Solving Systems of Strict Polynomial Inequalities

ADAM STRZEBIŃSKI

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
\[
\bigvee_{1 \leq i \leq l} \bigwedge_{1 \leq j \leq m} f_{i,j}(x_1, \ldots, x_n) \rho_{i,j} 0
\]
where \( f_{i,j} \in \mathbb{R}[x_1, \ldots, x_n] \), and each \( \rho_{i,j} \) is one of \(<, \leq, \geq, >, =, \text{ or } \neq \). 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 Lojasiewicz, 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

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

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 \( \infty \), 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

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

such that

\[
c_0(a_1, \ldots, a_k) \neq 0 \land f(a_1, \ldots, a_k, r(a_1, \ldots, a_k)) = 0
\]

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

$$ C = r_1 < x_1 < s_1 \land r_2(x_1) < x_2 < s_2(x_1) \land \cdots $$

$$ \land r_n(x_1, \ldots, x_{n-1}) < x_n < s_n(x_1, \ldots, x_{n-1}), $$

then

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

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

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

where $\text{Root}_{y,p} f(x_1, \ldots, x_n)$ is the $p$th 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 multiplicities. 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

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

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 $-\infty$, $\infty$, or algebraic numbers represented by $\text{Root}_{y,p} f$ and $\text{Root}_{y,q} g$ for univariate polynomials $f(y)$ and $g(y)$.

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

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

where $C_k$ is a representation of an open cell in $\mathbb{R}^k$, and $r$ and $s$ are $-\infty$, $\infty$, or algebraic functions, defined and continuous on $C_k$, represented by $\text{Root}_{y,p} f$ and $\text{Root}_{y,q} g$ 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

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

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 \( \Phi = \text{true} \) 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 explicitly 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

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

of strict polynomial inequalities.
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 $\Phi$. If $k > 0$, $\Phi$ 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 $\Psi_{k+1}$ which is a finite disjunction of representations of open cells forming an open set $A_{k+1} \subseteq \mathbb{R}^k$. If $k > 0$ the projection of each cell on $\mathbb{R}^k$ is equal to $C$.

A list $L_{k+1}$ of polynomials. If $B_{k+1} \subseteq \mathbb{R}^n$ is the set of zeros of the product of elements of $L_{k+1}$, the solution set $X(\Phi \land S)$ of $\Phi \land S$ satisfies

$$A_{k+1} \setminus B_{k+1} \subseteq X(\Phi \land S) \subseteq A_{k+1} \cup B_{k+1}.$$  

1. Set $F_n = \{ f_{i,j} : 1 \leq i \leq l, 1 \leq j \leq m \}$, $pr_n = SFRP(F_n)$.

2. Compute $F_{k-1} = \overline{PR}(pr_k, x_k)$, $pr_{k-1} = SFRP(F_{k-1})$, for $2 \leq k \leq n$.

3. Let $SFRP(L_{k+1}, x_k) = \{ (g_1, \ldots, g_s) \}$.

4. Let $g_{k+1}$ 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 $(g'_1, \ldots, g'_s) = (g_1, \ldots, g_s)$.

5. 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 $p$th root of $g'_j$ then

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

Set $h_0 = -\infty$ and $h_{t+1} = \infty$.

6. For $0 \leq i \leq t$, call RSFC recursively with $k_i = k + 1$,

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

and a sample point $(a_1, \ldots, a_k, p_i)$, obtaining formulas $\Psi_0, \ldots, \Psi_t$ and lists of polynomials $L_0, \ldots, L_t$.

7. 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 $h_i = \text{Root}_{x_{k+1}, p_j}$ and both $\Psi_{i-1}$ and $\Psi_i$ are not false.
(6) Return $\Psi_\Phi = \Psi_0 \lor \cdots \lor \Psi_t$ 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 $\text{Root}_{y,k} f$ 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 $B_k$ be the set of elements of $X$ for which $f$ has at least $k$ 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 $U \cap \mathbb{R}^n$ is an open neighborhood of $a$ contained in $A_k$.) $B_k$ 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 $\text{Root}_{y,k} f$, 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

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

and that, since the set $X(\Phi \land S)$ is open, $\text{graph}(h_i)$ may intersect $X(\Phi \land S)$ only if both $C_{i-1} \times \mathbb{R}^{n-k-1} \cap X(S) = X(\Phi_{i-1} \land S)$ and $C_i \times \mathbb{R}^{n-k-1} \cap X(S) = X(\Phi_i \land S)$ 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 $\Psi_1, \ldots, \Psi_j$
obtained in step 4 all have a form
\[ \Phi \land h_i(x_1, \ldots, x_k) < x_{k+1} < h_{i+1}(x_1, \ldots, x_k) \land \Xi \]
\[ \Phi \land h_{i+1}(x_1, \ldots, x_k) < x_{k+1} < h_{i+2}(x_1, \ldots, x_k) \land \Xi \]
\[ \vdots \]
\[ \Phi \land h_j(x_1, \ldots, x_k) < x_{k+1} < h_{j+1}(x_1, \ldots, x_k) \land \Xi \]
(with the same \( \Xi \)) RSFC can replace them with a single formula
\[ \Phi \land h_i(x_1, \ldots, x_k) < x_{k+1} < h_{j+1}(x_1, \ldots, x_k) \land \Xi. \]

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 \( \Phi \) from the formula \( \Psi \Phi \). Each \( \Psi_i \) has a form \( \Psi_i = \Phi \land \Upsilon_i \), so we can write \( \Psi \Phi = \Phi \land (\Upsilon_0 \lor \cdots \lor \Upsilon_t) \).

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
\[
x^4 + y^2 + z^2 + t^2 < 1 \land x^2 + y^4 + z^2 + t^2 < 1 \\
\land x^2 + y^2 + z^4 + t^2 < 1 \land x^2 + y^2 + z^2 + t^4 < 1 \land x^2 + y^2 + z^2 + t^2 < 1
\]
gives a one cell description of set \( A \)
\[
-1 < x < 1 \land \sqrt{1 - x^2} < y < \sqrt{1 - x^2} \land \sqrt{1 - x^2 - y^2} < z < \sqrt{1 - x^2 - y^2} \\
\land \sqrt{1 - x^2 - y^2 - z^2} < t < \sqrt{1 - x^2 - y^2 - z^2}
\]
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
\[
x^2 + y^2 + z^2 < 9 \land y^2 < x^2 + z^2 - 1
\]
produced with InequalityGraphics.
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 $\text{SFRP} \circ \text{PR}$ (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 $c_{\text{all}}$ of all four-dimensional cells the algorithms needed to construct, the number $c_{\text{out}}$ 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$p_4$</th>
<th>$d_4$</th>
<th>$p_3$</th>
<th>$d_3$</th>
<th>$p_2$</th>
<th>$d_2$</th>
<th>$p_1$</th>
<th>$d_1$</th>
<th>$c_{all}$</th>
<th>$c_{out}$</th>
<th>$p_B$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGCAD</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>19</td>
<td>16</td>
<td>142</td>
<td>96</td>
<td>6967</td>
<td>29</td>
<td>82</td>
<td>540</td>
</tr>
<tr>
<td>GCAD</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>16</td>
<td>16</td>
<td>72</td>
<td>53</td>
<td>29</td>
<td>10</td>
<td>36</td>
</tr>
</tbody>
</table>

Table 2. Systems of strict inequalities.

<table>
<thead>
<tr>
<th>Inequalities</th>
<th>CAD</th>
<th>#c</th>
<th>GCAD</th>
<th>#c</th>
<th>#p</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_1 \land B_2$</td>
<td>3.92</td>
<td>1</td>
<td>0.17</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$B_1 \land B_4$</td>
<td>0.67</td>
<td>0</td>
<td>0.08</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_1 \land B_2 \land B_3$</td>
<td>73.05</td>
<td>2</td>
<td>1.32</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$B_1 \land B_2 \land B_4$</td>
<td>30.45</td>
<td>0</td>
<td>0.59</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B_1 \land C_1$</td>
<td>17.62</td>
<td>1</td>
<td>0.65</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$B_1 \land C_2$</td>
<td>31.62</td>
<td>0</td>
<td>0.92</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T \land C_1$</td>
<td>774.8</td>
<td>17</td>
<td>5.55</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$T \land B_2$</td>
<td>3000+</td>
<td>?</td>
<td>1.1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$HB_1 \land HB_2 \land HB_3$</td>
<td>415</td>
<td>0</td>
<td>3.76</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$HT \land HB_2 \land HB_3$</td>
<td>3000+</td>
<td>?</td>
<td>6.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T \land C_1 \land B_2$</td>
<td>3000+</td>
<td>?</td>
<td>30.79</td>
<td>28</td>
<td>31</td>
</tr>
<tr>
<td>$HT \land C_1 \land HB_2$</td>
<td>3000+</td>
<td>?</td>
<td>19.74</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

- $B_1 = x^2 + y^2 + z^2 < 1$
- $B_2 = (x - 1)^2 + (y - 1)^2 + (z - 1)^2 < 1$
- $B_3 = (x - 1)^2 + (y - 1)^2 + \left( z + \frac{1}{2} \right)^2 < 1$
- $B_4 = \left( x - \frac{3}{2} \right)^2 + (y - 2)^2 + z^2 < 1$
\[ C_1 = x^2 + y^2 + z^2 + 2yz - 4y - 4z + 3 < 0 \]
\[ \land y - 1 < z \land z < y + 1 \]
\[ C_2 = x^2 + y^2 + z^2 + 2yz - 4y - 4z + 3 < 0 \]
\[ \land y + 1 < z \land z < y + 2 \]
\[ T = z^4 + (2y^2 + 2x^2 + 6)z^2 + y^4 + 2x^2y^2 - 10y^2 + x^4 - 10x^2 + 9 < 0 \]
\[ HB_1 = B_1 \land x + y + z < 0 \]
\[ HB_2 = B_2 \land x + y + z > 3 \]
\[ HB_3 = B_3 \land x + y + z < \frac{3}{2} \]
\[ HT = T \land x + y < 0. \]

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


Originally Received 15 July 1999
Accepted 29 November 1999