ ← CutInHalves(H)
27:             Grid(H′, L0, L1, φ, ϵ, n, V)                       ▷ Recursive Calls on Halves of H
28:             Grid(H′′, L0, L1, φ, ϵ, n, V)
29:         else if SatisfiabilityOracle(ψ_H ∧ φ, n, V) then
30:             A ← A ∪ {H}                                        ▷ Adding H to the overapproximation
```

$\varphi, \epsilon, n, \mathcal{V}, \mathcal{V}_1, \mathcal{V}_2$, and B as input, let $S := \{\mathbf{x} \in \mathbb{R}^n \;\; x \models \varphi\}$ be bounded by a ball of radius B around the origin. Then, Algorithm 1 (POQER), outputs an ϵ-approximation of $\pi(S)$, i.e. the projection of S onto $\mathcal{V}_1$, as desired.

2.3 Our Implication Oracle

As mentioned in the previous section, our algorithm depends on a sound oracle to check whether a given polynomial inequality (literal) ℓ of the form $f \geq 0$ or $f > 0$ holds over the entirety of a hyperrectangle H. In this section, we provide such an oracle. Specifically, given the inequalities ψ_H that define the hyperrectangle H, our goal is to check whether $\forall \mathbf{x} \in \mathbb{R}^n \;\; \psi_H \Rightarrow \ell$ holds. Our algorithm is sound and can also provide semi-completeness guarantees for *strict* literals, i.e. literals of the form $f > 0$.

Semi-group generated by Φ. Consider the set $\mathcal{V} = \{v_1, \ldots v_n\}$ of real-valued variables and the following system of linear inequalities over $\mathcal{V}$:

$$\Phi := \begin{cases} a_{1,0} + a_{1,1} \cdot v_1 + \ldots + a_{1,n} \cdot v_n \bowtie_1 0 \\ \qquad\qquad \vdots \\ a_{m,0} + a_{m,1} \cdot v_1 + \ldots + a_{m,n} \cdot v_n \bowtie_m 0 \end{cases}$$

where $\bowtie_i \in \{>, \geq\}$ for all $1 \leq i \leq m$. Let g_i be the left hand side of the i-th inequality, i.e. $g_i(v_1, \ldots, v_n) := a_{i,0} + a_{i,1} \cdot v_1 + \ldots a_{i,n} \cdot v_n$. The *semi-group* of Φ is

defined as: $SG(\Phi) := \left\{ \prod_{i=1}^{m} g_i^{k_i} \mid m \in \mathbb{N} \ \wedge \ \forall i \ \ k_i \in \mathbb{N} \cup \{0\} \right\}$. In other words, this semi-group contains all polynomials that can be obtained as a multiplication of the g_i's. Note that $1 \in SG(\Phi)$. We define $SG_d(\Phi)$ as the subset of polynomials in $SG(\Phi)$ of degree at most d.

Theorem 2 (Handelman's Theorem [41]). *Consider the following system of equations over $\mathcal{V}$:*

$$\Phi := \begin{cases} a_{1,0} + a_{1,1} \cdot v_1 + \ldots + a_{1,n} \cdot v_n \geq 0 \\ \vdots \\ a_{m,0} + a_{m,1} \cdot v_1 + \ldots + a_{m,n} \cdot v_n \geq 0 \end{cases}.$$

If Φ is satisfiable, its solution set is compact, $f \in \mathbb{R}[\mathcal{V}]$ and we have $\forall \boldsymbol{x} \in \mathbb{R}^n \ \ \Phi \Rightarrow f > 0$, then there exist non-negative real numbers $\lambda_0, \ldots \lambda_k$ and semigroup elements $h_1, \ldots, h_k \in SG(\Phi)$ such that $f = \lambda_0 + \lambda_1 \cdot h_1 + \cdots + \lambda_k \cdot h_k$.

Basic Idea of Our Implication Oracle. Consider the hyperrectangle H and its defining inequalities ψ_H which can be rewritten in the following form:

$$\psi_H = \begin{cases} g_1 := v_1 - \alpha_1 \geq 0 \\ g_2 := \beta_1 - v_1 \geq 0 \\ \vdots \\ g_{2 \cdot n - 1} := v_n - \alpha_n \geq 0 \\ g_{2 \cdot n} := \beta_n - v_n \geq 0 \end{cases}$$

It is clear by definition that every g_i is non-negative at every point in H. Thus, any multiplication $h \in SG_d(\psi_H)$ of the g_i's will also be non-negative throughout H. Finally, we can take any linear combination of such polynomials with non-negative coefficients, i.e.

$$f = \lambda_0 + \lambda_1 \cdot h_1 + \cdots + \lambda_k \cdot h_k \tag{1}$$

$h_i \in SG_d(\psi_H) \quad \lambda_i \geq 0$, and such an f will be non-negative at every point in H. Moreover, if $\lambda_0 > 0$, then f will be strictly positive at every point in H.

The Oracle. Our implication oracle is provided in Algorithm 2. Given a fixed degree $d \in \mathbb{N}$, it first generates $SG_d(\psi_H)$. It then symbolically computes a linear combination of the polynomials in $SG_d(\psi_H)$ by creating fresh variables for each λ_i as in the RHS of Eq. (1). This allows us to write Eq. (1) symbolically. Note that both sides of this equation are polynomials in $\mathbb{R}[\mathcal{V}]$, thus they are equal if and only if each monomial has the same coefficient on both sides. The algorithm computes the coefficient of each monomial on both sides and equates them. This leads to a linear programming instance over the λ_i's, which is in turn handled by an external solver.

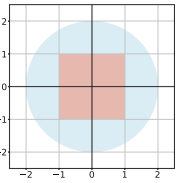

Fig. 3. The hyperrectangle H defined by ψ_H lying inside the region $f > 0$.

Example. Consider the literal $\ell = (f > 0)$ where $f = 4 - x^2 - y^2$. Let H be the hyperrectangle defined by the inequalities $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$. We have $\psi_H = \{x + 1 \geq 0, -x + 1 \geq 0, y + 1 \geq 0, -y + 1 \geq 0\}$. Let $d = 2$. Then, $SG_d(\psi_H)$ contains all polynomials of degree at most 2 that can be obtained as a multiplication of the g_i's. This includes $g_1^2, g_1 \cdot g_2, g_1 \cdot g_3, g_1 \cdot g_4, g_2^2, g_2 \cdot g_3, g_2 \cdot g_4, g_3^2, g_3 \cdot g_4, g_4^2, g_1, g_2, g_3, g_4, 1$. Since ℓ holds at every point in the hyperrectangle H, we can write f as a linear combination of these polynomials as follows: $4 - x^2 - y^2 = 1 \cdot (1 - x^2) + 1 \cdot (1 - y^2) + 2 \cdot 1 = 1 \cdot g_1 \cdot g_2 + 1 \cdot g_3 \cdot g_4 + 2 \cdot 1$ (Fig. 3).

Algorithm 2. Our Implication Oracle

```
1:  procedure ImplicationOracle(H, ℓ, n, V, d)
2:      LP ← ∅
3:      SG ← {1}                                   ▷ SG will become SG_d(ψ_H)
4:      M ← {1}                                    ▷ M will become the set of all monomials of degree ≤ d
5:      for 1 ≤ i ≤ d do
6:          SG ← SG ∪ {g · h | g ∈ ψ_H ∧ h ∈ SG}
7:          M ← M ∪ {v_i · h | v_i ∈ V ∧ h ∈ M}
8:      k ← |SG|
9:      Create k + 1 fresh variables λ_0, λ_1, ..., λ_k in LP
10:     if ℓ = (f > 0) then
11:         Add the constraint λ_0 > 0 to LP
12:     else if ℓ = (f ≥ 0) then
13:         Add the constraint λ_0 ≥ 0 to LP
14:     for 1 ≤ i ≤ k do
15:         Add the constraint λ_i ≥ 0 to LP
16:     LHS ← f where ℓ = (f > 0) or ℓ = (f ≥ 0)
17:     RHS ← Σ_{i=0}^{k} λ_i · SG[i]
18:     for all m ∈ M do
19:         l = coefficient of m in LHS
20:         r = coefficient of m in RHS
21:         Add the constraint l = r to LP
22:     if LP has a solution then
23:         return true
24:     else
25:         return false
```

Theorem 3 (Soundness and Semi-completeness, Proof in [6]). *Given a hyperrectangle H and a literal ℓ in the input, and a degree bound d, Algorithm 2 is sound in deciding whether $\forall \boldsymbol{x} \in \mathbb{R}^n \quad \psi_H \Rightarrow \ell$. Moreover, if ℓ is of the form $f > 0$, then there exists a degree bound d, depending on both H and ℓ, for which the algorithm is complete in deciding $\forall \boldsymbol{x} \in \mathbb{R}^n \quad \psi_H \Rightarrow \ell$.*

Runtime Analysis. In Algorithm 2, let d be the degree, n the number of variables, and m the number of linear inequalities in our hypothesis hyperrectangle H. Then, the size of $|M| = \binom{n+d}{d}$ and $|SG| = \binom{m+d}{d}$. For each element of SG, we add a λ_i to our linear programming instance. Similarly, for each monomial in M, we add a constraint equating its coefficients on the two sides. Thus, we have an LP instance with $O\left(\binom{m+d}{d}\right)$ variables and $O\left(\binom{n+d}{d}\right)$ constraints. We note that current state-of-the-art LP-solving algorithms work in polynomial-time $O(N^\omega)$ where N is their input size and ω is the matrix multiplication constant.

2.4 Removing the Satisfiability Oracle

Our approximate quantifier elimination algorithm in Sect. 2.2 requires two oracles: one for implication and another for satisfiability. As mentioned above, we use SMT calls for the satisfiability oracle, but the implication oracle (Sect. 2.3) is much more practical and relies only on linear programming. Moreover, it provides a semi-completeness guarantee (Theorem 3) which is not used in the main algorithm (Theorem 1). So, a natural question is whether we can remove the satisfiability oracle altogether. We first argue that this is unlikely to lead to an efficient algorithm with our notion of ϵ-approximation, since it is an ETR-hard problem. However, we can provide a weaker guarantee for positive formulas.

ETR-Hardness. Let $\psi := (\exists v_1, v_2, \ldots, v_n \ \ \varphi)$ be a formula in the existential theory of the reals. ψ holds if and only if $SAT(\varphi) \neq \emptyset$, but we have

$$SAT(\varphi) \neq \emptyset \Leftrightarrow \pi(SAT(\varphi)) \neq \emptyset \Leftrightarrow \mathcal{I}_\epsilon(\pi(SAT(\varphi))) \neq \emptyset.$$

Thus, to decide ψ, we can simply find an ϵ-approximation of $\pi(SAT(\varphi))$ and check its non-emptiness.

Positive Formulas. A formula φ is called *positive* if it is generated from the grammar below:

$$\begin{array}{cc} \varphi := \ell \mid \varphi \wedge \varphi \mid \varphi \vee \varphi & \text{positive formulas} \\ \ell := f \geq 0 \mid f > 0 & \text{literals} \\ f \in \mathbb{R}[\mathcal{V}] & \text{polynomials} \end{array}$$

The only difference between this grammar and that of Sect. 2.1 is the absence of the negation operator. We note that any formula can be written as an equivalent positive formula since the complement of each literal is itself a literal. Thus, in the remainder of this section, we assume that the formula φ is positive.

(ϵ, δ)-perturbation. Let $\epsilon, \delta > 0$ and φ be a positive formula. We define the (ϵ, δ)-perturbation $SAT_{\epsilon,\delta}(\varphi)$ of $SAT(\varphi)$ recursively as follows:

- For every literal $\ell = (f > 0)$ or $\ell = (f \geq 0)$ we have

$$SAT_{\epsilon,\delta}(\ell) = \mathcal{I}_\epsilon(SAT(f + \delta \geq 0)).$$

  Intuitively, we are overapproximating $SAT(\ell)$ in two ways: (i) we are allowing the value of f to decrease to $-\delta$ instead of just 0, and (ii) we are taking an ϵ-inflation of the resulting solutions. In other words, we are considering that our evaluation of f might have a numerical error of up to δ and that our approximation of the solution set might contain some extra points which are within ϵ distance to the original set.
- If $\varphi = \varphi_1 \wedge \varphi_2$, then $SAT_{\epsilon,\delta}(\varphi) := SAT_{\epsilon,\delta}(\varphi_1) \cap SAT_{\epsilon,\delta}(\varphi_2)$.
- If $\varphi = \varphi_1 \vee \varphi_2$, then $SAT_{\epsilon,\delta}(\varphi) := SAT_{\epsilon,\delta}(\varphi_1) \cup SAT_{\epsilon,\delta}(\varphi_2)$.

We remark that we always have $SAT(\varphi) \subseteq \mathcal{I}_\epsilon(SAT(\varphi)) \subseteq SAT_{\epsilon,\delta}(\varphi)$. We say that a set O is an (ϵ, δ)-*approximation* of $SAT(\varphi)$ if $SAT(\varphi) \subseteq O \subseteq SAT_{\epsilon,\delta}(\varphi)$.

We note that there are subtle yet important differences in the definitions of ϵ-approximation and (ϵ, δ)-approximation, thus an $(\epsilon, 0)$-approximation is not the same as an ϵ-approximation as defined in Sect. 2.1.

Modified Algorithm. We take the exact same algorithm as in Sect. 2.2 (Algorithm 1), but only change Step 2 (d)(ii) as follows:

- If the diameter of H is at most ϵ, for every literal $\ell \in L$ of the form $f > 0$ or $f \geq 0$, use the implication oracle to decide the following formula:
  - $\forall \mathbf{x} \in \mathbb{R}^n \quad \psi_H \Rightarrow -f - \delta > 0$

  If the check passes, update $\theta(\ell)$ to 0. Otherwise, update it to 1. Finally, compute $[|\varphi|]_\theta$ and if it is 1 then include H in the answer A.

Algorithm 3 in [6] provides a psuedocode of this variant. See [6] for a discussion of the intuition behind this approach.

Theorem 4 (Proof in [6]). *Assume that we have a sound and complete implication oracle. Given $\varphi, \epsilon, \delta, n, \mathcal{V}, \mathcal{V}_1, \mathcal{V}_2$ and B as input, let $SAT(\varphi)$ be bounded by a ball of radius B around the origin. Then, Algorithm 3 of [6] (Modified POQER), outputs a set X such that $\pi(SAT(\varphi)) \subseteq X \subseteq \pi(SAT_{\epsilon,\delta}(\varphi))$. In other words, it outputs an (ϵ, δ)-approximation of $SAT(\varphi)$.*

3 Experimental Results

We implemented our approach in a tool called POQER (Practical Overapproximate Quantifier Elimination for Reals) and performed two experiments:

- Our first experiment considers QER over formulas in Non-linear Real Arithmetic (NRA). To the best of our knowledge, we are providing the first approximate solution for quantifier elimination over NRA. Thus, we had to compare our scalability with previous *exact* solutions. Note that under-approximating the result of applying QER to a polynomial constraint φ is equivalent to complementing the over-approximation of QER applied to $\neg\varphi$. Hence, we focus only on over-approximating QER in this experiment. Moreover, since every Boolean combination of polynomial constraints can be equivalently expressed in disjunctive normal form, and since existential quantification distributes over disjunction, we focus only on conjunctions of polynomial constraints.
- In our second experiment, we considered the problem of satisfiability checking for mixed formulas in NRA+ADT, i.e. theories of Non-linear Real Arithmetic and Algebraic Data Types. We first used POQER to eliminate quantifiers in the NRA part of the formula, obtaining both over- and under-approximations, and writing it as a union of hyperrectangles. We then combined this approximation with the ADT part and passed it to a state-of-the-art tool for LRA+ADT, namely Z3EG. As baselines, we compared our performance with state-of-the-art SMT solvers Z3 and Z3EG.

Implementation and Environment Details. We implemented POQER (Algorithm 1) in `C++`, with Z3 as the satisfiability oracle (see Sect. 2). We used Gurobi [40] as our LP-solver. The results were obtained on a 3.5GHz Intel Core i5 1030NG7 Machine with 8 GB of RAM running MacOS. We will submit our tool (POQER) for artifact evaluation and make it publicly available as free and open-source software.

First Experiment (QER). Due to the lack of publicly-available tools performing *approximate* quantifier elimination in NRA (non-linear real arithmetic), we are unable to present an apples-to-apples comparison. However, we report comparisons with several tools that perform exact QER. There are several (academic and commercial) tools implementing CAD and its variants, but we observe that the result of QER given by them is often as high-degree polynomials or their radicals. This makes it practically impossible to use these results in downstream processing using modern SMT solvers. Hence, we study not only whether these tools are able to solve a QER problem within a time budget, but also the format in which they provide the answer. Although our algorithm is parallel, we only compare using a sequential variant to be as fair as possible.

CAD (and variant algorithms for QER) are reported to be implemented in publicly available SMT solvers such as SMT-RAT [31], Yices2 [36], Z3 [61] and cvc5 [9]. However, SMT solvers are decision procedures for checking satisfaction of (possibly quantified) formulas in a combination of theories. Hence, they do not provide the result of quantifying a subset of variables in a formula. While this suffices in applications where the goal is to check if a formula is satisfiable, it falls short of the requirements in other applications, viz. weakest pre-condition computation, where we genuinely require the result of quantifying a subset of variables from NRA constraints. SMT-RAT [31] appears to have had a soundness issue in the quantifier elimination for QER (as noted in [2]) which we were unable to circumvent. Therefore, we compare our approach to two state-of-the-art methods: (a) SageMath [72], a versatile open-source computer algebra system, that includes an implementation of QEPCAD [13,30], and (b) Mathematica [44], a widely-used and highly-optimized commercial computer algebra system, that employs a portfolio of powerful algorithms and heuristics for QER. We aim to answer the following research questions through our first experiment:

RQ1: Given a time of 30 min, how many QER tasks from our benchmark suite are solved by SageMath, Mathematica and POQER? We use the `Reduce` function in Mathematica and `qepcad` in SageMath.

RQ2: For each of the above three tools, is the output of a QER problem free of further NRA constraints?

RQ3: Does using Handelman's Theorem and linear programming in POQER help achieve better performance compared to the use of a state-of-the-art SMT solver (Z3)? To answer this, we performed an ablation study by removing our Handelman-based implication oracle and instead directly applying Z3 as both implication and satisfiability oracles.

Benchmarks. Given the lack of standard benchmarks for QER, we designed a suite of benchmarks, each of which is a conjunction of polynomial inequalities,

with range constraints on each dimension. Our benchmarks (see [6] for details) have 2–8 variables, degrees 2–6, and between 2 to 10 polynomials each.

Results. Our results are summarized in Table 1. We computed ϵ-approximations using POQER for three different values of ϵ to understand how POQER's performance scales with decreasing values of ϵ. We observe that SageMath failed to complete the QER task within the timeout in all but four cases, where it provided a solution in NRA, as indicated by the asterisks. Mathematica performed significantly better, generating solutions in NRA in most instances. In 4 instances, the solutions are generated in a form that would require NRA with quantifiers, if we were to encode them in SMT. We show these solutions in [6]. Clearly, these solutions are intractably complicated and pose serious challenges for downstream automated reasoning tasks. Since SageMath and Mathematica implement exact QER, it is not possible to circumvent these complicated solution forms in general. POQER with $\epsilon = 0.1$ successfully solved all benchmarks within the 30-minute timeframe, showcasing the effectiveness of our tool. POQER with $\epsilon = 0.05$ and 0.01, fell short only in two and three instances respectively. Thus, our experiments answer research question RQ1 in favor of POQER for all three values of ϵ considered, when compared to SageMath or Mathematica. RQ2 is answered in the positive for POQER in all cases, while it is mostly in the negative for SageMath and Mathematica, since they compute exact solutions which are often non-linear (in NRA) and sometimes even quantified. Finally, for RQ3, from columns Z3dir.05 and PQ.05 of Table 1, we can conclude that using Handelman's Theorem and LP-solving significantly improves the performance of POQER vis-a-vis using a state-of-the-art SMT solver (Z3) in all but one example. On Benchmark Ex11, the Z3 approach is unusually fast, which is presumably due to its internal heuristics. We also report the number of hyperrectangles that we generate as well as number of hyperrectangles after the projection of variables (shown in last two columns of Table 1). Finally, more details are reported in [6]. Looking deeper into the results, we observe that Mathematica tends to solve most benchmarks efficiently, but has difficulty in solving problems with a larger number of eliminations, as each quantifier elimination results in increasingly complex solutions. In contrast, our approach benefits from the simpler structure of our solutions, enabling us to deliver faster results even when multiple quantifiers must be eliminated. Furthermore, Mathematica produces solutions in a complicated format, i.e. as degree polynomial inequalities in multiple variables. See [6] for more details.

Second Experiment: NRA+ADT. The last observation above enables the use of approximate quantifier elimination in a combination of NRA and other theories, such as ADT (theory of algebraic data types), by reducing it to LRA+ADT. NRA+ADT formulas are often highly intractable and beyond the reach of modern SMT solvers. To the best of our knowledge, there are no approximate solutions for NRA+ADT in the literature, either. In contrast, an effective tool for LRA+ADT, called Z3EG, has recently been developed in [38].

Benchmarks. We took the Z3EG benchmarks which are in LRA+ADT and added a single NRA constraint to each of them, thus obtaining NRA+ADT

Table 1. Results of our First Experiment. **SM** refers to Sagemath, **MA** refers to Mathematica, **PQ.1, PQ.05, PQ.01** refer respectively to POQER with $\epsilon = 0.1, 0.05, 0.01$. ✓ indicates that the method terminates within 30 min, ✗ indicates a timeout. ◆ indicates that the solution is in QF-NRA. ▲ indicates NRA (with quantifiers). The columns **PQ.05** and **Z3dir.05** respectively refer to time taken in seconds by POQER and POQER where the Implication Oracle is replaced by Z3. **#H** is the number of hyperrectangles computed by POQER while **#PH** is the number of hyperrectangles after the projection.

#B	SM	MA	PQ.1	PQ.05	PQ.01	Z3dir.05	PQ.05	#H	#PH
Ex1	✗	◆	✓	✗	✗	1100	233	526	103
Ex2	✗	✗	✓	✗	✗	TO	TO	–	–
Ex3	✗	▲	✓	✓	✓	370	199	1144	256
Ex4	✗	✗	✓	✓	✓	TO	31	334	76
Ex5	◆	✓	✓	✓	✓	5	3	35	6
Ex6	✗	◆	✓	✓	✓	71	14	151	33
Ex7	◆	◆	✓	✓	✓	8	3	20	5
Ex8	◆	◆	✓	✓	✓	97	19	426	148
Ex9	✗	▲	✓	✓	✓	321	223	1144	256
Ex10	◆	✓	✓	✓	✓	17	7	46	6
Ex11	✗	▲	✓	✓	✓	191	340	695	140
Ex12	✗	✗	✓	✓	✓	212	46	16	8
Ex13	✗	✗	✓	✓	✗	409	91	89	26
Ex14	✗	✗	✓	✓	✓	204	40	16	8
Ex15	✗	▲	✓	✓	✓	231	97	687	140

formulas. The added NRA constraint is $\forall x \;\; x \in [-10, 10] \Rightarrow \exists y \in [-10, 10] \;\; x^3 + x \geq y^3 + 3 \cdot y + 4$, in which x is a variable already present in the original ADT formula and y is a fresh variable. Thus, our formulas combine NRA and ADT and are particularly challenging for modern SMT solvers. To each NRA+ADT benchmark, we first applied POQER to obtain over- and under-approximations in LRA+ADT. We then passed the resulting approximate formulas to Z3EG. As baseline comparisons, we also passed the same NRA+ADT benchmarks to Z3 and Z3EG. We observed that POQER significantly outperforms other tools on these SMT benchmarks. The results are summarized in Table 2.

Table 2. Results of our Second Experiment. TO stands for timeout > 2 mins.

Z3EG			Z3			POQER		
SAT	UNSAT	TO	SAT	UNSAT	TO	SAT	UNSAT	TO
1833	1489	1518	2096	836	1908	3262	1550	28

4 Conclusion

We presented an algorithm that computes ϵ- and (ϵ, δ)-approximations of QER, for every $\epsilon, \delta > 0$. Our approach combines adaptive dynamic gridding with application of Handelman's Theorem to solve the approximation problem via a sequence of linear programs (LP). We provide formal guarantees of soundness, and guarantee completeness under mild assumptions. Our approach also allows us to solve quantified SMT problems over mixed theories including NRA, such as NRA+ADT.

Acknowledgments. A longer version of this work, including appendices and proofs, is available at [6]. The research was supported by the SERB MATRICS grant MTR/2023/001167 of the Government of India, the Asian Universities Alliance Scholars Award Program (AUASAP), which financed a visit by S. Akshay to HKUST and another visit by A.K. Goharshady to IIT Bombay, as well as the Hong Kong Research Grants Council (RGC) ECS Project Number 26208122. The authors are grateful to the Schloss Dagstuhl – Leibniz Center for Informatics. This collaboration started at the Dagstuhl Seminar 23241: "Scalable Analysis of Probabilistic Models and Programs". Author names are ordered alphabetically.

Data Availability Statement. The artifact used to generate the experimental results, as well as its source code, are publicly available at GitHub. The artifact is also archived on Zenodo [5].

References

1. Z3. https://github.com/z3prover/z3
2. Github issue for QF_NRA formula (mcsat) (2020). https://github.com/ths-rwth/smtrat/issues/91
3. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. J. Log. Algebraic Methods Program. **119** (2021)
4. Akshay, S., Chakraborty, S., Goharshady, A.K., Govind, R., Motwani, H.J., Varanasi, S.T.: Automated synthesis of decision lists for polynomial specifications over integers. In: LPAR, vol. 100, pp. 484–502 (2024)
5. Akshay, S., Chakraborty, S., Goharshady, A.K., Govind, R., Motwani, H.J., Varanasi, S.T.: Practical approximate quantifier elimination for non-linear real arithmetic (artifact) (2024). https://doi.org/10.5281/zenodo.12600106
6. Akshay, S., Chakraborty, S., Goharshady, A.K., Govind, R., Motwani, H.J., Varanasi, S.T.: Practical approximate quantifier elimination for non-linear real arithmetic (long version). https://hal.science/hal-04629011 (2024)
7. Anai, H., Weispfenning, V.: Reach set computations using real quantifier elimination. In: HSCC, pp. 63–76 (2001)
8. Asadi, A., Chatterjee, K., Fu, H., Goharshady, A.K., Mahdavi, M.: Polynomial reachability witnesses via stellensätze. In: PLDI, pp. 772–787 (2021)
9. Barbosa, H., et al.: cvc5: A versatile and industrial-strength SMT solver. In: TACAS, pp. 415–442 (2022)

10. Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/3-540-33099-2
11. Bjørner, N.S., Janota, M.: Playing with quantified satisfaction. In: LPAR (short papers), vol. 35, pp. 15–27 (2015)
12. Brown, C.W.: Improved projection for cylindrical algebraic decomposition. J. Symb. Comput. **32**(5), 447–465 (2001)
13. Brown, C.W.: QEPCAD B: a program for computing with semi-algebraic sets using cads. SIGSAM Bull. **37**(4), 97–108 (2003)
14. Cachera, D., Jensen, T.P., Jobin, A., Kirchner, F.: Inference of polynomial invariants for imperative programs: a farewell to Gröbner bases. Sci. Comput. Program. **93**, 89–109 (2014)
15. Cai, Z., Farokhnia, S., Goharshady, A.K., Hitarth, S.: Asparagus: automated synthesis of parametric gas upper-bounds for smart contracts. Proc. ACM Program. Lang. **7**(OOPSLA2), 882–911 (2023)
16. Caviness, B.F., Johnson, J.R.: Quantifier elimination and cylindrical algebraic decomposition. Texts and Monographs in Symbolic Computation (1998)
17. Chatterjee, K., Fu, H., Goharshady, A.K.: Termination analysis of probabilistic programs through Positivstellensatz's. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 3–22. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_1
18. Chatterjee, K., Fu, H., Goharshady, A.K.: Non-polynomial worst-case analysis of recursive programs. In: CAV, vol. 10427, pp. 41–63 (2017)
19. Chatterjee, K., Fu, H., Goharshady, A.K.: Non-polynomial worst-case analysis of recursive programs. ACM Trans. Program. Lang. Syst. **41**(4), 20:1–20:52 (2019)
20. Chatterjee, K., Fu, H., Goharshady, A.K., Goharshady, E.K.: Polynomial invariant generation for non-deterministic recursive programs. In: PLDI, pp. 672–687 (2020)
21. Chatterjee, K., Fu, H., Goharshady, A.K., Okati, N.: Computational approaches for stochastic shortest path on succinct MDPs. In: IJCAI, pp. 4700–4707. ijcai.org (2018)
22. Chatterjee, K., Goharshady, A.K., Goharshady, E.K., Karrabi, M., Zikelic, D.: Sound and complete witnesses for template-based verification of LTL properties on polynomial programs. In: FM (2024)
23. Chatterjee, K., Goharshady, A.K., Meggendorfer, T., Zikelic, D.: Quantitative bounds on resource usage of probabilistic programs. In: OOPSLA (2024)
24. Chatterjee, K., Goharshady, A.K., Meggendorfer, T., Zikelic, D.: Sound and complete certificates for quantitative termination analysis of probabilistic programs. In: CAV, vol. 13371, pp. 55–78 (2022)
25. Chen, C., Maza, M.M.: Quantifier elimination by cylindrical algebraic decomposition based on regular chains. In: ISSAC, pp. 91–98. ACM (2014)
26. Chen, C., Maza, M.M.: Quantifier elimination by cylindrical algebraic decomposition based on regular chains. J. Symb. Comput. **75**, 74–93 (2016)
27. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT solver. In: Proceedings of TACAS (2013)
28. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In: Automata Theory and Formal Languages (1975)
29. Collins, G.E.: Quantifier elimination by cylindrical algebraic decomposition - twenty years of progress. In: Quantifier Elimination and Cylindrical Algebraic Decomposition, pp. 8–23 (1998)
30. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. **12**(3), 299–328 (1991)

31. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In: SAT, pp. 360–368 (2015)
32. Dantzig, G.B., Eaves, B.C.: Fourier-Motzkin elimination and its dual. J. Comb. Theory, Ser. A **14**(3), 288–297 (1973)
33. Dehnert, C., et al.: PROPhESY: a probabilistic parameter synthesis tool. In: CAV, vol. 9206, pp. 214–231 (2015)
34. Dolzmann, A., Sturm, T.: REDLOG: computer algebra meets computer logic. SIGSAM Bull. **31**(2), 2–9 (1997)
35. Dorato, P., Yang, W., Abdallah, C.T.: Robust multi-objective feedback design by quantifier elimination. J. Symb. Comput. **24**(2), 153–159 (1997)
36. Dutertre, B.: Yices 2.2. In: Computer Aided Verification, pp. 737–744 (2014)
37. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: LICS, pp. 305–314 (2012)
38. Garcia-Contreras, I., K., H.G.V., Shoham, S., Gurfinkel, A.: Fast approximations of quantifier elimination. In: CAV, pp. 64–86 (2023)
39. Goharshady, A.K., Hitarth, S., Mohammadi, F., Motwani, H.J.: Algebro-geometric algorithms for template-based synthesis of polynomial programs. Proc. ACM Program. Lang. **7**(OOPSLA1), 727–756 (2023)
40. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023). https://www.gurobi.com
41. Handelman, D.: Representing polynomials by positive linear functions on compact convex polyhedra. Pac. J. Math. **132**(1), 35–62 (1988)
42. Hong, H., Liska, R., Steinberg, S.L.: Testing stability by quantifier elimination. J. Symb. Comput. **24**(2), 161–187 (1997)
43. Huang, M., Fu, H., Chatterjee, K., Goharshady, A.K.: Modular verification for almost-sure termination of probabilistic programs. Proc. ACM Program. Lang. **3**(OOPSLA), 129:1–129:29 (2019)
44. Inc., W.R.: Mathematica, Version 14.0. https://www.wolfram.com/mathematica, Champaign, IL (2024)
45. Iwane, H., Yanami, H., Anai, H.: SyNRAC: a toolbox for solving real algebraic constraints. In: Hong, H., Yap, C. (eds.) Mathematical Software – ICMS 2014, pp. 518–522. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44199-2_78
46. Jirstrand, M.: Nonlinear control system design by quantifier elimination. J. Symb. Comput. **24**(2), 137–152 (1997)
47. John, A.K., Chakraborty, S.: A quantifier elimination algorithm for linear modular equations and disequations. In: CAV, vol. 6806, pp. 486–503 (2011)
48. John, A.K., Chakraborty, S.: A layered algorithm for quantifier elimination from linear modular constraints. Formal Methods Syst. Des. **49**(3), 272–323 (2016)
49. Jovanovic, D., de Moura, L.M.: Solving non-linear arithmetic. In: IJCAR, pp. 339–354 (2012)
50. Kapur, D.: A quantifier-elimination based heuristic for automatically generating inductive assertions for programs. J. Syst. Sci. Complex. **19**(3), 307–330 (2006)
51. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. Formal Methods Syst. Des. **48**(3), 175–205 (2016). https://doi.org/10.1007/S10703-016-0249-4
52. Kremer, G., Ábrahám, E.: Fully incremental cylindrical algebraic decomposition. J. Symb. Comput. **100**, 11–37 (2020)
53. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic reachability computation for families of linear vector fields. J. Symb. Comput. **32**(3), 231–253 (2001)

54. Loos, R., Weispfenning, V.: Applying linear quantifier elimination. Comput. J. **36**(5), 450–462 (1993)
55. Magron, V., Henrion, D., Lasserre, J.: Semidefinite approximations of projections and polynomial images of semialgebraic sets. SIAM J. Optim. **25**(4), 2143–2164 (2015)
56. McCallum, S.: Partial solution of a path finding problem using the cad method. Electron. Proc. IMACS ACA (1995)
57. McCallum, S.: On projection in cad-based quantifier elimination with equational constraint. In: ISSAC, pp. 145–149. ACM (1999)
58. McCallum, S.: On propagation of equational constraints in cad-based quantifier elimination. In: ISSAC, pp. 223–231. ACM (2001)
59. Monniaux, D.: A quantifier elimination algorithm for linear real arithmetic. In: LPAR, pp. 243–257 (2008)
60. Monniaux, D.: Automatic modular abstractions for linear constraints. In: POPL, pp. 140–151. ACM (2009)
61. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: TACAS, pp. 337–340 (2008)
62. Müller-Olm, M., Seidl, H.: Computing polynomial program invariants. Inf. Process. Lett. **91**(5), 233–244 (2004)
63. Pugh, W.W.: The omega test: a fast and practical integer programming algorithm for dependence analysis. In: SC, pp. 4–13. ACM (1991)
64. Rodríguez-Carbonell, E., Kapur, D.: Automatic generation of polynomial loop. In: ISSAC, pp. 266–273. ACM (2004)
65. Sadeghimanesh, A., England, M.: An SMT solver for non-linear real arithmetic inside maple. ACM Commun. Comput. Algebra **56**(2), 76–79 (2022)
66. Sankaranarayanan, S., Sipma, H., Manna, Z.: Non-linear loop invariant generation using Gröbner bases. In: POPL, pp. 318–329. ACM (2004)
67. Seidenberg, A.: A new decision method for elementary algebra. Ann. Math. **60**(2), 365–374 (1954)
68. Strzebonski, A.W.: Solving systems of strict polynomial inequalities. J. Symb. Comput. **29**(3), 471–480 (2000)
69. Strzebonski, A.W.: Cylindrical algebraic decomposition using validated numerics. J. Symb. Comput. **41**(9), 1021–1038 (2006)
70. Sun, Y., Fu, H., Chatterjee, K., Goharshady, A.K.: Automated tail bound analysis for probabilistic recurrence relations. In: CAV, vol. 13966, pp. 16–39 (2023)
71. Tarski, A.: A Decision Method for Elementary Algebra and Geometry: Prepared for Publication with the Assistance of J.C.C. McKinsey. RAND Corporation, Santa Monica, CA (1951)
72. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 10.2) (2023). https://www.sagemath.org
73. Wang, J., Sun, Y., Fu, H., Chatterjee, K., Goharshady, A.K.: Quantitative analysis of assertion violations in probabilistic programs. In: PLDI, pp. 1171–1186. ACM (2021)
74. Wang, P., Fu, H., Goharshady, A.K., Chatterjee, K., Qin, X., Shi, W.: Cost analysis of nondeterministic probabilistic programs. In: PLDI, pp. 204–220 (2019)
75. Weispfenning, V.: Quantifier elimination for real algebra - the cubic case. In: ISSAC, pp. 258–263. ACM (1994)
76. Weispfenning, V.: Quantifier elimination for real algebra - the quadratic case and beyond. Appl. Algebra Eng. Commun. Comput. **8**(2), 85–101 (1997)
77. Weispfenning, V.: Semilinear motion planning in REDLOG. Appl. Algebra Eng. Commun. Comput. **12**(6), 455–475 (2001)

78. Winkler, T., Junges, S., Pérez, G.A., Katoen, J.: On the complexity of reachability in parametric Markov decision processes. In: CONCUR, pp. 14:1–14:17 (2019)
79. Xue, B., Fränzle, M., Zhan, N.: Under-approximating reach sets for polynomial continuous systems. In: HSCC, pp. 51–60. ACM (2018)