J. Symbolic Computation (1991) 12, 299-328

Partial Cylindrical Algebraic Decomposition for Quantifier Elimination*

George E. Collins and Hoon Hong

Department of Computer Science, Ohio State University, Columbus, Ohio 43210, USA

(*Received 6 November 1989*)

The Cylindrical Algebraic Decomposition method (CAD) decomposes R^r into regions over which given polynomials have constant signs. An important application of CAD is quantifier elimination in elementary algebra and geometry. In this paper we present a method which intermingles CAD construction with truth evaluation so that parts of the CAD are constructed only as needed to further truth evaluation and aborts CAD construction as soon as no more truth evaluation is needed. The truth evaluation utilizes in an essential way any quantifiers which are present and additionally takes account of atomic formulas from which some variables are absent. Preliminary observations show that the new method is always more efficient than the original, and often significantly more efficient.

1 Introduction

Cylindrical Algebraic Decomposition (CAD) by Collins (1975) provides a potentially powerful method for solution of many important mathematical problems by means of quantifier elimination, provided that the required amount of computation can be sufficiently reduced. Arnon (1981) introduced the important method of clustering for reducing the required computation and McCallum (1984) introduced an improved projection operation which is also very effective in reducing the amount of computation. In this paper we introduce yet another method for reducing the amount of computation which we will call *partial CAD construction.*

The method *partial CAD construction* is based on the simple observation that we can *very often* complete quantifier elimination by a *partially* built CAD if we utilize more information contained in the input formula. Our method utilizes three aspects of the input formula: the quantifiers, the Boolean connectives, and the absence of some variables from some polynomials occurring in the input formula. Preliminary observations show that the new method is always more efficient than the original, and often significantly more efficient.

A simple example can illustrate how we utilize the quantifier information. Let us consider a sentence in two variables $(\exists x)(\exists y)F(x,y)$. The original CAD method computes

*This research was supported in part by NSF grant DCR-8696129.

a certain decomposition D_1 of R and then lifts this to a decomposition D_2 of R^2 by constructing a stack of cells in the cylinder over each cell of D_1. Then the quantifier elimination proceeds by determining the set of all cells of D_1 in which $(\exists y)F(x,y)$ is true. Finally, it computes the truth value of $(\exists x)(\exists y)F(x,y)$ by checking whether the set is empty. In contrast, our method constructs only one stack at a time, aborting the CAD construction as soon as a cell of D_1 is found which satisfies $(\exists y)F(x,y)$, if any such cell exists. If, instead, the given sentence were $(\forall x)(\exists y)F(x,y)$, our method would stop as soon as any cell is found in which $(\exists y)F(x,y)$ is false. The quantifier $(\exists y)$ could be changed to $(\forall y)$ without effect. The method illustrated above for two variables extends in an obvious way to more variables, with even greater effectiveness because the CAD construction can be partial in each dimension. This idea applies equally to formulas in which some variables are free, and the CAD construction can be partial again as in the above example.

Another simple example can illustrate how we utilize the Boolean connectives and the absence of some variables from some polynomials. Let $(Qx)(Qy)F(x,y)$ a sentence in two variables such that $F(x,y) = F_1(x) \wedge F_2(x,y)$, where F_1 and F_2 are quantifier-free formulas and Q's are arbitrary quantifiers. Note that the variable y is absent from the formula F_1 and thus from the polynomials contained in F_1. The original CAD method determines the truth value of $(Qy)F(x,y)$ in a cell c of D_1 by building a stack over it. In contrast, our method first determines the truth value of $F_1(x)$ by evaluating the associated polynomials on the sample point of c. If it is false, $(Qy)F(x,y)$ is clearly false in the cell c, so we do not need to build a stack over it. Likewise, if $F(x,y) = F_1(x) \vee F_2(x,y)$ and $F_1(x)$ is true in a cell c, $(Qy)F(x,y)$ is clearly true in the cell c, so we do not build a stack over it either. This method can be generalized in an obvious way to arbitrary number of variables and an arbitrary formula $F(x_1,\ldots,x_r)$.

Remember that our method builds only one stack at a time. But at a certain time there might be many candidate cells on which we can build stacks, and so we need to choose one of them. We have dealt with this problem as follows. We have designed an interactive program which allows the user to choose among candidate cells, or to choose one of a small number of selection algorithms, called cell-choice strategies, which the program will use repeatedly.

Some of these strategies were designed to be efficient for each of several sub-problem classes, namely collision problems in robot motion planning (Buchberger *et al.*, 1989), termination proof of term rewrite systems based on polynomial interpretation (Lankford 1979, Huet & Oppen 1980) , and consistency of polynomial strict inequalities (McCallum 1987). For these problem classes we achieved very large reductions in required computation time.

The plan of the paper is as following: In Section 2 we elaborate the main ideas underlying our method. In Section 3 we present the partial CAD construction algorithm. This algorithm is generic in the sense that it does not impose any particular cell-choice strategy. In Section 4 we describe various cell-choice strategies. In Section 5 we illustrate our method by showing a terminal session on a simple but "real" example. In Section 6, we present some empirical comparisons for several problems from diverse application areas.

2 Main Idea

In this section, we elaborate the main ideas underlying our method by showing how our main algorithm evolved from the original one. We assume that the reader is familiar with the basic terminology of (Arnon *et al.*, 1984), including that of CAD, induced CAD, cells, cell indices, cylinders, stacks, sample points, defining formulas, delineability, sign-invariance, and truth–invariance. We also assume that the reader is familiar with the basic terminology of graph theory and trees, including that of roots, leaves, children, parents, ancestors, descendants, and levels. (The level of the root node is defined to be 0.)

2.1 Original Algorithm

Let $F^* = (Q_{f+1}x_{f+1})\cdots(Q_r x_r)F(x_1,\ldots,x_r)$, $0 \le f < r$, be any formula of elementary algebra, where the formula $F(x_1,\ldots,x_r)$ is quantifier-free. Let D_r be any truth-invariant CAD of R^r for $F(x_1,\ldots,x_r)$. For $f \le k < r$, D_r induces a CAD D_k of R^k in which the formula $F_k^* = (Q_{k+1}x_{k+1})\cdots(Q_r x_r)F(x_1,\ldots,x_r)$ is truth-invariant. Thus if c is any cell of D_k, the formula F_k^* has a constant truth value throughout c. This observation leads to the following definition.

Definition. [Truth value] Let c be a cell of D_k, $f \le k \le r$, and let F_k^* be the formula $(Q_{k+1}x_{k+1})\cdots(Q_r x_r)\ F(x_1,\ldots,x_r)$. We define $v(c)$, called *the truth value of the cell c*, to be the constant truth value of F_k^* throughout c.

In order to treat the case $f = k = 0$ uniformly we hypothesize a decomposition D_0 of R^0 having a single cell, whose truth value is assumed to be the truth value of the sentence $(Q_1x_1)\cdots(Q_r x_r)F(x_1,\ldots,x_r)$. From Definition 2.1 the following theorem is immediate.

Theorem 2.1 (Evaluation). *Let c be a cell of D_r and let $s = (s_1,\ldots,s_r)$ be a sample point of c, where each s_i is the i-th coordinate of the sample point. Then $v(c) =$ the truth value of $F(s_1,\ldots,s_r)$.*

This theorem provides a way to evaluate the truth values of cells of D_r from their sample points. From Theorem 7 of Collins (1975) , the following theorem is immediate.

Theorem 2.2 (Propagation). *Let c be a cell of D_k, $f \le k < r$, and let $c_1,\ldots,c_n$ be the cells in the stack over c. If $Q_{k+1} = \exists$, $v(c) = \bigvee_{i=1}^{n} v(c_i)$. If $Q_{k+1} = \forall$, $v(c) = \bigwedge_{i=1}^{n} v(c_i)$.*

This theorem provides a way to propagate the truth values of the cells of D_{k+1} to the truth values of the cells of D_k. From Definition 2.1, the following theorem is immediate.

Theorem 2.3 (Solution). *Let $S = \{c \in D_f | v(c) = true\}$ and let $W = \bigcup_{c\in S} c$. Then $(Q_{f+1}x_{f+1})\cdots(Q_r x_r)F(x_1,\ldots,x_r) \iff (x_1,\ldots,x_f) \in W$.*

This theorem provides a way to compute the solution set of an input formula. From Theorems 2.1, 2.2, and 2.3, Collins' original quantifier elimination algorithm follows as:

Original Algorithm

(1) [CAD construction.] Build a CAD D_r of R^r.
(2) [Evaluation.] Evaluate the truth values of the cells of D_r by using Theorem 2.1.

(3) [Propagation.] For $k = r-1, r-2, \ldots, f$, determine the truth values of the cells of D_k from the truth values of the cells of D_{k+1} by using Theorem 2.2.
(4) [Solution.] Get the solution set W by using Theorem 2.3. (Now a defining formula for the set W is equivalent to the input formula.) □

2.2 First Improvement

Our first improvement of the original algorithm is based on a simple observation on Theorem 2.2 that the truth value of a cell c can *often* be determined from the truth values of *only some* of the cells in the stack over c. To be specific, let c be a cell of D_k, $f \le k < r$. In case $Q_{k+1} = \exists$, $v(c)$ can be determined to be *true* as soon as the truth value of one of the cells in the stack over c is known to be *true*. Likewise, in case $Q_{k+1} = \forall$, $v(c)$ can be determined to be *false* as soon as the truth value of one of the cells in the stack over c is known to be *false*.

This observation was already made in Collins (1975) (page 158) and used to reduce the number of truth evaluations in Step 2. We now, however, carry the observation further to reduce the number of cells constructed in Step 1. This can be done by intermingling CAD construction with truth evaluation so that parts of the CAD are constructed only as needed to further truth evaluation and aborts CAD construction as soon as no more truth evaluation is needed. By doing so, we can often complete quantifier elimination by a *partially built* CAD, thus reducing the amount of required computation.

In order to take the full advantage of this idea of *partial CAD construction*, we should also do our best to avoid any computations which will not be used later on in the process. This can be done by postponing all the computations which are not needed at a given moment, and carry them out only when it becomes clear that they are needed.

One such computation is the conversion of representation of a sample point. Let c be a cell of D_k, $1 \le k \le r$. As Arnon (1988) points out, during CAD construction, the sample point of the cell c is represented in either one of the two representations: (a) *primitive*, consisting of a real algebraic number α and a k-tuple of elements of $Q(\alpha)$. (b) *extended*, consisting of a real algebraic number α, and a $(k-1)$-tuple of elements of $Q(\alpha)$, a non-zero squarefree polynomial $g(x) \in Q(\alpha)[x]$, and an isolating interval for a real root of $g(x)$. (This root is the k-th coordinate of the sample point.) From now on we will call a sample point *an extended sample point* if it is in an extended representation, and *a primitive sample point* if it is in a primitive representation.

During CAD construction, we first get an extended sample point of a cell while building a stack to which the cell belongs (except when the cell is a sector or it belongs to D_1; in this case we trivially get a primitive sample point). Then the original CAD algorithm converts it into a primitive one immediately by using the *NORMAL* and *SIMPLE* algorithms of Loos (1982) . This conversion process, however, is known to be often very expensive. So we would like to postpone the conversion process and carry it out only when it is needed. A primitive sample point of a cell is needed only when we want to build a stack over the cell or to evaluate its truth value[1]. Therefore we should carry out the conversion process just before building a stack or evaluating truth value. In this way, we do not waste time converting the representation of sample points which will not be used in the remaining

[1]Actually, as you will see later in this section, we can carry out truth evaluation with an extended sample point.

process of the *partial* CAD construction.

Let us now devise an algorithm which utilizes these ideas. First, we define some terminology. A *partial CAD* is a tree of cells such that the root node is the hypothetical single cell of D_0, the children of the root node are the cells of D_1, and the children of a non-root node c are the cells in the stack over the cell (node) c. We denote the level of a cell c in a *partial* CAD tree as $l(c)$. A cell c is a leaf if either $l(c) = r$ or there is no stack built on it yet. Each cell in a partial CAD may have the following two data structures associated with it: *sample point* and *truth value*. The truth value associated with a cell c at any time can be one of the three: *true*, *false*, and *undetermined*. The value associated is *undetermined* if either $l(c) < f$ or it is not determined yet. A *candidate* cell is a leaf whose truth value is *undetermined*. Now here is an algorithm that utilizes the ideas discussed above:

First Improvement

(1) [Projection.] Compute the projection polynomials of all orders.
(2) [Initialization.] Initialize the partial CAD D as the single cell of D_0 whose truth value is set to be *undetermined*.
(3) [Choice.] If there is no candidate cell in the current partial CAD D, go to Step (8). Choose a candidate cell c from the current partial CAD D.
(4) [Conversion.] If $l(c) > 1$ and c is a section, convert the extended sample point of c into a primitive one by using the *NORMAL* and *SIMPLE* algorithms. If $l(c) = r$, go to Step (6).
(5) [Augmentation.] Construct a stack over c and form a sample point for each child cell. (If $l(c) = 0$, construct a primitive sample point for every child. Otherwise, construct a primitive sample point for each sector and an extended sample point for each section.) Go to step (3).
(6) [Evaluation.] Evaluate the truth value of the cell c by using Theorem 2.1
(7) [Propagation.] Determine the truth values of as many ancestors of c as possible by Theorem 2.2, removing from the tree the subtrees of each cell whose truth value is thus determined. Go to Step (3).
(8) [Solution.] (At this point, the truth value of every cell of D_f has been determined.) Get the solution set W by using Theorem 2.3. (Now a defining formula for the set W is equivalent to the input formula.) □

It is important to note that the efficiency of this algorithm depends on which candidate cell is chosen at each iteration. It is also important to note that this algorithm is generic in the sense that it does not impose any particular strategy for choosing candidate cells. Thus it naturally raises a question: *Which strategy for candidate cell choice is best?* We will discuss this problem in Section 4.

2.3 Second Improvement

Further improvement of the generic algorithm is based on the observation that we can *often* carry out truth evaluation on a cell c such that $l(c) < r$, if some variables are absent from some polynomials occurring in the input formula. This is important because if we succeed in evaluating the truth value of the cell c, we do not have to build a partial CAD over c, thus reducing the amount of computation.

In this section, we also make another minor improvement by allowing propagation below free variable space which often results in further tree pruning.

These improvements are essentially generalization of Definition 2.1 and Theorems 2.1, 2.2, and 2.3. So we will describe the improvements by showing how we generalize the definition and the theorems.

Definition 2.1 defined the *truth value* for a cell c such that $f \leq l(c) \leq r$ (i.e. on or above free variable space). Now we would like to define the *truth value* for a cell c such that $0 \leq l(c) < f$ (i.e. below free variable space). The following definition is found to be useful.

DEFINITION. [Truth value of a cell below free variable space] Let c be a cell of D_k, $0 \leq k < f$. Let $F^* = (Q_{f+1}x_{f+1}) \cdots (Q_r x_r) F(x_1, \ldots, x_r)$. If the formula F^* has a constant truth value, say w, throughout $c \times R^{f-k}$, then we define $v(c)$ to be the constant truth value w. Otherwise, $v(c)$ is undefined.

Let us now generalize Theorem 2.2. Theorem 2.2 provided a way to propagate truth values, but only *on* or *above* free variable space, and now we would like to propagate truth values even *below* free variable space.

THEOREM 2.4 (PROPAGATION BELOW FREE VARIABLE SPACE). *Let c be a cell of D_k, $0 \leq k < f$, and let $c_1, \ldots, c_n$ be the cells in the stack over c. If $v(c_i) = w$ for all i, $1 \leq i \leq n$, then $v(c) = w$.*

PROOF. Suppose that $v(c_i) = w$ for all i. Then from Definition 2.3 it is clear that $F^* = (Q_{f+1}x_{f+1}) \cdots (Q_r x_r) F(x_1, \ldots, x_r)$ has the constant truth value w throughout $c_i \times R^{f-(k+1)}$ for all i. Therefore F^* has the constant truth value w throughout $(\bigcup_{i=1}^{n} c_i) \times R^{f-(k+1)}$. Then, since $\bigcup_{i=1}^{n} c_i = c \times R$, F^* has the constant truth value w throughout $c \times R^{f-k}$. Thus $v(c)$ is defined and $v(c) = w$. □

Let us now generalize Theorem 2.1. Theorem 2.1 provided a way to carry out truth evaluation on a cell c such that $l(c) = r$. Now we would like to generalize it so that we can also carry out truth evaluation on a cell c such that $0 < l(c) < r$, if possible.

Let $f(x_1, \ldots, x_r)$ be a non-constant r-variate integral polynomial. We define $n(f)$, called *the effective number of variables in the polynomial f*, as the biggest integer k, $1 \leq k \leq r$, such that such that $deg_k(f) > 0$, where $deg_k(f)$ is the degree of the polynomial f with respect to the variable x_k. Let $A = \{A_1, \ldots, A_n\}$, $n \geq 1$, be the set of the non-constant r-variate integral polynomials occurring in an input formula $F(x_1, \ldots, x_r)$. Let $A^{[k]}$, $1 \leq k \leq r$, be the subset of A such that $A^{[k]} = \{f \in A | n(f) = k\}$. Let $A^{(k)}$, $1 \leq k \leq r$, be the subset of A such that $A^{(k)} = \{f \in A | n(f) \leq k\}$.

In quantifier elimination problems arising in applications, $A^{(k)}$ is often a non-empty set for some $k < r$. In this case, we can determine the signs of the polynomials in $A^{(k)}$ on a cell c of D_k by evaluating the polynomials on its sample point, and thereby the truth values of the atomic formulas in which the polynomials occur. Then it may happen that the truth value of $F(x_1, \ldots, x_r)$ is determined by the truth values of these atomic constituents. In other words, $F(s_1, \ldots, s_k, x_{k+1}, \ldots, x_r)$ may have a constant truth value throughout R^{r-k}, where $(s_1, \ldots, s_k)$ is the sample point of c.

For example, consider the formula $F(x, y) = A_1(x) > 0 \wedge A_2(x, y) < 0$. Let c be a cell of D_1, and let $s = (s_1)$ be its sample point. We can determine the sign of $A_1(x)$ by evaluating $A_1(x)$ on s_1. Suppose that $A_1(s_1) < 0$. Then we know right away $F(s_1, y)$ is *false* no matter what y may be.

In such cases we can evaluate $v(c)$ by using the following generalized evaluation theorem.

THEOREM 2.5 (GENERALIZED EVALUATION). *Let c be a cell of D_k, $1 \leq k \leq r$, and let $s = (s_1, \ldots, s_k)$ be its sample point. If $F(s_1, \ldots, s_k, x_{k+1}, \ldots, x_r)$ has a constant truth value, say w, throughout R^{r-k}, then $v(c) = w$.*

PROOF. We will use mathematical induction on k from $k = r$ down to $k = 1$. In case $k = r$, the theorem is reduced to Theorem 2.1, and so we are done. Assume that the theorem is true for k, $r \geq k \geq 2$. Now we only need to prove that the theorem is true for $k-1$. Let c be a cell of D_{k-1}, and let $c_1, \ldots, c_n$ be the cells in the stack over c. Let $s_1, \ldots, s_{k-1}$ be a sample point of c, and let $s_1, \ldots, s_{k-1}, s_k^{(i)}$ be a sample point of c_i $(1 \leq i \leq n)$. Suppose that $F(s_1, \ldots, s_{k-1}, x_k, \ldots, x_r)$ has a constant truth value, say w, throughout $R^{r-(k-1)}$. Then $F(s_1, \ldots, s_{k-1}, s_k^{(i)}, x_{k+1}, \ldots, x_r)$ has the constant truth value w throughout R^{r-k} for all i. So, by the induction hypothesis, $v(c_i) = w$ for all i. If $k - 1 \geq f$, from Theorem 2.2 we have $v(c) = w$. Otherwise, from Theorem 2.4 we have $v(c) = w$. $\square$

Now we need to generalize Theorem 2.3 in order to facilitate Theorems 2.4 and 2.5. From Definition 2.3 the following theorem is immediate.

THEOREM 2.6 (GENERALIZED SOLUTION). *Let $Z = \{c_1, \ldots, c_n\}$ be a set of cells such that $R^f = \bigcup_{c \in Z} c \times R^{f-l(c)}$ and that $v(c_i)$ is defined for all i. Let S be a subset of Z such that $S = \{c \in Z | v(c) = true\}$ and let $W = \biguplus_{c \in S} c \times R^{f-l(c)}$. Then $(x_1, \ldots, x_f) \in W \Longleftrightarrow (Q_{f+1}x_{f+1}) \cdots (Q_r x_r) F(x_1, \ldots, x_r)$.*

The following algorithm utilizes Theorems 2.5, 2.2, 2.4, and 2.6.

Second Improvement

(1) [Projection.] Compute the projection polynomials of all orders.

(2) [Initialization.] Initialize the partial CAD D as the single cell of D_0 whose truth value is set to be *undetermined*.

(3) [Choice.] If there is no candidate cell in the current partial CAD D, go to Step (8), otherwise, choose a candidate cell c from the current partial CAD D.

(4) [Conversion.] If $l(c) > 1$ and c is a section, convert the extended sample point of c into a primitive one by using the *NORMAL* and *SIMPLE* algorithms. If $A^{[l(c)]} \neq \emptyset$, go to Step (6).

(5) [Augmentation.] Construct a stack over c and form a sample point for each child cell. (If $l(c) = 0$, construct a primitive sample point for every child. Otherwise, construct a primitive sample point for each sector and an extended sample point for each section.) Go to Step (3).

(6) [Trial Evaluation.] Try to evaluate the truth value of the cell c by using the Boolean connectives and Theorem 2.5. If the evaluation fails, go to Step (5).

(7) [Propagation.] Determine the truth values of as many ancestors of c as possible by Theorems 2.2 and 2.4, removing from the tree the subtrees of each cell whose truth value is thus determined. Go to Step (3).

(8) [Solution.] Set $Z \leftarrow$ the set of all leaves of the partial CAD D. (At this point, the set Z satisfies the hypothesis of Theorem 2.6.) Get the solution set W by using Theorem 2.6. (Now a defining formula for the set W is equivalent to the input formula.) □

2.4 Final Improvement

We now make our final improvement based on the observation that we can carry out the trial truth evaluation with extended sample points. This is important because, as mentioned earlier, the conversion process is often very expensive.

Let $A = (A_1, \ldots, A_n)$, $n \geq 1$, be a set of r-variate integral polynomials. $r \geq 1$. Let f be the number of free variables in the input formula. Let $C = (C_1, \ldots, C_n)$ be the set of $(r-1)$-variate integral polynomials where C_j is the content of A_j for $1 \leq j \leq n$. Let $P = (P_1, \ldots, P_n)$ be the set of r-variate integral polynomials where P_j is the primitive part of A_j for $1 \leq j \leq n$. We include the signs into the contents so that $A_j = C_j P_j$ for $1 \leq j \leq n$. Let $B = (B_1, \ldots, B_l)$ be a finest square-free basis of P. Let $PROJ(B)$ denote any projection set for B sufficient to ensure delineability.[2] Then we define $Proj(A)$ to be $C \cup PROJ(B)$. We further define $Proj^0(A)$ as A itself and $Proj^k(A)$ as $Proj(Proj^{k-1}(A))$ for $1 \leq k \leq r-1$.

Let c be a cell of D_k, $1 \leq k < r$, and let $c_1, \ldots, c_n$ be the children of c. Arnon (1988) presents an algorithm, *InputSignaturesOverCell*, that evaluates the signs of the polynomials in $Proj^{r-(k+1)}(A)$ on $c_1, \ldots, c_n$, given the extended representations for the cells' sample points. In Section 3 we present another algorithm *SIGNJ* which does the same thing more efficiently. This algorithm together with the following theorem provides a way to determine the signs of the polynomials in $A^{(k+1)}$ on $c_1, \ldots, c_n$ even though the sample points of $c_1, \ldots, c_n$ exist in extended representations.

Theorem 2.7. *Let $A = \{A_1, \ldots, A_n\}$, $n \geq 1$, be a set of the non-constant r-variate integral polynomials. Then $A^{(k)} \subseteq Proj^{r-k}(A)$ for $1 \leq k \leq r$.*

Proof. We will use mathematical induction on k from $k = r$ down to $k = 1$. In case $k = r$, we have $A^{(r)} = A = Proj^0(A)$. So we are done. Assume that the theorem is true for k, $r \geq k \geq 2$. Now we only need to prove that the theorem is true for $k-1$. From the induction hypothesis and the obvious fact $A^{(k-1)} \subseteq A^{(k)}$, it is clear that $A^{(k-1)} \subseteq Proj^{r-k}(A)$. Let C be the set of the contents of the polynomials in $Proj^{r-k}(A)$. Then $A^{(k-1)} \subseteq C$. Since $C \subseteq Proj(Proj^{r-k}(A))$, we have $A^{(k-1)} \subseteq Proj(Proj^{r-k}(A)) = Proj^{r-(k-1)}$. □

In summary, let c be a cell of D_k, $1 \leq k < r$. and let $c_1, \ldots, c_n$ be the children of c. We try to evaluate the truth values of $c_1, \ldots, c_n$ in three steps: (1) Use *SIGNJ* to compute the signs of the polynomials in $Proj^{r-(k+1)}(A)$ on $c_1, \ldots, c_n$. (2) Use Theorem 2.7 in order to determine the signs of the polynomials in $A^{(k+1)}$ on $c_1, \ldots, c_n$. In fact, this amounts to searching through the set $Proj^{r-(k+1)}(A)$ to find of an element of $A^{(k+1)}$.[3] (3) Try to evaluate the truth values of $F(x_1, \ldots, x_r)$ on $c_1, \ldots, c_n$ from the signs of the polynomials

[2]For such projection sets, see Collins (1975), McCallum (1984), and Hong (1990a).

[3]Actually this search is not necessary if we compute the signs of only those elements of $Proj^{r-(k+1)}$ that are also in $A^{(k+1)}$. The details of this idea are discussed in Hong (1990b). But the statistics given in Section 6 of this paper are obtained without this further improvement.

in $A^{(k+1)}$ on $c_1, \ldots, c_n$. According to Theorem 2.5, $v(c_i)$ = the evaluated truth value if the evaluation was successful for the cell c_i.

The following algorithm utilizes this idea.

Final Improvement

(1) [Projection.] Compute the projection polynomials of all orders.

(2) [Initialization.] Initialize the partial CAD D as the single cell of D_0 whose truth value is set to be *undetermined*.

(3) [Choice.] If there is no candidate cell in the current partial CAD D, go to Step (8), otherwise, choose a candidate cell c from the current partial CAD D.

(4) [Conversion.] If $l(c) > 1$ and c is a section, convert the extended sample point of c into a primitive one by using the *NORMAL* and *SIMPLE* algorithms.

(5) [Augmentation.] Construct a stack over c and form a sample point for each child cell. (If $l(c) = 0$, construct a primitive sample point for every child. Otherwise, construct a primitive sample point for each sector and an extended sample point for each section.) If $A^{[l(c)+1]} = \emptyset$, go to Step (3).

(6) [Trial Evaluation.] Try to determine the truth values of the children of c by using *SIGNJ*, Theorem 2.7, the Boolean connectives, and Theorem 2.5.

(7) [Propagation.] Determine the truth value of c , if possible and if it is, the truth values of as many ancestors of c as possible by Theorems 2.2 and 2.4, removing from the tree the subtrees of each cell whose truth value is thus determined. Go to Step (3).

(8) [Solution.] Set $Z \leftarrow$ the set of all leaves of the partial CAD D. (At this point, the set Z satisfies the hypothesis of Theorem 2.6.) Get the solution set W by using Theorem 2.6. (Now a defining formula for the set W is equivalent to the input formula.) □

3 Partial CAD Construction Algorithm

The partial CAD construction algorithm consists of a main algorithm *QEPCAD* and seven subalgorithms: *PROJM*, *CHOOSE*, *CCHILD*, *EVALTV*, *SIGNJ*, *SIGNB*, and *PRPTV*.

The main algorithm *QEPCAD* (Fig 1) is a slightly more formal rendering of the algorithm developed in Section 2.

The subalgorithm *PROJM* (Fig 2) essentially computes the projections J of all orders. It, however, in preparation for the algorithm *SIGNJ*, keeps also several intermediate results, namely the contents C and primitive parts P of the projections J (see Equation 1), and the finest squarefree bases B of the primitive parts P (see Equation 2). For each squarefree basis it also computes a matrix E of the multiplicities of the basis elements in the primitive parts (see Equation 2).

$$J_i = C_i P_i \tag{1}$$

$$P_i = \prod_{j=1}^{m} B_j^{E_{i,j}} \tag{2}$$

The subalgorithm *CHOOSE* chooses a candidate cell c from the current partial CAD. Discussion of *CHOOSE* is deferred to Section 4

The subalgorithm *CCHILD* (Fig 3) builds a stack over the chosen cell c. Note that this subalgorithm, like *PROJM*, also keeps several intermediate results, namely substituted algebraic polynomials A^* (see Equation 3), their leading coefficients L^* and monic associates P^* (see Equation 4), the coarsest squarefree basis B^* of the monic associates P^* (see Equation 5), and the real roots I^* of the squarefree basis B^*. Furthermore, it computes a matrix E^* of multiplicities of the squarefree basis B^* in the monic associates P^* (see Equation 5).[4]

$$\begin{aligned} B_j(s, x_{k+1}) &= A_j^* && (3)\\ A_j^* &= L_j^* P_j^* && (4)\\ P_j^* &= \prod_{g=1}^{l} B_g^{*E_{j,g}^*} && (5) \end{aligned}$$

In the following discussions, let $\sigma(X)$ stand for the sign of the polynomial X on the chosen cell c, and $\sigma_i(X)$ stand for the sign of X on the i-th child of c.

The subalgorithm *SIGNB* (Fig 4), given the real roots I^* and the squarefree basis B or B^*, computes the signs of the squarefree basis element on each child, that is $\sigma_h(B_j)$ or $\sigma_h(B_g^*)$. This algorithm does the same thing as Arnon's *BasisSignaturesOverCell* (Arnon 1988), but it avoids some unnecessary computation.

The subalgorithm *SIGNJ* (Fig 5), given the signs of the squarefree basis elements on the children ($\sigma_h(B_j)$ or $\sigma_h(B_g^*)$), computes the signs of the projections polynomials on each children, that is $\sigma_h(J_i)$. First, note the following equations resulting immediately from Equations 1, 2, 3, 4, and 5. (In case $l(c) = 0$, Equations 6, 7, and 8 are irrelevant.)

$$\begin{aligned} \sigma_h(P_j^*) &= \prod_{g=1}^{l} \sigma_h(B_g^*)^{E_{j,g}^*} && (6)\\ \sigma_h(A_j^*) &= \sigma(L_j^*)\sigma_h(P_j^*) && (7)\\ \sigma_h(B_j) &= \sigma_h(A_j^*) && (8)\\ \sigma_h(P_i) &= \prod_{j=1}^{m} \sigma_h(B_j)^{E_{i,j}} && (9)\\ \sigma_h(J_i) &= \sigma(C_i)\sigma_h(P_i) && (10) \end{aligned}$$

In case $l(c) > 0$, we first compute $\sigma_h(B_g^*)$ by calling *SIGNB*, and then $\sigma(L_j^*)$ and $\sigma(C_i)$ by evaluating them on the primitive sample point of the cell c, then finally $\sigma_h(J_i)$ from $\sigma_h(B_g^*)$ and $\sigma(L_j^*)$ and $\sigma(C_i)$ by applying Equations 6, 7, 8, 9, and 10 successively. In case $l(c) = 0$, we first compute $\sigma_h(B_j)$ by calling *SIGNB*, and then $\sigma(C_i)$ (C_i are integers), then finally $\sigma_h(J_i)$ from $\sigma_h(B_j)$ and $\sigma(C_i)$ by applying Equations 9 and 10 successively.

The subalgorithm *EVALTV* (Fig 6), given the signs of the projection polynomials on the children ($\sigma_h(J_i)$), tries to evaluate the truth values of the children as discussed in Section 2.4.

The subalgorithm *PRPTV* (Fig 7) simply implements Theorems 2.2 and 2.4.

[4]For the details on how to compute multiplicity matrices along with squarefree bases, see Collins & Hong (1990) .

$F' \leftarrow QEPCAD(F^*)$

Quantifier Elimination by Partial Cylindrical Algebraic Decomposition.

Input : F^* is a quantified formula $(Q_{f+1}x_{f+1})\cdots(Q_r x_r)F(x_1,\ldots,x_r)$, where $0 \leq f \leq r$ and F is a quantifier-free formula.

Output : $F' = F'(x_1,\ldots,x_f)$ is a quantifier-free formula equivalent to F^*.

(1) [Projection.] Extract the integral polynomials $A = (A_1,\ldots,A_d)$ from the quantifier-free formula F. Compute the projections of A of all orders by calling $\hat{J} \leftarrow PROJM(A)$.

(2) [Initialization.] Initialize the partial CAD D as the single cell of D_0 whose truth value is set to be *undetermined*.

(3) [Choice.] If there is no candidate cell in the partial CAD D, go to Step 8. Choose a candidate cell c from the partial CAD D by calling $c \leftarrow CHOOSE(D)$.

(4) [Conversion.] If $l(c) > 1$ and c is a section, convert the extended sample point of c into a primitive one by using the *NORMAL* and *SIMPLE* algorithms.

(5) [Augmentation.] Augment the partial CAD D through constructing the children of the cell c by calling $T \leftarrow CCHILD(D,c,\hat{J})$. If $A^{[l(c)+1]} = \emptyset$, go to Step 3.

(6) [Trial Evaluation.] Try to determine the truth values of the children of the cell c by calling $EVALTV(D,c,F,A,T,\hat{J})$.

(7) [Propagation.] Determine the truth value of c , if possible and if it is, the truth values of as many ancestors of c as possible, removing from the partial CAD D the subtrees of each cell whose truth value is thus determined, by calling $PRPTV(D,c,Q,f)$. Go to Step 3.

(8) [Solution.] Let S be the set of all leaves whose truth values are *true*. Set $W \leftarrow \biguplus_{c \in S} c \times R^{f-l(c)}$. (Now a defining formula F' for the set W is equivalent to the input formula F^*.) □

Figure 1. Algorithm QEPCAD

$\hat{J} \leftarrow PROJM(A)$

Projection with Multiplicity information.

Input : $A = (A_1, \ldots, A_d)$, $d \geq 1$, is a list of r-variate integral polynomials of positive degree. $r \geq 1$.

Output : $\hat{J} = (\hat{J}_1, \ldots, \hat{J}_r)$ where each $\hat{J}_k$ is a list of 5-tuples (J, C, P, B, E); $J = (J_1, \ldots, J_n)$ is $Proj^{r-k}(A)$. $C = (C_1, \ldots, C_n)$ where C_i is the content of J_i. $P = (P_1, \ldots, P_n)$ where P_i is the primitive part of J_i. Note that the contents are signed so that $J_i = C_i P_i$. $B = (B_1, \ldots, B_m)$ is a finest square-free basis of P. $E = (E_{i,j})$ is the matrix of the multiplicities of B in P such that $P_i = \prod_{j=1}^{m} B_j^{E_{i,j}}$ for $1 \leq i \leq n$.

```
(1) [Initialize.] Set $J \leftarrow A$. Set $k \leftarrow r$.

(2) [Compute $C, P, B$ and $E$ from $J$.] Compute the contents $C$ and the primitive parts
    $P$ of $J$. Compute the finest square-free basis $B$ of the primitive parts $P$, obtaining the
    multiplicity matrix $E$ also. Set $\hat{J}_k \leftarrow (J, C, P, B, E)$.

(3) [Are we done?] If $k = 1$, return.

(4) [Compute $Proj(J)$ from $B$ and $C$.] Set $J \leftarrow C \cup PROJ(B)$. Set $k \leftarrow k - 1$. Go to
    Step 2. □
```

Figure 2. Algorithm PROJM

$T \leftarrow CCHILD(D, c, \hat{J})$

Construct the Children of a given cell.

Input : D is a partial CAD. c is a candidate cell of D. $\hat{J}$ is a structure produced by *PROJM*.

Output: The partial cad D is augmented with the children of the cell c. Let k be the level of the cell c. Let (J, C, P, B, E) be the $(k+1)$-th element of the list $\hat{J}$. Let $B = (B_1, \ldots, B_m)$. Then T is a 6-tuple $(A^*, L^*, P^*, B^*, E^*, I^*)$; Case $k = 0$. A^*, L^*, P^*, B^* and E^* are not computed. $I^* = (I_1, b_1, \ldots, I_u, b_u)$ where $I_1 < I_2 < \cdots < I_u$ are disjoint isolating intervals for all the real roots of $\prod_{j=1}^{m} B_j$ and each b_i is the unique B_j which has a root in I_i. Case $k > 0$. Let s be the primitive sample point of the cell c. $A^* = (A_1^*, \ldots, A_m^*)$ where each $A_j^* = B_j(s, x_{k+1})$. $L^* = (L_1^*, \ldots, L_m^*)$ where each L_j^* is the leading coefficient of A_j^*. $P^* = (P_1^*, \ldots, P_m^*)$ where each P_j^* is the monic associate of A_j^*. $B^* = (B_1^*, \ldots, B_l^*)$ is a square-free basis of P^*. $E^* = (E_{j,g}^*)$ is the matrix of the multiplicities of B^* in P^* such that $P_j^* = \prod_{g=1}^{l} B_g^{*E_{j,g}^*}$ for $1 \le j \le m$. $I^* = (I_1, b_1, \ldots, I_u, b_u)$ where $I_1 < I_2 < \cdots < I_u$ are disjoint isolating intervals for all the real roots of $\prod_{g=1}^{l} B_g^*$ and each b_i is the unique B_g^* which has a root in I_i.

(1) [Setup.] Let k be the level of the cell c. Let (J, C, P, B, E) be the $(k+1)$-th element of the list $\hat{J}$. If $k > 0$, go to Step 3.

(2) [If c is the root cell.] Isolate the real roots of the finest square-free basis B, obtaining I^*. Construct the children cells and the primitive sample point for each child. (A^*, L^*, P^*, B^* and E^* are not computed.) Set $T \leftarrow (A^*, L^*, P^*, B^*, E^*, I^*)$. Return.

(3) [If c is not the root cell.] Substitute the primitive sample point of the cell c into the square-free basis B, obtaining A^*. Compute the leading coefficients L^* and the monic associates P^* of A^*. Compute the square-free basis B^* of the monic polynomials P^*, also obtaining the multiplicity matrix E^*. Isolate the real roots of the square-free basis B^*, obtaining I^*. Construct the children cells and the sample point for each child. (For sectors construct primitive sample points and for sections construct extended sample points.) Set $T \leftarrow (A^*, L^*, P^*, B^*, E^*, I^*)$. Return. □

Figure 3. Algorithm CCHILD

$\Sigma \leftarrow SIGNB(I, B)$

Sign matrix of a square-free Basis.

Input : $B = (B_1, \ldots, B_v)$, $v \geq 0$, is a square-free basis. $I = (I_1, b_1, \ldots, I_u, b_u)$, $u \geq 0$, where $I_1 < I_2 < \cdots < I_u$ are disjoint isolating intervals for all the real roots of $\prod_{j=1}^{v} B_j$ and each b_i is the unique B_k which has a root in I_i.

Output : Σ is a $(2u+1)$ by v matrix $(\Sigma_{i,j})$ where $\Sigma_{i,j}$ is the sign of B_j in the i-th cell.

(1) [The rightmost cell. (i.e. the $(2u+1)$-th cell.)] Set $\Sigma_{2u+1,j} \leftarrow 1$ for $1 \leq j \leq v$. Set $i \leftarrow u$.

(2) [Are we done?] If $i = 0$, return.

(3) [The $(2i)$-th and $(2i-1)$-th cells.] Let k be such that $B_k = b_i$. For $j = 1, \ldots, v$ do { If $j = k$, set $\Sigma_{2i,j} \leftarrow 0$ and set $\Sigma_{2i-1,j} \leftarrow -\Sigma_{2i+1,j}$. Otherwise, set $\Sigma_{2i,j} \leftarrow \Sigma_{2i+1,j}$ and set $\Sigma_{2i-1,j} \leftarrow \Sigma_{2i+1,j}$. }. Set $i \leftarrow i - 1$. Go to Step 2. □

Figure 4. Algorithm SIGNB

$\Sigma \leftarrow$ *SIGNJ*$(c, T, \hat{J})$

Sign matrix of Projection polynomials.

Input : c is a cell in D. $\hat{J}$ is a structure produced by *PROJM*. T is a structure produced by *CCHILD*.

Output : Σ is a w by n matrix $(\Sigma_{h,i})$ where $\Sigma_{h,i}$ is the sign J_i in the h-th child of c. (For w,n, and J see Step 1.)

In this algorithm, $\sigma(X)$ stands for the sign of X on the cell c, and $\sigma_i(X)$ stands for the sign of X on the i-th child of the cell c.

(1) [Setup.] Let k be the level of the cell c. Let the $(k+1)$-th element of $\hat{J}$ be (J, C, P, B, E) where $J = (J_1, \ldots, J_n)$, $C = (C_1, \ldots, C_n)$, $P = (P_1, \ldots, P_n)$, $B = (B_1, \ldots, B_m)$, and E is a n by m matrix. Let T be $(A^*, L^*, P^*, B^*, E^*, I^*)$ where $A^* = (A_1^*, \ldots, A_m^*)$, $L^* = (L_1^*, \ldots, L_m^*)$, $P^* = (P_1^*, \ldots, P_m^*)$, $B^* = (B_1^*, \ldots, B_l^*)$, and E is a m by l matrix. Let $(c_1, \ldots, c_w)$ be the children of the cell c. If $k > 0$, go to Step 3.

(2) [If c is the root cell.] Compute $\sigma_h(B_j)$ for $1 \le j \le m$ and $1 \le h \le w$ by calling $\Sigma \leftarrow$ *SIGNB*(I^*, B), where $\Sigma_{h,j} = \sigma_h(B_j)$. Compute $\sigma(C_i)$ for $1 \le i \le n$. For $h = 1, 2, \ldots, w$ do { Set $\sigma_h(P_i) \leftarrow \prod_{j=1}^{m} \sigma_h(B_j)^{E_{i,j}}$ for $1 \le i \le n$. Set $\sigma_h(J_i) \leftarrow \sigma(C_i)\sigma_h(P_i)$ for $1 \le i \le n$. }. Return.

(3) [If c is not a root cell.] Compute $\sigma_h(B_g^*)$ for $1 \le g \le l$ and $1 \le h \le w$ by calling $\Sigma^* \leftarrow$ *SIGNB*(I^*, B^*), where $\Sigma_{h,g}^* = \sigma_h(B_g^*)$. Compute $\sigma(C_i)$ for $1 \le i \le n$ by evaluating C_i on the primitive sample point of the cell c. Compute $\sigma(L_j^*)$ for $1 \le j \le m$ by evaluating L_j^* on the primitive sample point of the cell c. For $h = 1, 2, \ldots, w$ do { Set $\sigma_h(P_j^*) \leftarrow \prod_{g=1}^{l} \sigma_h(B_g^*)^{E_{j,g}^*}$ for $1 \le j \le m$. Set $\sigma_h(A_j^*) \leftarrow \sigma(L_j^*)\sigma_h(P_j^*)$ for $1 \le j \le m$. Set $\sigma_h(B_j) \leftarrow \sigma_h(A_j^*)$ for $1 \le j \le m$. Set $\sigma_h(P_i) \leftarrow \prod_{j=1}^{m} \sigma_h(B_j)^{E_{i,j}}$ for $1 \le i \le n$. Set $\sigma_h(J_i) \leftarrow \sigma(C_i)\sigma_h(P_i)$ for $1 \le i \le n$. }. Return. □

Figure 5. Algorithm SIGNJ

$EVALTV(D, c, F, A, T, \hat{J})$

Evaluate Truth Values.

Input : D is a partial CAD. c is a cell in D. $F = F(x_1, \ldots, x_r)$ is a quantifier-free formula. $A = (A_1, \ldots, A_d)$ is the list of polynomials occurring in F. $\hat{J}$ is a structure produced by *PROJM*. T is a structure produced by *CCHILD*.

Output : the truth values of some children of c might be determined, resulting in modification of D.

In this algorithm, $\sigma_i(X)$ stands for the sign of X on the i–th child of the cell c.

(1) [Setup.] Let k be the level of the cell c. Let the $(k+1)$–th element of $\hat{J}$ be (J, C, P, B, E), where $J = (J_1, \ldots, J_n)$. Let $(c_1, \ldots, c_w)$ be the children of the cell c.

(2) [Compute the signs of the projection J.] Compute $\sigma_h(J_i)$ for $1 \leq h \leq w$ and $1 \leq i \leq n$ by calling $\Sigma \leftarrow$ *SIGNJ*$(c, T, \hat{J})$, where $\Sigma_{h,i} = \sigma_h(J_i)$.

(3) [Get the signs of the input polynomials A.] For $h = 1, 2, \ldots, w$ and $p = 1, 2, \ldots, d$ do { If there exists i such that $A_p = J_i$, set $\sigma_h(A_p) \leftarrow \sigma_h(J_i)$. }.

(4) [Try to evaluate the truth values of the children of c.] For $h = 1, 2, \ldots, w$ do { Try to determine $v(c_h)$ by trying to evaluate the truth value of the quantifier-free formula F on the cell c_h from $\sigma_h(A_1), \ldots, \sigma_h(A_d)$. }. Return. □

Figure 6. Algorithm EVALTV

PRPTV(D,c,Q,f)

Propagate Truth Value.

Input : D is a partial CAD. c is a cell in D. $Q = (Q_{f+1}, \ldots, Q_r)$ is the list of the quantifiers occurring in the quantified formula. f is the number of free variables.

Output : The truth values of the cell c and some of its ancestors might be determined. The descendants of each cell whose truth value has been determined through the propagation are removed from the partial CAD D.

(1) [Initialize.] Set $k \leftarrow$ the level of the cell c. Set $c' \leftarrow c$.

(2) [Are we done with propagating on or above the free variable space?] If $k < f$, go to Step 5.

(3) [Propagate on or above the free variable space.] **If** the truth value of every child of c' is *true* **then** set $v(c') \leftarrow$ *true* **else if** the truth value of every child of c' is *false* **then** set $v(c') \leftarrow$ *false* **else if** there exists a *true* child of c' and $Q_{i+1} = \exists$ **then** set $v(c') \leftarrow$ *true* **else if** there exists a *false* child of c' and $Q_{i+1} = \forall$ **then** set $v(c') \leftarrow$ *false* **else** return.

(4) [Remove the descendants of c' and loop.] Remove the descendants of the cell c' from the partial CAD D. Set $k \leftarrow k - 1$. Set $c' \leftarrow$ the parent of c'. Go to Step 2.

(5) [Are we done with propagating below the free variable space?] If $k < 0$, return.

(6) [Propagate below the free variable space.] **If** the truth value of every child of c' is *true* **then** set $v(c') \leftarrow$ *true* **else if** the truth value of every child of c' is *false* **then** set $v(c') \leftarrow$ *false* **else** return.

(7) [Remove the descendants of c' and loop.] Remove the descendants of the cell c' from the partial CAD D. Set $k \leftarrow k - 1$. Set $c' \leftarrow$ the parent of c'. Go to Step 5. □

Figure 7. Algorithm PRPTV

4 Strategy for Cell Choice

In this section, as promised, we discuss the algorithm *CHOOSE*, whose task is to choose a candidate cell from a partial CAD. We begin by recallng that the main algorithm *QEPCAD* repeatedly chooses a candidate cell—over which stack construction, trial evaluation and propagation are carried out—until there are no more candidate cells. Let c_i be the i–th cell chosen by Algorithm *CHOOSE*. We will call the list $(c_1, \ldots, c_n)$ a *choice path.* Note that for a given input formula many different choice paths will be possible and that each may require a different amount of time.

We implemented our partial CAD construction system as an interactive environment where a user can manually choose a candidate cell or invoke an already built-in strategy algorithm. When cells are chosen manually, a user can query for diverse information, such as the current partial CAD, the sample points, the signs of basis polynomials in a specified cell, and so on.[5]

In Fig 8 we present several cell-choice strategies devised for several problem-classes: the strategy HL-LI for collision problems from robot motion planning (Buchberger *et al.*, 1989), the strategy SR-HL-LI for consistency of a system of polynomial strict inequalities (McCallum 1987), the strategy TC-LD-HL-GI for termination proof of term rewrite systems (Lankford 1979, Huet & Oppen 1980), and the strategy TC-LD-HL-LI for any other unclassified problems, where HL stands for higher level first, LI for lesser index first, GI for greater index first, SR for sector first, TC for trivial conversion first, and LD for lesser degree of minimal polynomial first.

These strategies were based on a limited amount of experience and study, but the experiments showed that they significantly reduced the amount of required computation time.

The algorithm *ORDER* defines a total ordering among cells, in the way that it compares two input cells according to the ordering and returns the greater cell. Each strategy uses its own version of the algorithm *ORDER* (i.e. a definition of ordering among cells). The algorithm *CHOOSE* finds the greatest candidate cell by utilizing the algorithm *ORDER* in an obvious way.

In the algorithm *ORDER*, the following functions are used.

$l(c)$ denotes the level of the cell c.

$s(c)$ denotes the cell type of the cell c. It is 0 if c is a sector, and 1 if c is a section.

$i(c)$ denotes the index of the cell c. $i(c_1) > i(c_2)$ if $i(c_1)$ is after $i(c_2)$ lexicographically.

$d(c)$ denotes the presumed degree of the minimal polynomial of the primitive sample point of the cell c. More specifically, if the cell c already has a primitive sample point, $d(c)$ is m where m is the degree of the corresponding minimal polynomial. Otherwise, $d(c)$ is mm' where m is the degree of the minimal polynomial for the non-last coordinates and m' is the degree of the algebraic polynomial for the last coordinate.

$p(c)$ denotes the triviality of the conversion of the sample point of the cell c. It is 1 if the conversion is definitely trivial or not needed, and 0 if the conversion *can* be nontrivial. More specifically, $p(c)$ is 0 if the cell c has an extended sample point such that both

[5]The user's manual for the partial CAD system will be available to any interested researchers in the very near future.

```
          $c \leftarrow ORDER(c_1, c_2)$        (Strategy: TC-LD-HL-LI)
(1) [By the complexity of conversion. Trivial conversion first. (TC)]
    if $p(c_1) > p(c_2)$ then { set $c \leftarrow c_1$. return.}
    if $p(c_1) < p(c_2)$ then { set $c \leftarrow c_2$. return.}
(2) [By the degree of the minimal polynomial. Less degree first. (LD)]
    if $d(c_1) < d(c_2)$ then { set $c \leftarrow c_1$. return.}
    if $d(c_1) > d(c_2)$ then { set $c \leftarrow c_2$. return.}
(3) [By the level. Higher Level first. (HL)]
    if $l(c_1) > l(c_2)$ then { set $c \leftarrow c_1$. return.}
    if $l(c_1) < l(c_2)$ then { set $c \leftarrow c_2$. return.}
(4) [By the index. Less Index first. (LI)]
    if $i(c_1) < i(c_2)$ then { set $c \leftarrow c_1$. return.}
    if $i(c_1) > i(c_2)$ then { set $c \leftarrow c_2$. return.} □

          $c \leftarrow ORDER(c_1, c_2)$        (Strategy: TC-LD-HL-GI)
The same as TC-LD-HL-LI except replacing Step 4 with the following.
(4) [By the index. Greatest Index first. (GI)]
    if $i(c_1) > i(c_2)$ then { set $c \leftarrow c_1$. return.}
    if $i(c_1) < i(c_2)$ then { set $c \leftarrow c_2$. return.} □

          $c \leftarrow ORDER(c_1, c_2)$        (Strategy: SR-HL-LI)
The same as TC-LD-HL-LI except replacing Steps 1 and 2 with the following.
(1) [By the cell type. Sector first. (SR)]
    if $s(c_1) < s(c_2)$ then { set $c \leftarrow c_1$. return.}
    if $s(c_1) > s(c_2)$ then { set $c \leftarrow c_2$. return.}

          $c \leftarrow ORDER(c_1, c_2)$        (Strategy: HL-LI)
The same as TC-LD-HL-LI except omitting Steps 1 and 2.


          $c \leftarrow CHOOSE(D)$
(1) Let $c_1, \ldots, c_n$ be the candidate cells in $D$. Set $c \leftarrow c_1$.
    For $i = 2, \ldots, n$ do set $c \leftarrow ORDER(c, c_i)$ □.
```

Figure 8. Cell Choice Strategies

the minimal polynomial for the non-last coordinates and the algebraic polynomial for the last coordinate are nonlinear. Otherwise, $p(c)$ is 1.

Now we describe the motivation that led to the strategies presented here. Let us begin by TC-LD-HL-LI, the strategy for any unclassified problems. The motive for TC-LD is to choose a cell on which stack construction might be cheapest. The motive for HL is to choose a cell that might lead to truth evaluations earliest. LI is not significant, it could be replaced with a random choice.

In termination proof of term rewrite systems, GI is superior to LI for reasons explained in Hong & Kuechlin (1990). This is the motive for using GI in the strategy TC-LD-HL-GI.[6]

In consistency problems of a system of polynomial strict inequalities, as McCallum (1987) points out, we only need to consider sectors. This is the motive for using SR in the strategy SR-HL-LI.[7]

In collision problems, we are interested in whether several moving objects would collide, and if so, we may also be interested in finding the time of the earliest collision. The strategy HL-LI chooses candidate cells in such an order that this time can be easily detected.

5 Illustration

In this section we illustrate our algorithm by a simple collision problem from robot motion planning.

Consider two semi-algebraic objects: a circle and a square (See Fig 9). The circle has diameter 2 and is initially centered at $(0,0)$ and is moving with the velocity $v_x = 1$ and $v_y = 0$. The square has side-length 2 and is initially centered at $(0,-8)$ and is moving with the velocity $v_x = 17/16$ and $v_y = 17/16$. Now we want to decide if these two objects would collide.

The moving circle can be described algebraically as:

$$(x-t)^2 + y^2 \le 1$$

The moving square can be described algebraically as:

$$-1 \le x - \frac{17}{16}t \le 1 \quad \wedge \quad -9 \le y - \frac{17}{16}t \le -7$$

From these we can express the collision problem as a decision problem of a sentence in elementary algebra and geometry:

$$\begin{aligned}(\exists t)(\exists x)(\exists y)(\quad & t > 0 \\ \wedge\ & x - \frac{17}{16}t \ge -1 \\ \wedge\ & x - \frac{17}{16}t \le 1 \\ \wedge\ & y - \frac{17}{16}t \ge -9\end{aligned}$$

[6]In order to fully utilize this strategy, we also need to modify the propagation step slightly as discussed in Section 7.

[7]In order to fully utilize this strategy, we also need to modify the propagation step slightly as discussed in Section 7.

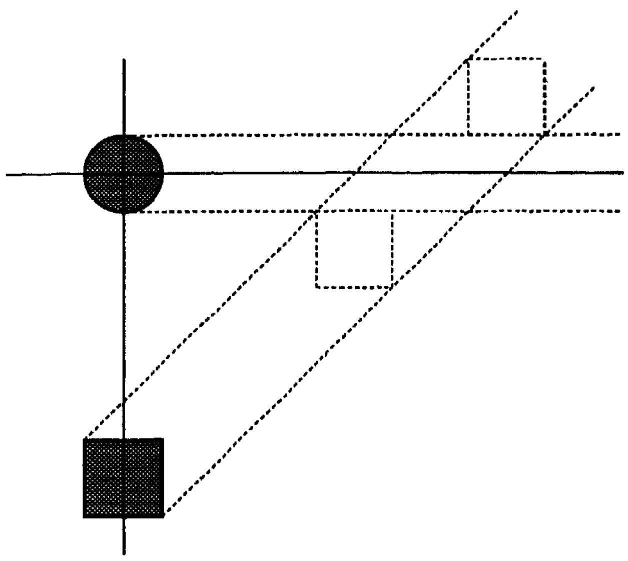

Figure 9. Collision Problem

$$\begin{aligned} \wedge\ & y - \frac{17}{16}t \leq -7 \\ \wedge\ & (x-t)^2 + y^2 \leq 1\ ) \end{aligned}$$

We can refine this sentence by observing that collision can occur only when the top side of the square is above or on the line $y = -1$ and the bottom side of the square is below or on the line $y = 1$. Translating this observation into a restriction on the range of t, we get the following sentence:

$$\begin{aligned} (\exists t)(\exists x)(\exists y)(\ \ & \frac{17}{16}t \geq 6 \\ \wedge\ & \frac{17}{16}t \leq 10 \\ \wedge\ & x - \frac{17}{16}t \geq -1 \\ \wedge\ & x - \frac{17}{16}t \leq 1 \\ \wedge\ & y - \frac{17}{16}t \geq -9 \\ \wedge\ & y - \frac{17}{16}t \leq -7 \\ \wedge\ & (x-t)^2 + y^2 \leq 1\ ) \end{aligned}$$

We entered this sentence into an implementation (Arnon 1981) of the original CAD algorithm, and it reported collision, after constructing 25 cells in 1–space, 263 cells in 2–space, and 1795 cells in 3–space, taking 1 hour and 44 minutes on a Sun3/50 running Unix.

We tried our partial CAD algorithm on the same sentence using the cell choice strategy HL-LI. It reported collision, after constructing 25 cells in 1–space, 11 cells in 2–space, and 25 cells in 3–space, taking 20.5 seconds (7.8 seconds for projection, 12.7 seconds for stack construction).

Below we present the trace of the actual computation carried out by the partial CAD system. In order to save space the trace output has been compressed. In this trace, a partial CAD is represented as a tree of cells, where each cell is represented by its cell-index along with one character indicating its truth value (`T,F,?` respectively for *true*, *false*, *undetermined*). The text in italics consists of comments inserted afterwards.

```
-----------------------------------------------------------
Enter a prenex formula:
(Et)(Ex)(Ey) (  17/16 t >= 6
              & 17/16 t <= 10
              & x - 17/16 t >= -1
              & x - 17/16 t <=  1
              & y - 17/16 t >= -9
              & y - 17/16 t <= -7
              & y^2 + x^2 - 2 t x + t^2 <= 1 )

-----------------------------------------------------------
The initial partial CAD:
() ?
```

This indicates that the initial partial CAD consists of a root cell whose truth value is undetermined.

```
-----------------------------------------------------------
The cell chosen: ()
Constructing children of the cell ()......
Trying to evaluate truth values of the children of ()......

The current partial CAD over ():
()---(1) F
  ---(2) F
  ---(3) F
  ---(4) F
  ---(5) F
  ---(6) ?
  ---(7) ?
  ---(8) ?
  ---(9) ?
  ---(10) ?
```

```
   ---(11) ?
   ---(12) ?
   ---(13) ?
   ---(14) ?
   ---(15) ?
   ---(16) ?
   ---(17) ?
   ---(18) ?
   ---(19) ?
   ---(20) ?
   ---(21) ?
   ---(22) ?
   ---(23) F
   ---(24) F
   ---(25) F
```

The truth values of the cells (1), (2), (3), (4), *and* (5) *are already determined to be false because* $\frac{17}{16}t \not> 6$ *for these cells. Likewise the truth values of the cells* (23), (24), *and* (25) *are already determined to be false because* $\frac{17}{16}t \not< 10$ *for these cells. Therefore we do not need to build stacks over them, resulting in reduction of the computation.*

```
--------------------------------------------------------------
The cell chosen: (6)
Constructing children of the cell (6)......
Trying to evaluate truth values of the children of (6)......

The current partial CAD over (6):
(6)---(6,1) F
   ---(6,2) F
   ---(6,3) F
   ---(6,4) ?
   ---(6,5) ?
   ---(6,6) ?
   ---(6,7) ?
   ---(6,8) ?
   ---(6,9) ?
   ---(6,10) ?
   ---(6,11) F
```

The truth values of the cells (6, 1), (6, 2), *and* (6, 3) *are already determined to be false because* $x - \frac{17}{16}t \not> -1$ *for these cells. Likewise the truth value of the cell* (6, 11) *is already determined to be false because* $x - \frac{17}{16}t \not< 1$ *for this cell.*

```
--------------------------------------------------------------
The cell chosen: (6,4)
Constructing children of the cell (6,4)......
```

```
Trying to evaluate truth values of the children of (6,4)......

The current partial CAD over (6,4):
(6,4)--(6,4,1) F
     --(6,4,2) F
     --(6,4,3) F
     --(6,4,4) F
     --(6,4,5) F
     --(6,4,6) F
     --(6,4,7) F
     --(6,4,8) F
     --(6,4,9) F

Propagating the truth values....

The current partial CAD over (6):
(6)---(6,1) F
   ---(6,2) F
   ---(6,3) F
   ---(6,4) F
   ---(6,5) ?
   ---(6,6) ?
   ---(6,7) ?
   ---(6,8) ?
   ---(6,9) ?
   ---(6,10) ?
   ---(6,11) F
```

The truth value of the cell (6, 4) *has been determined to be false because the truth values of all the children of* (6, 4) *are false.*

```
-------------------------------------------------------------
The cell chosen: (6,5)
```

Actually it is unnecessary to construct a stack over the cell (6,5). One can give an argument that because the objects in this problem are closed sets and because the truth value of the adjacent cell (6,4) is false, the truth value of the cell (6,5) must also be false.

```
Constructing children of the cell (6,5)......
Trying to evaluate truth values of the children of (6,5)......

The current partial CAD over (6,5):
(6,5)--(6,5,1) F
     --(6,5,2) F
     --(6,5,3) F
     --(6,5,4) F
```

```
      --(6,5,5) F
      --(6,5,6) F
      --(6,5,7) F
      --(6,5,8) F
      --(6,5,9) F

Propagating the truth values....

The current partial CAD over (6):
(6)---(6,1) F
   ---(6,2) F
   ---(6,3) F
   ---(6,4) F
   ---(6,5) F
   ---(6,6) ?
   ---(6,7) ?
   ---(6,8) ?
   ---(6,9) ?
   ---(6,10) ?
   ---(6,11) F
```

The truth value of the cell $(6,5)$ *has been determined to be false because the truth values of all the children of* $(6,5)$ *are false.*

```
-------------------------------------------------------
The cell chosen: (6,6)
Constructing children of the cell (6,6)......
Trying to evaluate truth values of the children of (6,6)......

The current partial CAD over (6,6):
(6,6)--(6,6,1) F
     --(6,6,2) F
     --(6,6,3) F
     --(6,6,4) T
     --(6,6,5) F
     --(6,6,6) F
     --(6,6,7) F

Propagating the truth values....

The current partial CAD over ():
() T
```

Now the truth value of the the root cell $()$ *is determined to be true and so the input sentence is also true, which tells us that the two objects will collide.*

The truth value of the root cell $()$ *is determined to be true since the truth value of the cell* $(6,6,4)$ *is true and the quantifiers* Q_3, Q_2, *and* Q_1 *are all existential.*

It is important to note that we did not have to build stacks over the cells (6, 7), (6, 8), (6, 9), (6, 10), *and* (7) *through* (22).

```
---------------------- Statistics -----------------------
                    Level-1    Level-2   Level-3
Number of stacks:    1           1         3
Number of cells:    25          11        25

Time taken for projection: 7.816 seconds
Time taken for choosing candidate cells: 0.250 seconds
Total time taken: 20.516 seconds
----------------------------------------------------------
```

6 Empirical Results

In this section we present empirical comparisons between the partial CAD method and the original method on several problems from diverse application areas.[8]

The original method has been implemented by Arnon (1981) in ALDES/SAC-2 computer algebra system (Collins & Loos 1980). We used this implementation for our experiments, but with a slight modification as follows. Arnon's implementation avoids some redundant computations in CAD construction by partitioning the set of cells of a CAD into disjoint subsets called *conjugacy equivalence classes* and by carrying out those computations only once for each class. This partitioning is done for CAD's of all levels. However, this process is not needed for a CAD of the last level since we do not need to build stacks over it. Experience also shows that this process can be very time consuming. Therefore we modified Arnon's implementation so that it does not carry out partitioning at the last level.

We implemented the partial CAD method also in ALDES/SAC-2. But we incorporated two more improvements into the implementation. First, NORMAL and SIMPLE operations are bypassed in certain trivial conversions of sample points. Let c be a cell of D_k which has an extended sample point consisting of: (1) a real algebraic number α and a $(k-1)$-tuple $(b_1, \ldots, b_{k-1})$ of elements of $Q(\alpha)$, and (2) a non-zero monic squarefree polynomial $g(x) \in Q(\alpha)[x]$, and an isolating interval for a real root β of $g(x)$. Now if $\deg(g) = 1$ (i.e. $g(x) = x + a_0$), we trivially have $\beta = -a_0 \in Q(\alpha)$. In this case, therefore, we can get a primitive sample point of the cell c without carrying out the NORMAL and SIMPLE operation, consisting of the real algebraic number α and the k-tuple $(b_1, \ldots, b_{k-1}, -a_0)$ of elements of $Q(\alpha)$.

Second, several databases are used in order to avoid some redundant computations. As mentioned earlier Arnon's implementation uses conjugacy equivalence classes for this purpose. This method, however, is not compatible with our partial CAD method for obvious reasons, and therefore we chose instead a database approach. When we need to compute a certain result, we first search an appropriate database for an earlier instance of the same computation. If the search is successful, the result is retrieved and used; otherwise

[8] ADDED DURING PROOF: More recent experimental results for these and various other problems are found in Hong (1990b, 1991). For example, in Hong (1990b) the x-axis ellipse problem and the quartic problem (Arnon & Mignotte 1988) are shown to be solved completely mechanically in less than a minute.

we compute the result and enter it into the database. The details will be discussed in a subsequent paper.

Table 1 shows the performance comparisons on several problems from diverse application areas. In this table, T is the total computation time in seconds and C_i is the number of cells constructed in i-space. For each problem three rows are given, the top one for the original method with the slight modification as mentioned above, the bottom one for our method, and the middle one for our method as modified to construct a full CAD.

Therefore the differences between the top one and the middle one are due to (1) the avoidance of NORMAL and SIMPLE operations at the last level, (2) the use of databases instead of conjugacy equivalence class computation, and (3) the bypassing of NORMAL and SIMPLE operation in trivial conversions of sample points. The differences between the middle and the bottom one are due to the construction of only a partial CAD through the use of propagation and trial truth evaluation.

All the experiments were carried out on a SUN3/50 running Unix using 4 megabytes of memory for lists.

Collision Problem

This problem is same as the one used in Section 5, except that the square moves with the velocity $v_x = 15/16$, $v_y = 15/16$.

$$\begin{aligned}(\exists t)(\exists x)(\exists y)(\quad & \frac{15}{16}t \geq 6 \\ \wedge \quad & \frac{15}{16}t \leq 10 \\ \wedge \quad & x - \frac{15}{16}t \geq -1 \\ \wedge \quad & x - \frac{15}{16}t \leq 1 \\ \wedge \quad & y - \frac{15}{16}t \geq -9 \\ \wedge \quad & y - \frac{15}{16}t \leq -7 \\ \wedge \quad & (x-t)^2 + y^2 \leq 1 \ )\end{aligned}$$

Consistency in Strict Inequalities (McCallum 1987)

This problem decides whether the intersection of the open ball with radius 1 centered at the origin and the open circular cylinder with radius 1 and axis the line $x = 0$, $y+z = 2$ is nonempty.

$$(\exists x)(\exists y)(\exists z)(x^2 + y^2 + z^2 < 1 \ \wedge \ x^2 + (y+z-2)^2 < 1)$$

Termination of Term Rewrite System (Lankford 1979, Huet & Oppen 1980)

This problem decides whether we should orient the equation $(xy)^{-1} = y^{-1}x^{-1}$ into $(xy)^{-1} \rightarrow y^{-1}x^{-1}$ in order to get a terminating rewrite system for group theory. It uses a polynomial interpretation: $xy \Rightarrow x + 2xy$, $x^{-1} \Rightarrow x^2$, and $1 \Rightarrow 2$.

$$(\exists r)(\forall x)(\forall y)(x > r \wedge y > r \implies x^2(1+2y)^2 > y^2(1+2x^2))$$

Problem	Method	T	C_1	C_2	C_3	C_4	Strategy
Collision	original	5,255	25	263	1,795		
	full	585	25	263	1,795		
	partial	22	25	11	33		HL-LI
Consistency	original	4,961	11	57	365		
	full	179	11	57	365		
	partial	26	11	15	43		SR-HL-LI
Termination	original	333	17	177	1,099		
	full	254	17	177	1,099		
	partial	7	17	13	7		TC-LD-HL-GI
Collins	original	1,759	19	269	2,149		
Johnson	full	641	19	269	2,149		
	partial	228	19	142	524		TC-LD-HL-LI
Davenport	original	1,007	7	73	667	4,949	
Heintz	full	1,464	7	73	667	4,949	
	partial	217	7	73	649	486	TC-LD-HL-LI

Table 1. Comparisons

Collins and Johnson (Collins & Johnson 1989)

The formula below gives necessary and sufficient conditions on the complex conjugate roots $a \pm bi$ so that there exists a cubic polynomial with a single real r root in $(0,1)$ yet more than one variation is obtained.

$$\begin{array}{rl} (\exists r)(& 3a^2r + 3b^2r - 2ar - a^2 - b^2 < 0 \\ \wedge & 3a^2r + 3b^2r - 4ar + r - 2a^2 - 2b^2 + 2a > 0 \\ \wedge & a \geq 1/2 \\ \wedge & b > 0 \\ \wedge & r > 0 \\ \wedge & r < 1\ ) \end{array}$$

Davenport and Heintz (Davenport & Heintz 1989)

This formula is a special case of a more general formula used by Davenport and Heintz in order to show the time complexity of the quantifier elimination in elementary algebra and geometry.

$$(\exists c)(\forall b)(\forall a)((a = d \wedge b = c)\ \vee\ (a = c \wedge b = 1)\ \implies\ a^2 = b)$$

7 Conclusion

In this paper we developed a method called *partial CAD construction*, that completes quantifier elimination by a *partially* built CAD, utilizing several aspects of the input formula such as: the quantifiers, the Boolean connectives, and the absence of some variables from some polynomials occurring in the input formula.

The resulting algorithm was nondeterministic in nature, and thus we also presented several strategies that heuristically try to guide the algorithm along a cheap computational path. These strategies were based on very limited amount of experience and study, but even these crude strategies significantly reduce the amount of computation as the above experiments show. We hope that these strategies can be refined more in order to achieve further reduction of the required computation.

We can also sometimes make further improvement of our method by customizing the propagation algorithm for each problem-class. For example, in termination proof of term rewrite systems, we can determine the truth value of the root cell as soon as the truth value of the cell with the greatest index in a CAD of 1-space is determined (Hong & Kuechlin 1990) . Similarly, in consistency problems of polynomial strict inequalities, the truth value of a cell can be determined to be *false* as soon as the truth values of all the sector-children of the cell are known to be *false*.

Finally, we hope that our method and Arnon's clustering method (Arnon 1988) may be combined in the near future in order to take advantage of both methods.

The authors would like to thank a referee who made some constructive criticisms.

References

Arnon, D. S. (1981). Algorithms for the geometry of semi-algebraic sets. Technical Report 436, Computer Sciences Dept, Univ. of Wisconsin-Madison, Ph.D. Thesis.

Arnon, D. S. (1988). A cluster-based Cylindrical Algebraic Decomposition algorithm. *J. Symbolic Comp.*, **5** (1,2), 189–212.

Arnon, D. S., Collins, G. E., McCallum, S. (1984). Cylindrical Algebraic Decomposition I: The basic algorithm. *SIAM J. Comp.*, **13**, 865–877.

Arnon, D. S., Mignotte, M. (1988). On mechanical quantifier elimination for elementary algebra and geometry. *J. Symbolic Comp.*, **5** (1,2), 237–260.

Buchberger, B., Collins, G. E., Kutzler, B. (1989). Algebraic methods for geometric reasoning. Manuscript in preparation.

Collins, G. E. (1975). Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Lecture Notes In Computer Science*, **33**, 134–183, Springer-Verlag, Berlin.

Collins, G. E., Hong, H. (1990). Computing coarsest squarefree bases and multiplicities. Technical Report OSU-CISRC-5/90 TR13, Computer Science Dept, The Ohio State University.

Collins, G. E., Johnson, J. (1989). Quantifier elimination and the sign variation method for real root isolation. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, 264–271.

Collins, G. E., Loos, R. (1980). The ALDES-SAC2 computer algebra system. Technical report.

Davenport, J. H., Heintz, J. (1988) Real quantifier elimination is doubly exponential. *J.*

Symbolic Comp., **5** (1,2).

Hong, H. (1990a). An improvement of the projection operator in cylindrical algebraic decomposition. In *Proceedings of the 1990 International Symposium on Symbolic and Algebraic Computation*, 261–264.

Hong, H. (1990b). Improvements in CAD–based Quantifier Elimination. PhD thesis, The Ohio State University.

Hong, H. (1991). Collision problems by an improved CAD-based quantifier elimination algorithm. Technical Report RISC-Linz series no. 91-05.0, Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria.

Hong, H., Kuechlin, W. (1990). Termination proof of term rewrite system by cylindrical algebraic decomposition. Manuscript in preparation.

Huet, G., Oppen, D. (1980). Equations and rewrite rules. In *Formal Language Theory. Perspectives and Open Problems*, 349–405, Academic Press, New York.

Lankford, D. S. (1979). On proving term rewriting systems are Noetherian. Technical Report MTP-3, Math. Dept., Lousiana Tech. University.

Loos, R. (1982). Computing in algebraic extensions. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer Algebra; Algebraic and Symbolic Computation*, 173–188. Springer-Verlag.

McCallum, S. (1984). An Improved Projection Operator for Cylindrical Algebraic Decomposition. PhD thesis, University of Wisconsin-Madison.

McCallum, S. (1987). Solving polynomial strict inequalities using cylindrical algebraic decomposition. Technical Report 87-25.0, RISC-LINZ, Johannes Kepler University, A-4040 Linz, Austria (Europe).