SOLVING ALGEBRAIC INEQUALITIES

ADAM STRZEBOŃSKI

ABSTRACT. We study the problem of solving, possibly quantified, systems of real algebraic equations and inequalities. We propose a way of representing solution sets in a computer algebra system and present an algorithm for computing the representation. We also discuss specialized algorithms for solving several important special cases, including finding "generic solutions", deciding existence of solutions, global optimization of algebraic functions subject to algebraic constraints, and solving linear equation and inequality systems. Finally, we give some examples and present results of some experiments with our implementation of the algorithms within Mathematica.

1. INTRODUCTION

Let us first state the main problem in precise terms. To this end, let us explain what do we mean by a system of real algebraic equations and inequalities.

Definition 1.1. A basic algebraic function given by a polynomial \( f(x_1, \ldots, x_n, y) \) and an integer \( k \) is the function

\[
\text{Root}_{y,k} f : \mathbb{R}^n \ni x_1, \ldots, x_n \mapsto \text{Root}_{y,k} f(x_1, \ldots, x_n) \in \mathbb{R}
\]

where \( \text{Root}_{y,k} f(x_1, \ldots, x_n) \) is the \( k \)-th 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 \( k \) real roots. The real roots are ordered by the increasing value, counting multiplicities.

A real algebraic function is an arbitrary composition of polynomials, basic algebraic functions, and rational powers. The domain of a real algebraic function \( f \) is a set of those points in \( \mathbb{R}^n \), for which all basic algebraic functions in \( f \) are defined, all negative powers in \( f \) have non-zero bases, and all non-integer rational powers in \( f \) have non-negative real arguments.

A system of real algebraic equations and inequalities in variables \( x_1, \ldots, x_n \) is an alternative of conjunctions of

\[
f_k(x_1, \ldots, x_n) \rho_k g_k(x_1, \ldots, x_n)
\]

where each \( \rho_k \) is one of \( <, \leq, \geq, >, =, \neq \), and each \( f_k \) and \( g_k \) is a real algebraic function.
A point \( (a_1, \ldots, a_n) \in \mathbb{R}^n \) is a solution of the system if for at least one term of the alternative, the point belongs to the domain of all algebraic functions in this term, and satisfies all the equations and inequalities in this term.

Example 1.2. We do not require that a solution must belong to domains of all algebraic functions in the entire system. For instance the solution set of

\[
x \geq 0 \land \sqrt{x} < 1 \lor x < 0 \land \sqrt{-x} < 1
\]

is \(-1 < x < 1\), even though only \( 0 \) belongs to domains of both radicals.

Date: January 4, 1999.
Definition 1.3. A quantified system of real algebraic equations and inequalities in free variables $x_1, \ldots, x_n$ and quantified variables $t_1, \ldots, t_m$ is a logical formula of the form

$$Q_1 t_1 \ldots Q_m t_m S(t_1, \ldots, t_m; x_1, \ldots, x_n)$$

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

By Tarski’s theorem (see [14]), solution sets of, possibly quantified, real algebraic equation and inequality systems are semialgebraic. (Algebraic functions can be successively replaced with new variables, and the condition of being the $k$-th root of a polynomial can be written as a quantified polynomial equation and inequality system.) In particular they need not to be finite so we cannot enumerate them. Instead by solving a system we will mean finding a description of the solution set in some simple and useful standard form not containing quantifiers. We claim that such a simple and useful form is the cylindrical solution form described below, and in the following sections we will give examples showing how the form is useful for instance in global optimization of algebraic functions subject to algebraic constraints, computing multidimensional integrals, and visualization of semialgebraic sets.

Definition 1.4. A cylindrical form in variables $x_k, \ldots, x_n$ with parameters $x_1, \ldots, x_{k-1}$ is defined recursively to be

$$B_1 \land C_1 \lor \ldots \lor B_m \land C_m$$

where $C_i$ is a cylindrical form in variables $x_{k+1}, \ldots, x_n$ with parameters $x_1, \ldots, x_k$, and $B_i$ is one of

$$f(x_1, \ldots, x_{k-1}) \quad \rho \quad x_k \quad \sigma \quad g(x_1, \ldots, x_{k-1})$$

$$\begin{align*}
    f(x_1, \ldots, x_{k-1}) & \quad \rho \quad x_k \\
    g(x_1, \ldots, x_{k-1}) & \quad \sigma \\
    x_k = g(x_1, \ldots, x_{k-1}) & \quad \text{true}
\end{align*}$$

where $f$ and $g$ are basic algebraic functions, and $\rho$ and $\sigma$ are $<$ or $\le$. A cylindrical form in no variables is the Boolean constant true.

A cylindrical solution form of an equation and inequality system

$$Q_1 t_1 \ldots Q_m t_m S(t_1, \ldots, t_m; x_1, \ldots, x_n)$$

is a cylindrical form in variables $x_1, \ldots, x_n$ with no parameters describing the solution set of the system.

In our implementation basic algebraic functions are represented by Mathematica Root objects. (See [15], [13]. Mathematica 4.0 no longer factors the defining polynomials of non-constant Root objects.) Basic algebraic functions given by polynomials of degree less than three are represented in terms of rational functions and square roots.

Example 1.5. A cylindrical solution form of $x^2 + y^2 + z^2 < 1$ is

$$-1 < x < 1 \land -\sqrt{1-x^2} < y < \sqrt{1-x^2} \land$$

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

A cylindrical solution form of

$$\exists x : x^8 + ax + b = 0 \land -1 < x < 1$$
SOLVING ALGEBRAIC INEQUALITIES

is

\[ a \leq -8 \land -1 + a < b < -1 - a \lor -8 < a \leq 0 \land -1 + a < b \leq r(a) \lor 0 < a < 8 \land -1 - a < b \leq r(a) \lor a \geq 8 \land -1 - a < b < -1 + a \]

where \( r(a) = \text{Root}_{y,1}(-823543a^8 + 16777216y^7). \)

The Cylindrical Algebraic Decomposition (CAD) algorithm (see [2], [1]) is a constructive proof of the fact that every semialgebraic set, and hence every solution set of a quantified algebraic equation and inequality system, can be represented by a cylindrical solution form. In Section 2 we present an algorithm (based on CAD) allowing to compute such representation. Next, we describe several simpler algorithms that can be used in some important for applications special cases. Finally, we will show some experimental results.

2. THE MAIN ALGORITHM

The input is a quantified system

\[ Q_1 t_1 \ldots Q_m t_m S(t_1, \ldots, t_m, x_1, \ldots, x_n) \]

of real algebraic equations and inequalities in free variables \( x_1, \ldots, x_n \) and quantified variables \( t_1, \ldots, t_m \), where \( m \) may be zero. The algorithm computes a cylindrical solution form of the system.

2.1. Polynomialization. First we successively replace algebraic functions with new variables, starting with the innermost basic algebraic functions or radicals. When replacing \( \text{Root}_{y,k} f(x_1, \ldots, x_n) \) with a new variable \( z \), we add the equation \( f(x_1, \ldots, x_n, z) = 0 \) to all terms of the alternative \( S \) which contained the replaced basic algebraic function. Similarly, we replace a radical \( f^{p/q} \) with \( z^p \) and add the equation \( z^q = f \) and the inequality \( f \geq 0 \) to all terms of the alternative \( S \) which contained the replaced radical. We keep track of which variables replace what algebraic functions and of the order in which they were replaced.

Next, we put all equations and inequalities in the form \( f < 0, f \leq 0, f = 0, \) or \( f \neq 0 \), put all rational functions in the “common denominator” form, and replace equations and inequalities involving rational functions using equivalences

\[
\begin{align*}
\frac{f}{g} < 0 & \iff f > 0 \land g < 0 \lor f < 0 \land g > 0 \\
\frac{f}{g} \leq 0 & \iff f \geq 0 \land g < 0 \lor f \leq 0 \land g > 0 \\
\frac{f}{g} = 0 & \iff f = 0 \land g \neq 0 \\
\frac{f}{g} \neq 0 & \iff f \neq 0 \land g \neq 0 
\end{align*}
\]

Finally, we put the, now polynomial, equation and inequality system in the disjunctive normal form.

2.2. Projection. This is the projection phase of the CAD algorithm (see [2], [1]). First we project with respect to the variables replacing algebraic function, in the reverse replacement order. Then we project the quantified variables starting with the innermost quantifiers i.e. from \( t_m \) to \( t_1 \). Finally, we project out free variables \( x_n \) through \( x_2 \). We can reorder variables within blocks of identical quantifiers and within the free variables if we do not have a preference as to the order of free variables in the solution. In this case we use a heuristics attempting to minimize the size and degrees of the projection. We use two types of projection.

We start with the “short projection”. If there are any equational constraints present or if the projected variable replaces an algebraic function, we use the equational constraint
case projection suggested in [3]. If there are several equational constraints, we select the pivot (and the projection variable, if we have a choice) based on first whether its all factors have constant leading coefficients, and second on how low is its degree. Equational constraints with nontrivial contents are disqualified. We propagate the remaining equational constraints using the fact that a resultant of equational constraints is an equational constraint. After we run out of replacement variables and equational constraints we continue with the McCallum’s projection operator for well-oriented sets of polynomials (see [7], [8]).

The solution form construction phase may fail if the short projection was used, and one of the polynomials of McCallum’s projection becomes identically zero on a positive-dimensional cell, or if an equational constraint used as a pivot becomes identically zero on a cell. In this case we use the full projection operator described in [5].

2.3. Construction of solution form. Since we are using algebraic functions to describe the solution set we, as opposed to the classical CAD algorithm, do not need to generate and store all the cells before constructing the solution form. Therefore we can use the following recursive algorithm RCSF which, on the k-th recursion level, generates the solution form for the first k variables (in the inverse projection order) belonging to a specified cell. (Remember, in this order the free variables come first, then the quantified variables, outermost quantifiers first, and at the end the variables replacing algebraic functions, in the replacement order.) To generate the full solution we call RCSF on the 0-th recursion level.

Algorithm 2.1. RCSF

Input:

- cell_data contains information about the cell over which we are constructing the solution. This includes a sample point in the cell, i.e. values for the first k variables, and information whether the cell is zero-dimensional and which (not all) elements of the alternative in S are marked as known to be false on the cell.
- proj_data contains all the information from the first two phases of the algorithm, i.e. information about the system of polynomial equations and inequalities S, the variables replacing algebraic functions, the subsequent projection types, projection variables, quantifiers, and sets of projection polynomials.

Output:

- A formula representing solutions of the system, for the first k variables belonging to the input cell. If the k+1-st variable is a free variable, it is a cylindrical form in the free variables left, with the first k variables as parameters. Otherwise it is true or false.

(1) Let v be the k+1 -st variable.
(2) If v is a variable replacing a basic algebraic function Root_{y,p,f}, find the real roots of f in y, after replacing the first k variables with coordinates of the sample point. (The chosen projection order guarantees that f after the replacement becomes a univariate polynomial.) If there are at least p roots, counting multiplicities, choose the p-th root as the k+1-st coordinate of the sample point, and check if substitution of the new sample point makes any more elements of the alternative in S false. Otherwise, choose 0 as the k+1-st coordinate and mark all the elements of the alternative in S which contain v as known to be false on the cell. If all the elements of the alternative in S are known to be false on the cell return false. If v was the last variable return true, else call RCSF recursively.
(3) If \( v \) is a variable replacing a radical \( f^{1/q} \), let \( a \) be \( f \) with the first \( k \) variables replaced with the coordinates of the sample point. The order of projection chosen guarantees that \( a \) is a constant. If \( a \) is nonnegative, choose the \( k+1 \)-st coordinate of the sample point to be \( a^{1/q} \), and check if substitution of the new sample point makes any more elements of the alternative in \( S \) false. Otherwise choose the coordinate to be 0 (since \( f \geq 0 \) was included in all elements of the alternative containing \( v \), these elements are already marked as known to be false). If all the elements of the alternative in \( S \) are known to be false on the cell return false. If \( v \) was the last variable return true, else call RCSF recursively.

(4) If \( v \) was projected out using an equational constraint let \( pts \) be all the real roots of factors of the pivot, after replacing the first \( k \) variables with coordinates of the sample point. Otherwise, let \( pts \) be a set of all the real roots of the \( k+1 \)-variate projection polynomials, after replacing the first \( k \) variables with coordinates of the sample point, and of rational points, one in every interval left after removing the roots from the real line. The algorithm can fail at this point if \( v \) was projected out using an equational constraint and any of factors of the pivot becomes identically zero after the substitution, or if the McCallum’s projection was used, the cell is not zero-dimensional, and one of the projection polynomials becomes identically zero after the substitution. In this case we start over with the full projection. If the McCallum’s projection was used, the cell is zero-dimensional, and one of the projection polynomials becomes identically zero after the substitution we use a suitable derivative instead, as described in [8].

(5) Take the subsequent elements of \( pts \) as the \( k+1 \)-st coordinates of the sample point, and determine the solution of the system over the corresponding cell. First check if substitution of the new sample point makes any more elements of the alternative in \( S \) false. If all the elements of the alternative in \( S \) are known to be false on the cell the solution is false. If \( v \) was the last variable, the solution is true. Otherwise we compute the solution calling RCSF recursively.

(6) If \( v \) is a quantified variable we may return the answer without looking at all the \( pts \). The possible solutions of the system over the cells corresponding to elements of \( pts \) are true and false, since all the remaining variables are quantified. If \( v \) is quantified by the existential quantifier and a solution over the cell corresponding to one of the \( pts \) is true we return true. If \( v \) is quantified by the general quantifier and a solution over the cell corresponding to one of the \( pts \) is false we return false. If we went through all the \( pts \) without returning early we return false for the existential quantifier and true for the general quantifier. Because in this case we may find the answer without looking at all \( pts \) we want to try the easier cases first. To this end we take the elements of \( pts \) in 5. in order of increasing degrees of their minimal polynomials.

(7) We are left with the case when \( v \) is a free variable. The polynomials used in 4. to compute \( pts \) are delineable on the cell, therefore the different real roots of these polynomials on the cell are values of non-intersecting basic algebraic functions given by these polynomials. We keep track of the basic algebraic function corresponding to each root, and so we can write the part of answer corresponding to each of the elements of \( pts \) as the conjunction of one of \( v = f, f < v < g, v < f, \) and \( v > f \), and the solution of the system over the cell corresponding to the element, where \( f \) and \( g \) are the basic algebraic functions corresponding to the appropriate roots. If \( v \) was projected out using an equational constraint we return the
alternative of the constructed parts of the answer, else if the solutions of the system over the cells corresponding to adjoining elements of $pts$ are identical we join the corresponding parts of the answer and return the alternative of the resulting formulas.

For finding roots of polynomials with algebraic number coefficients we use the algorithm described in [12], modified to compute real roots only. To avoid repeating computations we cache resultant, factorization, and real root isolation computations.

The recursive nature of the algorithm and joining of the cells which can be joined in 7. results in a much lower memory usage of our algorithm than that of the classical CAD algorithm. In practice we have not seen an example in which the space complexity rather than the time complexity would be the main limitation. Step 5. of the algorithm allows parallelization of computations, however our implementation is purely sequential.

3. THE FULL DIMENSIONAL CASE

An important category of problems for which the algorithm can be substantially simplified is when the solution set of the inequality system is open, or when we are interested only in the full dimensional part of the solution set. One such case is the decision problem for systems of strong polynomial inequalities, i.e. the problem of deciding whether an open semialgebraic set is nonempty. In this case the original system contains only strong polynomial inequalities and all variables are existentially quantified. In [9] McCallum noticed that it suffices to construct sample points in full dimensional cells only, thus eliminating the need for any algebraic number computations. In [11] the author has shown that in this case we can also use a simpler projection operator. Here we present an extension of this idea to systems containing free variables.

Suppose we have a system of real polynomial inequalities (no equations, algebraic functions, or quantifiers). In some applications, like multidimensional integration or graphical visualization, it may be enough to know a “generic” solution set, which is correct “up to a lower dimensional set”. Let us be more precise.

**Definition 3.1.** Let $S$ be a system of real polynomial inequalities in $n$ variables, and let $A \subseteq \mathbb{R}^n$ be the solution set of $S$. A semialgebraic set $B \subseteq \mathbb{R}^n$ is a generic solution set of $S$ if the error set $A \setminus B \cup B \setminus A$ is at most $n - 1$-dimensional.

The following algorithm computes a generic solution set, and gives an at most $n - 1$-dimensional set containing the error set.

**Algorithm 3.2.** GCAD

**Input:**
- A system $S$ of real polynomial inequalities.
- A cylindrical solution formula describing a generic solution set of $S$, and a set of equations, such that the error set is contained in the union of their zero sets.

**Output:**
- A cylindrical solution formula describing a generic solution set of $S$, and a set of equations, such that the error set is contained in the union of their zero sets.

Let us first note that if $S$ contains weak inequalities or inequations, and $S'$ is $S$ with weak inequalities replaced with their strong versions and inequations removed, then a generic solution set $B$ of $S'$ is a generic solution set of $S$, and the error set of $B$ as a solution of $S$ is contained in the union of the error set of $B$ as a solution of $S'$ and the zero sets of equations.
corresponding to the weak inequalities and the inequations. Therefore in the following we may assume that $S$ consists only of strong inequalities.

For a set of polynomials $\text{polys}$, let $\text{SFRP}(\text{polys})$ denote a set of square-free and relatively prime polynomials multiplicatively generating $\text{polys}$. For a set of square-free and relatively prime polynomials $\text{qolys}$, let $\text{GP}(\text{qolys}, v)$ denote the projection of $\text{qolys}$ with respect to $v$ used in [12]. By definition, $\text{GP}(\text{qolys}, v)$ consists of the leading coefficients, discriminants, and pairwise resultants of $\text{qolys}$.

Let $S$ be a system of strong polynomial inequalities in variables $x_1, \ldots, x_n$, transformed to a form with zero right hand sides of inequalities, and let $\text{polys}$ be the set of left hand sides of inequalities in $S$. First we compute the set of projections $\text{pro js} = (pr_n, \ldots, pr_1)$, where $pr_n = \text{SFRP}(\text{polys})$ and $pr_k = \text{SFRP}(\text{GP}(pr_{k+1}, x_{k+1}))$. Then we obtain the solution formula and the equations for the error set calling the following recursive algorithm with $k = 0$.

**Algorithm 3.3. **$\text{RGCSF}$

**Input:**
- $f \text{pro js} = (pr_n, \ldots, pr_{k+1})$
- $\text{epro js} = f \text{pro js}$ with $x_1, \ldots, x_k$ replaced with the (rational number) coordinates of a sample point of the $k$-dimensional cell $c$ over which we are constructing the solution. All polynomials of $pr_k$ have constant non-zero signs on $c$.
- $\text{ineqs}$ is $S$ with $x_1, \ldots, x_k$ replaced with the coordinates of the sample point.

**Output:**
- A cylindrical form $\text{cfm}$ in $x_{k+1}, \ldots, x_n$ with $x_1, \ldots, x_k$ as parameters, and a set $\text{eqns}$ of polynomials in $x_1, \ldots, x_n$ such that for any point $a = (a_1, \ldots, a_n)$, with $(a_1, \ldots, a_k)$ in the cell, if $a$ is not a zero of any of $\text{eqns}$ then $a$ is a solution of $S$ iff $a$ is a solution of $\text{cfm}$.

1. If $\text{ineqs}$ is true or false we return $\text{ineqs}$ and no equations.
2. Let $ps$ be the last element of $\text{epro js}$. $ps$ is a set of univariate polynomials in $x_{k+1}$. Isolate roots of $ps$, and find rational numbers $pts$ one in every interval left after removing the roots from the real line.
3. Polynomials $pr_{k+1}$ are delineable on $c$. (Their leading coefficients, discriminants, and pairwise resultants have constant non-zero signs, so they have a fixed number of real roots each and the roots do not intersect.) Hence the real roots of these polynomials on $c$ are values of non-intersecting basic algebraic functions given by these polynomials. If we take the subsequent elements of $pts$ as the $k + 1$-st coordinates of sample points, we get sample points of $k + 1$-dimensional cells on which elements of $pr_{k+1}$ have constant non-zero signs. We call $\text{RGCSF}$ recursively on each such cell.
4. We can write the cylindrical form corresponding to each element of $pts$ as a conjunction of one of $f < v < g$, $v < f$, and $v > f$, and the cylindrical form returned by the recursive call, where $f$ and $g$ are the basic algebraic functions corresponding to the appropriate roots. As $\text{eqns}$ we return all the polynomials given by the recursive calls, and all elements of $pr_{k+1}$ whose root separates two adjoining cells over which $S$ has solutions. If for cells corresponding to adjoining elements of $pts$ the cylindrical forms returned by the recursive calls are identical we join the corresponding cylindrical forms in the answer. We return the alternative of the resulting formulas.
Example 3.4. GCAD was used by Roger Germundsson in InequalityGraphics package. Here is a graphical representation of the solution set of inequality system

\[ x^2 + y^2 + z^2 \leq 9 \land y^2 \leq x^2 + z^2 - 1 \]

As another application of the GCAD algorithm we can compute the volume of the figure above, which is \( 64\pi/3 \). Here the integration is by far more time consuming than the GCAD computation. We can also use the cylindrical solution form to compute the volume numerically. (The results of numerical and symbolic computation do agree.)

4. GLOBAL OPTIMIZATION

The problem we investigate in this section is to find the infimum of values of a real algebraic function \( f(x_1, \ldots, x_n) \) subject to algebraic equation and inequality constraints \( S(x_1, \ldots, x_n) \), and, if possible, find a point in which the infimum is attained. (If the solution set of \( S \) is not compact the infimum may not be attained.)

The problem is equivalent to finding the infimum \( y_{inf} \) of values of the new variable \( y \) on the solution set of the following quantified system of real algebraic equations and inequalities,

\[ \exists (x_1, \ldots, x_n) \cdot S(x_1, \ldots, x_n) \land y \geq f(x_1, \ldots, x_n) \]

and if possible a point \( (a_1, \ldots, a_n) \) satisfying the constraints \( S \) and such that \( y_{inf} = f(a_1, \ldots, a_n) \).

For this purpose we use a modified version of the Main Algorithm. We use the identical polynomialization and projection phases. The last projection gives us a set of univariate polynomials in \( y \). In step 5. of first call to RCSF, with the main variable \( y \), we order \( pts \) with respect to their increasing values, and call RCSF recursively with subsequent elements of \( pts \) until we get answer \textit{true}. We also modify RSCF to return a sample set of values for existentially quantified variables which satisfies the system. If the smallest element of \( pts \), for which we get \textit{true} from the recursive call, is a root of one of the projection polynomials,
then it is the infimum and the sample point returned by the recursive call is a point at which
the infimum is attained. Otherwise the infimum is equal to the largest root of one of the
projection polynomials smaller than the element of pts, or $-\infty$ if there are no smaller roots,
and the infimum is not attained.

To find an infimum of a polynomial or a rational function subject to strong polynomial
inequality constraints (or if we know that the set of points satisfying the constraints is
contained in the closure of its interior), we can use a simpler algorithm based on ideas from
Section 3. We project out variables $x_1, \ldots, x_n$ using the projection described in Section 3,
find rational numbers pts one in every interval left after removing the roots of the last
projection from the real line, and then run recursively a modification of RCSF, with all
$x_1, \ldots, x_n$ existentially quantified, over the subsequent pts in order of increasing values,
until we get true. Then the infimum is equal to the largest root of one of the projection
polynomials smaller than the current element of pts, or $-\infty$ if there are no smaller roots.
The modified RCSF constructs only full dimensional cells, i.e. in step 4. takes only the
rational numbers between the roots of projection polynomials.

5. The Linear Case

For linear decision problem, i.e. for solving quantified equation and inequality systems
with all variables existentially quantified and all equations and inequalities linear we use a
method based on the Simplex algorithm from Linear Programming. Since
$$\exists x : a(x) \lor b(x) \iff \exists x : a(x) \lor \exists x : b(x)$$
we may assume that the system is a conjunction of equations and inequalities. We use the
following algorithm.

Algorithm 5.1. LINSIM

Input:
• A system $S$ which is a conjunction of linear equations and inequalities in $x_1, \ldots, x_k$.

Output:
• true or false depending on whether $S$ has solutions. If the answer is true the
algorithm can also give a point $(a_1, \ldots, a_n)$ satisfying the system $S$.

(1) As in the Simplex algorithm, we add new variables to replace inequalities with
equations, with the only difference being that for strong inequalities we require
that the new variables be strictly positive.

(2) We Gaussian eliminate the original unrestricted variables. We get a linear system
$LS_+$ of equations in positive and non-negative variables, and an upper-triangular
matrix which allows to find values of the original variables, once we have a solu-
tion of $LS_+$.

(3) We use the first phase of the Simplex algorithm to find a solution of $LS_+$ with all
variables nonnegative. If there is no such solution we return false.

(4) We look if there is a variable which should be positive, but is zero in the solution.
If no we compute the values of the original variables from the matrix in step 2.
and return true. Otherwise we use the second phase of Simplex algorithm to find
the maximal value $\max$ of the variable.

(5) If $\max = 0$ we return false. Otherwise we set the value of the variable to $\max/2$
(the solution set of a conjunction of linear equations and inequalities is convex),
use the first phase of Simplex algorithm to find non-negative solutions for the
remaining variables, and go to step 4.
In [6] Loos and Weispfenning describe an algorithm which allows to eliminate a quantifier if all the equations and inequalities of the system are linear in the quantifier’s variable. The result of the elimination however is not given in the cylindrical form. We use this algorithm as a preprocessor to the main algorithm, i.e. we eliminate the innermost quantifiers if the system is linear in their variables and then call the main algorithm on the result. In the last section we compare this approach with the main algorithm without the linear preprocessing. We also use this algorithm whenever we do not insist on the cylindrical form of the result (for example because there are too many free variables in too high degrees).

6. Experimental Results

In this section show some experimental results obtained with our implementations of the algorithms presented in this paper. The algorithms were implemented in the C kernel of Mathematica. The examples were run on a Pentium II, 233 MHz computer with 64 MB of RAM. The timings are in seconds.

6.1. The full solution vs. the generic solution. Here we use strong inequality examples given in [9]. We compare the timings of deciding the problems with all variables existentially quantified (the DEC column), finding a cylindrical solution form using the Main Algorithm (the MAIN column), and finding a cylindrical solution form for a generic solution set using the GCAD algorithm (the GCAD column). The cylindrical forms are computed in the \((x,y,z)\) order of variables. The \#c columns give the total number of cells after putting the solutions given by MAIN and GCAD in the disjunctive normal form (answer false counts as no cells). The \#e column gives the number of equations whose solutions cover the error set, as given by the GCAD algorithm. Following the notation of [9] let us put.

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

The decision algorithm uses a heuristic to determine the order of projection, and because of this the timings of DEC and GCAD may differ even if there are no solutions.
6.2. **Equational constraints.** We compare timings of the Main Algorithm (**MAIN**) and a version of the algorithm which does not use the special case projection using equational constraints (**NOEQC**).

(1) Catastrophe Surface and Sphere from [8]

\[ x^2 + y^2 + z^2 = 1 \land z^3 + xz + y = 0 \]

**MAIN** returns a solution containing 8 cells after 1.18 seconds. **NOEQC** returns a solution containing 16 cells after 81.26 seconds.

(2) Solotareff’s problem from [8]

\[
\begin{align*}
\exists b \exists u \exists v : & -1 < u < v < 1 \land b < 2 \land \\
& a^4 + 2a^3 - au^2 - bu + a + b - 3 = 0 \land \\
& v^4 + 2v^3 - av^2 - bv + a - b + 1 = 0 \land \\
& 4a^3 + 6a^2 - 2au - b = 0 \land \\
& 4v^3 + 6v^2 - 2av - b = 0
\end{align*}
\]

Both algorithms return \( a = \text{Root}_{y,1}(81y^3 - 180y^2 + 448y - 432) \), **MAIN** after 1.86 seconds, **NOEQC** after 41.21 seconds.

(3) When the quintic \( x^5 + ax^2 + bx + c \) has at least two different real roots?

\[ \exists x \exists y : x^5 + ax^2 + bx + c = 0 \land y^5 + ay^2 + by + c = 0 \land x \neq y \]

Both algorithms return a solution consisting of 4 cells, **MAIN** after 21 seconds, **NOEQC** after 317 seconds. The solution is

\[
\begin{align*}
a < 0 & \land \ (b < r(a) \land r_1(a,b) \leq c \leq r_2(a,b)) \lor \\
& b = r(a) \land r_3(a,b) \leq c \leq r_4(a,b) \lor \\
& r(a) < b < s(a) \land r_1(a,b) \leq c \leq r_2(a,b)) \lor \\
a \geq 0 & \land \ b < s(a) \land r_1(a,b) \leq c \leq r_2(a,b)
\end{align*}
\]
where
\[
\begin{align*}
r(a) &= \text{Root}_{y,1}(320y^3 + 27a^4) \\
s(a) &= \text{Root}_{y,1}(80y^3 - 27a^4) \\
r_k(a, b) &= \text{Root}_{x,3}(3125y^3 + 2250a^2by^2 - 1600ab^3y + 108a^5y + 256b^5 - 27a^4b^2)
\end{align*}
\]

(4) When the difference between the largest and the smallest root of the cubic \( x^3 + ax + b \) is at least 1?
\[
\text{Root}_{y,3}(y^3 + ay + b) - \text{Root}_{y,1}(y^3 + ay + b) \geq 1
\]

Both algorithms return
\[
a \leq -\frac{1}{2} \wedge -\frac{2\sqrt{-a^3}}{3\sqrt{3}} \leq b \leq \frac{2\sqrt{-a^3}}{3\sqrt{3}} \vee -\frac{1}{2} < a < -\frac{1}{2} \wedge
\left\{-\frac{\sqrt{-4a^3 - 9b^2 - 6ab - 1}}{3\sqrt{3}} \leq b \leq \frac{\sqrt{-4a^3 - 9b^2 - 6ab - 1}}{3\sqrt{3}} \vee
\right\}
\]
\[a = -\frac{1}{2} \wedge b = 0
\]

**MAIN** after 2.46 seconds, **NOEQC** after 9.12 seconds.

6.3. **Global optimization.** Here we use our global optimization algorithm and discuss the use of its two versions, one constructing all cells **ACGO** and the other constructing only the full dimensional cells **FDGO**.

(1) A geometric problem from [10]. Find all values of \( k \) for which the inequality
\[
a^3 + b^3 + c^3 \geq 3abc + k(a - b)(b - c)(c - a)
\]
is true for all \( a, b, \) and \( c \) sides of a triangle. Since by permutation of \( a, b, \) and \( c \) we can change the sign of the coefficient at \( k \), we see that the allowable values for \( k \) form an interval symmetric with respect to zero. Therefore it is enough to compute the maximum of \( k \) for which the inequality is true for all \( a, b, \) and \( c \), which is the same as the infimum of \( k \) for which there are \( a, b, \) and \( c \), such that the opposite inequality is true. So we need to compute the infimum of \( k \) on
\[
a > 0 \wedge b > 0 \wedge c > 0 \wedge k > 0 \wedge
\]
\[
a < b + c \wedge b < c + a \wedge c < a + b \wedge
\]
\[
a^3 + b^3 + c^3 < 3abc + k(a - b)(b - c)(c - a)
\]
We have strong inequalities only, so we can use **FDGO**, which returns \( \text{Root}_{y,2}(y^4 - 72y^2 - 432) \) after 12.35 seconds. In radicals, the answer is \( 2\sqrt{9} + 6\sqrt{3} \). **ACGO** did not do this example in an hour.

(2) Maximize \( \sqrt{x} + \sqrt{y} \) on the ellipse \( x^2 + \frac{y^2}{4} = 1 \). We can solve the problem directly using **ACGO** in 2.32 seconds, or we can notice that the maximum is equal to the supremum of \( x + y \) on \( \frac{x^2}{4} + \frac{y^2}{9} < 1 \), which we can compute using **FDGO** in 0.12 seconds. The answer is \( \text{Root}_{y,1}(y^{12} - 39y^8 - 465y^4 - 2197) \), with **ACGO** we also get a point at which the maximum is attained, with **FDGO** we don’t.

(3) Find the minimal distance between the largest and the smallest root of the cubic \( x^3 + ax + b \) assuming the cubic has three real roots and its discriminant is \(-1\). We need to use **ACGO** here. We get the answer \( 2^{2/3} \) for \( a = -2^{-2/3} \) and \( b = 0 \), after 5.58 seconds.
6.4. Linear quantifier elimination as preprocessor. Here we investigate whether the Loos-Weispfenning linear quantifier elimination algorithm may be useful as a preprocessor to the MAI N algorithm. The examples are conjunctions of randomly generated weak polynomial inequalities with all but one variable existentially quantified. The coefficients are rational numbers with 3 decimal digit numerators and denominators. \( \#v \) gives the total number of variables in the system, \( \#ln \) gives the number of linear variables, \( \text{deg} \) is the total degree of the polynomials in the remaining variables, \( \#in \) gives the number of inequalities, and \( \text{dns} \) gives the density of polynomials. In examples 1 through 6 the coefficients at the linear variables are polynomials of total degree \( \text{deg} \) in the remaining variables, in examples 7 through 12 the coefficients at linear variables are constant (this is marked in the \( \text{cc} \) column). The algorithm LWPP eliminates the linear variables using the Loos-Weispfenning linear quantifier elimination algorithm and then calls the Main Algorithm. We require that the result be a cylindrical solution form so we have to call the Main Algorithm even if all quantified variables are linear.

The LWPP is faster in all examples except of 9 and 10, where the number of inequalities is larger.

### References


WOLFRAM RESEARCH INC. AND Jagiellonian University, 100 TRADE CENTRE DRIVE, CHAMPAIGN, IL 61820, U.S.A.
E-mail address: adams@wolfram.com