A First Complete Algorithm for Real Quantifier Elimination in Isabelle/HOL

Katherine Kosaian
Carnegie Mellon University
Pittsburgh, PA, USA
kcordwel@cs.cmu.edu

Yong Kiam Tan
Carnegie Mellon University
Pittsburgh, PA, USA
yongkiat@alumni.cmu.edu

André Platzer
Karlsruhe Institute of Technology
Karlsruhe, Germany
platzer@kit.edu

Abstract

We formalize a multivariate quantifier elimination (QE) algorithm in the theorem prover Isabelle/HOL. Our algorithm is complete, in that it is able to reduce *any* quantified formula in the first-order logic of real arithmetic to a logically equivalent quantifier-free formula. The algorithm we formalize is a hybrid mixture of Tarski's original QE algorithm and the Ben-Or, Kozen, and Reif algorithm, and it is the first complete multivariate QE algorithm formalized in Isabelle/HOL.

CCS Concepts: • **Theory of computation → Logic and verification**.

Keywords: quantifier elimination, theorem proving, real arithmetic, multivariate polynomials

ACM Reference Format:
Katherine Kosaian, Yong Kiam Tan, and André Platzer. 2023. A First Complete Algorithm for Real Quantifier Elimination in Isabelle/HOL. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '23), January 16–17, 2023, Boston, MA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3573105.3575672

1 Introduction

Real arithmetic problems appear in many application domains, including safety-critical application domains, such as the verification of cyber-physical systems (CPS). Very often, these problems involve ∃ and ∀ quantifiers, which pose theoretical and practical computational challenges [14, 35, 41]. The best known way of handling arbitrary quantified statements is with *quantifier elimination (QE)*, which transforms quantified statements into logically equivalent quantifier-free formulas, which are then evaluated. Alfred Tarski [40] proved that the theory of real-closed fields is decidable, by establishing that algorithms to perform quantifier elimination on formulas in the first-order logic of real arithmetic exist; in practice, these algorithms tend to be complicated.

Given the safety-critical nature of real arithmetic questions [34], it is not surprising that considerable attention has been given to formally verifying algorithms for real QE [8, 12, 17, 19, 25, 27, 29–32, 35, 37]. However, while considerable progress has been made on verifying *univariate* QE methods (methods for QE problems that only involve one variable, and so have at most one quantifier) [12, 17, 25, 30, 31], and while a variety of works have focused on verifying *special-purpose* QE methods (that is, methods which target some fragment of multivariate QE problems) [19, 32, 35, 37], only limited progress has been made on verifying *complete* multivariate QE algorithms (i.e., algorithms that are capable of resolving *any* real QE problem). Multivariate QE algorithms are significantly more challenging. Multivariate polynomials are unlike univariate polynomials, because they may have infinitely many roots, their leading coefficients are themselves polynomials and may have zeros, polynomial division is not always unique, and ideal computations use Gröbner bases instead of Euclidian division. Additionally, whereas univariate QE problems only involve a single quantifier and always reduce to True or False, multivariate QE problems can involve nested (alternating) quantifiers and free variables.

To our knowledge, the main published progress on verifying complete multivariate QE algorithms in theorem provers is threefold: first, Mahboubi [27] *implemented* (but did not yet verify) the fastest-known QE algorithm, *cylindrical algebraic decomposition (CAD)* [9] in Coq; second, McLaughlin and Harrison developed a *proof-producing* (but not verified) procedure based on the Cohen-Hörmander algorithm in HOL Light [29]; finally, Cohen and Mahboubi verified Tarski's original QE algorithm in Coq [6, 8]. Unfortunately, both Tarski's original QE algorithm and the Cohen-Hörmander algorithm have non-elementary complexity (i.e. the complexity is not bounded by any tower of powers of two). While

CPP '23, January 16–17, 2023, Boston, MA, USA

ACM ISBN 979-8-4007-0026-2/23/01.
https://doi.org/10.1145/3573105.3575672

McLaughlin and Harrison's procedure can solve simple microbenchmarks, they acknowledge considerable experimental limitations [29].[1] Similarly, Cohen and Mahboubi consider their work to be primarily a theoretical contribution [8].

The dearth of efficient formally-verified support for QE is in part a consequence of the intricacy of QE algorithms. There is arguably a tradeoff [37] between the computational efficiency of an algorithm and the tractability of verification. Most notably, the CAD algorithm is efficient but complex and tremendously difficult to verify; only the significantly simpler univariate case has been fully verified (independently, in Isabelle/HOL [25] and PVS [30]). Further, in order for CAD to realize its full potential for efficiency, many further insights [3, 10, 16, 28] beyond the original development [9] are needed, and improving CAD and algorithms for real QE at large is an active area of research.

The lack of efficient *verified* QE methods is also a consequence of the challenge posed by verification. Working within the formal setting of a theorem prover adds both a considerable layer of rigor but also intricacy, which is why even small progress needs significant effort. For example, Mahboubi [27] discusses the many challenges involved in implementing CAD in Coq—a significantly more arduous and involved task than implementing CAD in an unverified computer algebra system (which also took decades [4, 39]).

In our work, we target a potential *sweet spot* within the tradeoff between complexity and verification amenability [37] by verifying a *complete* multivariate QE algorithm loosely based on the *Ben-Or, Kozen, and Reif (BKR)* algorithm [2] (but presently with less efficiency). The BKR algorithm shares some theoretical similarity to Tarski's original QE algorithm (in that it uses a matrix equation to store sign information for polynomials), but it includes an additional reduction step for greater efficiency. Although the multivariate complexity analysis in the paper describing BKR was flawed [5], rendering its stated bounds inaccurate, this was nevertheless an influential algorithm which was later extended into a number of improved and/or generalized variants with highly compelling parallel complexity bounds, including ones by Renegar [36], Canny [5], and Cucker *et al.* [13]. As prior work [21] has drawn a strong distinction between computational complexity and practical efficiency (with particular attention to Renegar [36]), these complexity bounds will not necessarily translate into immediate practical efficiency. However, a followup work [20] argued for the potential of algorithms with strong theoretical complexity bounds to realize efficiency on fragments of real arithmetic, and these algorithms remain influential.

Our prior work [12] verified the *univariate* case of BKR in Isabelle/HOL; we argue there that BKR is likely more amenable to formalization than CAD, and potentially complementary to CAD. We now extend this development [11, 12] into a multivariate QE algorithm. Our multivariate algorithm is something of a hybrid: it is a mixture of Tarski's original QE algorithm [40] and BKR [2], with insights from Renegar [36]. It currently does not exploit *all* of the reduction from BKR, which limits its efficiency. Thus, like Cohen and Mahboubi [8], we view our contribution as being primarily theoretical *from the perspective of efficiency*. However, we also view our algorithm as being a significant stepping stone towards the BKR algorithm and, eventually, its variants. In particular, it would be of considerable interest to verify a method that more closely realizes the parallel complexity bounds of Renegar [36]. Such a method will naturally take time to develop, and will likely only be realized in stages.

Contributions. (1) Our work is the first complete multivariate QE algorithm formalized in Isabelle/HOL. (2) To our knowledge, it is the first formalized multivariate QE algorithm to include insights from BKR, and it is a first step towards a less complex verified algorithm (e.g. in the style of Renegar [36]), which could ideally complement an eventual formalized algorithm based on CAD. (3) Because much of the source material is either sparsely written (e.g. [2]) or highly mathematical (e.g. [1, 36]), it was not a priori obvious what the formalized algorithm should look like (this formalization barrier is discussed in Sec. 3.1). The rigorous nature of verification forced us to clearly identify the essential building blocks of the algorithm: In our formalization, *all* definitions are mathematically precise and verifiable, and *all* their correctness properties are identified and proved.

The formalization is approximately 8500 lines of code and is available on the Archive of Formal Proofs (AFP) [23]. It includes various advances to Isabelle/HOL's existing libraries, particularly the library for multivariate polynomials, which could help pave the way for future multivariate QE algorithms in Isabelle/HOL.

2 Quantifier Elimination

Our QE algorithm works by eliminating one quantifier at a time. Hence, if we have polynomials in $n+1$ variables, we can consider them as univariate polynomials in a variable of interest with coefficient polynomials in n variables. For example, if x is our variable of interest, then we can treat $3xyz^2 + 6x^2wv + 5xy + 1$ as the following polynomial in x: $(6wv)x^2 + (3yz^2 + 5y)x + 1$. For clarity, and WLOG, we assume throughout this section that our variable of interest is x.

The key component of both multivariate and univariate BKR is a *sign-determination algorithm* which is concerned with finding all *consistent sign assignments* to a set of polynomials $\{q_1, \ldots, q_k\}$. A *sign assignment* is a mapping that assigns each polynomial to a *sign*, i.e. positive, zero, or negative (represented by $1, 0$, and -1). A sign assignment is called *consistent* if it is actually realized at some real point.

[1] This is not only due to the complexity of the Cohen-Hörmander algorithm, but also because proof-producing algorithms are not verified once and for all but, instead, have to produce a new proof of correctness per question, which incurs significant overhead compared to fully verified ones [29, 35].

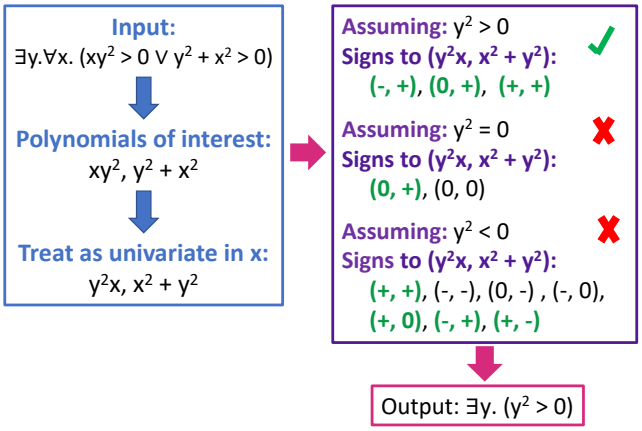

Figure 1. A visual overview of the QE algorithm.

At the heart of the sign-determination algorithm that we formalize is a *matrix equation* that is capable of storing sign information for a set of polynomials in variables $x, y_1, \ldots, y_n$, under a set of assumptions on polynomials in $y_1, \ldots, y_n$. Our overall quantifier elimination algorithm takes a formula and identifies the polynomials that occur in the formula. It then generates a number of matrix equations, each of which captures some sign information for the polynomials, subject to some list of assumptions. Collectively, it is important that the generated matrix equations have exhaustive assumptions—in the sense that for every possible set of assumptions, there is at least one corresponding matrix equation. We call sets of assumptions *branches*. Branches are refined throughout the construction with additional assumptions until each multivariate matrix equation has assumptions that generate a unique matrix equation. Initial branches, which are not fully refined, may still have multiple associated matrix equations.

WLOG, we assume that we are eliminating a $\forall$ quantifier (because $\exists$ quantifiers can be transformed into $\forall$ quantifiers with appropriate negations). We do some initial branching (this is needed to guide the computations of the matrix equations), and for each branch, we check whether *all* of the associated matrix equations describe a sign assignment on our polynomials that satisfies the original formula. We filter our initial branches to pick out the ones that satisfy this property. Finally, we return a disjunction of all assumptions of the initial branches in this filtered list.

Fig. 1 visualizes how this QE algorithm works on an example. We begin with formula $\exists y.\forall x.(xy^2 > 0 \vee y^2 + x^2 > 0)$, where our focus is on eliminating the $\forall x$ quantifier. We first identify the polynomials of interest in this formula and view them as univariate polynomials in x (with coefficients that are polynomials in y): these are y^2x and $x^2 + y^2$. Next, we determine all consistent sign assignments to these polynomials of interest given all *possible*[2] sign assumptions on y^2, where y^2 is significant because it is the *leading coefficient* of y^2x (technically our algorithm will do some additional and unnecessary branching, but for the clarity of this example we focus on the branch on y^2; see Sec. 2.1 for a more in-depth discussion of the branching). Internally, our algorithm performs sign determination using matrix equation constructions (but this is not pictured in the figure). We then pick out the sign assignments that solve our original QE problem—that is, we are looking for one of our polynomials of interest, y^2x or x^2+y^2, to be positive. Signs that satisfy this condition are pictured in green. Then, we filter our branches to find the ones where *every* sign assignment satisfies the original QE problem. This happens only in the branch where y^2 is assumed to be positive. This means that $y^2 > 0$ is logically equivalent to $\forall x.(y^2x > 0 \vee x^2 + y^2 > 0)$, which means that $\exists y.\forall x.(xy^2 > 0 \vee y^2 + x^2 > 0)$ is logically equivalent to $\exists y.(y^2 > 0)$, whose quantifier $\exists y$ can be eliminated further.

If our original QE question was instead $\exists y.\forall x.(xy^2 \geq 0 \vee x^2 + y^2 > 0)$, then both the branch with assumption $y^2 > 0$ and the branch with assumption $y^2 = 0$ would satisfy our QE problem. This means that the disjunction $y^2 > 0 \vee y^2 = 0$ is logically equivalent to $\forall x.(y^2x \geq 0 \vee x^2 + y^2 > 0)$, and so our output in this case would be $\exists y.(y^2 > 0 \vee y^2 = 0)$.

Here it is important to note that there are many logically equivalent outputs to any given QE problem. For example, if our original QE question were $\forall x.((xy^2 = 0 \wedge x^2 + y^2 = 0) \vee (xy^2 = 0 \wedge x^2 + y^2 < 0))$, then two possible correct outputs that are logically equivalent are $y^2 = 0$, and $y^2 < 0 \vee y^2 = 0$. Here, $y^2 = 0$ is the simplest output. While the output of our QE algorithm is always logically correct, it is *not* guaranteed to be in the simplest form. In particular, assumptions for branches that are inconsistent will often be included in the final disjunction, which has no impact on logical correctness, only formula complexity.

We now turn to more detailed descriptions of the sign determination procedure, the multivariate matrix equation, and the full quantifier elimination procedure.

2.1 Sign Determination

Finding sign information for polynomials $q_1, \ldots q_k$ in variables $x, y_1, \ldots, y_n$ is, on the surface, a continuous problem—the most obvious way to determine the sign information would be to evaluate $(q_1, \ldots, q_k)$ on $\mathbb{R}^k$, which is clearly not computationally viable. To account for this, BKR and Renegar reduce the sign-determination problem to a problem with the following format: find sign information for $q_1, \ldots, q_k$ *at the roots* of some cleverly chosen polynomial p. This problem is clearly computationally viable for *univariate* polynomials, because polynomials in one variable only have finitely many roots. It is a (non-obvious) key insight that it is also computationally viable for *multivariate* polynomials [2, 36].

[2]Here, we differ from the BKR algorithm, which would branch on all *consistent* sign assumptions on y^2. That is, we consider a branch where $y^2 < 0$, because this is a possible (but inconsistent) sign assumption: even though y^2 is never negative, our algorithm does not discern this when branching.

Intuitively, the output of the univariate algorithm only depends on the *signs* of the real polynomial coefficients and not on the actual *values* of those coefficients. Thus, the algorithm lifts to the multivariate case by making *sign assumptions* on (multivariate) polynomial coefficients in variables $y_1, \ldots, y_n$.

In our multivariate setting, $p = (\prod q_i) \cdot \frac{\partial}{\partial x}(\prod q_i)$ is chosen for p. To see what makes this particular polynomial useful, consider some valuation ν on $y_1, \ldots, y_n$ (i.e., some assignment of $y_1, \ldots, y_n$ to real values). Let $\nu(f)$ denote the evaluation of polynomial f in valuation ν; note that $\nu(f)$ is univariate in x. Now, the roots of $\nu(p) = (\prod \nu(q_i)) \cdot \frac{\partial}{\partial x}(\prod \nu(q_i)) = (\prod \nu(q_i)) \cdot \frac{d}{dx}(\prod \nu(q_i))$ contain all of the roots of the $\nu(q_i)$'s (since each $\nu(q_i)$ divides $\nu(p)$), as well as sample points from intervals between the roots (by Rolle's theorem [12]). Because these intervals are *sign-invariant*—that is, no $\nu(q_i)$ changes sign in any of these intervals, since no $\nu(q_i)$ can change sign without passing through a root—sign information at a single point within any of these intervals is *representative* of sign information for the entire interval. So, we see that the only intervals which the roots of $\nu(p)$ do not adequately cover are the extreme intervals—the leftmost and rightmost, which lie beyond any of the roots of $\nu(p)$—for which sign information can be computed with a limit calculation on the $\nu(q_i)$'s.[3] So, this polynomial p allows a natural lifting from the univariate QE algorithm to the multivariate case, but the correctness justification needs an extensive covering of the influence of all possibilities for valuation ν.

This is visualized in Fig. 2. Here, we have polynomials $q_1 = y^2x+1$ and $q_2 = yx+1$, so $p = (y^2x+1)(yx+1)(2xy^3+y^2+y)$. For the purposes of illustration, we consider two sample valuations: in ν_1, we set $y = 2$, and in ν_2, we set $y = -1$. As depicted, in both valuations, to find sign information for q_1 and q_2, it suffices to find sign information for q_1 and q_2 at the roots of p and the limit points.

We formalize this procedure for sign determination in the `sign_determination` function. The first input to this function is a list of polynomials `qs` of type `rmpoly`, where `rmpoly` is our abbreviation for `real mpoly poly`. Here, `poly` is Isabelle/HOL's type for univariate polynomials, `mpoly` is the type for multivariate polynomials, and `real` is the type for real numbers, so an `rmpoly` is a univariate polynomial whose coefficients are real multivariate polynomials. Say initially we have polynomials in variables $x, y_1, \ldots, y_n$; then type `rmpoly` arises when we treat those polynomials as being univariate in x with coefficients in $y_1, \ldots, y_n$. Unlike in computer algebra, these polynomials are not restricted to have any particular representation; rather, they are elements of the free term algebra. The next input to `sign_determination` is a list

[3]In the formalization of the univariate case [12], the polynomial p was chosen so as to directly sample from these intervals by using the Cauchy root bound, a mathematical quantity that bounds the roots of a set of univariate polynomials. This followed BKR's original work [2]. However, since the Cauchy root bound is for univariate polynomials only, we must work instead with limit computations as Renegar does [36].

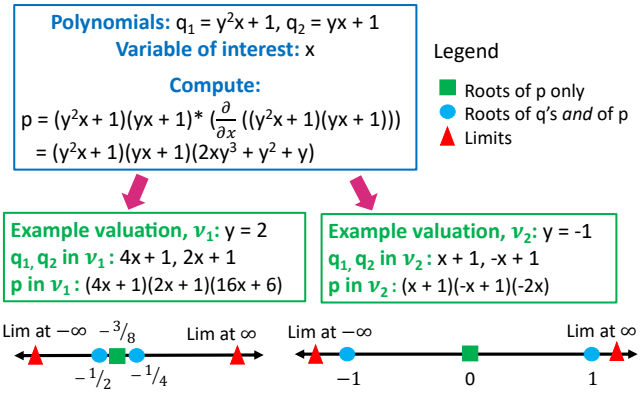

Figure 2. An example of sign determination.

of initial assumptions of type `(real mpoly × rat) list`, which we abbreviate as `assumps`. Here, `rat` is Isabelle/HOL's type for rational numbers, and so each assumption in the list pairs a real multivariate polynomial with an associated rational number that indicates a sign condition on the polynomial (0, 1, or -1). This type is useful in specifying any known sign information on polynomials in $y_1, \ldots, y_n$. The output of `sign_determination` is a list of pairs of assumptions and associated sign assignments to `qs`. Each sign assignment has type `rat list`.[4] The assumptions have type `assumps` (for the same reason as before), and as each assumption may have multiple associated sign assignments, each assumption is paired with a *list* of associated sign assignments, as demonstrated by the `assumps × (rat list list)` type. The output, of type `(assumps × (rat list list)) list`, contains an exhaustive set of assumptions (in order to capture *all* consistent sign assignments for the q_i's).

```
fun sign_determination:: "rmpoly list ⇒ assumps ⇒
(assumps × rat list list) list"
  where "sign_determination qs assumps =
  (let branches =
     lc_assump_generation_list qs assumps in
   concat (map (λbranch. let
      poly_p_branch = poly_p_in_branch branch;
      (pos_limit_branch, neg_limit_branch) =
        limit_points_on_branch branch;
      mat_eq_signs_on_branch = extract_signs
        (calculate_data_assumps_M poly_p_branch
        (snd branch) (fst branch)) in
    map (λ(a, signs).
      (a, pos_limit_branch#neg_limit_branch#signs))
       mat_eq_signs_on_branch) branches))"
```

Here, the `lc_assump_generation_list` function generates an exhaustive list of *possible* branches, `branches`, that contain assumptions on the signs of the leading coefficients of the input polynomials `qs`. An important subtlety is that the

[4]Technically, we could use `int list` for sign assignments, since each member of the sign assignment list is 1, 0, or −1, but as noted elsewhere [12], it is easier to work with `rat list` in the matrix equation construction.

leading coefficient of the polynomial q_i may be different in different branches. For example, the leading coefficient of $(y+1)x^2+yx+2$ is $y+1$ in a branch where $y+1$ is assumed to be nonzero, y in a branch where $y+1$ is zero and y is assumed to be nonzero, and 2 in a branch where both $y+1$ and y are assumed to be zero. To best account for this subtlety, each element of `branches` contains both the generated assumptions (which determine the branch) *and* a list of polynomials which contains a simplified version of the `qs`: to be precise, $q_i = c_1x^{d_1} + \cdots + c_mx^{d_m}$ simplifies to $c_jx^{d_j} + \cdots + c_mx^{d_m}$ iff $c_1, \ldots, c_{j-1}$ are all assumed to be zero and c_j is assumed to be nonzero. For example, given a list of input polynomials $[(y+1)x^2+yx+2, y^2+(y+1)x^5]$, an element of `branches` could be: $([(y+1, 0), (y, 1), (y^2, 1)], [yx+2, y^2+(y+1)x^5])$. The list of assumptions $[(y+1, 0), (y, 1), (y^2, 1)]$ specifies that, in this branch, $y+1$ is assumed to be 0 and y and y^2 are assumed to be positive. Under these assumptions, $(y+1)x^2+yx+2$ simplifies to $yx+2$ and $y^2+(y+1)x^5$ simplifies to $y^2+(y+1)x^5$ (as the purpose of the simplification is to determine the leading coefficient, it is not mission critical to fully simplify $y^2+(y+1)x^5$ to y^2, and our code is not optimized to do so).

Currently, `lc_assump_generation_list` naively generates branches by branching on *all possible* sign assignments to the leading coefficients, rather than on all *consistent* ones as BKR would. Thus, branches with inconsistent assumptions can be generated: for example, the branch $([(y+1, 0), (y, 1), (y^2, -1)], [yx + 2, y^2 + (y + 1)x^5])$ could be generated by the function `lc_assump_generation` despite its inconsistent assumptions (y^2 is assumed to be negative). Additionally, although `lc_assump_generation_list` takes an input list of assumptions, `assumps`, as an argument, it does not enforce consistency of the output branches with `assumps`; however, before splitting on the sign of a polynomial f, it will check whether `assumps` already contains sign information for f.

Branching on the signs of the leading coefficients of the `qs` provides important information for two reasons: First, because these signs are relevant for the matrix equation computation (Sec. 2.2); and second, because knowing the sign of the first non-zero leading coefficient for every q_i allows us to easily compute the signs at the limit points.[5]

The `sign_determination` function maps over `branches`, and for each computes the polynomial $p = (\prod q_i) \cdot \frac{\partial}{\partial x}(\prod q_i)$, stored in `poly_p_branch` (cross reference Fig. 2). Although it would suffice to compute p beforehand, and then simplify it appropriately on each branch given the associated assumptions (for example, in a branch where $y = 0$, $q_1 = y^2x+1$, and $q_2 = yx+1$, the polynomial $p = (y^2x+1)(yx+1)(2xy^3+y^2+y)$ simplifies to $p = 0$), it is more direct to compute p in each branch.[6] That is, given $q_1 = y^2x + 1$, and $q_2 = yx + 1$, if in a given branch we know that $y = 0$, we also know that the leading coefficient of q_1 is 1 and the leading coefficient of q_2 is 1, which means that $q_1 = 1$ and $q_2 = 1$, and so $p = (1 \cdot 1) \cdot (\frac{\partial}{\partial x}(1 \cdot 1)) = 0$.

Next, for each branch, `sign_determination` performs a calculation (formalized in our `limit_points_on_branch` function) to find the signs of `qs` at ∞ and $-\infty$. These are stored in `pos_limit_branch` and `neg_limit_branch`, respectively.

Then, it makes a call to our `calculate_data_assumps_M` function (discussed in Sec. 2.2) to calculate a list of matrix equations for each branch, each of which stores sign information under some assumptions (assumptions in our formalization only accumulate, so the output assumptions contain the original branch's assumptions). It pulls out the assumptions and sign conditions from the matrix equations with the `extract_signs` function, which returns a list of type `(assumps × rat list list) list`. This list is stored in `mat_eq_signs_on_branch`.

Finally, the positive and negative limit sign conditions `pos_limit_branch` and `neg_limit_branch` are preprended to each list of sign conditions calculated with the matrix equations (the # operator in Isabelle/HOL prepends an element to a list), and the resulting list of assumptions and associated sign conditions is returned.

It is now time to discuss the matrix equation.

2.2 The Multivariate Matrix Equation

The multivariate matrix equation, like the univariate matrix equation, is concerned with finding sign information for a set of polynomials $q_1, \ldots, q_n$ at the roots of an auxiliary polynomial p. One advantage of formalizing a multivariate QE algorithm based on BKR and Tarski is that the construction of the multivariate matrix equation is very similar to the construction of the univariate matrix equation.

Thus, to understand the multivariate matrix equation, we first need to consider the construction of the univariate matrix equation. At its core, the univariate matrix equation relies on computing *Tarski queries*, so we start there.

2.2.1 Computing Multivariate Tarski Queries.

Tarski queries are defined as follows:

Definition 2.1. [12] Given *univariate* polynomials p, q with $p \neq 0$, the *Tarski query* $N(p, q)$ is:

$$N(p, q) = \#\{x \in \mathbb{R} \mid p(x) = 0, q(x) > 0\} - \#\{x \in \mathbb{R} \mid p(x) = 0, q(x) < 0\}.$$

These Tarski queries can be computed from the Euclidean remainder sequence that starts with p and $p'q$:

Proposition 2.2. *(Sturm-Tarski Theorem) Let $p \neq 0$ and q be real* univariate *polynomials. Let $p_1 = p$, $p_2 = p'q$, $p_3, \ldots, p_k$*

[5] The sign of q_i at ∞ equals the sign of its leading coefficient, whereas the sign of q_i at $-\infty$ is the sign of its leading coefficient multiplied by $(-1)^{\deg q_i}$, where $\deg q_i$ is the degree of q_i.

[6] Our polynomials do not have any fixed representation, and equality checking is a potentially costly operation. Further, even if two polynomials are not identically equivalent, they may be so under a branch's assumptions (for example, $y^2 + y + 1$ is equivalent to y^2 if $y + 1$ is assumed to be 0).

be the Euclidean remainder sequence of p and $p'q$, where

$$p_i = c_i p_{i+1} - p_{i+2},$$

for $c_i \in \mathbb{R}[x]$ and where $\deg(p_{i+2}) < \deg(p_{i+1})$. Let a_i be the leading coefficient of p_i and let $d_i := \deg(p_i)$. Let $S^+(p,q)$ denote the number of sign changes in the sequence $a_1, \ldots, a_k$, and let $S^-(p,q)$ denote the number of sign changes in the sequence $(-1)^{d_1}a_1 \ldots, (-1)^{d_k}a_k$. Then $N(p,q) = S^-(p,q) - S^+(p,q)$.

This result is from the literature [36, Prop. 8.1] (with an unnecessary assumption removed that is not included in other references [1] or in Isabelle's existing formalization [24] of the Sturm-Tarski theorem). Critically, in the Sturm-Tarski theorem, it is not the values of $a_1, \ldots, a_k$ that matter; rather, it is the signs that matter; this is what enables the multivariate generalization [2].

Consider polynomials $p \neq 0$ and q in x with polynomial coefficients in $y_1, \ldots, y_n$ (i.e., $p, q \in \mathbb{R}[y_1, \ldots, y_n][x]$). Then, we can form Euclidean remainder sequences of p and $p'q$ with respect to x. The Euclidean remainder sequence is no longer unique—instead, there are multiple sequences, each depending on the signs of the coefficients of p and q (as coefficients that are polynomials can have different signs at different points). Once we fix a sequence and find the leading coefficients, we need to consider (by branching) *all* possible sign assignments to those coefficients,[7] and output a list of Tarski queries and the assumptions they are subject to.

For example, if we take polynomials $p = y^2x + 1$ and $q = yx + 1$, then if $y^2 = 0$, then $y = 0$ so $p = q = 1$, and the Euclidean remainder sequence is just 1, and $N(p,q) = 0$.[8] However, if $y \neq 0$, then our Euclidean remainder sequence is $y^2x+1, y^3x+y^2, -(1-y)$, where we have calculated $y^2x+1 = \frac{1}{y} \cdot (y^3x + y^2) + (1 - y)$, using assumption $y \neq 0$ for $\frac{1}{y}$.

Now, continuing the computation of $N(y^2x+1, yx+1)$, we find that the leading coefficients of our Euclidean remainder sequence (assuming $y \neq 0$) are y^2, y^3, and $-(1-y)$. Next, we consider the possible sign assignments to y^2, y^3, and $-(1-y)$. For example, $(+,+,-)$ is one such sign assignment. So, we have Tarski query $N(p,q) = S^-(p,q) - S^+(p,q) = 0 - 1 = -1$ under the assumptions that: $y \neq 0$, $y^2 > 0$, $y^3 > 0$, and $-(1-y) < 0$. Our output for $N(y^2x+1, yx+1)$ would be a list of all the Tarski queries under all possible assumptions. This computation is visualized in Fig. 3 (where, for purposes of space, only two output branches are shown explicitly).

Note that Euclidean remainder sequences for multivariate polynomials sometimes contain fractions. While we could have chosen to work with Euclidean remainder sequences in a *fraction field*, this would require complicated type switching in the formalization. Instead, we use *pseudo-remainder sequences* for multivariate polynomials. Pseudo-remainder sequences are essentially Euclidean remainder sequences for polynomials, but normalized so as not to contain fractions (ours are additionally normalized so as not to affect the result of the Sturm-Tarski computation [25]). We develop pseudo-remainder sequences for multivariate polynomials of type `rmpoly` (currently, our formalization naively branches on the signs of the leading coefficients of the relevant polynomials). Here, we benefit from prior work: The Sturm-Tarski theorem was formalized in Isabelle/HOL by Wenda Li [24]; Li and Paulson later extended this to *bivariate* polynomials [26] using pseudo-remainder sequences, and Li, Passmore, and Paulson also developed univariate Tarski queries with pseudo-remainder sequences [25].

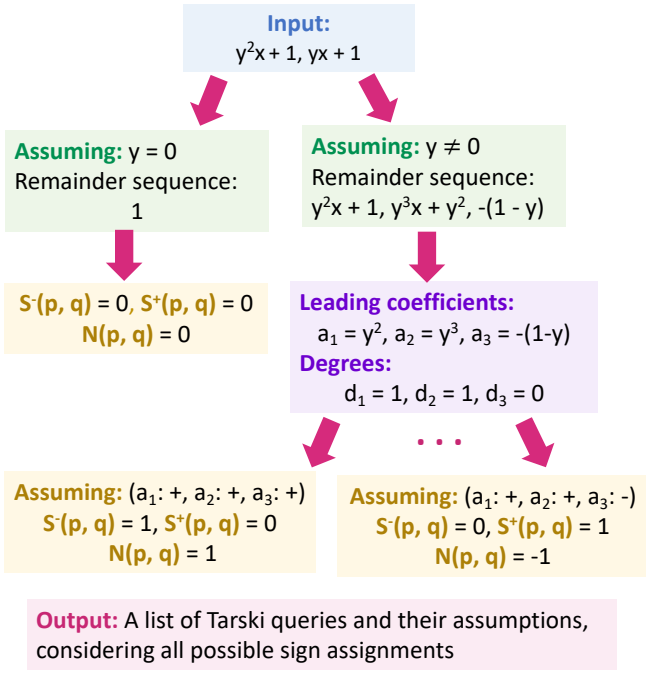

Figure 3. Computing Tarski queries for $p = y^2x+1, q = yx+1$.

Remark. *For self-containedness, we briefly describe pseudo-remainder sequences.* Polynomial pseudo-quotients *(pquo)* and pseudo-remainders *(prem) satisfy this property [15, 25]:*

$$(lead\_coeff\ q)^{(1+deg\ p - deg\ q)} p = pquo(p,q) \cdot q + prem(p,q),$$

where $deg\ prem(p,q) < deg\ q$ or $q = 0$. For example, when considering polynomials $p = yx^2 + 1$ and $q = y^3x + 1$ as univariate polynomials in x, then $pquo(p,q) = y^4x - y$ and $prem(p,q) = y^6 + y$, as $(y^3)^2 p = (y^4x - y)q + (y^6 + y)$ and $deg(y^6 + y) = 0 < deg\ q = 1$. Notice how there are no fractions in pquo or prem, unlike the fractions in the usual Euclidean remainder sequence (assuming $y \neq 0$ for well-definedness).

[7] Full BKR would consider all consistent sign assignments instead. This makes the algorithm highly recursive, which adds a considerable layer of difficulty to its verification.

[8] Technically, our formalization would do more branching than this for two reasons: First, it will branch on $y^2 = 0$, $y^2 > 0$, and (unnecessarily) $y^2 < 0$; and second, because it will not determine that $y^2 = 0$ implies $y = 0$—and so it will not know that $q = 1$ whenever $y^2 = 0$.

We use signed *pseudo-remainder sequences, where $p_1 = p$, $p_2 = p'q$, and $p_3, \ldots, p_k$ satisfy the following equation for a special choice of coefficients s_i, explained below:*

$$p_{i+2} = s_i \cdot prem(p_i, p_{i+1})$$

This sequence is normalized so that, in any valuation, the number of sign changes in the evaluated pseudo-remainder sequence is the same as in the Euclidean remainder sequence for the evaluated polynomials, so that the result of the Sturm-Tarski computation is unaffected by the normalization. For this, we follow the style of [25] and normalize as follows: if $(1 + deg\ p_i - deg\ p_{i+1})$ is even, we multiply $prem(p_i, p_{i+1})$ by $s_i = -1$; else, by $s_i = -lead\_coeff\ p_{i+1}$. To understand this intuitively, note that the pseudo-remainder $prem(p, q)$ effectively normalizes by $(lead\_coeff\ q)^{(1+deg\ p - deg\ q)}$. Then, note that remainder sequences in the Sturm-Tarski theorem always negate prem (cross-reference Proposition 2.2). So, if $(1 + deg\ p - deg\ q)$ is even, we have not changed the sign of prem and we need only negate it. However, if $(1 + deg\ p - deg\ q)$ is odd, we have potentially changed the sign of prem—depending on the sign of $(lead\_coeff\ q)$—so we not only negate prem but also multiply it by $(lead\_coeff\ q)$.

Since QE is concerned with sign information for multiple polynomials simultaneously, it is useful to generalize the notion of Tarski queries to *sets* of polynomials [12] as follows:

Definition 2.3. Given a polynomial p and a list of polynomials $q_1, \ldots, q_n$, let I and J be subsets of $\{1, \ldots, n\}$. Then, the Tarski query $N(I, J)$ with respect to p is

$$N(I,J) = N(p^2 + \left(\Sigma_{i\in I} q_i^2\right), \Pi_{j\in J}\ q_j) =$$
$$\#\{x \in \mathbb{R} \mid p(x) = 0, \forall i \in I.\ q_i(x) = 0, \Pi_{j\in J}\ q_j(x) > 0\} -$$
$$\#\{x \in \mathbb{R} \mid p(x) = 0, \forall i \in I.\ q_i(x) = 0, \Pi_{j\in J}\ q_j(x) < 0\}.$$

The matrix equation determines the signs of $q_1, \ldots, q_n$ at the zeros of p by computing $N(I, J)$ for a representative set of combinations of subsets I, J of $q_1, \ldots, q_n$ (see Sec. 2.2.2).

There are two key lemmas that we prove about multivariate Tarski queries. The first is a soundness lemma showing that the resulting multivariate Tarski queries agree, on every point satisfying the associated assumptions, with what the univariate Tarski query would have been:

```
lemma multiv_tarski_query_correct:
  assumes inset: "(assumps, tarski_query) ∈
    set (construct_NofI_M p acc I J)"
  assumes val: "⋀f n. (f,n) ∈ set assumps ⟹
    satisfies_evaluation val f n"
  shows "tarski_query = construct_NofI_R
    (eval_mpoly_poly val p)
    (eval_mpoly_poly_list val I)
    (eval_mpoly_poly_list val J)"
```

Here, the `construct_NofI_M` function constructs a list of multivariate Tarski queries and the assumptions they are subject to. As input, it takes a polynomial `p`, an initial set of assumptions `acc`, and two lists of polynomials `I` and `J`. Both `p` and all of the polynomials in `I` and `J` have type `rmpoly`, i.e. they are univariate polynomials in x with polynomial coefficients in some variables $y_1, \ldots, y_n$. The `inset` assumption assumes that we have some particular Tarski query `tarski_query` that is subject to the assumptions `assumps`, which are assumptions on polynomials in $y_1, \ldots, y_n$. Now, the `construct_NofI_R` function is the function to compute univariate Tarski queries from our prior work [12], so the conclusion of the lemma is that `tarski_query` is exactly the (unique) univariate Tarski query that would be computed from evaluating `p` and all of the polynomials in `I`, `J` on `val` (using the `eval_mpoly_poly` and `eval_mpoly_poly_list` functions), where `val` is any assignment of real values to $y_1, \ldots y_n$ where the assumptions `assumps` are realized.

The second key lemma is a completeness result:

```
lemma multiv_tarski_queries_complete:
  assumes "⋀f n. (f,n) ∈ set init_assumps ⟹
    satisfies_evaluation val f n"
  shows  "∃ (assumps, tq) ∈
    set (construct_NofI_M p init_assumps I J).
   (∀(p,n)∈set assumps. satisfies_evaluation val p n)"
```

Here, this shows that if initial assumptions `init_assumps` are satisfied by valuation `val`, then there is some resulting assumptions and Tarski query pair `(assumps, tq)` where all final assumptions `assumps` are satisfied by `val`.

Together, these two lemmas give a strong result: the soundness lemma shows that the multivariate results coincide with univariate results in all projections meeting the final assumptions, and the completeness lemma shows that for any projection meeting the initial assumptions, there is some corresponding Tarski query whose associated (final) assumptions are met by the projection. Or, on a more intuitive level, the completeness lemma shows that our function to compute multivariate Tarski queries generates useful output whenever it is given useful input, and the soundness lemma shows that useful output has the desired mathematical meaning.

2.2.2 Using Multivariate Tarski Queries.

The matrix equation connects a vector of information about *possible sign assignments* for a set of multivariate polynomials—i.e., sign assignments that are not necessarily consistent—on the LHS, to a vector of multivariate Tarski queries on the RHS.

The univariate matrix equation is defined as follows, where we closely follow the definition of the univariate matrix equation in our earlier work [12], but adapted to our purposes:[9]

Definition 2.4. Fix univariate polynomials of interest p and $q_1, \ldots, q_k$. Let $\tilde{\Sigma} = \{\tilde{\sigma}_1, \ldots, \tilde{\sigma}_m\}$ be a set of possible sign assignments to $q_1, \ldots, q_k$, and assume $\tilde{\Sigma}$ contains all consistent

[9] The univariate BKR paper [12] follows the matrix equation developed in Ben-Or, Kozen, and Reif's original paper [2], where p is assumed to be coprime with each q_i. Because this assumption no longer makes sense for multivariate polynomials, we use the matrix equation developed by Renegar [36]. While our prior work [12] formalized both styles of matrix equation [11], only the former was discussed at length in the paper.

sign assignments to $q_1, \ldots, q_k$ at the roots of p. Let S be a set of pairs of subsets $(I_1, J_1), \ldots, (I_l, J_l)$ where for all $1 \leq i \leq l$, $I_i \subseteq \{1, \ldots, k\}$ and $J_i \subseteq \{1, \ldots, k\}$. Then the *matrix equation* for $\tilde{\Sigma}$ and S is the relationship $M \cdot w = v$ between the following three entities:

- M, the l-by-m matrix with entries

$$M_{i,j} = \left(\Pi_{\ell \in I_i}(1 - (\tilde{\sigma}_j(q_\ell))^2)\right) \cdot \left(\Pi_{\ell \in J_i} \tilde{\sigma}_j(q_\ell)\right) \in \{-1, 0, 1\}$$

  for $(I_i, J_i) \in S$ and $\tilde{\sigma}_j \in \tilde{\Sigma}$,
- w, the length m vector whose entries count the number of roots of p where $q_1, \ldots, q_k$ has sign assignment $\tilde{\sigma}$, i.e., $w_i = \#\{x \in \mathbb{R} \mid p(x) = 0, \operatorname{sgn}(q_\ell(x)) = \tilde{\sigma}_i(q_\ell)$ for all $1 \leq \ell \leq k\}$,
- v, the length l vector consisting of Tarski queries for the subsets, i.e., $v_i = N(I_i, J_i)$.

Intuitively, as noted by our prior work [12], the meaning of a matrix equation is captured by its associated list of signs and list of (pairs of) subsets. Both the matrix M and the RHS vector v are fully computable from these two lists, and w, which stores information about which possible sign assignments are consistent (sign assignment $\tilde{\sigma}_i$ is consistent iff w_i is nonzero), is calculated as $M^{-1} \cdot v$.

For multivariate polynomials the situation is more complicated. We can still construct a matrix equation for multivariate polynomials—the definition of the matrix M is the same as it was in the univariate setting, but the righthandside vector uses our function to construct a list of Tarski queries for multivariate polynomials. Each RHS vector—and so each matrix equation—comes with an associated list of assumptions which were generated by the multivariate Tarski queries. So, for an input list of multivariate polynomials p and $q_1, \ldots, q_k$, we construct a *list* of multivariate matrix equations that store sign information for these polynomials, subject to certain assumptions on polynomials in one fewer variable.

The overall construction is very similar to that in the univariate case [12]. It proceeds by induction on the number of q's, so that the base case is for a single q. Smaller matrix equations are successively combined and reduced to form the matrix equation for $q_1, \ldots, q_n$. The reduction is what differentiates the matrix equation of BKR from that of Tarski: information for inconsistent sign assignments is removed at appropriate intervals, which decreases the size of the matrix equation. In the univariate case, the size of the matrix equation is bounded by $\#\{x.\ p(x) = 0\})^2$, where $\#\{x.\ p(x) = 0\}$ is the number of roots of the polynomial p. The size of a multivariate matrix equation is bounded by the number of roots of p in a valuation satisfying the associated assumptions. As the univariate reduction step mainly involves computations on the matrix M, which is unchanged in the multivariate setting, it generalizes quite naturally, and so our hybrid algorithm essentially inherits reduction in the matrix equation construction, thus incorporating insights from BKR into our hybrid algorithm.

We formalize our multivariate matrix equation construction in the `calculate_data_assumps_M` function (cross reference Sec. 2.1), and prove the following two key lemmas:

```
lemma multivariate_calculate_data_correct:
  assumes mat_eq: "(assumps, mat_eq) ∈
    set (calculate_data_assumps_M p qs init_assumps)"
  assumes "⋀p n. (p,n) ∈ set assumps ⟹
    satisfies_evaluation val p n"
  assumes "eval_p = eval_mpoly_poly val p"
  assumes "eval_qs = map (eval_mpoly_poly val) qs"
  assumes p_nonzero: "eval_mpoly_poly val p ≠ 0"
  shows "calculate_data_R eval_p eval_qs = mat_eq"
```

This first lemma connects the behavior of our multivariate matrix equation constructor function to the Renegar-style univariate matrix equation function (`calculate_data_R`) formalized in our prior work [12]. That is, on any valuation `val` that satisfies the assumptions `assumps`, the associated multivariate matrix equation `mat_eq`, which finds the consistent sign assignments for `qs` at the zeros of some `p` in the valuation `val`, is equal to the univariate matrix equation that find the consistent sign assignments for `eval_qs` at the zeros of `eval_p` , where `eval_p` is `p` evaluated on `val` and `eval_qs` is `qs` evaluated on `val`. This is a soundness lemma, since it explains that whenever our output is useful, it has the correct mathematical meaning.

```
lemma multivariate_calculate_data_complete:
  assumes "⋀p n. (p,n) ∈ set init_assumps ⟹
    satisfies_evaluation val p n"
  shows  "∃ (assumps, mat_eq) ∈
    set (calculate_data_assumps_M p qs init_assumps).
    (∀ (p,n) ∈ set assumps.
     satisfies_evaluation val p n)"
```

This second lemma shows that when we give logically consistent input assumptions to `calculate_data_assumps_M`, some output with logically consistent assumptions will be generated (i.e., useful input generates useful output). These lemmas are analogous to those discussed for multivariate Tarski queries; taken together, they help us prove key correctness properties of our `elim_forall` method, which serves to eliminate a single universal quantifier. We now turn to a discussion of our top-level QE methods, including `elim_forall`.

2.3 Overall Quantifier Elimination Algorithm

To best explain our formalized QE algorithm, we must first touch on the framework we are working with.

We build on our prior framework [37] that verified (in Isabelle/HOL) the virtual substitution algorithm, an efficient QE method that applies to a low-degree fragment of real arithmetic. This prior development sets up a framework for multivariate QE (including a type for real QE problems and a function to evaluate QE problems at real-valued points); by building on this, we are ultimately able to link together our verified (complete, inefficient) QE method with verified

virtual substitution [37], using this (incomplete but experimentally promising) QE method as a preprocessing step.

Accordingly, we work with formulas of type `atom fm` [37], which have the following grammar:

$$F, G ::= \text{TrueF} \mid \text{FalseF} \mid (\text{Atom}(\text{Eq } p)) \mid (\text{Atom}(\text{Less } p)) \mid (\text{Atom}(\text{Leq } p)) \mid (\text{Atom}(\text{Neq } p)) \mid \text{And } F\ G \mid \text{Or } F\ G \mid \text{Neg } F \mid \text{ExQ } F \mid \text{AllQ } F \mid \text{ExN } n\ F \mid \text{AllN } n\ F,$$

where p is a real polynomial and $n \in \mathbb{N}$. Here, (Atom(Eq p)) captures the relationship $p = 0$, (Atom(Less p)) captures $p < 0$, (Atom(Leq p)) captures $p \leq 0$, and (Atom(Neq p)) captures $p \neq 0$. Further, And F G captures the logical meaning of $F \wedge G$, Or F G captures $F \vee G$, and Neg F captures $\neg F$. Finally, ExQ F indicates that formula F is quantified by an existential quantifier, AllQ F indicates that F is quantified by a universal quantifier, ExN n F indicates that F is quantified by a block of n existential quantifiers, and AllN n F indicates that F is quantified by a block of n universal quantifiers.

In these formulas, variables are represented with de Bruijn indices; `Var 0` is the variable quantified by the innermost quantifier, `Var 1` is the variable quantified by the second innermost quantifier, and so on. We operate on quantifiers inside-out, i.e. we start with the quantifier attached to `Var 0`.

Our `elim_forall` function is designed to eliminate a single ∀ quantifier. It parallels the method visualized in Fig. 1.

```
fun elim_forall:: "atom fm ⇒ atom fm"
  where "elim_forall F = (let
    qs = extract_polys F;
    univ_qs = univariate_in qs 0;
    reindexed_univ_qs = map
      (map_poly (lowerPoly 0 1)) univ_qs;
    initial_data = sign_determination
      reindexed_univ_qs [];
    filtered_data = filter (λ(assumps, signs_list).
    list_all (λ signs.
      lookup_sem_M F (zip qs signs) = (Some True))
    signs_list
    ) initial_data
 in create_disjunction filtered_data)"
```

Here, `extract_polys` finds the polynomials `qs` in our formula `F`, and `univariate_in qs 0` transforms our polynomials `qs` to have the `rmpoly` type (so that they are univariate polynomials in `Var 0`, with coefficients that are multivariate polynomials in bigger variables). The resulting list of polynomials is called `univ_qs`. Then, in `reindexed_univ_qs`, we transform the coefficients of every polynomial in `univ_qs` (which do not contain `Var 0`) by lowering every variable index by 1. This lowering is crucial for finding all possible signs/assumptions pairs for our multivariate polynomial coefficients (cross reference Sec. 2.1), as `sign_determination` expects polynomials in `Var 0`. We then retain all the sign assignments that satisfy our formula of interest, and return a disjunction of the associated assumptions. If our original formula involved polynomials in variables `Var 0`, `Var 1`, …, `Var n`, then, because of the transformation and reindexing, these assumptions will be polynomials in variables `Var 0`, …, `Var (n - 1)`. Our new `Var 0`, which was previously `Var 1`, will correctly match to the new innermost quantifier, which was previously the second innermost quantifier, and so on.

Our top-level QE method, named `qe`, heavily relies on `elim_forall` and `elim_exist` (where `elim_exist F` is defined as `Neg (elim_forall (Neg F))`):

```
fun qe:: "atom fm ⇒ atom fm"
  where
    "qe TrueF = TrueF"
  | "qe FalseF = FalseF"
  | "qe (Atom a) = (Atom a)"
  | "qe (And F1 F2) = And (qe F1) (qe F2)"
  | "qe (Or F1 F2) = Or (qe F1) (qe F2)"
  | "qe (Neg F) = Neg (qe F)"
  | "qe (ExQ F) = elim_exist (qe F)"
  | "qe (AllQ F) = elim_forall (qe F)"
  | "qe (AllN n F) = (elim_forall ^^ n) (qe F)"
  | "qe (ExN n F) = (elim_exist ^^ n) (qe F)"
```

Our top-level correctness theorem says that for any assignment v of the free variables in `F` to real numbers, our original formula `F` has the same truth-value as `qe F`; or, in other words, `F` and `qe F` are logically equivalent:

```
theorem qe_correct:
 fixes F:: "atom fm"
 shows "eval F v = eval (qe F) v"
```

Here, `eval` is the function formalized by Scharager *et al.* [37] to evaluate formulas of type `atom fm` on valuations. This function accounts for the reindexing of free variables that naturally takes place during QE. For example, $\forall x.\ x^2 y \leq 0$ is logically equivalent to $y \leq 0$, but since variables are represented with de Bruijn indices, where the innermost quantifier corresponds with `Var 0`, $\forall x.\ x^2 y \leq 0$ is represented in the `atom fm` type as `AllQ (Leq ((Var 0)^2 · Var 1))` whereas $y \leq 0$ is represented as `Leq (Var 0)`. In `eval`, this subtlety is handled by defining, e.g., `eval (AllQ F) v` as `(∀ x. (eval F (x#v)))`, where `x#v` is the list with head `x` and tail `v`. So, `qe_correct` shows that `F` evaluated on any mapping of free variables to real numbers is equal to `qe F` evaluated on that same mapping, which establishes that `qe` is sound.

We also show that `qe` fully removes quantifiers in the following lemma, where `countQuantifiers` counts the number of existential or universal quantifiers in formula `F`:

```
theorem qe_complete:
  shows "countQuantifiers (qe F) = 0"
```

This result shows that `qe` is complete.

To our knowledge, `qe` is the first sound and complete algorithm for real QE to be formalized in Isabelle/HOL (previous work [25, 32, 37] was sound but not complete). We now turn to some further details regarding our formalization.

3 Formalization Details

Isabelle/HOL is well-suited for us; we not only benefit considerably from the well-developed libraries (including aforementioned prior work [12, 25, 37]), but also from Isabelle/HOL's support for automated proof search in Sledgehammer [33].

However, at the same time, working in the formal setting of Isabelle/HOL poses considerable challenges. In this section, we begin by discussing some of those challenges, followed by some of the high-level proof techniques that helped us succeed in our formalization. We then discuss some useful low-level details regarding our extensions to Isabelle/HOL's multivariate polynomials library. Finally, we discuss our code export and the performance of our algorithm.

3.1 Challenges

Many design decisions for the functions described in Sec. 2 were not initially evident. For example, the need to consistently track assumptions and pass them in as an argument to our functions throughout the calculation of the matrix equation was initially not obvious. At first, we wrote a function that was nearly identical to `calculate_data_assumps_M`, with the one major difference that we did not include `assumps` as an argument to this function. While this function was fully capable of generating a multivariate matrix equation, we soon realized we had made a major mistake when we tried to extend it into a larger QE algorithm. After this, we were careful to always include an argument for assumptions in our functions if it could possibly be applicable, regardless of whether or not it seemed immediately relevant.

The challenge of correctly formalizing the algorithm in Isabelle/HOL is heightened because the precision of formalization sometimes identifies details that were underspecified in the source material. Indeed, BKR's discussion of the multivariate QE algorithm was limited to only two pages and proceeds at a very high level [2]. Renegar [36] is considerably more detailed, but is also written in the style of mathematics, which necessitates significant translation to the level of formalization. For example, the way in which the limit point calculation should be formalized, while entirely obvious in retrospect, did not become clear to us until we fixed a method of branching—and indeed, our initial method of formalizing the limit point calculation, which was agnostic to branching, did not make it into the final code for the algorithm. Of this calculation, Renegar writes the following, in which he uses the notation g_i where we use q_i, and f instead of p [36]: ". . . each consistent sign vector of $\{g_i\}_i$ occurs at some real zero of f except, perhaps, for the sign vectors of points to the right or left of all real zeros of $\prod_i g_i$. However, the latter two consistent sign vectors are trivially determined from the leading coefficients of the polynomials g_i." While this completely describes the mathematical use of the limit point calculations, it took some time to translate it into Isabelle/HOL definitions and proofs.

Finally, a last challenge is that even simple details can become complex in the formalized setting of a theorem prover. For example, working with multivariate polynomials in Isabelle/HOL poses a challenge, as the formal setting requires rigor even for operations that are simple on paper but may become much more involved when formalized. For example, the transformation to treat a multivariate polynomial as univariate in some variable of interest is immediate on paper, but in Isabelle/HOL it is more subtle, precisely because the type of our object is changing: $3xyz^2 + 6x^2wv + 5xy + 1$ has type `real mpoly`, whereas $(6wv)x^2 + (3yz^2 + 5y)x + 1$ has type `rmpoly` (see also Sec. 2.1).

3.2 High Level Proof Techniques

Though treating multivariate polynomials as univariate in some variable of interest poses low-level challenges in our formal setting, it affords significant high-level simplifications. Many of our proofs rely on the technique of universal *projection*—we assume fixed real values for all variables aside from a variable of interest, which lets us work with *truly* univariate polynomials. Projection allows us to connect functions in our multivariate construction to corresponding functions in the univariate construction from our prior work [12]. This works because the multivariate case of the BKR algorithm builds rather directly on the univariate case, making it amenable to formalization, as noted previously [12].

In consequence, each key function involved in the construction of the multivariate matrix equation requires two top-level associated lemmas. The first is a soundness lemma which connects the behavior of the multivariate function to a corresponding univariate function [12] through projection. The second is a completeness lemma which establishes that data for all possible projections is captured by the function for some assumptions. Some examples of these soundness and completeness lemmas are seen in Sec. 2.2 (e.g. the soundness lemma `multiv_tarski_query_correct` and the completeness lemma `multiv_tarski_query_complete`); there are many more in the actual proof development. This proof structure does not seek to closely mimic the (highly mathematical) proofs in the source material [12, 36], but rather to translate the key intuition into a shape which is amenable to formalization.

Our construction and proofs are designed to be modular, and we often rely on induction to prove key properties of helper functions. In particular, we found it very helpful to use custom induction theorems, supplementing those automatically generated by Isabelle/HOL. For example, the `spmods_multiv_aux` function shown (abridged) below computes a list of pseudo-remainder sequences for polynomials `p` and `q` together with corresponding sign assumptions on the leading coefficients of the polynomials in each sequence.

```
function spmods_multiv_aux::
  "rmpoly ⇒ rmpoly ⇒ assumps ⇒
  (assumps × rmpoly list) list" where
```

```
"spmods_multiv_aux p q assumps = (
if q = 0 then [(assumps, [p])]
else
case (lookup_assump_aux (lead_coeff q) assumps) of
None ⇒
  let lcz = spmods_multiv_aux p (one_less_degree q)
            ((lead_coeff q, 0) # assumps) in
  let lcp = spmods_multiv_aux q (mul_pseudo_mod p q)
            ((lead_coeff q, 1) # assumps) in
  let lcn = spmods_multiv_aux q (mul_pseudo_mod p q)
            ((lead_coeff q, -1) # assumps) in
    ... /* combine lcz, lcp, lcn */
| (Some i) ⇒  ... /* two recursive branches */ )"
```

The function branches depending on whether q is the zero polynomial, otherwise, it recurses on the (possible) signs of its leading coefficient `lead_coeff q`. Here, `assumps` specifies a list of assumed input sign conditions, which are checked for assumptions on `lead_coeff q`. Notably, `spmods_multiv_aux` is *not* structurally recursive; its termination uses the fact that, on each recursive call, the degree of the polynomial arguments `one_less_degree q` or `mul_pseudo_mod p q` strictly decreases. For such functions, Isabelle/HOL automatically generates induction theorems, but these theorems lack the usual case-splitting support for structurally recursive functions [42]. The following snippet shows the Isabelle/HOL subgoal (cases) structure that results from applying induction with the generated theorem for `spmods_multiv_aux`.

```
// apply (induct ... spmods_multiv_aux.induct)
Proof outline with cases:
  case (1 p q assumps)
  ...
qed
```

Although `spmods_multiv_aux.induct` can, *in principle*, be used to prove the aforementioned soundness and completeness properties for `spmods_multiv_aux`, we found the proofs tedious in practice because they lack the case structuring benefits of Isabelle/HOL's structured proof language [42]. Instead, we manually prove an alternative induction theorem that mimics the branching structure of `spmods_multiv_aux` (one base case, three branches with recursion). As before, a snippet of the Isabelle/HOL subgoal (cases) structure is shown below (comments illustrate the branching structure).

```
// apply (induct ... spmods_multiv_aux_induct)
Proof outline with cases:
  case (Base p q assumps)
  ... // base case (q = 0)
next
  case (Rec p q assumps)
  ... // lookup_assump_aux returns None
next
  case (Lookup0 p q assumps)
  ... // lookup_assump_aux returns Some 0
next
  case (LookupN0 p q assumps r)
  ... // otherwise
qed
```

Though some manual effort is needed to state and prove `spmods_multiv_aux_induct`, our subsequent, repeated use of this customized induction theorem makes it well worth the initial investment. We expect similar induction theorems to be broadly useful for structuring proofs about non-structural recursive functions, including in other proof assistants. Indeed, manual induction theorems are also used elsewhere in the development, particularly to verify invariant properties of the helper function that underlies the branching function `lc_assump_generation_list` (see Sec. 2.1).

3.3 Library Extensions

We turn to some of our key results for multivariate polynomials and the library extensions they prompted.

As seen in Sec. 3.1, we need a function to convert polynomials of type `real mpoly` to polynomials of type `real mpoly poly`. Eberl and Thiemann formalized one such way of doing this in their `mpoly_to_mpoly_poly` definition [18]. We provide the following alternate definition, which is executable:

```
definition mpoly_to_mpoly_poly_alt :: "nat ⇒
  'a :: comm_ring_1 mpoly ⇒ 'a mpoly poly"
  where "mpoly_to_mpoly_poly_alt x p =
   (∑ i∈{0..MPoly_Type.degree p x} .
     monom (isolate_variable_sparse p x i) i)"
```

This definition applies to multivariate polynomials with coefficients in a commutative ring with unity (denoted by `comm_ring_1`). It relies on the `isolate_variable_sparse` function [38], where `isolate_variable_sparse p x i` finds the coefficient of `x^i` in `p`. For each `i` from 0 to the degree of `x` in `p`, we find this coefficient and construct a monomial of type `poly` with degree `i` and this coefficient. Our final polynomial is the sum of all of these monomials.

We connect our new definition to `mpoly_to_mpoly_poly` in the following lemma:

```
lemma multivar_as_univar:
  shows "mpoly_to_mpoly_poly_alt x p =
    mpoly_to_mpoly_poly x p"
```

This enables a natural interface between Eberl and Thiemann's work [18] and the large and powerful collection of lemmas regarding `isolate_variable_sparse` [38], from which we benefit in the formalization.

We benefit from Eberl and Thiemann's lemmas regarding `mpoly_to_mpoly_poly` in one of our main results regarding polynomials, which is useful in our correctness proof for `elim_forall` (cross reference Sec. 2.3):

```
lemma reindexed_univ_qs_eval:
  assumes "univ_qs = univariate_in qs 0"
  assumes "reindexed_univ_qs =
    map (map_poly (lowerPoly 0 1)) univ_qs"
  shows "map (eval_mpoly (x#xs)) qs =
  (map (λp. (poly p x))
   (map (λq. eval_mpoly_poly xs q) reindexed_univ_qs))"
```

This lemma relates the evaluation of multivariate polynomials, of type `real mpoly`, and multivariate polynomials

treated as univariate polynomials in the variable of interest `Var 0`, of type `rmpoly`. To fully understand it, we must explain a few Isabelle/HOL operators that manipulate multivariate polynomials. Here, `eval_mpoly` is our name for the natural definition of multivariate polynomial evaluation which substitutes real values for variables. Because variables are represented with de Bruijn indices, we can store the values to substitute in a list `L`, where the element of `L` at position 0 is then substituted for `Var 0`, the element of `L` at position 1 is substituted for `Var 1`, and so on. If the length of `L` is shorter than the number of variables, a default value of 0 is substituted for any variables that are not covered by `L`. This definition was implicitly used in prior work [37], but without being explicitly stated and named:

```
definition eval_mpoly:: "real list ⇒ real mpoly ⇒ real"
where "eval_mpoly L p = insertion (nth_default 0 L) p"
```

The `eval_mpoly_poly` function maps `eval_mpoly` over the coefficients of a `real mpoly poly`.

Continuing to unpack the `reindexed_univ_qs_eval` lemma, the `lowerPoly` function is from Scharager *et al.* [37]; here, it serves to reindex variables in multivariate polynomials, so that `lowerPoly 0 1 q` lowers every variable index in `q` by 1. The `univariate_in` operator is our function to perform this multivariate to univariate transformation. Let q_i be the polynomial at the ith index of `qs`, and uq_i be the polynomial at the ith index of `univ_qs`—then the first assumption of `reindexed_univ_qs_eval` says that uq_i is the polynomial that we obtain by treating q_i as univariate in `Var 0`.

Next, the second assumption in `reindexed_univ_qs_eval` says that `reindexed_univ_qs` is the list of polynomials obtained by lowering all variable indices in the *coefficients* of the `univ_qs` by 1. Let us call ruq_i the polynomial at the ith index of `reindexed_univ_qs`. Then, lemma `reindexed_univ_qs_eval` captures the mathematical equivalence of q_i and ruq_i by showing that evaluating q_i on the valuation v = `x#xs` gives the same result as evaluating the *coefficients* of ruq_i on `xs` and then evaluating the resulting univariate polynomial (which now has constant coefficients) on `x`.

The proof of this key lemma required that we first prove the following fundamental extensionality result, which says that if two polynomials `p` and `q` (in n variables) have identical evaluations on $\mathbb{R}^n$, then they are themselves identical:

```
lemma same_evaluations_same_mpoly:
  assumes "(⋀ L. eval_mpoly L p = eval_mpoly L q)"
  shows "p = q"
```

Since real multivariate polynomials are fundamental to many areas of mathematics, it is our hope that our library developments will be useful to others, including in the formalization of other QE algorithms, but also more widely.

3.4 Code Export

We export our multivariate QE algorithm to SML code, which removes overhead and allows us to better test our algorithm on examples.[10] Building on the framework of Scharager *et al.* (by using the same type for QE formulas and the same evaluation function for formulas) makes the connection with the verified virtual substitution algorithm [37] very easy.[11] This means that we are able to retain efficiency [37] on examples that are tractable for virtual substitution.

However, because virtual substitution is *not* a complete QE method (i.e., it is not able to solve all QE problems), the efficiency, or lack thereof, of our (complete) algorithm is still significant. Unfortunately (but not unexpectedly), without the link to virtual substitution, our hybrid multivariate algorithm is not at all efficient; it appears to hang on all but the simplest univariate examples. However, we do not consider our algorithm's present inefficiency to be a fatal flaw, since we envision it as being a (major) stepping stone on the way towards an optimized algorithm. As noted previously [37], unverified computer algebra systems have realized efficient QE in part because many have been extensively optimized over several decades; thus, it is natural that optimized verified algorithms will similarly take time to develop.

While inefficiency is not unexpected given that even Renegar may not realize practical efficiency in its current state [20, 21], at present, we suspect that part of the efficiency bottleneck for our algorithm is the untenable branching in the computation of the multivariate Tarski queries; this can be significantly reduced in the future by implementing an algorithm that more closely follows BKR. We also believe that our algorithm's lack of inherent optimizations is another contributing factor; as one example, we currently branch unnecessarily on the signs of constant coefficients. Further, we are not currently exploiting the algorithm's inherent parallelism. However, it does not make sense to focus on optimizing our algorithm at this stage (optimizations may be brittle). Once the branching reflects the full reduction of BKR, then inefficiencies (such as the unnecessary branching on constant coefficients) should be identified and handled appropriately.

4 Related Work

From a theoretical standpoint, the most closely related work is one by Cyril Cohen, who formalized a sign-determination algorithm with reduction in Coq that, to our understanding, uses the same matrix equation as our algorithm, although the details of his formalization look quite different from

[10] This step requires trusting Isabelle/HOL's code generator in addition to the theorem prover's trusted core. Partial progress has been made on verifying Isabelle's code generator [22].

[11] The top-level correctness theorems for verified virtual substitution [37] have a very similar shape to `qe_correct`, as they state that for each top-level formalized virtual substitution method `V` and valuation v, `eval F` v equals `eval (V F)` v. This makes it easy to verify that, for any valuation v, `eval F` v equals `eval ((qe ∘ V) F)` v.

ours.[12] To our knowledge, he has not yet used this improved sign-determination algorithm for a QE algorithm, and this work is unpublished, but a writeup is available on his webpage [7]. Additionally, because the algorithm we verify is a hybrid between Tarski's QE algorithm and BKR, our work shares some theoretical overlap with Cohen and Mahboubi's formalization of Tarski's algorithm in Coq [6, 8].

From a practical standpoint, we benefit from the well-developed Isabelle/HOL libraries. This includes, of course, our previous verification of univariate BKR [12] and our verification of virtual substitution [37], which have already been discussed at length. Additionally, we build on the formalization of pseudo-remainder sequences (recently made available on the AFP [24]) described by Li, Passmore, and Paulson [25]. Although we formalize our own functions to generate pseudo-remainder sequences, which interface well with our assumptions-based framework (and which are specialized to the `rmpoly` type), we derive insights from Li's code and mimic some of his structure in our functions, adapted appropriately to our purposes. We also benefit from proving a connection between our functions and his.

5 Conclusion and Future Work

We develop and formalize Isabelle/HOL's first complete multivariate quantifier elimination (QE) algorithm for the first-order logic of real arithmetic. Our algorithm mixes ideas from Tarski's original QE algorithm [40] and more efficient algorithms by BKR [2] and Renegar [36]; the formalization requires rigorizing high-level mathematical insights [2, 36]. We realize a number of ideas suggested in our prior work by extending a formalization of univariate BKR [12] to the multivariate case and by building on the framework of Scharager *et al.* [37] in order to link our work with an efficient verified virtual substitution QE algorithm. While our algorithm (on its own) currently has prohibitive inefficiency, its nontrivial library extensions and theoretical interest (including its potential to be extended into variant algorithms that have promising parallel complexity [5, 13, 36]) make it a meaningful contribution.

Future work includes first extending our algorithm to one that realizes the full reduction of BKR [2]. After this, it would be interesting to identify other areas of inefficiency and aggressively optimize. In addition to fine-tuning the branching to avoid splitting on trivial cases (most notably, on constants), one very significant (and challenging) task will be to optimize the computation of the Tarski queries; this was previously noted in the univariate case also [12]. Overall, our contribution lays considerable groundwork for more optimized verified QE algorithms with inherent parallelism.

[12] This is in part because the setup is considerably different: while we extended a univariate QE procedure with reduction into multivariate, Cohen added reduction to an already multivariate sign-determination procedure.

Acknowledgments

We thank the anonymous CPP reviewers for their helpful feedback on the paper.

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, by the AFOSR under grant number FA9550-16-1-0288, by A*STAR, Singapore, and the Alexander von Humboldt Foundation. 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, AFOSR, or A*STAR.

References

[1] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. 2006. *Algorithms in Real Algebraic Geometry* (second ed.). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-33099-2

[2] Michael Ben-Or, Dexter Kozen, and John H. Reif. 1986. The Complexity of Elementary Algebra and Geometry. *J. Comput. Syst. Sci.* 32, 2 (1986), 251–264. https://doi.org/10.1016/0022-0000(86)90029-2

[3] Christopher W. Brown. 2001. Improved Projection for Cylindrical Algebraic Decomposition. *J. Symb. Comput.* 32, 5 (2001), 447–465. https://doi.org/10.1006/jsco.2001.0463

[4] Christopher W. Brown. 2003. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *SIGSAM Bull.* 37, 4 (2003), 97–108. https://doi.org/10.1145/968708.968710

[5] John F. Canny. 1993. Improved Algorithms for Sign Determination and Existential Quantifier Elimination. *Comput. J.* 36, 5 (1993), 409–418. https://doi.org/10.1093/comjnl/36.5.409

[6] Cyril Cohen. 2012. *Formalized algebraic numbers: construction and first-order theory.* Ph. D. Dissertation. École polytechnique. https://perso.crans.org/cohen/papers/thesis.pdf

[7] Cyril Cohen. 2021. Formalization of a sign determination algorithm in real algebraic geometry. (2021). Preprint on webpage at https://hal.inria.fr/hal-03274013/document.

[8] Cyril Cohen and Assia Mahboubi. 2012. Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination. *Log. Methods Comput. Sci.* 8, 1 (2012). https://doi.org/10.2168/LMCS-8(1:2)2012

[9] George E. Collins. 1975. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages (LNCS, Vol. 33)*, H. Barkhage (Ed.). Springer, 134–183. https://doi.org/10.1007/3-540-07407-4_17

[10] George E. Collins and H. Hong. 1991. Partial Cylindrical Algebraic Decomposition for Quantifier Elimination. *J. Symb. Comput.* 12, 3 (1991), 299–328. https://doi.org/10.1016/S0747-7171(08)80152-6

[11] Katherine Cordwell, Yong Kiam Tan, and André Platzer. 2021. The BKR Decision Procedure for Univariate Real Arithmetic. *Archive of Formal Proofs* (April 2021). https://www.isa-afp.org/entries/BenOr_Kozen_Reif.html, Formal proof development.

[12] Katherine Cordwell, Yong Kiam Tan, and André Platzer. 2021. A Verified Decision Procedure for Univariate Real Arithmetic with the BKR Algorithm. In *ITP (LIPIcs, Vol. 193)*, Liron Cohen and Cezary Kaliszyk (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:20. https://doi.org/10.4230/LIPIcs.ITP.2021.14

[13] Felipe Cucker, Hervé Lanneau, Bud Mishra, Paul Pedersen, and Marie-Françoise Roy. 1992. NC Algorithms for Real Algebraic Numbers. *Appl. Algebra Eng. Commun. Comput.* 3 (1992), 79–98. https://doi.org/10.1007/BF01387193

[14] James H. Davenport and Joos Heintz. 1988. Real Quantifier Elimination is Doubly Exponential. *J. Symb. Comput.* 5, 1/2 (1988), 29–35. https:

//doi.org/10.1016/S0747-7171(88)80004-X
[15] Leonardo Mendonça de Moura and Grant Olney Passmore. 2013. Computation in Real Closed Infinitesimal and Transcendental Extensions of the Rationals. In *CADE (LNCS, Vol. 7898)*, Maria Paola Bonacina (Ed.). Springer, 178–192. https://doi.org/10.1007/978-3-642-38574-2_12
[16] Andreas Dolzmann, Andreas Seidl, and Thomas Sturm. 2004. Efficient projection orders for CAD. In *ISSAC*, Jaime Gutierrez (Ed.). ACM, 111–118. https://doi.org/10.1145/1005285.1005303
[17] Manuel Eberl. 2015. A Decision Procedure for Univariate Real Polynomials in Isabelle/HOL. In *CPP*, Xavier Leroy and Alwen Tiu (Eds.). ACM, 75–83. https://doi.org/10.1145/2676724.2693166
[18] Manuel Eberl and René Thiemann. 2021. Factorization of Polynomials with Algebraic Coefficients. *Archive of Formal Proofs* (November 2021). https://isa-afp.org/entries/Factor_Algebraic_Polynomial.html, Formal proof development.
[19] John Harrison. 2007. Verifying Nonlinear Real Formulas Via Sums of Squares. In *TPHOLs (LNCS, Vol. 4732)*, Klaus Schneider and Jens Brandt (Eds.). Springer, 102–118. https://doi.org/10.1007/978-3-540-74591-4_9
[20] Joos Heintz, Marie-Françoise Roy, and Pablo Solernó. 1993. On the Theoretical and Practical Complexity of the Existential Theory of Reals. *Comput. J.* 36, 5 (1993), 427–431. https://doi.org/10.1093/comjnl/36.5.427
[21] Hoon Hong. 1991. *Comparison of Several Decision Algorithms for the Existential Theory of the Reals.* Technical Report. RISC. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.8707
[22] Lars Hupel and Tobias Nipkow. 2018. A Verified Compiler from Isabelle/HOL to CakeML. In *ESOP (LNCS, Vol. 10801)*, Amal Ahmed (Ed.). Springer, 999–1026. https://doi.org/10.1007/978-3-319-89884-1_35
[23] Katherine Kosaian, Yong Kiam Tan, and André Platzer. 2022. A First Complete Algorithm for Real Quantifier Elimination in Isabelle/HOL. *Archive of Formal Proofs* (Dec. 2022). https://www.isa-afp.org/entries/Quantifier_Elimination_Hybrid.html, Formal proof development.
[24] Wenda Li. 2014. The Sturm-Tarski Theorem. *Archive of Formal Proofs* (Sept. 2014). https://isa-afp.org/entries/Sturm_Tarski.html, Formal proof development.
[25] Wenda Li, Grant Olney Passmore, and Lawrence C. Paulson. 2019. Deciding Univariate Polynomial Problems Using Untrusted Certificates in Isabelle/HOL. *J. Autom. Reason.* 62, 1 (2019), 69–91. https://doi.org/10.1007/s10817-017-9424-6
[26] Wenda Li and Lawrence C. Paulson. 2016. A modular, efficient formalisation of real algebraic numbers. In *CPP*, Jeremy Avigad and Adam Chlipala (Eds.). ACM, 66–75. https://doi.org/10.1145/2854065.2854074
[27] Assia Mahboubi. 2007. Implementing the cylindrical algebraic decomposition within the Coq system. *Math. Struct. Comput. Sci.* 17, 1 (2007), 99–127. https://doi.org/10.1017/S096012950600586X
[28] Scott McCallum. 1985. An Improved Projection Operation for Cylindrical Algebraic Decomposition. In *EUROCAL (LNCS, Vol. 204)*, B. F. Caviness (Ed.). Springer, 277–278. https://doi.org/10.1007/3-540-15984-3_277
[29] Sean McLaughlin and John Harrison. 2005. A Proof-Producing Decision Procedure for Real Arithmetic. In *CADE (LNCS, Vol. 3632)*, Robert Nieuwenhuis (Ed.). Springer, 295–314. https://doi.org/10.1007/11532231_22
[30] César A. Muñoz, Anthony J. Narkawicz, and Aaron Dutle. 2018. A Decision Procedure for Univariate Polynomial Systems Based on Root Counting and Interval Subdivision. *J. Formaliz. Reason.* 11, 1 (2018), 19–41. https://doi.org/10.6092/issn.1972-5787/8212
[31] Anthony Narkawicz, César A. Muñoz, and Aaron Dutle. 2015. Formally-Verified Decision Procedures for Univariate Polynomial Computation Based on Sturm's and Tarski's Theorems. *J. Autom. Reason.* 54, 4 (2015), 285–326. https://doi.org/10.1007/s10817-015-9320-x
[32] Tobias Nipkow. 2010. Linear Quantifier Elimination. *J. Autom. Reason.* 45, 2 (2010), 189–212. https://doi.org/10.1007/s10817-010-9183-0
[33] Lawrence C. Paulson and Jasmin Christian Blanchette. 2010. Three years of experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers. In *IWIL (EPiC Series in Computing, Vol. 2)*, Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska (Eds.). EasyChair, 1–11.
[34] André Platzer. 2018. *Logical Foundations of Cyber-Physical Systems.* Springer, Cham. https://doi.org/10.1007/978-3-319-63588-0
[35] André Platzer, Jan-David Quesel, and Philipp Rümmer. 2009. Real World Verification. In *CADE (LNCS, Vol. 5663)*, Renate A. Schmidt (Ed.). Springer, 485–501. https://doi.org/10.1007/978-3-642-02959-2_35
[36] James Renegar. 1992. On the Computational Complexity and Geometry of the First-Order Theory of the Reals, Part III: Quantifier Elimination. *J. Symb. Comput.* 13, 3 (1992), 329–352. https://doi.org/10.1016/S0747-7171(10)80005-7
[37] Matias Scharager, Katherine Cordwell, Stefan Mitsch, and André Platzer. 2021. Verified Quadratic Virtual Substitution for Real Arithmetic. In *FM (LNCS, Vol. 13047)*, Marieke Huisman, Corina S. Pasareanu, and Naijun Zhan (Eds.). Springer, 200–217. https://doi.org/10.1007/978-3-030-90870-6_11
[38] Matias Scharager, Katherine Cordwell, Stefan Mitsch, and André Platzer. 2021. Verified Quadratic Virtual Substitution for Real Arithmetic. *Archive of Formal Proofs* (October 2021). https://isa-afp.org/entries/Virtual_Substitution.html, Formal proof development.
[39] Adam Strzeboński. 2000. Solving algebraic inequalities. *The Mathematica Journal* 7, 4 (2000), 525–541.
[40] Alfred Tarski. 1951. *A Decision Method for Elementary Algebra and Geometry.* RAND Corporation, Santa Monica, CA. https://www.rand.org/pubs/reports/R109.html
[41] Volker Weispfenning. 1988. The Complexity of Linear Problems in Fields. *J. Symb. Comput.* 5, 1-2 (1988), 3–27. https://doi.org/10.1016/S0747-7171(88)80003-8
[42] Makarius Wenzel. 2006. Structured Induction Proofs in Isabelle/Isar. In *MKM (LNCS, Vol. 4108)*, Jonathan M. Borwein and William M. Farmer (Eds.). Springer, 17–30. https://doi.org/10.1007/11812289_3

Received 2022-09-21; accepted 2022-11-21