Verified Quadratic Virtual Substitution for Real Arithmetic*

Matias Scharager[1 [0000−0002−1668−7359]], Katherine Cordwell[1 [0000−0002−9336−6006]], Stefan Mitsch[1 [0000−0002−3194−9759]], and André Platzer[1 [0000−0001−7238−5710]]

Carnegie Mellon University, Pittsburgh PA 15213, USA
{mscharag,kcordwel,smitsch,aplatzer}@cs.cmu.edu

Abstract. This paper presents a formally verified quantifier elimination (QE) algorithm for first-order real arithmetic by linear and quadratic virtual substitution (VS) in Isabelle/HOL. The Tarski-Seidenberg theorem established that the first-order logic of real arithmetic is decidable by QE. However, in practice, QE algorithms are highly complicated and often combine multiple methods for performance. VS is a practically successful method for QE that targets formulas with low-degree polynomials. To our knowledge, this is the first work to formalize VS for quadratic real arithmetic including inequalities. The proofs necessitate various contributions to the existing multivariate polynomial libraries in Isabelle/HOL. Our framework is modularized and easily expandable (to facilitate integrating future optimizations), and could serve as a basis for developing practical general-purpose QE algorithms. Further, as our formalization is designed with practicality in mind, we export our development to SML and test the resulting code on 378 benchmarks from the literature, comparing to Redlog, Z3, Wolfram Engine, and SMT-RAT. This identified inconsistencies in some tools, underscoring the significance of a verified approach for the intricacies of real arithmetic.

Keywords: virtual substitution · quantifier elimination · real-closed fields · theorem proving.

1 Introduction

Quantifier elimination (QE) is the process of transforming quantified formulas into logically equivalent quantifier-free formulas. In this paper, we consider QE for the first-order logic of real arithmetic ($\text{FOL}_{\mathbb{R}}$), so quantifiers range over the real numbers. The Tarski-Seidenberg theorem proves that QE is admissible for

* This material is based upon work supported by the National Science Foundation under Grant No. CNS-1739629, a National Science Foundation Graduate Research Fellowship under Grants Nos. DGE1252522 and DGE1745016, and by the AFOSR under grant number FA9550-16-1-0288. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or of AFOSR.

the theory of real-closed fields [25, 29]. Real quantified statements arise in a number of application domains, including geometry, chemistry, life sciences, and the verification of cyber-physical systems (CPS) [27]. Many of the applications which require QE are safety-critical [18,19]; accordingly, it is crucial to have both efficient and formally verified support for QE to trust the resulting decisions.

Unfortunately, QE algorithms are quite intricate, which makes it difficult to formally verify their correctness. In practice, this necessitates the use of unverified tools. For example, the theorem prover KeYmaera X [8], which is designed to formally verify models of CPS (such as planes and surgical robots) uses Mathematica/Wolfram Engine and/or Z3 as blackbox solvers for QE. While these are admirable tools, they are unverified, and their use introduces a weak link [7] into what would otherwise be a (fully verified [1]) trustworthy proof.

To help fill this gap, we formally verify linear and quadratic *virtual substitution (VS)* due to Weispfenning [30, 32], which focuses on QE for a quantified variable x occurring in polynomials $f(x)$ of at most degree 2 in x, although variations [12,31] handle higher degree polynomials. Linear and quadratic VS are of practical significance. They serve to improve QE [17] and SMT tools and are the basis of the experimentally successful [28] Redlog solver [6]. To our knowledge, ours is the first formally verified algorithm for VS with quadratic inequalities.

As we focus on correct and practical VS, we export our verified Isabelle/HOL code to SML for experimentation. We test our exported formalization of the equality VS algorithm (Sect. 3.2) and of the general VS algorithm (Sect. 3.3). We compare to four tools that implement real QE: Redlog, SMT-RAT [5], Z3 [14], and Wolfram Engine. With 304 examples, we solve more examples than SMT-RAT in quantifier elimination mode (solves 191) and come close to virtual substitution in Wolfram Engine (solves 322). The remaining tools solve almost all examples; this is to be expected given that those tools have been optimized and fine-tuned (some for decades) and use efficient general-purpose fallback QE algorithms when VS does not succeed. However, as we found 137 inconsistencies in other solvers, it is significant that ours is the only VS implementation with associated correctness proofs (assuming the orthogonal challenge of correct code generation from Isabelle [10]).

Our formalization is approximately 23,000 lines in Isabelle/HOL and is available on the Archive of Formal Proofs (AFP) [22].

2 Related Work

The fastest known QE algorithm is Cylindrical Algebraic Decomposition (CAD) [4], which has not yet been fully formally verified. There are few general-purpose formally verified QE algorithms, and there appears to exist a tradeoff between the practicality of an algorithm and the ease of formalization. Mahboubi and Cohen verified Tarski's original QE algorithm [3] and McLaughlin and Harrison have a proof-producing QE procedure based on Cohen-Hörmander [13]; unfortunately, Tarski's algorithm and Cohen-Hörmander both have non-elementary complexity, which limits the computational feasibility of these formalizations.

There has already been some work on formally verified VS: Nipkow [16] formally verified a VS procedure for *linear* equations and inequalities. The building blocks of $\text{FOL}_{\mathbb{R}}$ formulas, or "atoms," in Nipkow's work only allow for linear polynomials $\sum_i a_i x_i \sim c$, where $\sim \in \{=, <\}$, the x_i's are quantified variables and c and the a_i's are real numbers. These restrictions ensure that linear QE can always be performed, and they also simplify the substitution procedure and associated proofs. Nipkow additionally provides a generic framework that can be applied to several different kinds of atoms (each new atom requires implementing several new code theorems in order to create an exportable algorithm). While this is an excellent theoretical framework—we utilize several similar constructs in our formulation—we create an independent formalization that is specific to general $\text{FOL}_{\mathbb{R}}$ formulas, as our main focus is to provide an efficient algorithm in this domain. Specializing to one type of atom allows us to implement several optimizations, such as our modified DNF algorithm, which would be unwieldy to develop in a generic setting.

Chaieb [2] extends Nipkow's work to quadratic equalities. His formalizations are not publicly available, but he generously provided us with the code. While this was helpful for reference, we chose to build on a newer Isabelle/HOL polynomial library, and we focus on VS as an exportable standalone procedure, whereas Chaieb intrinsically links VS with an auxiliary QE procedure.

Other related work includes some unverified solvers. For example, some work has been done in constraint solving with falsification: RSolver [21] was designed for hybrid systems verification and can find concrete counterexamples for fully quantified existential QE problems on *compact* domains. dReal [9] is based on similar ideas and slightly relaxes the notion of satisfiability to δ-satisfiability. Constraint solving has also been considered in SMT-solving with Z3's nlsat [11], which uses CDCL to decide systems of nonlinear inequalities and equations.

3 The Virtual Substitution Algorithm

Informally (and broadly) speaking, VS discretizes the QE problem by solving for the roots of one or more low-degree polynomials $f_1(x), \ldots, f_n(x)$. VS focuses on these roots and the intervals around them to identify and substitute appropriate representative "sample points" for x into the rest of the formula. However, these sample points may contain fractions, square roots, and/or other extensions of the logical language, and so they must be substituted "virtually": That is, VS creates a formula *in* $\textit{FOL}_{\mathbb{R}}$ *proper* that models the behavior of the direct substitution, which would be outside of $\text{FOL}_{\mathbb{R}}$. VS applies in two cases: an equality case and a general case. We formalize both, and discuss each in turn.

Remark 1. The VS algorithms need to work for *multivariate* polynomials. But as the VS correctness proofs show the equivalence is true for every real value of the free variables, they often implicitly treat all but one variable as having fixed (but arbitrary) real values. That is why most correctness lemmas (but not the top-level algorithmic constructions) suffice for *univariate* polynomials with *real coefficients*. We utilize this trick to simplify difficult proofs for general VS.

3.1 Example

Example 1. Say that we want to perform QE on the formula $\exists x.(x^2 = 2 \wedge xy^2 + 2y + 1 = 0)$. One might notice that $x^2 = 2$ forces $x = \pm\sqrt{2}$ and accordingly wish to substitute. Direct substitution yields the following expression: $(\sqrt{2}y^2+2y+1 = 0 \vee -\sqrt{2}y^2 + 2y + 1 = 0)$. However, as its mention of the $\sqrt{\cdot}$ operator makes it an illegal $\mathrm{FOL}_{\mathbb{R}}$ formula, we will need some further tricks.

Cleverly, VS finds that $\sqrt{2}y^2+2y+1 = 0$ is logically equivalent to $y^2 \cdot (2y+1) \leq 0 \wedge 2y^4 - (2y+1)^2 = 0$, which is a $\mathrm{FOL}_{\mathbb{R}}$ formula[1]. Similarly, VS identifies a $\mathrm{FOL}_{\mathbb{R}}$ formula that is logically equivalent to $-\sqrt{2}y^2 + 2y + 1 = 0$. Then, VS returns the following quantifier-free $\mathrm{FOL}_{\mathbb{R}}$ formula which is logically equivalent to $\exists x.(x^2 = 2 \wedge xy^2 + 2y + 1 = 0)$:

$$\big((y^2 \cdot (2y+1) \leq 0 \wedge 2y^4 - (2y+1)^2 = 0) \\ \vee (-y^2 \cdot (2y+1) \leq 0 \wedge 2y^4 - (2y+1)^2 = 0)\big).$$

Remark 2. If instead our starting formula were $\exists x.\exists y.(x^2 = 2 \wedge xy^2 + 2y + 1 = 0)$, where now y is quantified, then (following the same method as above) VS would identify the following logically equivalent $\mathrm{FOL}_{\mathbb{R}}$ formula with fewer variables:

$$\exists y.\big((y^2 \cdot (2y+1) \leq 0 \wedge 2y^4 - (2y+1)^2 = 0) \\ \vee (-y^2 \cdot (2y+1) \leq 0 \wedge 2y^4 - (2y+1)^2 = 0)\big). \tag{1}$$

Unfortunately, here we are left with a quantified formula with no linear or quadratic equations or inequalities. As we are thus outside of the fragment of $\mathrm{FOL}_{\mathbb{R}}$ that standard VS applies to, at this point we would want to outsource (1) to a general-purpose QE algorithm (like CAD) to eliminate the quantifier on y.

Example 1 was relatively simple, because it involved a quadratic equation with constant coefficients for x. However, nothing in our reasoning was limited to constant coefficients: To perform QE on $\exists x.(x^2 = c \wedge xy^2 + 2y + 1 = 0)$, where c is a polynomial in the variable z, we could handle substituting $x = \pm\sqrt{c}$ in the exact same way as for $x = \pm\sqrt{2}$, but the answer must distinguish the case of $c \geq 0$ symbolically. More difficult is the generalization to inequalities, which seemingly require uncountably infinitely many values to be virtually substituted. We first turn to the general equality case, and then discuss inequalities.

3.2 Equality Virtual Substitution Algorithm

Let a, b and c be arbitrary polynomials with real coefficients that do not mention the variable x. Consider the formula $\exists x.(ax^2 + bx + c = 0 \wedge F)$. There are three possible cases: Either $a \neq 0$, or $a = 0$ and b is nonzero, or all of a, b, c are zero

[1] Notice that if $y = 0$, then both $\sqrt{2}y^2 + 2y + 1 = 0$ and $y^2 \cdot (2y+1) \leq 0 \wedge 2y^4 - (2y+1)^2 = 0$ are false. If instead $y \neq 0$, then $\sqrt{2}y^2 + 2y + 1 = 0$ is true exactly when $\sqrt{2} = -(2y+1)/y^2$, or exactly when $-(2y+1)/y^2 \geq 0 \wedge 2y^4 - (2y+1)^2 = 0$, which is logically equivalent to $y^2 \cdot (2y+1) \leq 0 \wedge 2y^4 - (2y+1)^2 = 0$, as desired.

(so $ax^2 + bx + c = 0$ is uninformative). Letting F_x^r denote the substitution of $x = r$ for x in F, and solving for the roots of $ax^2 + bx + c$, we have the following:

$$\begin{aligned}
&\exists x.(ax^2 + bx + c = 0 \wedge F) \longleftrightarrow \\
&\Big((a = 0 \wedge b = 0 \wedge c = 0 \wedge \exists x.F) \vee \\
&\quad (a = 0 \wedge b \neq 0 \wedge F_x^{-c/b}) \vee \\
&\quad (a \neq 0 \wedge b^2 - 4ac \geq 0 \wedge (F_x^{(-b+\sqrt{b^2-4ac})/(2a)} \vee F_x^{(-b-\sqrt{b^2-4ac})/(2a)}))\Big).
\end{aligned}$$

Conditions such as $b^2 - 4ac \geq 0$ are needed to ensure $(-b \pm \sqrt{b^2 - 4ac})/(2a)$ are well-defined; these are symbolic formulas unless a, b, c are concrete numbers.

Similarly as in Example 1, if we were to substitute $F_x^{-c/b}$, $F_x^{(-b+\sqrt{b^2-4ac})/(2a)}$, and $F_x^{(-b-\sqrt{b^2-4ac})/(2a)}$ directly (for polynomials a, b, and c that do not involve x), the resulting formula would no longer be in $\mathrm{FOL}_{\mathbb{R}}$. Instead, VS avoids directly dividing polynomials or taking square roots with equivalent rewritings in $\mathrm{FOL}_{\mathbb{R}}$. This involves two procedures: one for fractions, and one for square roots.

To virtually substitute a fraction p/q of polynomials where $q \neq 0$ into the atom $\sum_{i=0}^{n} a_i x^i \sim 0$, where $\sim \in \{=, <, \leq, \neq\}$ and each a_i is an arbitrary polynomial expression not involving x, it suffices to normalize the denominator of the LHS, with the caveat that we must not flip the direction of the inequality for $<$ and $\leq$ atoms by normalizing by a value that might be negative. When n is even, $q^n \geq 0$ under any possible valuation, so normalizing by q^n does not flip the inequality. Alternatively, if n is odd, $q^{n+1} \geq 0$. We formalize this in our `linear_substitution` function (see [23, Appendix A.1]).

Next, we consider substituting $x = \sqrt{c}$ into an atom $\sum_{i=0}^{n} a_i x^i \sim 0$, where c is an arbitrary polynomial expression not involving x that satisfies $c \geq 0$, each a_i is an arbitrary polynomial expression not involving x, and $\sim \in \{=, <, \leq, \neq\}$. Its direct substitution can be separated out into even and odd exponents:

$$\sum_{i=0}^{n} a_i \cdot (\sqrt{c})^i = \sum_{i=0}^{n/2} a_{2i} c^i + \sum_{i=0}^{n/2} a_{2i+1} c^i \sqrt{c}$$

Now our polynomial has the form $A + B\sqrt{c}$, where A and B and c are symbolic polynomial expressions not involving x. Then, we have the following cases:

$$\begin{aligned}
&A + B\sqrt{c} = 0 \longleftrightarrow AB \leq 0 \wedge A^2 - B^2 c = 0 \\
&A + B\sqrt{c} < 0 \longleftrightarrow (A < 0 \wedge B^2 c - A^2 < 0) \vee (B \leq 0 \wedge (A < 0 \vee A^2 - B^2 c < 0)) \\
&A + B\sqrt{c} \leq 0 \longleftrightarrow (A \leq 0 \wedge B^2 c - A^2 \leq 0) \vee (B \leq 0 \wedge A^2 - B^2 c \leq 0) \\
&A + B\sqrt{c} \neq 0 \longleftrightarrow -AB < 0 \vee A^2 - B^2 c \neq 0
\end{aligned}$$

The equivalences for $=$ and $\neq$ atoms are derived from the observation that if $B \neq 0$, $A + B\sqrt{c} = 0$ can be solved to find $\sqrt{c} = -A/B$, which holds iff $A^2 = B^2 c$ and $-A/B \geq 0$. The inequality cases involve casework to determine

when polynomial A is negative and dominates $B\sqrt{c}$ as $A^2 > B^2c$, and when B is negative and $B\sqrt{c}$ dominates A as $B^2c > A$. We formalize the VS procedure for quadratic roots in `quadratic_sub` (see [23, Appendix A.2]).

3.3 General Virtual Substitution Algorithm

As we have seen, QE very naturally leads to finitely many cases (discretizes) for formulas that involve quadratic equality atoms (we call this the *equality case*). The VS algorithm for the *general case*, which also handles inequality atoms, is more involved, because, unlike equalities, inequalities may have uncountably many solutions. General VS only directly applies to a very specific fragment of $\text{FOL}_{\mathbb{R}}$ formulas: conjunctions of polynomials that are at most quadratic in the variable of interest. However, we can extend general VS to apply to more formulas with the help of a disjunctive normal form (DNF) transformation.

As a simple example, consider the formula $\exists x.(p < 0 \land q < 0)$, where p and q are the univariate quadratic polynomials (in variable x) depicted in Fig. 1. Noting that the roots of p and q cannot possibly satisfy the strict inequalities, we partition the number line into ranges in between these zeros.

We recognize a key property: In each of the ranges between the roots of p, q, the signs of both p and q do not change. Since the ranges cover all roots of p, q, the truth value of the formula at a single point in a range is representative of the truth value of the formula on the entire range. To discretize the QE problem, we need only pick one sample point for each range.

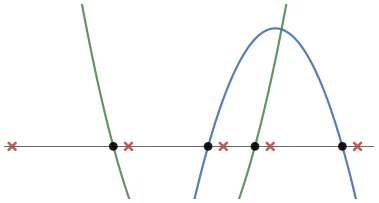


Fig. 1. Two quadratics, their roots (black dots) and off-roots (red x's)

However, we want to pick appropriate sample points *for any* possible p and q. The points we pick as representatives are called the off-roots, which occur ϵ units away from the roots, where $\epsilon > 0$ is arbitrarily small. We additionally need a representative for the leftmost range, which we represent with the point $-\infty$, where $-\infty$ is arbitrarily negative. Of course, we cannot directly substitute ϵ and $-\infty$: they are not real numbers! However, we can *virtually* substitute them.

Negative Infinity. Given any formula F, the VS of $-\infty$ should satisfy the equivalence $F_x^{-\infty} \longleftrightarrow \exists y.\ \forall x{<}y.\ F(x)$ (where y does not occur in F). Intuitively, this says that $-\infty$ acts as if it is arbitrarily negative (so less than the x component of all roots of the polynomials in F) and captures information for the leftmost range on the real number line in any valuation of the non-x variables.

If formula $\exists y.\ \forall x{<}y.\ ax^2 + bx + c = 0$ is true, where a, b, c are polynomials that do not involve x, then $ax^2 + bx + c = 0$ holds at infinitely many x; since nonzero polynomials have finitely many roots, this can only happen if ax^2+bx+c is the zero polynomial in x, i.e., it holds that:

$$(ax^2 + bx + c = 0)_x^{-\infty} \longleftrightarrow a = 0 \land b = 0 \land c = 0 \tag{2}$$

The negation of (2) captures the behavior of $\neq$ atoms. For $<$ atoms, note that the sign value at $-\infty$ is dominated by the leading coefficient, so:

$$(ax^2+bx+c<0)_x^{-\infty} \longleftrightarrow a<0 \vee (a=0 \wedge (b>0 \vee (b=0 \wedge c<0)))$$

Finally, $(ax^2+bx+c\leq 0)_x^{-\infty} \longleftrightarrow (ax^2+bx+c=0)_x^{-\infty} \vee (ax^2+bx+c<0)_x^{-\infty}$.

In Isabelle/HOL, we formalize that our virtual substitution of $-\infty$ satisfies the desired equivalence (on $\mathbb{R}$ using Remark 1) in the following lemma:

```
lemma infinity_evalUni: shows "(∃y. ∀x<y. aEvalUni At x) =
  (evalUni (substNegInfinityUni At) x)"
```

To explain this lemma, we need to take a slight detour and discuss a few structural details of our framework (which is discussed in greater detail in Sect. 4). The datatype `atomUni` contains a triple of real numbers (which represent the coefficients of a univariate quadratic polynomial) and a sign condition:

```
datatype atomUni = LessUni "real*real*real" | EqUni "real*real*real"
  | LeqUni "real*real*real" | NeqUni "real*real*real"
```

The `aEvalUni` function has type `atomUni ⇒ real ⇒ bool`; that is, it takes a sign condition with a triple of real numbers (a,b,c) and a real number x and evaluates whether ax^2+bx+c satisfies the sign condition. The `evalUni` function has type `atomUni fmUni ⇒ real ⇒ bool`, where an `atomUni fmUni` is a formula that involves conjunctions and disjunctions of elements of type `atomUni` (and "True" and "False"). That is, the `evalUni` function takes such a formula and a real number and evaluates whether the formula is true at the real number. Thus, `infinity_evalUni` states that, given `At` of type `atomUni`, with tuple (a,b,c) and sign condition $\sim \in \{<,=,\leq,\neq\}$, $\texttt{At}_x^{-\infty}$ holds iff $\exists y.\forall x{<}y.ax^2+bx+c \sim 0$. This captures the desired equivalence.

Infinitesimals. Given arbitrary r (not containing x), VS for $r+\epsilon$ for variable x should capture the equivalence $F_x^{r+\epsilon} \longleftrightarrow \exists y{>}r.\forall x{\in}(r,y].\ F(x)$, where F does not contain y. Intuitively, this says that (in any valuation of the non-x variables) $r{+}\epsilon$ captures information for the interval between r and the next greatest x-root.

For $=$ and $\neq$ atoms, we proceed in the same manner as we did with $-\infty$, as $(r,y]$ contains infinitely many points and only the zero polynomial has infinitely many solutions. As before, $\leq$ atoms turn into disjunctions of the inequality and equality representations at $r+\epsilon$. We are left only to consider $<$ atoms.

Consider $(p{<}0)_x^{r+\epsilon}$ where $p=ax^2+bx+c$ with polynomials a,b,c not containing x, and an arbitrary r not containing x. Notice that if $(p{<}0)_x^r$, then because polynomials are continuous, we can choose a small enough y so that $\forall x{\in}(r,y].\ p{<}0$. If instead $(p=0)_x^r$, then consider the partial derivative of p evaluated at r. If $\frac{\partial p}{\partial x}(r)$ is negative, then $\exists y{>}r.\forall x{\in}(r,y].\ p{<}0$ holds, because p is decreasing in x locally after $x{=}r$. If $\frac{\partial p}{\partial x}(r)$ is positive, then $\exists y{>}r.\forall x{\in}(r,y].\ p{<}0$ *does not* hold, because p is increasing in x after $x{=}r$. If $\frac{\partial p}{\partial x}(r)$ is zero, then to ascertain whether $\exists y{>}r.\forall x{\in}(r,y].\ p{<}0$, we will need to check higher derivatives.

This pattern forms the following recurrence, with the base case $(p < 0)_x^{r+\epsilon} = (p < 0)_x^r$ for polynomials p of degree zero:

$$(p < 0)_x^{r+\epsilon} \stackrel{\text{def}}{=} (p < 0)_x^r \vee \left((p = 0)_x^r \wedge ((\partial p/\partial x) < 0)_x^{r+\epsilon}\right)$$

We use the VS algorithm from Section 3.2 to characterize $(p < 0)_x^r$ and $(p = 0)_x^r$.

In Isabelle/HOL, we show that given a quadratic root r, the virtual substitution of $r + \epsilon$ satisfies the desired equivalence in the following theorem (on $\mathbb{R}$ using Remark 1; we have an analogous lemma for linear roots r):

```
lemma infinitesimal_quad:
  fixes A B C D:: "real"
  assumes "D≠0"
  assumes "C≥0"
  shows "(∃ y::real>((A+B * sqrt(C))/(D)).
      ∀ x::real ∈{((A+B * sqrt(C))/(D))<..y}. aEvalUni At x)
      = (evalUni (substInfinitesimalQuadraticUni A B C D At) x)"
```

Note that `{r<..y}` in Isabelle stands for the range $(r, y]$. This says that, given `At` of type `atomUni`, with tuple (a, b, c) and sign condition $\sim \in \{<, =, \leq, \neq\}$, $\texttt{At}_x^{r+\epsilon}$ holds iff $\exists y > r. \forall x \in (r, y]. ax^2 + bx + c \sim 0$, which is the desired equivalence.

The General VS Theorem. Now that we have explained virtually substituting $-\infty$ and infinitesimals, we are ready to state the general VS theorem.

Let F be a formula of the following shape, where each a_i, b_i, c_i, and d_i is a polynomial that is at most quadratic in variable x:

$$F = \left(\bigwedge a_i = 0\right) \wedge \left(\bigwedge b_i < 0\right) \wedge \left(\bigwedge c_i \leq 0\right) \wedge \left(\bigwedge d_i \neq 0\right).$$

Let $R(p)$ denote the set of symbolic expressions of the form $(g_1 + g_2\sqrt{g_3})/g_4$ that, as in Sect. 3.2, are roots of the polynomial p in x, where the g_i's are polynomials not involving x. For the zero polynomial, let $R(0) = \emptyset$. Note that, as in Sect. 3.2, the g_i's come with certain well-definedness checks that we retain implicitly in the construction (for example, $g_4 \neq 0$ and $g_3 \geq 0$). We now define:

$$A = \bigcup R(a_i) \quad B = \bigcup R(b_i) \quad C = \bigcup R(c_i) \quad D = \bigcup R(d_i)$$

Then we obtain the following QE equivalence, where for simplicity we elide the relevant crucial well-definedness checks (cross-reference [19, Theorem 21.1]):

$$(\exists x.F) \longleftrightarrow F_x^{-\infty} \vee \bigvee_{r \in A \cup C} F_x^r \vee \bigvee_{r \in B \cup C \cup D} F_x^{r+\epsilon} \tag{3}$$

Intuitively, this formula states that if there is a particular x that satisfies F, then it must be the case that x is one of the equality roots from $A \cup C$, or that x falls in one of the particular ranges (including $-\infty$ as a range) obtained by partitioning the number line by the roots in $B \cup C \cup D$.

Equation (3) can be optimized further by eliding C from the off-roots:

$$(\exists x.F) \longleftrightarrow F^{-\infty} \vee \bigvee_{r \in A \cup C} F_x^r \vee \bigvee_{r \in B \cup D} F_x^{r+\epsilon}. \tag{4}$$

Intuitively, this optimization holds because polynomials are continuous. More precisely, if F has the shape $F = (p \leq 0 \wedge G)$, and if r is an x-root of p, then r already satisfies $p \leq 0$ in any valuation of the non-x variables, so including $r + \epsilon$ as a sample point on account of $p \leq 0$ is redundant. It is possible that G contains some atom $q < 0$ or $q \neq 0$ where r is an x-root of q. In this case, $r + \epsilon$ will already be a sample point on account of q, and we do not need to add it in on account of p. Alternatively, if G does not contain such a q, then, in any valuation of the non-x variables, it is impossible for G to be satisfied by $r + \epsilon$ and not r, meaning that it is redundant to include $r + \epsilon$ as a sample point on account of G.

The general QE theorem is proved in Isabelle/HOL as the following, using Remark 1 to restrict to the univariate case and avoid well-definedness formulas:

```
theorem general_qe:
  defines "R ≡ {(=), (<), (≤), (≠)}"
  assumes "∀rel∈R. finite (Atoms rel)"
  defines "F ≡ (λx. ∀rel∈R. ∀(a,b,c)∈(Atoms rel). rel (a*x^2+b*x+c) 0)"
  defines "Fε ≡ (λr. ∀rel∈R. ∀(a,b,c)∈(Atoms rel). ∃y>r. ∀x∈{r<..y}.
    rel (a*x^2+b*x+c) 0)"
  defines "F_inf ≡ (∀rel∈R. ∀(a,b,c)∈(Atoms rel). ∃x. ∀y<x.
    rel (a*y^2+b*y+c) 0)"
  defines "roots ≡ (λ(a,b,c).
     if a=0 ∧ b≠0 then {-c/b} else
     if a≠0 ∧ b^2-4*a*c≥0 then {(-b+sqrt(b^2-4*a*c))/(2*a)}
        ∪ {(-b-sqrt(b^2-4*a*c))/(2*a)} else {})"
  shows "(∃x. F(x)) = (F_inf ∨
            (∃r∈⋃(roots ‘ (Atoms (=) ∪ Atoms (≤))). F r) ∨
            (∃r∈⋃(roots ‘ (Atoms (<) ∪ Atoms (≠))). Fε r))"
```

Here, ‘ is the Isabelle/HOL syntax for mapping a function over a set. This theorem says that if a finite-length formula `F` is of the requisite shape, then there exists an x satisfying `F` iff `F` is satisfied at $-\infty$ (captured by `F_inf`), or there is a root `r` of one of the $=$ or $\leq$ atoms where `F r` holds, or if there is a root `r` of one of the $<$ or $\neq$ atoms where `Fε r` holds. The proof is quite lengthy and involves a significant amount of casework; however, because we are working with univariate polynomials thanks to Remark 1, this casework mostly reduces to arithmetic computations and basic real analysis for univariate polynomials, and some of what we need, such as properties of discriminants and continuity properties of polynomials, is already formalized in Isabelle/HOL's standard library.

3.4 Top Level Algorithms

We develop several top-level algorithms that perform these VS procedures on multivariate polynomials; these are described in more detail in [23, Appendix

B]. Crucially, each features its own proof of correctness. For example, for the `VSEquality` algorithm, which performs equality VS repeatedly, we have:

```
theorem VSEquality_eval: "∀xs. eval (VSEquality φ) xs = eval φ xs"
```

Here, the `eval` function expresses the truth value of the (multivariate) input formula given a valuation `xs`, represented as a list of real numbers. Since we quantify over all possible valuations and express that they are the same before and after running the algorithm, we prove the soundness of `VSEquality`. The correctness of this theorem only relies on Isabelle/HOL's trusted core.

As our algorithms are general enough to handle formulas with high degree polynomials where VS does not apply, we cannot assert that the result is quantifier free (it might not be). To demonstrate the practical usefulness of these algorithms, we export our code to SML and experimentally show that these algorithms solve many benchmarks. The code exports rely on the correctness of Isabelle/HOL's code export, which ongoing work is attempting to establish [10].

4 Framework

We turn to a discussion of our framework, which is designed with two key goals in mind: First, perform VS as many times as possible on any given formula. Second, reduce unwieldy multivariate proofs to more manageable univariate ones.

4.1 Representation of Formulas

We define our type for formulas in the canonical datatype `fm`:

```
datatype (atoms: 'a) fm =  TrueF | FalseF | Atom 'a
  | And "'a fm" "'a fm" | Or "'a fm" "'a fm" | Neg "'a fm"
  | ExQ "'a fm" | AllQ "'a fm" | ExN "nat" "'a fm" | AllN "nat" "'a fm"
```

As in Nipkow's previous work [16], we use De Bruijn indices to express the variables: That is, the 0th variable represents the innermost quantifier, and variables greater than the number of quantifiers represent the free variables.

We have two constructors for each type of quantifier: `ExQ F` (resp. `AllQ F`) indicates a single existential (universal) quantifier, and `ExN n F` (resp. `AllN n F`) represents a *block* of `n` existential (universal) quantifiers. These representations are interchangeable and converted back and forth in our algorithm; we include the block representation for variable ordering heuristics (see [23, Appendix C.3]).

We utilize the multivariate polynomial library [26] to define our atoms:

```
datatype atom = Less "real mpoly" | Eq "real mpoly" | Leq "real mpoly"
  | Neq "real mpoly"
```

Each atom is normalized without loss of generality, so that the atom `Less p` means $p < 0$, `Eq p` means $p = 0$, and so on.

For example, the $\mathrm{FOL}_{\mathbb{R}}$ formula $\forall x.((\exists y.xa = y^2b) \wedge \neg(\forall y.5x^2 \leq y))$ is represented in our framework as follows, where `Const n` represents the constant $n \in \mathbb{R}$, and `Var i` represents the ith variable:

```
AllQ (And (ExQ (Atom (Eq (Var 1 * Var 2 - (Var 0)^2 * Var 3))))
          (Neg (AllQ (Atom (Leq (Const 5 * (Var 1)^2 - Var 0)))))).
```

Note that we could restrict ourselves to the $\top, \neg, \vee, \exists$ connectives and normalize $\leq$ and $\neq$ atoms to combinations of $<$ and $=$ atoms, and we could still express all of $\mathrm{FOL}_{\mathbb{R}}$. We avoid this for two reasons: because it would linearly increase the size of the formula, and because we want to handle $\leq$ atoms in the optimized way discussed in Sect. 3.3 (see (4)). We do, however, allow for the normalization of $p = q$ into $p - q = 0$. This does not affect the size of the formula, and can afford simplifications: For example, $x^3 + x^2 + x + 1 = x^3$ becomes $x^2 + x + 1 = 0$.

4.2 Modified Disjunctive Normal Form

Nipkow's prior work [16] avoided incurring cases where linear VS does not apply by constraining atoms to be linear. In order to develop a general-purpose VS method which can be used, e.g., as a preprocessing method for CAD, we must reason about cases where VS fails to perform QE for a specific quantifier, and still continue the execution of the algorithm to the remaining quantifiers to simplify the formula as much as possible. To help with this, we implement a modified disjunctive normal form (DNF) that allows expressions to involve quantifiers.

Contextual Awareness. Let us analyze how to increase the informational content in a formula with respect to a quantified variable of interest.

Say we wish to perform VS to eliminate variable x in the formula $\exists x.F$, where F is not necessarily quantifier free. In linear time, we remove all negations from the formula by converting it into negation normal form. We can then normalize $\exists x.F$ into the following form, where the $A_{n,i}$'s are (quantifier-free) atoms:

$$\exists x. \bigvee_n \Big(\bigwedge_i A_{n,i} \wedge \bigwedge_j (\forall y. F_{n,j}) \wedge \bigwedge_k (\exists z. F_{n,k}) \Big).$$

This normalization procedure is similar to standard DNF, as it handles quantified formulas as if they were atomic formulas. We can distribute the existential quantifier across the disjuncts, which results in the equivalent formula:

$$\bigvee_n \exists x. \Big(\bigwedge_i A_{n,i} \wedge \bigwedge_j (\forall y. F_{n,j}) \wedge \bigwedge_k (\exists z. F_{n,k}) \Big). \tag{5}$$

Now we run the VS algorithm, i.e. the input to VS is a conjunction of atomic formulas and quantified formulas in the shape of (5). Notice that if equality VS applies to atom $A_{n,i}$, then the relevant roots can be substituted into the quantified formulas $F_{n,j}$ and $F_{n,k}$, but roots from $F_{n,j}$ or $F_{n,k}$ cannot be substituted into $A_{n,i}$ since they feature quantified variables which are undefined in the broader context. So, our informational content is greatest when the number of $A_{n,i}$ atoms is maximized and the sizes of the $F_{n,j}$ and $F_{n,k}$ are minimized.

Innermost Quantifier Elimination. The innermost quantifier has an associated formula which is entirely quantifier free (and thus has no $F_{n,j}$ and $F_{n,k}$). As such, we opt to perform VS recursively, starting with the innermost quantifier and moving outwards, hoping that VS is successful and the quantifier-free property is maintained. This is not always optimal. Consider the following formula:

$$\exists x.(x = 0 \wedge \exists y.\ xy^3 + y = 0).$$

If we attempt to perform quadratic VS on the innermost y quantifier, it is cubic and will fail. However, performing VS on the x quantifier first fixes $x = 0$, which converts the cubic $xy^3+y = 0$ equality into the linear $y = 0$. So, an (unoptimized) run of inside-out VS would produce $\exists y.y = 0$, and we could completely resolve the QE query by running VS again.

Reaching Under Quantifiers. We would like to recover usable information from the $F_{n,k}$ formulas to increase the informational content going into our QE algorithm. It would be ideal if we could "reach underneath" the existential binders and "pull out" the atoms from the formulas. We can achieve this through a series of transformations. Let k range from 0 to K_n. If we pull out each existential quantifier one by one, we get the following formula, which is equivalent to formula (5):

$$\bigvee_n \exists z_0. \cdots \exists z_{K_n}.\exists x.\Big(\bigwedge_i A_{n,i} \wedge \bigwedge_j (\forall y.F_{n,j}) \wedge \bigwedge_k F_{n,k}\Big)$$

This works because the rest of the conjuncts do not mention the quantified variable z_k and adjacent existential quantifiers can be swapped freely (without changing the logical meaning of the formula).

We can then recursively unravel the formulas $F_{n,k}$, moving as many existential quantifiers as possible to the front. Our implementation does this via a bottom-up procedure, starting underneath the innermost existential quantifier and building upwards, normalizing the formula into the form:

$$\bigvee_n \exists z_0. \cdots \exists z_{K_n}.\exists x.\Big(\bigwedge_i A_{n,i} \wedge \bigwedge_j \forall y.F_{n,j}\Big)$$

On paper, these transformations are simple as they involve named quantified variables; however, because our implementation uses a locally nameless form for quantifiers with DeBruijn indices, shifting an existential quantifier requires a "lifting" procedure $A\uparrow$ which increments all the variable indices in A by one. This allows for the following conversion: $A \wedge \exists z.F \longleftrightarrow \exists z.((A\uparrow) \wedge F)$.

4.3 Logical Evaluation

Our proofs show that the input formula and the output formula (after VS) are *logically equivalent*, i.e., have the same truth value under any valuation. This

needs a method of "plugging in" the real-valued valuation into the variables of the polynomials. Towards this, we define the `eval` function, which accumulates new values into the valuation as we go underneath quantifiers, and the `aEval` function, which homomorphically evaluates a polynomial at a valuation.

When proving correctness, we focus our attention on one quantifier at a time. By Remark 1, correctness of general VS follows when considering a formula F with a single quantifier, where F contains only polynomials of at most degree two (otherwise general VS does not apply). With these restrictions, we can substitute a valuation into the non-quantified variables, transforming multivariate polynomials into univariate polynomials. For example, let a, b, and c be arbitrary multivariate polynomials that do not mention variable x. Let $\hat{p} = \gamma(p)$ denote the evaluation of polynomial p at valuation γ ($\hat{p}$ is a real number). We obtain the following conversion between multivariate and univariate polynomials:

$$\textbf{eval}\ (ax^2 + bx + c = 0)\ \gamma \longleftrightarrow \textbf{evalUni}\ (\hat{a}x^2 + \hat{b}x + \hat{c} = 0)\ \hat{x}$$

As such, we develop an alternative VS algorithm for univariate polynomials, where atoms are represented as triples of real-valued coefficients (as seen in Sect. 3.3), and show that under this specific valuation, the multivariate output is equivalent per valuation with the output of the univariate case. Thus, we finish the proof of the multivariate case by lifting the proof for the univariate case.

4.4 Polynomial Contributions

We build on the polynomials library [26], which was designed to support executable multivariate polynomial operations. This choice naturally comes with trade-offs, and a number of functions and lemmas that we needed were missing from the library. For example, we needed an efficient way to isolate the coefficient of a variable within a polynomial, which we define in the `isolate_variable_sparse` function. The following particularly critical lemma rewrites a multivariate polynomial in $\mathbb{R}[a_1, \ldots, a_n, x]$ as a nested polynomial $\mathbb{R}[a_1, \ldots, a_n][x]$, i.e., a univariate polynomial in x with coefficients that are polynomials in $\mathbb{R}[a_1, \ldots, a_n]$:

```
lemma sum_over_degree: "(p :: real mpoly)
  = (∑ i≤degree p x. isolate_variable_sparse p x i * Var x^i)"
```

This is needed rather frequently within VS, as we often seek to re-express polynomials with respect to a single quantified variable of interest, and although it is mathematically quite obvious, its verification was somewhat involved.

Additionally, to utilize the variables within polynomials as DeBruijn indices, we implemented various lifting and substitution operations. These include the `liftPoly` and `lowerPoly` variable reindexing functions. These and other contributions to the polynomials library are discussed in [23, Appendix B.4].

5 Experiments

The benchmark suite consists of 378 QE problems in category CADE09 collected from 94 examples [20], and category Economics with 45 QE problems [15].

CADE09 and Economics examples were converted into decision problems, powers were flattened to multiplications, and CADE09 were additionally rewritten to avoid polynomial division. For sanity checking, we also negated the CADE09 examples [20]. We run on commodity hardware.[2] The benchmark examples, as well as all scripts to rerun the experiments are in [24].

Tools. We compare the performance of *a*) our VSEquality (**E**), VSGeneral (**G**), VSLucky (**L**), and VSLEG (**LEG**) algorithms [23, Appendix B] to *b*) Redlog [6] snapshots 2021-04-13[3] ($\mathbf{R}_{\lightning}$) and 2021-07-16[4] ($\mathbf{R}_{\checkmark}$, which includes bug fixes for contradictions we reported to Redlog developers), *c*) SMT-RAT 21.05[5] [5] quantifier elimination ($\mathbf{S\text{-}QE}_{\checkmark}$) and satisfiability checking ($\mathbf{S\text{-}SAT}_{\lightning}$), *d*) the SMT solver Z3 4.8.10[6] [14] (**Z3**), and *e*) Wolfram Engine 12.3.1(**W-VS**, **W-QE**). All tools were run in Docker containers on Ubuntu 18.04 with 8GB of memory and 6 CPU cores. Tool syntax translations from SMT-LIB format were done prior to benchmarking: For our VS algorithms, examples were translated to SML data structures and compiled with MLton[7]; as a result, measurements do not include parsing. For W-VS and W-QE, examples were translated into Wolfram syntax, including configuration options restricting QE to quadratic virtual substitution in W-VS. For $\text{S-QE}_{\checkmark}$, `check-sat` was replaced with `eliminate-quantifiers`.

Results. Each example has a timeout of 30s. Figure 2 summarizes the performance on the CADE09 and Economics examples in terms of the cumulative time needed to solve (return "true", "false", "sat", or "unsat") the fastest n problems with a logarithmic time axis: more problems solved and a flatter curve is better.

Wolfram Engine solves all problems in the CADE09 category, closely trailed by Redlog, Z3. The near constant computation time offset of Redlog in comparison to Z3, SMT-RAT, and Wolfram Engine may be attributable to the additional step of entering an SMT REPL. Our verified VSEquality (E), VSGeneral (G), VSLucky (L), and VSLEG (LEG) algorithms rank in performance between the basic quantifier elimination implementation in SMT-RAT ($\text{S-QE}_{\checkmark}$), virtual substitution in Wolfram Engine (W-VS), full SMT approaches ($\text{S-SAT}_{\lightning}$, Z3), and combined virtual substitution plus CAD implementations ($\text{R}_{\checkmark}$, W-QE). The reduced startup time of our algorithms is attributable to the omitted parsing step. Overall, VSEquality and VSLucky solve examples fast, but the wider applicability of VSGeneral and VSLEG allows them to solve considerably more examples. Though we have already implemented a number of optimizations for VS [23, Appendix C] we do not expect to outperform prior tools at this stage, as many of them have been optimized over a period of many years.

A comparison of duration per problem is in Fig. 3. Though there is considerable overlap between VSEquality, VSGeneral, and VSLucky, mutually exclusive

[2] MacBook Pro 2019 with 2.6GHz Intel Core i7 (model 9750H) and 32GB memory (2667MHz DDR4 SDRAM).

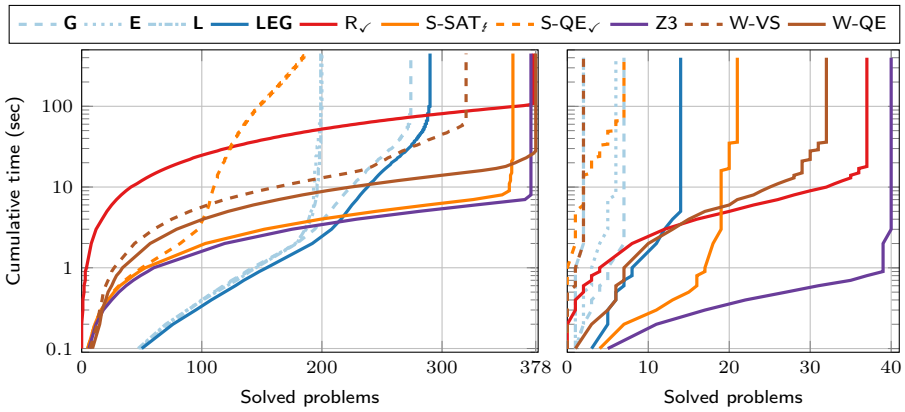
[3] https://sourceforge.net/projects/reduce-algebra/files/snapshot_2021-04-13/

[4] https://sourceforge.net/projects/reduce-algebra/files/snapshot_2021-07-16/

[5] https://github.com/ths-rwth/smtrat/releases/tag/21.05

[6] https://github.com/Z3Prover/z3/releases/tag/z3-4.8.10

[7] http://mlton.org/



(a) CADE09 (378 examples) (b) Economics (45 examples)

Fig. 2. Cumulative time to solve fastest n problems (flatter and more is better)

sets of solved examples (and considerable performance differences on a number of examples) foreshadow the performance achievable with the combined VSLEG algorithm.

Contradictions. In Fig. 4, we compare the CADE09 results to the results on negated CADE09 examples to highlight *contradictions* between answers (e.g., both A and $\neg A$ are claimed to be true). Wolfram Engine and Z3 answer consistently on both formula sets, and solve (almost) all examples. Redlog, the main VS implementation, in R$_\lightning$ and previous versions in general does not perform well on the negated formulas and reports 96 contradictory answers; the contradictory examples were shared with the developers and triggered several bug fixes that are now available in R$_\checkmark$ (no contradictions found on the benchmark set). SMT-RAT performs better than R$_\lightning$ on the negated formulas, but in satisfiability mode contradicts itself on 41 examples by silently ignoring quantifiers in the input; in quantifier elimination mode, SMT-RAT supports quantifiers and does not report contradictions, but SMT-RAT then incurs a significant performance

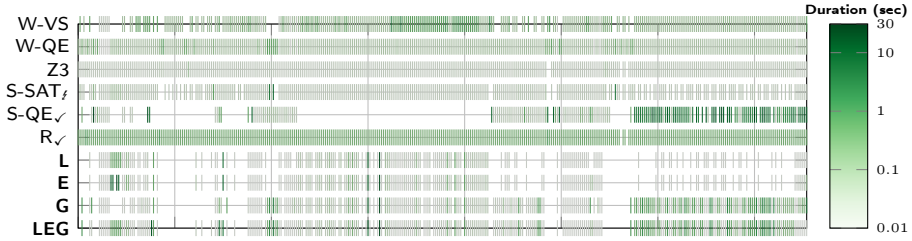


Fig. 3. CADE09 duration per problem (color indicates duration, lighter is better)

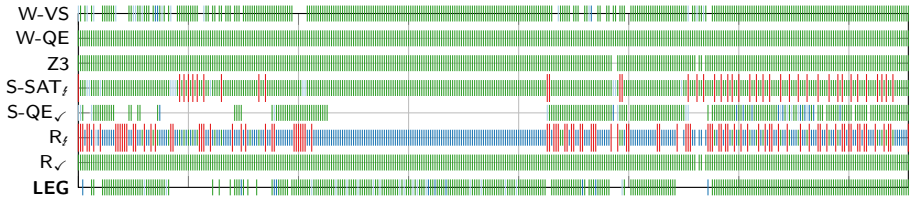


Fig. 4. CADE09 consistency comparison between original and negated formula: color indicates discrepancies within tools (green (■): answer on original and negated formula agree, dark-blue (■): only original solved, light-blue (■): only negated solved, red+long (■): **contradictory** answers (both formulas unsat/proved or both sat/disproved), empty: both timeout/unknown)

loss (S-SAT$_{\not\checkmark}$ reports 359 answers while S-QE$_{\checkmark}$ only solves 187). No contradictions were found across tools, i.e., whenever a tool's answers were consistent internally, the answers agreed with those of other tools. Our VSLEG algorithm has similar performance for proving and disproving in terms of absolute number of solved examples, but combining proving and disproving would still solve more examples than just one question individually (as for S-QE$_{\checkmark}$ and W-VS).

In summary, the performance of our verified virtual substitution QE on the benchmark set is encouraging. The number of solved examples is close to other VS implementations (304 examples by our VSLEG vs. 322 by W-VS) and the cumulative solving time reveals that the majority of examples are solved fast.

6 Conclusion and Future Work

We verify linear and quadratic virtual substitution for real arithmetic; our algorithms are *provably correct* up to Isabelle/HOL's trusted core and code export. Developing practical verified VS in Isabelle/HOL required significant low-level improvements and extensions to Isabelle's multivariate polynomials library. Our extensive experiments both reveal the benefits of our current optimizations and indicate room for future improvements. Further optimizations to the polynomial libraries, such as efficient coefficient lookup for polynomials using red black trees, would be welcome. Expanding our framework to handle formulas that involve polynomial division would also be of practical significance. Continuing to develop our formalization with such improvements is of especial significance given that our experiments found long-standing errors in existing unverified real arithmetic tools. This demonstrates that, even if verification were not a virtue in and of itself, real arithmetic is so subtle that formal verification is the best way toward an implementation that is both useful and correct in practice.

Acknowledgment. We wish to thank Fabian Immler for his substantial contributions at CMU to the polynomial theories of Isabelle/HOL and regret that his current industry position precludes our ability to include him as a coauthor. Thank you also to the anonymous FM reviewers for their useful feedback.

References

1. Bohrer, B., Rahli, V., Vukotic, I., Völp, M., Platzer, A.: Formally verified differential dynamic logic. In: Bertot, Y., Vafeiadis, V. (eds.) CPP. pp. 208–221. ACM, New York (2017). doi: 10.1145/3018610.3018616
2. Chaieb, A.: Automated methods for formal proofs in simple arithmetics and algebra. Ph.D. thesis, Technische Universität München (2008), https://mediatum.ub.tum.de/doc/649541/649541.pdf
3. Cohen, C., Mahboubi, A.: Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination. Log. Methods Comput. Sci. **8**(1) (2012). doi: 10.2168/LMCS-8(1:2)2012
4. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Barkhage, H. (ed.) Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer (1975). doi: 10.1007/3-540-07407-4_17
5. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In: Heule, M., Weaver, S.A. (eds.) SAT. LNCS, vol. 9340, pp. 360–368. Springer (2015). doi: 10.1007/978-3-319-24318-4_26
6. Dolzmann, A., Sturm, T.: REDLOG: computer algebra meets computer logic. SIGSAM Bull. **31**(2), 2–9 (1997). doi: 10.1145/261320.261324
7. DurÃąn, A.J., PÃľrez, M., Varona, J.L.: The misfortunes of a trio of mathematicians using computer algebra systems. can we trust in them? Notices of the AMS **61**(10), 1249–1252 (2014). doi: 10.1090/noti1173
8. Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE. LNCS, vol. 9195, pp. 527–538. Springer (2015). doi: 10.1007/978-3-319-21401-6_36
9. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE. LNCS, vol. 7898, pp. 208–214. Springer (2013). doi: 10.1007/978-3-642-38574-2_14
10. Hupel, L., Nipkow, T.: A verified compiler from Isabelle/HOL to CakeML. In: Ahmed, A. (ed.) ESOP. LNCS, vol. 10801, pp. 999–1026. Springer (2018). doi: 10.1007/978-3-319-89884-1_35
11. Jovanovic, D., de Moura, L.M.: Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR. LNCS, vol. 7364, pp. 339–354. Springer (2012). doi: 10.1007/978-3-642-31365-3_27
12. Košta, M.: New concepts for real quantifier elimination by virtual substitution. Ph.D. thesis, Universität des Saarlandes (2016)
13. McLaughlin, S., Harrison, J.: A proof-producing decision procedure for real arithmetic. In: Nieuwenhuis, R. (ed.) CADE. LNCS, vol. 3632, pp. 295–314. Springer (2005). doi: 10.1007/11532231_22
14. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008). doi: 10.1007/978-3-540-78800-3_24
15. Mulligan, C.B., Bradford, R.J., Davenport, J.H., England, M., Tonks, Z.: Quantifier elimination for reasoning in economics. CoRR **abs/1804.10037** (2018), http://arxiv.org/abs/1804.10037
16. Nipkow, T.: Linear quantifier elimination. J. Autom. Reason. **45**(2), 189–212 (2010). doi: 10.1007/s10817-010-9183-0

17. Passmore, G.O.: Combined Decision Procedures for Nonlinear Arithmetics, Real and Complex. Ph.D. thesis, School of Informatics, University of Edinburgh (2011)
18. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010). doi: 10.1007/978-3-642-14509-4
19. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer, Cham (2018). doi: 10.1007/978-3-319-63588-0
20. Platzer, A., Quesel, J.D., Rümmer, P.: Real world verification. In: Schmidt, R.A. (ed.) CADE. LNCS, vol. 5663, pp. 485–501. Springer, Berlin (2009). doi: 10.1007/978-3-642-02959-2_35
21. Ratschan, S., Smaus, J.: Verification-integrated falsification of non-deterministic hybrid systems. In: Cassandras, C.G., Giua, A., Seatzu, C., Zaytoon, J. (eds.) ADHS. IFAC Proceedings Volumes, vol. 39, pp. 371–376. Elsevier (2006). doi: 10.3182/20060607-3-IT-3902.00068
22. Scharager, M., Cordwell, K., Mitsch, S., Platzer, A.: Verified quadratic virtual substitution for real arithmetic. Archive of Formal Proofs (Aug 2021), https://www.isa-afp.org/entries/Virtual_Substitution.html, Formal proof development
23. Scharager, M., Cordwell, K., Mitsch, S., Platzer, A.: Verified quadratic virtual substitution for real arithmetic. CoRR **abs/2105.14183** (2021), http://arxiv.org/abs/2105.14183
24. Scharager, M., Cordwell, K., Mitsch, S., Platzer, A.: Verified quadratic virtual substitution for real arithmetic: Benchmark examples and scripts. Zenodo (2021). doi: 10.5281/zenodo.5189881
25. Seidenberg, A.: A new decision method for elementary algebra. Annals of Math. **60**(2), 365–374 (1954)
26. Sternagel, C., Thiemann, R.: Executable multivariate polynomials. Archive of Formal Proofs (Aug 2010), https://www.isa-afp.org/entries/Polynomials.html, Formal proof development
27. Sturm, T.: A survey of some methods for real quantifier elimination, decision, and satisfiability and their applications. Math. Comput. Sci. **11**(3-4), 483–502 (2017). doi: 10.1007/s11786-017-0319-z
28. Sturm, T.: Thirty years of virtual substitution: Foundations, techniques, applications. In: Kauers, M., Ovchinnikov, A., Schost, É. (eds.) ISSAC. pp. 11–16. ACM (2018). doi: 10.1145/3208976.3209030
29. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. RAND Corporation, Santa Monica, CA (1951)
30. Weispfenning, V.: The complexity of linear problems in fields. J. Symb. Comput. **5**(1/2), 3–27 (1988). doi: 10.1016/S0747-7171(88)80003-8
31. Weispfenning, V.: Quantifier elimination for real algebra - the cubic case. In: MacCallum, M.A.H. (ed.) ISSAC. pp. 258–263. ACM (1994). doi: 10.1145/190347.190425
32. Weispfenning, V.: Quantifier elimination for real algebra - the quadratic case and beyond. Appl. Algebra Eng. Commun. Comput. **8**(2), 85–101 (1997). doi: 10.1007/s002000050055