An Incremental Algorithm for Computing Cylindrical Algebraic Decompositions

Changbo Chen, Marc Moreno Maza

ORCCA, University of Western Ontario (UWO)
London, Ontario, Canada
{cchen252,moreno}@csd.uwo.ca

Abstract

In this paper, we propose an incremental algorithm for computing cylindrical algebraic decompositions. The algorithm consists of two parts: computing a complex cylindrical tree and refining this complex tree into a cylindrical tree in real space. The incrementality comes from the first part of the algorithm, where a complex cylindrical tree is constructed by refining a previous complex cylindrical tree with a polynomial constraint. We have implemented our algorithm in Maple. The experimentation shows that the proposed algorithm outperforms existing ones for many examples taken from the literature.

1 Introduction

Cylindrical algebraic decomposition (CAD) is a fundamental tool in real algebraic geometry. It was invented by G.E. Collins in 1973 [15] for solving real quantifier elimination (QE) problems. In the last forty years, following Collins' original projection-lifting scheme, many enhancements have been performed in order to ameliorate the efficiency of CAD construction, including adjacency and clustering techniques [1], improved projection methods [26, 23, 9, 4], partially built CADs [17, 27, 33], improved stack construction [18], efficient projection orders [20], making use of equational constraints [16, 29, 7, 30], and so on. Moreover, CADs can be computed by several software packages, such as Qepcad [24, 5], Mathematica [33, 34], Redlog [21] and SyNRAC [25].

In [14], together with B. Xia and L. Yang, we presented a different way for computing CADs based on triangular decomposition of polynomial systems. In that paper, we introduced the concept of cylindrical decomposition of the complex space (CCD), from which a CAD can be easily derived. The concept of CCD is reviewed in Section 2. In the rest of the present paper, we use TCAD to denote CAD based on triangular decompositions while PCAD refers to CAD based on Collins' projection-lifting scheme.

The CCD part of TCAD can be seen as an enhanced projection phase of PCAD. However, w.r.t. PCAD (especially when the projection operator is using Collins' [15] or Hong's [23]), the "case discussion" scheme of TCAD avoids unnecessary computations that projection operator performs on unrelated branches. In addition, one observes that the reason why McCallum's [28] (including Brown's [4]) projection operators may fail for some examples is due to the fact that they are missing a "case discussion" scheme. McCallum's operator relies on the assumption that generically all coefficients of a polynomial[1] will not vanish simultaneously above a positive-dimensional component. If this assumption fails, then this operator is replaced by Collins-Hong projection-operator [23]. The fact that all coefficients of polynomial could vanish simultaneously above some component is never a problem in TCAD. For this reason, we view it as an improvement of previous works.

[1] More precisely, a multivariate polynomial regarded as a univariate one with respect to its main variable.

Trying to use sophisticated algebraic elimination techniques to improve CAD constructions is not a new idea. In papers [8, 39], the authors investigated how to use Gröbner bases to preprocess the input system in order to make the subsequent CAD computations more efficient. The main difference between these two works and the work of [14] is that the former approach is about preprocessing input for CAD while the latter one presents a different way of constructing CADs.

In [14], the focus was on how to apply triangular decomposition techniques to compute CADs. To this end, lots of existing high-level routines were used to facilitate explaining ideas. These high-level routines involve many black-boxes, which hide many unnecessary or redundant computations. As a result, the computation time of TCAD is much higher than that of PCAD, although TCAD computes usually less cells [10].

In the present paper, we abandon those black-boxes and compute TCAD from scratch. It turns out that the key solution for avoiding redundant computations is to compute CCD in an *incremental manner*. The same motivation and a similar strategy appeared in [32, 12] in the context of triangular decomposition of algebraic sets. The core operation of such an incremental algorithm is an Intersect operation, which refines an existing cylindrical tree w.r.t. a polynomial. We dedicate Section 4 to presenting a complete incremental algorithm for computing TCAD by means of this Intersect operation.

In [35], the author presented an algorithm for computing with semi-algebraic sets represented by cylindrical algebraic formulas. That algorithm also allows computing CAD in an incremental manner. The underlying technique is based on the projection-lifting scheme where one first computes projection factor sets by a global projection operator. In contrast, the incremental algorithm presented here, is conducted by refining different branches of an existing tree via GCD computations.

This Intersect operation can systematically take advantage of equational constraints. The problem of making use of equational constraints in CAD has been studied by many researchers [16, 29, 7, 30]. In Section 6, we provide a detailed discussion on how we solve this problem.

When applied to a polynomial system having finitely many complex solutions, our incremental CCD algorithm specializes into computing a triangular decomposition, say $\mathcal{D}$, such that the zero sets of the output regular chains are disjoint. Moreover, such a decomposition has no critical pairs in the sense of the equiprojectable decomposition algorithm of [19]. This implies that only the "Merge" part of the "Split & Merge" algorithm of [19] is required for turning $\mathcal{D}$ into an equiprojectable decomposition (which is a canonical representation of the input variety, once the variable order is fixed). Consequently, one could hope extending the notion of equiprojectable decomposition (and related algorithms) to positive dimension by means of our incremental CCD algorithm. This perspective can be seen as an indirect application of CAD to triangular decomposition.

As we shall review in Section 2, a CCD is encoded by a tree data-structure. Then each path of this tree is a simple system in the sense of [36, 37]. So the work presented here can also be used to compute a Thomas decomposition of a polynomial system [37, 2]. Moreover, the decomposition we compute is not only disjoint, but also cylindrically arranged.

The complexity of our algorithm cannot be better than doubly exponential in the number of variables [6]. So the motivation of our work is to suggest possible ways to improve the practical applicability of CAD. The benchmark in Section 7 shows that TCAD outperforms QEPCAD [24, 5] and Mathematica [33] for many well-known examples. The algorithm presented in this paper can support QE. We have realized a preliminary implementation of an algorithm for doing QE via TCAD. We will report on this work in a future paper.

2 Complex cylindrical tree

Throughout this paper, we consider a field $\mathbf{k}$ of characteristic zero and denote by $\mathbf{K}$ the algebraic closure of $\mathbf{k}$. Let $\mathbf{k}[\mathbf{x}]$ be the polynomial ring over the field $\mathbf{k}$ with ordered variables $\mathbf{x} = x_1 < \cdots < x_n$. Let $p \in \mathbf{k}[\mathbf{x}]$ be a non-constant polynomial and $x \in \mathbf{x}$ be a variable. We denote by $\deg(p, x)$ and $\mathrm{lc}(p, x)$ the degree and the leading coefficient of p w.r.t. x. The greatest variable appearing in p is called the *main variable*, denoted by $\mathrm{mvar}(p)$. The leading coefficient, the degree, the reductum of p w.r.t. $\mathrm{mvar}(p)$ are called the *initial*, the *main degree*, the *tail* of p; they are denoted by $\mathrm{init}(p)$, $\mathrm{mdeg}(p)$, $\mathrm{tail}(p)$ respectively. The integer k such that $x_k = \mathrm{mvar}(p)$ is called the *level* of the polynomial p. We denote by $\mathrm{der}(p)$ the derivative of p w.r.t. $\mathrm{mvar}(p)$. The notions presented below were introduced in [14] and they are illustrated at the beginning of Section 3.

Separation. Let C be a subset of $\mathbf{K}^{n-1}$ and $P \subset \mathbf{k}[x_1, \ldots, x_{n-1}, x_n]$ be a finite set of level n polynomials. We say that P *separates above* C if for each $\alpha \in C$:

- for each $p \in P$, the polynomial $\mathrm{init}(p)$ does not vanish at α,
- the polynomials $p(\alpha, x_n) \in \mathbf{K}[x_n]$, for all $p \in P$, are squarefree and coprime.

Note that this definition allows C to be a semi-algebraic set, see Theorem 3.

Cylindrical decomposition. By induction on n, we define the notion of a *cylindrical decomposition of* $\mathbf{K}^n$ together with that of the *tree associated with a cylindrical decomposition of* $\mathbf{K}^n$. For $n = 1$, a cylindrical decomposition of $\mathbf{K}$ is a finite collection of sets $\mathcal{D} = \{D_1, \ldots, D_{r+1}\}$, where either $r = 0$ and $D_1 = \mathbf{K}$, or $r > 0$ and there exists r non-constant coprime squarefree polynomials $p_1, \ldots, p_r$ of $\mathbf{k}[x_1]$ such that for $1 \le i \le r$ we have $D_i = \{x_1 \in \mathbf{K} \mid p_i(x_1) = 0\}$, and $D_{r+1} = \{x_1 \in \mathbf{K} \mid p_1(x_1) \cdots p_r(x_1) \neq 0\}$. Note that the D_i's, for all $1 \le i \le r+1$, form a partition of $\mathbf{K}$. The tree associated with $\mathcal{D}$ is a rooted tree whose nodes, other than the root, are $D_1, \ldots, D_r, D_{r+1}$ which all are leaves and children of the root. Now let $n > 1$, and let $\mathcal{D}' = \{D_1, \ldots, D_s\}$ be any cylindrical decomposition of $\mathbf{K}^{n-1}$. For each D_i, let r_i be a non-negative integer and let $\{p_{i,1}, \ldots, p_{i,r_i}\}$ be a set of polynomials which separates above D_i. If $r_i = 0$, set $D_{i,1} = D_i \times \mathbf{K}$. If $r_i > 0$, set

$$D_{i,j} = \{(\alpha, x_n) \in \mathbf{K}^n \mid \alpha \in D_i \text{ and } p_{i,j}(\alpha, x_n) = 0\},$$

for $1 \le j \le r_i$ and set $D_{i,r_i+1} = \left\{(\alpha, x_n) \in \mathbf{K}^n \mid \alpha \in D_i \text{ and } \left(\prod_{j=1}^{r_i} p_{i,j}(\alpha, x_n)\right) \neq 0\right\}$. The collection $\mathcal{D} = \{D_{i,j} \mid 1 \le i \le s, 1 \le j \le r_i + 1\}$ is called a *cylindrical decomposition* of $\mathbf{K}^n$. The sets $D_{i,j}$ are called the *cells* of $\mathcal{D}$. If T' is the tree associated with $\mathcal{D}'$ then the tree T associated with $\mathcal{D}$ is defined as follows. For each $1 \le i \le s$, the set D_i is a leaf in T' which has all $D_{i,j}$'s for children in T; thus the $D_{i,j}$'s are the leaves of T.

Note that each node N of T is either associated with no constraints, or associated with a polynomial constraint, which itself is either an equation or an inequation. Note also that, if the level of the polynomial defining the constraint at N is ℓ, then ℓ is the length of a path from N to the root. Moreover, the polynomial constraints along a path from the root to a leaf form a polynomial system called a *cylindrical system of* $\mathbf{k}[x_1, \ldots, x_n]$ *induced by* T. Let S be such a cylindrical system. We denote by $Z(S)$ the zero set of S. Therefore, each cell of $\mathcal{D}$ is the zero set of a cylindrical system induced by T.

Let Γ be a sub-tree of T such that the root of Γ is that of T. Then, we call Γ a *cylindrical tree of* $\mathbf{k}[x_1, \ldots, x_n]$ *induced by* T. This cylindrical tree Γ is said *partial* if it admits a non-leaf node N such that the zero set of the constraint of N is not equal to the union of the zero sets of the constraints of the children of N. If Γ is not partial, then it is called *complete*.

In the algorithms of Section 4, the cylindrical tree is an essential data structure. Section 3 discusses the main properties and operations on this data structure.

Let $F = \{f_1, \ldots, f_s\}$ be a finite set of polynomials of $\mathbf{k}[x_1 < \cdots < x_n]$. A cylindrical decomposition $\mathcal{D}$ of $\mathbf{K}^n$ is called *F-invariant* if for any given cell D of $\mathcal{D}$ and any given polynomial $f \in F$, either f vanishes at all points of D or f vanishes at no points of D.

Example 1. *Let $F := \{y^2 + x, y^2 + y\}$. An F-invariant cylindrical decomposition of $\mathbb{C}^2$ is illustrated by Figure 1.*

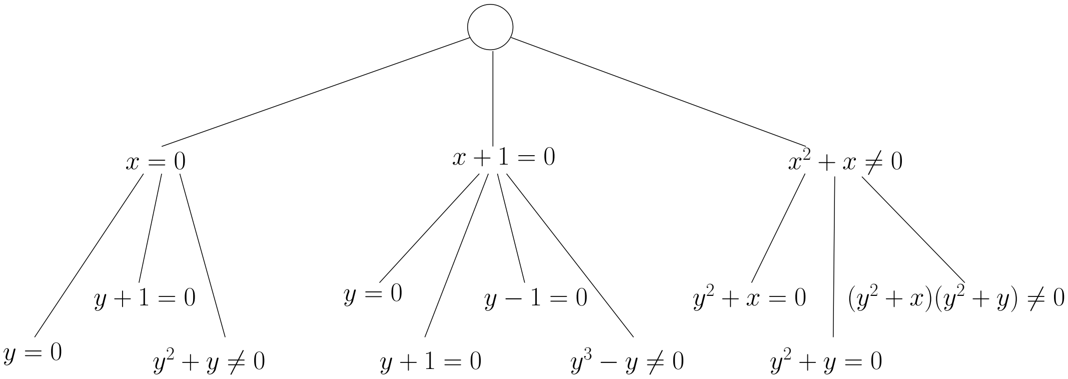

Figure 1: an $F := \{y^2 + x, y^2 + y\}$ invariant complex cylindrical tree

We observe that every cylindrical system induced by a cylindrical tree is a *simple system*, as defined by Wang in [37]. This notion was first introduced by Thomas in 1937 [36]. Simple systems have many nice properties. For example, if $[A, B]$ is a simple system, then the pair $[A, \prod_{p \in B} p]$ is a squarefree regular system, as defined by Wang in [37, 38].

Let Γ be a cylindrical system of $\mathbf{k}[\mathbf{x}]$ and let p be a polynomial of $\mathbf{k}[\mathbf{x}]$. We say that p is *invertible modulo* Γ if for any $\alpha \in Z(\Gamma)$, we have $p(\alpha) \neq 0$. We say that p is *zero modulo* Γ if for any $\alpha \in Z(\Gamma)$, we have $p(\alpha) = 0$. We say that p is *sign invariant above* Γ if p is either zero or invertible modulo Γ. Let q be another polynomial of $\mathbf{k}[\mathbf{x}]$. We say that $p = q$ modulo Γ if $Z(\Gamma) \cap Z(p) = Z(\Gamma) \cap Z(q)$.

Greatest common divisor (GCD). Let p and f be two level n polynomials in $\mathbf{k}[\mathbf{x}]$. Let Γ be a cylindrical system of $\mathbf{k}[x_1, \ldots, x_{n-1}]$. For any $u \in \mathbf{K}^{n-1}$ of $Z(\Gamma)$, assume at least one of $\mathrm{lc}(p, x_n)(u)$ and $\mathrm{lc}(f, x_n)(u)$ is not zero. A polynomial $g \in \mathbf{k}[\mathbf{x}]$ is called a *GCD of p and f modulo* Γ if for any $u \in \mathbf{K}^{n-1}$ of $Z(\Gamma)$,

- $g(u)$ is a GCD of $p(u)$ and $f(u)$ in $\mathbf{K}[x_n]$, and
- we have $\mathrm{lc}(g, x_n)(u) \neq 0$.

Let $d_p = \deg(p, x_n)$, $d_f = \deg(f, x_n)$. Recall that we assume $d_p, d_f \geq 1$. Let $\lambda = \min(d_p, d_f)$. Let Γ be a cylindrical system of $\mathbf{k}[x_1, \ldots, x_{n-1}]$. Let $S_0, \ldots, S_{\lambda-1}$ be the subresultant polynomials [31, 22] of p and f w.r.t. x_n. Let $s_i = \mathrm{coeff}(S_i, x_n^i)$ be the principle subresultant coefficient of S_i, for $0 \leq i \leq \lambda - 1$. If $d_p \geq d_f$, we define $S_\lambda = f$, $S_{\lambda+1} = p$, $s_\lambda = \mathrm{init}(f)$ and $s_{\lambda+1} = \mathrm{init}(p)$. If $d_p < d_f$, we define $S_\lambda = p$, $S_{\lambda+1} = f$, $s_\lambda = \mathrm{init}(p)$ and $s_{\lambda+1} = \mathrm{init}(f)$.

Theorem 1. *Let j be an integer, with $1 \leq j \leq \lambda + 1$, such that s_j is invertible modulo Γ and such that for any $0 \leq i < j$, we have $s_i = 0$ modulo Γ. Then S_j is a GCD of p and f modulo Γ.*

Proof. It can be easily proved by the specialization property of subresultant chains. In particular, it is a direct corollary of Theorem 5 in [13]. □

3 Data structure for cylindrical decomposition

In this section, we describe the data-structures that are used by the algorithms presented in this paper for computing cylindrical decompositions. To understand the motivation of our algorithm design, let us consider a simple example with $n = 2$ variables. Let a, b be two coprime squarefree non-constant univariate polynomials in $k[x_1]$. Observe that $L := k[x_1]/\langle a\, b\rangle$ is a direct product of fields. Let also c, d be two bivariate polynomials of $k[x_1, x_2]$, such that $\deg(c, x_2) > 0$, $\deg(d, x_2) > 0$, and $\mathrm{lc}(c, x_2) = \mathrm{lc}(d, x_2) = 1$ hold and such that c, d are coprime and squarefree univariate as polynomials of $L[x_2]$. Therefore the following four polynomial systems are simple systems

$$\left\{ \begin{array}{c} a(x_1)b(x_1) = 0 \\ c(x_1, x_2) = 0 \end{array} \right. , \left\{ \begin{array}{c} a(x_1)b(x_1) = 0 \\ d(x_1, x_2) = 0 \end{array} \right. , \left\{ \begin{array}{c} a(x_1)b(x_1) = 0 \\ c(x_1, x_2)d(x_1, x_2) \neq 0 \end{array} \right. , \left\{ \; a(x_1)b(x_1) \neq 0 \right.$$

that we denote respectively by S_1, S_2, S_3, S_4. It is easy to check that the zero sets $Z(S_1)$, $Z(S_2)$, $Z(S_3)$, $Z(S_4)$ are the cells of a cylindrical decomposition $\mathcal{D}$ of $\mathbf{K}^2$.

Let $f \in k[x_1]$ be another univariate polynomial. Assume that one has to *refine* $\mathcal{D}$ into a cylindrical decomposition of $\mathbf{K}^2$ which is required to be $\{f\}$-invariant. That is, one has to test whether f is invertible or zero modulo each of the systems S_1, S_2, S_3, S_4, and further decompose when appropriate. Assume that the polynomial a divides f whereas b, f are coprime. Assume also that the system S_1 is processed first in time. By computing $\gcd(f, ab)$, which yields a, one splits S_1 into the following two sub-systems that we denote by $S_{1,1}$ and $S_{1,2}$.

$$\left\{ \begin{array}{c} a(x_1) = 0 \\ c(x_1, x_2) = 0 \end{array} \right. , \text{ and } \left\{ \begin{array}{c} b(x_1) = 0 \\ c(x_1, x_2) = 0. \end{array} \right.$$

Assume that S_2 is processed next. By computing $\gcd(f, ab)$ (again) one splits S_2 into the following two sub-systems that we denote by $S_{2,1}$ and $S_{2,2}$.

$$\left\{ \begin{array}{c} a(x_1) = 0 \\ d(x_1, x_2) = 0 \end{array} \right. , \text{ and } \left\{ \begin{array}{c} b(x_1) = 0 \\ d(x_1, x_2) = 0. \end{array} \right.$$

Consequently, in the course of the creation of $S_{1,1}, S_{1,2}$, $S_{2,1}$ and $S_{2,2}$, the same polynomial GCD and the same field extensions (namely $k[x_1]/\langle a\rangle$ and $k[x_1]/\langle b\rangle$) were computed twice. This duplication of calculation and data is a common phenomenon and a performance bottleneck in most algorithms for decomposing polynomial systems.

Mathematically, each constructible set should not be represented more than once in a computer program. To implement this idea, all constructible sets manipulated during the execution of a given computer program should be seen as part of the same *universe*, say $\mathbf{K}^n$. Moreover, the subroutines of this program should have the same view on the universe, which is then a *shared data-structure*, such that whenever a subroutine modifies the universe all subroutines have immediate access to the modified universe. Satisfying these requirements is a well-known challenge in computer science, an instance of which is the question of *memory* *consistency* for shared-memory parallel computer architectures, such as multicores. With our above example, even if we do not intend to run computations concurrently, we are concerned with the practical efficiency and ease-of-use of the mechanisms that maintain *up-to-date* all views on the universe.

Recall that a cylindrical decomposition can be identified to a tree where each node is a constructible set of $\mathbf{K}^n$ given by either an equation constraint, or an inequation constraint, or no constraints at all. In this latter case, the corresponding constructible set is the whole space. All algorithms in Section 4 work on a given cylindrical decomposition $\mathcal{D}$ encoded by a tree T (as defined in Section 2). That is, the tree T is regarded as the universe.

We assume that there is a procedure for updating the tree T, which, given a "node-to-be-replaced" N and its "replacing nodes" $N_1, \ldots, N_e$, is called $\mathsf{split}(N; N_1, \ldots, N_e)$ and works as follows:

1. for $i = 1, \ldots, e$, for each child C of N deeply copy (thus creating new nodes) the sub-tree rooted at C and make that copy of C a child of N_i,
2. update the parent of N such that $N_1, \ldots, N_e$ are new children of the the parent of N,
3. remove the entire sub-tree rooted at N from the universe, including N.

We assume that all updates are performed sequentially (thus using mutual exclusion mechanism in case of concurrent execution of the algorithms of Section 4) such that no data-races can occur.

We also assume that each node N (whether it is a node in the present or has been removed from the universe) has a unique key, called $\mathsf{key}(N)$, and a data field, called $\mathsf{value}(N)$, storing various information including:

- a time stamp PAST or PRESENT,
- if PAST, the list of its replacing nodes (as specified with the split procedure) and the list of its children at the time it was replaced,
- if PRESENT, the list of its children and a pointer to the parent.

All nodes are stored in a *dictionary* H which can be accessed by all subroutines. Modifying the universe means updating H using the split procedure. Since all our algorithms stated in Section 4 are sequential, no synchronization issue has to be considered. The mechanism described above allows us to achieve our goals.

4 Constructing a cylindrical tree incrementally

In this section, we present an incremental algorithm for computing a cylindrical tree, as defined in Section 2. We start by commenting on the style of the pseudo-code. Secondly, we present the specifications of the algorithm and related subroutines. Thirdly, we state all the algorithms in pseudo-code style. Finally, proof sketches of the algorithms are provided at the end of this section.

Following the principles introduced in Section 3, our procedures operate on a "universe" (which is a cylindrical tree T) that they modify when needed. These modifications are of two types:

- splitting a node,
- attaching information to a node.

In addition to the attributes described in Section 3, a node has attributes corresponding to the results of operations like Squarefree, Gcd, Intersect. In other words, our procedures do not return values; instead they store their results in the nodes of the universe. This technique greatly simplifies pseudo-code.

Since attributes of nodes are intensively used in our pseudo-code, we use the standard "dot" notation of object oriented programming languages. In addition, since a node can have many attributes, we make the following convention. Suppose that a node V is split into two nodes V_1 and V_2. Some attributes are likely to have different values in V_1 (resp. V_2) and V. But most of them will often have the same values in both nodes. Therefore, after setting up the values of the attributes that differ, we simply write V_1.others := V.others to define the attributes of V_1 whose values are unchanged w.r.t. V.

Several procedures iterate through all the paths of the universe T. By path, we mean a path (in the sense of graph theory) from the root of T to a leaf of T. The current path is often denoted by Γ or C. Recall from Section 2 that a path in T corresponds to a simple system, say S. Computing modulo S may split S and thus modify the universe *automatically*, that is, in a transparent manner in the pseudo-code. However, splitting S also changes the current path. For clarity, we explicitly invoke a function called UpdatePath, which updates its first argument (namely the current path) from the universe.

In order to iterate through all the paths of the universe T, we use a function NextPathToDo. This command is a *generator* or an *iterator* in the sense of the theory of programming languages. That is, it views T as a stream of paths and returns the next path-to-be-visited, if any. Thanks to the fact that the universe is always up-to-date, the function NextPathToDo is able to return the next path-to-be-visited in the current state of the universe.

A frequently used operation on the universe and its paths is ExtractProjection, see for instance Algorithm 6. When applied to the universe T and an integer k (for $0 \leq k < n$, where n is the length of a path from the root of T to a leaf of T) ExtractProjection returns a "handle" on the universe "truncated" at level k, that is, the universe where all nodes of level higher than k are ignored (thus viewing the level k nodes as leaves). When applied to path, ExtractProjection has a similar output.

We often say that a function (see for instance Algorithm 5) returns a refined cylindrical decomposition. This is another way of saying that the universe is updated to a new state corresponding to a cylindrical decomposition refining (in the sense of a partition of a set refining another partition of the same set) the cylindrical decomposition of the previous state.

After these preliminary remarks on the pseudo-code, we present the specifications of the algorithm and related subroutines.

The top level algorithm for computing a cylindrical tree is described by Algorithm 4. It takes a set F of non-constant polynomials in $\mathbf{k}[x_1 < \cdots < x_n]$ as input and returns an F-invariant cylindrical decomposition of $\mathbf{K}^n$. This algorithm relies on a core operation, called Intersect, which computes a cylindrical decomposition in an incremental manner.

The Intersect operation is described by Algorithm 5. It takes a cylindrical tree T and a polynomial p of $\mathbf{k}[x_1 < \cdots < x_n]$ as input. It refines tree T such that p is sign invariant above each path of the refined tree T. This operation is achieved by refining each path of T with IntersectPath.

The IntersectPath operation is described by Algorithm 6. It takes a polynomial p, a cylindrical tree T and a path Γ of T in $\mathbf{k}[x_1 < \cdots < x_n]$ as input. It refines Γ and updates the tree T accordingly such that p is sign invariant above each path derived from Γ in the updated tree T. This operation finds the node N in Γ whose level is the same as that of p. Let Γ_N be the sub-path of Γ from N to the root of T. The IntersectPath operation then calls the routine IntersectMain so as to refine Γ_N into a tree T_N such that p becomes sign invariant w.r.t. T_N.

The routine IntersectMain is described by Algorithm 7. It takes a cylindrical tree T, a path Γ of T, and a polynomial of the same level as the leaves of T in $\mathbf{k}[x_1 < \cdots < x_n]$ as input. It refines Γ and updates the tree T accordingly such that p becomes sign invariant above each path derived from Γ in the updated tree.

The routine $\mathsf{IntersectMain}$ works in the following way. It first splits Γ such that above the projection C_{n-1} of each new branch C of Γ in $\mathbf{K}^{n-1}$, the number of distinct roots of p w.r.t. x_n is invariant. This is achieved by the operation $\mathsf{Squarefree}$, described by Algorithm 8. The squarefree part of p above a branch C is denoted by sp. If p has no roots or is identically zero above C_{n-1}, the sign of p above C is determined immediately. Otherwise, a case discussion is made according to the structure of the leaf node V of C. If V has no constraints associated to it, then V is simply split into two new nodes $sp = 0$ and $sp \neq 0$. Assume now that V has a constraint, which can be either of the form $f = 0$ or of the form $f \neq 0$, where f is a level n polynomial squarefree modulo C_{n-1}. This case is handled by computing the GCD g of sp and f modulo C_{n-1}. The node V then splits based on the GCD g and the co-factors of sp and f.

The GCD is computed by the operation Gcd, described by Algorithm 9 and 10. The co-factors are computed by Algorithm 11. The $\mathsf{Squarefree}$ and Gcd operations rely on the operation $\mathsf{MakeLeadingCoefficientInvertible}$, described by Algorithm 12. This latter operation takes as input a polynomial p of $\mathbf{k}[x_1 < \cdots < x_n]$, a cylindrical tree T of $\mathbf{k}[x_1 < \cdots < x_{n-1}]$ and a path Γ of T. Then, it refines Γ and updates T accordingly such that above each path C of T derived from Γ, the polynomial p is either zero or its leading coefficient is invertible.

All the algorithms also rely on the following three operations which perform manipulations and traversal of the tree data structure. For these three operations, only specifications are provided below while their algorithms are explained in Section 3.

Algorithm 1: $\mathsf{UpdatePath}(\Gamma, T)$

- Input: A cylindrical tree T. A path Γ in some past state of T.
- Output: A subtree ST in present state of T. ST is derived from Γ according to the historical data of T.

Algorithm 2: $\mathsf{ExtractProjection}(T, k)$

- Input: A cylindrical tree T of $\mathbf{k}[x_1 < \cdots < x_n]$. An integer k, $0 \leq k \leq n$.
- Output: A cylindrical tree T_k in $\mathbf{k}[x_1 < \cdots < x_k]$ such that T_k is the projection of T in $\mathbf{k}[x_1 < \cdots < x_k]$.

Algorithm 3: $\mathsf{NextPathToDo}_n(\mathrm{T})$

- Input: A cylindrical tree T in $\mathbf{k}[x_1 < \cdots < x_n]$.
- Output: For a fixed traversal order of a tree, return the first "ToDo" path Γ of T.

Theorem 2. *For a set of polynomials in $\mathbf{k}[x_1, \ldots, x_n]$, Algorithm 4 computes an F-invariant cylindrical decomposition of $\mathbf{K}^n$.*

Proof. Firstly, we prove the termination. The basic mutual calling graph of its subroutines are:

$$\mathsf{IntersectMain}_n \rightarrow \mathsf{Squarefree_n} \rightarrow \mathsf{IntersectMain}_{n-1} \rightarrow \cdots ,$$

Algorithm 4: CylindricalDecompose(F)

```
Input: $F$ is a set of non-constant polynomials in $\mathbf{k}[x_1 < \cdots < x_n]$.
Output: An $F$-invariant cylindrical decomposition of $\mathbf{K}^n$.
begin
    create a tree $T$ with only one vertex $V_0$: the root of $T$;
    for $i$ from 1 to $n$ do
        create a vertex $V_i$; $V_i.signs := \emptyset$; $V_i.formula :=$ "any $x_i$";
        $V_{i-1}.child := V_i$;
    for $p \in F$ do
        Intersect$_n(p, T)$;
    return $T$;
end
```

Algorithm 5: Intersect$_n(p, T)$

```
Input: A cylindrical tree $T$ of $\mathbf{k}[x_1 < \cdots < x_n]$. A non-constant polynomial $p$ of
       $\mathbf{k}[x_1 < \cdots < x_n]$.
Output: A refined cylindrical decomposition such that $p$ is sign invariant above
        each path of $T$.
while $\Gamma :=$ NextPathToDo$_n(T) \neq \emptyset$ do
    IntersectPath$_n(p, \Gamma, T)$;
```

Algorithm 6: IntersectPath$_n(p, \Gamma, T)$

```
Input: A cylindrical tree $T$ of $\mathbf{k}[x_1 < \cdots < x_n]$. A path $\Gamma$ of $T$. A polynomial $p$ of
       $\mathbf{k}[x_1 < \cdots < x_n]$.
Output: A refined cylindrical decomposition $T$ such that $p$ is sign invariant above
        each path derived from $\Gamma$.
begin
    if $p \in \mathbf{k}$ then
        return;
    else
        $k :=$ level($p$);
        if $k = n$ then
            IntersectMain$_n(p, \Gamma, T)$;
        else
            $T_k :=$ ExtractProjection($T, k$);$\Gamma_k :=$ ExtractProjection($\Gamma, k$);
            IntersectMain$_k(p, \Gamma_k, T_k)$;
            UpdatePath($\Gamma, T$);
            for each leaf $V$ of $\Gamma$ do
                Let $L_k$ be the ancestor of $V$ of level $k$; $V.signs[p] := L_k.signs[p]$ ;
end
```

Algorithm 7: $\mathsf{IntersectMain}_n(p, \Gamma, T)$

Input: A cylindrical tree T of $\mathbf{k}[x_1 < \cdots < x_n]$. A path Γ of T. A polynomial p of level n in $\mathbf{k}[x_1 < \cdots < x_n]$.

Output: A refined cylindrical decomposition T such that p is sign invariant above each path derived from Γ.

```
begin
    $T_{n-1} := \mathsf{ExtractProjection}(T, n-1)$; $\Gamma_{n-1} := \mathsf{ExtractProjection}(\Gamma, n-1)$;
    $\mathsf{Squarefree}_n(p, \Gamma_{n-1}, T_{n-1})$;
    $\mathsf{UpdatePath}(\Gamma, T)$;
    while $C := \mathsf{NextPathToDo}_n(\Gamma) \neq \emptyset$ do
        $V := C.leaf$; $C_{n-1} := \mathsf{ExtractProjection}(C, n-1)$;
        $sp := C_{n-1}.leaf.Squarefree[p]$;
        if $sp = 0$ then
            $V.signs[p] := 0$;
        else if $sp = 1$ then
            $V.signs[p] := 1$;
        else if $V.formula$ is "any $x_n$" then
            split $V$ into two new vertices $V_1$ and $V_2$;
            $V_1.formula := sp = 0$; $V_1.signs := V.signs$; $V_1.signs[p] := 0$;
            $V_2.formula := sp \neq 0$; $V_2.signs := V.signs$; $V_2.signs[p] := 1$;
            $V_1.others := V.others$; $V_2.others := V.others$;
            $C_{n-1}.leaf.children := V_1, V_2$;
        else
            // $V.formula$ is of the form $f = 0$ or $f \neq 0$
            $\mathsf{Gcd}_n(sp, f, C_{n-1}, T_{n-1})$;
            $\mathsf{UpdatePath}(C, T)$;
            for each leaf $V$ of $C$ do
                let $L$ be the parent of $V$;
                $cp, g, cf := \mathsf{CoFactor}(sp, L.Gcd[sp, f], f)$;
                if $V.formula$ is of the form $f = 0$ then
                    if $g = 1$ then
                        $V.signs[p] := 1$;
                    else if $cf = 1$ then
                        $V.signs[p] := 0$;
                    else
                        split $V$ into two new vertices $V_1$ and $V_2$;
                        $V_1.formula := g = 0$;$V_1.signs := V.signs$; $V_1.signs[p] := 0$;
                        $V_2.formula := cf = 0$;$V_2.signs := V.signs$; $V_2.signs[p] := 1$;
                        $V_1.others := V.others$; $V_2.others := V.others$;
                        $L.children := V_1, V_2$;
                else
                    if $cp = 1$ then
                        $V.signs[p] := 1$;
                    else
                        split $V$ into two new vertices $V_1$ and $V_2$;
                        $V_1.formula := cp = 0$; $V_1.signs := V.signs$; $V_1.signs[p] := 0$;
                        $V_2.formula := (f * cp) \neq 0$;
                        $V_2.signs := V.signs$; $V_2.signs[p] := 1$;
                        $V_1.others := V.others$; $V_2.others := V.others$;
                        $L.children := V_1, V_2$;
end
```

Algorithm 8: $\mathsf{Squarefree}_n(p, \Gamma, T)$

Input: A cylindrical tree T of $\mathbf{k}[x_1 < \cdots < x_{n-1}]$. A path Γ of T. A polynomial p of level n.

Output: A refined cylindrical tree T of $\mathbf{k}[x_1 < \cdots < x_{n-1}]$. Above each path C of T derived from Γ, there is a dictionary $C.leaf.Squarefree$. Let $p^* := C.leaf.Squarefree[p]$. We have:

- $p = p^*$ modulo C.
- If p^* is of level n, then both $\mathrm{init}(p^*)$ and $\mathrm{discrim}(p^*)$ are invertible modulo C.
- If p^* is of level less than n, then p^* is either 0 or 1.

```
begin
    if $n = 1$ then
        let $r$ be the root of $T$; $r.Squarefree[p] := \mathsf{SquarefreePart}(p)$;
        return
    $\mathsf{MakeLeadingCoefficientInvertible}_n(p, p, \Gamma, T)$;
    while $C := \mathsf{NextPathToDo}_{n-1}(\Gamma) \neq \emptyset$ do
        $f := C.leaf.InvertLc[p]$;
        if $\mathrm{level}(f) < n$ or $\deg(f, x_n) = 1$ then
            $C.leaf.Squarefree[p] := f$
        else
            $\mathsf{Gcd}_n(f, \mathrm{der}(f), C, T)$;
            for each leaf $L$ of $C$ do
                $g := L.Gcd[f, \mathrm{der}(f)]$;
                if $g = 1$ then
                    $L.Squarefree[p] := f$
                else
                    $L.Squarefree[p] := \mathrm{pquo}(f, g)$
end
```

Algorithm 9: $\mathsf{Gcd}_n(p, f, \Gamma, T)$

Input: A cylindrical tree T of $\mathbf{k}[x_1 < \cdots < x_{n-1}]$. A polynomial $p \in \mathbf{k}[x_1 < \cdots < x_n]$ of level n. A path Γ of T. A polynomial f of level n such that $\mathrm{init}(f)$ is invertible modulo Γ.

Output: A refined cylindrical tree T. Above each path C of T derived from Γ, there is a dictionary $C.leaf.Gcd$ such that $C.leaf.Gcd[p, f]$ is a GCD of p and f modulo C.

```
begin
    let $S$ be the subresultant chain of $p$ and $f$;
    if $\mathrm{mdeg}(p) \geq \mathrm{mdeg}(f)$ then
        $d := \mathrm{mdeg}(f)$
    else
        $d := \mathrm{mdeg}(p) + 1$
    return $\mathsf{Gcd}_n(p, f, S, d, 0, \Gamma, T)$;
end
```

Algorithm 10: $\mathsf{Gcd}_n(p, f, S, d, i, \Gamma, T)$

Input:

- A polynomial $p \in \mathbf{k}[x_1 < \cdots < x_n]$ of level n.
- A polynomial f of level n such that $\mathrm{lc}(f)$ is invertible modulo Γ.
- The subresultant chain S of p and f w.r.t. x_n.
- A non-negative integer d (as defined in the pseudo-code of Algorithm 9) and such that the principle subresultant coefficient s_d is invertible modulo Γ.
- A non-negative integer i such that $0 \leq i \leq d$ and the principle subresultant coefficient s_j is zero modulo Γ, for all $0 \leq j < i$.
- A path Γ of T.
- A cylindrical tree T of $\mathbf{k}[x_1 < \cdots < x_{n-1}]$.

Output: A refined cylindrical tree T. Above each path C of T derived from Γ, there is a dictionary $C.leaf.Gcd$ such that $C.leaf.Gcd[p, f]$ is a GCD of p and f modulo C.

```
begin
    if $i = d$ then
        $\Gamma.leaf.Gcd[p, f] := S_i$;
        return;
    $\mathsf{IntersectPath}_{n-1}(s_i, \Gamma, T)$;
    while $C := \mathsf{NextPathToDo}_{n-1}(\Gamma) \neq \emptyset$ do
        if $C.leaf.signs[s_i] = 1$ then
            if $i = 0$ then
                $C.leaf.Gcd[p, f] := 1$
            else
                $C.leaf.Gcd[p, f] := S_i$
        else
            $\mathsf{Gcd}_n(p, f, S, d, i + 1, C, T)$
end
```

Algorithm 11: CoFactor(p, g, f)

Input: Two polynomials p and f of level n in $\mathbf{k}[x_1 < \cdots < x_n]$. A polynomial g which is either 1 or of level n in $\mathbf{k}[x_1 < \cdots < x_n]$.

Output: As described by the algorithm.

```
begin
    if $g = 1$ then
        $cp := p$; $gg := 1$; $cf := f$;
    else if $\mathrm{mdeg}(g) = \mathrm{mdeg}(f)$ then
        $gg := f$;
        if $\mathrm{mdeg}(g) = \mathrm{mdeg}(p)$ then
            $cf := 1$; $cp := 1$;
        else
            $cf := 1$; $cp := \mathrm{pquo}(p, gg)$
    else if $\mathrm{mdeg}(g) = \mathrm{mdeg}(p)$ then
        $gg := p$; $cf := \mathrm{pquo}(f, gg)$; $cp := 1$;
    else
        $cp := \mathrm{pquo}(p, g)$; $cf := \mathrm{pquo}(f, g)$; $gg := g$;
    return $cp, gg, cf$;
end
```

Algorithm 12: MakeLeadingCoefficientInvertible$_n(p, \bar{p}, \Gamma, T)$

Input: A polynomial p of $\mathbf{k}[x_1 < \cdots < x_n]$. A polynomial $\bar{p}$ of $\mathbf{k}[x_1 < \cdots < x_n]$ such that $p = \bar{p}$ modulo Γ. A cylindrical tree T of $\mathbf{k}[x_1 < \cdots < x_{n-1}]$. A path Γ of T.

Output: A refined cylindrical tree T of $\mathbf{k}[x_1 < \cdots < x_{n-1}]$. Above each path C of T derived from Γ, there is a dictionary $C.leaf.InvertLc$. Let p^* be the polynomial $C.leaf.InvertLc[p]$. Then, we have:

- $p = p^*$ modulo C.
- If p^* is of level n, then $\mathrm{init}(p^*)$ is invertible modulo the path C.
- If p^* is of level less than n, then p^* is either 0 or 1.

```
begin
    IntersectPath$_{n-1}(\mathrm{lc}(\bar{p}, x_n), \Gamma, T)$;
    while $C :=$ NextPathToDo$_{n-1}(\Gamma) \neq \emptyset$ do
        if $C.leaf.signs[\mathrm{lc}(\bar{p}, x_n)] = 1$ then
            if $\mathrm{level}(\bar{p}) < n$ then
                $C.leaf.InvertLc[p] := 1$
            else
                $C.leaf.InvertLc[p] := \bar{p}$
        else
            if $\mathrm{level}(\bar{p}) < n$ then
                $C.leaf.InvertLc[p] := 0$
            else
                MakeLeadingCoefficientInvertible$_n(p, \mathrm{tail}(\bar{p}), C, T)$
end
```

and

$$\mathsf{IntersectMain}_n \to \mathsf{Gcd_n} \to \mathsf{IntersectMain}_{n-1} \to \cdots$$

So the termination is easily proved by induction. The correctness follows from the specification of its subroutines and Theorem 1. □

Example 2. *In this example, we illustrate the operation* IntersectPath*. Let $F := \{y^2 + x, y^2 + y\}$. The incremental algorithm first computes an $y^2 + x$ sign invariant complex cylindrical tree, which is described by the following tree T.*

$$T := \begin{cases} x = 0 & \begin{cases} y = 0 & : \quad y^2 + x = 0 \\ y \neq 0 & : \quad y^2 + x \neq 0 \end{cases} \\ & \\ x \neq 0 & \begin{cases} y^2 + x = 0 & : \quad y^2 + x = 0 \\ y^2 + x \neq 0 & : \quad y^2 + x \neq 0 \end{cases} \end{cases}$$

Let Γ be the path $\{x = 0, y \neq 0\}$ of T. Calling $\mathsf{IntersectPath}(y^2 + y, \Gamma, T)$ *will update T into the following tree.*

$$\begin{cases} x = 0 & \begin{cases} y = 0 & : \quad y^2 + x = 0 \\ y = -1 & : \quad y^2 + x \neq 0 \wedge y^2 + y = 0 \\ \text{otherwise} & : \quad y^2 + x \neq 0 \wedge y^2 + y \neq 0 \end{cases} \\ & \\ x \neq 0 & \begin{cases} y^2 + x = 0 & : \quad y^2 + x = 0 \\ y^2 + x \neq 0 & : \quad y^2 + x \neq 0 \end{cases} \end{cases}$$

5 Building a CAD tree from a complex cylindrical tree

In this section, we review briefly how to compute a CAD of $\mathbb{R}^n$ from a cylindrical decomposition of $\mathbb{C}^n$. The reader may refer to [14] for more details. Recall that $n \geq 1$ holds. We denote by π_{n-1} the standard projection from $\mathbb{R}^n$ to $\mathbb{R}^{n-1}$ that maps $(x_1, \ldots, x_{n-1}, x_n)$ onto $(x_1, \ldots, x_{n-1})$.

Stack over a connected semi-algebraic set. Let S be a connected semi-algebraic subset of $\mathbb{R}^{n-1}$. The *cylinder* over S in $\mathbb{R}^n$ is defined as $Z_{\mathbb{R}}(S) := S \times \mathbb{R}$. Let $\theta_1 < \cdots < \theta_s$ be continuous semi-algebraic functions defined on S. The intersection of the graph of θ_i with $Z_{\mathbb{R}}(S)$ is called the *θ_i-section* of $Z_{\mathbb{R}}(S)$. The set of points between two consecutive sections of $Z_{\mathbb{R}}(S)$ is a connected semi-algebraic subset of $\mathbb{R}^n$, called a *sector* of $Z_{\mathbb{R}}(S)$. All the sections and sectors of $Z_{\mathbb{R}}(S)$ form a disjoint decomposition of $Z_{\mathbb{R}}(S)$, called a *stack* over S.

Cylindrical algebraic decomposition. A finite partition $\mathcal{D}$ of $\mathbb{R}^n$ is called a *cylindrical algebraic decomposition* (CAD) of $\mathbb{R}^n$ if one of the following properties holds.

- Either $n = 1$ and $\mathcal{D}$ is a stack over $\mathbb{R}^0$.
- Or the set of $\{\pi_{n-1}(D) \mid D \in \mathcal{D}\}$ is a CAD of $\mathbb{R}^{n-1}$ and each $D \in \mathcal{D}$ is a section or sector of the stack over $\pi_{n-1}(D)$.

When this holds, the elements of $\mathcal{D}$ are called *cells*.

Sign invariance and delineability. Let p be a polynomial of $\mathbb{R}[x_1, \ldots, x_n]$, and let S be a subset of $\mathbb{R}^n$. The polynomial p is called *sign invariant* on S if the sign of $p(\alpha)$ does not change when α ranges over S. Let $F \subset \mathbb{R}[x_1, \ldots, x_n]$ be a finite polynomial set. We say S is F-invariant if each $p \in F$ is invariant on S. A cylindrical algebraic decomposition $\mathcal{D}$ is

F-invariant if F is invariant on each cell $D \in \mathcal{D}$. Let p be a polynomial of $\mathbb{R}[x_1, \ldots, x_n]$, and let S be a connected semi-algebraic set of $\mathbb{R}^{n-1}$. We say that p is *delineable* on S if the real zeros of p define continuous semi-algebraic functions $\theta_1, \ldots, \theta_s$ such that, for all $\alpha \in S$ we have $\theta_1(\alpha) < \cdots < \theta_s(\alpha)$. In other words, p is delineable on S if its real zeros naturally determine a stack over S. We recall the following Theorem introduced in [14].

Theorem 3. *Let $P = \{p_1, \ldots, p_r\}$ be a finite set of polynomials in $\mathbb{R}[x_1 < \cdots < x_n]$ of level n. Let S be a connected semi-algebraic subset of $\mathbb{R}^{n-1}$. If P* separates *above S, then each p_i is delineable on S. Moreover, the product of the $p_1, \ldots, p_r$ is also delineable on S.*

Let F be a finite set of polynomials in $\mathbb{Q}[x_1 < \cdots < x_n]$. Let CT be an F-invariant complete cylindrical tree of $\mathbb{C}^n$. Applying Theorem 3 to polynomials in CT, we can derive an F-invariant cylindrical algebraic decomposition of $\mathbb{R}^n$ by induction on n. A procedure MakeSemiAlgebraic, was introduced in [14] to derive a CAD from a CT via real root isolation of zero-dimensional regular chains.

Example 3. *Let $F := \{y^2 + x\}$. An F-invariant cylindrical algebraic decomposition is described by the following tree.*

$$T := \begin{cases} x < 0 & \begin{cases} y < -\sqrt{|x|} & : \quad y^2 + x > 0 \\ y = -\sqrt{|x|} & : \quad y^2 + x = 0 \\ y > -\sqrt{|x|} \wedge y < \sqrt{|x|} & : \quad y^2 + x < 0 \\ y = \sqrt{|x|} & : \quad y^2 + x = 0 \\ y > \sqrt{|x|} & : \quad y^2 + x > 0 \end{cases} \\ & \\ x = 0 & \begin{cases} y < 0 & : \quad y^2 + x > 0 \\ y = 0 & : \quad y^2 + x = 0 \\ y > 0 & : \quad y^2 + x > 0 \end{cases} \\ & \\ x > 0 & \text{for any } y \ : \ y^2 + x > 0 \end{cases}$$

6 Making use of equational constraints and other optimizations

In this section, we discuss several possible optimizations to algorithms presented in Section 4.

Firstly, we discuss how to compute a CAD dedicated to a semi-algebraic system, which provides a systematic solution for making use of equational constraints when computing CADs. The motivation for making use of equational constraints comes from quantifier elimination. Let

$$PF := (Q_{k+1}x_{k+1} \cdots Q_n x_n) FF(x_1, \ldots, x_n),$$

be a prenex formula, where FF is a DNF formula. To perform QE by CAD, the first computation step is to collect all the polynomials appearing in FF as a polynomial set F and compute an F-invariant CAD of $\mathbb{R}^n$. This process of computing an F-invariant CAD exhausts all possible sign combinations of F, including those which do not appear in FF, and thus often computes much more than needed for solving the input QE problem. Different techniques in the literature have been proposed for taking advantage of the structure of the input problem. These methods include partial CAD [17] for lazy lifting, simplified projection operator for handling pure strict inequalities [27, 33], smaller projection sets for making use of equational constraints [16, 29, 7, 30].

To make the discussion clear, we first quote a paragraph of [7]. “The idea is as follows: if an input formula includes the constraint $f = 0$, then decompose $\mathbb{R}^r$ into regions in which f has invariant sign, and then refine the decomposition so that the other polynomials have invariant sign in those cells in which $f = 0$. The signs of the other polynomials in cells in which $f \neq 0$ are, after all, irrelevant. Additionally, the method of equational constraints seeks to deduce and use constraints that are not explicit in the input formula, but rather arise as consequences of two or more explicit constraints (e.g. if $f = 0$ and $g = 0$ are explicit constraints, then $\mathrm{res}(f, g) = 0$ is also a constraint.)”

This idea, of course, is attractive. Much progress on it has also been made. However, the reason why it is a generally hard problem for CAD is that the framework of PCAD does not have much flexibility to allow propagation of equational constraints. In the world of PCAD, one always tries to obtain a generic projection operator and then applies the same projection operator recursively. To obtain a generic projection operator for handling equational constraints is hard because many problems inherently require different projection operators during projection. Therefore case discussion is important.

In fact, case discussion is very common in algorithms for computing triangular decompositions. For such algorithms, equational constraints are natural input of these algorithms. The two keys ideas “splitting only above $f = 0$” and “if $f = 0$ and $g = 0$ are explicit constraints, then $\mathrm{res}(f, g) = 0$ is also a constraint” have already been systematically taken care of in the Intersect operation of the authors’ paper for computing triangular decompositions [12].

Next we explain how to modify algorithms presented in Section 4 to automatically implement these ideas.

Suppose now that the input of Algorithm CylindricalDecompose is a system of equations or inequations, this algorithm will then compute a partial cylindrical tree such that its zero set is exactly the zero set of input system. This can be simply achieved by passing an equation or inequation to the function Intersect. W.l.o.g., let us assume that an equation $p = 0$ is passed as an argument of Intersect. Then for this function and all its called subroutines, we will cut the computation branches above which p is known to be nonzero and never proceed with computation branches above which p cannot be zero. For example, we will not create a new vertex at step $15, 32, 42$ in Algorithm IntersectMain. We will delete the vertex V at step 11, 26, 37 since p is nonzero on V.

The first important optimization in IntersectMain which can be implemented is to avoid Squarefree computation at step 3 if $\Gamma.leaf$ is an equational constraint. This idea is quite close to “splitting only above $f = 0$”. Another important optimization can be done at step 19 of IntersectMain. Assume that $V.formula$ is an equational constraint $f = 0$, then when Gcd is called, in step 5 of Algorithm 10, we can do as follows. If $i = 0$, then s_i is the resultant of p and f. Thus we should pass $s_i = 0$ to the IntersectPath operation in order to avoid useless computations on the branch $s_i \neq 0$. This addresses the idea “if $f = 0$ and $g = 0$ are explicit constraints, then $\mathrm{res}(f, g) = 0$ is also a constraint.” Moreover, these optimizations are systematically performed during the whole computation.

Next we briefly mention several other important optimizations. Let V be a leaf of a path Γ of a cylindrical tree. Assume that $V.formula$ is of the form $f \neq 0$ or of the form $f = 0$. We can safely replace f by its primitive part since $\mathrm{lc}(f)$ is invertible modulo Γ_{n-1}. Replacing f by its irreducible factors over $\mathbb{Q}$ is often a more efficient choice. Last but not least, recall that a path Γ in the cylindrical tree is a simple system. Writing Γ as two parts

$\Gamma := [T, H]$, where T is a set of equations and H is a set of inequations. We know that T is a regular chain and Γ is a squarefree regular system. Thus the Zariski closure of Γ is the variety of the saturated ideal of T. We can call the pseudo division operation $\mathrm{prem}(p, T)$ or $\mathrm{prem}(f, T)$ to test whether p or f is zero modulo Γ. And sometimes replacing p by $\mathrm{prem}(p, T)$ and f by $\mathrm{prem}(f, T)$ also ease the computations.

Example 4. *Let $F := \{y^2 + x = 0, y^2 + y = 0\}$ be a system of equations. Taking F as input, Algorithm* CylindricalDecompose *generates the following partial cylindrical tree T of $\mathbb{C}^2$ such that the zero set of F is exactly the union of the zero sets of the paths in T.*

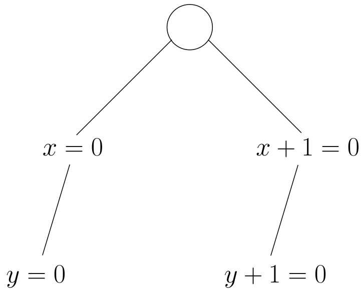

Figure 2: A partial cylindrical tree T adapted to F

7 Benchmark

In this section, we report on the experimental results of a preliminary implementation in the `RegularChains` library of Maple of the algorithms of Sections 4 and 5.

The examples in Table 1 and Table 2 are from papers on polynomial system solving, such as [11, 3] and the references therein. All the tests were launched on a machine with Intel Core 2 Quad CPU (2.40GHz) and 8.0Gb total memory. The time-out is set as 1 hour. In the tables, the symbol $> 1h$ means time-out.

The Maple functions are launched in Maple 15 with the latest `RegularChains` library. The memory usage is limited to 60% of total memory. The software Qepcad is launched with the option $+N500000000 + L200000$, where the first option specifies the memory to be pre-allocated (about 23% of total memory for our machine) and the second option specifies the number of prime numbers to be used.

In Table 1, we report on timings for computing cylindrical decomposition of the complex space with different algorithms and options. Each input system is a set of polynomials. The notation tcd-rec denotes an implementation of the original recursive algorithm in [14], while the notation tcd-inc denotes the incremental algorithm presented in Section 4. Both tcd-rec and tcd-inc take a set of polynomials as input. The notation tcd-eqs refers to an optimized version of tcd-inc which makes use of equational constraints, as explained in Section 6. With the implementation tcd-eqs, every input polynomial set is regarded as a set of equations (equating each input polynomial to zero). As we can see in Table 1, the incremental algorithm presented in this paper is much more efficient than the original recursive algorithm. The timings of tcd-eqs show that the optimizations presented in Section 6 for making use of equational constraints are very effective.

Table 1: Timings for computing cylindrical decomposition of the complex space

System	tcd-rec	tcd-inc	tcd-eqs	System	tcd-rec	tcd-inc	tcd-eqs
AlkashiSinus	3373.966	14.568	4.168	MontesS10	$> 1h$	$> 1h$	2.952
Alonso	9.636	1.404	0.700	MontesS12	$> 1h$	$> 1h$	7.528
Arnborg-Lazard-rev	2759.940	2419.543	16.233	MontesS15	$> 1h$	$> 1h$	77.048
Barry	39.346	1.808	0.556	MontesS16	$> 1h$	$> 1h$	8.228
blood-coagulation-2	235.310	9.472	0.808	MontesS4	556.390	102.122	0.488
Bronstein-Wang	255.427	35.990	1.120	MontesS5	1449.810	119.059	1.004
cdc2-cyclin	$> 1h$	68.920	65.976	MontesS7	$> 1h$	$> 1h$	1.060
circles	276.389	2.280	0.520	MontesS9	269.636	4.212	0.980
genLinSyst-3-2	916.245	19.537	1.384	nql-5-4	$> 1h$	1.056	0.528
genLinSyst-3-3	$> 1h$	160.406	12.408	r-5	68.364	3.232	0.876
Gerdt	$> 1h$	$> 1h$	1.188	r-6	1456.883	46.458	1.200
GonzalezGonzalez	141.072	53.451	0.732	Raksanyi	1471.351	118.227	1.000
hereman-2	$> 1h$	40.042	0.908	Rose	$> 1h$	51.855	1.072
lhlp5	31.069	3.984	0.648	Wang93	$> 1h$	$> 1h$	18.877
Maclane	$> 1h$	$> 1h$	6.420	YangBaxterRosso	54.895	1.560	0.844

Table 2: Timings for computing CAD

System	qepcad	qepcad-eqs	mathematica-eqs	tcad	tcad-eqs
Alonso	7.516	5.284	0.74	61.591	5.776
Arnborg-Lazard-rev	$> 1h$	$> 1h$	0.952	$> 1h$	17.325
Barry	Fail	216.425	0.032	8.580	1.004
blood-coagulation-2	$> 1h$	$> 1h$	$> 1h$	985.709	7.260
Bronstein-Wang	$> 1h$	$> 1h$	26.726	333.892	2.564
cdc2-cyclin	$> 1h$	$> 1h$	0.208	574.127	503.863
circles	21.633	5.996	41.211	$> 1h$	40.902
GonzalezGonzalez	10.528	10.412	0.012	214.213	1.136
lhlp2	960.756	5.076	0.016	3.124	0.952
lhlp5	10.300	10.068	0.016	35.338	1.084
MontesS4	$> 1h$	$> 1h$	0.004	2682.391	0.888
MontesS5	Fail	Fail	$> 1h$	$> 1h$	9.400
nql-5-4	93.073	5.420	1303.07	113.675	1.004
r-5	$> 1h$	1802.676	0.016	1282.928	1.208
r-6	$> 1h$	$> 1h$	0.024	$> 1h$	1.500
Rose	Fail	$> 1h$	$> 1h$	606.361	3.136
AlkashiSinus	$> 1h$	$> 1h$	2.232	$> 1h$	58.775
genLinSyst-3-2	Fail	Fail	217.062	3013.764	6.588
MontesS10	$> 1h$	$> 1h$	$> 1h$	$> 1h$	22.797
MontesS12	$> 1h$	$> 1h$	$> 1h$	$> 1h$	330.996
MontesS15	$> 1h$	$> 1h$	0.004	$> 1h$	395.964
MontesS7	$> 1h$	$> 1h$	245.807	$> 1h$	2.452
MontesS9	Fail	Fail	$> 1h$	110.902	4.944
Wang93	Fail	Fail	$> 1h$	$> 1h$	152.673

In Table 2, we report on timings for computing CAD with three different computer algebra packages: QEPCAD, the CylindricalDecomposition command of Mathematica and the algorithm presented in Section 4. Each system is a set of polynomials. Two categories of experimentation are conducted. The first category is concerned with the timings for computing a full CAD of a set of polynomials. For Mathematica, we cannot find any options of CylindricalDecomposition for computing a full CAD of a set of polynomials. Therefore for this category, only the timings of QEPCAD and TCAD are reported. The second category is concerned with the timings for computing a CAD of a variety. For this category, the timings for QEPCAD, Mathematica and TCAD are all reported.

The notation qepcad denotes computations that QEPCAD performs by (1) treating each input system as a set of non-strict inequalities and, (2) treating all variables as free variables and, (3) executing with the "full-cad" option. The notation tcad corresponds to computations that TCAD performs by (1) treating each input system as a set of non-strict inequalities and, (2) computing a sign invariant full CAD of polynomials in the input system and, (3) selecting the cells which satisfy those non-strict inequalities. In this way, both qepcad and TCAD compute a full CAD of a set of polynomials.

The notation qepcad-eqs denotes the computations that QEPCAD performs by (1) treating each input system as a set of equations and, (2) treating all variables as free variables and, (3) executing with the default option. The notation mathematica-eqs represents computations where the CylindricalDecomposition command of Mathematica treats each input system as a set of equations. The notation tcad-eqs corresponds to computations where TCAD treats each input system as a set of equations.

From Table 2, we make the following observations. When full CADs are computed, within one hour time limit, QEPCAD only succeeds on 6 out of 24 examples while TCAD succeeds on 14 out of 24 examples. When CADs of varieties are computed, for all the 10 out of 24 examples that QEPCAD can solve within one hour time limit, both Mathematica and TCAD succeed with usually less time. For the rest 14 examples, TCAD solves all of them while Mathematica only succeeds on 7 of them.

8 Conclusion

In this paper, we present an incremental algorithm for computing CADs. A key part of the algorithm is an Intersect operation for refining a given complex cylindrical tree. If this operation is supplied with an equational constraint, it only computes a partial cylindrical tree, which provides an automatic solution for propagating equational constraints. We have implemented our algorithm in MAPLE. The experimentation shows that the new algorithm is much more efficient than our previous recursive algorithm. We also compared our implementation with the software packages QEPCAD and Mathematica. For many examples, our implementation outperforms the other two. This incremental algorithm can support quantifier elimination. We will present this work in a future paper.

Acknowledgments

The authors would like to thank the readers who helped improve the earlier versions of this paper. This research was supported by Academic Development Fund ADF-Major-27145 of The University of Western Ontario.

References

[1] D. S. Arnon, G. E. Collins, and S. McCallum. Cylindrical algebraic decomposition II: an adjacency algorithm for the plane. *SIAM J. Comput.*, 13(4):878–889, 1984.

[2] T. Bächler, V. Gerdt, M. Lange-Hegermann, and D. Robertz. Thomas decomposition of algebraic and differential systems. In *Proc. of CASC'10*, pages 31–54, 2010.

[3] F. Boulier, C. Chen, F. Lemaire, and M. Moreno Maza. Real root isolation of regular chains. In *Proc. of ASCM'09*, pages 15–29, 2009.

[4] C. W. Brown. Improved projection for cylindrical algebraic decomposition. *J. Symb. Comput.*, 32(5):447–465, 2001.

[5] C. W. Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *SIGSAM Bull.*, 37(4):97–108, 2003.

[6] C. W. Brown and J. H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proc. ISSAC'07*, pages 54–60.

[7] C. W. Brown and S. McCallum. On using bi-equational constraints in CAD construction. In *ISSAC'05*, pages 76–83, 2005.

[8] B. Buchberger and H. Hong. Speeding-up quantifier elimination by Gröbner bases. Technical Report 91-06, RISC (Research Institute for Symbolic Computation), Johannes Kepler University, Linz, Austria, February 1991.

[9] B. Caviness and J. Johnson, editors. *Quantifier Elimination and Cylindical Algebraic Decomposition, Texts and Mongraphs in Symbolic Computation.* Springer, 1998.

[10] C. Chen. *Solving Polynomial Systems via Triangular Decomposition.* PhD thesis, University of Western Ontario, 2011.

[11] C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. Comprehensive triangular decomposition. In *Proc. of CASC'07*, volume 4770 of *Lecture Notes in Computer Science*, pages 73–101. Springer Verlag, 2007.

[12] C. Chen and M. Moreno Maza. Algorithms for computing triangular decompositions of polynomial systems. In *Proc. of ISSAC'11*, pages 83–90, 2011.

[13] C. Chen and M. Moreno Maza. Algorithms for computing triangular decomposition of polynomial systems. *Journal of Symbolic Computation*, 47(6):610 – 642, 2012.

[14] C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *ISSAC'09*, pages 95–102, 2009.

[15] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. *Springer Lecture Notes in Computer Science*, 33:515–532, 1975.

[16] G. E. Collins. Quantifier elimination by cylindrical algebraic decomposition - twenty years of progress. In B. Caviness and J. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 8–23. Springer, 1998.

[17] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 12(3):299–328, 1991.

[18] G. E. Collins, J. R. Johnson, and W. Krandick. Interval arithmetic in cylindrical algebraic decomposition. *J. Symb. Comput.*, 34(2):145–157, 2002.

[19] X. Dahan, M. Moreno Maza, É. Schost, W. Wu, and Y. Xie. Lifting techniques for triangular decompositions. In *ISSAC'05*, pages 108–115. ACM Press, 2005.

[20] A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. In *Proc. ISSAC '04*, pages 111–118. ACM, 2004.

[21] A. Dolzmann and T. Sturm. Redlog computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31:2–9, 1996.

[22] L. Ducos. Optimizations of the subresultant algorithm. *Journal of Pure and Applied*

Algebra, 145:149–163, 2000.

[23] H. Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In *ISSAC '90*, pages 261–264. ACM, 1990.

[24] H. Hong *et al.* QEPCAD B, `www.usna.edu/Users/cs/qepcad/`.

[25] H. Iwane, H. Yanami, H. Anai, and K. Yokoyama. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In *Proc. of SNC'2009*, pages 55–64, 2009.

[26] S. McCallum. An improved projection operation for cylindrical algebraic decomposition of 3-dimensional space. *J. Symb. Comput.*, 5(1-2):141–161, 1988.

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

[28] S. McCallum. An improved projection operator for cylindrical algebraic decomposition. In B. Caviness and J. Johnson, editors, *Quantifier Elimination and Cylindical Algebraic Decomposition, Texts and Mongraphs in Symbolic Computation.* Springer, 1998.

[29] S. McCallum. On propagation of equational constraints in CAD-based quantifier elimination. In *Proc. of ISSAC '01*, pages 223–231, 2001.

[30] S. McCallum and C.W. Brown. On delineability of varieties in CAD-based quantifier elimination with two equational constraints. In *Proc. of ISSAC '09*, pages 71–78, 2009.

[31] B. Mishra. *Algorithmic Algebra.* Springer-Verlag, New York, 1993.

[32] M. Moreno Maza. On triangular decompositions of algebraic varieties. Technical Report TR 4/99, NAG Ltd, Oxford, UK, 1999. http://www.csd.uwo.ca/∼moreno.

[33] A. Strzeboński. Solving systems of strict polynomial inequalities. *J. Symb. Comput.*, 29(3):471–480, 2000.

[34] A. Strzeboński. Cylindrical algebraic decomposition using validated numerics. *Journal of Symbolic Computation*, 41(9):1021 – 1038, 2006.

[35] A. Strzeboński. Computation with Semialgebraic Sets Represented by Cylindrical Algebraic Formulas. In *Proc. of ISSAC'2010*, pages 61–68, 2010.

[36] J. M. Thomas. *Differential System.* American Mathematical Society, New York, 1937.

[37] D. M. Wang. Decomposing polynomial systems into simple systems. *J. Symb. Comp.*, 25(3):295–314, 1998.

[38] D. M. Wang. Computing triangular systems and regular systems. *J. Sym. Comp.*, 30(2):221–236, 2000.

[39] D.J. Wilson, R.J. Bradford, and J.H. Davenport. Speeding up cylindrical algebraic decomposition by Gröbner bases. In *AISC/MKM/Calculemus*, pages 280–294, 2012.