QEPCAD B — a program for computing with semi-algebraic sets using CADs

Christopher W. Brown
Department of Computer Science, Stop 9F
United States Naval Academy
572C Holloway Road
Annapolis, MD 21402
wcbrown@usna.edu

Abstract

This report introduces QEPCAD B, a program for computing with real algebraic sets using cylindrical algebraic decomposition (CAD). QEPCAD B both extends and improves upon the QEPCAD system for quantifier elimination by partial cylindrical algebraic decomposition written by Hoon Hong in the early 1990s. This paper briefly discusses some of the improvements in the implementation of CAD and quantifier elimination via CAD, and provides somewhat more detail on extensions to the system that go beyond quantifier elimination. The author is responsible for most of the extended features of QEPCAD B, but improvements to the basic CAD implementation and to the SACLIB library on which QEPCAD is based are the results of many people's work, including: George E. Collins, Mark J. Encarnación, Hoon Hong, Jeremy Johnson, Werner Krandick, Richard Liska, Scott McCallum, Nicolas Robidoux, and Stanly Steinberg. Source code, documentation and installation instructions for QEPCAD B are all available at `www.cs.usna.edu/~qepcad`.

1 Introduction

In the early 1970s, George Collins developed Cylindrical Algebraic Decomposition (CAD) as the basis of his quantifier elimination algorithm. This algorithm was first implemented by Dennis Arnon in 1980. Subsequent implementations include Hoon Hong's QEPCAD [11], Strzebonski's implementation in Mathematica [20], and a current project to implement CAD in the Redlog system [13]. This paper describes QEPCAD B (version 1.6), an extension of Hong's QEPCAD system that not only improves the basic implementation of CAD and CAD-based quantifier elimination, but provides additional functionality as well. It is a command-line program written in C (small portions also in C++), and based on the SACLIB library of computer algebra functions. Source code and documentation are available at: `http://www.cs.usna.edu/~qepcad`.

The following section, Section 2, provides a brief description of CADs, semi-algebraic sets, and the quantifier elimination problem. It is only intended to provide sufficient background to understand what kinds of problems QEPCAD B is able to solve. A more complete description of these topics may be found in, for example, [18], [11] or [9]. Section 2 describes the basic functionality of QEPCAD B — namely, quantifier elimination and quantifier-free formula simplification — and Section 4 describes features of QEPCAD B that facilitate more efficient use of CAD by applying it directly to problems rather than through the intermediate language of quantified Tarski formulas. Both Section 2 and Section 4 separate the user's view of QEPCAD B from implementation issues, so they may be (hopefully!) accessible for for people who are merely interested in what QEPCAD B can do as well as those who are interested in how QEPCAD B works.

Several examples of the use of QEPCAD B appear in this paper. They are intended to be compact, easily understood, and to illustrate how the facilities offered by the program solve problems. Timing comparisons of QEPCAD B to QEPCAD and to other pieces of software or examples of large "real-world" problems being solved by QEPCAD B are beyond the scope of this paper. The primary goal of this paper is to describe functionality that is, to the author's knowledge, unique and new in QEPCAD B: quantifier-free formula simplification though CAD, quantifier elimination for an extension of the language of Tarski formulas, and quantifier elimination for an extended class of quantifiers.[1]

2 CAD, semi-algebraic sets, and the quantifier elimination problem

CADs provide an *explicit* representation for semi-algebraic sets, which are subsets of real affine space that are defined by boolean combinations of polynomial equalities and inequalities (which we will henceforth refer to *Tarski formulas*, or simply as formulas). In introducing the concept of a CAD [10], Collins showed how to construct CAD representations of semi-algebraic sets from formulas, and vice-versa. Figure 1 shows a semi-algebraic set represented by both a defining formula and a CAD. The connection between CADs and quantification is that projection onto lower dimensional spaces is trivial in the CAD representation; after all, $\exists$ is simply projection, and $\forall$ is the negation of a projection (and negation is also trivial in the CAD representation). So, supposing we are given a quantified formula $(Q_1 x_{k+1}) \cdots (Q_{n-k} x_n) F$, where F is a formula in the variables $x_1, \ldots, x_n$ and the Q_i are in $\{\exists, \forall\}$, we can

[1] Some of this functionality has been available from the author as extensions to various versions of the QEPCAD system.

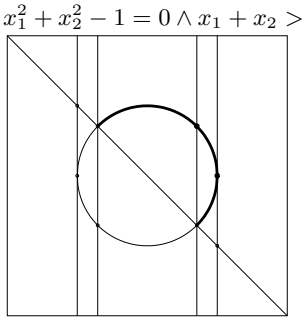

Figure 1: A semi-algebraic set represented by a defining formula and by a CAD.

1. construct a CAD of $\mathbb{R}^n$ representing the set defined by F,
2. apply the projections implied by $(Q_1 x_{k+1}) \cdots (Q_{n-k} x_n)$, which results in a CAD of $\mathbb{R}^k$ representing the set defined by $(Q_1 x_{k+1}) \cdots (Q_{n-k} x_n) F$, and
3. construct a formula representation of the set now represented by the CAD of $\mathbb{R}^k$.

What results from this process is a quantifier-free formula in the variables $x_1, \ldots, x_k$ that is logically equivalent over the reals to $(Q_1 x_{k+1}) \cdots (Q_{n-k} x_n) F$. Constructing such an equivalent, quantifier-free formula is what is known as the quantifier elimination problem, and the above procedure is Collins' algorithm for solving this problem.

Collins' CAD-based algorithm for quantifier elimination was not the first quantifier elimination algorithm. The first procedure was invented by Alfred Tarski in the 1930s [21]. However, it was the first algorithm with any hope of practical utility, having a complexity that is doubly exponential in the number of variables, but polynomial in the length, number of polynomials, maximum degree of polynomials, and maximum coefficient length of polynomials in the input formula. General quantifier elimination algorithms with better asymptotic complexities have been invented since (see [19], for example), as have special purpose methods with better practical performance on restricted types of input (see [22], for example). However, CAD-based quantifier elimination remains, to the best of the author's knowledge, the most efficient general quantifier elimination algorithm to be implemented. Moreover, CADs can be used for more than just quantifier elimination, and one of the main goals of QEPCAD B is to efficiently apply CADs to solving problems concerning semi-algebraic sets other than quantifier elimination.

3 Basic functionality

In this section we describe the various functions QEPCAD B performs, both from the user perspective and the implementation perspective.

3.1 Quantifier Elimination

The user perspective: The user's view of basic quantifier elimination in QEPCAD B is the same as in QEPCAD: the user enters a variable ordering and a quantified formula, and the program returns a quantifier-free equivalent formula. To provide an example, we consider the following problem: When does a real monic quadratic polynomial have roots only in the unit disc in the complex plane? A generic monic polynomial is of the form $P(z) = z^2 + bz + c$, and we're interested in z ranging over $\mathbb{C}$. QEPCAD B, however, computes over $\mathbb{R}$, so we substitute $x + iy$ for z and separate P into its real and imaginary parts:

$$P_r = x^2 - y^2 + bx + c, \;\; P_i = 2xy + by$$

Our question is, when is it true that $\forall x, y[P_r = 0 \wedge P_i = 0 \Longrightarrow x^2 + y^2 - 1 < 0]$? This is what a QEPCAD B session answering that question looks like:

```
==========================================================
Enter an informal description  between '[' and ']':
[When does a quadratic have all roots in the unit disc?]
Enter a variable list:
(b,c,x,y)
Enter the number of free variables:
2
Enter a prenex formula:
(Ax)(Ay)[ [ x^2 - y^2 + b x + c = 0 /\ 2 x y + b y = 0 ] ==> x^2 + y^2 - 1 < 0 ].
==========================================================

Before Normalization >
finish

An equivalent quantifier-free formula:

c - 1 < 0 /\ c - b + 1 > 0 /\ c + b + 1 > 0

==========================================================
```

The "answer" $c-1 < 0 \wedge c-b+1 > 0 \wedge c+b+1 > 0$ completely characterizes all real, monic, quadratic polynomials whose roots lie in the unit disc in the complex plane.

The implementation perspective: Although basic quantifier elimination in QEPCAD B is not much different from QEPCAD from the user perspective, the implementation is different in many ways, the most important of which are described below.

For the projection phase of CAD construction, QEPCAD B uses the reduced McCallum projection [5] when it can determine that a projection factor is never nullified, and the McCallum projection [17] otherwise. As described in McCallum's paper, his projection may fail, though this can always be detected in the lifting phase of CAD construction. QEPCAD B will issue an error message if failure of the McCallum projection is detected (at which point the user must restart the computation and explicitly instruct the program not to use McCallum's projection). Using the criteria described in [6], QEPCAD B is able to verify the validity of the McCallum projection in many cases for which the original paper does not guarantee its validity. QEPCAD, by contrast, uses Hong's projection by default for all projections beyond level 3, and McCallum's projection for levels 2 and 3. The reduction in the size of the projection factor set resulting from improved projection can have a dramatic effect an both the time and space requirements of CAD construction.

In the lifting phase of CAD construction, QEPCAD B makes some use of validated numerical computation, as described

in [12]. The implementation in QEPCAD B and SACLIB is due to Collins, Johnson and Krandick, the authors of [12].

The solution formula construction phase of QEPCAD B is based on [3, 4, 8]. It offers several different methods for constructing simple solution formulas, each with different ideas of what constitutes "simple". Perhaps most importantly, barring failure of McCallum's projection (and lack of computational resources), QEPCAD B is guaranteed to produce a solution formula, and produce one using simple solution formula construction techniques. This was not true of the original QEPCAD (see [4] for details).

3.2 Quantifier-free formula simplification

Simplifying quantifier-free formulas is a very important problem. Other quantifier elimination algorithms, like the method of virtual term substitution implemented in Redlog [22, 23, 24] or the combinatorial approach described in [14], tend to produce very large quantifier-free equivalent formulas. Moreover, a variety of problems outside of quantifier elimination have solutions that are boolean combinations of polynomial equalities and inequalities, and these solutions do not necessarily come in the simplest possible form. If given a quantifier-free formula as input, QEPCAD B attempts to produce a simple equivalent formula.

The user perspective: If a formula that is already quantifier free is entered into QEPCAD B, the program produces a simple equivalent formula. As an interesting example of formula simplification QEPCAD B, consider the following: Figure 2 shows a triangle ABC with what we might call the "external bisector of vertex B with respect to A". It is

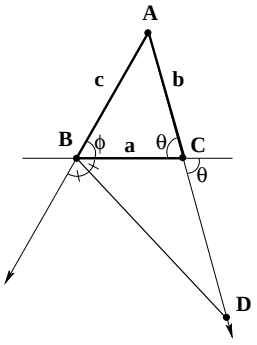

Figure 2: Triangle ABC with the external bisector of vertex B with respect to A.

fairly clear from the diagram that this external bisector exists as drawn if and only if $\theta > (\pi - \phi)/2$. Suppose we want a characterization in terms of the side lengths a, b and c rather than the angles θ and ϕ. Straightforward application of the common trigonometric identities produces the equivalent characterization that, if a, b and c are the side lengths of a non-degenerate triangle, this external bisector exists if and only if

$$b^2 + a^2 - c^2 \leq 0 \vee c(b^2 + a^2 - c^2)^2 < ab^2(2ac - (c^2 + a^2 - b^2)). \quad (1)$$

Of course a, b and c are the side lengths of a non-degenerate triangle if and only if

$$a > 0 \wedge b > 0 \wedge c > 0 \wedge a < b + c \wedge b < a + c \wedge c < a + b. \quad (2)$$

The conjunction of (1) and (2) completely characterizes the triples of real numbers (a, b, c) that are side-lengths of non-degenerate triangles for which the above described external bisector exists. Entering this conjunction as input to QEPCAD B produces the equivalent formula $c > 0 \wedge b > 0 \wedge a - b + c > 0 \wedge a - c < 0 \wedge a + b - c > 0$, which is a little hard to interpret, because it's unclear which conditions simply specify a non-degenerate triangle, and which are specific to characterizing the existence of the external bisector. QEPCAD B allows us to declare "assumptions", which are conditions on the free variables that are "assumed" by the solution formula produced. A true solution formula is actually the conjunction of the assumptions and the solution formula QEPCAD B produces. Here is a QEPCAD B session for this simplification problem:

```
======================================================
Enter an informal description  between '[' and ']':
[ Charaterizing triangles with external bisectors ]
Enter a variable list:
(c,b,a)
Enter the number of free variables:
3
Enter a prenex formula:
[ b^2 + a^2 - c^2 <= 0 \/ c (b^2 + a^2 - c^2)^2 < a b^2 (2 a c - (c^2 + a^2 - b^2)) ].
======================================================

Before Normalization >
assume [ a > 0 /\ b > 0 /\ c > 0 /\ a < b + c /\ b < a + c /\ c < a + b ]

Before Normalization >
finish

An equivalent quantifier-free formula:

a - c < 0

================================================
```

In other words, given the assumption (i.e. the conditions that a, b and c really form a triangle), the input formula is equivalent to $a - c < 0$. Simplification has "discovered" the theorem that the external bisector described above exists if and only if side c is longer than side a.

It is impractical to try to simplify a *very* large formula by a single application of QEPCAD B. For such formulas it is better to simplify pieces of the original formula, put the simplified pieces together and try to simplify that result as described in [7]. A program called SLFQ (Simplifying Large Formulas with QEPCAD B), which applies QEPCAD B over and over again in this manner is available through the QEPCAD B website.

The implementation perspective: The same general technique that Hong used to create simple solution formulas for quantifier elimination problems [15] can be applied to simplifying formulas that are already quantifier free. However, the time requirements of Hong's method are too great to make this practical when the input formula is large — which is precisely when we really need to simplify formulas. QEPCAD B uses CAD simplification [3] before attempting solution formula construction, and the solution formula construction algorithms described in [8], which are typically faster than Hong's algorithm, to speed up solution formula construction on the simplified CAD.

3.3 Plotting of 2-dimensional CADs

QEPCAD B is able to produce eps-file plots of 2-dimensional CADs that are guaranteed to be topologically correct. This part of the program is not very developed in terms of user interface or optimization for speed, but the topological accuracy of the plots QEPCAD B produces may be of interest to people and so it is included. The CAD illustrations in Figure 1 and Figure 4 were produced with QEPCAD B's `p-2d-cad` command.

With the addition of information about adjacencies between cells, a CAD of $\mathbb{R}^2$ provides a complete topological description of the set represented by the CAD (in [1], Arnon describes this for sets of the form $p(x, y) = 0$). Adjacency information can be computed fairly easily by the "ladder" method described in [2], which essentially amounts to repeated integral polynomial real root isolation. QEPCAD B first attempts to deduce adjacencies based on information about multiplicities of polynomials in section cells, falling back on the ladder method when multiplicity information does not suffice. The plots QEPCAD B produces always respect the topological description based on adjacency information.

4 More efficient use of CADs

Quantifier elimination is, in some sense, an inefficient way to use CADs. A CAD representation of a semi-algebraic set contains far more information about the set than is needed for quantifier elimination. By attacking problems directly with CADs, rather than translating them first into the language of quantified Tarski formulas and then using CADs for quantifier elimination, we can make use of this extra information and solve problems more efficiently. One of the major goals in constructing QEPCAD B has been to facilitate this more efficient use of CADs, and the remainder of this section describes several original features of QEPCAD B that address this goal.

4.1 Simplification and quantifier elimination in an extended language

QEPCAD B can both read input and produce output in an extension of the language of Tarski formulas — one which also allows atomic formulas of the form

$$x_i \ \sigma \ \text{root}_k \, f(x_1, \ldots, x_i) \tag{3}$$

where $\sigma \in \{=, \neq, >, <, \geq, \leq\}$, k is a nonzero integer, and f is an integral polynomial of positive degree in x_i. This atomic formula is true at point α if and only if $f(\alpha_1, \ldots, \alpha_{i-1}, x_i)$ is a non-zero polynomial with at least k roots, and α_i holds relation σ to the $|k|$th of those roots, ordered smallest to largest when $k > 0$ and largest to smallest when $k < 0$. The language of Tarski formulas augmented by such atomic formulas is the language of *extended Tarski formulas*. (Note that the variable ordering $x_1, \ldots, x_i$ plays a role in this definition!) From the perspective of CADs, the language of extended Tarski formulas provides a more efficient means for representing semi-algebraic sets than Tarski formulas. For example, as a Tarski formula two polynomials are needed to represent the set $\{\sqrt{2}\}$, e.g. $x^2 - 2 = 0 \wedge x > 0$, while a single polynomial suffices in the language of extended Tarski formulas, e.g. $x = \text{root}_2 \, x^2 - 2$.

We can convert back and forth between the extended Tarski formula representation of a set and the CAD representation of a set more efficiently than between the regular language of Tarski formulas and CADs. [7] described how computing directly with simplified CAD representations of semi-algebraic sets could lead to efficient simplification of very large formulas and efficient quantifier elimination for non-prenex formulas; using extended Tarski formulas as an intermediate representation during computation is essentially equivalent. For details on the language of extended Tarski formulas and computing with them via CADs see [8].

The user perspective: Formula simplification and quantifier elimination using the extended language is not much different than for the usual language. Here is a toy example using the extended language for input: Consider the point traced by a ray from the origin into the first quadrant as it exits the circle with radius 1 centered at $(1, 1)$ (see Figure 3). Can we produce a semi-algebraic description of the distance of this point from the origin as a function of m, the slope of the ray? Substituting $y = mx$ (we assume that $m > 0$)

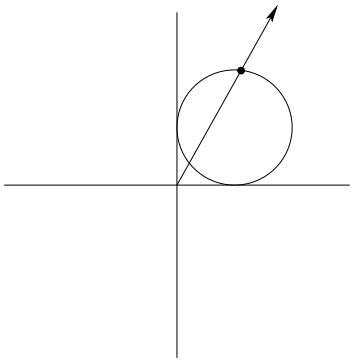

Figure 3: Point traced by a ray from the origin though the first quadrant as it exits the circle with radius 1 centered at $(1, 1)$.

into $(x-1)^2 + (y-1)^2 - 1$, the equation of the circle, we get $(x-1)^2 + (mx-1)^2 - 1$, which has two positive roots. The x-coordinate of the point we want is the larger of these two roots, in other words $x = \text{root}_2(x-1)^2 + (mx-1)^2 - 1$. Similarly, we find that $y = \text{root}_2(y-m)^2 + (my-m)^2 - m^2$. The QEPCAD B session in Figure 4 produces a description of d as a function of m, where $m > 0$, both in the extended language and in the usual language of Tarski formulas. Notice that more polynomials are required to produce the Tarski formula description. Any subsequent computations with QEPCAD B involving this semi-algebraic set will be faster if the extended Tarski formula description is used.

The implementation perspective: A complete description of how CAD can be used to perform quantifier elimination in the extended language may be found in [8]. The basic idea is this: CAD decomposes $\mathbb{R}^n$ into cylinders in

```
======================================================
Enter an informal description  between '[' and ']':
[ Find a semi-algebraic description of d as a funciton of m. ]
Enter a variable list:
(m,d,x,y)
Enter the number of free variables:
2
Enter a prenex formula:
(Ex)(Ey)[
x = _root_2 (x - 1)^2 + (m x - 1)^2 - 1
/\
y = _root_2 (y - m)^2 + (m y - m)^2 - m^2
/\
d^2 = x^2 + y^2
/\
d > 0
].
======================================================
Before Normalization >
assume [ m > 0 ]

Before Normalization >
go

Before Projection (y) >
go

Before Choice >
go

Before Solution >
solution E

An equivalent quantifier-free formula:

d = _root_-1 m^2 d^4 + d^4 - 2 m^2 d^2 - 8 m d^2 - 2 d^2 + m^2 + 1

Before Solution >
solution T

An equivalent quantifier-free formula:

d >= 0 /\ m^2 d^2 + d^2 - 2 m^2 d - 2 m d + m^2 >= 0 /\
m^2 d^2 + d^2 - 2 m d - 2 d + 1 >= 0 /\
m^2 d^4 + d^4 - 2 m^2 d^2 - 8 m d^2 - 2 d^2 + m^2 + 1 = 0
```

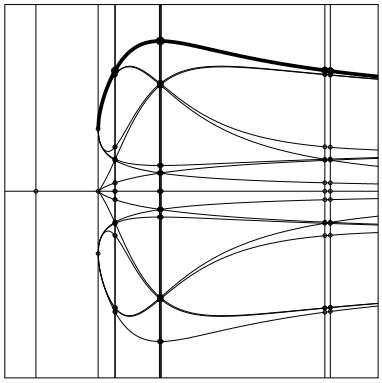

Figure 4: QEPCAD B session and plot of the CAD it produces

which the zero sets of n-level polynomials are *delineable* — meaning they are the graphs of finitely many continuous real-valued functions over the cylinder's base that are non-intersecting. These graphs are called *sections* of the polynomials of which they are zero sets, and because they are non-intersecting, sections are ordered in an obvious way. If polynomial P is zero in section S, we can count the number of sections below S that are sections of P, and if there are k such sections we have determined that S is the $(k+1)$st root of P. Any sector (regions between sections are sectors) or section above S satisfies, for example, $x_n > \text{root}_{k+1}\, P$. Thus, if we can decide the signs of a polynomial p in a stack of a CAD of $\mathbb{R}^n$, we can also decide atomic formulas of the form $x_n\ \sigma\ \text{root}_i\, p(x_1, \ldots, x_n)$, where $\sigma \in \{=, \neq, >, <, \geq, \leq\}$.

[8] shows how to construct simple solution formulas in the extended language from a CAD. Moreover, it shows that a formula representation of any set defined by a CAD can be constructed in the extended language using only the polynomials in that CAD's projection factor set. This is what makes the extended language a better "intermediate format" for CAD-based manipulations of semi-algebraic sets. Converting from a CAD representation to a Tarski formula representation often requires "adding" polynomials to the CAD's projection factor set — polynomials which appear in the resulting formulas. Using the extended language, this is unnecessary.

4.2 Variant quantifiers

QEPCAD B provides several "quantifiers" in addition to $\forall$ and $\exists$ with which the user can pose his problem — quantifiers that the program can handle particularly efficiently using CADs. In all cases these new quantifiers express with one variable what would take several if only $\forall$ and $\exists$ were allowed, and the savings in variables is real, meaning that if k variables appear in the input, QEPCAD B will solve the problem by constructing a CAD of $\mathbb{R}^k$. Since CAD construction is very sensitive to the number of variables, QEPCAD B can solve problems expressed with the new quantifiers much more quickly than if expressed using only $\forall$ and $\exists$. The quantifiers available in QEPCAD B in addition to $\forall$ and $\exists$ are:

- "Exist exactly k": This quantifier means what one might expect, if formula $F(x)$ is quantified with "there exist exactly k x such that", then the resulting formula is true if an only if there exist exactly k distinct real values for x such that $F(x)$ is true. To recast a formula containing a "there exist exactly k" quantifier using just $\forall$ and $\exists$ requires k extra variables. One odd aspect of this quantifier is that subformulas in the unquantified variables cannot always be moved in and out of the quantified block. For example, using $\exists_k$ to denote the quantifier, we see

  $$(\exists_2 b)[a > 2 \vee b^2 - a = 0] \not\Leftrightarrow a > 2 \vee (\exists_2 b)[b^2 - a = 0]$$

  which may be a bit counterintuitive.
- "Exists infinitely many" and "for all but finitely many": Denoted `F` and `G` respectively in QEPCAD B, the meanings of these quantifiers should be clear. Either can be

replaced with $\forall$ and $\exists$ at the cost of two extra variables. For example: `F`$x[H(x)]$ can be rewritten without the new quantifier as $\exists x, y[x < y \wedge \forall z[x < z < y \Longrightarrow H(z)]]$. In addition to requiring fewer variables, these quantifiers can be decided even if values at several points are undetermined. QEPCAD B uses this to substantially improve the time and space requirements of CAD construction.

In fact, CAD-based quantifier elimination can handle "quantifiers" corresponding to any arrangement of points and intervals along a line, all equally well. For example, QEPCAD B also has a "for a connected subset" quantifier. A future implementation could easily provide facilities for user-defined "quantifiers".

The user perspective: The goal in using new "quantifiers" as a way of signalling QEPCAD B that certain optimizations can be made is to keep the user's perspective of the program pretty much the same. The "Exists exactly k" quantifier is given as `X` followed by the value for k. So, for example, the following QEPCAD B session determines when a monic quadratic has a unique positive root.

```
======================================================
Enter an informal description  between '[' and ']':
[When is the positive root of a monic quadratic unique?]
Enter a variable list:
(a,b,x)
Enter the number of free variables:
2
Enter a prenex formula:
(X1 x)[ x^2 + a x + b = 0 /\ x > 0 ].
======================================================

Before Normalization >
finish

An equivalent quantifier-free formula:

4 b - a^2 <= 0 /\ [ b < 0 \/ [ a < 0 /\ 4 b - a^2 >= 0 ]
\/ [ a < 0 /\ b <= 0 ] ]

======================================================
```

Using QEPCAD B's interactive solution formula construction commands, it's possible to find a somewhat nicer solution formula: $a < 0 \wedge 4b - a^2 = 0 \vee b <= 0 \wedge a < 0 \vee b < 0$.

While "there exist exactly k" is useful for phrasing certain problems with as few variables as possible, the "exists infinitely many" and "for all but finitely many" quantifiers are probably useful more for speeding up CAD construction in situations where one would normally use $\exists$ and $\forall$. They may be viewed as allowing limited error in QEPCAD B. In particular, the evaluation of F$x\mathcal{H}(x)$ (or G$x\mathcal{H}(x)$) by CAD is correct even if finitely many points in $\mathbb{R}$ are miscategorized with respect to $\mathcal{H}(x)$. Extending this to the multivariate case, the "exists infinitely many" and "for all but finitely many" quantifiers can be eliminated correctly from F$x\mathcal{H}(t_1, \ldots, t_k, x)$ (or G$x\mathcal{H}(t_1, \ldots, x)$) even if some cells in the CAD representation of $\mathcal{H}(t_1, \ldots, x)$ have the wrong (or unknown) truth values, as long as these cells have measure zero in their stack. This means that we can ignore certain cells when the "exists infinitely many" and "for all but finitely many" quantifiers are used, and the ones we ignore are the ones for which truth value determination requires the most work.

This same idea can be carried over into free variable space — i.e. the user may allow QEPCAD B to miscategorize some points in free variable space, so long as the set of miscategorized points is a measure zero subset of free variable space. Indeed, QEPCAD B has a command `measure-zero-error` with which the user allows it the freedom to miscategorize a measure zero subset of free variable space in the formula it produces, in return for which QEPCAD B can dramatically reduce the time and space requirements of the computation. (Mathematica's implementation of CAD offers essentially the same thing as "generic CAD".) The following example illustrates this command and the error that is allowed in the output formula.

```
Enter an informal description  between '[' and ']':
[ Allowing limited error in the output formula. ]
Enter a variable list:
(a,b,c,x)
Enter the number of free variables:
3
Enter a prenex formula:
(E x)[ a x^2 + b x + c = 0 ].
======================================================

Before Normalization >
measure-zero-error

Before Normalization >
finish

An equivalent quantifier-free formula:

4 a c - b^2 <= 0

============================================================
```

Here QEPCAD B is told to characterize the polynomials of the form $ax^2 + bx + c$ that have at least one real root. The formula it returns, however, miscategorizes the points $a = b = 0 \wedge c \neq 0$.

The implementation perspective: First we consider what kind of "quantifier" we can effectively decide using CAD in the one-dimensional case. Given a formula $F(x)$, we construct a CAD that decomposes $\mathbb{R}^1$ into finitely many points and open intervals, each marked true or false according to whether or not F is satisfied in that cell. One may "quantify" x with any finite description of the topology of a subset of the line, and simply check the decomposition to see whether the points and intervals for which $F(x)$ holds satisfy that topology. For example, if we quantify x with "there exist infinitely many" we need to check our CAD of $\mathbb{R}^1$ to see if any of the open intervals are marked true, since only then are there infinitely many points satisfying x. If, on the other hand, we quantify x with "there exist exactly k", we need to check that there are no open intervals marked true and that there are exactly k single-point cells marked true.

Given a formula $G(x_1, \ldots, x_k)$, we construct a CAD that decomposes $\mathbb{R}^k$ into finitely many cells, which are "stacked" above regions in $\mathbb{R}^{k-1}$. If A is a region in $\mathbb{R}^{k-1}$ above which cells of the CAD are stacked, all lines $\alpha \times \mathbb{R}$, where $\alpha \in A$, are decomposed by their intersections with the cells in the stack in topologically the same way. So analyzing the entire stack can be affected by analyzing a single line $\alpha \times \mathbb{R}$ — which is exactly the information contained in a CAD datastructure. Thus, applying quantifiers in the multi-dimensional case really boils down to applying quantifiers in the one-dimensional case.

The quantifiers "exist infinitely many" and "for all but finitely many" can be decided particularly efficiently because we can ignore the truth values of section cells in stacks (single points in the one-dimensional case). This means that

if x_k is quantified with either of these quantifiers there is no need to lift over k-level section cells, and lifting over section cells is what increases the algebraic degrees of sample point coordinates. This is essentially the observation underlying [16]. Moreover, McCallum's projection can be improved when projecting $(k+1)$-level polynomials if we're not going to lift over any k-level section cells, as pointed out in [20]. Both of these observations carry into free-variable space with the `measure-zero-error` command. In fact, if the `measure-zero-error` command is used and "exist infinitely many" and "for all but finitely many" are used for all but the highest level variable, all computation with algebraic numbers is avoided, which can result in a huge reduction of QEPCAD B's time and space requirements.

5 Conclusion

This paper has provided a brief description of the functionality of the QEPCAD B system, and given examples of how this functionality allows the user to solve problems. While QEPCAD B's implementation of quantifier elimination by CAD improves upon that of QEPCAD, the system it extends, much of what's unique to the program are facilities that allow the user to apply CAD directly to a problem, rather than through the intermediate language of quantified Tarski formulas. This can result in CAD construction problems in fewer variables, fewer polynomials and requiring fewer algebraic number computations — all of which substantially reduces time and space requirements. Additionally, QEPCAD B is very effective at simplifying quantifier free formulas, which is an important problem in its own right.

Performance is an issue that has not been discussed in this paper. Practically speaking, QEPCAD B is limited in the size of problem it can solve in a reasonable amount of time and space, but there is no good characterization of what kinds of problems it solves quickly in practice. Experimental comparisons with other programs and methods of quantifier elimination or formula simplification would comprise another paper entirely.

References

[1] ARNON, D. S. Topologically reliable display of algebraic curves. In *Proceedings of SIGGRAPH* (1983), pp. 219–227.

[2] ARNON, D. S., COLLINS, G. E., AND MCCALLUM, S. Cylindrical algebraic decomposition II: An adjacency algorithm for the plane. *SIAM Journal on Computing 13*, 4 (1984), 878–889.

[3] BROWN, C. W. Simplification of truth-invariant cylindrical algebraic decompositions. In *Proc. International Symposium on Symbolic and Algebraic Computation* (1998), pp. 295–301.

[4] BROWN, C. W. Guaranteed solution formula construction. In *Proc. International Symposium on Symbolic and Algebraic Computation* (1999), pp. 137–144.

[5] BROWN, C. W. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation 32*, 5 (November 2001), 447–465.

[6] BROWN, C. W. The McCallum projection, lifting, and order-invariance. See http://www.cs.usna.edu/~wcbrown/research/techreports.html, September 2001.

[7] BROWN, C. W. Simple CAD construction and its applications. *Journal of Symbolic Computation 31*, 5 (May 2001), 521–547.

[8] BROWN, C. W. *Solution Formula Construction for Truth Invariant CAD's*. PhD thesis, University of Delaware, 99.

[9] CAVINESS, B., AND JOHNSON, J. R., Eds. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Texts and Monographs in Symbolic Computation. Springer-Verlag, 1998.

[10] COLLINS, G. E. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Lecture Notes In Computer Science* (1975), vol. Vol. 33, Springer-Verlag, Berlin, pp. 134–183. Reprinted in [9].

[11] COLLINS, G. E., AND HONG, H. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation 12*, 3 (Sep 1991), 299–328.

[12] COLLINS, G. E., JOHNSON, J. R., AND KRANDICK, W. Interval arithmetic in CAD computation. to appear in the Journal of Symbolic Computation.

[13] DOLZMANN, A., AND STURM, T. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bulletin 31*, 2 (June 1997), 2–9.

[14] GONZALEZ-VEGA, L. A combinatorial algorithm solving some quantifier elimination problems. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, B. Caviness and J. Johnson, Eds., Texts and Monographs in Symbolic Computation. Springer-Verlag, Vienna, 1998.

[15] HONG, H. Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In *Proc. International Symposium on Symbolic and Algebraic Computation* (1992), pp. 177–188.

[16] MCCALLUM, S. Solving polynomial strict inequalities using cylindrical algebraic decomposition. *The Computer Journal 36*, 5 (1993), 432–438.

[17] MCCALLUM, S. An improved projection operator for cylindrical algebraic decomposition. In *Quantifier Elimination and Cylindrical Algebraic Decomposition* (1998), B. Caviness and J. Johnson, Eds., Texts and Monographs in Symbolic Computation, Springer-Verlag, Vienna.

[18] MISHRA, B. *Algorithmic Algebra*. Springer-Verlag New York, Inc., 1993.

[19] RENEGAR, J. On the computational complexity and geometry of the first-order theory of the reals, parts I-III. *Journal of Symbolic Computation 13* (1992), 255–352.

[20] Strzebonski, A. Solving systems of strict polynomial inequalities. *Journal of Symbolic Computation 29* (2000), 471–480.

[21] Tarski, A. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, Berkeley, 1951. second ed., rev. Reprinted in [9].

[22] Weispfenning, V. The complexity of linear problems in fields. *Journal of Symbolic Computation 5* (1988), 3–27.

[23] Weispfenning, V. Quantifier elimination for real algebra — the cubic case. In *Proc. International Symposium on Symbolic and Algebraic Computation* (1994), pp. 258–263.

[24] Weispfenning, V. Quantifier elimination for real algebra — the quadratic case and beyond. *AAECC 8* (1997), 85–101.