Available online at www.sciencedirect.com

ScienceDirect

Journal of Symbolic Computation 41 (2006) 1021–1038

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

Cylindrical Algebraic Decomposition using validated numerics

Adam W. Strzeboński*

Wolfram Research Inc., 100 Trade Centre Drive, Champaign, IL 61820, USA

Received 10 March 2005; accepted 2 June 2006
Available online 7 August 2006

Abstract

We present a version of the Cylindrical Algebraic Decomposition (CAD) algorithm which uses interval sample points in the lifting phase, whenever the results can be validated. This gives substantial time savings by avoiding computations with exact algebraic numbers. We use bounds based on Rouche's theorem combined with information collected during the projection phase and during construction of the current cell to validate the singularity structure of roots. We compare empirically our implementation of this variant of CAD with implementations of CAD using exact algebraic sample points (our and QEPCAD) and with our implementation of CAD using interval sample points with validation based solely on interval data.
© 2006 Elsevier Ltd. All rights reserved.

Keywords: Cylindrical Algebraic Decomposition, CAD; Solving inequalities; Algebraic inequalities; Semialgebraic sets; Quantifier elimination

1. Introduction

The Cylindrical Algebraic Decomposition (CAD) algorithm is the best currently known algorithm for solving many classes of problems related to systems of real polynomial equations and inequalities. It not only allows elimination of quantifiers, but also provides an explicit description of the solution set in terms of real algebraic functions. The CAD algorithm has been used in practice in several areas including control system design (Dorato et al., 1997; Jirstrand, 1997), stability analysis (Hong et al., 1997), multidimensional integration and

* Tel.: +1 217 398 0700; fax: +1 217 398 0747.
E-mail address: adams@wolfram.com.

0747-7171/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jsc.2006.06.004

graphical representation of semialgebraic sets (Strzeboński, 2000a), and global optimization and assumption propagation in a computer algebra system (Strzeboński, 2000b).

The lifting phase of the CAD algorithm requires computation of roots of polynomials with algebraic number coefficients as well as determination of signs of multivariate polynomials at points with algebraic number coordinates. Several authors (Hong, 1993; Strzeboński, 1999; Collins et al., 2002; Anai and Yokoyama, 2005) explored the possibility of speeding up the CAD by using numeric approximations and isolating intervals instead of exact algebraic numbers. However, knowing isolating intervals is not sufficient to prove that a polynomial with algebraic number coefficients has a multiple real root, or two polynomials with algebraic number coefficients have a common root, or a multivariate polynomial is zero at a point with algebraic number coordinates. Hence, an algorithm using solely isolating interval information to validate the results has to use exact algebraic number computation each time a projection polynomial has a multiple root or two projection polynomials have a common root. Since multiple and common roots arise in CAD construction systematically, this is a serious disadvantage.

Our idea is to use information collected during CAD construction to validate the multiple and common root structure of projection polynomials.

Example 1.1. Let f be a univariate polynomial of degree n, let $U_{1,2}, U_3, \ldots, U_n$ be disjoint subsets of $\mathbb{C}$, such that $U_{1,2}$ contains two roots of f, multiplicities counted, and each U_i, $3 \leq i \leq n$, contains one root of f. If $Res(f, f') = 0$, then $U_{1,2}$ contains a double root of f. This criterion allows us to prove multiplicity of a root, even though we only have isolating sets for the roots, provided we know from the construction done so far that the resultant of f and f' is zero. Similar criteria using resultants and higher order principal subresultant coefficients can be used to obtain multiple and common root structure information required in the lifting phase of the CAD algorithm. Note that we need isolating set information for all complex roots.

2. Cylindrical Algebraic Decomposition

In this section we give a brief description of the CAD algorithm. For more details see Caviness and Johnson (1998) and Collins (1975).

A *system of polynomial equations and inequalities* in variables $x_1, \ldots, x_n$ is a formula

$$S(x_1, \ldots, x_n) = \bigvee_{1\leq i\leq l} \bigwedge_{1\leq j\leq m} f_{i,j}(x_1, \ldots, x_n)\rho_{i,j}0$$

where $f_{i,j} \in \mathbb{R}[x_1, \ldots, x_n]$, and each $\rho_{i,j}$ is one of $<, \leq, \geq, >, =$, or $\neq$.

A subset of $\mathbb{R}^n$ is *semialgebraic* if it is a solution set of a system of polynomial equations and inequalities.

A *quantified system of real polynomial equations and inequalities* in free variables $x_1, \ldots, x_n$ and quantified variables $t_1, \ldots, t_m$ is a formula

$$Q_1t_1 \ldots Q_mt_mS(t_1, \ldots, t_m; x_1, \ldots, x_n)$$

where Q_i is $\exists$ or $\forall$, and S is a system of real polynomial equations and inequalities in $t_1, \ldots, t_m, x_1, \ldots, x_n$.

By Tarski's theorem (Tarski, 1951), solution sets of quantified systems of real polynomial equations and inequalities are semialgebraic.

Every semialgebraic set can be represented as a finite union of disjoint *cells* (Łojasiewicz, 1964), defined recursively as follows.

1. A cell in $\mathbb{R}$ is a point or an open interval.
2. A cell in $\mathbb{R}^{k+1}$ has one of the two forms

$$\{(a_1, \ldots, a_k, a_{k+1}) : (a_1, \ldots, a_k) \in C_k \wedge a_{k+1} = r(a_1, \ldots, a_k)\}$$
$$\{(a_1, \ldots, a_k, a_{k+1}) : (a_1, \ldots, a_k) \in C_k \wedge r_1(a_1, \ldots, a_k) < a_{k+1} < r_2(a_1, \ldots, a_k)\}$$

where C_k is a cell in $\mathbb{R}^k$, r is a continuous algebraic function, and r_1 and r_2 are continuous algebraic functions, $-\infty$, or ∞, and $r_1 < r_2$ on C_k. By an algebraic function we mean a function $r : C_k \to \mathbb{R}$ for which there is a polynomial

$$f = c_0 x_{k+1}^m + c_1 x_{k+1}^{m-1} + \cdots + c_m \in \mathbb{R}[x_1, \ldots, x_k, x_{k+1}]$$

such that

$$c_0(a_1, \ldots, a_k) \neq 0 \wedge f(a_1, \ldots, a_k, r(a_1, \ldots, a_k)) = 0$$

for all $(a_1, \ldots, a_k) \in C_k$.

The CAD algorithm, introduced by Collins (Collins, 1975), allows us to compute a cell decomposition of any semialgebraic set presented by a quantified system of polynomial equations and inequalities. The objective of the original Collins algorithm was to eliminate quantifiers from a quantified system of polynomial equations and inequalities and to produce an equivalent quantifier-free system of polynomial equations and inequalities. After finding a cell decomposition the algorithm performed an additional step of finding an implicit representation of the semialgebraic set in terms of polynomial equations and inequalities in the free variables. Our objective here is somewhat different. Given a semialgebraic set presented by a system of polynomial equations and inequalities, quantified or not, we find a cell decomposition of the set, explicitly written in terms of algebraic functions represented as follows.

A *real algebraic function* given by a polynomial $f(x_1, \ldots, x_n, y)$ and an integer p is the function

$$Root_{y,p} f : \mathbb{R}^n \ni x_1, \ldots, x_n \longrightarrow Root_{y,p} f(x_1, \ldots, x_n) \in \mathbb{R} \tag{1}$$

where $Root_{y,p} f(x_1, \ldots, x_n)$ is the pth real root of $f(x_1, \ldots, x_n, y)$ treated as a univariate polynomial in y. The function is defined for those values of $x_1, \ldots, x_n$ for which $f(x_1, \ldots, x_n, y)$ has at least p real roots. The real roots are ordered by increasing value, counting multiplicities. (See Strzeboński (1996, 2000a,b) for more details on how algebraic functions can be implemented in a computer algebra system.)

Finding a cell decomposition of a semialgebraic set S using the CAD algorithm consists of two phases, projection and lifting. For simplicity we will assume the set S is presented by a quantifier-free system, see Collins and Hong (1991) and Caviness and Johnson (1998) for more details on quantifier elimination. In the projection phase we start with the set A_n of factors of the polynomials present in the system, and eliminate variables one by one using a projection operator P such that

$$P_{k+1} : \mathbb{R}[x_1, \ldots, x_k, x_{k+1}] \supseteq A_{k+1} \longrightarrow A_k \subseteq \mathbb{R}[x_1, \ldots, x_k]$$

and, generally speaking, if all polynomials of A_k have constant signs on a cell $C \subseteq \mathbb{R}^k$, then all polynomials of A_{k+1} are *delineable* over C, i.e. each has a fixed number of real roots on C as a polynomial in x_{k+1}, the roots are continuous functions on C, they have constant multiplicities, and two roots of two of the polynomials are equal either everywhere or nowhere in C. This way the roots of polynomials of $A_1, \ldots, A_n$ are the algebraic functions needed in the construction of the cell decomposition of S.

There have been several improvements made reducing the size of the original Collins projection. The currently best projection operator applicable in all cases is due to Hong (Hong, 1990), however in most situations we can use a smaller projection operator given by McCallum (McCallum, 1988, 1998), with an improvement by Brown (Brown, 2001). There are even smaller projection operators that can be applied in some special cases. When equational constraints are present we can use the projection operator suggested by Collins in Collins (1998), and developed and proven by McCallum in McCallum (1999, 2001). When there are no equations and only strict inequalities, and there are no free variables, or we are interested only in the full-dimensional part of the semialgebraic set we can use the projection operator given in Strzeboński (1994, 2000a).

In the lifting phase we find a cell decomposition of S. Generally speaking (the actual details depend on the projection operator used), we start with cells in $\mathbb{R}^1$ consisting of all distinct roots of A_1, and the open intervals obtained by removing the roots from $\mathbb{R}^1$. We find a sample point in each of the cells, and remove the cells whose sample points do not satisfy the system describing the semialgebraic set (the system may contain conditions involving only x_1). Now we lift the cells to cells in $\mathbb{R}^n$, one dimension at a time. Suppose we have lifted the cells to $\mathbb{R}^k$. To lift a cell $C \subseteq \mathbb{R}^k$ to $\mathbb{R}^{k+1}$ we find the real roots of A_{k+1} with $x_1, \ldots, x_k$ replaced with the coordinates of the sample point c in C. Since the polynomials of A_{k+1} are delineable on C, each root r is a value of a continuous algebraic function at c, and the function can be represented (as in (1)) by a polynomial $f \in A_{k+1}$ such that $f(c, r) = 0$ and a root number p such that r is the pth root of $f(c, x_{k+1})$. The lifting of the cell C to $\mathbb{R}^{k+1}$ consists of graphs of these algebraic functions, and of the slices of $C \times \mathbb{R}$ obtained by removing the graphs. The sample points in each of the new cells are obtained by adding the $k+1$th coordinate to c equal to one of the roots, or to a number between two subsequent roots. Similarly as in the first step, we remove those lifted cells whose sample points do not satisfy the system describing the semialgebraic set. Cells in $\mathbb{R}^n$ constructed during the lifting phase give us a cell decomposition of S.

3. Improvement

The coordinates of sample points computed in the lifting phase of CAD are in general algebraic numbers. Algebraic number computations have a high complexity and often dominate the computation time of CAD. Our improvement is to compute only intervals containing the sample point coordinates, use information collected during the construction done so far to validate the root structure results, and revert to algebraic number computation only if we do not have enough information to validate the results.

Let us state precisely the problems that need to be solved in order to lift a cell.

Problem 3.1 (*Root Structure*). Given a finite set of polynomials

$$A \subset \mathbb{Z}[x_1, \ldots x_k, x_{k+1}]$$

where $k \geq 0$, and a tuple $a = (a_1, \ldots, a_k)$ of algebraic numbers, decide for which polynomials $f \in A$, $f(a, x_{k+1})$ is identically zero, and find algebraic numbers $r_1 < \cdots < r_l$ and a function *mult* $: A \times \{1, \ldots, l\} \rightarrow \mathbb{N}$, such that $r_1, \ldots, r_l$ are all real roots of polynomials $\{f(a, x_{k+1}) : f \in A\}$, and r_i is a root of $f(a, x_{k+1})$ of multiplicity *mult*(f, i), for all $f \in A$ and $1 \leq i \leq l$.

Problem 3.2 (*Polynomial Sign*). Given a polynomial $f \in \mathbb{Z}[x_1, \ldots x_k]$, where $k \geq 1$, and a tuple $a = (a_1, \ldots, a_k)$ of algebraic numbers, find the sign of $f(a)$.

Polynomial sets A for which we need to compute root structure and polynomials f for which we need to compute signs depend on the projection used, but the problems that need to be solved always have this form. For details see Hong (1990), McCallum (1988, 1998, 1999, 2001) and Brown (2001). Algorithms solving Problems 3.1 and 3.2 are standard parts of the CAD algorithm and from now on we will assume that we have algorithms *ExactRootStructure* and *ExactPolynomialSign* that solve the problems.

In our version of the algorithm instead of exact values of $a = (a_1, \ldots, a_k)$ we have intervals $I_1, \ldots, I_k$ such that $a_i \in I_i$, for $1 \le i \le k$, and a list of properties of a we collected during the construction done so far. The possible properties are

- *Polynomial*(f) means that $f \in \mathbb{Z}[x_1, \ldots x_k]$ is zero at a,
- *Coefficient*(f, y, j) means that the coefficient at y^j of $f \in \mathbb{Z}[x_1, \ldots x_k, y]$ is zero at a,
- *PSC*(f, y, j) means that the jth principal subresultant coefficient of f and $\partial f/\partial y$ in y is zero at a,
- *PSC*(f, g, y, j) means that the jth principal subresultant coefficient of f and g in y is zero at a.

Note that the *PSC* notation includes resultants, as $Res(f, g, y) = PSC_0(f, g, y)$. In order to lift a cell we need to solve the following problems.

Problem 3.3 (*Root Structure*). Given a finite set of polynomials

$$A \subset \mathbb{Z}[x_1, \ldots x_k, x_{k+1}]$$

where $k \ge 0$, a tuple $(I_1, \ldots, I_k)$ of intervals, such that $a_i \in I_i$, for $1 \le i \le k$, and a list L of properties of $a = (a_1, \ldots, a_k)$, decide for which polynomials $f \in A$, $f(a, x_{k+1})$ is identically zero, and find intervals $J_1 < \cdots < J_l$ and a function $mult : A \times \{1, \ldots, l\} \to \mathbb{N}$, such that the polynomials $\{f(a, x_{k+1}) : f \in A\}$ have l distinct real roots $r_1, \ldots, r_l$, and $r_i \in J_i$ is a root of $f(a, x_{k+1})$ of multiplicity $mult(f, i)$, for all $f \in A$ and $1 \le i \le l$.

Problem 3.4 (*Polynomial Sign*). Given a polynomial $f \in \mathbb{Z}[x_1, \ldots x_k]$, where $k \ge 1$, a tuple $(I_1, \ldots, I_k)$ of intervals, such that $a_i \in I_i$, for $1 \le i \le k$, and a list L of properties of $a = (a_1, \ldots, a_k)$, find the sign of $f(a)$.

4. Subprocedures and lemmas

Before we present algorithms for solving Problems 3.3 and 3.4 we need to describe subprocedures and lemmas used by the algorithms.

4.1. Stored information

During the projection phase we annotate each projection polynomial p with "genealogy" information. The possible annotations are

- *Polynomial*(f) when p is a factor of a polynomial f from the input system,
- *Coefficient*(f, x, j) when p is a factor of a coefficient at x^j of f,
- *PSC*(f, x, j) when p is a factor of the jth principal subresultant coefficient of f and $\partial f/\partial x$ in x,
- *PSC*(f, g, x, j) when p is a factor of the jth principal subresultant coefficient of f and g in x.

With each sample point constructed during the lifting phase we store a list of properties. The list of properties of a sample point $(a_1, \ldots, a_k, a_{k+1})$, where $k \geq 0$, is taken to be the list of all properties of $(a_1, \ldots, a_k)$ and all annotations of elements f of A such that $f(a_1, \ldots, a_k, a_{k+1}) = 0$.

Each time we add a coordinate a_{k+1} to a sample point $(a_1, \ldots, a_k)$ we store a polynomial $f \in A$ and a number p such that a_{k+1} is the pth real root of $f(a, x_{k+1})$ or, if a_{k+1} is a rational number between consecutive roots of polynomials from A, the value of the rational number.

Whenever we compute exact algebraic number values of sample point coordinates $(a_1, \ldots, a_k)$ we store all algebraic numbers computed in the process (see the next subsection).

4.2. *Reverting to algebraic number computations*

When we cannot solve Problem 3.3 or Problem 3.4 using intervals containing coordinates of the sample point $(a_1, \ldots, a_k)$ and the properties of the sample point, we need to compute the exact algebraic number values of $a_1, \ldots, a_k$. In the first step of the lifting phase we stored a univariate polynomial f and a number p such that a_1 is the pth root of f, or, if a_1 is a rational number we stored its value. This suffices to compute algebraic number representation of a_1. Suppose we have computed the algebraic number values of $a_1, \ldots, a_l$. If a_{l+1} is a rational number between consecutive roots of projection polynomials, we have stored the value of a_{l+1}. Otherwise we have stored a polynomial $f \in \mathbb{Z}[x_1, \ldots x_l, x_{l+1}]$ and a number p, such that a_{l+1} is the pth real root of $f(a_1, \ldots a_l, x_{l+1})$. If we have reverted to algebraic number computations for some cell whose sample point extends $(a_1, \ldots, a_l)$, we might have computed and stored all real roots of $f(a_1, \ldots a_l, x_{l+1})$. If not, we compute and store all real roots of $f(a_1, \ldots a_l, x_{l+1})$ (see Strzeboński (1997)). We obtain a_{l+1} by selecting the pth real root of $f(a_1, \ldots a_l, x_{l+1})$.

4.3. *Roots of interval polynomials*

Our procedure finding complex roots of interval polynomials is based on the following.

Proposition 4.1. *Let $f \in \mathbb{C}[x]$ be a polynomial of degree n, $x_0 \in \mathbb{C}$, $r > 0$, and let $c_i := |\frac{f^{(i)}(x_0)}{i!}|$. Suppose that*

$$\max_{0 \leq i < k} \left(\frac{nc_i}{c_k}\right)^{\frac{1}{k-i}} < r < \min_{k < i \leq n} \left(\frac{c_k}{nc_i}\right)^{\frac{1}{i-k}}.$$

Then f has exactly k roots, multiplicities counted, in the disc $|x - x_0| < r$.

Proof. If $0 \leq i < k$, $(\frac{nc_i}{c_k})^{\frac{1}{k-i}} < r$. Hence $\frac{nc_i}{c_k} < r^{k-i}$, and so $c_i r^i < \frac{1}{n} c_k r^k$. If $k < i \leq n$, $r < (\frac{c_k}{nc_i})^{\frac{1}{i-k}}$. Hence $r^{i-k} < \frac{c_k}{nc_i}$, and so $c_i r^i < \frac{1}{n} c_k r^k$ as well. Put

$$F(x) := \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

$$G(x) := \sum_{i=0}^{k-1} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i + \sum_{i=k+1}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i.$$

For $|x - x_0| = r$, $|G(x)| \leq \sum_{i=0}^{k-1} c_i r^i + \sum_{i=k+1}^{n} c_i r^i < n(\frac{1}{n} c_k r^k) = |F(x)|$. Hence, by Rouche's theorem, $f(x) = F(x) + G(x)$ and $F(x)$ have the same number of roots in the disc $|x - x_0| < r$, multiplicities counted. □

Algorithm 4.2 (*IntervalRoots*).
Input: Real intervals $I_0, \ldots, I_n$, $0 \notin I_n$.
Output: Complex numbers $z_1, \ldots, z_m$, positive numbers $r_1, \ldots, r_m$, and positive integers $k_1, \ldots, k_m$, such that $k_1 + \cdots + k_m = n$, for any $a_0 \in I_0, \ldots, a_n \in I_n$ and any $1 \leq i \leq m$ polynomial $a_n x^n + \cdots + a_0$ has exactly k_i roots in the disc $D_i := |x - z_i| < r_i$, multiplicities counted, for any $1 \leq i < j \leq m$, $D_i \cap D_j = \emptyset$, and if $D_i \cap \mathbb{R} \neq \emptyset$, then $z_i \in \mathbb{R}$. The other possible output is *Failed*.

1. Set $f_c = a_n x^n + \cdots + a_0$, where $a_0, \ldots, a_n$ are center points of $I_0, \ldots, I_n$. Use a numeric root finding algorithm to find approximations of all complex roots of f_c.
2. Use a heuristic to identify root clusters, select representatives $z_1, \ldots, z_m$ for root clusters and let $k_1, \ldots, k_m$ be equal to the number of root approximations in the corresponding cluster. Select real numbers to represent clusters that contain real roots or contain roots on both sides of the real axis.
3. Let $f = I_n x^n + \cdots + I_0$. For each $1 \leq j \leq m$ and $0 \leq i \leq n$ set
$$c_{i,j} := \left| \frac{f^{(i)}(z_j)}{i!} \right|.$$
Using interval arithmetic compute r_j and s_j such that
$$\max_{0 \leq i < k_j} \left(\frac{n c_{i,j}}{c_{k_j,j}} \right)^{\frac{1}{k_j - i}} < r_j$$
and
$$s_j < \min_{k_j < i \leq n} \left(\frac{c_{k_j}}{n c_i} \right)^{\frac{1}{i - k_j}}.$$
4. If any of the r_j and s_j are infinite or $s_j < r_j$ for some j return *Failed*.
5. Set $D_j := |x - z_j| < r_j$. If for any $1 \leq i < j \leq m$, $D_i \cap D_j \neq \emptyset$, or if $D_i \cap \mathbb{R} \neq \emptyset$ and $z_i \notin \mathbb{R}$ return *Failed*.
6. Return $z_1, \ldots, z_m, r_1, \ldots, r_m$, and $k_1, \ldots, k_m$.

Our implementation of *IntervalRoots* uses *Mathematica* arbitrary-precision floating point numbers to represent intervals. Arbitrary-precision numbers are pairs (a, pa) where pa is a positive number and either a is a nonzero floating point number with at least pa digits and the pair represents the interval
$$(a - 10^{-pa}|a|, a + 10^{-pa}|a|)$$
or $a = 0$ and the pair represents the interval $(-10^{-pa}, 10^{-pa})$. The arithmetic of arbitrary-precision numbers is in effect interval arithmetic (see Keiper and Withoff (1992) and Wolfram (2003)). To compute root approximations our implementation uses *Mathematica* function NRoots, which implements the Jenkins–Traub method (Jenkins, 1969). Other numeric methods could be used as well. The correctness of the algorithm does not depend on the numerical method used. If another numerical method gives worse approximations, *IntervalRoots* will return larger radii $r_1, \ldots, r_m$ or *Failed*.

Mathematica function NRoots returns root approximations represented as arbitrary-precision numbers, where the sizes of intervals (or rectangles in the complex plane) represented by the numbers are determined by the numeric algorithm. We use the interval size information in the heuristic for determining root clusters required in step 2 (connected components of the union of all intervals give the different clusters). We used this heuristic because it is simple and works well in practice. Another possible heuristic, not using the error estimates provided by the numeric method, would be to use Proposition 4.1 for each approximation to compute the disc radius corresponding to the lowest k for which the proposition applies, and then determine root clusters as connected components of the union of so obtained discs.

4.4. Validation of root structure

Our method of validating multiple and common root structure is based on the following two propositions.

Proposition 4.3. *Let $f \in \mathbb{C}[x]$ be a polynomial of degree n, let $U_1, \ldots, U_s$ be pairwise disjoint subsets of $\mathbb{C}$, such that f has at least k_i roots, multiplicities counted, in U_i. Suppose that $k_1 + \cdots + k_s = n$, and $PSC_0(f, f', x) = \cdots = PSC_{n-s-1}(f, f', x) = 0$. Then f has exactly one root in U_i of multiplicity k_i, for $1 \le i \le s$. Moreover, if $f \in \mathbb{R}[x]$ and for each U_i such that $U_i \cap \mathbb{R} \neq \emptyset$, $(U_i \cup \overline{U_i}) \cap U_j = \emptyset$, for $j \neq i$, then the real roots of f are exactly the roots of f inside U_i with $U_i \cap \mathbb{R} \neq \emptyset$.*

Proof. Follows from basic properties of principal subresultant coefficients (see e.g. Collins (1975)). □

Proposition 4.4. *Let $f, g \in \mathbb{C}[x]$ be polynomials of degrees n and m. Let $u_1 \in U_1, \ldots, u_s \in U_s$ and $v_1 \in V_1, \ldots, v_t \in V_t$ be such that*

1. *u_i is a root of f of multiplicity k_i for $1 \le i \le s$,*
2. *v_j is a root of g of multiplicity l_j for $1 \le j \le t$,*
3. *$k_1 + \cdots + k_s = n$,*
4. *$l_1 + \cdots + l_t = m$,*
5. *$U_i \cap U_j = \emptyset$, for $i \neq j$,*
6. *$V_i \cap V_j = \emptyset$, for $i \neq j$,*
7. *$U_i \cap V_i \neq \emptyset$, for $1 \le i \le r$,*
8. *$U_i \cap V_j = \emptyset$, for $i \neq j$ or $i = j > r$.*

Let $d = \min(k_1, l_1) + \cdots + \min(k_r, l_r)$. If $PSC_0(f, g, x) = \cdots = PSC_{d-1}(f, g, x) = 0$, then $u_i = v_i$ for $1 \le i \le r$. Moreover, if $r = 1$ and $PSC_0(f, g, x) = 0$, then $u_1 = v_1$.

Proof. Follows from basic properties of principal subresultant coefficients (see e.g. Collins (1975)). □

5. The main algorithms

To compute the root structure (Problem 3.3) we use the following algorithm.

Algorithm 5.1 (*IntervalRootStructure*).
Input: A finite set of polynomials $A \subset \mathbb{Z}[x_1, \ldots x_k, x_{k+1}]$, where $k \ge 0$, a tuple $(I_1, \ldots, I_k)$ of intervals, such that $a_i \in I_i$, for $1 \le i \le k$, and a list L of properties of $a = (a_1, \ldots, a_k)$.
Output: The set *idzero* of polynomials $f \in A$, such that $f(a, x_{k+1})$ is identically zero, intervals $J_1 < \cdots < J_l$ and a function *mult* $: A \times \{1, \ldots, l\} \to \mathbb{N}$, such that the polynomials $\{f(a, x_{k+1}) : f \in A\}$ have l distinct real roots $r_1, \ldots, r_l$, and $r_i \in J_i$ is a root of $f(a, x_{k+1})$ of multiplicity *mult*(f, i), for all $f \in A$ and $1 \le i \le l$.

1. Set *idzero* $:= \emptyset$. If $k = 0$, represent the polynomials in A in terms of a square free and relatively prime basis, use a real root isolation algorithm to get disjoint isolating intervals $J_1 < \cdots < J_l$ for real roots of the basis polynomials, and compute root multiplicity function *mult* using representation of polynomials in A in terms of the basis polynomials. Return *idzero*, $J_1, \ldots, J_l$, and *mult*.
2. For each polynomial $f \in A$:
   (a) Set $f_{\text{red}} := f$. Let $f_{\text{red}} = c_n x_{k+1}^n + \cdots + c_0$. While $\mathit{Coefficient}(f, x_{k+1}, n)$ is present in L, set $f_{\text{red}} := f_{\text{red}} - c_n x_{k+1}^n$ and $n := \mathit{degree}_{x_{k+1}} f_{\text{red}}$.
   (b) If $f_{\text{red}} = 0$, $f(a, x_{k+1})$ is identically zero. Append f to *idzero* and continue the loop.
   (c) Substitute $(I_1, \ldots, I_k)$ for $(x_1, \ldots, x_k)$ in $c_0, \ldots, c_n$. Perform interval arithmetic operations to obtain interval coefficients $C_0, \ldots, C_n$.
   (d) If $0 \in C_n$, compute algebraic number values of $(a_1, \ldots, a_k)$. While $0 \in C_n$, use *ExactPolynomialSign* to compute $s := \mathit{sign}(c_n(a))$. If $s = 0$, set $f_{\text{red}} := f_{\text{red}} - c_n x_{k+1}^n$ and $n := \mathit{degree}_{x_{k+1}} f_{\text{red}}$, otherwise go to step 5.
   (e) Call *IntervalRoots* with $C_0, \ldots, C_n$. If the returned result is *Failed*, go to step 5.
   (f) Let $z_{f,1}, \ldots, z_{f,m_f}$, $r_{f,1}, \ldots, r_{f,m_f}$, and $k_{f,1}, \ldots, k_{f,m_f}$ be the output of *IntervalRoots*, and let μ be maximal such that $\mathit{PSC}(f_{\text{red}}, x_{k+1}, j)$, for $0 \le j < \mu$, are all present in L.
   (g) If $\mu < n - m_f$, compute $p_j := \mathit{PSC}_j(f_{\text{red}}, \partial f_{\text{red}}/\partial x_{k+1}, x_{k+1})$, for $\mu \le j < n - m_f$. Find algebraic number values of $(a_1, \ldots, a_k)$, and use *ExactPolynomialSign* to compute $s_j := \mathit{sign}(p_j(a))$. If $s_j \ne 0$, for some $\mu \le j < n - m_f$, go to step 5.
   (h) Proposition 4.3 guarantees that polynomial f_{red} has exactly one root in the disc $D_{f,i} := |x - z_{f,i}| < r_{f,i}$, of multiplicity $k_{f,i}$, for $1 \le i \le m_f$, and the root is real iff $z_{f,i} \in \mathbb{R}$.
3. For each pair of polynomials $f, g \in A$:
   (a) If for some i and $j_1 \ne j_2$, $D_{f,i} \cap D_{g,j_1} \ne \emptyset$ and $D_{f,i} \cap D_{g,j_2} \ne \emptyset$ or $D_{g,i} \cap D_{f,j_1} \ne \emptyset$ and $D_{g,i} \cap D_{f,j_2} \ne \emptyset$, go to step 5.
   (b) Let, possibly after a renumbering of roots, $D_{f,i} \cap D_{g,i} \ne \emptyset$, for $1 \le i \le p_{f,g}$, and $D_{f,i} \cap D_{g,j} = \emptyset$, for $i \ne j$ or $i = j > r$. Set $d_{f,g} := \min(k_{f,1}, k_{g,1}) + \cdots + \min(k_{f,p_{f,g}}, k_{g,p_{f,g}})$.
   (c) Let μ be maximal such that $\mathit{PSC}(f_{\text{red}}, g, x_{k+1}, j)$ or $\mathit{PSC}(f, g_{\text{red}}, x_{k+1}, j)$ is present in L, for each $0 \le j < \mu$.
   (d) If $p_{f,g} = 1$ and $\mu = 0$ or $p_{f,g} > 1$ and $\mu < d_{f,g}$, compute $p_j := \mathit{PSC}_j(f_{\text{red}}, g_{\text{red}}, x_{k+1})$, for $\mu \le j < d_{f,g}$. Find algebraic number values of $(a_1, \ldots, a_k)$, and use *ExactPolynomialSign* to compute $s_j := \mathit{sign}(p_j(a))$. If $s_j \ne 0$, for some $\mu \le j < d_{f,g}$, go to step 5.
   (e) Lemma 1 of Hong (1990) and Proposition 4.4 guarantee that $D_{f,i} \cap D_{g,i}$ contains a common root of f and g, for $1 \le i \le p_{f,g}$.
4. Let D_j, for $1 \le j \le m$, be all $D_{f,i}$, for $f \in A$, such that $z_{f,i} \in \mathbb{R}$, ordered by the left endpoints. Let $D_{j_{i-1}+1} \cup \cdots \cup D_{j_i}$, for $0 = j_0 < j_1 < \cdots < j_l = m$, be the connected components of $D_1 \cup \cdots \cup D_m$. Set $J_i := D_{j_{i-1}+1} \cap \cdots \cap D_{j_i}$, for $1 \le i \le l$. Step 3 guarantees that J_i are nonempty and $\mathit{mult}(f, i)$ is equal to $k_{f,j}$ if $D_{f,j}$ is among $D_{j_{i-1}+1}, \ldots, D_{j_i}$ and is zero otherwise. Return $J_1, \ldots, J_l$ and *mult*.
5. Revert to algebraic number computations. Compute algebraic number values of $(a_1, \ldots, a_k)$. Use *ExactRootStructure* to find the set *idzero* of polynomials $f \in A$, such that $f(a, x_{k+1})$ is identically zero, and find algebraic numbers $r_1 < \cdots < r_l$ and a function $\mathit{mult} : A \times \{1, \ldots, l\} \to \mathbb{N}$, such that $r_1, \ldots, r_l$ are all real roots of polynomials $\{f(a, x_{k+1}) : f \in A\}$, and r_i is a root of $f(a, x_{k+1})$ of multiplicity $\mathit{mult}(f, i)$, for all $f \in A$ and $1 \le i \le l$. Return *idzero*, disjoint isolating intervals of $r_1, \ldots, r_l$, and *mult*.

Correctness of this algorithm follows from Propositions 4.1, 4.3 and 4.4, and Lemma 1 of Hong (1990).

To compute polynomial signs (Problem 3.4) we use the following algorithm.

Algorithm 5.2 (*IntervalPolynomialSign*).
Input: A polynomial $f \in \mathbb{Z}[x_1, \dots x_k]$, where $k \geq 1$, a tuple $(I_1, \dots, I_k)$ of intervals, such that $a_i \in I_i$, for $1 \leq i \leq k$, and a list L of properties of $a = (a_1, \dots, a_k)$.
Output: The sign of $f(a)$.

1. If *Polynomial*(f) is present in L, return 0.
2. Substitute $(I_1, \dots, I_k)$ for $(x_1, \dots, x_k)$ in f. Perform interval arithmetic operations to obtain interval J. If $J > 0$ return 1. If $J < 0$ return -1.
3. Revert to algebraic number computations. Compute algebraic number values of $(a_1, \dots, a_k)$. Return the sign of $f(a)$ found using *ExactPolynomialSign*.

The correctness of this algorithm is obvious.

Remark 5.3. When selecting isolating intervals for algebraic numbers in steps 1 and 5 of *IntervalRootStructure* we can choose the size of the interval, as long as it is small enough to make all the intervals disjoint. In our implementation we use interval size corresponding to binary precision pr, that is the interval for an algebraic number $a \neq 0$ has a radius at most $2^{-pr}|a|$. By default $pr = 100$.

Remark 5.4. In our implementation we do not replace rational number sample point coordinates with intervals. More precisely, we know that a coordinate of a sample point is a rational number, either in steps 1 or 5 of *IntervalRootStructure* or when we extend a sample point by a rational number between roots of projection polynomials. In such a case we leave it as a rational number. *Mathematica* implements mixed arithmetic of intervals (represented as arbitrary-precision numbers) and rational numbers.

When all coordinates of a sample point a are represented as rational numbers our implementation uses exact real root isolation methods instead of *IntervalRootStructure* to find the root structure, and exact rational number arithmetic instead of *IntervalPolynomialSign* to determine polynomial signs. Experiments suggest that this is most of the time a better solution. Examples where computing with approximate numbers for rational sample points was faster involved low degree random polynomial systems with large integer coefficients.

Remark 5.5. In step 2(h) of *IntervalRootStructure*, after we have successfully validated the multiplicity structure of roots of f_{red}, we can compute bounding discs for multiple roots of f_{red} by applying a numerical root finding algorithm and Proposition 4.1 to $z_{f,i}$ and $f_{\text{red}}^{(k_{f,i}-1)}$. If a so obtained bounding disc is a proper subset of $D_{f,i}$, which is usually the case, our implementation replaces $D_{f,i}$ with the smaller disc.

Remark 5.6. Annotations telling which polynomial is zero on a given cell are also used by our implementation of CAD using exact algebraic number coordinates of sample points. They require no additional computations, just keeping track of information, and allow us to reduce the number of necessary sign computations. In some examples introducing the annotations reduced the CAD computation time by up to 25%.

6. Example

Let us describe computation of a CAD of $f \leq 0$, for

$$f = (y-1)^3 - (x^3-3)(y-1) + x(x^3-3).$$

The two variable projection set consists of f, annotated *Polynomial*(f). Using McCallum's projection operator we obtain the univariate projection set

$$\begin{aligned} g_1 &= x^3 - 3 \\ g_2 &= 4x^3 - 27x^2 - 12 \end{aligned}$$

both polynomials annotated $PSC(f, y, 0)$. Using exact real root isolation we find out that g_1 and g_2 have one real root each, and $a_{1,1} = Root_{x,1}(g_1) < a_{1,2} = Root_{x,1}(g_2)$. Isolating intervals of the roots in the arbitrary-precision number format are

$$\begin{aligned} I_{1,1} &= (1.4422495703074083823216383107801095883918692534999, 30.103) \\ I_{1,2} &= (6.8146011676829401651989664633326507946898493779\,13, 30.103) \end{aligned}$$

where (a, pa) represents the interval $(a - 10^{-pa}|a|, a + 10^{-pa}|a|)$. The cells in $\mathbb{R}^1$ are represented by sample points

$$(0), (a_{1,1}), (4), (a_{1,2}), (8).$$

Cells $(a_{1,1})$ and $(a_{1,2})$ have property lists consisting of $PSC(f, y, 0)$, the other three cells have empty property lists. Since there are only two variables, lifting of the cells in $\mathbb{R}^1$ represented by rational numbers will be done entirely using exact rational number computations. Let us describe lifting of the two cells represented by algebraic numbers.

To lift $(a_{1,1})$ we start with computing the root structure of $f(a_{1,1}, y)$ using *IntervalRootStructure*. The leading coefficient of f is constant, hence $f_{\text{red}} = f$. Substituting $I_{1,1}$ for x in the coefficients of f we get interval coefficients

$$((-1., 28.62), (3., 29.63), -3, 1)$$

where the coefficients that are explicit rational numbers are not turned into intervals. *IntervalRoots* applied to this set of interval coefficients gives $z_1 = 0.9999999999999920064$, $r_1 = 4.688 \times 10^{-10}$, and $k_1 = 3$. Validation of a triple root requires showing that $PSC_0(f, \partial f/\partial y, y)$ and $PSC_1(f, \partial f/\partial y, y)$ are zero at $(a_{1,1})$. $PSC(f, y, 0)$ is on the property list of $(a_{1,1})$, but since principal subresultant coefficients of orders greater than zero are not a part of McCallum's projection, we need to revert to algebraic number computations to prove that $PSC_1(f, \partial f/\partial y, y)$ is zero at $(a_{1,1})$. We compute $p_1 := PSC_1(f, \partial f/\partial y, y) = -6x^3 + 18$. Since p_1 is divisible by g_1, $p_1(a_{1,1}) = 0$, which proves that $f(a_{1,1}, y)$ indeed has a triple root. *IntervalRootStructure* returns an interval $I_{2,1} := (1., 29.32)$. Note that $I_{2,1}$ has a center different from z_1 and a radius smaller than r_1, as explained in Remark 5.5. The cells in $\mathbb{R}^2$ extending $(a_{1,1})$ are represented by sample points

$$(a_{1,1}, 0), (a_{1,1}, a_{2,1}), (a_{1,1}, 2).$$

The property lists of all cells contain $PSC(f, y, 0)$, the property list of the second cells additionally contains *Polynomial*(f). The last task to perform is finding the sign of f on each of the cells. Since the property list of the second cell contains *Polynomial*(f), f is zero on this cell.

After substituting $(I_{1,1}, 0)$ and $(I_{1,1}, 2)$ for (x, y) in f we get $(-1., 28.76)$ and $(1., 29.1)$, which proves that the signs of f on the first and third cells are -1 and 1.

To lift $(a_{1,2})$ we start with computing the root structure of $f(a_{1,2}, y)$ using *IntervalRootStructure*. The leading coefficient of f is constant, hence $f_{\text{red}} = f$. Substituting $I_{1,2}$ for x in the coefficients of f we get interval coefficients

$$\begin{gathered}(2448.579153464263990934563124717296187002\,8, 29.51)\\(310.46182625345341818510491931836079832294\,3, 29.62)\\-3\\-1\end{gathered}$$

IntervalRoots applied to this set of interval coefficients gives

$$\begin{aligned}z_1 &= -19.44380350304882049559689938999795238407\\z_2 &= 11.22190175152441024779827128992\end{aligned}$$

$r_1 = 4.732 \times 10^{-30}$, $r_2 = 3.675 \times 10^{-15}$, $k_1 = 1$, and $k_2 = 2$. The property list of $(a_{1,2})$ contains $PSC(f, y, 0)$, hence by Proposition 4.3, $D_{f,2} := |y - z_2| < r_2$ contains a double root of $f(a_{1,2}, y)$. *IntervalRootStructure* returns intervals

$$\begin{aligned}I_{2,2} &= (-19.44380350304882049559689938999795238407, 29.32)\\I_{2,3} &= (11.22190175152441024779844969499897619203, 29.31)\end{aligned}$$

Note that $I_{2,3}$ has a center different from z_2 and a radius smaller than r_2, as explained in Remark 5.5. The cells in $\mathbb{R}^2$ extending $(a_{1,2})$ are represented by sample points

$$(a_{1,2}, -32), (a_{1,2}, a_{2,2}), (a_{1,2}, 0), (a_{1,2}, a_{2,3}), (a_{1,2}, 16)$$

Property lists of all cells contain $PSC(f, y, 0)$, property lists of the second and fourth cells additionally contain $Polynomial(f)$. The last task to perform is finding the sign of f on each of the cells. Since the property lists of the second and fourth cells contain $Polynomial(f)$, f is zero on these cells. After substituting $(I_{1,2}, -32)$, $(I_{1,2}, 0)$, and $(I_{1,2}, 16)$ for (x, y) in f we get respectively

$$\begin{gathered}(-23456.64240642522662714207945709515826666296, 29.87)\\(2448.579153464263990934563124717296187002\,82, 29.51)\\(809.18993340900929997288441562352341383\,57, 29.01)\end{gathered}$$

which proves that the signs of f on the first, third, and fifth cells are respectively -1, 1, and 1.

7. Empirical results

We compare four implementations of CAD. The first three are our implementations. They all use the same algorithm framework and differ only in the way the lifting phase represents sample points and computes root structure and polynomial signs. *ACAD*, for "annotations" added to projection polynomials ("genealogy") and cell coordinates ("property lists"), uses *IntervalRootStructure* and *IntervalPolynomialSign* described in this paper. *ECAD* uses exact algebraic number computations in the lifting phase. *ICAD* uses sample points with interval coordinates, but reverts to exact algebraic number computations each time root structure cannot be obtained and validated by purely interval methods (which includes all cells on which the leading coefficient of a projection polynomial is zero, a projection polynomial has a multiple root or two projection polynomials have a common root). *ACAD*, *ECAD* and *ICAD* are implemented in C, as a part of the kernel of *Mathematica*. The fourth algorithm used in the comparison is *QEPCAD*, version B 1.44 (Collins and Hong, 1991; Brown, 2003).

The experiments have been conducted on a 1.8 GHz Pentium M computer, with 1.7 GB of RAM available. *ACAD*, *ECAD* and *ICAD* were given a time limit of 3600 s. *QEPCAD* was called with $+N10000000$ and $+L10000$ command line options. In all but one example it either completed the computation or exited without completing the computation giving the "Prime list exhausted" message. In one example ($c-2$) the *QEPCAD* computation did not finish in 10000 s and was aborted.

All timings given in the tables are in seconds. The column *Proj* contains the times used by the projection phase of *ACAD* (it is the same for *ECAD* and *ICAD*). The columns *ACAD*, *ECAD*, *ICAD* and *QEPCAD* contain total computation times for the corresponding algorithms. The cases where *QEPCAD* exited without completing the computation are marked with F (time used) in the Time column.

The statistics tables contain computation statistics for *ACAD*, *ECAD* and *ICAD*. The Cells column gives the total number of "fully lifted" cells constructed by *ACAD* during the lifting phase. A cell is "fully lifted" if it is a cell in $\mathbb{R}^n$, where n is the total number of variables in the system, or if it is a lower dimensional cell that gets rejected, because its sample point makes the system false. The Reverts columns give the number of times *ACAD* and *ICAD* needed to revert to exact computations. The Degree columns contain the maximal degrees of minimal polynomials of algebraic numbers that appeared in the computation.

The *Mathematica* and *QEPCAD* inputs for all compared examples are available at http://members.wolfram.com/adams/CADExamples.tar.gz.

7.1. Examples from applications

Example 7.1. Stability of the Dormand–Prince fifth-order embedded seven-stage method (Example 4.4 from Hong et al. (1997)).

Show that

$$\forall x, y \in \mathbb{R} \left(x < 0 \wedge x^2 + y^2 < \frac{99\,438}{100\,000} \right) \Rightarrow R(x + \mathrm{i}y)R(x - \mathrm{i}y) < 1$$

where

$$R(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24} + \frac{z^5}{120} + \frac{z^6}{600}.$$

Example 7.2. Stability of a six-point upwind-biased second-order accurate scheme for approximating a two-dimensional advection equation (Example 5.4 from Hong et al. (1997)).

Let

$$\begin{aligned}
A &= C_2^4(\alpha - \beta + 1)(\alpha - \beta - 1)(\alpha - \beta)^2 \\
B &= 2C_2^4\beta(3\alpha^2\beta - 2\alpha^2 - 2\alpha\beta^2 + \alpha + \beta^3 - \beta) + 4C_2^3\alpha\beta(\alpha^2 - \alpha + \beta^2 - \beta) \\
&\quad + 2C_2^2\alpha(\alpha^3 - 2\alpha^2\beta + 3\alpha\beta^2 - \alpha - 2\beta^2 + \beta) \\
C &= C_2^4\beta^2(\beta^2 - 1) + 4C_2^3\alpha\beta^2(\beta - 1) + 2C_2^2\alpha\beta(3\alpha\beta - 2\alpha - 2\beta + 1) \\
&\quad + 4C_2\alpha^2\beta(\alpha - 1) + \alpha^2(\alpha^2 - 1) \\
D &= C_2^2R + 2C_2S + T
\end{aligned}$$

$$\begin{aligned}
R &= 8\alpha^2\beta^2 - 12\alpha^2\beta + 5\alpha^2 - 8\alpha\beta^3 + 8\alpha\beta^2 + 2\alpha\beta - 4\alpha + 4\beta^4 - 4\beta^3 - 3\beta^2 + 4\beta \\
S &= 4\alpha^3\beta - 2\alpha^3 - 4\alpha^2\beta^2 - 2\alpha^2\beta + \alpha^2 + 4\alpha\beta^3 - 2\alpha\beta^2 + 2\alpha\beta - 2\beta^3 + \beta^2 \\
T &= 4\alpha^4 - 8\alpha^3\beta - 4\alpha^3 + 8\alpha^2\beta^2 + 8\alpha^2\beta - 3\alpha^2 - 12\alpha\beta^2 + 2\alpha\beta + 4\alpha + 5\beta^2 - 4\beta.
\end{aligned}$$

1. Show that

$$\forall \alpha, \beta, C_2 \in \mathbb{R}\, (\alpha \geq 0 \wedge \beta \geq 0 \wedge 4(\alpha^2 + \beta^2) < 1) \Rightarrow (B \leq 0 \vee D \leq 0)$$

(reduced form of (5.14) from Hong et al. (1997)).

2. Show that

$$\forall \alpha, \beta, C_2 \in \mathbb{R}\, (0 \leq \alpha \leq 1 \wedge 0 \leq \beta \leq 1) \Rightarrow A \leq 0 \wedge C \leq 0 \wedge (B \leq 0 \vee D \leq 0)$$

(full form of (5.14) from Hong et al. (1997)).

Example 7.3. Robust multi-objective feedback design (Example 4.2 from Dorato et al. (1997)).
Find the set of $\frac{n}{d}$ satisfying

$$\begin{aligned}
&\exists q_1, q_2 \in \mathbb{R}\, \forall w \in \mathbb{R}\, q_1 > 1 \wedge q_2 > 0 \wedge \frac{n}{d} > 0 \wedge \\
&\left(\frac{n}{d} - q_1^2\right) w^4 + \left(\frac{n}{d}((1+q_1)^2 - 2q_2) - (q_2^2 + q_1^2)\right) w^2 + \left(\frac{n}{d} - 1\right) q_2^2 \geq 0 \wedge \\
&\left(\frac{n}{d} - q_1^2\right) w^4 + \left(\frac{n}{d}((-1+q_1)^2 - 2q_2) - (q_2^2 + q_1^2)\right) w^2 + \left(\frac{n}{d} - 1\right) q_2^2 \geq 0.
\end{aligned}$$

The answer is

$$\frac{n}{d} > 4.$$

This is the full form of Example 4.2 from Dorato et al. (1997), the paper reports being able to eliminate the quantifiers for two particular values of $\frac{n}{d}$.

Example 7.4. Stationary orientations of F16 aircraft (Example 2.3 of Jirstrand (1997)).
Let

$$\begin{aligned}
C_L &= -38x_2 - 170x_1x_2 + 148x_1^2x_2 + 4x_2^2 \\
&\quad + u_1(-52 - 2x_1 + 114x_1^2 - 79x_1^3 + 7x_2^2 + 14x_1x_2^2) \\
&\quad + u_3(14 - 10x_1 + 37x_1^2 - 48x_1^3 + 8x_1^4 - 13x_2^2 - 13x_1x_2^2 + 20x_1^2x_2^2 + 11x_2^4) \\
C_M &= -12 - 125u_2 + u_2^2 + 6u_2^3 + 95x_1 - 21u_2x_1 \\
&\quad + 17u_2^2x_1 - 202x_1^2 + 81u_2x_1^2 + 139x_1^3 \\
C_N &= 139x_2 - 112x_1x_2 - 388x_1^2x_2 + 215x_1^3x_2 - 38x_2^3 + 185x_1x_2^3 \\
&\quad + u_1(-11 + 35x_1 - 22x_1^2 + 5x_2^2 + 10x_1^3 - 17x_1x_2^2) \\
&\quad + u_3(-44 + 3x_1 - 63x_1^2 + 34x_2^2 + 142x_1^3 + 63x_1x_2^2 - 54x_1^4 - 69x_1^2x_2^2 - 26x_2^4).
\end{aligned}$$

Find a cylindrical algebraic decomposition of the set of (x_1, x_2) satisfying

$$\exists u_1, u_2, u_3 \in \mathbb{R}\, C_L = 0 \wedge C_M = 0 \wedge C_N = 0 \wedge u_1^2 \leq 1 \wedge u_2^2 \leq 1 \wedge u_3^2 \leq 1.$$

The answer is a disjunction of 60 cylindrical terms whose description involves algebraic numbers and functions with minimal polynomials of degrees up to 54. (The original problem from Jirstrand (1997) does not require a description of the full solution set.) Neither of the four algorithms can solve this system directly. *Mathematica* solves it by eliminating u_1 and u_3 using

Examples from applications (timings)

Example	Time *Proj*	Time *ACAD*	Time *ECAD*	Time *ICAD*	Time *QEPCAD*
1	0.08	0.12	3.96	3.93	0.83
2 − 1	0.22	1.94	>3600	>3600	389
2 − 2	0.27	2.61	>3600	>3600	1235
3	0.37	19.6	>3600	>3600	*F*(3446)
4	6.43	408	>3600	>3600	*F*(402)

Examples from applications (statistics)

Example	Cells *ACAD*	Reverts *ACAD*	Reverts *ICAD*	Degree *ACAD*	Degree *ECAD*	Degree *ICAD*
1	81	0	4	28	30	30
2 − 1	4 853	0	?	20	?	?
2 − 2	5 053	1	?	16	?	?
3	32 606	58	?	32	?	?
4	373 187	20	?	55	?	?

the Loos–Weispfenning algorithm (Loos and Weispfenning, 1993) and then computing the CAD. For this comparison all algorithms were given the output of the Loos–Weispfenning algorithm.

7.2. Randomly generated systems

CAD timings for randomly generated systems with fixed parameters, that is a fixed number of variables, fixed numbers of equations, strict and weak inequalities, a fixed degree of polynomials, a fixed number of terms in each polynomial and a fixed coefficient size still vary widely. Hence, if we choose parameters so that any system with these parameters can be done in the allowed time, a randomly chosen example with these parameters is likely to take less than a second. For our comparison we wanted to use some more difficult systems, hence we chose a parameter size

Randomly generated examples (timings)

Example	Time *Proj*	Time *ACAD*	Time *ECAD*	Time *ICAD*	Time *QEPCAD*
$a-1$	0.25	1.49	>3600	>3600	*F*(306)
$a-2$	0.28	2.19	>3600	>3600	*F*(270)
$a-3$	0.03	1.07	2123	1624	3.68
$a-4$	0.04	1.23	>3600	>3600	0.34
$a-5$	0.88	3.09	>3600	>3600	*F*(211)
$b-1$	0.05	15.7	>3600	>3600	*F*(1272)
$b-2$	13.4	15.5	>3600	>3600	*F*(265)
$b-3$	0.32	19.1	>3600	>3600	*F*(498)
$b-4$	14.2	24.3	>3600	>3600	*F*(317)
$b-5$	0.11	34.5	>3600	>3600	*F*(3278)
$c-1$	0.64	173	>3600	>3600	*F*(1042)
$c-2$	0.23	104	>3600	>3600	>10000
$c-3$	1.05	196	>3600	>3600	*F*(279)
$c-4$	4.32	854	>3600	>3600	*F*(1766)
$c-5$	65.7	606	>3600	>3600	*F*(624)

Randomly generated examples (statistics)

Example	Cells *ACAD*	Reverts *ACAD*	Reverts *ICAD*	Degree *ACAD*	Degree *ECAD*	Degree *ICAD*
$a-1$	470	3	?	34	?	?
$a-2$	403	1	?	77	?	?
$a-3$	12 55	2	115	4	72	72
$a-4$	210	0	?	12	?	?
$a-5$	790	0	?	35	?	?
$b-1$	8 859	1	?	17	?	?
$b-2$	733	0	?	32	?	?
$b-3$	1 600	6	?	105	?	?
$b-4$	4 392	4	?	88	?	?
$b-5$	60 467	1	?	18	?	?
$c-1$	2 189	1	?	90	?	?
$c-2$	219 087	9	?	12	?	?
$c-3$	13 055	8	?	126	?	?
$c-4$	257 313	12	?	288	?	?
$c-5$	2 011	10	?	122	?	?

range in which we can solve many, but not all systems, and chose the randomly generated systems we could solve. More precisely, we randomly generated systems with three or four variables, at most one equation and one strict inequality and one weak inequality, polynomials of degree $2 \leq d \leq 4$, $5 \leq k \leq 7$ terms in each polynomial and random 10-bit integer coefficients. Our examples consist of the first five examples whose *ACAD* timings fell into each of the following timing ranges: (a) 1–10 s, (b) 10–100 s, (c) 100–1000 s. The *Mathematica* and *QEPCAD* inputs for the examples are available at http://members.wolfram.com/adams/CADExamples.tar.gz.

7.3. Conclusions

ACAD was significantly faster than the other algorithms in all examples, except $a-4$, where it was somewhat slower than *QEPCAD*. For systems that required computations with high degree algebraic numbers *ACAD* was the only algorithm able to complete the task.

ACAD reverts to algebraic number computation in two situations. First, sizes of intervals representing sample point coordinates are too large, and therefore interval approximations of nonzero polynomials or coefficients contain zero, or distinct roots are put in the same cluster by *IntervalRoots*, or bounding discs of distinct roots of two polynomials intersect. This case requires execution of step 5 of *IntervalRootStructure*, which can result in a significant slowdown. Second, the property list L does not contain enough information to validate the correct root structure obtained by using *IntervalRoots*, or to prove that polynomials or coefficients whose interval approximations contain zero are indeed zero. This case requires execution of steps 2(d), 2(g), or 3(d) of *IntervalRootStructure*, but does not require execution of step 5. The first situation happened only once, in example $c-1$. However, after increasing the precision to which initial intervals bound the roots from 100 to 200 binary digits, *ACAD* was able to compute example $c-1$ without having to execute step 5, and the computation time decreased from 173 to 5.6 seconds. The second situation should never happen when Hong's projection (Hong, 1990) is used, because all leading coefficients and all principal subresultant coefficients necessary to determine the number of multiple and common roots are a part of the projection. When using a smaller projection, the second situation can and does happen, however our experiments suggest that reverting to exact computation in steps 2(d), 2(g), or 3(d) of *IntervalRootStructure* does not affect the computation time as much as executing step 5.

Cells on which the leading coefficient of a projection polynomial is zero, a projection polynomial has a multiple root, or two projection polynomials have a common root appear systematically in CAD computation. *ICAD* had to revert to exact computations for each such cell. The observed performance of *ICAD* was not significantly better than the performance of *ECAD*, which suggests that lifting of these cells dominates the computation time of the lifting phase.

For some of the examples *QEPCAD* preformed significantly better than *ECAD*. An implementational difference between *QEPCAD* and *ECAD*, that we are aware of, is in the method of representing algebraic numbers. *QEPCAD* computes primitive elements of extensions generated by sample point coordinates, and then finds polynomial roots over the obtained simple extensions. *ECAD* represents sample point coordinates using minimal polynomials with integer coefficients and isolating intervals, and finds polynomial roots over multiple extensions using the algorithm given in Strzeboński (1997). For completeness, let us give an example where the method used by *ECAD* is faster. Let

$$\begin{aligned} f(x) = {} & 569x^{10} + 614x^9 - 548x^8 + 495x^7 - 544x^6 + 786x^5 + 42x^4 \\ & + 58x^3 - 615x^2 - 947x + 88. \end{aligned}$$

Find the CAD of

$$f(x_1) \geq 0 \wedge f(x_2) \geq 0 \wedge x_1^2 + x_3 \leq 3 \wedge x_2^3 + x_3 \leq 3$$

in the variable order (x_1, x_2, x_3). *ECAD* completes the computation in 0.22 s and *QEPCAD* fails after 2260 s.

Acknowledgements

The author would like to thank the referees for helpful comments.

References

Anai, H., Yokoyama, K., 2005. CAD via numerical computation with validated symbolic reconstruction. In: Dolzmann A., Seidl A., Sturm T. (Eds.), Proceedings of A3L 2005, A3L.

Brown, C.W., 2001. Improved projection for cylindrical algebraic decomposition. J. Symbolic Comput. 32, 447–465.

Brown, C.W., 2003. An Overview of QEPCAD B: A tool for real quantifier elimination and formula simplification. J. JSSAC 10, 13–22.

Caviness, B., Johnson, J., 1998. Quantifier Elimination and Cylindrical Algebraic Decomposition. Springer-Verlag, Wien, New York.

Collins, G.E., 1975. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. Lect. Notes Comput. Sci. 33, 134–183.

Collins, G.E., 1998. Quantifier elimination by cylindrical algebraic decomposition — twenty years of progress. In: Caviness, B., Johnson, J. (Eds.), Quantifier Elimination and Cylindrical Algebraic Decomposition. Springer-Verlag, pp. 8–23.

Collins, G.E., Hong, H., 1991. Partial cylindrical algebraic decomposition for quantifier elimination. J. Symbolic Comput. 12, 299–328.

Collins, G.E., Johnson, J.R., Krandick, W., 2002. Interval arithmetic in cylindrical algebraic decomposition. J. Symbolic Comput. 34, 145–157.

Dorato, P., Yang, W., Abdallah, C., 1997. Robust multi-objective feedback design by quantifier elimination. J. Symbolic Comput. 24, 153–160.

Hong, H., 1990. An improvement of the projection operator in cylindrical algebraic decomposition. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation. pp. 261–264.

Hong, H., 1993. Efficient method for analyzing topology of plane real algebraic curves. In: Proceedings of IMACS-SC 93, Lille, France.

Hong, H., Liska, R., Steinberg, S., 1997. Testing stability by quantifier elimination. J. Symbolic Comput. 24, 161–188.

Jenkins, M.A.G., 1969. Three-stage variable-shift iterations for the solution of polynomial equations with a posteriori error bounds for the zeros. Ph.D. Dissertation. Stanford University.

Jirstrand, M., 1997. Nonlinear control system design by quantifier elimination. J. Symbolic Comput. 24, 137–152.

Keiper, J.B., Withoff, D., 1992. Numerical Computation in Mathematica, Course Notes. In: Mathematica Conference.

Łojasiewicz, S., 1964. Ensembles semi-analytiques. I.H.E.S., Bures sur Yvette (preprint).

Loos, R., Weispfenning, V., 1993. Applying linear quantifier elimination. The Comput. J. 36, 450–461.

McCallum, S., 1988. An improved projection for cylindrical algebraic decomposition of three dimensional space. J. Symbolic Comput. 5, 141–161.

McCallum, S., 1998. An improved projection for cylindrical algebraic decomposition. In: Caviness, B., Johnson, J. (Eds.), Quantifier Elimination and Cylindrical Algebraic Decomposition. Springer-Verlag, pp. 242–268.

McCallum, S., 1999. On projection in CAD-Based quantifier elimination with equational constraint. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation. ACM Press, pp. 145–149.

McCallum, S., 2001. On propagation of equational constraints in CAD-Based quantifier elimination. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation. ACM Press, pp. 223–230.

Strzeboński, A., 1994. An algorithm for systems of strong polynomial inequalities. The Math. J. 4 (4), 74–77.

Strzeboński, A., 1996. Algebraic numbers in mathematica 3.0. The Math. J. 6 (4), 74–80.

Strzeboński, A., 1997. Computing in the field of complex algebraic numbers. J. Symbolic Comput. 24, 647–656.

Strzeboński, A., 1999. A real polynomial decision algorithm using arbitrary-precision floating point arithmetic. Reliab. Comput. 5 (3), 337–346.

Strzeboński, A., 2000a. Solving systems of strict polynomial inequalities. J. Symbolic Comput. 29, 471–480.

Strzeboński, A., 2000b. Solving algebraic inequalities. The Math. J. 7 (4), 525–541.

Tarski, A., 1951. A Decision Method for Elementary Algebra and Geometry. University of California Press, Berkeley.

Wolfram, S., 2003. The Mathematica Book, 5th ed. Wolfram Media.