CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Apollo

J Autom Reasoning
DOI 10.1007/s10817-017-9424-6

Deciding Univariate Polynomial Problems Using Untrusted Certificates in Isabelle/HOL

Wenda Li[1] · **Grant Olney Passmore**[2] · **Lawrence C. Paulson**[1]

Received: 25 October 2016 / Accepted: 29 July 2017
© The Author(s) 2017. This article is an open access publication

Abstract We present a proof procedure for univariate real polynomial problems in Isabelle/HOL. The core mathematics of our procedure is based on univariate cylindrical algebraic decomposition. We follow the approach of untrusted certificates, separating solving from verifying: efficient external tools perform expensive real algebraic computations, producing evidence that is formally checked within Isabelle's logic. This allows us to exploit highly-tuned computer algebra systems like Mathematica to guide our procedure without impacting the correctness of its results. We present experiments demonstrating the efficacy of this approach, in many cases yielding orders of magnitude improvements over previous methods.

Keywords Interactive theorem proving · Isabelle/HOL · Decision procedure · Cylindrical algebraic decomposition

1 Introduction

Nonlinear polynomial systems are ubiquitous in science and engineering. As real-world applications of formal verification continue to grow and diversify, there is an increasing

The first author was funded by the China Scholarship Council, via the CSC Cambridge Scholarship programme. The development of MetiTarski was supported by the Engineering and Physical Sciences Research Council [Grant Numbers EP/I011005/1, EP/I010335/1].

✉ Wenda Li
wl302@cam.ac.uk

Grant Olney Passmore
grant.passmore@cl.cam.ac.uk

Lawrence C. Paulson
lp15@cam.ac.uk

[1] Computer Laboratory, University of Cambridge, Cambridge, UK

[2] Aesthetic Integration, London and Clare Hall, University of Cambridge, Cambridge, UK

Published online: 08 August 2017

need for proof assistants (e.g., ACL2, Coq, Isabelle [27], HOL Light and PVS) to provide automation for reasoning about nonlinear systems over the reals [17,24,25].

Cylindrical algebraic decomposition (CAD) [8] is one of the most powerful known techniques for analysing non-linear polynomial systems. CAD-based methods have been implemented in various systems such as Z3 [9], QEPCAD [3], Mathematica and Maple. However, implementing CAD-based decision procedures within proof assistants has been hindered by the difficulty in formalising the mathematics justifying CAD computations.

In this paper, we present a formally verified procedure[1] based on CAD for univariate polynomial problems with rational coefficients. Goals such as

$$\forall x.\ \left(x^2 > 2 \wedge x^{10} - 2x^5 + 1 \geq 0\right) \vee x < 2$$
$$\exists x.\ \left(x^2 = 2 \wedge (x > 1 \vee x < 0)\right)$$

can be discharged by our tactic automatically. It should be noted that certifying a general *multivariate* CAD procedure is much harder, and the univariate version we describe in the paper is only a first step in that direction.

A key feature of our procedure is its certificate-based design in which an external untrusted (but ideally highly efficient) program is used to find certificates, and those certificates are then checked by verified internal procedures. Overall, the soundness of our procedure depends solely on the soundness of Isabelle's logic (and code generation[2]) rather than trusted external oracles. This is much like Isabelle's sledgehammer tactic, which sceptically incorporates various external tools.

Our main contributions are:

- An efficient formalised theory of Tarski queries,
- An efficient approach to univariate sign determination at real algebraic points,
- A practical formally verified procedure for real algebraic problems based on univariate CAD.

The paper continues at follows: A motivating example (Sect. 2) and a description of the overall design (Sect. 3) sketch the general idea of our procedure. The construction and manipulation of real algebraic numbers is developed in (Sect. 4), including a sign determination procedure for evaluating polynomials at real algebraic points (Sect. 5). The main proof is described in (Sect. 6), which is followed by a discussion of interaction with external solvers (Sect. 7). Next, experiments and related work (Sect. 8) are described along with further discussion of our tactic (Sect. 9). We then conclude with a look towards the future (Sect. 10).

2 A Motivating Example

Unlike the general case of $\mathbb{R}^n$, the restriction of CAD to univariate problems (i.e., to $\mathbb{R}^1$) is relatively straight-forward. Suppose we wish to prove

$$\forall x.\, P(x) > 0 \vee Q(x) \geq 0$$

where

$$P(x) = \frac{1}{2}x^2 - 1$$

[1] Code is available from https://bitbucket.org/liwenda1990/src_jar_2017.

[2] As our tactic is computationally intense, our procedure makes use of the proof by reflection technique [16].

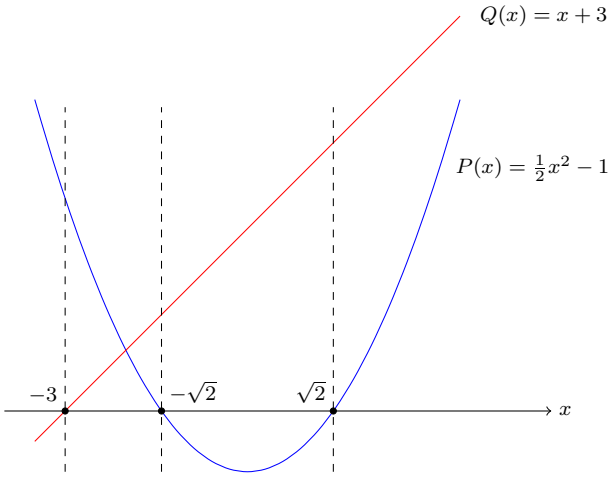

Fig. 1 The plot of $P(x) = \frac{1}{2}x^2 - 1$ and $Q(x) = x + 3$

$$Q(x) = x + 3.$$

To do so, we can *decompose* $\mathbb{R}$ into disjoint connected components induced by the roots of P and Q. This is illustrated in Fig. 1:

$$\mathfrak{D} = \{(-\infty, -3), \{\ -3\ \}, (-3, -\sqrt{2}), \{\ -\sqrt{2}\ \}, (-\sqrt{2}, \sqrt{2}), \{\ \sqrt{2}\ \}, (\sqrt{2}, \infty)\}$$

root of Q roots of P

and it can be observed that both P and Q have invariant signs over each of these components. For example, as can be seen from Fig. 1, $P(x) < 0$ and $Q(x) > 0$ hold for all $x \in (-\sqrt{2}, \sqrt{2})$. To decide the conjecture, we can pick sample points from each of these components and evaluate $\lambda x.\, P(x) > 0 \vee Q(x) \geq 0$ at these points. That is,

$$\begin{aligned}
&\forall x.\, P(x) > 0 \vee Q(x) \geq 0 \\
&= \forall D \in \mathfrak{D}.\, \forall x \in D.\, P(x) > 0 \vee Q(x) \geq 0 \\
&= \forall x \in \{-4, -3, -2, -\sqrt{2}, 0, \sqrt{2}, 2\}.\, P(x) > 0 \vee Q(x) \geq 0 \\
&= (P(-4) > 0 \vee Q(-4) \geq 0) \wedge (P(-3) > 0 \vee Q(-3) \geq 0) \wedge \ldots \\
&\quad \wedge (P(2) > 0 \vee Q(2) \geq 0) \\
&= \text{True}
\end{aligned} \tag{1}$$

since

$$\begin{aligned}
-4 &\in (-\infty, -3) \\
-3 &\in \{-3\} \\
-2 &\in (-3, -\sqrt{2}) \\
-\sqrt{2} &\in \{-\sqrt{2}\} \\
0 &\in (-\sqrt{2}, \sqrt{2}) \\
\sqrt{2} &\in \{\sqrt{2}\}
\end{aligned}$$

$$2 \in (\sqrt{2}, \infty).$$

Analogously, to decide an existential formula

$$\exists x.\, P(x) = 0 \wedge Q(x) > 0,$$

we have

$$\begin{aligned}
&\exists x.\, P(x) = 0 \wedge Q(x) > 0 \\
&= \exists D \in \mathfrak{D}.\, \exists x \in D.\, P(x) = 0 \wedge Q(x) > 0 \\
&= \exists x \in \{-4, -3, -2, -\sqrt{2}, 0, \sqrt{2}, 2\}.\, P(x) = 0 \wedge Q(x) > 0 \\
&= (P(-4) = 0 \wedge Q(-4) > 0) \vee (P(-3) = 0 \wedge Q(-3) > 0) \vee \ldots \\
&\quad \vee (P(2) = 0 \wedge Q(2) > 0) \\
&= \text{True}.
\end{aligned} \tag{2}$$

In performing these arguments, there were a few "obvious" subtleties:

- The decomposition of $\mathbb{R}$ into the seven regions given *covered* the entire real line. That is,

$$(-\infty, -3) \cup \{-3\} \cup (-3, -\sqrt{2}) \cup \{-\sqrt{2}\} \cup (-\sqrt{2}, \sqrt{2}) \cup \{\sqrt{2}\} \cup (\sqrt{2}, \infty) = \mathbb{R}.$$

- The "sign-invariance" of P and Q over each region was exploited to allow only a single sample point to be selected from each region. This property holds as by the Intermediate Value Theorem, P and Q can only change sign by passing through a root.
- The signs of univariate polynomials were evaluated at irrational real algebraic points like $\sqrt{2}$ to determine the truth values of atomic formulas.

In creating our automatic proof procedure, all of this routine reasoning must, of course, be formalised. Moreover, the isolation of polynomial roots (and thus sign-invariant regions) and the sign determination for polynomials at real algebraic points are computationally expensive operations. Computer algebra systems like Mathematica have decades of tuning in their implementations of these core algebraic algorithms. To have a practical proof procedure, we wish to take advantage of these highly tuned external tools as much as possible. Let us next describe how this can be done.

3 A Sketch of Our Certificate-Based Design

There is a rich history of certificate-based, sceptical integrations between proof assistants and external solvers. Examples include John Harrison's sums-of-squares method [17] and the Sledgehammer [31] command in Isabelle.

Certificate-based approaches are motivated by many observations, including:

- External solvers are often highly tuned and run much faster than verified ones.
- Verification of certificates from external solvers is usually much easier than finding them. Such verification ensures the soundness of the overall tactic.
- Switching between different external solvers does not require changes in formal proofs.

Algorithm 1 sketches our idea for univariate universal formulas. In particular, in line 3, we use external programs to return real roots of polynomials (i.e., $\mathfrak{P}$) from the quantifier-free part of the formula (i.e., $F(x)$). Those roots (i.e., *roots*) correspond to a decomposition such that

Algorithm 1 Prove univariate universal formulas over reals

Require: $F(x)$ is a quantifier-free formula over reals
Ensure: Return true if $\forall x.\, F(x)$ holds
1: **procedure** UNIVERSAL($\forall x.\, F(x)$)
2: $\mathfrak{P} \leftarrow$ extract polynomials from $F(x)$ ▷ $\mathfrak{P} \subseteq \mathbb{Z}[X]$
3: *roots* $\leftarrow$ real roots of $\mathfrak{P}$ ▷ Roots returned by external programs
4: *samples* $\leftarrow$ construct sample points from *roots*
5: **if** $(\forall x \in \mathit{samples}.\, F(x)) \wedge$ (*roots* are indeed all real roots of $\mathfrak{P}$) **then**
6: **return** true
7: **end if**
8: **end procedure**

each polynomial from $\mathfrak{P}$ has a constant sign over each component of this decomposition. Since the roots are returned by untrusted programs, in line 5, we not only check $\forall x \in \mathit{samples}.\, F(x)$ as in Eq. (1) but also certify that these roots are indeed all real roots of $\mathfrak{P}$.

The step in line 3 in Algorithm 1 is more commonly referred as *(real) root isolation*, which is a classic and well-studied topic in symbolic computing. Although we can in principle formalise our own root isolation procedure (e.g., using the Sturm–Tarski theorem), it is utterly unlikely that our implementation will be competitive with state-of-the-art ones, especially for polynomials of high degree, large bit-width, or whose roots are very close together. Therefore, we delegate this computationally expensive step to external tools.

Algorithm 2 Prove univariate existential formulas over reals

Require: $F(x)$ is a quantifier-free formula over reals
Ensure: Return true if $\exists x.\, F(x)$ holds
1: **procedure** EXISTENTIAL($\exists x.\, F(x)$)
2: $r \leftarrow$ solution to $F(x)$ ▷ Solution returned by external programs
3: **if** $F(r)$ **then**
4: **return** true
5: **end if**
6: **end procedure**

With existential formulas, the situation is even simpler as illustrated in Algorithm 2, since we do not need to deal with the decomposition internally. Rather, all we need is a real algebraic witness that satisfies $\lambda x.\, F(x)$ to certify $\exists x.\, F(x)$. What is more interesting is that the satisfaction problem for $\lambda x.\, F(x)$ can be not only solved by a CAD procedure, which is complete but not very fast due to its symbolic nature, but also be complemented by highly efficient incomplete numerical methods. Thus it is natural to externalize the step in line 2 in Algorithm 2.

4 Encoding Real Algebraic Numbers

External programs in either Algorithms 1 and 2 can return real algebraic numbers (e.g. $\sqrt{2}$). In this section, we see how to formalise such numbers in Isabelle/HOL.

The real algebraic numbers ($\mathbb{R}_{\text{alg}}$) are real roots of non-zero polynomials with integer (equivalently, rational) coefficients. They form a countable, computable subfield of the real numbers. To encode them, we use a polynomial with integer coefficients and a root selection method to "pin down" the root in question. Common root selection methods include isolating

intervals, root indices or Thom encodings. We use the root interval approach, that is, a real algebraic number $r \in \mathbb{R}_{\text{alg}}$ will be given by

- A polynomial $p \in \mathbb{Z}[x]$ s.t. $p(r) = 0$, and
- Two rationals $a, b \in \mathbb{Q}$ s.t. r is the only root of p contained in $[a, b]$.

To reason over the reals, we define a function `Alg` to embed those real algebraic numbers into the reals:

```
Alg:: "int poly ⇒ float ⇒ float ⇒ real"
```

where `int poly` is a polynomial with integer coefficients and the two `float` arguments represent an interval. Note, a `float` in Isabelle/HOL is a dyadic rational number of the form

$$a2^b \quad \text{where} \quad a, b \in \mathbb{Z}.$$

Compared to our previous work [21], where a pair of rational numbers is used to represent an interval, the dyadic rational approach is more efficient due to the elimination of ubiquitous greatest common divisor (gcd) operations within rational arithmetic.

In Isabelle/HOL, a real number is represented as a Cauchy sequence of type `nat ⇒ rat`, where a Cauchy sequence is defined as

```
definition
  cauchy :: "(nat ⇒ rat) ⇒ bool"
where
  "cauchy X ⟷ (∀r>0. ∃k. ∀m≥k. ∀n≥k. |X m - X n| < r)"
```

We then convert an encoding of a real algebraic number into a sequence of type `nat ⇒ rat`. The idea is to bisect the isolating interval through each recursive call, and proceed with the half where the sign of the polynomial changes at its end points:

```
fun to_cauchy:: "rat poly × rat × rat ⇒ nat ⇒ rat" where
  "to_cauchy (_, lb, ub) 0 = (lb+ub)/2"|
  "to_cauchy (p, lb, ub) (Suc n) = (
    let c=(lb+ub)/2
    in if poly p lb * poly p c ≤ 0
       then to_cauchy (p, lb, c) n
       else to_cauchy (p, c, ub) n)"
```

where `poly p x` evaluates the polynomial `p` at the point `x`. Note, `rat poly × rat × rat` encodes a real algebraic number here (rather than `int poly × float × float`), as we can embed `int` and `float` into `rat`.

It can be then shown that the sequence constructed by `to_cauchy (p, lb, ub)` is indeed a Cauchy sequence and the real number represented by this sequence resides within the interval $[lb, ub]$, provided $lb < ub$:

```
lemma to_cauchy_cauchy:
  fixes p::"rat poly" and lb ub ::rat
  assumes "lb<ub"
  defines "X≡to_cauchy (p,lb,ub)"
  shows "cauchy X"

lemma to_cauchy_bound:
  fixes p::"rat poly" and lb ub ::rat
  defines "X≡to_cauchy (p,lb,ub)"
  assumes "lb<ub"
  shows "lb ≤ Real X" "Real X ≤ ub"
```

Note, the function `Real` of type `(nat ⇒ rat) ⇒ real` constructs a real number from its underlying representation (i.e. a Cauchy sequence).

Finally, we can finish the definition of `Alg`:

```
definition valid_alg::"int poly ⇒ float ⇒ float ⇒ bool" where
  "valid_alg p lb ub = (lb < ub ∧ poly p lb * poly p ub < 0
    ∧ card ({x::real. poly p x = 0 ∧ lb < x ∧ x < ub}) = 1)"

definition Alg:: "int poly ⇒ float ⇒ float ⇒ real" where
  "Alg p lb ub = (if valid_alg p lb ub
       then Real (to_cauchy (p, lb, ub))
       else undefined)"
```

where `valid_alg p lb ub` ensures

- `lb < ub`,
- The polynomial `p` is of different signs (and non-zero) at `lb` and `ub`,
- The polynomial `p` has exactly one real root within the interval `(lb,ub)`.

With the help of `Alg`, we can now encode the real algebraic number $\sqrt{2}$ as

```
Alg [:-2,0,1:] 1 2
```

where `[:-2,0,1:]` corresponds to the polynomial $-2x^0 + 0x^1 + 1x^2 = x^2 - 2$, and 1 and 2 are the lower bound and upper bound respectively, such that $\sqrt{2}$ is the only root of $x^2 - 2$ within the interval $(1, 2)$.

Furthermore, we can formally derive that `Alg p lb ub` is indeed a root of `p` within the interval `(lb,ub)`:

```
lemma alg_bound_and_root:
  fixes p::"int poly" and lb ub::float
  assumes "valid_alg p lb ub"
  shows "lb < Alg p lb ub" and "Alg p lb ub < ub"
    and "poly (of_int_poly p) (Alg p lb ub) = 0"
```

where `of_int_poly p` embeds the integer polynomial `p` into a real one.

5 Deciding the Sign of a Univariate Polynomial at Real Algebraic Points

In the previous section, we described how to encode a real algebraic number as an integer polynomial and two dyadic rational numbers. Now, suppose we have

$$\sqrt{2} = (x^2 - 2, 1, 2)$$

where $(x^2 - 2, 1, 2)$ is abbreviated from `Alg [:-2,0,1:] 1 2` for the sake of readability. How can we computationally prove that

$$P(\sqrt{2}) = 0 \quad \text{where} \quad P(x) = \frac{1}{2}x^2 - 1 \ ?$$

Considering that $\mathbb{R}_{\text{alg}}$ is a computable subfield of $\mathbb{R}$ and has decidable arithmetic and comparison operations, it is natural to evaluate such formulas through algebraic arithmetic:

$$P(\sqrt{2}) = \frac{1}{2} \times_{\text{alg}} (x^2 - 2, 1, 2) \times_{\text{alg}} (x^2 - 2, 1, 2) -_{\text{alg}} 1$$

$$
\begin{aligned}
&= \frac{1}{2} \times_{\text{alg}} (x - 2, 1, 3) -_{\text{alg}} 1 \\
&= \left(x - 1, \frac{1}{2}, \frac{3}{2}\right) -_{\text{alg}} 1 \\
&= 0,
\end{aligned}
$$

where $\times_{\text{alg}}$ and $-_{\text{alg}}$ are exact algebraic arithmetic operations that usually involve calculation of bivariate resultants. Although such computations are currently possible in Isabelle/HOL [21,36], they are far from efficient.

In this section, we describe a verified procedure to decide the sign of univariate polynomials with rational coefficients at real algebraic points which uses *only* rational (or dyadic rational) arithmetic rather than costly algebraic arithmetic.

5.1 The Sturm–Tarski Theorem

We abbreviate $\mathbb{R} \cup \{-\infty, \infty\}$ as $\overline{\mathbb{R}}$, the extended real numbers.

Definition 1 (*Tarski Query*) The Tarski query $\text{TaQ}(Q, P, a, b)$ is

$$\text{TaQ}(Q, P, a, b) = \sum_{x \in (a,b), P(x)=0} \text{sgn}(Q(x))$$

where $a, b \in \overline{\mathbb{R}}$, $P, Q \in \mathbb{R}[X]$, $P \neq 0$ and $\text{sgn} : \mathbb{R} \to \{-1, 0, 1\}$ is the sign function.

The Sturm–Tarski theorem [23, Chapter 8] (or Tarski's theorem [2, Chapter 2]) is essentially an effective way to compute Tarski queries through some remainder sequences:

Theorem 1 (Sturm–Tarski) *The Sturm–Tarski theorem states*

$$\text{TaQ}(Q, P, a, b) = \text{Var}(\text{SRemS}(P, P'Q); a, b)$$

where $P \neq 0$, $P, Q \in \mathbb{R}[X]$, P' *is the first derivative of* P, $a, b \in \overline{\mathbb{R}}$, $a < b$ *and are not roots of* P, $\text{SRemS}(P, P'Q)$ *is the signed remainder sequence of* P *and* $P'Q$, *and*

$$
\begin{aligned}
&\text{Var}([p_0, p_1, \ldots, p_n]; a, b) \\
&\quad = \text{Var}([p_0(a), p_1(a), \ldots, p_n(a)]) - \text{Var}([p_0(b), p_1(b), \ldots, p_n(b)])
\end{aligned}
$$

is the difference in the number of sign variations (after removing zeroes) in the polynomial sequence $[p_0, p_1, \ldots, p_n]$ *evaluated at* a *and* b.

Note that the more famous Sturm's theorem, which counts the number of distinct real roots (of a univariate polynomial) within an interval, is a special case of the Sturm–Tarski theorem when $Q = 1$.

5.2 A Formal Proof of the Sturm–Tarski Theorem

Our proof of the Sturm–Tarski theorem in Isabelle is based on Basu et al. [2, Chapter 2] and Cohen's formalisation in Coq [6].

The core idea of our formal proof is built around the *Cauchy index*. First defined by Cauchy in 1837, the Cauchy index of a real rational function encodes deep properties of its roots and poles, and can be used as the basis of an algebraic method for computing Tarski queries.[3]

[3] Besides the application described in this section, the Cauchy index also plays a critical role in the Routh–Hurwitz theorem. Interested readers may consult [32, Chapter 10, 11] for historical notes.

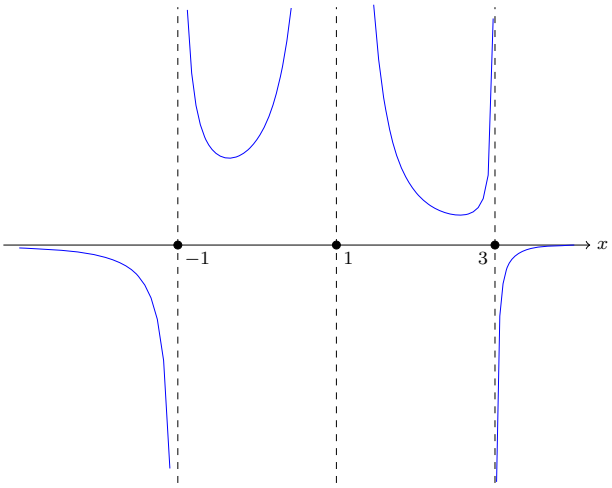

Fig. 2 Graph of the rational function $(x-4)/((x-3)(x-1)^2(x+1))$

Definition 2 Given $P, Q \in \mathbb{R}[x]$ and $x \in \mathbb{R}$, $\text{jump}(P, Q, x)$ is defined as

$$\text{jump}(P, Q, x) = \begin{cases} -1 & \text{if } \lim_{u \to x^-} \frac{Q(u)}{P(u)} = \infty \text{ and } \lim_{u \to x^+} \frac{Q(u)}{P(u)} = -\infty \\ 1 & \text{if } \lim_{u \to x^-} \frac{Q(u)}{P(u)} = -\infty \text{ and } \lim_{u \to x^+} \frac{Q(u)}{P(u)} = \infty \\ 0 & \text{otherwise.} \end{cases}$$

For example, let $Q = x - 4$ and $P = (x-3)(x-1)^2(x+1)$. The graph of Q/P is shown in Fig. 2. We have

$$\text{jump}(P, Q, x) = \begin{cases} 1 & \text{when } x = -1 \\ -1 & \text{when } x = 3 \\ 0 & \text{otherwise.} \end{cases}$$

The Cauchy index `cindex_poly a b q p` is the sum of the jumps of q/p over the interval (a, b):

```
definition cindex_poly:: "real ⇒ real ⇒ real poly ⇒ real poly ⇒ int"
    where
  "cindex_poly a b q p≡ (∑x∈{x. poly p x=0 ∧ a < x ∧ x < b}.
jump_poly q p x)"
```

By case analysis, we can prove a connection between the Tarski query and the Cauchy index:

```
lemma cindex_poly_taq:
  fixes p q::"real poly" and a b::real
  shows "taq {x. poly p x = 0 ∧ a < x ∧ x < b} q
          =cindex_poly a b (pderiv p * q) p"
```

where `taq` is a formal definition of the Tarski query

```
definition taq :: "'a::linordered_idom set ⇒ 'a poly ⇒ int" where
  "taq s q = (∑x∈s. sign (poly q x))"
```

and `pderiv p` is the first derivative of `p`.

Moreover, the Cauchy index can be related to Euclidean division (`mod`) on polynomials by a recurrence:

```
cindex_poly_rec:
  fixes p q::"real poly" and a b::real
  assumes "a < b" and "poly (p * q) a ≠0"
      and "poly (p * q) b ≠0"
  shows "cindex_poly a b q p = cross (p * q) a b
          + cindex_poly a b (- (p mod q)) q"
```

where

$$\text{cross } p\ a\ b = \begin{cases} 0 & \text{if } p(a)p(b) \geq 0 \\ 1 & \text{if } p(a)p(b) < 0 \text{ and } p(a) < p(b) \\ -1 & \text{if } p(a)p(b) < 0 \text{ and } p(a) \geq p(b). \end{cases}$$

A similar recurrence relation holds for the number of sign variations of the signed remainder sequences (`changes_itv_smods`):

```
lemma changes_itv_smods_rec:
  fixes p q::"real poly" and a b::real
  assumes "a < b" and "poly (p * q) a ≠ 0"
        and "poly (p * q) b ≠ 0"
  shows "changes_itv_smods a b p q = cross (p * q) a b
          + changes_itv_smods a b q (- (p mod q))"
```

where `changes_itv_smods` is defined as

```
definition changes_itv_smods::
    "real ⇒ real ⇒ real poly ⇒ real poly ⇒ int" where
  "changes_itv_smods a b p q = (
      let
        ps = smods p q
      in
        changes_poly_at ps a - changes_poly_at ps b)"
```

and the signed remainder sequence (`smods`) is defined as

```
function smods:: "real poly ⇒ real poly ⇒ (real poly) list" where
  "smods p q= (if p = 0 then
                 []
               else
                 p # (smods q (-(p mod q))))"
```

and `changes_poly_at ps a` returns the number of sign changes when evaluating a list of polynomials (`ps`) at `a`.

Finally, by combining `cindex_poly_taq`, `cindex_poly_rec` and `changes_itv_smods_rec`, we derive the Sturm–Tarski theorem:

```
theorem sturm_tarski_interval:
  fixes p q::"real poly" and a b::real
  assumes "a < b" and "poly p a ≠ 0" and "poly p b ≠ 0"
  shows "taq {x. poly p x = 0 ∧ a < x ∧ x < b} q
      = changes_itv_smods a b p (pderiv p * q)"
```

Note, this is just the bounded case of the Sturm–Tarski theorem. Proofs for the unbounded and half-bounded cases are similar.

5.3 Sign Determination Through the Sturm–Tarski Theorem

Given a polynomial q with rational coefficients and our encoding of a real algebraic number α

$$\alpha = (p, lb, ub)$$

where p is an integer polynomial, and lb and ub are dyadic rationals, we can effectively decide the sign of $q(\alpha)$ using the Sturm–Tarski theorem, provided `valid_alg p lb ub` holds. The rationale behind is that `valid_alg p lb ub` ensures α is the only root of p within the interval (lb, ub), hence

$$\begin{aligned}\operatorname{sgn}(q(\alpha)) &= \sum_{x \in (lb,ub),\, p(x)=0} \operatorname{sgn}(q(x)) \\ &= \operatorname{TaQ}(q, p, lb, ub) \\ &= \operatorname{Var}(\operatorname{SRemS}(p, p'q); lb, ub).\end{aligned}$$

Importantly, it can be observed that evaluating $\operatorname{Var}(\operatorname{SRemS}(p, p'q); lb, ub)$ requires only rational arithmetic rather than costly algebraic arithmetic.

To be even more efficient, we refine the procedure further to make use of dyadic rational arithmetic. The main advantage of dyadic rational arithmetic over rational arithmetic are reduced normalization steps and possible bit-level operations. For example, consider two rational numbers $\frac{a_1}{b_1}$ and $\frac{a_1}{b_2}$ where $a_1, b_1, a_2, b_2 \in \mathbb{Z}$, their sum is

$$\begin{gathered}\frac{a_1}{b_1} + \frac{a_2}{b_2} = \frac{a_1 b_2 + a_2 b_1}{b_1 b_2} = \frac{(a_1 b_2 + a_2 b_1)/c}{(b_1 b_2)/c} \\ \text{where } c = \gcd(a_1 b_2 + a_2 b_1, b_1 b_2).\end{gathered}$$

To counter the growth in the size of representations, we usually need to normalize the result by factoring out the gcd. Such gcd operations can be the source of major computational expense. Thankfully, they are unnecessary in the context of dyadic rationals. The sum of two dyadic rationals (a_1, e_1) and (a_2, e_2) where $a_1, e_1, a_2, b_2 \in \mathbb{Z}$ is

$$a_1 2^{e_1} + a_2 2^{e_2} = \begin{cases} (a_1 2^{e_1 - e_2} + a_2) 2^{e_2} & \text{if } e_1 > e_2 \\ (a_1 + a_2 2^{e_2 - e_1}) 2^{e_1} & \text{otherwise.} \end{cases}$$

Moreover, multiplications by powers of two, such as $a_1 2^{e_1 - e_2}$, can be optimised by shift operations.

However, the problem with dyadic rational numbers is that they do not have the division operation (e.g. 1×2^0 divided by 3×2^0 is no longer a dyadic rational), hence they do not form a field, while Euclidean division only works for polynomials over a field. This problem can be solved if we switch from Euclidean division (`mod` and `div`):

$$P = (P \operatorname{div} Q)\, Q + (P \bmod Q) \;\text{ and }\; (Q = 0 \vee \deg(P \bmod Q) < \deg(Q))$$

to pseudo-division (`pmod` and `pdiv`) [10]:

$$\begin{gathered}\operatorname{lc}(Q)^{1+\deg(P)-\deg(Q)} P = (P \operatorname{pdiv} Q)\, Q + (P \operatorname{pmod} Q) \\ \text{and } (Q = 0 \vee \deg(P \bmod Q) < \deg(Q)) \\ \text{where } \operatorname{lc}(Q) \text{ is the leading coefficient of } Q,\end{gathered}$$

since pseudo-division can be carried out by polynomials over an integral domain (rather than a field).

Based on pseudo-division, the signed pseudo-remainder sequence (SPRemS) can be defined:

```
function spmods :: "'a::idom poly ⇒ 'a poly ⇒ ('a poly) list" where
  "spmods p q= (if p=0 then [] else
      let
        m=(if even(degree p+1-degree q) then -1 else -lead_coeff q)
      in
        Cons p (spmods q (smult m (p pmod q))))"
```

where `smult` is the scalar product on polynomials and `lead_coeff q` is the leading coefficient of `q`. Accordingly, the function to count the difference in sign variations can be refined:

```
definition changes_itv_spmods::
  "'a ::linordered_idom ⇒ 'a ⇒ 'a poly ⇒ 'a poly ⇒ int" where
  "changes_itv_spmods a b p q= (let ps = spmods p q in
        changes_poly_at ps a - changes_poly_at ps b)"
```

and linked to the previous one based on signed remainder sequences (SRemS):

```
lemma changes_spmods_smods:
  fixes p q::"float poly" and a b::"float"
  shows "changes_itv_spmods a b p q
          = changes_itv_smods (real_of_float a) (real_of_float b)
                              (of_float_poly p) (of_float_poly q)"
```

where `real_of_float` embeds a `float` into `real` and `of_float_poly` coverts a `float poly` (i.e. polynomial with dyadic rational coefficients) to a `real poly` by embedding each of the coefficients into `real`.

Finally, we define a function `sgn_at` that returns the sign of a univariate polynomial at some point:

```
definition "(sgn_at::real poly⇒real⇒real) = (λq x. sgn (poly q x))"
```

Note, for now, if either `x` or any coefficient of `q` is an irrational real number (e.g. an irrational real algebraic number), evaluating `sgn_at q x` will raise an exception, as Isabelle/HOL, by default, only supports rational arithmetic. Although we can eliminate some such exceptions by loading any of the recent algebraic arithmetic libraries [21,36], we consider exact algebraic arithmetic too slow for our purpose as stated at the beginning of Sect. 5. Alternatively, by proving some code equations, we can restore the executability of `sgn_at q x` when `x` is constructed by `Alg p lb ub` and coefficients of `q` are rational reals:

```
lemma sgn_at_code_alg[code]:
  fixes q::"real poly" and p::"int poly" and lb ub::float
  shows "sgn_at q (Alg p lb ub) = (
    if valid_alg p lb ub ∧ (∀x∈set (coeffs q). is_rat x) then
      (let
          p'::float poly=of_int_poly p;
          q'::float poly=of_int_poly (int_poly q)
       in
      of_int (changes_itv_spmods lb ub p' (pderiv p' * q')))
    else Code.abort (STR ''Invalid sgn_at'')
        (λ_. sgn_at q (Alg p lb ub)))"
```

where

- `∀x∈set (coeffs q). is_rat x` checks if each coefficient of `q` is rational,
- `of_int_poly` converts an integer polynomial into a dyadic rational one,
- `int_poly` clears denominators in the coefficients by multiplying each coefficient by the least common multiple (of the denominators),
- `Code.abort` throws an exception, if either (p, lb, ub) is an invalid representation of a real algebraic number or the polynomial `q` has any non-rational coefficient.

And note that evaluating `changes_itv_spmods lb ub p' (pderiv p' * q')` requires only dyadic arithmetic, which is much more efficient than exact algebraic arithmetic.

Moreover, the executability of `valid_alg` is restored similarly as well:

```
lemma [code]:
  fixes p::"int poly" and lb ub::float
  shows "valid_alg p lb ub = (lb < ub
    ∧ (sgn (poly (of_int_poly p) lb) * sgn (poly (of_int_poly p)
ub) < 0)
    ∧ changes_itv_spmods lb ub (of_int_poly p) (pderiv (of_int_poly p))
= 1)"
```

where

```
changes_itv_spmods lb ub (of_int_poly p) (pderiv (of_int_poly p)) = 1
```

checks if the polynomial p has exactly one real root within the interval (lb, ub) by exploiting Sturm's theorem (a special case of our formalised Sturm–Tarski theorem).

After restoring executability of `sgn_at` on real algebraic numbers, we can now check the sign of $P(x) = \frac{1}{2}x^2 - 1$ at $\sqrt{2}$ by typing the following command:

```
value "sgn_at [:-1,0,1/2:] (Alg [:-2,0,1:] 1 2)"
```

which returns 0 (i.e. $P(\sqrt{2}) = 0$).

5.4 Remark

A formal proof of the Sturm–Tarski theorem is not new among proof assistants: it has been formalised in PVS [25] and Coq [6]. However, as far as we know, we are the first to exploit this theorem to build a verified sign determination procedure of real algebraic numbers, which uses only rational or dyadic rational arithmetic.

Real algebraic numbers are essential in symbolic computing, and well studied. In general, exact real algebrac arithmetic is rarely used in modern computer algebra systems due to its extreme inefficiency. For example, consider the problem of isolating the real roots of a polynomial with real algebraic coefficients. Modern approaches usually use sophisticated techniques to soundly approximate those coefficients to a certain precision rather than carrying out exact algebraic arithmetic [5,33,35], relying on exact symbolic procedures as a fall-back in degenerate cases.

Following these efficient modern approaches, our sign determination procedure can be improved in at least the following ways:

- Sophisticated interval arithmetic can be used to decide the sign before resorting to a remainder sequence, as has been done in Z3 [10]. This approach should help when the sign is non-zero.
- Pseudo-division, which we are currently using for building remainder sequences, is not good for controlling coefficients growth. More sophisticated approaches, such as subresultant sequences and modular methods, can be used to optimise the calculation of remainder sequences.

6 The Formal Development of the Decision Procedure

In this section, we describe the main proof underlying our tactic.

6.1 Parsing Formulas

The first step of our tactic is to parse the target formula into a structured form. This process is usually referred as reification [4] in Isabelle/HOL. More specifically, given an Isabelle/HOL term e of type τ, we define a (more structured) datatype δ and an interpretation function *interp* of type $\delta \Rightarrow \tau$ *list* $\Rightarrow \tau$, such that for some e‘ of type δ

$$e = \mathit{interp}\ e\text{‘}\ \mathit{xs}$$

where *xs* is a list of free variables in e. Subsequently, instead of directly dealing with e, we now convert it into a more pleasant form *interp* e‘ *xs* where e‘ is in fact a formal language that captures the structure of e.

The datatypes we defined to capture the structure of target univariate formulas are as follows:

```
datatype num = C real — Constant
    | Var nat — Variable index
    | Add num num | Minus num | Mul num num | Power num nat

datatype norm_num2 =
    Pol "int poly" nat — an integer polynomial and its variable index
    | Const real — constant
    | Abnorm num — in case of anomalies (e.g., bivariate)

datatype qf_form2 =
    Pos norm_num2 — is positive  | Zero norm_num2 — is zero
    | Neg qf_form2 — negation
    | Conj qf_form2 qf_form2 — conjunction
    | Disj qf_form2 qf_form2 — disjunction
    | T — true  | F — false

datatype norm_form2 =
    QF qf_form2 — quantifier free
    | ExQ norm_form2 — existential
    | AllQ norm_form2 — universal
```

and the interpretation functions:

```
fun num_interp:: "num ⇒ real list ⇒ real" where
  "num_interp (C i) vs = i"|
  "num_interp (Var v) vs = vs!v"|
  "num_interp (Add num1 num2) vs = num_interp num1 vs + num_interp num2
vs "|
  "num_interp (Minus num) vs = - num_interp num vs "|
  "num_interp (Mul num1 num2) vs = num_interp num1 vs * num_interp num2
vs "|
  "num_interp (Power num n) vs = (num_interp num vs)^n"

fun norm_num2_interp :: "norm_num2 ⇒ real list ⇒ real" where
  "norm_num2_interp (Pol p v) vs = poly (of_int_poly p) (vs!v)"|
  "norm_num2_interp (Const c) vs = c"|
  "norm_num2_interp (Abnorm num) vs = num_interp num vs" — anomaly

fun qf_form2_interp:: "qf_form2 ⇒ real list ⇒ bool" where
  "qf_form2_interp (Pos norm_num) vs = (norm_num2_interp norm_num
vs > 0)"|
  "qf_form2_interp (Zero norm_num) vs = (norm_num2_interp norm_num
vs = 0)"|
  "qf_form2_interp (Neg qf_form) vs = (¬ qf_form2_interp qf_form vs)" |
```

```
  "qf_form2_interp (Conj qf_form1 norm_form2) vs
    = (qf_form2_interp qf_form1 vs ∧ qf_form2_interp norm_form2 vs)"|
  "qf_form2_interp (Disj qf_form1 qf_form2) vs
    = (qf_form2_interp qf_form1 vs ∨ qf_form2_interp qf_form2 vs)"|
  "qf_form2_interp T vs = True"|
  "qf_form2_interp F vs = False"

fun norm_form2_interp:: "norm_form2 ⇒real list ⇒ bool" where
  "norm_form2_interp (QF qf) vs = qf_form2_interp qf vs"|
  "norm_form2_interp (ExQ norm_form) vs
      = (∃x. norm_form2_interp norm_form (x#vs))"|
  "norm_form2_interp (AllQ norm_form) vs
      = (∀x. norm_form2_interp norm_form (x#vs))"
```

Given the definition of a (structured) datatype *norm_form2* and the corresponding interpretation function *norm_form2_interp*, target formulas can now be parsed. For example, we can convert a univariate formula

```
"∀x::real. x > 1/2 ∨ x < 1"
```

into an equivalent form

```
"norm_form2_interp
   (AllQ (QF (Disj (Pos (Pol [:- 1, 2:] 0))
                   (Pos (Pol [:1, - 1:] 0))
             )))
   []"
```

In particular, note

```
qf_form2_interp (Pos (Pol [:- 1, 2:] 0)) [x]
= (poly [:- 1, 2:] x > 0)
= (x > 1/2)
```

in which inequalities have been parsed into a polynomial sign determination problem.

On the contrary, a bivariate non-closed formula such as

```
"∃x::real. x + y >0"
```

will be converted into

```
"norm_form2_interp
  (ExQ (QF (Pos
        (Abnorm
    (Add (Add (Add (C 0) (Mul (Var 0) (Add (C 1) (Mul (Var 0) (C 0))))))
         (Add (C 0) (Mul (Var 1) (Add (C 1) (Mul (Var 1) (C 0))))))
     (C 0))))))
     [y]"
```

where the *Abnorm* constructor indicates that such formula is not supported by our current tactic.

6.2 Existential Case

To discharge a univariate existential formula is easy: we can computationally check if a certificate (i.e., a real algebraic number) returned by an external solver satisfies the quantifier-free part of the formula:

```
lemma ExQ_intro:
  fixes x::"alg_float" and qf_form::qf_form2
  assumes "qf_form2_interp qf_form [of_alg_float x]"
  shows "norm_form2_interp (ExQ (QF qf_form)) []"
```

where `x` of type `alg_float`

```
datatype alg_float =
    Arep "int poly" float float — representation of a real algebraic number
  | Flt float — a small optimization in case the number is dyadic rational
```

is a certificate that is supposed to be instantiated by an external solver. The function `of_alg_float` converts `x` from `alg_float` to `real`. In other words, to prove an existential formula:

```
"norm_form2_interp (ExQ (QF qf_form)) []"
```

we can computationally check the truth value of the quantifier-free part of the formula at `x`:

```
"qf_form2_interp qf_form [of_alg_float x]"
```

which is possible due to the sign determination procedure described in Sect. 5.

6.3 Universal Case

For the universal case, the core lemma is as follows:

```
lemma utilize_samples:
  fixes P::"real ⇒ bool" and decomps::"real set set"
    and samples::"real set" and f::"real set ⇒ real"
  assumes "⋃decomps = ℝ"
      and "∀d∈decomps. ∀x1∈d.∀x2∈d. P x1 = P x2"
      and "∀d∈decomps. f d∈d" and "bij_betw f decomps samples"
  shows "(∀x. P x) = (∀pt∈samples. P pt)"
```

where `bij_betw f decomps samples` states that `f::real set ⇒ real` is a bijective function between the decomposition `decomps::real set set` and the sample points `samples::real set`. Essentially, what the lemma `utilize_samples` shows is that given a predicate `P::real ⇒ bool`, an unbounded universal formula `∀x. P x` is equivalent to a bounded one `∀pt∈samples. P pt`, if the truth value of `P` is constant over each component of the decomposition: `∀d∈decomps. ∀x1∈d.∀x2∈d. P x1 = P x2`.

On top of the lemma `utilize_samples`, we similarly convert an unbounded univariate real formula into a bounded one:

```
lemma allQ_subst:
  fixes root_reps::"alg_float list" and pols::"float poly set"
    and qf_form::qf_form2
  defines "samples≡map of_alg_float (mk_samples root_reps)"
  assumes "Some pols = extractPols qf_form"
      and "ordered_reps root_reps"
      and "contain_all_roots root_reps pols"
      and "valid_list root_reps"
  shows "norm_form2_interp (AllQ (QF qf_form)) vs
    = (∀x ∈ (set samples). norm_form2_interp (QF qf_form) (x#vs))"
```

where

- `root_reps::alg_float list` is a certificate that should be instantiated by an external solver. More specifically, `root_reps` should be the representation of a list of real roots (in ascending order) of polynomials from the quantifier-free part of the target formula,
- `map of_alg_float (mk_samples root_reps)` constructs sample points from the representation of a list of roots,
- `extractPols qf_form` extracts polynomials from the quantifier-free part `qf_form`,

- `ordered_reps root_reps` and `valid_list root_reps` together ensure that the representation of roots are valid and those roots are in ascending order,
- `contain_all_roots roots_reps pols` checks if `root_reps` is a representation of all real roots of the polynomials `pols`. Specifically, by Sturm's theorem, the number of total distinct real roots of each `p ∈ pols` can be computed, which can be then compared with the number of `r ∈ root_reps` that `p(r)=0`.

Most importantly, all assumptions of the lemma `allQ_subst` and its right-hand side

```
(∀x ∈ (set samples). norm_form2_interp (QF qf_form) (x#vs))
```

can be computationally checked, through which we can prove an unbounded univariate universal formula: `norm_form2_interp (AllQ (QF qf_form)) vs`.

7 Linking to an External Solver

Certificates for both existential and universal cases can be produced by any program performing univariate CAD. For now, we implement the program on top of Mathematica. More specifically, the universal certificates are constructed by the Mathematica command *SemialgebraicComponentInstances*, which gives sample points in each connected component of a semialgebraic set. The existential certificates are constructed by the command *FindInstance*, which incorporates powerful numerical methods to accelerate the search for real algebraic sample points.

Also, it may be worth mentioning that after a certificate has been found, our tactic will record it (as a string) so that repeating the proof no longer requires the external solver. This is much like the sums-of-squares tactic [17].

In general, the certificate-based design grants us much flexibility: We can easily switch to a more efficient external solver without modifying existing formal proofs. In fact, we were first using an implementation of univariate CAD built within MetiTarski, which turned out to be not very efficient, and we simply switched to the current one based on Mathematica. In the future, we plan to experiment with other open-source CAD implementations such as Z3 and QEPCAD to provide more options with external solvers.

8 Experiments and Related Work

The most relevant work is the recent `tarski` strategy by Narkawicz et al. [25] in PVS. Both their work and ours rely on a formal proof of the Sturm–Tarski theorem (which they call Tarski's theorem) and handle roughly the same class of problems[4] (i.e., first-order univariate formulas over reals). There are two main differences between their work and ours:

- Their procedure resembles Tarski's original quantifier elimination [2, Chapter 2] and Cyril Cohen's quantifier elimination procedure in Coq [6, Chapter 12] by making use of both the Sturm–Tarski theorem and matrices. In contrast, our tactic is based on CAD and real algebraic numbers (instead of matrices).
- Their procedure is entirely built within PVS, while ours sceptically makes use of efficient external programs to generate certificates.

[4] In fact, their tactic does not handle arbitrary boolean expressions like ours, but we believe this should not be too hard to overcome.

$$\text{ex1}: \quad \forall x.\, \neg(x \geq -9x < 10 \wedge x^4 > 0) \vee x^{12} > 0$$

$$\begin{aligned}\text{ex2}: \quad & \forall x.\, \neg((x-2)^2(-x+4) > 0 \wedge x^2(x-3)^2 \geq 0 \\ & \wedge x - 1 \geq 0 \wedge -(x-2)^2 + 1 > 0) \vee (-(x - \frac{11}{12}))^3 (x - \frac{41}{10})^3 \geq 0\end{aligned}$$

$$\begin{aligned}\text{ex3}: \quad & \exists x.\, x^5 - x - 1 = 0 \wedge x^{12} + \frac{425}{23}x^{11} - \frac{228}{23}x^{10} - 2x^8 - \frac{896}{23}x^7 - \frac{394}{23}x^6 + \\ & \frac{456}{23}x^5 + x^4 + \frac{471}{23}x^3 + \frac{645}{23}x^2 - \frac{31}{23}x - \frac{228}{23} = 0 \wedge x^3 + 22x^2 - 31 \geq 0 \\ & \wedge x^{22} - \frac{234}{567}x^{20} - 419x^{10} + 1948 > 0\end{aligned}$$

$$\begin{aligned}\text{ex4}: \quad & \forall x.\, x > 0 \vee \frac{20}{9}x^3 + \frac{5}{9}x^2 - \frac{61}{9}x > -4 \vee 1 \leq x \vee x \leq 0 \vee \frac{10}{9}x^2 - \frac{19}{9}x \leq -1 \\ & \vee \frac{1}{18}x^3 + \frac{31}{45}x^2 - \frac{13}{9}x \leq -\frac{7}{10} \vee \frac{20}{9}x^3 + \frac{5}{9}x^2 - \frac{61}{9}x \leq -4\end{aligned}$$

$$\begin{aligned}\text{ex5}: \quad & \forall x.\, -\frac{x^3}{3} - \frac{10}{3}x^2 - \frac{5}{6}x > 0 \vee \frac{1}{3}x^3 + \frac{10}{3}x^2 + \frac{5}{6}x > 0 \vee 1 \leq x \vee x \leq 0 \\ & \vee \frac{10}{9}x^2 - \frac{19}{9}x \leq -1 \vee \frac{1}{18}x^3 + \frac{31}{45}x^2 - \frac{13}{9}x \leq -\frac{7}{10} \\ & \vee \frac{14}{15}x^3 - \frac{64}{15}x^2 - \frac{101}{30}x \leq -\frac{11}{5} \vee \frac{20}{9}x^3 + \frac{5}{9}x^2 - \frac{61}{9}x \leq -4\end{aligned}$$

$$\begin{aligned}\text{ex6}: \quad & \exists x.\, -70x^6 - \frac{2052}{5}x^5 - \frac{4329}{5}x^4 - \frac{5409}{10}x^3 - \frac{267}{2}x^2 - \frac{51}{10}x > -\frac{7}{10} \wedge \frac{49}{162}x^9 + \frac{49}{3}x^8 \\ & + \frac{175}{18}x^7 + \frac{115774}{405}x^6 + \frac{77743}{135}x^5 - \frac{57328}{135}x^4 - \frac{135853}{810}x^3 - \frac{71681}{270}x^2 - \frac{10327}{270}x > -\frac{721}{90} \\ & \wedge \frac{7}{27}x^8 + \frac{280}{27}x^7 - \frac{595}{54}x^6 + \frac{18964}{135}x^5 + \frac{2698}{135}x^4 - \frac{24217}{270}x^3 - \frac{251}{6}x^2 - \frac{2981}{90}x > -\frac{206}{45} \\ & \wedge \frac{7}{54}x^7 + \frac{112}{27}x^6 + \frac{329}{90}x^5 + \frac{2672}{135}x^4 - \frac{7933}{270}x^3 + \frac{169}{18}x^2 - \frac{799}{90}x > -\frac{103}{90} \wedge \frac{7}{27}x^8 + \frac{280}{27}x^7 \\ & + \frac{935}{54}x^6 + \frac{7264}{135}x^5 + \frac{11323}{135}x^4 - \frac{12217}{270}x^3 - \frac{701}{6}x^2 - \frac{781}{90}x > -\frac{77}{15} \wedge \frac{2}{9}x^7 + \frac{52}{9}x^6 - \frac{17}{6}x^5 \\ & + \frac{2353}{90}x^4 + \frac{307}{45}x^3 - \frac{811}{30}x^2 - \frac{361}{30}x > -\frac{44}{15} \wedge \frac{1}{9}x^6 + 2x^5 + \frac{2}{15}x^4 + \frac{41}{90}x^3 - \frac{2}{15}x^2 \\ & - \frac{33}{10}x > -\frac{11}{15} \wedge \frac{49}{162}x^8 + \frac{1540}{81}x^7 + \frac{1109}{27}x^6 + \frac{23483}{810}x^5 + \frac{65378}{405}x^4 - \frac{11549}{270}x^3 - \frac{70225}{324}x^2 \\ & - \frac{1339}{405}x > -\frac{721}{60} \wedge \frac{7}{27}x^7 + \frac{203}{18}x^6 - \frac{52}{9}x^5 + \frac{7753}{270}x^4 + \frac{5191}{180}x^3 - \frac{2263}{45}x^2 - \frac{10741}{540}x > -\frac{103}{15} \\ & \wedge \frac{2}{9}x^6 + \frac{59}{9}x^5 - \frac{493}{36}x^4 + \frac{2113}{90}x^3 - \frac{811}{180}x^2 - \frac{1481}{90}x > -\frac{22}{5} \wedge \frac{1}{9}x^5 + \frac{17}{9}x^4 - \frac{257}{60}x^3 + \frac{563}{90}x^2 \\ & - \frac{913}{180}x > -\frac{11}{10} \wedge \frac{20}{9}x^4 - \frac{5}{2}x^3 + \frac{10}{3}x^2 - \frac{91}{18}x > -2 \wedge \frac{10}{9}x^3 - \frac{25}{18}x^2 - \frac{2}{9}x > -\frac{1}{2} \wedge \frac{20}{9}x^3 \\ & + \frac{5}{9}x^2 - \frac{61}{9}x > -4 \wedge 1 > x \wedge x > 0 \wedge \frac{10}{9}x^2 - \frac{19}{9}x > -1 \wedge \frac{1}{18}x^3 + \frac{31}{45}x^2 - \frac{13}{9}x > -\frac{7}{10} \wedge \frac{1}{9}x^4 \\ & + \frac{34}{15}x^3 - \frac{53}{30}x^2 - \frac{253}{90}x > -\frac{11}{5} \wedge \frac{2}{9}x^5 + \frac{82}{9}x^4 + \frac{86}{15}x^3 - \frac{2051}{90}x^2 - \frac{97}{90}x > -\frac{44}{5} \wedge \frac{8}{81}x^8 \\ & + \frac{931}{81}x^7 + \frac{3113}{27}x^6 - \frac{289811}{1620}x^5 + \frac{264373}{810}x^4 + \frac{30583}{270}x^3 - \frac{298609}{810}x^2 - \frac{93307}{1620}x > -\frac{193}{5} \\ & \wedge \frac{7}{27}x^7 + \frac{38}{3}x^6 + \frac{28}{9}x^5 - \frac{2686}{135}x^4 + \frac{6397}{60}x^3 - \frac{9151}{90}x^2 - \frac{4741}{540}x > -\frac{77}{10}\end{aligned}$$

Fig. 3 Comparison between our tactic in Isabelle and the tarski strategy in PVS: univ_rcf includes certificate searching and checking, while univ_rcf_cert includes only checking

$$
\begin{aligned}
\text{ex7}: \quad & \forall x.\, x < -1 \vee 0 > x \vee \frac{1}{8}x^7 + \frac{1207}{35}x^6 + \frac{7083}{10}x^5 + 4983x^4 + \frac{64405}{4}x^3 + 26169x^2 \\
& + \frac{41613}{2}x > -6435 \vee 35x^{12} + 22461058620x^2 + 11821609800x \leq 46204x^{11} \\
& + 5263834x^{10} + 144537452x^9 + 1758662439x^8 + 10317027768x^7 + 31842714428x^6 \\
& + 54212099480x^5 + 45938678170x^4 + 4171407240x^3 \vee x \leq 0 \vee 753x^{10} + 58568x^9 \\
& + 938908x^8 + 6857016x^7 + 27930066x^6 + 68338600x^5 + 102560612x^4 + 92372280x^3 \\
& + 45805760x^2 + 9609600x \leq 0 \vee 10x^{11} + 1101329460x^2 + 788107320x \leq 9179x^{10} \\
& + 1061504x^9 + 24397102x^8 + 240283734x^7 + 1063536663x^6 + 2362290448x^5 \\
& + 2625491260x^4 + 782617220x^3 \vee 5x^{10} + 81290790x^2 + 90935460x \leq 2828x^9 \\
& + 356071x^8 + 6846880x^7 + 51834563x^6 + 161529144x^5 + 237512625x^4 \\
& + 125595120x^3 \vee 207x^9 + 11237x^8 + 138652x^7 + 794964x^6 + 2505504x^5 + 4581220x^4 \\
& + 4837448x^3 + 2735040x^2 + 640640x \leq 0 \vee 5x^8 \leq 608x^7 + 10261x^6 + 63520x^5 \\
& + 192458x^4 + 303324x^3 + 238560x^2 + 73920x \vee 98x^8 + 3514x^7 + 32711x^6 + 142928x^5 \\
& + 332962x^4 + 424284x^3 + 278880x^2 + 73920x \leq 0 \vee x \leq -1
\end{aligned}
$$

	Time (s)		
Formula	univ_rcf (Isabelle)	univ_rcf_cert (Isabelle)	tarski (PVS)
ex1	0.9	0.3	2.0
ex2	1.4	0.6	6.8
ex3	1.6	0.7	13.0
ex4	1.3	0.5	20.1
ex5	1.6	0.6	315.7
ex6	5.6	3.9	timeout
ex7	38.4	34.9	timeout

Note: timeout indicates failure to terminate within 24 hours

Fig. 3 continued

To compare both tactics empirically, we have conducted experiments on several typical examples from their paper[5] and the MetiTarski project[6] [29]. The experiments are run on a desktop with an Intel Core 2 Quad Q9400 (quad core, 2.66 GHz) CPU and 8 gigabytes RAM. Results of the experiments are illustrated in Fig. 3, where our `univ_rcf` tactic includes both certificate searching and checking process, while the `univ_rcf_cert` does the checking part only (when repeating a proof with certificates already recorded as a string).

In general, the experiments indicate that our tactic outperforms the `tarski` strategy in PVS. Particularly, the advantage of our tactic becomes greater as the problems become more complex, which can be attributed to the fact that our tactic has much better worst-case computational complexity (polynomial vs. exponential in the number of polynomials).

In the case of general multivariate problems, the CAD procedure is doubly exponential while Tarski's quantifier elimination procedure is non-elementary in the number of variables [2, Chapter 11]). When limited to univariate problems, the CAD procedure degenerates to root isolation and sign determination on a set of univariate polynomials, which is of polynomial complexity in the number of polynomials and their degree bound [2, Chapter 10]).

[5] http://shemesh.larc.nasa.gov/people/cam/Tarski/.

[6] http://www.cl.cam.ac.uk/~gp351/cicm2012/.

In comparison, Tarski's quantifier elimination procedure, even when limited to univariate problems, is still exponential in the number of polynomials [7].

In addition, it is worth noting that as the problems become more complex (e.g., ex6 and ex7 in Fig. 3), certificate checking becomes the bottleneck factor of our tactic (especially for universal problems). This indicates that, despite the fact that certificate searching is much harder than certificate checking, the Mathematica implementation is still much more efficient than our verified certificate-checking procedure. This leaves much room for future optimisations.

Our work has also been greatly inspired by Cyril Cohen's PhD thesis [6], within which a quantifier elimination procedure has been built upon the Sturm–Tarski theorem and real algebraic numbers formalised within the Coq theorem prover. However, our goals and approaches are very different.

Cohen's work is part of a large project that has formalised the Feit–Thompson theorem (odd order theorem) in Coq [15], and focuses more on theoretical developments than we do. For example, they proved the Sturm–Tarski theorem to construct an RCF quantifier elimination procedure in the spirit of Tarski's original method, which has important theoretical properties but is not practical as a proof procedure. Moreover, he has formalised arithmetic on real algebraic numbers and shown that they form a real closed field via resultants. We have not formalised resultants at all. Our sign determination algorithm uses the Sturm–Tarski theorem, which is significantly more efficient in practice than using resultants. On the other hand, as it was unnecessary for our proof procedure, we have not proved in Isabelle that the real algebraic numbers form a real closed field. In general, compared to his work, ours stresses the practical side over the theoretical. Fundamentally, we want to build procedures to solve non-trivial problems in practice.

Decision procedures based on Sturm's theorem have been implemented in Isabelle and PVS before [14,26]. Their core idea is to count the number of real roots within a certain (bounded or unbounded) interval. Generally, they can only handle formulas involving a single polynomial, so they are not complete for first-order formulas (unlike our tactic and the `tarski` strategy in PVS).

Assia Mahboubi [22] has implemented the executable part of a general CAD procedure in Coq, but as far as we know, the correctness proof for her implementation is still ongoing. This is also one of the reasons for us to choose the certificate-based approach rather than directly verifying an implementation.

There are other methods to handle nonlinear polynomial problems in theorem provers, such as sums of squares [17], which is good for multivariate universal problems but is not applicable when the existential quantifier arises, and interval arithmetic [18,34], which is very efficient for some cases but is not complete. These methods and ours should be used in a complementary way.

9 Discussion and Applications

One of our driving motivations is the integration of MetiTarski with Isabelle. MetiTarski [1] is a first-order theorem prover for real number inequalities involving transcendental functions such as sin, tan and exp. It can automatically prove formulas like

$$\forall x \in (0, 1.25).\ \tan(x)^2 \leq 1.75 \times 10^{-7} + \tan(1)\tan(x^2)$$

$$\forall x > 0.\ \frac{1 - e^{-2x}}{2x(1 - e^{-x})^2} - \frac{1}{x^2} \leq \frac{1}{12}$$

$$\forall x \in (0, 1).\ 1.914 \frac{\sqrt{1+x} - \sqrt{1-x}}{4 + \sqrt{1+x} + \sqrt{1-x}} \leq 0.01 + \frac{x}{2 + \sqrt{1-x^2}}.$$

The main idea behind MetiTarski is to approximate transcendental functions by polynomial or rational function bounds, and then solve the formula by a combination of a resolution theorem proving and an external Real Closed Field (RCF) decision procedure (QEPCAD, Mathematica or Z3). MetiTarski is a version of Joe Hurd's Metis prover [19], modified to include arithmetic simplification and integration with RCF decision procedures, along with many other refinements.

Applications of MetiTarski include verification problems arising in air traffic control [13] and analogue circuit designs [11]. As some of the applications are safety critical, it is natural to consider to integrate MetiTarski with an existing interactive theorem prover, whose internal logic can be used to ensure the correctness of MetiTarski's proofs. Besides, the automation provided by MetiTarski is generally useful to interactive theorem provers.

MetiTarski has been integrated with the PVS theorem prover [28] as a trusted oracle [12]. The authors state that the automation introduced by MetiTarski for closing sequents containing real-valued functions considerably outperforms existing tactics in PVS. However, this tactic should not be used in a certification environment, where external oracles are not allowed.

Our eventual goal is to integrate MetiTarski into the Isabelle/HOL theorem prover. Isabelle can verify purely logical inferences (in fact, it contains an internal copy of the Metis theorem prover), and the third author has just formalised most of the bounds of transcendental functions used by MetiTarski [30]. The primary remaining hurdle is the RCF decision procedure, and the work presented here is the first step towards it.

Finally, let us say a bit about how our work might be generalised to multivariate problems. In doing so, we plan to continue our certificate-based approach, as we are unlikely to implement a verified internal CAD procedure comparable in efficiency to a state-of-the-art implementation. It is still not obvious to us where the clear separation between *search* and *verification* should be in the multivariate case, but we have already made some progress:

- The bivariate sign determination procedure based on recursive application of the Sturm–Tarski theorem described in our previous work [21] can be easily generalised to a multivariate one (i.e., a procedure to decide the sign of a multivariate polynomial at real algebraic points), which can be then used to efficiently certify purely existential multivariate formulas over reals.
- Our recent formalisation of Cauchy's residue theorem [20] can be used to certify a key theorem used in general CAD: that the complex roots of a polynomial continuously depend on its coefficients.

10 Conclusion

We have described our work of building a procedure for first-order univariate polynomial problems in Isabelle/HOL. Compared to existing tactics among proof assistants, noticeable features of our tactic are

- It is based on univariate cylindrical algebraic decomposition (CAD).

- It sceptically integrates efficient external solvers in a certificate-based way, so that its soundness solely depends on Isabelle's logic (and code generation machinery) rather than the external solvers.

This is made possible by certificate-based approaches to real root isolation and sign-determination for evaluating polynomials at real algebraic points. As much of the novelty in our work is motivated by practical efficiency considerations, we have performed experiments comparing our procedure with another real algebraic proof procedure, the `tarski` method in PVS. By making use of efficient external solvers, our procedure is shown to empirically outperform this other method by substantial margins. We believe this adds further impetus to the certificate-based methods for a wide variety of formal proof procedures.

Certificate-based methods can be compared on the basis of how much mathematics and computation are required both to find and check their certificates. For example, to convert a Positivstellensatz certificate into a HOL-Light proof of a universal theorem, Harrison's sums-of-squares tactic only requires simple sign-based reasoning and rational arithmetic, while in our case, we need more mathematics (e.g., real algebraic numbers and the Sturm–Tarski theorem) and more computation (especially for the universal case). A good certificate design needs to balance the difficulty of the formalisation effort and verified computation required to check the certificates with the efficiency improvements offered by offloading the construction of the certificates to high-performance external tools.

Acknowledgements We thank Florian Haftmann for helping with code generation for our procedure. We are also grateful to the anonymous referees for their constructive suggestions.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Akbarpour, B., Paulson, L.: MetiTarski: an automatic theorem prover for real-valued special functions. J. Autom. Reason. **44**(3), 175–205 (2010)
2. Basu, S., Pollack, R., Roy, M.F.: Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics). Springer, New York (2006)
3. Brown, C.W.: QEPCAD B: a program for computing with semi-algebraic sets using CADs. ACM SIGSAM Bull. **37**(4), 97–108 (2003)
4. Chaieb, A., et al.: Automated methods for formal proofs in simple arithmetics and algebra. Dissertation, Technische Universität, München (2008)
5. Cheng, J.S., Gao, X.S., Yap, C.K.: Complete numerical isolation of real zeros in zero-dimensional triangular systems. In: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation, pp. 92–99. ACM (2007)
6. Cohen, C.: Formalized algebraic numbers: construction and first-order theory. Ph.D. thesis, École polytechnique (2012)
7. Cohen, C., Mahboubi, A., et al.: Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination. Log. Methods Comput. Sci. **8**(1: 02), 1–40 (2012)
8. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition: a synopsis. ACM SIGSAM Bull. **10**(1), 10–12 (1976)
9. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340. Springer, Berlin (2008)
10. De Moura, L., Passmore, G.O.: Computation in real closed infinitesimal and transcendental extensions of the rationals. In: International Conference on Automated Deduction, pp. 178–192. Springer, Berlin (2013)

11. Denman, W., Akbarpour, B., Tahar, S., Zaki, M.H., Paulson, L.C.: Formal verification of analog designs using MetiTarski. In: Formal Methods in Computer-Aided Design, 2009. FMCAD 2009, pp. 93–100. IEEE (2009)
12. Denman, W., Muñoz, C.: Automated real proving in PVS via MetiTarski. In: FM 2014: Formal Methods, pp. 194–199. Springer (2014)
13. Denman, W., Zaki, M.H., Tahar, S., Rodrigues, L.: Towards flight control verification using automated theorem proving. In: NASA Formal Methods, pp. 89–100. Springer (2011)
14. Eberl, M.: A decision procedure for univariate real polynomials in Isabelle/HOL. In: Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP '15, pp. 75–83. ACM, New York (2015). doi:10.1145/2676724.2693166
15. Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Le Roux, S., Mahboubi, A., O'Connor, R., Ould Biha, S., Pasca, I., Rideau, L., Solovyev, A., Tassi, E., Théry, L.: A machine-checked proof of the odd order theorem. In: Blazy S., Paulin-Mohring C., Pichardie D. (eds.) Interactive Theorem Proving: 4th International Conference, ITP 2013, Rennes, France, July 22–26. Lecture Notes in Computer Science, vol. 7998, pp. 163–179. Springer, Berlin (2013)
16. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: International Symposium on Functional and Logic Programming, pp. 103–117. Springer (2010)
17. Harrison, J.: Verifying nonlinear real formulas via sums of squares. In: K. Schneider, J. Brandt (eds.) Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2007, Lecture Notes in Computer Science, vol. 4732, pp. 102–118. Springer, Kaiserslautern (2007)
18. Hölzl, J.: Proving inequalities over reals with computation in Isabelle/HOL. In: International Workshop on Programming Languages for Mechanized Mathematics Systems, pp. 38–45 (2009)
19. Hurd, J.: Metis first order prover. http://gilith.com/software/metis (2007)
20. Li, W., Paulson, L.C.: A formal proof of Cauchy's residue theorem. In: ITP 2016: Seventh International Conference on Interactive Theorem Proving (2016, to appear)
21. Li, W., Paulson, L.C.: A modular, efficient formalisation of real algebraic numbers. In: Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, pp. 66–75. ACM (2016)
22. Mahboubi, A.: Implementing the cylindrical algebraic decomposition within the Coq system. Math. Struct. Comput. Sci. **17**(1), 99–127 (2007)
23. Mishra, B.: Algorithmic Algebra. Springer, New York (1993)
24. Muñoz, C., Narkawicz, A.: Formalization of Bernstein polynomials and applications to global optimization. J. Autom. Reason. **51**(2), 151–196 (2013). doi:10.1007/s10817-012-9256-3
25. Narkawicz, A., Munoz, C., Dutle, A.: Formally-verified decision procedures for univariate polynomial computation based on Sturm's and Tarski's theorems. J. Autom. Reason. **54**(4), 285–326 (2015)
26. Narkawicz, A.J., Muñoz, C.A.: A formally-verified decision procedure for univariate polynomial computation based on Sturm's theorem. Technical Memorandum NASA/TM-2014-218548, NASA, Langley Research Center, Hampton VA 23681-2199, USA (2014)
27. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer, Berlin (2002)
28. Owre, S., Rushby, J.M., Shankar, N.: PVS: a prototype verification system. In: International Conference on Automated Deduction, pp. 748–752. Springer (1992)
29. Passmore, G.O., Paulson, L.C., De Moura, L.: Real algebraic strategies for MetiTarski proofs. In: International Conference on Intelligent Computer Mathematics, pp. 358–370. Springer (2012)
30. Paulson, L.C.: Real-valued special functions: upper and lower bounds. Archive of Formal Proofs (2014)
31. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: IWIL-2010, vol. 1 (2010)
32. Rahman, Q., Schmeisser, G.: Analytic Theory of Polynomials. London Mathematical Society Monographs. Clarendon Press, Oxford (2002). https://books.google.co.uk/books?id=FzFEEVO3PXYC
33. Sagraloff, M.: A general approach to isolating roots of a bitstream polynomial. Math. Comput. Sci. **4**(4), 481–506 (2010)
34. Solovyev, A., Hales, T.C.: Formal verification of nonlinear inequalities with Taylor interval approximations. In: NASA Formal Methods, pp. 383–397. Springer, Berlin (2013)
35. Strzeboński, A.W.: Cylindrical algebraic decomposition using validated numerics. J. Symb. Comput. **41**(9), 1021–1038 (2006)
36. Thiemann, R., Yamada, A.: Algebraic numbers in Isabelle/HOL. Archive of Formal Proofs (2015). http://isa-afp.org/entries/Algebraic_Numbers.shtml. Formal proof development