



Simple CAD Construction and its Applications*

CHRISTOPHER W. BROWN[†]

Department of Computer Science, United States Naval Academy, U.S.A.

This paper presents a method for the simplification of truth-invariant cylindrical algebraic decompositions (CADs). Examples are given that demonstrate the usefulness of the method in speeding up the solution formula construction phase of the CAD-based quantifier elimination algorithm. Applications of the method to the construction of truth-invariant CADs for very large quantifier-free formulas and quantifier elimination of non-prenex formulas are also discussed.

© 2001 Academic Press

1. Introduction

The method of quantifier elimination by cylindrical algebraic decomposition (CAD) takes a formula from the elementary theory of real closed fields as input, and constructs a CAD of the space of unquantified variables. This decomposition is *truth invariant* with respect to the input formula, meaning that the formula is either identically true or identically false in each cell of the decomposition. The method determines the truth of the input formula for each cell of the CAD, and then uses the CAD to construct a solution formula—a quantifier-free formula that is equivalent to the input formula (see Collins, 1975; Collins and Hong, 1991; Hong, 1992). Simple equivalent formulas can be constructed from a truth-invariant CAD (see Hong, 1992; Brown and Collins, 1996), which motivates the consideration of both quantified and unquantified input formulas. There are other uses for this truth-invariant CAD as well, such as determining the topology of the input formula's solution space or producing a visualization of the solution space.

Often there will exist a “simpler” truth-invariant CAD than the one produced by the method.

DEFINITION 1. CAD B is *simpler* than CAD A if A is a proper refinement of B , i.e. each cell in B is the union of some cells of A , and A and B are not equal.

Subsequent computations requiring a CAD that is truth invariant with respect to the input formula may benefit from using a simpler CAD. In this paper a method is presented for simplifying truth-invariant CADs. The method has been implemented and integrated into QEPCAD, an implementation of quantifier elimination by CAD, based on the ideas found in Hong (1990), Collins and Hong (1991), Hong (1992). Examples are presented which demonstrate the performance of the method, the amount of simplification it can

*This article is a revised and expanded version of Brown (1998), presented at the 1998 International Symposium on Symbolic and Algebraic Computation.

[†]E-mail: wcbrown@usna.edu

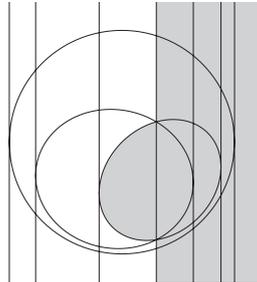


Figure 1. A CAD which could be simpler.

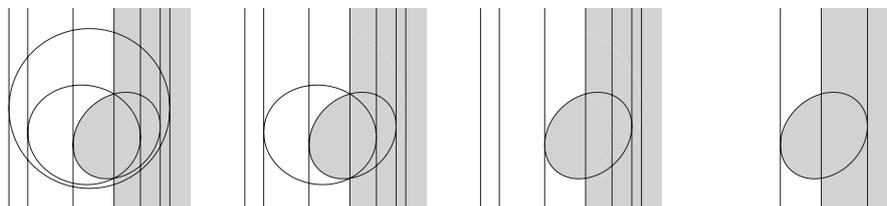


Figure 2. Simplifying a CAD by progressively removing projection factors.

achieve, and the effects of its use on solution formula construction. Two new algorithms are given, both of which make substantial use of CAD simplification, as further application of the method. First, a method for constructing truth-invariant CADs for very large quantifier-free formulas is discussed and applied in an example. Then a method for performing quantifier elimination on non-prenex formulas efficiently is described.

2. A Motivating Example

The following example is intended to illustrate the idea of a simpler truth-invariant CAD and show how simplification might be accomplished. The quantified formula

$$(\exists z) \left[\begin{array}{l} 19z - 10x + 10y < 0 \wedge \\ [x^2 + y^2 + (z - 3)^2 < 9 \vee 2x + 19z + 10y \geq 11] \end{array} \right]$$

defines a sphere and two half-spaces and asks for the (x, y) pairs which are projections of points in either the intersection of the first half-space and the sphere or the intersection of the two half-spaces. Figure 1 shows the CAD produced for this problem. The shaded region is the solution space, i.e. the (x, y) pairs which satisfy the quantified formula.

Clearly the circle can be removed without destroying the truth-invariance property. In fact the left ellipse can be removed as well, and even all tangents to the circle and the left ellipse, and a truth-invariant CAD will still remain—a *simpler* truth-invariant CAD than the original (see Figure 2). This is an example of one way in which CADs can be simplified: by removing projection factors from the projection factor set, and removing sections of these polynomials from the CAD. The simplification algorithm presented in the following sections works in exactly this way. It determines that certain polynomials may be removed without destroying the property of truth invariance, and removes sections of those polynomials from the decomposition.

3. The General Idea

Suppose that x_1, \dots, x_k are the free variables of the input formula and that a CAD has been constructed according to this variable ordering. A polynomial p is said to be an i -level polynomial if it has positive degree in x_i , and degree zero in x_{i+1}, \dots, x_k . In this section we only consider removing k -level polynomials from the projection factor set. Applied to the earlier example, this corresponds to removing the circle and the left ellipse but not their vertical tangents, as in the third CAD in Figure 2.

Clearly, any k -level polynomial can be removed without destroying the truth invariance of the decomposition if none of its sections form the boundary between a true and a false region in k -space. It is this criterion that makes it obvious that the circle can be removed from Figure 1, and this observation provides the nucleus of the method of CAD simplification presented in this paper.

Since only k -level projection factors are being removed, all the boundaries between different stacks will remain in the simpler CAD regardless of which polynomials are removed. Therefore, only boundaries between true and false regions within the same stack need to be considered. These boundaries must be section cells, and if c is a section cell, we will call it a *truth-boundary cell* if c and its two stack neighbors do not all have the same truth value. A k -level section cell is by definition the zero set of one or more k -level projection factors. As long as one such projection factor is retained in the projection factor set, the section cell will remain in the simpler CAD. Thus, each truth-boundary cell gives a condition on the set of k -level projection factors that are kept in the projection factor set. If P_k is the set of k -level projection factors, and Q is a subset of P_k , the polynomials in Q may be removed from the projection factor set without destroying the property of truth invariance if and only if for each truth-boundary cell c there is a polynomial in $P_k - Q$ which is zero in c . This condition can be used to construct a Q such that $P_k - Q$ is minimal, in the sense that adding any other k -level projection factor to Q would leave a truth-boundary cell in which none of $P_k - Q$'s polynomials are zero. Polynomials are chosen from P_k one at a time and it is determined whether adding the chosen element to Q would leave a boundary cell in which no polynomial in $P_k - Q$ is zero. If not, the chosen polynomial is added to Q . In this way, a minimal $P_k - Q$ can be constructed in $O(N \cdot |P_k|)$ time, where N is the number of boundary cells (assuming that the boundary cells have already been determined).

Given a truth-invariant CAD D with projection factor set P , the set Q can be constructed, its elements removed from P , and sections of those elements removed from D . Because Q is minimal in the above sense, the resulting CAD cannot be simplified any further by removing k -level projection factors.

4. Level $k - 1$ and Below

We now consider removing projection factors from levels lower than k . How we proceed to do this depends on what kinds of simplification we want to allow.

4.1. SIGN-INVARIANT CADS

In Collins' original algorithm for quantifier elimination by CAD (Collins, 1975), a CAD of free variable space is produced that is *sign-invariant* with respect to the projection factor set.

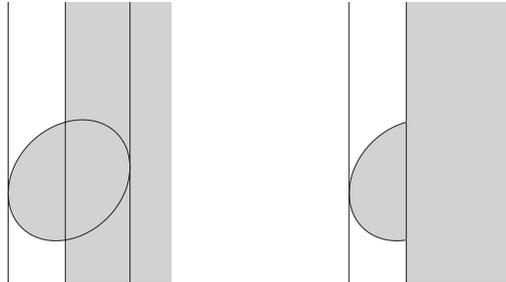


Figure 3. The left CAD is sign invariant, the right is not.

DEFINITION 2. A CAD is sign-invariant with respect to its projection factor set if, for any cell in the CAD and any projection factor, the sign of the projection factor is the same at every point in the cell.

The partial CAD introduced by Hong has a slightly weaker condition that we will call *level sign invariance*. It says that for any cell c there is a level j such that the projection factors of level less than or equal to j have invariant sign in c , and if the point $(\alpha_1, \alpha_2, \dots, \alpha_r)$ is in c then any points whose first j coordinates are $(\alpha_1, \alpha_2, \dots, \alpha_j)$ is in c . Figure 3 shows two CADs of 2-space which are truth invariant with respect to the quantified formula from Section 2. The decomposition on the left is sign-invariant, that on the right is not. The polynomial defining the ellipse does not have invariant sign within the right-most cell of the right-hand CAD, nor does the 1-level projection factor defining the ellipse's tangents.

Which projection factors are removed and which are kept may depend on whether or not it is desirable to retain either of these two sign-invariance properties. In solution formula construction, sign invariance is useful because a sign-invariant CAD contains enough information to decide whether or not a formula constructed from the projection factors is a solution formula. With that motivation, we devote the following two sections of this paper to a method for simplification that retains sign-invariance with respect to the reduced projection factor set. Modification of the method to deal correctly with partial CADs (i.e. to retain level sign invariance) is discussed in Section 6.

Sign invariance will be preserved by requiring that the reduced projection factor set be closed under the projection operator. This requirement is easy to satisfy, because the reduced projection factor set is a subset of the original projection factor set, and the projection of all those polynomials has already been computed. So ensuring that the reduced projection factor set is closed under the projection operator just requires some bookkeeping. As projection factors of level less than k are removed, the two properties of truth-invariance and closure under the projection operator will be retained in the resulting CAD.

4.2. SIMPLIFYING WHILE RETAINING SIGN-INVARIANCE

Suppose that D is a truth-invariant CAD with projection factor set P , and that \overline{D} is a simpler truth-invariant CAD with projection factor set $P_1 \cup \dots \cup P_i \cup \overline{P}_{i+1} \cup \dots \cup \overline{P}_k$. We will consider the problem of simplifying \overline{D} by removing i -level projection factors.

Given the above discussion it is clear that, if S is the set of i -level polynomials in the closure under the projection operator of $\overline{P}_{i+1} \cup \dots \cup \overline{P}_k$, S must be retained in the

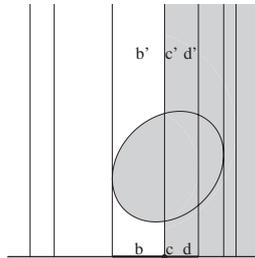


Figure 4. An example of a truth-boundary of lower level.

projection factor set. The set $P_1 \cup \dots \cup P_{i-1} \cup S \cup \overline{P}_{i+1} \cup \dots \cup \overline{P}_k$, defines a sign-invariant CAD, but not necessarily one that is truth invariant. It may be that other polynomials from P_k have to be kept in order to ensure truth invariance (see last two decompositions in Figure 2). Which polynomials to keep can be decided in a way analogous to the way it was decided for level k : if Q is the set of polynomials to be removed from P_i , then for each truth-boundary cell c there must be a polynomial in $P_i - Q$ which is zero in c . However, the truth-boundary cell cannot be defined as it was for level k . In the k -level case, truth-boundary cells are sections which form a boundary between true and false regions within a stack. Since the CAD is a decomposition of k -space, such cells must be defined by k -level polynomials. When considering the removal of polynomials of level i , section cells in the induced CAD of i -space are considered, since they are defined by i -level polynomials. However, a cell in the induced CAD of i -space does not in general have a truth value. Instead, the cylinder above it is partitioned into truth-invariant regions. So the definition of “truth-boundary cell” must be extended to one that makes sense for levels less than k .

DEFINITION 3. An i -level cell c is said to be *over* a j -level cell d if $j \leq i$ and the projection onto j -space of c is d . The stack *over* d consists of all $(j + 1)$ -level cells over d .

Consider an i -level section cell c . Suppose that all polynomials of level greater than i are delineable over the union of c and its two stack neighbors, call them b and d . In this case there is a natural correspondence between k -level cells over b , c , and d . If the cells b , c , and d are merged into one cell, each k -level cell over $b \cup c \cup d$ is the union of three corresponding cells over a , b , and c . If there are three corresponding cells which do not all have the same truth value then their union is not a truth-invariant region, and c defines a boundary between true and false regions in the CAD.

DEFINITION 4. Cell c is a truth-boundary cell if there exists some triple of corresponding cells over b , c , and d which do not all have the same truth value.

Figure 4 shows a section cell c of level 1 and its two stack neighbors b and d from the now familiar example of Section 2. Cell c is a truth-boundary cell because the cells b' , c' , and d' are corresponding cells which do not all have the same truth value. Were cells b , c , and d to be merged there would be cells in the resulting stack over $b \cup c \cup d$ that would not be truth invariant— $b' \cup c' \cup d'$, for example.

A sufficient condition for the delineability of the polynomials of level greater than i (i.e. $\overline{P}_{i+1}, \dots, \overline{P}_k$) over $b \cup c \cup d$ is that no i -level polynomial that is zero in c is in the

projection of $\overline{P}_{i+1}, \dots, \overline{P}_k$. So if S is, as above, the set of i -level projection factors in the projection of $\overline{P}_{i+1}, \dots, \overline{P}_k$, the set of truth-boundary cells are chosen from the set of i -level section cells that are not sections of polynomials in S . Some truth-boundary cells may be missed this way, but only cells that are sections of some polynomial in S , and S is going to be retained in the projection factor set. Just as in the k -level case, each truth-boundary cell gives a condition on the set of polynomials to be kept. For truth-boundary cell c , let l_c be the set of i -level projection factors which are zero in c . The set of i -level projection factors to be retained must have non-empty intersection with l_c for every truth-boundary cell c . Such a set is called a *hitting set* for the set of all l_c 's. The set $P_k - Q$ constructed in the previous section was a minimal hitting set, and the i -level minimal hitting set problem can be solved the same way. Let S' be such a minimal hitting set for the set of all l_c 's. Any i -level projection factor in neither S nor S' may be removed from P_i without destroying the property of truth-invariance or of sign-invariance in the resulting CAD. All truth-boundary cells remain, since for each truth-boundary cell c an element of either S or S' must be zero in c .

5. An Algorithm

The algorithm SIMPLECAD, which simplifies a truth-invariant CAD, is presented in this section. In addition, implementation issues are discussed, and an analysis of SIMPLECAD's computational complexity given.

5.1. ALGORITHM DESCRIPTION

Suppose P is the initial set of projection factors and D is the sign-invariant CAD constructed from P . SIMPLECAD constructs a sign-invariant CAD \overline{D} with projection factor set \overline{P} , such that \overline{D} is a simpler truth-invariant CAD than D .

\overline{D} and \overline{P} are constructed iteratively, a level at a time, starting from level k and working down to level 1. At the beginning of the iteration corresponding to level i , \overline{D} is the sign-invariant CAD defined by $P_1 \cup \dots \cup P_i \cup \overline{P}_{i+1} \dots \cup \overline{P}_k$. At each iteration this CAD retains the property of truth invariance with respect to the input formula.

Algorithm SIMPLECAD.

Inputs: Projection factor set P and CAD D defined by P . D is sign-invariant with respect to P and truth invariant with respect to the input formula.

Outputs: \overline{P} , a subset (if possible a proper subset) of P that is closed under the projection operator, and \overline{D} , a CAD that is sign-invariant with respect to \overline{P} and truth invariant with respect to the input formula.

- (1) Set $\overline{D} = D$.
- (2) For i from k down to 1 do
 - (a) Construct S , the set of i -level projection factors in the closure under the projection operator of $\overline{P}_{i+1} \dots \cup \overline{P}_k$.
 - (b) Construct C , the set of all i -level truth-boundary cells in \overline{D} which are not sections of any elements of S .
 - (c) Set L equal to $\{l_c | c \in C\}$, where l_c is the set of all elements of P_i which are zero in cell c .

- (d) Set S' to a minimal hitting set for L . (S' is a subset of P_i .)
 (e) Set $\overline{P}_i = S \cup S'$ and modify \overline{D} for the next iteration.
- (3) Set $\overline{P} = \bigcup_{i=1}^k \overline{P}_i$.

5.2. IMPLEMENTATION ISSUES

The complexity of SIMPLECAD cannot be examined without some kind of information about the data structures defining CADs and projection factors. Therefore, some implementation issues have to be addressed before attempting any kind of complexity analysis.

Since our implementation is built into QEPCAD, assumptions about CAD and projection factor data structures will be based on those structures in QEPCAD. In particular:

cells Given a cell c , a list of the cells in the stack over c (ordered bottom to top) can be retrieved in $O(1)$ time. These cells are c 's children.

truth value The truth value of a cell can be determined from its data structure in $O(1)$ time.

sign information For an i -level cell c , a list of the signs of the i -level projection factors in c can be retrieved in $O(1)$ time.

projection Given a projection factor data structure p , a list of all *derivations* of p can be retrieved in $O(1)$ time. A derivation of a projection factor describes where it came from; was it a factor of some polynomial appearing in the input formula, or a factor of a discriminant of some projection factor, a factor of a coefficient of some projection factor, or a factor of the resultant of some pair of projection factors? (Assuming the McCallum projection operator (McCallum, 1998), these are the possibilities.)

One implementation issue is the representation of the simpler CAD. In our implementation, each cell of the simpler CAD is represented as a data structure with two fields. The first is a list of the cell's children, the second a pointer to a "representative cell" in the original CAD. The representative cell is one of the group of cells from the original CAD which comprise the represented cell from the simpler CAD. Information about the signs of projection factors, truth value, or sample points can all be read off from the representative cell. Thus, the simpler CAD requires very little space.

Another issue concerns minimal hitting set problems. It may be desirable to have hitting sets be as small as possible, as that corresponds to a fairly intuitive notion of "as simple as possible" for the resulting CAD. For example, if the truth-invariant CAD is to be used for solution formula construction, then few projection factors in the truth-invariant CAD may correspond to a formula containing few polynomials. So one might want to ask for hitting sets which are of *minimum* cardinality rather than just *minimal*. The minimum hitting set problem, it turns out, is NP-Hard (Garey and Johnson, 1979). While the problem instances created by SIMPLECAD may not also be NP-Hard, minimum hitting set algorithms could have time complexity exponential in P_i . In practice, however, P_i has moderate size, and most truth-boundary cells are sections of few of the elements of P_i . In fact, often there will be truth-boundary cells which are sections of exactly one i -level projection factor. A polynomial which is zero in such a cell *must* be included in the reduced set of i -level projection factors, so this allows one to simplify the minimum hitting set problem. In practice, it is usually not difficult computationally

to find a hitting set of minimum cardinality, so our implementation of SIMPLECAD constructs minimum hitting sets. For complexity analysis, however, it will be assumed that minimal hitting sets are constructed via a method similar to that outlined for the k -level case.

5.3. COMPLEXITY ANALYSIS

PROPOSITION 1. *Given a CAD with projection factor set P and assuming that the CAD data structure has the operations and complexities given above, the time required for SIMPLECAD is $O((N + kn + |P|^2) \cdot |P|)$, where N is the number of cells in the CAD, and $n = \max_i(n_i)$, where n_i is the maximum degree in x_i of any i -level projection factor.*

Consider the time complexity for each step of the loop in Step 2. During the loop iteration corresponding to level i , step (a) determines which elements of P_i are in the projection of $\overline{P}_{i+1} \cup \dots \cup \overline{P}_k$. For each $p \in P_i$ there is a list of *derivations* of p . To decide whether p is in the projection, potentially each derivation must be examined. Examining a derivation means determining whether the polynomials in the derivation are in $\overline{P}_{i+1} \cup \dots \cup \overline{P}_k$. This can be done in constant time. The number of derivations for P must be less than the total number of possible derivations. There are $|P|$ possible discriminants, $n|P|$ possible coefficients, and $|P|^2$ possible resultants, so the time required for step (a) is $O((n|P| + |P|^2) \cdot |P_i|)$. This bound is, of course, quite pessimistic.

In step (b) the set of all truth-boundary cells is constructed. In determining whether a given i -level section cell is a truth-boundary cell, it may be that the truth values of all descendants of the cell and both of its stack neighbors need to be inspected. Since an i -level sector cell may neighbor two sections in its stack, some cells may be examined twice, but never more. Thus, if N is the number of cells in the CAD, fewer than $2N$ examinations per iteration are made. When a k -level cell is examined, its truth value is determined, which requires time $O(1)$. When a lower level cell is examined it is determined whether it is a section of some element of S . This operation is $O(|P_i|)$, since S is a subset of P_i . Otherwise its child cells are fetched for examination, which is a constant time operation. Thus, the complexity of the step for the i -level iteration is $O(N \cdot |P_i|)$.

Step (c) is $O(N \cdot |P_i|)$, since for each of at most N cells a subset of P_i must be chosen. Since we require only a minimal hitting set, the method outlined in Section 2 can be adapted to construct a minimal $P_i - Q$ to perform step (d) in $O(N \cdot |P_i|)$ time.

In step (e), \overline{D} must be modified to reflect setting \overline{P}_i to $S \cup S'$. Since some i -level projection factors have been removed, some cells in stacks over $(i-1)$ -level cells may need to be merged. Specifically, sections cells in which no element of $S \cup S'$ are zero must be removed. This simply means examining the child list of each $(i-1)$ -level cell for section cells in which no element of $S \cup S'$ is zero. Such a section cell and the following sector are removed from the child list. The cell data structure for the sector preceding such a section represents the union of all three cells. The $(i-1)$ -level cells must be collected, and the signs of the polynomials in $S \cup S'$ must be examined for every section cell in the stack over each $(i-1)$ -level cell. Thus, step (e) requires $O(N \cdot |P_i|)$ time.

Over all iterations, each of steps (b) through (e) require $O(N \cdot |P|)$ time. So together with step (a), the total time requirement is $O((N + kn + |P|^2) \cdot |P|)$.

6. Extension to Partial CADs

The sole problem in extending SIMPLECAD to deal correctly with partial CADs is the identification of truth-boundary cells. Definition 4 states that an i -level section cell c is a truth-boundary cell if there is no triple of corresponding cells over c and its two stack neighbors in which not all three cells have the same truth value. That there is a natural correspondence between k -level cells over c and its two stack neighbors is guaranteed because:

- (1) the projection factors of level greater than i are assumed to be delineable over c , and
- (2) the CAD is assumed to be sign-invariant.

Partial CADs are level-sign-invariant but not sign-invariant, so the second condition fails. However, the definition of truth-boundary cell can be extended so that the algorithm “works” for partial CADs. Suppose D is a partial CAD with projection factor set P . “Works” means that the same simplified projection factor set is constructed by SIMPLECAD for D as would have been constructed for the sign-invariant CAD defined by P . Indeed, this basically provides the extended definition for “truth-boundary cell”.

Once again suppose D is a partial CAD with projection factor set P , and let D' be the sign-invariant CAD defined by P . For any k -level cell c in D , there is a level j such that all projection factors of level j and lower are sign-invariant in c . The projection of c onto j -space is some j -level cell, call it c^* , which must also be a j -level cell in D' .

DEFINITION 5. A j -level cell c^* is a truth-boundary cell if:

- (1) All projection factors of level j or lower are sign-invariant in c^* .
- (2) c^* is a section cell.
- (3) c^* is a truth-boundary cell in D' .

With this extended definition of truth-boundary cells, SIMPLECAD performs identically for sign-invariant and level-sign-invariant CADs.

Deciding whether a given cell satisfies Definition 5 is not quite straightforward. The first two criteria are easily checked, but the third is more difficult, since it is a condition on a cell in D' , which presumably has not even been constructed. The question can, however, be decided without considering cells in the truth-invariant CAD D' .

If b' , c' , and d' are corresponding cells, then they are said to *agree* if

- the stacks over each of a' , b' , and c' consist of cells in which all $(j+1)$ -level projection factors have invariant sign, and each triple of corresponding $(j+1)$ -level cells over b' , c' , and d' agree,
- or
- all k -level cells over a' , b' , and c' have the same truth value.

Let c be a j -level section cell in D such that all projection factors of level greater than j are delineable over c and its stack neighbors, call them b and d . Cells b , c , and d agree if and only if c is not a truth-boundary cell. This characterization involves only cells in D , and provides an easy procedure for deciding whether a cell is a truth-boundary cell.

Table 1. *x*-axis ellipse problem.

Level	Proj. fac.'s			Cells		
	1	2	3	1	2	3
Original CAD	7	9	7	15	105	635
Simple CAD	3	6	3	7	37	103

7. Examples

In this section we present some quantifier elimination problems, look at how SIMPLECAD performs for these examples, and examine the effect of truth-invariant CAD simplification on solution formula construction. It is important to note that it may happen that a solution formula can be constructed from the projection factor set of the original CAD but not the simpler CAD. This situation can be dealt with quite simply (Brown and Collins, 1996) but, as it applies solely to solution formula construction, is outside the scope of this paper. As stated in the introduction, these examples were investigated using a version of QEPCAD extended by an implementation of SIMPLECAD. Computations were performed on a Sun Ultra-2/1170 with 320 MB of memory. Times for garbage collection are not included.

7.1. THE *x*-AXIS ELLIPSE PROBLEM

The *x*-axis ellipse problem, a special case of the general ellipse problem posed by Kahan (1975), is a traditional benchmark example for quantifier elimination (see Hong, 1992; Dolzmann and Sturm, 1997). The problem asks when the ellipse $(x - c)^2/a^2 + y^2/b^2 = 1$ lies in the unit circle. Of course we require a and b to be non-zero, and in fact we are only interested in the case where they are positive. The formula

$$(\forall x)(\forall y) \left[\begin{array}{l} a > 0 \wedge b > 0 \wedge [b^2(x - c)^2 + a^2y^2 - \\ a^2b^2 = 0 \longrightarrow x^2 + y^2 - 1 \leq 0] \end{array} \right]$$

expresses this as a quantifier elimination problem. With the variable ordering $a < b < c < x < y$, QEPCAD produces a truth-invariant CAD for this input in about 22 seconds. Our program uses this CAD to construct a simple CAD in about 34 milliseconds. Table 1 shows the difference in the number of projection factors and in the number of cells in these two truth-invariant CADs.

Using the simplified CAD as input, QEPCAD's solution formula construction program (Hong, 1992) produces a solution formula in 846 milliseconds. Using the original CAD as input, 20.4 seconds is required, and a slightly longer formula is produced.

In Hong (1990, 1992) this problem is phrased somewhat differently. Hong argues that because of symmetry, we need only consider values of c greater than zero. He also points out some obvious necessary conditions and arrives at

$$(\forall x)(\forall y) \left[\begin{array}{l} 0 < a \leq 1 \wedge 0 < b \leq 1 \wedge 0 \leq c < 1 - a \wedge \\ [c - a < x < c + a \wedge b^2(x - c)^2 + \\ a^2y^2 - a^2b^2 = 0] \longrightarrow x^2 + y^2 - 1 \leq 0 \end{array} \right]$$

as his formulation of the *x*-axis ellipse problem. QEPCAD requires 2549 milliseconds to construct a truth-invariant CAD for this input. From this CAD, QEPCAD constructs an equivalent formula in 72 milliseconds. Our algorithm produces a simplified CAD from

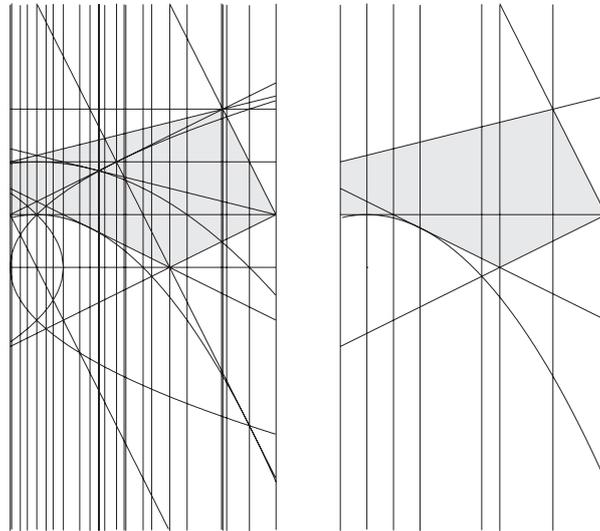


Figure 5. The original CAD and the simple CAD.

the original in 22 milliseconds. When Hong's solution formula construction method is run with this simpler CAD as input, the same formula is returned in 5 milliseconds.

7.2. THE COMPLEX PRODUCT OF AN EDGE AND A SQUARE

Consider the complex segment $L = \{x + iy \mid x \in [0, 2]\}$, and the complex square $S = \{x + iy \mid x \in [2, 4], y \in [-1, 1]\}$. Quantifier elimination can be used to determine the complex product of S and L . Since there are the easily derived necessary conditions that the product lies in the box $[-1, 9] \times i[-6, 6]$, the product can be expressed as all pairs (x, y) satisfying

$$(\exists x_1)(\exists x_2)(\exists y_2) \left[\begin{array}{l} x = x_1x_2 - y_2 \wedge y = x_1y_2 + x_2 \wedge \\ 0 \leq x_1 \leq 2 \wedge 2 \leq x_2 \leq 4 \wedge \\ -1 \leq y_2 \leq 1 \\ \wedge -1 \leq x \leq 9 \wedge -6 \leq y \leq 6 \end{array} \right].$$

These necessary conditions are added to the quantified formula because QEPCAD can use them to speed up its computations. Figure 5 shows two CADs that are truth invariant with respect to this input formula—the shaded regions are those in which the formula is satisfied. The first CAD is the original one produced by the quantifier elimination process. The second is the minimal truth invariant CAD constructed from the original CAD by SIMPLECAD. Table 2 gives a quantitative comparison of the two CADs. Construction of the original CAD was accomplished by QEPCAD in 171 seconds, with the variable ordering $x < y < x_1 < x_2 < y_2$. The simpler truth invariant CAD was constructed in 134 milliseconds. With the original truth-invariant CAD as input, QEPCAD required 10298 seconds to produce a solution formula. Using the simplified CAD as input, a formula was produced in 191 seconds.

One property that both of these examples have in common is that the hitting set problems that arise during SIMPLECAD's computations (there is one problem for each

Table 2. Complex product of an edge and a square.

Level	Proj. fac.'s		Cells	
	1	2	1	2
Original CAD	73	18	143	1795
Simple CAD	14	6	27	179

level) have unique minimal solutions. Our algorithm constructs a simple CAD with two important properties: (1) truth invariance, and (2) a projection factor set which is closed under the projection operator and is a subset of the original projection factor set. If all the hitting set problems during the computation have unique minimal solutions, then the CAD SIMPLECAD produces is the simplest possible with these two properties.

8. Simple CADs for Very Large Quantifier-free Formulas

This section shows how CAD simplification can be used to help construct truth-invariant CADs for very large quantifier-free input formulas—formulas for which the usual methods are impractical. Very large quantifier-free formulas sometimes arise in practice. The quantifier elimination method of Weispfenning (1994, 1997), for example, is able to construct solution formulas for problems for which CAD-based quantifier elimination is impractical, but the formulas it produces are often quite large. This is true even for formulas describing relatively simple semi-algebraic sets.

Simplification of these very large quantifier-free formulas is not easy to achieve (see Dolzmann and Sturm, 1997, for work in this area). However, if we can construct a truth-invariant CAD for a formula, we can use the method of Hong (1992) and CAD simplification to construct a simple equivalent formula. In fact, depending on the application, a truth-invariant CAD might be more useful than any formula, no matter how simple (for computing dimension or plotting, for example). Constructing a truth-invariant CAD directly from a very large formula may be impractical due to the number of polynomials in the formula. But if the formula describes a relatively simple semi-algebraic set, many of those polynomials will be removed by the CAD simplification process, leaving a simplified CAD with a small projection factor set and few cells.

The next section describes a problem that leads to a very large quantifier-free formula which, as we will see, describes a relatively simple semi-algebraic set. The next section outlines an algorithm, Form2SCAD, that uses CAD simplification to help with the construction of a truth-invariant CAD for a very large formula. Finally, Form2SCAD is applied to the example formula to obtain a simple CAD, and even a simple equivalent formula.

8.1. A MORE DIFFICULT COMPLEX PRODUCT PROBLEM

The edge-square complex product problem is a difficult one for the method of quantifier elimination by CAD, but a tractable one. Adding another dimension and considering the complex product of a rectangle and a square results in problems that are too large for QEPCAD. For example, consider the complex product of the square $[1, 2] \times i[1, 2]$ and the rectangle $[0, 1] \times i[-3, -1]$. This can be expressed as the complex numbers $x + iy$

such that

$$\exists x_1 \exists y_1 \exists x_2 \exists y_2 \left[\begin{array}{l} x = x_1 x_2 - y_1 y_2 \wedge y = x_1 y_2 + x_2 y_1 \wedge \\ x_1 \geq 1 \wedge x_1 \leq 2 \wedge y_1 \geq 1 \wedge y_1 \leq 2 \wedge \\ x_2 \geq 0 \wedge x_2 \leq 1 \wedge y_2 \geq -3 \wedge y_2 \leq -1 \end{array} \right].$$

QEPCAD cannot produce a truth-invariant CAD for this formula in a reasonable amount of time and space. Using the McCallum projection, the program computes a projection factor set containing 161 bivariate and 8873 univariate polynomials. With even 12 times QEPCAD's default size for garbage collected space, memory is exhausted before the CAD of 1-space can be constructed.

However, this input formula satisfies the degree requirements of the method of Weispfenning (1994, 1997), which has been implemented in the Redlog system (Dolzmann and Sturm, 1996). Redlog is able to produce a quantifier-free equivalent formula for this input, but the formula is very large. Printed out, it consists of 344,346 ASCII characters. At the top level, it is the disjunction of 131 sub-formulas. It contains 1141 distinct atomic formulas, built from 442 distinct irreducible bivariate and eight distinct irreducible univariate polynomials. QEPCAD cannot construct a truth-invariant CAD from this formula in a reasonable amount of time and space, it is simply too big. Given 140 minutes of CPU time QEPCAD was unable to even complete its projection phase.

The next section describes a method for using CAD simplification to construct a truth-invariant CAD for a very large quantifier-free formula—like the one produced by Redlog for this problem. In Section 8.3, an approximation of the method is applied to Redlog's quantifier-free formula for the rectangle-square product. As will be seen, the resulting CAD is quite small, and the equivalent formula

$$\begin{aligned} x - 1 \geq 0 \wedge y + 3x - 20 \leq 0 \wedge y - x + 2 \leq 0 \wedge 2y + x - 5 \leq 0 \wedge \\ y + 6 \geq 0 \wedge y + 2x \geq 0 \wedge y - x + 12 \geq 0 \end{aligned}$$

is easily obtained from it.

8.2. TRUTH-INVARIANT CADS FROM VERY LARGE FORMULAS

Divide and conquer is one of the most common techniques used in algorithm design and, as this section describes, it can be applied to the construction of truth-invariant CADs from very large formulas. It is assumed that the very large formula given as input describes a relatively simple semi-algebraic set, so one expects that most of the polynomials appearing in the original formula are irrelevant as far as constructing a simple truth-invariant CAD is concerned. If this is indeed the case, then simplifying the original formula's constituent sub-formulas eases the task of constructing a truth-invariant CAD, since it removes many of the irrelevant polynomials.

To make the discussion a bit clearer: for a set of polynomials, A , let us call the sign-invariant CAD that is produced for A by the usual CAD construction process *the* CAD defined by A . (Note that this CAD will depend on the projection operator used.) Denote by $PROJ(A)$ the projection factor set produced for this CAD. The usual method for constructing a truth-invariant CAD for a formula F is to construct the CAD for the set of polynomials appearing in F . For very large formulas, however, this approach (constructing a CAD directly from F) is not always best.

For sets of polynomials A and B , $PROJ(A) \cup PROJ(B)$ is contained in $PROJ(A \cup B)$, and the former is typically much smaller than the later (how much depends on the overlap between A and B , the number of variables involved, and the number of coincidental

common factors that spring up). This means that the time and space required for constructing the CAD defined by A and the CAD defined by B is typically much smaller than what is required to construct the CAD defined by $A \cup B$.

Let $F = F_A \text{ op } F_B$, where op is either \wedge or \vee , be a quantifier-free formula. Let A be the set of polynomials in F_A , and let B be the set of polynomials in F_B . The CAD defined by $A \cup B$ is a truth-invariant CAD for F . Suppose A' and B' are the projection factor sets of simplified truth-invariant CADs for F_A and F_B respectively. Then clearly the CAD defined by $A' \cup B'$ must also be truth invariant with respect to F . If a significant amount of simplification takes place, the CAD defined by $A' \cup B'$ will be substantially smaller than the CAD defined by $A \cup B$, and take a lot less time to construct.

The preceding paragraph points to a method for constructing a truth invariant CAD for F that has the potential to be significantly faster and require a lot less space than constructing a CAD directly from F . A truth-invariant CAD for F can be constructed by:

- (1) constructing CADs for F_A and F_B ,
- (2) simplifying the CADs for F_A and F_B , and
- (3) constructing a CAD for $A' \cup B'$.

Provided that there is not too much overlap between the sets A and B , and provided that a substantial amount of simplification takes place in Step 2, this is much more efficient than constructing a CAD directly from F . However, to simplify the resulting CAD for F , the truth value of F must be determined for each cell of the CAD. This is actually easily accomplished using the simplified CADs for F_A and F_B . The CAD for F is a refinement of the CAD for F_A , and is also a refinement of the CAD for F_B . Each cell of the decomposition for F is contained in a cell from the CAD for F_A as well as a cell from the CAD for F_B . If we make a traversal of the cells in the CAD for F , we are simultaneously making traversals of the cells of the CADs for F_A and F_B . We can then assign truth values to cells in the CAD for F using the truth values of the corresponding cells in the CADs for F_A and F_B .

Algorithm Form2SCADversion1.

Input: a formula $F = F_A \text{ op } F_B$.

Output: a simple truth-invariant CAD for F .

- (1) Construct C_A a truth-invariant CAD for F_A , and simplify it.
- (2) Construct C_B a truth-invariant CAD for F_B , and simplify it.
- (3) Construct C , a sign-invariant CAD for A' and B' , the simplified projection factor sets for C_A and C_B , respectively.
- (4) Using C_A and C_B , assign the cells of C truth values with respect to F .
- (5) Simplify C .

If a simple truth-invariant CAD for F can be constructed more efficiently via the algorithm **Form2SCADversion1(F)** than by direct means, why not apply the same technique in Steps 1 and 2 in order to construct C_A and C_B ? That would be the true divide-and-conquer technique, and that gives the method proposed in this section:

Algorithm Form2SCAD.

Input: a formula $F = F_A \text{ op } F_B$.

Output: a simple truth-invariant CAD for F .

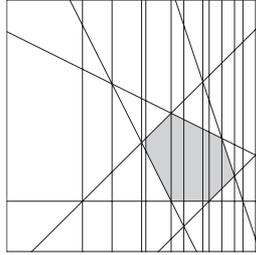


Figure 6. Simplified truth-invariant CAD for the rectangle–square complex product problem.

- (1) If F is an atomic formula, construct by usual means a truth-invariant CAD for F , and return it.
- (2) $C_A = \text{Form2SCAD}(F_A)$, $C_B = \text{Form2SCAD}(F_B)$.
- (3) Construct C , a sign-invariant CAD for A' and B' , the projection factor sets for C_A and C_B , respectively.
- (4) Using C_A and C_B , assign the cells of C truth values with respect to F .
- (5) Simplify C , and return the result.

8.3. CONSTRUCTING THE RECTANGLE–SQUARE PRODUCT

An approximation of **Form2SCAD** was performed on Redlog’s quantifier-free formula for the rectangle–square product problem. The method was only approximated since defining formulas for the simplified CADs were used in Step 4 of the method (the step that assigns truth values to cells) rather than what was described above. This was unavoidable unless QEPCAD was to be heavily modified. Fortunately, there was not too much of a penalty for constructing defining formulas for this example since almost all of the simplified CADs constructed during this process were projection-definable, and since small, simple formulas were not required. In another departure from **Form2SCAD** as outlined above, the divide-and-conquer approach was not used for sub-formulas of fewer than 10 atoms—i.e. in Step 1 a CAD was constructed directly not just for atomic formulas, but for any formula consisting of fewer than 10 atoms. This kind of optimization is common in divide-and-conquer algorithms where, at some point, the overhead of dividing problems and combining solutions outweighs any benefits. The best cutoff point probably depends on polynomial degrees and the number of variables involved.

Basically, the method was carried out as a script, parsing formulas, creating input files for QEPCAD, launching QEPCAD (3069 times for this example!), and parsing the resulting output files. An implementation that was integrated with QEPCAD would presumably be somewhat faster. As it was, the truth invariant CAD in Figure 6 was constructed from Redlog’s output formula in about 37 minutes. The formula

$$\begin{aligned} x - 1 \geq 0 \wedge y + 3x - 20 \leq 0 \wedge y - x + 2 \leq 0 \wedge 2y + x - 5 \leq 0 \wedge \\ y + 6 \geq 0 \wedge y + 2x \geq 0 \wedge y - x + 12 \geq 0 \end{aligned}$$

is easily obtained from the simple truth-invariant CAD, either by inspection or automated means.

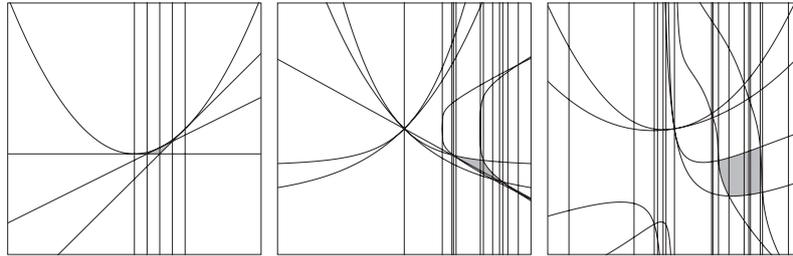


Figure 7. Simple CADs defined by sub-formulas from Redlog's output.

8.4. COMMENTARY ON FORM2SCAD

Redlog's formula for the rectangle-square product problem is quite complicated in that it describes the simple region from figure 6 as the union of some very complicated regions. (Figure 7 gives examples of some of these.) Taken to even further extremes, this property could cause problems for Form2SCAD.

The rectangle-square product problem also has just two variables. Formulas in more variables are, obviously, more likely to prove too difficult for Form2SCAD. The fact that a problem as large as this one could be completed in a reasonable amount of time indicates that Form2SCAD can be effective for very large formulas in two variables. In even just three variables, however, the presence of very large extraneous polynomials, or of sub-formulas describing complex regions, could render the method inapplicable in practice.

9. Non-prenex Input Formulas

The CAD-based method of quantifier elimination assumes an input formula in prenex form, which is not natural for some applications. Any quantified formula can be put into prenex form, but possibly at the cost of additional variables. For example, the formula $\exists x[P(x)] \wedge \exists x[Q(x)]$, where P and Q are quantifier-free formulas, can only be put into prenex form by introducing another variable, call it y , which yields $\exists x\exists y[P(x) \wedge Q(y)]$. Since quantifier elimination by CAD is doubly exponential in the number of variables in the input formula, introducing new variables is not attractive[†]. This section describes how CAD simplification and the methods of the previous section can be used to deal efficiently with non-prenex input formulas.

9.1. THE OBVIOUS APPROACH AND ITS FLAWS

Any method for quantifier elimination that is able to deal with prenex input formulas is automatically able to deal with non-prenex formulas without adding any variables. We simply replace any innermost quantified subformula, which is clearly prenex, with

[†]In fact, the situation is a little bit complicated. While it is true that quantifier elimination by CAD is doubly-exponential in the number of variables, it does not follow that the algorithm is doubly exponential in n for an n -variable prenex formula representing a k -variable non-prenex formula, where $k < n$. Projection will typically be easier with such formulas than with arbitrary n -variable formulas because many polynomials will be constant in many of the variables. It is clear, for example, that no polynomial will contain all n -variables. However, stack construction will still be quite expensive, since the number of cells will still grow at least exponentially in n , and since stack construction will still require n -level towers of algebraic extensions.

the result of performing quantifier elimination on that subformula. This is repeated until all quantifiers have been eliminated. Considering once again the non-prenex formula $\exists x[P(x)] \wedge \exists x[Q(x)]$, we perform quantifier elimination to compute a quantifier-free equivalent to $\exists x[P(x)]$, call it F , and eliminate the quantifiers from $\exists x[Q(x)]$ to obtain a quantifier-free equivalent formula, call it G . Finally, the two subformulas from the original problem are replaced with their quantifier-free equivalents yielding $F \wedge G$. For the CAD-based method of quantifier elimination, this approach, which we will call *the obvious approach*, will typically be considerably more efficient than performing quantifier-elimination on $\exists x\exists y[P(x) \wedge Q(y)]$.

This obvious approach has flaws, however. In most cases simply returning $F \wedge G$ is insufficient, and a CAD representation of the set defined by $F \wedge G$ has to be constructed—first of all because $F \wedge G$ may not be a simple formula for the set it defines (it may even define the empty set), and secondly because the problem may be embedded in a larger quantifier elimination problem. The question then becomes as follows: Is it more efficient to construct F from $\exists x[P(x)]$, G from $\exists x[Q(x)]$, and a CAD from $F \wedge G$, or to construct a CAD directly from $\exists x\exists y[P(x) \wedge Q(y)]$? In essence that comes down to the question: Which is smaller, A —the set of projection factors used in computing F , G , and CAD representation of $F \wedge G$, or B —the projection factor set produced from $\exists x\exists y[P(x) \wedge Q(y)]$? As long as A is not “larger” than B , then the obvious approach is clearly more efficient. Otherwise it becomes more difficult to compare the two methods. Unfortunately, we have the following possibilities: $A \subset B$, $B \subset A$, $A = B$, and A and B incomparable.

The only projection factors in x or y are those directly from the input formula, and they appear in both A and B (though possibly with different variable names), so we may restrict our attention to projection factors in the free variables only.

The obvious approach constructs truth-invariant CADs of free-variable space from $\exists x[P(x)]$ and from $\exists x[Q(x)]$. Call the projection factor sets of these two CADs S_P and S_Q . The free-variable projection factors in B come from projecting $\exists x\exists y[P(x) \wedge Q(y)]$, and are actually exactly the closure under projection of $S_P \cup S_Q^\dagger$. The obvious approach then proceeds to construct the formulas F and G . Let S_F be the projection of the polynomials appearing in F , and let S_G be the projection of the polynomials appearing in G . The projection factor set A of the CAD produced from $F \wedge G$ is the closure under projection of $S_F \cup S_G$.

In the “best case”, CAD simplification takes place before F and G are constructed, so $S_F \subset S_P$ and $S_G \subset S_Q$. Thus, barring coincidental common factors, $A \subset B$.

In the “worst case”, the augmented projection (Collins, 1975) must be used to produce the formulas F and G and no simplification takes place, so that $S_P \subset S_F$ and $S_Q \subset S_G$. Thus, barring coincidental common factors, $B \subset A$.

Clearly, if neither case holds we may have $A = B$, and by mixing cases we may have A and B incomparable.

In this section it will be demonstrated that by representing F and G as simple CADs rather than formulas, we may follow the obvious approach and yet ensure that $A \subseteq B$, thus providing a method of dealing with non-prenex input formulas that is unambiguously superior to the method of converting to prenex form. Of course, this method will be applicable to any non-prenex quantified formula.

[†]Projecting to eliminate x only involves polynomials in $P(x)$ and produces polynomials in only the free variables. Projecting to eliminate y only involves polynomials in $Q(y)$ and produces polynomials in only the free variables. Thus, projecting to eliminate quantified variables in $\exists x\exists y[P(x) \wedge Q(y)]$ is exactly like projecting to eliminate the quantified variable x in the two separate problems $\exists x[P(x)]$ and $\exists x[Q(x)]$.

9.2. UNION AND INTERSECTION OF CADS REPRESENTING SEMI-ALGEBRAIC SETS

In Section 8 we described how the union and intersection of semi-algebraic sets can be computed efficiently using the simple CAD representation. There it was assumed that there was an ordering $x_1 < x_2 < \dots < x_k$ of the variables labeling the axes of \mathbb{R}^k , and that the CADs representing the sets that were to be combined by union and intersection were all CADs of k -dimensional space with respect to the same ordering $x_1 < x_2 < \dots < x_k$.

DEFINITION. We will call $x_{i_1} < x_{i_2} < \dots < x_{i_j}$, where $1 \leq i_1 < i_2 < \dots < i_j \leq k$, a *suborder* of $x_1 < x_2 < \dots < x_k$.

Here we again consider the union and intersection of two sets, A and B , represented by CADs. The CAD representing A is constructed with respect to some suborder o_A of $x_1 < x_2 < \dots < x_k$. The CAD representing B is constructed with respect to some suborder o_B of $x_1 < x_2 < \dots < x_k$. Defining the union of two suborders in the obvious way, we would like a CAD representation of $A \cup B$ or $A \cap B$ with respect to the variable order $o = o_A \cup o_B$. (Note that o is also a suborder of $x_1 < x_2 < \dots < x_k$.) This can be done using the technique employed in Section 8. We simply construct a CAD D with respect to o from the union of the projection factor sets of the CADs representing A and B . As we make a traversal of the cells of D , we are simultaneously traversing the cells of the CADs representing A and B . Thus, we can assign truth values to the cells of D based on the truth values of corresponding cells in the CAD representations of A and B .

9.3. AN ALGORITHM FOR QUANTIFIER ELIMINATION FOR NON-PRENEC FORMULAS

Suppose F is a quantified formula in the parameters $\alpha_1, \dots, \alpha_k$ and the quantified variables x_1, \dots, x_t . We will represent F by an expression tree whose interior nodes are labeled with the operators $\wedge, \vee, \neg, \exists x_i,$ or $\forall x_i$, and whose leaf nodes are labeled with polynomial equalities or inequalities. Obviously, nodes labeled $\neg, \exists x_i,$ or $\forall x_i$ will have a single child, and nodes labeled \wedge or \vee will have two children. For node N , let F_N denote the subformula of F represented by the tree rooted at N .

CADs are constructed with respect to a variable ordering, which we will represent as a permutation of the sequence $\alpha_1, \dots, \alpha_t, x_1, \dots, x_k$. If o is an order, let $mb(o, x_i)$ be the order represented by moving x_i to the end of the sequence o . We now define the algorithm NPQE—non-prenex quantifier elimination.

Algorithm NPQE(N, o).

Inputs: N , a node, and o , an order

Outputs: a simple CAD representation of the set defined by the tree rooted at N constructed with respect to some suborder of o

- (1) if N is a leaf node
 - (a) let C be a simple CAD constructed from N 's label with respect to the suborder of o containing exactly the same variables as N 's label
 - (b) return C
- (2) if N is labeled \neg
 - (a) let $C = \text{NPQE}(N, o)$

-
- (b) negate the truth values attached to each cell in C
 - (c) return C
- (3) if N is labeled $\exists x_i$ or $\forall x_i$
- (a) let $C' = \text{NPQE}(N', mb(o, x_i))$, where N' is the child of N
 - (b) since x_i is the last variable in the ordering with respect to which C' was constructed, we can easily propagate truth values to get a representation of $\exists x_i[F_{N'}]$ or $\forall x_i[F_{N'}]$ as a CAD, which we then can simplify. Call this CAD C .
 - (c) return C
- (4) otherwise N is labeled with \vee or \wedge
- (a) let N_L and N_R be the left and right children of N , respectively
 - (b) $C_L = \text{NPQE}(N_L, o)$
 - (c) $C_R = \text{NPQE}(N_R, o)$
 - (d) let C be the union (if $N = \vee$) or intersection (if $N = \wedge$) of C_L and C_R
 - (e) simplify C
 - (f) return C

It is clear that Steps 1 and 2 satisfy the output specifications of NPQE. To see why Step 3 also does, let $o = x_1 < x_2 < \cdots < x_k$. Step 3a produces the CAD C' that is constructed with respect to some suborder o' of

$$x_1 < \cdots < x_{i-1} < x_{i+1} < \cdots < x_k < x_i.$$

If x_i does not appear in o' , then o' is also a suborder of o . Moreover, in this case no propagation occurs, and C is C' . If x_i does appear in o' , then the propagation process removes the dimension associated with x_i from the CAD, leaving the CAD C constructed with respect to a suborder of $x_1 < \cdots < x_{i-1} < x_{i+1} < \cdots < x_k$, which is also a suborder of o . Thus, the CAD C that is returned in Step 3 satisfies the output specification that it be constructed with respect to a suborder of o .

Step 4, of course, satisfies the output specifications as well. Both C_L and C_R are constructed with respect to suborders of o , so the CAD representing their union or intersection will also be constructed with respect to a suborder of o .

9.4. DEMONSTRATING THAT NPQE REPRESENTS AN IMPROVEMENT

Algorithm NPQE is clearly better than the obvious approach of Section 9.1. Essentially, it does the same thing, differing only in that it uses simple CAD representations instead of formula representations. The advantage of using simple CAD representations is that the “extra” polynomials needed to make projection-undefinable CADs projection-definable are never added. In other words, we never need to use the augmented projection, or the method of adding derivatives to produce a projection-definable CAD described in Brown (1999a).

However, NPQE is also unambiguously superior to the method of adding variables to put the input formula into prenex form. What follows are three theorems that make this assertion precise. Theorem 9.1, the first of the three, essentially shows that the polynomials that arise as projection factors in computing $\text{NPQE}(F, o)$ also arise as projection factors in performing CAD-based quantifier elimination on a prenex equivalent of F . So even in the worst case, $\text{NPQE}(F, o)$ uses the same projection factors as converting

to prenex. However, instead of computing one CAD of high dimension, $NPQE(F, o)$ computes several CADs of lower dimension.

Let F be a non-prenex quantified formula in the variables x_1, \dots, x_k and the parameters $\alpha_1, \dots, \alpha_t$, and let o be the variable ordering $\alpha_1 < \dots < \alpha_t < x_1 < \dots < x_k$. Let g be a function that maps occurrences of variables in F to elements of $\{y_1, \dots, y_m\}$. If two occurrences of x_i refer to distinct variables, g maps the occurrences to distinct elements of $\{y_1, \dots, y_m\}$. If two occurrences of x_i refer to the same variables, g maps the occurrences to the same element of $\{y_1, \dots, y_m\}$. By the obvious extension, g maps F and any subformula of F to a formula in the variables y_1, \dots, y_m and the parameters $\alpha_1, \dots, \alpha_t$. The formula $g(F)$ can be put into prenex form by simply moving all quantifiers to the front, keeping them in the same left–right order. Let F' denote this formula. Without loss of generality, assume that from left to right in F' the variables are introduced in the order y_1, \dots, y_m . This means that in performing quantifier elimination by CAD on F' we project with respect to the bound-variable order $y_1 < \dots < y_m$.

If N is a node in the tree representation of F , let F_N denote the subformula of F represented by the tree rooted at N . Any free occurrence of x_i in F_N becomes mapped to the same y_j by g . Let $g_N : \{x_1, \dots, x_k\} \rightarrow \{y_1, \dots, y_m\}$ be the partial function that maps any variable x_i that appears free in F_N to the appropriate y_j . In the obvious way, extend g_N to polynomials and sets of polynomials.

THEOREM 9.1. *Let Q be the projection factor set that is computed when performing quantifier elimination by CAD on F' . Let N be a node in the tree representation of F . Let P be the projection factor set of the CAD returned by $NPQE(F_N, o_N)$. $g_N(P) \subseteq Q$*

PROOF. To prove this theorem, we require the following lemma, which essentially shows that as NPQE moves variables in orders, it is always dealing with orders of the x_i 's that correspond to suborders of $y_1 < \dots < y_m$. This will be key in connecting the projection computed by NPQE to the projection computed by the usual CAD-based quantifier elimination method for the formula F' using the order $y_1 < \dots < y_m$. \square

LEMMA 9.1. *Let N be a node in the tree representing F , and let o_N be the variable ordering associated with that node. If $x_{i_1} < \dots < x_{i_j}$ is the suborder of o_N containing exactly the free variables in F_N , then $g_N(x_{i_1}) < \dots < g_N(x_{i_j})$ is a suborder of $y_1 < \dots < y_m$.*

PROOF. Suppose x_a and x_b are two distinct free variables in F_N such that $x_a < x_b$ in o_N . Let N_a be the last node on the path from the root to N labeled $Q_a x_a$, and let N_b be the last node on the path from the root to N labeled $Q_b x_b$, where $Q_a, Q_b \in \{\exists, \forall\}$. Since $x_a < x_b$ in o_N , N_b must follow N_a in the path from the root to N . Thus, $g(F_{N_a}) = Q_a y_s [\dots Q_b y_t [\dots]]$, where $y_s = g_N(x_a)$, and $y_t = g_N(x_b)$. Therefore $g_N(x_a) < g_N(x_b)$ in $y_1 < \dots < y_m$. \square

We will proceed with our proof of Theorem 9.1 by induction on the depth of the tree rooted at N .

Suppose the depth of the tree rooted at N is 0. Then N is labeled by a single polynomial equality or inequality. Suppose that polynomial is $p(x_{i_1}, \dots, x_{i_j})$. Obviously, x_{i_1}, \dots, x_{i_j} are all free in F_N , so $q = p(g_N(x_{i_1}), \dots, g_N(x_{i_j}))$ appears in F' , and thus its factors are elements of Q . Moreover, by Lemma 9.1, the suborder of o_N consisting of the variables x_{i_1}, \dots, x_{i_j} maps under g_N to a suborder of $y_1 < \dots < y_m$. So the projection of p with

respect to o_N is exactly the projection of q with respect to $y_1 < \dots < y_m$, except that the variable names are different. Therefore, if P is the projection factor set of the CAD returned by $NPQE(F_N, o_N)$, then $g_N(P) \subseteq Q$.

Suppose the depth of the tree rooted at N is greater than 0. There are three cases to distinguish:

- (1) N is labeled with \neg . In this case, the projection factor set P of the CAD returned by $NPQE(F_N, o_N)$ is the same as the projection factor set P' of the CAD returned by $NPQE(F_{N'}, o_{N'})$, where N' is the child node of N . However, by induction we have $g_{N'}(P') \subseteq Q$. Clearly $g_N = g_{N'}$, so $g_N(P) \subseteq Q$.
- (2) N is labeled with $\forall x_i$ or $\exists x_i$. Let P' be the projection factor set of the CAD returned by $NPQE(F_{N'}, o_{N'})$, where N' is the child node of N . The projection factor set P of the CAD returned by $NPQE(F_N, o_N)$ is $\{p \in P' \mid \text{deg}_{x_i}(p) = 0\}$. The function g_N is exactly $g_{N'}$, except that x_i is not in the domain of g_N . Thus, $g_N(P) = g_{N'}(P) \subseteq g_{N'}(P')$. But by our inductive hypothesis, $g_{N'}(P') \subseteq Q$, so $g_N(P) \subseteq Q$.
- (3) N is labeled with \wedge or \vee . Let N' and N'' be the left and right children of N . Clearly $o_N = o_{N'} = o_{N''}$. If P' is the projection factor set for the CAD resulting from $NPQE(F_{N'}, o_{N'})$ and P'' is the projection factor set for the CAD resulting from $NPQE(F_{N''}, o_{N''})$, then by our inductive hypothesis we have $g_N(P') \cup g_N(P'') \subseteq Q$. Moreover, because Q is closed under projection, the complete projection with respect to $y_1 < \dots < y_m$ of $g_N(P') \cup g_N(P'')$ is also contained in Q . But, $NPQE(F_N, o_N)$ produces a CAD by computing P^* , the complete projection of $P' \cup P''$, and then simplifying the resulting CAD. Thus, $P \subseteq P^*$. But, projecting $P' \cup P''$ with respect to o_N differs from projecting $g_N(P') \cup g_N(P'')$ with respect to $y_1 < \dots < y_m$ only in the names of variables. Thus $P^* \subseteq Q$, which yields $P \subseteq Q$. \square

Theorem 9.1 shows that in the worst case, when no CAD simplification takes place during the running of NPQE, all projection factors computed are also projection factors computed by the usual CAD-based method of quantifier elimination on F' , our prenex form of the input formula. Thus, using projection factors as a measure, NPQE is certainly no worse than adding variables to put a non-prenex input formula in prenex form. To illustrate why NPQE is, typically, better, consider the following example input:

$$Q_1x[p_1(x, \alpha)\sigma_10] \wedge Q_2x[p_2(x, \alpha)\sigma_20] \wedge \dots \wedge Q_mx[p_m(x, \alpha)\sigma_m0]$$

where each p_i is of the form $a_ix + \alpha + b_i$, a_i, b_i are nonzero integers, each Q_i is an element of $\{\exists, \forall\}$, and each σ_i is an element of $\{>, <, =, \neq, \leq, \geq\}$. Let us compute the number of cells constructed for this problem by NPQE, and compare it to the number of cells resulting from putting the formula in prenex form and performing the usual CAD-based quantifier elimination.

In processing this input, NPQE will construct a CAD for each $Q_ix[p_i(x)\sigma_i0]$ consisting of one 1-level cell and three 2-level cells. After propagation, the resulting CAD will consist of a single cell. The process of merging these CADs will result in $m/2$ CADs of 1-space being constructed, then $m/4$ CADs of 1-space being constructed, \dots , down to 1 CAD of 1-space being constructed. Each of these will consist of a single cell. Assuming m is a power of two, $5m - 1$ cells will be constructed.

Putting the same formula into prenex form we get

$$Q_1y_1Q_2y_2 \dots Q_my_m[p_1(y_1, \alpha)\sigma_10 \wedge p_2(y_2, \alpha)\sigma_20 \wedge \dots \wedge p_m(y_m, \alpha)\sigma_m0]$$

Each stack construction into the dimension corresponding to y_i will result in three new cells—corresponding to the regions in which $a_i y_i + \alpha + b_i$ is negative, zero, and positive. Therefore, the CAD constructed for this formula has one 1-level cell, three 2-level cells, nine 3-level cells, \dots , 3^m $(m + 1)$ -level cells, for a total of $(3^{m+1} - 1)/2$.

Of course, the previous comparison is quite naive. It only compares the number of cells constructed, ignoring the fact that certain cells are much more costly to construct. When higher degree polynomials appear in the input, stack constructions involve algebraic number computations. At each successive level, the degree of the algebraic numbers involved grows. If, in the previous example, the p_i were quadratic instead of linear, some stack constructions involved in constructing a CAD from the prenex input formula would require computing in algebraic extensions of degree 2^{m-1} . NPQE, on the other hand, requires no computations with algebraic numbers for this example, regardless of the degrees of the p_i .

These examples suggest that NPQE constructs fewer cells than are constructed converting to prenex form, and that the sample points for these cells are simpler (i.e. lower degree algebraic extension) than sample points computed converting to prenex form. The following two theorems show that this is indeed always the case.

Let F be a non-prenex input formula in k variables (both free and bound), and let F' be a prenex form of F in n variables, $n > k$. For what follows, let us assume that only full sign-invariant CADs are constructed—i.e. let us leave aside the possibility of partial CADs.

If C' is the CAD produced from F' , and C is a CAD constructed by NPQE, then in a sense, each cell in C is the union of one or more cells in C' . The following theorem makes this statement precise.

THEOREM 9.2. *Let N be a node in the tree representation of F . Let C be the CAD returned by $NPQE(F_N, o_N)$. Let x_{i_1}, \dots, x_{i_r} be the suborder of o_N consisting of the variables that appear free in F_N . Let $y_{j_1} = g_N(x_{i_1}), \dots, y_{j_r} = g_N(x_{i_r})$. Let $h : \mathbb{R}^m \rightarrow \mathbb{R}^r$ be defined as follows:*

$$h((\alpha_1, \alpha_2, \dots, \alpha_m)) = (\alpha_{j_1}, \alpha_{j_2}, \dots, \alpha_{j_r}).$$

Let c' be a cell in C' and c a cell in C , then either $h(c') \cap c = \emptyset$, or $h(c') \subseteq c$.

PROOF. Let P be the projection factor set for C and let Q be the projection factor set for C' , then $g_N(P) \subseteq Q$. By definition, c' is a region in which all elements of Q have invariant sign. Let a and b be two points in $h(c')$ and let p be an element of P . Let a' and b' be points in c' such that $h(a') = a$ and $h(b') = b$, and let $q \in Q$ be $g_N(p)$. Then $p(a) = q(a')$ and $p(b) = q(b')$. So q has the same sign at both a and b . Thus, $h(c')$ is a region in which all elements of P have invariant sign—i.e. it is contained in some cell of C . \square

Here we will compare the algebraic computations performed during stack construction for NPQE and the method of converting to prenex form. One detail makes this comparison difficult, and that is that rational sample point coordinates for sectors are chosen arbitrarily during stack construction. To better compare these two methods, we will assume that both methods “make the same choice” when possible. Specifically, during stack construction rational sample point coordinates are chosen from open intervals. We will assume that this choice is made in such a way that if t is chosen for an interval (α, β) , then t will be chosen for any subinterval of (α, β) containing t . (One method for “choos-

ing” points that satisfies the criteria would be this: choose the rational number in (α, β) with smallest denominator, breaking ties by choosing the smaller magnitude numerator, breaking those ties by choosing the smaller rational number.) The following theorem shows that every sample point computed by NPQE is also computed in constructing a CAD for F' .

THEOREM 9.3. *Let N be a node in the tree representation of F . Let C be the CAD returned by NPQE(F_N, o_N). Let x_{i_1}, \dots, x_{i_r} be the suborder of o_N consisting of the variables that appear free in F_N . Let $y_{j_1} = g_N(x_{i_1}), \dots, y_{j_r} = g_N(x_{i_r})$. Let $h_s : \mathbb{R}^{j_s} \rightarrow \mathbb{R}^s$ be defined as follows:*

$$h_s((\alpha_1, \alpha_2, \dots, \alpha_{j_s})) = (\alpha_{j_1}, \alpha_{j_2}, \dots, \alpha_{j_s}).$$

Let C' be the CAD constructed from F' . If u is a sample point for an s -level cell in C , then some cell in C' has sample point v such that $h_s(v) = u$.

PROOF. We will prove this by induction on s . For $s = 0$, the root cell, the theorem is trivially true. Suppose $s > 0$. Let $u = (u_1, \dots, u_s)$ be the sample point of a cell c in C . Let b be the $(s - 1)$ -level base cell of the stack in which c resides. The sample point of b is then $w = (u_1, \dots, u_{s-1})$. By induction, there is a j_{s-1} -level cell b' in C' with sample point w' such that $h_{s-1}(w') = w$. We now distinguish two cases.

Case 1, c is a section cell. This means that u_s is a root of $p(u_1, \dots, u_{s-1}, x)$, where p is an s -level projection factor of C . By Theorem 9.1, there is a j_s -level projection factor q of C' such that $g_N(p) = q$. Stack construction over b' will construct cells for each section of q . The sample points of these cells will be $(\alpha_1, \dots, \alpha_{j_s-1}, \beta)$, where $w' = (\alpha_1, \dots, \alpha_{j_s-1})$, for each root β of $q(\alpha_{j_1}, \dots, \alpha_{j_{s-1}}, x)$. However,

$$q(\alpha_{j_1}, \dots, \alpha_{j_{s-1}}, x) = q(u_1, \dots, u_{s-1}, x) = p(u_1, \dots, u_{s-1}, x)$$

so $(\alpha_1, \dots, \alpha_{j_s-1}, u_s)$ is a sample point of a cell in C' , and

$$h_s((\alpha_1, \dots, \alpha_{j_s-1}, u_s)) = (\alpha_{j_1}, \dots, \alpha_{j_{s-1}}, u_s) = (u_1, \dots, u_{s-1}, u_s) = u.$$

Case 2, c is a sector cell. We will prove the result assuming that c is neither the first nor the last cell in the stack, since these other cases can be proven the same way. In this case, let $(u_1, \dots, u_{s-1}, \alpha)$ be the sample point of the cell directly below c and let $(u_1, \dots, u_{s-1}, \beta)$ be the sample point of the cell directly above c . Both cells are section cells. The sample point of c is (u_1, \dots, u_{s-1}, t) , where t is chosen from (α, β) .

From the preceding case, we see that there is a j_s -level cell in C' with sample point $z = (z_1, \dots, z_{j_s})$, such that $h_s(z) = (u_1, \dots, u_{s-1}, \alpha)$. Let p be a projection factor of which the cell directly above c is a section. Let q be $g_N(p)$. There are j_s -level section cells in C' with sample points $(z_1, \dots, z_{j_s-1}, \gamma)$ for each root γ of $q(z_{j_1}, \dots, z_{j_{s-1}}, x) = p(u_1, \dots, u_{s-1}, x)$. Thus, in particular, there is a cell with sample point $(z_1, \dots, z_{j_s-1}, \beta)$. There are one or more cells in between these two, since they are both sections. Either some section cell between them has sample point $(z_1, \dots, z_{j_s-1}, t)$, in which case we are done, or some sector cell has sample point $(z_1, \dots, z_{j_s-1}, t')$, where t' is chosen from some interval contained within (α, β) and containing t . But given our assumption about choosing rational sample points, t' equals t , and thus

$$h_s((z_1, \dots, z_{j_s-1}, t')) = (u_1, \dots, u_{s-1}, t). \quad \square$$

Putting these three theorems together, we are justified in saying that the set of alge-

braic problems NPQE has to solve in order to find a quantifier-free equivalent to F is actually a proper subset of the problems that have to be solved by performing CAD-based quantifier elimination on F' . (It is true that *NPQE* might have to solve each problem several times, but a clever implementation would simply remember intermediate results, thus removing this objection.) What is more, it is clear that this claim cannot be made for the obvious approach of replacing quantified prenex subformulas with equivalent quantifier-free formulas, since that may involve adding polynomials (either through the augmented projection or the method of Brown, 1999a) that are not used by NPQE and not used in performing CAD-based quantifier elimination on F' . Thus, NPQE is superior to both alternative approaches.

9.5. EXAMPLES

NPQE has not been implemented as part of QEPCAD. However, its behavior can be simulated by running QEPCAD multiple times and taking advantage of QEPCAD's interactive user interface.

In practice, NPQE can consider as a leaf node any quantifier-free subformula. Moreover, the tree representation of the formula need not be binary. In the following examples, these kinds of obvious improvements are made.

9.5.1. EXAMPLE 1

This example is intended to illustrate the benefits of not converting formulas to prenex form.

Consider the family of quadratic curves in x and y defined by $p(\alpha, \beta, x, y) = x^2 + \alpha xy + \beta y^2 - 1 = 0$. The question we wish to answer is this: For what parameter values does this curve have a component that is a straight line? Probably the most obvious way to phrase this as a quantifier elimination problem is as follows:

$$F_1 = \exists a, b, c \forall x, y [ax + by + c = 0 \implies x^2 + \alpha xy + \beta y^2 - 1 = 0]$$

This uses the fact that any line can be represented as $ax + by + c = 0$. Another possibility is to use the parameterization $y = mx + k$ to represent all non-vertical lines, and $x = x_1$ to represent all vertical lines. Putting these together yields the non-prenex formula

$$F_2 = \exists m, k \forall x [x^2 + \alpha x(mx + k) + \beta(mx + k)^2 - 1 = 0] \vee \exists x \forall y [x^2 + \alpha xy + \beta y^2 - 1 = 0]$$

in five rather than seven variables. Of course, this could be put into prenex form, yielding yet another formula:

$$F_3 = \exists x_1 \forall y_1 \exists m, k \forall x_2 [x_1^2 + \alpha x_1 y_1 + \beta y_1^2 - 1 = 0 \vee x_2^2 + \alpha x_2(mx_2 + k) + \beta(mx_2 + k)^2 - 1 = 0].$$

After more than half an hour, QEPCAD failed to compute a quantifier-free equivalent of F_1 . The equivalent formula $4\beta - \alpha^2 = 0$ was computed from F_2 in 0.45 seconds by computing simple CADs for the two quantified subformulas separately, computing a CAD for the union of the two sets, simplifying, and producing a solution formula. The same equivalent formula was produced from F_3 in 3.35 seconds. In total, 2,142 cells were constructed in producing an answer from F_2 using the strategy of NPQE. Applying QEPCAD to F_3 resulted in 13,302 cells being constructed.

9.5.2. EXAMPLE 2

This example is wholly artificial, intended to illustrate the benefits of using the simple CAD representation of an algebraic set, rather than the simple formula representation. The form of this example is

$$\exists a, b[\exists x[F_1(a, b, x)] \wedge \exists x, y[F_2(a, b, x, y)]],$$

where $F_1 = x^2 + y^2 - 1 = 0 \wedge y - 3x < 0 \wedge 16(a - x)^2 + 16(b - y)^2 - 1 \leq 0$, and $F_2 = ((2a - 1) - 2x) - 2(2(x + 2) - 1)(b - (2x + 3)) = 0 \wedge (2x - (2a - 1))^2 + 4((2x + 3) - b)^2 - 4 \leq 0$.

Both $\exists x[F_1(a, b, x)]$ and $\exists x, y[F_2(a, b, x, y)]$ result in CADs that are not projection definable, meaning that projection factors need to be added to their projection factor sets in order to construct solution formulas. This is not required, of course, if we use simple CADs to represent solutions.

Putting this input into prenex form results in an impractically large problem. Consider solving this problem by first computing F'_1 , a simple quantifier-free equivalent to $\exists x[F_1(a, b, x)]$. This takes QEPCAD 11.58 seconds, and requires adding polynomials to the projection factor set to produce a quantifier-free equivalent formula. Next the formula F'_2 , a simple quantifier-free equivalent to $\exists x, y[F_2(a, b, x, y)]$, is computed. This takes QEPCAD 2.81 seconds, and also requires adding polynomials to the projection factor set. Finally, quantifier-elimination commences on the formula $\exists a, b[F'_1 \wedge F'_2]$. QEPCAD returns FALSE for this input after 3.32 seconds. The entire process requires 17.71 seconds.

Suppose instead that we use NPQE. Computing a simple CAD representation of F_1 takes QEPCAD 10.39 seconds. Computing a simple CAD representation of F_2 takes QEPCAD 1.65 seconds. Combining these two CADs and using truth propagation to eliminate dimensions associated with a and b takes QEPCAD 0.29 seconds. Thus, FALSE is returned after 12.43 seconds.

Of course, the time difference is not so dramatic. But it illustrates the advantage of using the simple CAD representation instead of the formula representation.

9.6. COMMENTARY ON NPQE

It is important to note that the prenex quantified subformulas that NPQE solves could be sent off to other quantifier elimination packages to solve. Form2SCAD could then be used to construct a simple CAD representation of the result for further use by NPQE. Thus, NPQE has the potential to interact well with other methods in solving difficult problems.

There is another, equivalent way of viewing NPQE (and, in fact, Form2SCAD). It is shown in Brown (1999b) that the language of first order real algebra can be easily extended in such a way that any CAD is projection definable with respect to the extended language, i.e. there is always a defining formula in the extended language that uses only the polynomials in the projection factor set. Using this extended language to represent semi-algebraic sets is, as far as these algorithms are concerned, equivalent to using simple CADs. There is, in fact, a CAD-based quantifier elimination for this extended language. The extended language, which allows reference to indexed roots of polynomials, is no more expressive than the usual language of first order real algebra. However, certain sets can be described with fewer polynomials using the extended language. (Certain sets can also be defined easily with fewer variables.)

10. Conclusions

We have presented an algorithm for simplifying the truth-invariant CAD produced by the partial CAD algorithm of Collins and Hong (1991). Compared to the construction of the original truth-invariant CAD, constructing the simplified CAD is very fast. Example problems in Section 7 demonstrate that the simplified CAD may have far fewer cells, and a much smaller projection factor set than the original, although this is not always the case. These examples also show that using the simplified CAD as input to Hong's solution formula construction method can significantly improve its performance. Section 8 demonstrates that CAD simplification can be used during the CAD construction process to construct a truth-invariant CAD for a very large quantifier-free formula—a formula for which the direct application of Collins and Hong (1991) would be impractical. Finally, Section 9 shows that CAD simplification allows us to deal with non-prenex formulas in an efficient manner.

Semi-algebraic sets are usually represented as formulas from elementary real algebra. However, truth-invariant CADs provide another means of representation. Sections 8 and 9 show that union and intersection can be accomplished directly from this representation without having to compute equivalent formulas, which is important because constructing a formula from a truth-invariant CAD may require adding polynomials to the CADs projection factor set. CAD simplification provides a fast way to remove unnecessary polynomials after combining via union or intersection, so that a minimal representation can be retained at each step in a sequence of unions and intersections.

One direction for further research is the investigation of efficient methods for additional operations on semi-algebraic sets represented as truth-invariant CADs. Union and intersection have been discussed in this paper, but there are other important operations. For example, A CAD is constructed with respect to some variable ordering. One interesting question is how to efficiently change the variable ordering. Another problem is the construction of a truth-invariant CAD representation of a semi-algebraic set described by "substitution" into another semi-algebraic set. (For example, if S is a semi-algebraic set in 3-space, one might ask for the set of all points (x, y, z) such that $(x+y, y+z, x+z) \in S$. The new set would be defined by "substitution" into S .) One could perform either of these operations by switching to the quantifier-free formula representation, but this sometimes involves adding projection factors, which can be expensive.

Acknowledgements

I would like to thank George E. Collins for his invaluable help. Support during the writing of much this paper was provided by the University of Delaware through a University Fellowship. Some of the research represented here was done while under the support of Austrian FWF Project No. P8572-PHY. Further support has been received from the National Science Foundation under Grant No. CCR-9712246.

References

- Brown, C. W. (1998). Simplification of truth-invariant cylindrical algebraic decompositions. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pp. 295–301.
- Brown, C. W. (1999a). Guaranteed solution formula construction. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pp. 137–144.
- Brown, C. W. (1999b). Solution Formula Construction for Truth Invariant CAD's. Ph.D. Thesis, University of Delaware.

- Brown, C. W., G., E. Collins (1996). Simple truth invariant CAD's and solution formula construction. Technical Report 96-19, Research Institute for Symbolic Computation (RISC-Linz).
- Caviness, B. F., Johnson, J. R. (eds) (1998). *Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation*. Springer-Verlag.
- Collins, G. E. (1975). In *Quantifier Elimination for the Elementary Theory of Real Closed Fields by Cylindrical Algebraic Decomposition*, LNCS **33**, pp. 134–183. Berlin, Springer-Verlag. Reprinted in Caviness and Johnson (1998).
- Collins, G. E., Hong, H. (Sep 1991). Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, **12**, 299–328.
- Dolzmann, A., Sturm, T. (1996). Redlog—computer algebra meets computer logic. Technical Report MIP-9603, FMI, Universität Passau.
- Dolzmann, A., Sturm, T. (Aug. 1997). Simplification of quantifier-free formulae over ordered fields. *J. Symb. Comput.*, **24**, 209–231. Special Issue on Applications of Quantifier Elimination.
- Garey, M. R., Johnson, D. S. (1979). *Computers and Intractability—A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.
- Hong, H. (1990). Improvements in CAD-based Quantifier Elimination. Ph.D. Thesis, The Ohio State University.
- Hong, H. (1992). Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pp. 177–188.
- Kahan, W. (1975). Problem no. 9: An ellipse problem. *SIGSAM Bull. Assoc. Comp. Mach.*, **9**, 11.
- McCallum, S. (1998). An improved projection operator for cylindrical algebraic decomposition. In Caviness, B., Johnson, J. eds, *Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation*. Vienna, Springer-Verlag.
- Weispfenning, V. (1994). Quantifier elimination for real algebra—the cubic case. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pp. 258–263.
- Weispfenning, V. (1997). Quantifier elimination for real algebra—the quadratic case and beyond. *AAECC*, **8**, 85–101.

*Originally Received 13 November 1998
In revised form 20 March 2000
Accepted 16 May 2000*