J. Symbolic Computation (2000) **29**, 471–480
doi: 10.1006/jsco.1999.0327
Available online at http://www.idealibrary.com on IDEAL

Solving Systems of Strict Polynomial Inequalities

ADAM STRZEBOŃSKI

Wolfram Research Inc. and Jagiellonian University, 100 Trade Centre Drive, Champaign, IL 61820, U.S.A.

We present an algorithm for finding an explicit description of solution sets of systems of strict polynomial inequalities, correct up to lower dimensional algebraic sets. Such a description is sufficient for many practical purposes, such as volume integration, graphical representation of solution sets, or global optimization over open sets given by polynomial inequality constraints. Our algorithm is based on the cylindrical algebraic decomposition algorithm. It uses a simplified projection operator, and constructs only rational sample points.

1. Introduction

A *system of polynomial equations and inequalities* is a formula

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

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

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

Every semialgebraic set can be represented as a finite union of disjoint *cells* (see 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

$$r_1(a_1,\ldots,a_k)<r_2(a_1,\ldots,a_k)$$

for all $(a_1,\ldots,a_k)\in C_k$. By an algebraic function we mean a function $r:C_k\to\mathbb{R}$ for which there is a polynomial

$$f=c_0x_{k+1}^m+c_1x_{k+1}^{m-1}+\cdots+c_m\in\mathbb{R}[x_1,\cdots,x_k,x_{k+1}]$$

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

0747–7171/00/030471 + 10 $35.00/0

The cylindrical algebraic decomposition (CAD) algorithm (see Collins, 1975; Caviness and Johnson, 1998) allows us to compute decomposition of semialgebraic sets into finite unions of cells. In this paper we present a faster algorithm which allows us to compute a somewhat weaker result for open solution sets of systems of strict polynomial inequalities (i.e. systems of inequalities (1) with $\rho_{i,j}$ being one of $<$, $>$, or $\neq$).

Let S be a system of strict polynomial inequalities. Without loss of generality we can write S in the form

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

Our algorithm gives subsets A and B of $\mathbb{R}^n$, such that A is an open set represented as a finite union of open cells, B is an at most $(n-1)$-dimensional algebraic set represented by a list of polynomials whose product is zero on B, and the solution set $X(S)$ of S satisfies

$$A \setminus B \subseteq X(S) \subseteq A \cup B.$$

Note, that for many practical applications finding A instead of $X(S)$ may be enough. For instance, since A and $X(S)$ differ by a set of measure zero,

$$\int_{X(S)} f dm = \int_A f dm$$

where dm denotes the n-dimensional Lebesgue measure, and f is a Lebesgue integrable function. Therefore one can use our algorithm to compute integrals over sets described by means of inequalities. The form in which open cells are represented (described in more detail in the following section) is convenient for multiple integration: if an open cell is given by

$$C = r_1 < x_1 < s_1 \wedge r_2(x_1) < x_2 < s_2(x_1) \wedge \cdots$$
$$\wedge r_n(x_1, \ldots, x_{n-1}) < x_n < s_n(x_1, \ldots, x_{n-1}),$$

then

$$\int_A f dm = \int_{r_n(x_1,\ldots,x_{n-1})}^{s_n(x_1,\ldots,x_{n-1})} \cdots \int_{r_2(x_1)}^{s_2(x_1)} \int_{r_1}^{s_1} f dx_1 dx_2 \ldots dx_n.$$

Similarly, open cells produced by our algorithm can be used for graphical visualization of sets described using strict inequalities.

We propose a representation of open cells using algebraic functions. The representation of algebraic functions and open cells used in the output of our algorithm is described in the following section. We have implemented algebraic functions and the main algorithm as a part of *Mathematica* computer algebra system. In the last section we show examples of applications of our implementation in computation of multiple integrals and in graphical visualization of semialgebraic sets.

The third section gives a description of the main algorithm. The algorithm is based on the CAD algorithm, however it uses a simpler projection operator, constructs only open cells, which allows it to use sample points with rational coordinates, and returns the answer written in terms of algebraic functions. Simplified algorithms for deciding existence of solutions of systems of strict polynomial inequalities using rational sample points have been described in McCallum (1993) and Strzeboński (1994). The second also uses a simplified projection operator. In the last section we give an example which compares our improved projection operator with McCallum's projection operator (see McCallum, 1998).

Finally, the last section also contains an example which compares timings of our algorithm with the full CAD algorithm. (By the full CAD algorithm we mean here an algorithm which uses McCallum's projection operator and gives a description of the full solution set in terms of algebraic functions.)

2. Representation of Algebraic Functions and Open Cells

Definition 2.1. A real algebraic function given by a polynomial $f \in \mathbb{R}[x_1, \ldots, x_n][y]$ and an integer p is the function

$$\mathrm{Root}_{y,p} f : \mathbb{R}^n \ni x_1, \ldots, x_n \longrightarrow \mathrm{Root}_{y,p} f(x_1, \ldots, x_n) \in \mathbb{R}$$

where $\mathrm{Root}_{y,p} f(x_1, \ldots, x_n)$ is the pth real root of $f(x_1, \ldots, x_n) \in \mathbb{R}[y]$. The function is defined for those values of $x_1, \ldots, x_n$ for which $f(x_1, \ldots, x_n)$ has at least p real roots. The real roots are ordered by the increasing value, counting multiplicities.

Remark 2.2. All the algebraic functions used in this paper are defined by single roots, so in the above definition we could as well not require counting root mulitiplicities. In a more general context of implementing algebraic functions in a computer algebra system it seems more convenient to count the roots with multiplicities because algebraic functions defined this way are more regular. For instance

$$\mathrm{Root}_{y,2}(y - x_1)(y - x_2)$$

is defined for all x_1 and x_2, and equal to $\max(x_1, x_2)$ for real x_1 and x_2. Without taking multiplicities into account the function would not be defined for $x_1 = x_2$. Also, a monic polynomial of degree n has n algebraic function roots, which are defined for all values of parameters (and for instance can be used for plotting the Riemann surface of the polynomial).

Let us now describe recursively our representation of open cells.

(1) An open cell in $\mathbb{R}$ is represented as

$$r < x_1 < s$$

where r and s are $-\infty$, ∞, or algebraic numbers represented by $\mathrm{Root}_{y,p} f$ and $\mathrm{Root}_{y,q} g$ for univariate polynomials $f(y)$ and $g(y)$.

(2) An open cell in $\mathbb{R}^{k+1}$ is represented as

$$C_k \wedge r(x_1, \ldots, x_k) < x_{k+1} < s(x_1, \ldots, x_k)$$

where C_k is a representation of an open cell in $\mathbb{R}^k$, and r and s are $-\infty$, ∞, or algebraic functions, defined and continuous on C_k, represented by $\mathrm{Root}_{y,p} f$ and $\mathrm{Root}_{y,q} g$ for $(k+1)$-variate polynomials $f(x_1, \ldots, x_k, y)$ and $g(x_1, \ldots, x_k, y)$, and

$$r(x_1, \ldots, x_k) < s(x_1, \ldots, x_k)$$

for all $(x_1, \ldots, x_k) \in C_k$.

3. The Algorithm

Algorithm 3.1. (GCAD ("Generic Cylindrical Algebraic Decomposition"))

Input: A system

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

of strict polynomial inequalities.
Output: An open set $A \subseteq \mathbb{R}^n$ represented by a finite disjunction of representations of open cells described above, and an at most $(n-1)$-dimensional algebraic set $B \subseteq \mathbb{R}^n$ represented by a list of polynomials whose product is zero on B. The solution set $X(S)$ of S satisfies

$$A \setminus B \subseteq X(S) \subseteq A \cup B.$$

(1) Call the subalgorithm GPROJ described below, with S as the input. Get $(F_1, \ldots, F_n)$, $(pr_1, \ldots, pr_n)$.
(2) Compute the required representations of sets A and B by calling the recursive subalgorithm RSFC described below, with S, $(F_1, \ldots, F_n)$, $(pr_1, \ldots, pr_n)$, $k = 0$, and $\Phi = true$ as the input.

Subalgorithm GPROJ corresponds to the projection phase of the CAD algorithm. We use a projection operator which is simpler than the projection operators used in CAD. The projection operator was used (but not explicitely defined) in Strzeboński (1994). Subalgorithm RSFC is a recursive algorithm which combines the sample point construction phase of the CAD algorithm with construction of the solution formula. In the following we use the standard CAD terminology (see Collins, 1975; Caviness and Johnson, 1998).

Let us define the projection operator.

Definition 3.2. For a set of polynomials F, let $SFRP(F)$ denote a set of square-free and relatively prime polynomials multiplicatively generating F. For a set of square-free and relatively prime polynomials G, let $PR(G, v)$ denote the set of the leading coefficients, discriminants, and pairwise resultants of elements of G as polynomials in v.

Remark 3.3. $SFRP(F)$ is not uniquely determined. For the following it does not matter which set of square-free and relatively prime polynomials multiplicatively generating F we use, so we just assume that we have a procedure for computing one. In our implementation for polynomials with rational number coefficients we use the set of irreducible factors of F, and our experience is that polynomial factorization is not a significant part of the execution time of the whole algorithm.

Algorithm 3.4. (GPROJ ("Generic Projection"))

Input: A system

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

of strict polynomial inequalities.

Output: Lists $(F_1, \ldots, F_n)$, $(pr_1, \ldots, pr_n)$. For $1 \leq k \leq n$, F_k and pr_k are lists of polynomials in $x_1, \ldots, x_k$.

(1) Set $F_n = \{f_{i,j} : 1 \leq i \leq l, 1 \leq j \leq m\}$, $pr_n = SFRP(F_n)$.
(2) Compute $F_{k-1} = PR(pr_k, x_k)$, $pr_{k-1} = SFRP(F_{k-1})$, for $2 \leq k \leq n$.

Algorithm 3.5. (RSFC ("Recursive Solution Formula Construction"))

Input:

A system S of strict polynomial inequalities.
Lists $(F_1, \ldots, F_n)$, $(pr_1, \ldots, pr_n)$ computed from S using GPROJ.
$0 \leq k \leq n$.
A formula Φ. If $k > 0$, Φ is a description of an open cell $C \subseteq \mathbb{R}^k$, such that all polynomials of pr_k have constant non-zero signs on C.
If $k > 0$, a sample point $(a_1, \ldots, a_k) \in C$ with all coordinates rational.

Output:

A formula Ψ_Φ which is a finite disjunction of representations of open cells forming an open set $A_\Phi \subseteq \mathbb{R}^n$. If $k > 0$ the projection of each cell on $\mathbb{R}^k$ is equal to C.
A list L_Φ of polynomials. If $B_\Phi \subseteq \mathbb{R}^n$ is the set of zeros of the product of elements of L_Φ, the solution set $X(\Phi \wedge S)$ of $\Phi \wedge S$ satisfies

$$A_\Phi \setminus B_\Phi \subseteq X(\Phi \wedge S) \subseteq A_\Phi \cup B_\Phi. \tag{2}$$

(1) Let S' be S with $(x_1, \ldots, x_k)$ replaced by $(a_1, \ldots, a_k)$. If S' is $true$ return $\Psi_\Phi = \Phi$ and $L_\Phi = \{\}$. If S' is $false$ return $\Psi_\Phi = false$ and $L_\Phi = \{\}$. Note that this may happen even if $k < n$ because some of the polynomials $f_{i,j}$ may depend only on $(x_1, \ldots, x_k)$. S' must be $true$ or $false$ if $k = n$, so in the following steps $k < n$.
(2) Let $pr_{k+1} = (g_1, \ldots, g_s)$. If $k > 0$ let $(g'_1, \ldots, g'_s)$ be the univariate polynomials in x_{k+1} obtained from $(g_1, \ldots, g_s)$ by replacing $(x_1, \ldots, x_k)$ with $(a_1, \ldots, a_k)$. If $k = 0$ we set $(g'_1, \ldots, g'_s) = (g_1, \ldots, g_s)$.
(3) Let $r_1 < r_2 < \cdots < r_t$ denote all real roots of $(g'_1, \ldots, g'_s)$. Isolating real roots of $(g'_1, \ldots, g'_s)$ we find rational numbers $p_0, \ldots, p_t$ such that

$$p_0 < r_1 < p_1 < r_2 < \cdots < r_t < p_t < r_{t+1}$$

and algebraic functions $h_1, \ldots, h_t$ such that, for $1 \leq i \leq t$, if r_i is the pth root of g'_j then

$$h_i = \mathrm{Root}_{x_{k+1},p} g_j.$$

Set $h_0 = -\infty$ and $h_{t+1} = \infty$.
(4) For $0 \leq i \leq t$, call RSFC recursively with $k_i = k + 1$,

$$\Phi_i = \Phi \wedge h_i(x_1, \ldots, x_k) < x_{k+1} < h_{i+1}(x_1, \ldots, x_k)$$

and a sample point $(a_1, \ldots, a_k, p_i)$, obtaining formulas $\Psi_0, \ldots, \Psi_t$ and lists of polynomials $L_0, \ldots, L_t$.
(5) Let L be a list of those elements g_j of pr_{k+1}, for which there exist $1 \leq i \leq t$, such that $h_i = \mathrm{Root}_{x_{k+1},p} g_j$ and both Ψ_{i-1} and Ψ_i are not $false$.

(6) Return $\Psi_\Phi = \Psi_0 \vee \cdots \vee \Psi_t$ and $L_\Phi = L \cup L_0 \cup \cdots \cup L_t$.

To show correctness of GCAD it suffices to show the correctness of RSFC.

It easily follows from the definition of PR and SFRP that all elements of F_n which depend only on the first k variables are multiplicatively generated by pr_k. This proves the correctness of step 1, since we assume that the elements of pr_k have constant signs on C.

To complete the proof of correctness of RSFC we will use the following lemma.

LEMMA 3.6. *Let $f \in \mathbb{R}[x_1, \ldots, x_n, y]$, and let $X \subseteq \mathbb{R}^n$ be a connected open set on which the leading coefficient and the discriminant of f, as a polynomial in y, have constant non-zero signs. Then there is a constant m, such that f has m real roots on X, and $\mathrm{Root}_{y,k} f$ is a continuous function on X for all $1 \leq k \leq m$.*

PROOF. Let A_k be the set of elements of X for which f has at least k non-real roots, and let B_l be the set of elements of X for which f has at least l real roots. A_k is open by the implicit function theorem applied to f treated as a function $\mathbb{C}^n \times \mathbb{C} \to \mathbb{C}$. (For any $a = (a_1, \ldots, a_n) \in A_k$ let $b_1, \ldots, b_k$ be k different non-real roots of $f(a_1, \ldots, a_n, y)$, and let $U_1, \ldots, U_k$ be disjoint neighborhoods of $b_1, \ldots, b_k$ in $\mathbb{C} \setminus \mathbb{R}$. There is a neighborhood U of a in $\mathbb{C}^n$ such that $f(c_1, \ldots, c_n, y)$ has a root in each of $U_1, \ldots, U_k$ for any $(c_1, \ldots, c_n) \in U$. Then $U \bigcap \mathbb{R}^n$ is an open neighborhood of a contained in A_k.) B_l is open by the implicit function theorem applied to f treated as a function $\mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$. Let C_k be the set of elements of X for which f has exactly k real roots. Then

$$C_k = A_{d-k} \cap B_k$$

where d is the degree of f in y. Sets C_k for $0 \leq k \leq d$ are disjoint, open, and they cover X, so there is $0 \leq m \leq d$ such that $C_m = X$. Then functions $\mathrm{Root}_{y,k} f$, for $1 \leq k \leq m$, are defined on X and they are continuous by the implicit function theorem.□

The leading coefficients, discriminants, and the pairwise resultants of elements of pr_{k+1} (as polynomials in x_{k+1}) are elements of F_k, so they are multiplicatively generated by elements of pr_k and hence have constant non-zero signs on C. By Lemma 3.6, the elements of pr_{k+1} have a constant number of real roots each, and the real roots are continuous functions. This means that the algebraic functions $h_1, \ldots, h_t$ are defined and continuous on C, so $\Phi_0, \ldots, \Phi_t$ are valid representations of open cells $C_0, \ldots, C_t$ in $\mathbb{R}^{k+1}$. The graphs of the roots do not intersect, because the pairwise resultants of elements of pr_{k+1} have constant non-zero signs. Therefore the elements of pr_{k+1} have constant non-zero signs on each of $C_0, \ldots, C_t$.

The correctness of (2) follows from the asserted correctness of recursive calls, and the facts that

$$(C \times \mathbb{R}^{n-k}) \setminus (C_0 \cup \cdots \cup C_t) = \mathrm{graph}(h_1) \cup \cdots \cup \mathrm{graph}(h_t)$$

and that, since the set $X(\Phi \wedge S)$ is open, $graph(h_i)$ may intersect $X(\Phi \wedge S)$ only if both $C_{i-1} \times \mathbb{R}^{n-k-1} \cap X(S) = X(\Psi_{i-1} \wedge S)$ and $C_i \times \mathbb{R}^{n-k-1} \cap X(S) = X(\Psi_i \wedge S)$ are not empty.

REMARK 3.7. We can reduce the size of the description of the set A returned by GCAD by making RSFC join some of the adjacent cells. If subsequent formulas $\Psi_i, \ldots, \Psi_j$

obtained in step 4 all have a form

$$\Phi \wedge h_i(x_1,\ldots,x_k) < x_{k+1} < h_{i+1}(x_1,\ldots,x_k) \wedge \Xi$$
$$\Phi \wedge h_{i+1}(x_1,\ldots,x_k) < x_{k+1} < h_{i+2}(x_1,\ldots,x_k) \wedge \Xi$$
$$\ldots$$
$$\Phi \wedge h_j(x_1,\ldots,x_k) < x_{k+1} < h_{j+1}(x_1,\ldots,x_k) \wedge \Xi$$

(with the same Ξ) RSFC can replace them with a single formula

$$\Phi \wedge h_i(x_1,\ldots,x_k) < x_{k+1} < h_{j+1}(x_1,\ldots,x_k) \wedge \Xi.$$

Our experiments suggest that this situation happens very often. Note that while the sets A and B, returned by GCAD without the reduction described above, do not intersect, adding this reduction can make $A \cap B$ non-empty. This is why we use $A \setminus B$ in the specification of the output of GCAD. Another way to make the output shorter is to factor out Φ from the formula Ψ_Φ. Each Ψ_i has a form $\Psi_i = \Phi \wedge \Upsilon_i$, so we can write $\Psi_\Phi = \Phi \wedge (\Upsilon_0 \vee \cdots \vee \Upsilon_t)$.

We use a recursive algorithm which finds sample points and constructs the solution formula at the same time, because it allows us to save memory by not storing the sample points. We can do this because we use algebraic functions defined by the projection polynomials to describe cells. The size of solution formula produced by GCAD (using Remark 3.7) is often much smaller compared to the number of sample points that need to be constructed to find it.

Example 3.8. GCAD (with Remark 3.7 implemented) applied to the system of inequalities

$$x^4 + y^2 + z^2 + t^2 < 1 \wedge x^2 + y^4 + z^2 + t^2 < 1$$
$$\wedge x^2 + y^2 + z^4 + t^2 < 1 \wedge x^2 + y^2 + z^2 + t^4 < 1 \wedge x^2 + y^2 + z^2 + t^2 < 1$$

gives a one cell description of set A

$$-1 < x < 1 \wedge -\sqrt{1-x^2} < y < \sqrt{1-x^2} \wedge -\sqrt{1-x^2-y^2} < z < \sqrt{1-x^2-y^2}$$
$$\wedge -\sqrt{1-x^2-y^2-z^2} < t < \sqrt{1-x^2-y^2-z^2}$$

and 244 equations describing set B. The computation constructs a total of 2 401 264 sample points, so storing all sample points would require storing 9 605 056 rational number coordinates. However, the maximal number of sample point coordinates that the recursive algorithm needs to store at the same time is only 931.

4. Examples and Experimental Results

Example 4.1. Our implementation of GCAD in the C kernel of *Mathematica* has been used in Roger Germundsson's InequalityGraphics and MultipleIntegration packages. Figure 1 gives a graphical representation of the solution set of the inequality system

$$x^2 + y^2 + z^2 < 9 \wedge y^2 < x^2 + z^2 - 1$$

produced with InequalityGraphics.

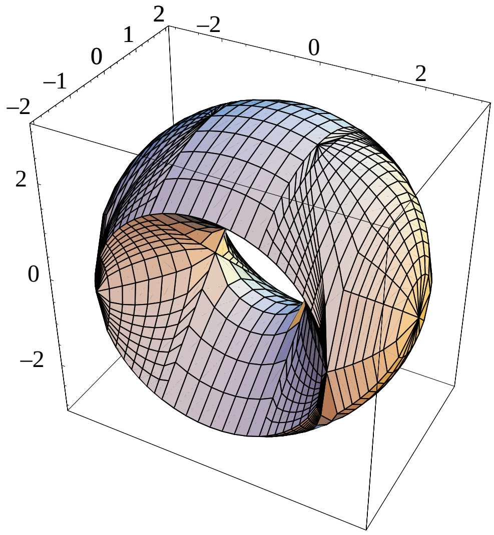

Figure 1. The solution set of $x^2 + y^2 + z^2 < 9 \wedge y^2 < x^2 + z^2 - 1$.

Example 4.2. Using the GCAD algorithm we can compute the volume of the set pictured in Figure 1. Symbolic integration using the MultipleIntegration package gives $64\pi/3$. Here the integration is by far more time consuming than the GCAD computation. Symbolic integration may not be able to handle the general form of algebraic functions, but we can also use the answer from GCAD to compute integrals numerically. Numerical approximations of values of arbitrary real algebraic functions can be computed using polynomial root finding algorithms. In fact, *Mathematica* has built-in computation with algebraic functions and algebraic numbers (see Strzeboński, 1996, 1997). We have computed the volume of the set pictured in Figure 1 numerically and obtained 67.0206 (which agrees with the result of the exact computation).

Example 4.3. This example compares our improved projection operator $SFRP \circ PR$ (see Definition 3.2) used in the GCAD algorithm with McCallum's projection operator (see McCallum, 1998). MGCAD denotes an algorithm which is identical to GCAD, except that it uses McCallum's projection operator. We have run both algorithms on the following inequality

$$ax^3 + (a + b + c)x^2 + (a^2 + b^2 + c^2)x + a^3 + b^3 + c^3 > 1$$

where the variables are ordered (a, b, c, x) (the last variable is projected first). Table 1 gives the number p_r of polynomials in the r-variate projection, the maximal total degree d_r of polynomials in the r-variate projection, for $r = 4, 3, 2, 1$, the number c_{all} of all four-dimensional cells the algorithms needed to construct, the number c_{out} of cells in the output, after combining cells using Remark 3.7, the number p_B of polynomials describing the set B (see Algorithm 3.1), and the total time t in seconds used by each algorithm. The example was run on a Pentium II, 330 MHz computer with 128 MB of RAM. In this example using our improved projection operator reduced the number of polynomials in the last projection by a factor of 8.9, the number of cells to construct by a factor of 120, and the total time by a factor of 15.

Table 1. Comparison of projections.

Algorithm	p_4	d_4	p_3	d_3	p_2	d_2	p_1	d_1	c_{all}	c_{out}	p_B	t
MGCAD	1	4	5	8	19	16	142	96	6967	29	82	540
GCAD	1	4	2	8	4	16	16	72	53	29	10	36

Table 2. Systems of strict inequalities.

Inequalities	CAD	$\#c$	GCAD	$\#c$	#p
$B1 \wedge B2$	3.92	1	0.17	1	0
$B1 \wedge B4$	0.67	0	0.08	0	0
$B1 \wedge B2 \wedge B3$	73.05	2	1.32	2	3
$B1 \wedge B2 \wedge B4$	30.45	0	0.59	0	0
$B1 \wedge C1$	17.62	1	0.65	1	1
$B1 \wedge C2$	31.62	0	0.92	0	0
$T \wedge C1$	774.8	17	5.55	9	8
$T \wedge B2$	3000+	?	1.1	1	2
$HB1 \wedge HB2 \wedge HB3$	415	0	3.76	0	0
$HT \wedge HB2 \wedge HB3$	3000+	?	6.99	0	0
$T \wedge C1 \wedge B2$	3000+	?	30.79	28	31
$HT \wedge C1 \wedge HB2$	3000+	?	19.74	0	0

Example 4.4. We have implemented a variant of the cylindrical algebraic decomposition algorithm which gives exact solution sets of arbitrary systems of polynomial equations and inequalities in terms of, not necessarily open, cells represented using our representation of real algebraic functions. Let us call this algorithm CAD. It uses the improved projection operator from McCallum (1988) and McCallum (1998), with further improvements for equations based on Collins (1998). In case of not well-oriented systems it uses the projection operator from Hong (1990). The sample point construction phase uses ideas from Collins and Hong (1991). In the following we compare timings of CAD and GCAD on systems of strict polynomial inequalities. The examples come from McCallum (1993). Both algorithms were implemented in the C kernel of *Mathematica*. The examples were run on a Pentium Pro, 233 MHz computer with 64 MB of RAM. The timings in seconds are given in the CAD and GCAD columns. The $\#c$ columns give the total number of cells produced by each algorithm (answer $false$ counts as no cells). The $\#p$ column gives the number of polynomials describing the error set B given by the GCAD algorithm. The variables are ordered (x, y, z). Following the notation of McCallum (1993) let us put

$$B1 = x^2 + y^2 + z^2 < 1$$
$$B2 = (x-1)^2 + (y-1)^2 + (z-1)^2 < 1$$
$$B3 = (x-1)^2 + (y-1)^2 + \left(z + \frac{1}{2}\right)^2 < 1$$
$$B4 = \left(x - \frac{3}{2}\right)^2 + (y-2)^2 + z^2 < 1$$

$$
\begin{aligned}
C1 &= x^2 + y^2 + z^2 + 2yz - 4y - 4z + 3 < 0 \\
&\quad \wedge y - 1 < z \wedge z < y + 1 \\
C2 &= x^2 + y^2 + z^2 + 2yz - 4y - 4z + 3 < 0 \\
&\quad \wedge y + 1 < z \wedge z < y + 2 \\
T &= z^4 + (2y^2 + 2x^2 + 6)z^2 + y^4 + 2x^2y^2 \\
&\quad -10y^2 + x^4 - 10x^2 + 9 < 0 \\
HB1 &= B1 \wedge x + y + z < 0 \\
HB2 &= B2 \wedge x + y + z > 3 \\
HB3 &= B3 \wedge x + y + z < \frac{3}{2} \\
HT &= T \wedge x + y < 0.
\end{aligned}
$$

We can see that using $GCAD$ instead of CAD gives large speed-ups, and the relative speed-ups are larger for more complicated problems. This is because $GCAD$, which uses rational sample points only, avoids the complexity growth coming from the need for doing computations with algebraic numbers which are roots of polynomials of increasingly high degrees.

References

Caviness, B., Johnson, J., eds (1998). *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Wien, Springer-Verlag.

Collins, G. E. (1975). Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In LNCS **33**, pp. 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*, pp. 8–23. Springer-Verlag.

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

Hong, H. (1990). An improvement of the projection operator in cylindrical algebraic decomposition. In *Proceedings of ISSAC*, pp. 261–264. ACM Press.

Łojasiewicz, S. Ensembles semi-analytiques, *I.H.E.S. in Lojasiewicz, 1964*.

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

McCallum, S. (1993). Solving polynomial strict inequalities using cylindrical algebraic decomposition. *Comput. J.*, **36**, 432–438.

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

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

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

Strzeboński, A. (1997). Computing in the field of complex algebraic numbers. *J. Symb. Comput.*, **24**, 647–656.

Originally Received 15 July 1999
Accepted 29 November 1999