Cylindrical Algebraic Decompositions With Monotone Cells

Hollie Baker

2024-08-19

Contents

Chapter 1

Introduction

Algebraic geometry is a fundamental and active area of research in mathematics and, loosely speaking, is concerned with studying sets which can be defined by polynomials. A fundamental object in algebraic geometry is the algebraic variety, a set defined by the simultaneous solutions to a system of polynomial equations. Sometimes, a wider class of sets, semialgebraic sets, which are defined by a Boolean combination of polynomial equations and inequalities, are studied. Complex algebraic geometry involves working over an algebraically closed field, where the fundamental theorem of algebra holds, while in real algebraic geometry we are restricted to the real numbers. The focus of this research is on algorithms for computing with semialgebraic sets defined over the real numbers. Computational algebraic geometry spans both mathematics and computer science and involves developing algorithms for working with the fundamental objects of algebraic geometry, e.g., algebraic varieties, polynomial ideals and semialgebraic sets. Some fundamental results in this area were presented at the International Symposium on Symbolic and Algebraic Manipulation which was held in Marseille in 1979 (now part of ISSAC). Buchberger (1979) presented his criterion and an algorithm for computing a Groebner basis for a polynomial ideal defined over a real closed field, which is still one of the most efficient methods for finding the complex solution of a systems of multivariate polynomial equations. Lazard (1979) also presented an algorithm, based on Macalay's multivariate resultants, for solving systems of homogeneous polynomial equations over any field (algebraically closed or otherwise), which has complexity singly exponential in the number of unknowns. Arnon (1979) described an extension to Collins' algorithm for computing cylindrical decompositions of real space, which allows one to determine the dimension of a semialgebraic set and deduce some of its topological properties. Since then, there have been many advances in these areas, both in extending the applications of these algorithms and making them more efficient.

A fundamental result in real algebraic geometry is Tarski's Theorem (Van den

Dries et al., 1988), which states that semialgebraic sets are closed under projection. This result is not true for algebraic sets, neither real nor complex. Indeed, consider the projection of the solutions of the polynomial $xy = 1$ onto the x-axis. We get $\mathbb{R} \setminus \{0\}$, which is not an algebraic set, but could be represented if we are allowed to use inequalities. Thus, we will be working over the category of semialgebraic sets: sets of points defined by Boolean combinations of polynomial equations and inequalities. Alfred Tarski first alluded, without proof, to his famous theorem in the abstract of Sierpiński and Tarski (1930). He then proved the theorem in (Tarski, 1998) whose planned publication in 1939 was delayed due to the war. Seidenberg (1954) published "A New Decision Method for Elementary Algebra", which applies Tarski's theorem to quantifier elimination problems over real closed fields, leading to the well-known Tarski–Seidenberg Theorem. Lojasiewicz (1964) gives a constructive proof of the Tarski–Seidenberg theorem using successive projections onto $n-1$-dimensional space. This is the idea behind the cylindrical algebraic decomposition (CAD) algorithm. The notion and algorithm for computing a CAD was presented by Collins (1975). The algorithm partitions $\mathbb{R}^n$ into cylindrical cells – semialgebraic sets such that the projection of any two of them is either disjoint or coincides. Collins' algorithm is the first computationally tractable proof of the Tarski–Seidenberg theorem, the algorithm having complexity doubly exponential in the number of variables. CAD has wide ranging applications in real algebraic geometry. Perhaps the most famous application is the one Collins was thinking of – quantifier elimination: given a Boolean formula where literals are polynomial equations and inequalities and some variables are bound by $\forall$ and $\exists$ quantifiers, return a Boolean formula without quantifiers representing the same set. A similar problem uses CAD to decide whether a formula in which all variables are quantified is true or false – the decision problem. More recently, alternative methods for performing quantifier elimination have been proposed, e.g., the critical point method (see Safey El Din (2013)). Other notable uses include determining whether a semialgebraic set is empty, computing the real roots of a system of multivariate polynomials and determining the dimension of a semialgebraic set. Basu et al. (1998) employs the critical point method to the emptiness problem, obtaining a bound singly exponential in the number of variables and Koiran (1999) proved that determining the dimension of a semialgebraic set is $\mathrm{NP}_{\mathbb{R}}$-complete, obtaining a singly exponential bound. CAD can also be used to deduce topological properties of semialgebraic sets, for example by computing homologies.

TO DO: maybe ask James about further applications of CAD.

The CAD algorithm is still an active and interesting area of study, with two main strands of research. Since the complexity of computing a CAD is doubly exponential in the number of variables, the algorithm can be intractable for even moderately-sized problems. It is impossible to lower this theoretical bound. Indeed Davenport and Heintz (1988) prove that quantifier elimination by CAD is doubly exponential in the number of variables. They present an example of a first-order Boolean formula with quantifiers which requires a linear number

of symbols while the formula obtained after quantifier elimination requires a doubly exponential number of symbols. Brown and Davenport (2007) later presented a simple and constructive proof of the same thing, giving an example of a formula with only one free variable and containing only linear polynomials whose quantifier-free counterpart is doubly exponential in length.

CAD depends on a fixed coordinate ordering and this can greatly influence the number of cells produced. In the same paper, Brown and Davenport (2007) show that the same input may produce a linear number of cells for one ordering, but a doubly exponential number of cells for another. They also present an example for which every coordinate ordering yields a doubly exponential number of cells (Brown and Davenport, 2007). In some applications, we may be able to change the coordinate ordering. As such, one way to "speed up" CAD construction would be to find a coordinate ordering which yields the smallest number of cells. Hong (1993) was one of the first to do this, using best first search, a heuristic search technique in AI, to find an optimal ordering. More recent developments include del Río and England (2022), who used complexity analysis to improve upon the current state-of-the-art heuristics, obtaining benchmark results 17 times slower than the theoretical best, while the state-of-the-art performs 25 times slower than the theoretical best. In addition, modern AI and machine learning techniques are being investigated as a means of finding the best coordinate ordering. For example, Huang et al. (2014) employed support vector machines and Chen et al. (2020) used artificial neural networks to identify an optimal ordering. Both of these techniques outperformed classical heuristics, making this a promising application of AI and ML.

However, in many cases we cannot change the coordinate ordering and, even if we could, Brown and Davenport (2007) showed that for some inputs all coordinate orderings yield a doubly exponential number of cells. As such, making optimisations in the CAD algorithm itself has been an active area of study since Collins published his original algorithm in the 70s. Collins' algorithm works on the principal of projection and lifting. First, successive projections of input polynomials are performed, until we obtain a set of polynomials in one variable. The roots of these polynomials are isolated and then "stacks" of cells above each root and each interval in between the roots are constructed. The CAD of n-dimensional space is obtained by iterating this process, called the "lifting phase". Minimising the number of polynomials produced by the projection operation means fewer computations are needed during the lifting phase. McCallum (1998) proposed an improved projection operator, observing that some polynomials included in Collins' original projection operator can be removed. Many other projection operators, e.g., Hong's (see Hong (1990)) Lazard's projection (see McCallum et al. (2019)) have been proposed, each offering different properties. Another optimisation is to construct a "partial CAD", proposed by Collins and Hong (1991). The idea is to use truth evaluation while constructing cells, taking quantifiers into account, so that only the cells we care about will be computed. As we do not have to ensure constant sign no every polynomial, this approach leads to savings during the lifting phase. The

CAD algorithm has also been modified, often by weakening its properties, to solve specific problems. E.g., McCallum (1993) presents an algorithm, based on CAD, for deciding the consistency of a system of strict polynomial inequalities. These modifications often have much better complexity than the classical CAD, while producing sufficient decompositions to solve the problem at hand.

A different strand of research, which is the main focus of this thesis, is to compute cylindrical decompositions in which the cells have certain desirable properties. This contrasts with the optimisations previously discussed, as these CADs often contain more cells than those constructed using the classical algorithm. However, this trade-off is worthwhile because the additional conditions make the CADs more powerful. Indeed, Schwartz and Sharir (1983) apply cylindrical algebraic decomposition to a motion planning problem in which one wants to find a path, or prove that none exists, for a rigid (possibly hinged) body to move through a space defined by a configuration of walls. This has obvious applications in robotics. In order to solve this problem, cells are required to be arranged in a particular way: the closure of every cell must be a union of cells in the decomposition. Using classical CAD, this condition cannot always be satisfied, but Schwartz and Sharir (1983) prove the classical algorithm is sufficient if a linear change of coordinates is made. These CADs are also useful if we wish to compute topological properties of semialgebraic sets. For example, Reif (1979) gives an algorithm for computing the homology groups of an arbitrary real algebraic variety. As we alluded to earlier, over the reals, a formula cannot tell us much about the set it defines, so this is useful for deducing properties of semialgebraic sets, e.g., how many "holes" it has, or whether it is connected. Along similar lines, Arnon et al. (1984) present an algorithm to compute a CAD of the plane along with information about which cells "touch each other". They extend this result to 3-dimensional space in (Arnon et al., 1987). In order to compute these cell adjacencies, we again need the property that the closure of every cell is a union of cells in the decomposition. Arnon et al. (1987) achieve this without making a change of coordinates.

Another desirable property is that every cell in a cylindrical decomposition is a topologically regular cell. Loosely speaking, this means that each k-dimensional cell and its closure is homeomorphic to a k-dimensional sphere and its closure. This property can fail in the region of points where polynomials are not "well behaved". E.g., points where a polynomial vanishes over an open interval (for the precise definition, see Definition 2.22). For example, consider the Whitney umbrella, the set of points in $\mathbb{R}^3$ which satisfy the equation $f = 0$ where

$$f := x^2 - y^2 z.$$

At almost every point $(x, y) \in \mathbb{R}^2$, f is well-behaved. having only one value of z at which it is equal to zero. However, f "blows up" above the origin, vanishing at every point. This causes interesting behaviour as (x, y) approaches the origin, because the z-coordinate tends to infinity. Thus, the set

$$\{0 < x < 1, -1 < y < 1, x^2 - y^2 z = 0\}, \tag{1.1}$$

which is a cylindrical cell in $\mathbb{R}^3$, is not topologically regular and its closure contains the half-line $\{x = 0, y = 0, z \geq 0\}$. We will study CAD containing blow-up points and cells which are not topologically regular throughout this thesis. Lazard (2010) presents an algorithm which produces such a decomposition, in the potential presence of these blow-up points, for dimension ≤ 3. This so-called "strong" cylindrical algebraic decomposition consists of only topologically regular cells and the closure of every cell is a union of some cells in the decomposition. Lazard (2010) takes an algebraic approach, employing techniques such as computing the saturation of a polynomial ideal, to make computations efficient and explains how the algorithm can be implemented in Maple. Many inputs to the CAD algorithm do not contain blow-up points. Davenport et al. (2020) prove that the classical CAD algorithm will produce a strong decomposition for the given coordinate ordering and regardless of the method used to compute it. This is a very useful result as it allows us to take advantage of the nice properties of strong decompositions without doing any extra work to obtain them. However, sometimes blow-up points are unavoidable. For example, Basu et al. (2015) propose an algorithm for computing a triangulation of a definable monotone family which relies on a strong cylindrical decomposition of that family. Since these families depend on a fixed coordinate ordering and frequently contain blow-up points, we must be able to compute these strong decompositions in arbitrary dimension, in the potential presence of blow-ups and without having to make a change of coordinates.

Note that, in Equation (1.1), we write a basic semialgebraic set of the form

$$\{\mathbf{x} \in \mathbb{R}^n \mid f_1(\mathbf{x}) = 0 \wedge \ldots \wedge f_k(\mathbf{x}) = 0 \wedge g_1(\mathbf{x}) > 0 \wedge \ldots \wedge g_\ell(\mathbf{x}) > 0\}$$

as

$$\{f_1(\mathbf{x}) = 0, \ldots, f_k(\mathbf{x}) = 0, g_1(\mathbf{x}) > 0, \ldots, g_\ell(\mathbf{x}) > 0\}.$$

I.e., commas represent conjunction. When not ambiguous, this convention will be used throughout.

Basu et al. (2015) present a constructive proof that, given a family $V := (V_1, \ldots, V_k)$ of bounded definable sets in $\mathbb{R}^n$ such that $\dim(V_i) \leq 2$ for all $1 \leq i \leq k$, there exists a cylindrical decomposition in which each $V_i, 1 \leq i \leq k$ is a union of cells, such that every cell contained in V_i is monotone and its closure is the union of cells in the decomposition. A monotone cell is a stronger property than topological regularity and, by Basu et al. (2013) Theorem 1, all monotone cells are topologically regular. Their proof explains how such a decomposition can be constructed and depends on classical CAD and simple tools such as refinements by linear equations and splitting two-dimensional cells by using a given curve interval. The main goal of this thesis will be to design and implement an algorithm, based on this result, for constructing such a decomposition in the semialgebraic case. The constructive proof works for more general definable sets and, in contrast to e.g. Collins (1975) or Lazard (2010), is geometric in nature. As such, the properties we need to satisfy in order to complete the construction are described in detail but not the methods used to obtain them. For example,

we are asked to consider critical points of the smooth 2-dimensional locus of a union of definable sets, or consider the boundary of a 2-dimensional cylindrical cell. We will use a combination of fundamental tools in differential geometry, e.g., partial derivatives and Jacobi determinants, well-known algorithms in algebraic geometry such as smooth stratification and quantifier elimination, and ideas from optimisation, such as Lagrange multipliers.

A similar idea to that presented by Basu et al. (2015) is discussed by Van den Dries (1998) (Section 2.19), who introduces the concept of a Van-den-Dries regular cell.

Definition 1.1. (Van den Dries, 1998, Section 2.19) We say that a cylindrical cell $C \subset \mathbb{R}^n$ is Van-den-Dries regular if, for all $i \in \{1, \ldots, n\}$, if two distinct points $\mathbf{x}, \mathbf{y} \in C$ are such that $x_1 = y_1, \ldots, x_{i-1} = y_{i-1}, x_{i+1}, \ldots, x_n = y_n$, but $x_i < y_i$, then for all z_i such that $x_i < z_i < y_i$, $(x_1, \ldots, x_{n-1}, z_i, x_{i+1}, \ldots, x_n) \in C$.

All two-dimensional Van-den-Dries regular cells are topologically regular, so if $\dim(V_1, \ldots, V_k)$, the decomposition described by Van den Dries (1998) will consist of topologically regular cells. The result of Basu et al. (2015) is stronger in that the monotone decomposition is obtained for a family of sets which individually have dimension at most two.

Part of the result from Basu et al. (2015) will be generalised to sets of arbitrary dimension. In particular, a novel algorithm for constructing a cylindrical decomposition, compatible with a semialgebraic set and suc that the boundary of every cell is the union of cells in the decomposition, will be presented. (This work was presented, in extended abstract form, at ISSAC 2023 and has also been submitted to the Journal of Symbolic computation.) It will be proved that such a decomposition can be constructed in any dimension and without a preliminary change of coordinates even in the presence of blow-ups. A CAD is compatible with a set S if S is the union of some cells of the CAD. Frontier condition means that the closure of every cell is a union of some cells in the decomposition. To our knowledge, this is the first proof that such a CAD exists without the change of coordinates. We present the proof in the form of an algorithm which constructs a CAD compatible with a semialgebraic set $S \subset \mathbb{R}^n$ and satisfying the frontier condition. An upper bound on complexity is obtained. This is also an upper bound on the number of cells, number of polynomials and degree of polynomials in the CAD. The algorithm has *elementary* complexity (in the sense of L. Kalmar, see e.g., (Kleene, 1952, §57)). This means that the complexity can be expressed as a power tower of finite height.

The frontier condition is useful in computing topological properties of semialgebraic sets defined by first-order Boolean formulas. For example, these decompositions can be viewed as closure-finite weak cell complexes (CW-complexes) and their homologies can be computed. In addition, cell adjacencies can easily be computed and motion-planning problems, for instance the well-known "piano-mover's problem" presented by Schwartz and Sharir can be solved, leading to

applications in robotics.

It was shown by Schwartz and Sharir (1983) that it is relatively easy to satisfy the frontier condition after a generic linear rotation of coordinates in $\mathbb{R}^n$. The rotation removes the need to lift over blow-up points. However, such a rotation of coordinates may be undesirable, or even impossible, in some applications.

An important application relates to the work of Basu et al. (2015), who present an algorithm for computing the triangulation of a definable monotone family. This algorithm requires a CAD of the family such that every cell is topologically regular and satisfies the frontier condition. Since the definable monotone family depends on the initial coordinate ordering, a rotation of coordinates is not allowed. Davenport et al. (2020) prove that if a CAD is *well-based* then it satisfies these properties. A CAD is well-based if it contains no *bad cells*: cells above which one of the polynomials defining the CAD vanishes identically. These occur frequently in definable monotone families. Therefore, an algorithm for computing CAD with frontier condition, compatible with sets containing blow-ups and without a change of coordinates, is needed. Such a CAD algorithm is presented by the authors for sets of dimension not greater than two. (Basu et al., 2015) The algorithm presented in this paper provides a step towards extending their result to decompositions compatible with sets (and potentially triangulations of definable monotone families) of arbitrary dimension.

In addition, the result can be applied to other categories, e.g., semialgebraic sets defined by *fewnomials* (see Section @ref{sec:pfaffian)), whose structure is destroyed by a change of coordinates.

The result can also be easily extended to semialgebraic sets defined by first-order Boolean formulas with quantifiers. Indeed, by Theorem @ref{thm:proj}, the projection of a CAD with frontier condition to any dimension also satisfies the frontier condition, and quantifier elimination requires considering projections of the CAD.

Arnon et al. (1984) also made important developments in constructing cylindrical algebraic decompositions with frontier condition. They propose an algorithm for constructing a "proper'' CAD of $\mathbb{R}^2$, a similar notion to the *well-based* CAD. Their algorithm also computes the cell adjacencies for this CAD. The same authors extend this result to proper CADs of $\mathbb{R}^3$, along with computing cell adjacencies.(Arnon et al., 1987) Lazard (2010) also presents an algorithm for obtaining a CAD in $\mathbb{R}^n$ for $n \leq 3$ satisfying the frontier condition without a change of coordinates. Lazard takes an algebraic approach and the complexity of the algorithm presented is the same as that of classical CAD, while our construction uses a recursion on the lexicographical order of cell indices. A novel approach was needed because it is not clear how the above results could be extended to dimension greater than 3.

Finally, we will also describe an algorithm for computing a smooth stratification of a semialgebraic set. A smooth stratification is a finite partition of a set into smooth manifolds called "strata". We present an algorithm, based on the work

of Gabrielov and Vorobjov (1995) for computing a smooth stratification of a semialgebraic set such that each stratum is nicely defined, meaning that each stratum of codimension k is defined by k different polynomials. The algorithm is designed to handle input sets which are not nicely defined. E.g., sets defined by polynomials whose first partial derivatives vanish at every point in the set. For example, the algebraic set

$$S := \{xy = 0\}$$

has codimension one, and thus can be defined by a single polynomial. However, it is not smooth, since $S' = (0, 0)$ is a singular point of S. S' has codimension 2 and it is clear that it can be written using two polynomials

$$S' = \{x = 0, y = 0\}.$$

Observe that x and y are the first derivatives of the defining function xy. However,

$$\{x^m y^n = 0, m \geq 1, n \geq 1\}$$

defines the set S and, if $m > 1$,

$$\frac{\partial x^m y^n}{\partial x} = m x^{m-1} y^n$$

is equal to zero at the same points as $x^m y^n$. Thus, it may not be sufficient to find the points at which only the first derivatives vanish. This is the basic idea of the smooth stratification algorithm we will present. The algorithm also ensures that only the k functions required to define a stratum of codimension k are included, "pruning" functions which do not add any additional information.

TO DO: applications of the smooth stratification?

1.1 Structure of this thesis

Chapter 2 gives some necessary definitions, then presents some useful results from Basu et al. (2015). The CAD is then formally defined, and the flow of the algorithm, following the description given by Coste (2000) is outlined. Chapter 3 presents the smooth stratification algorithm from Gabrielov and Vorobjov (1995), Theorem 2. The construction is made concrete, leading to an algorithm, presented as psuedo-code, for the semialgebraic case. This algorithm has been implemented in C on top of SACLIB. Saclib is an open-source C/C++ library for manipulating real polynomials and real algebraic numbers and working with their solutions as real algebraic numbers (see sac (2024)). The following is proved, the main contribution being the complexity result for the semialgebraic case.

Theorem 1.1. *Let*

$$X := \{\mathbf{x} \in \mathbb{R}^n \mid f_1 = 0, \ldots, f_k = 0, g_1 > 0, \ldots, g_\ell > 0\},$$

be a semialgebraic set defined by $s = k + \ell$ different polynomials of maximum degree d. Then there is an algorithm, without oracle, which partitions X into a family

$$\mathcal{X} = (X_0, \ldots, X_n)$$

such that, if X is nonsingular, $X_0 = X$ and all other sets are empty. Otherwise, $X_0 = \emptyset$ and each $X_k, 1 \leq k \leq n$ is a possibly empty, effectively nonsingular stratum of codimension k. This algorithm has complexity

$$3^s \left(s(d+1)\right)^{O(n)^2}.$$

The number of strata does not exceed $s^n(d+1)^2$, and each stratum is defined by at most $s(d+1)^2$ polynomial equations and inequalities of maximum degree $(d+1)^2$.

Chapters 4, 5 and 6.1 concern the construction from Basu et al. (2015), Theorem 3.20.

Proposition 1.1. *(Basu et al., 2015, Theorem 3.20)*

Let $V_1, \ldots, V_k$ be bounded definable subsets in $\mathbb{R}^n$ with $\dim V_i \leq 2$ for each $1 \leq i \leq k$. Then there is a cylindrical cell decomposition of $\mathbb{R}^n$ satisfying the frontier condition, and monotone with respect to each $V_1, \ldots, V_k$.

The main contribution is an algorithm for constructing the cylindrical decomposition described in Basu et al. (2015), Theorem 3.20, for the semialgebraic case.

Theorem 1.2. *Let*

$$F_1, \ldots, F_k$$

be quantifier-free Boolean formulas containing s different polynomials $\mathbf{F} = \{f_1, \ldots, f_s\} \in \mathbb{Z}[x_1, \ldots, x_n]$, having maximum degree d, such that $F_i, 1 \leq i \leq k$ defines a bounded semialgebraic set $V_i \subset \mathbb{R}^n$ such that $\dim(V_i) \leq 2$.

Then there is an algorithm, taking $F_1, \ldots, F_k$ as input, which constructs a cylindrical algebraic decomposition $\mathcal{D}$ of $\mathbb{R}^n$ satisfying the frontier condition, and monotone with respect to each $V_1, \ldots, V_k$. This algorithm has complexity

$$\left(s\,(d+1)\right)^{O(1)^n},$$

which is also an upper bound on the number of cells in the CAD, number of polynomials and their degrees.

Informally, an algorithm, which takes family of quantifier-free Boolean formulas representing some semialgebraic sets of dimension at most two as input and produces a cylindrical decomposition in which each of the input sets is a union of its cells. This decomposition has the property that each cylindrical cell contained

in one of the input sets is *monotone* (see Definition 2.18) and its closure is a union of cells in the decomposition. A complexity bound, only slightly higher than the "classical" CAD is obtained.

This algorithm follows the construction given in the proof of Basu et al. (2015), Theorem 3.20, which proceeds in three stages. First, a cylindrical cell decomposition of $\mathbb{R}^n$, compatible with each $V_1, \ldots, V_k$ and such that each cell $C \subset V_1 \cup \ldots \cup V_k$ is the graph of a quasi-affine map. In Chapter 4, an algorithm, following Basu et al. (2015), Lemma 3.19 is presented for the semialgebraic case. The following is proved.

Theorem 1.3. *Let*

$$F_1, \ldots, F_k$$

be quantifier-free Boolean formulas containing s different polynomials $\mathbf{F} = \{f_1, \ldots, f_s\} \in \mathbb{Z}[x_1, \ldots, x_n]$, having maximum degree d, such that $F_i, 1 \leq i \leq k$ defines a bounded semialgebraic set $V_i \subset \mathbb{R}^n$ such that $\dim(V_i) \leq 2$.

Then there is an algorithm, taking $F_1, \ldots, F_k$ as input, which constructs an $\mathbf{F}$-invariant CAD $\mathcal{D}$ of $\mathbb{R}^n$, which is obviously compatible with each $V_1, \ldots, V_k$, such that each cell $C \subset V_1 \cup \ldots \cup V_k$ is a smooth manifold and the graph of a quasi-affine map. This algorithm has complexity

$$(s\,(d+1))^{O(1)^n},$$

which is also an upper bound on the number of cells in the CAD, number of polynomials and their degrees.

Chapter 5 discusses, following the proofs of Basu et al. (2015), Theorem 3.20 and Theorem 3.18, how the CAD $\mathcal{D}$ produced by the algorithm from Theorem 4.1 can be refined such that cells in the refinement are monotone with respect to each $V_1, \ldots, V_k$. An algorithm for constructing this refinement, starting with the polynomials which have constant sign on cells of $\mathcal{D}$, is presented.

Theorem 1.4. *Let*

$$F_1, \ldots, F_k$$

be quantifier-free Boolean formulas such that $F_i, 1 \leq i \leq k$ defines a bounded semialgebraic set $V_i \subset \mathbb{R}^n$ such that $\dim(V_i) \leq 2$. Let $\mathbf{F} = \{f_1, \ldots, f_s\}$ be a set of polynomials in $\mathbb{Z}[x_1, \ldots, x_n]$ of maximum degree d and $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$ compatible with each $V_1, \ldots, V_k$ and such that each cell $C \subset V_1 \cup \ldots \cup V_k$ is a smooth manifold and the graph of a quasi-affine map.

Then there is an algorithm, taking $\mathcal{D}$ as input, which produces a refinement $\mathcal{D}'$ of $\mathcal{D}$ monotone with respect to each $V_1, \ldots, V_k$. This algorithm has complexity

$$(s\,(d+1))^{O(1)^n},$$

which is also an upper bound on the number of cells in the CAD, number of polynomials and their degrees.

Finally, Chapter 6 describes how to complete the construction described in Basu et al. (2015), Theorem 3.20, by refining the CAD such that it satisfies the frontier condition. Section 6.1 describes how the construction given in the last part of the proof of Basu et al. (2015), Theorem 3.20 can be applied to the CAD $\mathcal{D}'$, produced by the algorithm from Theorem 5.1 to produce a refinement $\mathcal{D}''$ of $\mathcal{D}'$ such that the closure of every cell of $\mathcal{D}'$ contained in $V_1 \cup \ldots \cup V_k$ is the union of some cells of $\mathcal{D}''$. In Section 6.2, an algorithm, due to Lazard (2010), for constructing an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n, n \leq 3$ satisfying the frontier condition and containing only topologically regular cells is presented. Section 6.3 explores how this method can be generalised to our situation. In order to do this, the results of Lazard (2010), Section 5.3 are generalised from $\mathbb{R}^3$ to the two-dimensional cells in a CAD of $\mathbb{R}^n, n > 3$. The following is proved.

Theorem 1.5. *Let $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$ be a set of s polynomials with maximum degree d. Let $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$, monotone with respect to each bounded semialgebraic set $V_1, \ldots, V_k$ ($\dim(V_i) \leq 21 \leq i \leq k$).*

Then there is an algorithm, taking $\mathcal{D}$ as input, which produces a refinement $\mathcal{D}'$ of $\mathcal{D}$ such that each cell $C \subset V_1 \cup \ldots \cup V_k$ of $\mathcal{D}'$ satisfies the frontier condition. This algorithm has complexity...

$$(s\,(d+1))^{O(1)^n},$$

which is also an upper bound on the number of cells in the CAD, number of polynomials and their degrees.

All of these algorithms are presented as psuedo-code. Chapter 7 discusses the implementation of this algorithm on top of Brown's QEPCAD (Brown, 2003) and presents some test cases.

The final contribution, in Chapter 8, is a novel algorithm for constructing a CAD of $\mathbb{R}^n$, compatible with a semialgebraic set of arbitrary dimansion and satisfying the frontier condition. This algorithm relies on a recursion on the lexicographical order of indices $(i_1, \ldots, i_n) \in \{0, 1\}^n$ associated with each cell in a cylindrical decomposition. The following is proved.

Theorem 1.6. *Let $S \subset \mathbb{R}^n$ be a semialgebraic set defined by a quantifier-free Boolean formula F with s different polynomials of maximum degree d in $\mathbb{R}[x_1, \ldots, x_n]$. There is an algorithm, taking F as input, which outputs a cylindrical decomposition $\mathcal{D}$ of $\mathbb{R}^n$ compatible with S and satisfying the frontier condition. The complexity of this algorithm is $(sd)^{O(1)^{n2^n}}$. This is also an upper bound on the number of cells in $\mathcal{D}$, number of polynomials defining cells and their degrees.*

Finally, Chapter 9 discusses conclusions and further work.

Chapter 2

Background

2.1 Semialgebraic sets and monotone cells

2.1.1 Sets definable over O-minimal structures

We will be working mostly in the category *Semialgebraic Set* and sometimes in the more general category of sets definable over an O-minimal structure. A semialgebraic subset of $\mathbb{R}^n$ is the set of points in $\mathbb{R}^n$ satisfying a Boolean combination of polynomial equations and inequalities with coefficients in $\mathbb{R}$. A polynomial with coefficients in a field K and variables $x_1, \ldots, x_n$ will be denoted as an element of $K[x_1, \ldots, x_n]$.

Definition 2.1. The semialgebraic subsets of $\mathbb{R}^n$ are the smallest class SA_n of subsets of $\mathbb{R}^n$ satisfying the following properties.

1. If $f \in \mathbb{R}[x_1, \ldots, x_n]$, then $\{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) = 0\}$ and $\{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) > 0\}$ are elements of SA_n.
2. If $A, B \in SA_n$, then $A \cup B$, $A \cap B$ and $\mathbb{R}^n \setminus A$ are elements of SA_n.

Definition 2.2. Let $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$ be definable sets. Let $f : X \to Y$ be a mapping from X to Y. The graph of f is the definable set

$$\{(\mathbf{x}, f(\mathbf{x})) \in \mathbb{R}^{n+m} \mid \mathbf{x} \in X\}.$$

Sometimes, the domain of f is $\mathbb{R}^n$. In this case, we often use the restriction $f|_X$ so that only images of f at points in X are considered, The graph of f on X may be denoted by $f(X)$.

Definition 2.3. Let $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$. A mapping $f : X \to Y$ is called semialgebraic if its graph (see Definition 2.2)

$$\{(\mathbf{x}, f(\mathbf{x})) \in \mathbb{R}^{n+m} \mid \mathbf{x} \in X\}$$

is a semialgebraic set.

A semialgebraic set can be represented as a first-order Boolean formula with or without quantifiers. A first-order formula which does not contain quantifiers will be called a quantifier-free Boolean formula.

Definition 2.4. A quantifier-free Boolean formula is obtained by the following rules:

1. If $f \in \mathbb{R}[x_1 \ldots, x_n]$ then $f = 0$ and $f > 0$ are quantifier-free Boolean formulas,
2. if F and G are quantifier-free Boolean formulas then $F \wedge G$, $F \vee G$ and $\neg F$ are quantifier-free Boolean formulas.

Definition 2.5. A first-order Boolean formula is obtained by rules 1 and 2 from Definition 2.4 and

3. If F is a first-order Boolean formula and y is a variable ranging over $\mathbb{R}$, then $\exists y\ F$ and $\forall y\ F$ are first-order Boolean formulas.

Semialgebraic sets are closed under taking finite unions and intersections and the difference of two semialgebraic sets is semialgebraic. A fundamental result in semialgebraic geometry is the Tarski–Seidenberg Theorem, which asserts that semialgebraic sets are also closed under taking projection. The proof of Tarski's theorem was published in (Tarski, 1998), whose release was delayed due to the war. It is very likely that Tarski actually proved the theorem much earlier, since Tarski mentioned this result in Sierpiński and Tarski (1930). The following statement of the theorem comes from (Van den Dries et al., 1988), but the theorem should obviously be attributed to Tarski.

Theorem 2.1. *(Van den Dries et al., 1988) Let $S \subset \mathbb{R}^{n+1}$ and $\mathrm{proj}_{\mathbb{R}^n} : \mathbb{R}^{n+1} \to \mathbb{R}^n$ be the projection map onto the first n co-ordinates. Then $\mathrm{proj}_{\mathbb{R}^n}(S) \subset \mathbb{R}^n$ is a semialgebraic set.*

One important consequence of this theorem is that quantifier elimination is possible over real numbers. In other words, a subset of $\mathbb{R}^n$ defined by a first-order formula with quantifiers is semialgebraic and can be represented by a quantifier-free Boolean formula. Since the closure of a semialgebraic set may be represented by a first-order Boolean formula with quantifiers, it follows from the Tarski–Seidenberg Theorem that the closure of a semialgebraic set is a semialgebraic set.

We present an algorithm for performing quantifier elimination over real numbers. This algorithm will be used to provide theoretical complexity bounds for solving quantifier elimination problems later.

Proposition 2.1. *(Basu et al., 2006, Algorithm 14.21)*

Let $y_1 < \ldots < y_k < x_1 < \ldots < x_k$ be an ordered list of variables and K a real closed field. Define a first-order Boolean formula

$$F := Q_1(y_1, \ldots, y_{k_1}), Q_2(y_{k_1+1}, \ldots, y_{k_1+k_2}) \ldots Q_\omega(y_{y_1+\ldots+y_{\omega-1}+1}, \ldots, y_k)G$$

where $Q_i, 1 \leq i \leq \omega$, is a quantifier $\forall$ or $\exists$, such that Q_i and Q_{i+1} are different and G is a quantifier-free Boolean formula containing s different polynomials of maximum degree d in $K[y_1, \ldots, y_k, x_1, \ldots, x_n]$. Note that variables $y_1, \ldots, y_k$ are partitioned into ω "blocks" $Y_1, \ldots, Y_\omega$ such that $Y_i, 1 \leq i \leq \omega$ contains k_i variables, each of which is bound by the same quantifier Q_i. We say that the formula F contains ω quantifier alternations.

Let $S \subset \mathbb{R}^n$ be a semialgebraic set defined by F. There is a quantifier-free Boolean formula defining s of the form

$$F' := \vee_{1 \leq i \leq I} \wedge_{1 \leq j \leq J_i} \left(\vee_{1 \leq \ell \leq L_{i,j}} f_{i,j,\ell}(x_1, \ldots, x_n) *_{i,j,\ell} 0 \right)$$

where each $f_{i,j,n}$ is a polynomial in $K[x_1, \ldots, x_n]$ and $*_{i,j,\ell} \in \{<, =, >\}$ determines its sign. Bounds on this formula are as follows

- $I \leq s^{(k_1+1)(k_2+1)\ldots(k_\omega+1)(n+1)} d^{O(k_1)O(k_2)\ldots O(k_\omega)}$,
- $J_i \leq s^{(k_1+1)(k_2+1)\ldots(k_\omega+1)} d^{O(k_1)O(k_2)\ldots O(k_\omega)}$,
- $L_{i,j} \leq d^{O(k_1)O(k_2)\ldots O(k_\omega)}$,
- *polynomials $f_{i,j,\ell}$ have maximum degree $d^{O(k_1)O(k_2)\ldots O(k_\omega)}$.*

There is an algorithm, taking F as input, which computes F' with complexity

$$s^{(k_1+1)(k_2+1)\ldots(k_\omega+1)(n+1)} d^{O(k_1)O(k_2)\ldots O(k_\omega)O(n)}.$$

In many quantifier elimination problems, there is only a small number of quantifier alternations and this algorithm gives a favourable complexity bound (better than, e.g., using CAD). For example, the formula for computing the closure of a semialgebraic set contains only two quantifier alternations, and this algorithm gives a complexity bound singly exponential in the number of variables (see Lemma 8.1).

Another useful property of semialgebraic sets is that they admit a basic representation, as stated below.

Every semialgebraic subset of $\mathbb{R}^n$ is the (not necessarily disjoint) union of finitely many semialgebraic subsets of the kind

$$\{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) = 0, g_1(\mathbf{x}) > 0, \cdots, g_k(\mathbf{x}) > 0\}$$

where $k \in \mathbb{N}$ and $f, g_1, \ldots, g_k \in \mathbb{R}[x_1, \ldots, x_n]$. :::

This follows from the fact that the class of finite unions of this kind satisfy properties listed in Definition 2.1.

We now define the O-minimal structures over the reals.

Definition 2.6. A structure expanding the real closed field R is a collection

$$S := (S^n)_{n\in\mathbb{N}}$$

where each S^n is a set of subsets of the affine space R^n satisfying the following conditions.

1. All algebraic subsets of R^n are in S^n.
2. S^n is a Boolean subalgebra of the powerset of R^n.
3. If $A \in S^n$ and $B \in S^m$, then $A \times B \in S^{n+m}$.
4. Let $\text{proj} : R^{n+1} \to R^n$ be the projection onto the first n coordinates. If $X \in R^{n+1}$, then $\text{proj}(X) \in R^n$.

Elements of S^n (satisfying properties 1-4) are called the definable subsets of R^n. If the structure S satisfies properties 1-4 and the following property, it is called an O-minimal structure.

5. The elements of S^1 consist of the finite unions of points and open intervals of R.

Definition 2.7. A map $f : A \to R^m$, where $A \subset R^n$, is called *definable* if its graph $G \subset R^{n+m}$ is a definable set.

Remark. Since definable sets are closed under projection, it can be easily deduced that A is a definable set.

Remark. For convenience, we will call a definable map $f : R^n \to R$ a definable function.

It is easy to see that semialgebraic sets satisfy the properties of sets definable in an O-minimal structure over the reals. Another example is the sub-pfaffian sets.

Definition 2.8. (Gabrielov and Vorobjov, 2004, Definition 2.1) A Pfaffian chain of order $r \geq 0$ and degree $\alpha \geq 1$ in an open domain $G \subset \mathbb{R}^n$ is a sequence of analytic functions

$$f_1, \ldots, f_r$$

in G satisfying the differential equations

$$df_j(\mathbf{x}) = \sum_{i=1}^{n} g_{ij}(\mathbf{x}, f_1(\mathbf{x}), \ldots, f_j(\mathbf{x}))dx_i$$

for $1 \leq j \leq r$, where $g_{ij}(x_1, \ldots, x_n), y_1, \ldots, y_j)$ are polynomials of degree not greater than α.

A function $f(\mathbf{x}) = P(\mathbf{x}, f_1(\mathbf{x}), \ldots, f_r(\mathbf{x}))$, where $P(\mathbf{x}, f_1(\mathbf{x}), \ldots, f_r(\mathbf{x}))$ is a polynomial of degree not greater than $\beta \geq 1$ is called a Pfaffian function of order r and degree (α, β). Note that f is only defined in the domain G, where all functions $f_1, \ldots, f_r$ are analytic, even if f itself can be extended as an analytic function in a larger domain.

Examples of Pfaffian functions include the exponential, logarithmic, reciprocal and trigonometric functions, as well as the polynomials.

Definition 2.9. (Gabrielov and Vorobjov, 2004, Definition 2.7) A set $X \subset \mathbb{R}^n$ is called semi-Pfaffian in an open domain $G \subset \mathbb{R}^n$ if it consists of points in G satisfying a Boolean combination F of some atomic equations and inequalities $f = 0$, $g > 0$, where f, g are Pfaffian functions having a common Pfaffian chain defined in G.

X is called restricted in G if $\mathrm{cl}\,(X)$ is contained in G.

X is called basic if the Boolean combination F is a conjunction.

Definition 2.10. (Gabrielov and Vorobjov, 2004, Definition 2.8) A set $X \subset \mathbb{R}^n$ is called sub-Pfaffian in an open domain $G \subset \mathbb{R}^n$ if

$$X := \mathrm{proj}_{\mathbb{R}^n}(Y),$$

where $Y \subset \mathbb{R}^m$ is a semi-Pfaffian set and $\mathbb{R}^n$ is a subspace of $\mathbb{R}^m$.

Definition 2.11. (Gabrielov and Vorobjov, 2004, 2.9) Let $\mathcal{I}^k := [-1, 1]^k$ be the closed cube in an open domain $G \subset \mathbb{R}^k$. $X \subset \mathcal{I}^n$ is called restricted sub-Pfaffian if $X := \mathrm{proj}_{\mathbb{R}^n}(Y)$, where $Y \subset \mathcal{I}^{n+m}$ is a restricted semi-Pfaffian set.

The restricted sub-pfaffian sets form a Boolean subalgebra. Restricted sub-pfaffian sets are clearly closed under taking finite unions and intersections. The property that the compliment of restricted sub-pfaffian set is restricted sub-pfaffian is a particular case of Gabrielov's compliment theorem (Gabrielov, 1996). The restriction to the closed cube ensures that the restricted sub-Pfaffian sets are definable in an O-minimal structure. Furthermore, Gabrielov and Vorobjov (2001) present an algorithm for constructing a cylindrical decomposition compatible with a restricted sub-Pfaffian set. Note that these properties and results may not be applied to (restricted) semi-Pfaffian sets. Since polynomials are a type of Pfaffian function, semialgebraic sets are a particular case of sub-Pfaffian sets and, due to the Tarski–Seidenberg Theorem, semialgebraics sets satisfy the compliment theorem.

Definition 2.12. (Gabrielov and Vorobjov, 2004, Definition 2.11)

Consider a semi-Pfaffian set

$$X := \bigcup_{1 \le i \le M} \{\mathbf{x} \in \mathbb{R}^s \mid f_{i,1} = 0, \ldots, f_{i,I_i} = 0, g_{i,1} > 0, \ldots, g_{i,J_i} > 0\} \subset G$$

where f_{ij} and g_{ij} are Pfaffian functions of order r and degree (α, β) defined in an open domain G The format of X is a tuple

$$(r, N, \alpha, \beta, s)$$

where $N \ge (I_1 + J_1) + \cdots + (I_M + J_M)$.

2.1.2 Monotone cells

We now present, closely following Basu et al. (2015), some useful properties of definable sets. These properties can also be applied to semialgebraic sets, being a particular case of definable sets.

Definition 2.13. (Basu et al., 2015, Definition 2.1) Let $L_{j,c} := \{(x_1, \ldots, x_n) \mid x_j = c\}$ for some $1 \leq j \leq n$ and $c \in \mathbb{R}$. Each intersection of the kind

$$S := L_{j_1,c_1} \cap \ldots \cap L_{j_m,c_m}$$

where $0 \leq j \leq m$, $1 \leq j_1 < \cdots < j_m \leq n$ and $c_1, \ldots, c_m \in \mathbb{R}$ is called an affine coordinate subspace of $\mathbb{R}^n$.

Definition 2.14. (Basu et al., 2015, Definition 2.2) Let $\mathbf{f} = (f_1, \ldots, f_k) : X \to \mathbb{R}^k$ be a bounded continuous map defined on an open, bounded, non-empty subset $X \subset \mathbb{R}^n$ having the graph $Y \subset \mathbb{R}^{n+k}$.

$\mathbf{f}$ is called quasi-affine if, for any coordinate subspace $L \subset \mathbb{R}^{n+k}$, the restriction $\mathrm{proj}_L|_Y$ is injective if and only if $\mathrm{proj}_L(Y)$ is n-dimensional.

Definition 2.15. (Basu et al., 2015, Definition 2.3) Let $\mathbf{f} : X \to \mathbb{R}^k$ be a bounded, continuous, quasi-affne map defined on an open, bounded, non-empty subset $X \subset \mathbb{R}^n$ having the graph $Y \subset \mathbb{R}^{n+k}$. $\mathbf{f}$ is called monotone if, for each affine coordinate subspace $S \subset \mathbb{R}^{n+k}$, the intersection $Y \cap S$ is connected.

Definition 2.16. (Basu et al., 2015, Notation 2.4)

Let the space $\mathbb{R}^n$ have coordinate functions $x_1, \ldots, x_n$. Given a subset

$$I = \{x_{j_1}, \ldots, x_{j_m}\} \subset \{x_1, \ldots, x_n\},$$

let W be the linear subspace of $\mathbb{R}^n$ where all coordinates in I are equal to zero. We write $\mathrm{span}\{x_{j_1}, \ldots, x_{j_m}\}$ to mean the quotient space $\mathbb{R}^n/W$. Similarly, for any affine coordinate subspace $L \subset \mathbb{R}^n$ on which all the functions $x_j \in I$ are constant, we will identify L with its image under the canonical surjection to $\mathbb{R}^n/W$. We write $\mathbb{R}^k$, where $k \leq n$, to mean $\mathrm{span}\{x_1, \ldots, x_k\}$.

Definition 2.17. (Basu et al., 2015, Definition 2.10) Let $X \subset \mathrm{span}\{x_1, \ldots, x_n\}$ be a monotone cell and $\mathbf{f} : X \to \mathrm{span}\{y_1, \ldots, y_k\}$ be a continuous map having graph $Y \subset \mathrm{span}\{x_1, \ldots, x_n, y_1, \ldots, y_k\}$. If Y is a monotone cell, then $\mathbf{f}$ is called monotone on X.

Definition 2.18. (Basu et al., 2015, Definition 2.5) $Y \subset \mathrm{span}\{x_1, \ldots, x_n\}$ is called a monotone cell if it is the graph of a monotone map $\mathbf{f} : X \to H$ where $H \subset \mathrm{span}\{x_1, \ldots, x_n\}$ and $X \subset \mathrm{span}\{x_1, \ldots, x_n\} \setminus H$.

Example 2.1.

1. Observe that any quasi-affine map defined on a connected subset and having the graph in $\mathbb{R}^2$ is also monotone. This quasi-affine map will fail to be monotone if the set it is defined on is not connected.

2. Let $\Delta := \{0 < x < 1, y > 0, x + y < 1\} \subset \mathbb{R}^2$ and $\varphi(x, y) = x^2 + y^2$. Let $Y \subset \mathbb{R}^3$ be the graph of function ϕ on Δ. Observe that $\phi|_Delta$ is a quasi-affine map. However, $y \cap \{z = 3/4\}$ is not connected, hence $\phi|_\Delta$ is not a monotone function (Basu et al., 2015, Example 4.3).

Proposition 2.2. *(Basu et al., 2013, Theorem 1)*

Every monotone cell is a topologically regular cell.

The following properties of monotone cells are used in the proof of Basu et al. (2015), Theorem 3.20.

Corollary 2.1. *(Basu et al., 2013, Corollary 7, Theorem 11) Let $X \subset \mathbb{R}^n$ be a monotone cell, then*

1. *$X \cap \{x_i < c\}$, $X \cap \{x_i = c\}$ and $X \cap \{x_i > c\}$, for every $1 \leq i \leq n$ and $c \in \mathbb{R}$, are either empty or monotone cells.*
2. *Let $Y \subset X$ be a monotone cell such that $\dim(Y) = \dim(X) - 1$ and $\mathrm{fr}\,(Y) \subset \mathrm{fr}\,(X)$. Then $X \setminus Y$ is a disjoint union of two monotone cells.*

Proposition 2.3. *(Basu et al., 2013, Theorem 10)*

Let $X \subset \mathbb{R}^n$ be a monotone cell. Then $\mathrm{proj}_L(X)$, for any affine coordinate subspace $L \subset \mathbb{R}^n$ is also a monotone cell.

Remark. (Basu et al., 2015, Remark 2.11) Let $Y \subset \mathbb{R}^n$ be a monotone cell and L be a coordinate subspace such that $\mathrm{proj}_L|_Y$ is injective. Then Y is the graph of a monotone map defined on $\mathrm{proj}_L(Y)$, by (Basu et al., 2013, Theorem 7 and Corollary 5).

We finally revise the definition of Big O notation, following Gabrielov and Vorobjov (2009), Definition 6.1.

Definition 2.19. (Gabrielov and Vorobjov, 2009, Definition 6.1) Let $f, g, h : \mathbb{N}^\ell \to \mathbb{N}$ be functions and $n \in \mathbb{N}$.

- The expression $f \leq O(g)^n$ means that there exists $c \in \mathbb{N}$ such that $f < (cg)^n$ at every point in $\mathbb{N}^\ell$.
- The expression $f \leq g^{O(h)}$ means that there exists a $c \in \mathbb{N}$ such that $f \leq g^{ch}$ at every point in $\mathbb{N}^\ell$.

2.2 Cylindrical algebraic decomposition

We now give some background on Cylindrical Algebraic Decomposition (CAD). Collins (1975) introduced both the concept of Cylindrical Algebraic Decomposition along with an algorithm to construct a CAD, which is a finite partition of $\mathbb{R}^n$ into so-called Cylindrical Cells, such that each polynomial in $\mathbf{F} \subset \mathbb{R}[x_1, \ldots, x_n]$ has a constant sign on every cylindrical cell in the decomposition.

Before describing Collins' algorithm, we will first define, closely following Basu et al. (2015), a cylindrical cell and cylindrical decomposition. Note that Basu et al. (2015) work in the broader class of definable sets.

Definition 2.20. Cylindrical cells are defined by induction on $n \geq 1$. Each cylindrical cell C is a connected definable subset of $\mathbb{R}^n$ with an *index*

$$(i_1, \ldots, i_n) \in \{0, 1\}^n.$$

- When $n = 0$, there is a unique cylindrical cell $\mathbf{0} \in \mathbb{R}^0$.
- When $n = 1$, a (0)-cell (section cell) is a point $c \in \mathbb{R}$ and a (1)-cell (sector cell) is an open interval: (a, b), $(-\infty, b)$, (a, ∞), $(-\infty, \infty)$ where $a, b \in \mathbb{R}$.
- When $n > 1$, suppose that $(i_1, \ldots, i_{n-1})$-cells are already defined and let C' be one of these cells. An $(i_1, \ldots, i_{n-1}, 0)$-cell (section cell) is the graph of a continuous definable function $f : C' \to \mathbb{R}$. An - $(i_1, \ldots, i_{n-1}, 1)$-cell (sector cell) is a subset of the cylinder $C' \times \mathbb{R}$, either

$$\begin{aligned}
&\{(\mathbf{x}, t) \mid \mathbf{x} \in C', f(\mathbf{x}) < t < g(\mathbf{x})\},\\
&\{(\mathbf{x}, t) \mid \mathbf{x} \in C', -\infty < t < g(\mathbf{x})\},\\
&\{(\mathbf{x}, t) \mid \mathbf{x} \in C', f(\mathbf{x}) < t < \infty\},\\
&\{(\mathbf{x}, t) \mid \mathbf{x} \in C', t \in \mathbb{R}\}
\end{aligned}$$

  where $f, g : C' \to \mathbb{R}$ are continuous definable functions such that $f(\mathbf{x}) < g(\mathbf{x})$ for all $\mathbf{x} \in C'$.

Definition 2.21. Let $S \subset \mathbb{R}^n$ be a definable subset of $\mathbb{R}^n$ and define the *frontier of* S

$$\mathrm{fr}\,(S) := \mathrm{cl}\,(S) \setminus S,$$

where $\mathrm{cl}\,(S)$ is the closure of S in the Euclidean topology.

If $C \subset \mathbb{R}^n$ is a cylindrical cell, then it will be convenient to partition $\mathrm{fr}\,(C)$ into three subsets: the top (C_T), bottom (C_B) and side-wall (C_W).

First let C be a sector $(i_1, \ldots, i_{n-1}, 1)$-cell and let $C' := \mathrm{proj}_{\mathbb{R}^{n-1}}(C)$. If C is bounded from below by a continuous function $f : C' \to \mathbb{R}$, then C_B is the graph of f. Similarly, if C is bounded from above by the graph of a continuous function $g : C' \to \mathbb{R}$, then C_T is the graph of g. It is clear that the top and bottom of every cylindrical sector cell (if non-empty) are always cylindrical section cells.

Now let C be an $(i_1, \ldots, i_k, 0, \ldots, 0)$-cell, where $i_k = 1$, and $C' := \text{proj}_{\mathbb{R}^k}(C)$. Since C' is a sector cell, C'_B and C'_T are cylindrical section cells. The bottom C_B of C is the pre-image of C'_B by the projection map $\text{proj}_{\mathbb{R}^k}|_{\text{cl}(C)}$ and the top C_T of C is the pre-image of C'_T by the projection map $\text{proj}_{\mathbb{R}^k}|_{\text{cl}(C)}$. It may be simpler to think of C_B as the definable set $\text{cl}(C) \cap (C'_B \times \mathbb{R}^{n-k+1})$ and C_T as the definable set $\text{cl}(C) \cap (C'_T \times \mathbb{R}^{n-k+1})$. It is important to note that C_B and C_T may not be cylindrical cells. Indeed, they may fail to be graphs of continuous functions (see Definition 2.22 and Example 8.1.

For both section and sector cells, let the side wall, C_W, be $\text{fr}(C) \setminus (C_B \cup C_T)$. There is no requirement that C_W will be (a union of) cylindrical cells.

Definition 2.22. (Basu et al., 2015, Definition 9.2)

Let $\varphi : W \to \mathbb{R}$ be a continuous function on a semialgebraic set $W \subset \mathbb{R}^n$, and let $\Phi \subset \mathbb{R}^{n+1}$ be its graph.

A point $\mathbf{x} \in \overline{W} \setminus W$ is called a blow-up point of Φ if

$$\text{proj}^{-1}_{\mathbb{R}^n}(\mathbf{x}) \cap \overline{\Phi}$$

contains an open interval.

Remark. The closure of the graph of a continuous definable function is the graph of a continuous definable function everywhere except, possibly, over a subset of codimension at least 2 (see the proof of Basu et al. (2015), Lemma 3.14).

Example 2.2. In $\mathbb{R}^3$, let $C' := \{-1 < x < 1, |x| < y < 1\}$ and $\varphi(x, y) = |x/y|$ so that C is the graph of $\varphi|_{C'}$. Consider

$$C_B := \{-1 < x < 1, y = |x|, z = 1\} \cup \{(0, 0, z) \mid 0 \leq x \leq 1\}.$$

C_B is not the graph of a continuous function due to the blow-up point of φ at the origin.

We now define the cylindrical decomposition.

Definition 2.23. A cylindrical decomposition is defined by induction on $n \geq 0$.

- When $n = 0$, the unique cylindrical decomposition of $\mathbb{R}^0$ is the unique point in $\mathbb{R}^0$.
- When $n > 0$: Let $\mathcal{D}$ be a partition of $\mathbb{R}^n$ into cylindrical cells. Define $\mathcal{D}'$ to be the set of all projections $C' := \text{proj}_{\mathbb{R}^{n-1}}(C)$ for all C in $\mathcal{D}$. $\mathcal{D}$ is a cylindrical decomposition of $\mathbb{R}^n$ if $\mathcal{D}'$ is a cylindrical decomposition of $\mathbb{R}^{n-1}$. We call $\mathcal{D}'$ the decomposition of $\mathbb{R}^{n-1}$ induced by $\mathcal{D}$.

Less formally, a partition $\mathcal{D}$ of $\mathbb{R}^n$ into definable subsets is cylindrical if the projection of any two cells of $\mathcal{D}$ onto $\mathbb{R}^{n-1}$ is either disjoint or coincides.

Definition 2.24. Let $S \subset \mathbb{R}^n$ be a definable set. A cylindrical decomposition $\mathcal{D}$ is compatible with S if every cell C of $\mathcal{D}$ is either a subset of S or disjoint from S.

Definition 2.25. A cylindrical decomposition $\mathcal{E}$ is called a refinement of $\mathcal{D}$ if $\mathcal{E}$ is compatible with every cell of $\mathcal{D}$. in other words, every cell of $\mathcal{D}$ is a union of cells in the refinement $\mathcal{E}$.

Remark. (Basu et al., 2015, Remark 3.8) Let $\mathcal{D}$ be a cylindrical decomposition of $\mathbb{R}^n$ and C be a cylindrical cell of $\mathcal{D}$ such that $c := \text{proj}_{\mathbb{R}^1}(C)$ is a single point. Then it follows immediately from the definition that $\mathcal{D}$ is compatible with the hyperplane $\{x_1 = c\}$ and the set of all cells of $\mathcal{D}$ contained in this hyperplane form a cylindrical decomposition $\mathcal{E}$ of $\mathbb{R}^{n-1}$. Moreover, any refinement of $\mathcal{E}$ is also a refinement of $\mathcal{D}$.

We now present some useful results proved by Basu et al. (2015) relating to desirable properties of cylindrical decompositions.

Definition 2.26. Let $S \subset \mathbb{R}^n$ be a definable set. A cylindrical decomposition $\mathcal{D}$ of $\mathbb{R}^n$, compatible with S, is called monotone with respect to S if every cell $C \subset S$ of $\mathcal{D}$ is monotone.

Definition 2.27. Let $\mathcal{D}$ be a cylindrical decomposition of $\mathbb{R}^n$ and C be a cell of $\mathcal{D}$. C satisfies the frontier condition in $\mathcal{D}$ is $\text{fr}\,(C)$ is a union of cells of $\mathcal{D}$ of smaller dimension. If it is unambiguous that C is a cell of $\mathcal{D}$, then we just say that C satisfies the frontier condition. If every cell in $\mathcal{D}$ satisfies the frontier condition, then the cylindrical decomposition $\mathcal{D}$ satisfies the frontier condition.

We saw in Example 8.1 that the top and bottom of a 2-dimensional cylindrical section cell, even in $\mathbb{R}^3$, need not be a union of cylindrical cells. However, it is clear from the cylindrical property that, if C is a 2-dimensional section cell satisfying the frontier condition in a decomposition $\mathcal{D}$, then the top and bottom of C are single cylindrical cells. If C is a 2-dimensional section cell, then it is said to satisfy the strong frontier condition if the one-dimensional components of its side-wall are 1-dimensional section cells. This condition may be impossible, even in $\mathbb{R}^3$, as Example 2.3 demonstrates.

Example 2.3. Consider the two cylindrical cells

$$V := \{x > y > 0, z > 0, y = xz\}$$

and

$$W := \{x > y > 0, z > 0, y = 2xz\}$$

in $\mathbb{R}^3$. Any cylindrical decomposition compatible with V and W is also compatible with the two intervals

$$I_1 = \{(0, 0, z) \mid 0 \leq z \leq 1/2\}, I_2 = \{(0, 0, z) \mid 1/2 \leq z \leq 1\}$$

and the point

$$v = (0, 0, 1/2).$$

Observe that $I = \{(0, 0, z) \mid 0 \leq z \leq 1\}$ is the only one-dimensional component of the side-wall of V and I_1 is the only one-dimensional component of the side-wall of W. In order to obtain the strong frontier condition, we need to partition V into (at least) three cylindrical cells V', V'', V''', one of which, say V'', will be one-dimensional. V'' should have endpoint v (i.e., $\mathrm{cl}\,(V'') \cap I = v$). The tangent at the origin of $c := \mathrm{proj}_{\mathbb{R}^2}(V'')$ is $1/2$. The pre-image of the projection map $W' := {\mathrm{proj}_{\mathbb{R}^2}}^{-1}|_{\mathrm{cl}(W)}(c)$ would satisfy the condition $\mathrm{cl}\,(W') \cap I = v'$, where $v' = (0, 0, 1/4)$. However, v' must be a zero-dimensional cell of the decomposition, which means we need to perform another refinement of I. An infinite sequence of refinements, introducing a new 0-dimensional cell $(0, 0, 1/2^k), k \in \mathbb{N}_{>0}$, will take place. Thus, a decomposition in in which the side-walls of V and W are single 1-dimensional cells cannot exist.

Proposition 2.4. *(Basu et al., 2015, Lemma 3.3)*

Let C be an $(i_1, \ldots, i_{k-1}, 0, i_{k+1}, \ldots, i_n)$-cell. Then

$$C' := \mathrm{proj}_{\mathrm{span}\{x_1,\ldots,x_{k-1},x_{k+1},\ldots,x_n\}}(C)$$

is a cylindrical $(i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_n)$-cell and C is the graph of a monotone map $f : C' \to \mathrm{span}\{x_k\}$.

Proposition 2.5. *(Basu et al., 2015, Lemma 3.4)*

Let $C \subset \mathbb{R}^n$ be a 2-dimensional cylindrical cell which is the graph of a quasi-affine map. Then the side-wall $W \subset \mathrm{fr}\,(C)$ has exactly two connected components, each of which is either a point or a closed curve interval.

Remark. It was proved by Schwartz and Sharir (1983) that a cylindrical decomposition, compatible with a semialgebraic set and satisfying the frontier condition, can be obtained following a linear rotation of coordinates. Davenport et al. (2020) prove that a cylindrical decomposition having constant sign on a set of input polynomials will contain topologically regular cells and satisfy the frontier condition, regardless of the method used to construct it, as long as the input polynomials do not blow up anywhere. Proposition 2.2 (Basu et al. (2013), Theorem 1) asserts that every monotone cell is topologically regular, so Basu et al. (2015), Theorem 3.20 describes the construction of a CAD, compatible with a family $V_1, \ldots, V_k$ of definable subsets of $\mathbb{R}^n$, each having dimension at most 2, in the potential presence of blow-up points and without the need to make a change of coordinates.

2.2.1 Constructing a CAD

Remark. A cylindrical algebraic decomposition (CAD) is a cylindrical decomposition in which every cell is a semialgebraic set. While the results of Basu et al.

(2015) are proved for the wider class of cylindrical decompositions compatible with definable sets, we will restrict ourselves to CADs, allowing us to take advantage of properties of polynomials.

Definition 2.28. Let $\mathbf{F} := (f_1, \ldots, f_s) \subset \mathbb{Z}[x_1, \ldots, x_n]$ be a set of polynomials with integer coefficients. A cylindrical algebraic decomposition $\mathcal{D}$ is called sign-invariant with respect to $\mathbf{F}$ (or simply $\mathbf{F}$-invariant) if each polynomial $f \in \mathbf{F}$ has a constant sign (either < 0, $= 0$ or > 0) on every cylindrical cell C of $\mathcal{D}$.

Remark. The fundamental theorem of algebra implies that for the coefficients of $\mathbf{F}$, we may choose any ring $K \subset \mathbb{R}$. $\mathbb{Z}$ has been chosen for convenience, but some authors (e.g., Schwartz and Sharir (1983)) choose $\mathbb{Q}$ instead. It is also common to take polynomials with coefficients over $\mathbb{R}$. However, the choice has been made to avoid this so as to work over the algebraic numbers $\mathcal{A}$, whose elements are easily represented in, and manipulated by, computers.

Let $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$ be a set of polynomials with integer coefficients. We will now describe the algorithm proposed by Collins (1975), closely following Coste (2000), for constructing an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$.

Collins' algorithm uses the idea of projection and lifting. More precisely, successive projections of the set of input polynomials to smaller and smaller dimensions are taken, until a set of univariate polynomials is obtained. The CAD of $\mathbb{R}^1$ is constructed by isolating the roots of these univariate polynomials, with section cells being the roots and sector cells being the open intervals in between them. The lifting phase then works recursively: given a collection of cells in $\mathbb{R}^k$ and a set of polynomials in $\mathbb{R}^{k+1}$, the roots of these polynomials within each cell C are isolated. The section cells in the cylinder $C \times \mathbb{R}$ are the roots of these polynomials, while the sector cells are the bands in between the roots. The projection operation was designed such that the cylindrical property holds. I.e., above C, each polynomial has a constant number of roots and the roots of no two polynomials "cross over" each other. The second property is called delineability.

Let us begin the description of the algorithm by considering a single polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$. We want to partition $\mathbb{R}^{n-1}$ into connected semialgebraic sets C' such that, for all $\mathbf{x} \in C'$, $f(\mathbf{x}, x_n)$ has constant degree and a constant number of roots.

Proposition 2.6. *(Coste, 2000, Proposition 2.16)*

Let $f \in \mathbb{Z}[x_1, \ldots, x_n]$ and, $C' \subset \mathbb{R}^{n-1}$ be a connected semialgebraic set and $k \leq d \in \mathbb{N}$ such that, for every point $\mathbf{x} \in C'$, the univariate polynomial $f(\mathbf{x}, x_n) \in \mathbb{A}[x_n]$ has degree d and exactly k distinct (complex) roots in C'. Then there are $\ell \leq k$ distinct semialgebraic functions

$$\psi_1, \ldots, \psi_\ell : C' \to \mathbb{R}$$

such that, for every $\mathbf{x} \in C'$, the set of real roots of $f(\mathbf{x}, x_n)$ is exactly

$$\{\psi_1(\mathbf{x}, \ldots, \psi_\ell(\mathbf{x})\}.$$

Moreover, the multiplicity of each of these roots is constant.

Proof. The argument relies on the "continuity of roots":

> Fix $\mathbf{c} \in C'$ and let $z_1, \ldots, z_k$ be the distinct roots of $f(\mathbf{c}, x_n)$ with multiplicities $m_1, \ldots, m_k$. Choose $\varepsilon > 0$ small enough that the open discs $D(z_i, \varepsilon), z_i \in \mathbb{C}$ (center z_i, radius ε) are disjoint. If $\mathbf{b} \in C'$ is sufficiently close to $\mathbf{c}$, then the polynomial $f(\mathbf{b}, x_n)$ has exactly m_i roots, counted with multiplicities, in $D(z_i, \varepsilon)$ for $1 \leq i \leq k$.

Since $f(\mathbf{b}, x_n)$ has k distinct complex roots, and $d = m_1 + \ldots + m_k$ complex roots counted with multiplicities, it follows that each $D(z_i, \varepsilon)$ contains exactly one root, denoted ζ_i, of $f(\mathbf{b}, x_n)$ with multiplicity m_i. If z_i is real, then ζ_i is real, otherwise the complex conjugate of ζ_i would be another root of $f(\mathbf{b}, x_n)$ in the disc $D(z_i, \varepsilon)$. If z_i is complex then ζ_i is also complex, since the conjugation of every point in $D(z_i, \varepsilon)$ lies outside $D(z_i, \varepsilon)$, forming another disc $D(\overline{z_i}, \varepsilon)$. It follows that, if $\mathbf{b} \in C'$ is close enough to $\mathbf{c}$, then $f(\mathbf{c}, x_n)$ has the same number of real roots as $f(\mathbf{b}, x_n)$. Since C' is connected, $f(\mathbf{x}, x_n)$ has the same number of real roots at every point $\mathbf{x} \in C'$, say ℓ. Define $\psi_i(\mathbf{x}) : C' \to \mathbb{R}$ to be the continuous (by making ε small enough) semialgebraic function sending $\mathbf{x} \in C'$ to the i-th (in ascending order) real root of $f(\mathbf{x}, x_n)$. It follows from the connectedness of C' that each ψ_i has constant multiplicity. Observe that the graph of each ψ_i can be expressed by a first-order Boolean formula, using existential quantifiers to express each of the ℓ roots of $f(\mathbf{x}, x_n)$ and an equality condition to pick out the i-th root. It follows that the graph of each functin ψ_i is a semialgebraic set. □

To extend this result to more than one polynomial, we also need to ensure that the graphs corresponding to each root never intersect.

Proposition 2.7. *(Coste, 2000, Proposition 2.18)*

Let $f, g \in \mathbb{Z}[x_1, \ldots, x_n]$, $C' \subset \mathbb{R}^{n-1}$ such that, for all $\mathbf{x} \in C'$, the degree and number of roots of $f(\mathbf{x}, x_n)$ and $g(\mathbf{x}, x_n)$ is constant and the degree of the GCD of $f(\mathbf{x}, x_n)$ and $g(\mathbf{x}, x_n)$ is constant. Let $\varphi, \psi : C' \to \mathbb{R}$ be continuous semialgebraic functions such that $f(\mathbf{x}, \varphi(\mathbf{x})) = 0$ and $g(\mathbf{x}, \psi(\mathbf{x})) = 0$ for all $\mathbf{x} \in C'$. If there exists $\mathbf{c} \in C'$ such that $\varphi(\mathbf{c}) = \psi(\mathbf{c})$, then $\varphi(\mathbf{x}) = \psi(\mathbf{x})$ for all $\mathbf{x} \in C'$.

Proof. We use the same method of proof as in the previous proposition. For an arbitrary element $\mathbf{c} \in C'$, let $z_1 = \varphi(\mathbf{c}) = \psi(\mathbf{c}), \ldots, z_k$ be the distinct roots in $\mathbb{C}$ of the product

$$f(\mathbf{c}, x_n)\ g(\mathbf{c}, x_n)$$

(recall that $fg = 0$ if either $f = 0$ or $g = 0$). Let m_i (resp p_i) be the multiplicity of z_i as a root of $f(\mathbf{x}, x_n)$ (resp. $g(\mathbf{x}, x_n)$) where multiplicity zero indicates that z_i is not a root. The degree of $\gcd(f(\mathbf{x}, x_n), g(\mathbf{x}, x_n))$ is $\min(m_1, p_1) + \cdots + \min(m_k, p_k)$ and each z_i has multiplicity $\min(m_i, p_i)$ as a root of this GCD. Choose $\varepsilon > 0$ small enough that the discs $D(z_i, \varepsilon)$ are disjoint. For each $\mathbf{b} \in C'$ close enough

to $\mathbf{c}$, each disc contains a root of multiplicity m_i of $f(\mathbf{b}, x_n)$ and a root of multiplicity p_i of $g(\mathbf{b}, x_n)$. Since the degree of the GCD of f and g (evaluated at any $\mathbf{x} \in C'$) is equal to $\min(m_1, p_1) + \cdots + \min(m_k, p_k)$, this GCD must have one root of multiplicity $\min(m_i, p_i)$ in each disc $D(z_i, \varepsilon)$ such that $\min(m_i, p_i) > 0$. In particular, it follows that $\varphi(\mathbf{b}) = \psi(\mathbf{b})$. Since C' is connected, this equality holds for all $\mathbf{x} \in C'$. □

Now we have algebraic conditions on the polynomials in the set of level-$(n-1)$ projection polynomials $\mathbf{F}' = \text{proj}(\mathbf{F})$. We now describe how these properties can be satisfied and, as such, how the set $\mathbf{F}'$ can be constructed. We will need the concept of the principal subresultant coefficient of two polynomials f and g.

Definition 2.29. Let $f, g \in \mathbb{Z}[x_1, \ldots, x_{n-1}][x_n]$ be polynomials, with degrees d and e respectively, such that

$$f(x_n) = a_d x_n^d + \cdots + a_1 x_n + a_0$$
$$g(x_n) = b_e x_n^e + \cdots + b_1 x_n + b_0.$$

Then the Sylvester matrix associated to f and g is the $(d+e) \times (d+e)$-matrix defined as follows.

$$S_{f,g}(f,g) = \begin{pmatrix} a_d & \ldots & a_0 & 0 & \ldots & \ldots & 0 \\ 0 & a_d & \ldots & a_0 & 0 & \ldots & 0 \\ \ldots & 0 & a_d & \ldots & a_0 & \ldots & 0 \\ b_e & \ldots & b_0 & 0 & \ldots & \ldots & 0 \\ 0 & b_e & \ldots & b_0 & 0 & \ldots & 0 \\ \ldots & 0 & b_e & \ldots & b_0 & \ldots & 0 \end{pmatrix}.$$

Definition 2.30. Let $f, g \in \mathbb{Z}[x_1, \ldots, x_{n-1}][x_n]$ be polynomials with degrees d and e respectively. The principal subresultant coefficient of order j of f and g, denoted $\text{psrc}_j(f,g)$, is the determinant of the $(m+n-2j) \times (m+n-2j)$-matrix obtained from the Sylvester matrix associated to f and g by deleting the first and last j rows and columns.

Let $f, g \in \mathbb{Z}[x_1, \ldots, x_{n-1}][x_n]$ and $X \subset \mathbb{Z}[x_1, \ldots, x_n]$.

- f has a constant number of complex roots over X if

  $$\text{psrc}_k(f, \partial f / \partial x_n),$$

  for $1 \leq k \leq \deg(f)$, is either zero or nonzero everywhere in X.
- f and g have constant GCD in X if

  $$\text{psrc}_k(f, g),$$

  for $1 \leq k \leq \min(\deg(f), \deg(g))$, is either zero or nonzero and $\deg(f) = \deg(g)$ everywhere in X.

- If, for the leading term of f (or g) vanishes at some points in $\mathbb{Z}[x_1, \ldots, x_{n-1}]$, we need to take the principal subresultant coefficients of the reductum of f (or g). (Coste, 2000, pp36)

From these properties of the principal subresultant coefficient, we can define a projection operator.

Definition 2.31. Consider $f \in \mathbb{Z}[x_1, \ldots, x_n]$ as a univariate polynomial in x_n. I.e.,

$$f(x_n) = a_d x_n^d + \cdots + a_1 x_n + a_0$$

where $a_d, \ldots, a_0 \in \mathbb{Z}[x_1, \ldots, x_{n-1}]$. Let $\operatorname{lc} f = a_d$ denote the leading coefficient of f and $\operatorname{red}(f) = a_{d-1} x_n^{d-1} + \cdots + a_1 x_n^1 + a_0$ be the reductum of f.

Let $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$. We define the projection operator, $\operatorname{proj}(\mathbf{F})$ as follows:

- Let $d := \deg(f_i)$. If $d > 1$, then $\operatorname{proj}(f_1, \ldots, f_i, \ldots, f_s)$ contains

  $$\operatorname{psrc}_k(f_i, \partial f_i / \partial x_n)$$

  for all $1 \leq k \leq d$.

- Let $d := \min(\deg(f_i), \deg(f_j))$. If $d > 0$, then $\operatorname{proj}(f_1, \ldots, f_i, \ldots, f_j, \ldots, f_s)$ contains

  $$\operatorname{psrc}_k(f_i, f_j)$$

  for all $1 \leq k \leq d$.

- If $\deg(f_i) > 0$ and $\operatorname{lc}(f_i)$ is non-constant, then $\operatorname{proj}(f_1, \ldots, f_i, \ldots, f_s)$ contains

  $$\operatorname{lc}(f_i) \text{ and } \operatorname{proj}(f_1, \ldots, \operatorname{red}(f_i), \ldots, f_n).$$

Theorem 2.2. *(Coste, 2000, Theorem 2.19) Let $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$ be a family of polynomials and let C' be a connected $\operatorname{proj}(\mathbf{F})$-invariant semialgebraic subset of $\mathbb{R}^{n-1}$ – an $(i_1, \ldots, i_{n-1})$-cell. Then there exist continuous definable functions*

$$\psi_1, \ldots, psi_\ell : C' \to \mathbb{R}$$

such that for all $\mathbf{x} \in C'$, the set $\{\psi_i(\mathbf{x}), \ldots, \psi_k(\mathbf{x})\}$ coincides with the real roots of polynomials in $\mathbf{F}$, defining the $(i_1, \ldots, i_{n-1}, 0)$-cells in the cylinder $C' \times \mathbb{R}$ and $\{-\infty < t < \psi_1(\mathbf{x}), \ldots, \psi_i(\mathbf{x}) < t < \psi_{i+1}(\mathbf{x}), \ldots, \psi_k(\mathbf{x}) < t < \infty\}$ are the $(i_1, \ldots, i_{n-1}, 1)$-cells in the cylinder $C' \times \mathbb{R}$.

This follows from the results previously proved. Applying Theorem 2.2, we are able to pass from a $\operatorname{proj}_{\mathbb{R}^{k-1}}(\mathbf{F})$-invariant CAD of $\mathbb{R}^{k-1}$ to a $\operatorname{proj}_{\mathbb{R}^k}(\mathbf{F})$-invariant CAD of $\mathbb{R}^k$. Thus, by iterating the projection operation until we obtain a set of polynomials in $\mathbb{Z}[x_1]$, isolating their roots, and then iteratively applying Theorem 2.2$, we are able to obtain an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$.

2.2.2 Bounds and variations

We have described how to construct a CAD of $\mathbb{R}^n$ which is sign-invariant with respect to a set of polynomials. We now present the bounds, along with some variations, from Basu et al. (2006).

Proposition 2.8. *(Basu et al., 2006, Algorithm 11.2)*

Let $\mathbf{F} := f_1, \ldots, f_s$ be a set of polynomials in $\mathbb{Z}[x_1, \ldots, x_n]$ with maximum degree d. There is an algorithm, taking $\mathbf{F}$ as input, which produces an $\mathbf{F}$-invariant CAD $\mathcal{D}$ of $\mathbb{R}^n$. The complexity of the algorithm is

$$(sd)^{O(1)^n}.$$

This is also an upper bound on the number of cells in $\mathcal{D}$, number of polynomials defining cells and their degrees.

Observe that these bounds are doubly exponential in the number of variables. It is not possible to improve upon this doubly exponential bound. Indeed, Davenport and Heintz (1988) give an example of a CAD in which a doubly exponential number of cells is obtained. In practice, we can obtain a CAD with fewer cells, e.g., by constructing a truth invariant or partial CAD.

Corollary 2.2. *Let $S_1, \ldots, S_k$ be a finite collection of semialgebraic subsets of $\mathbb{R}^n$ defined by quantifier-free Boolean formulas $F_1, \ldots, F_k$ respectively. Together, these formulas include s different polynomials in $\mathbb{Z}[x_1, \ldots, x_n]$ with maximum degree d. There is an algorithm, taking $\{F_1, \ldots, F_k\}$ as input, which produces a cylindrical decomposition $\mathcal{E}$ of $\mathbb{R}^n$ compatible with each set $S_i, 1 \leq i \leq k$. Complexity, number of cells, number of polynomials and degrees are the same as in Proposition 2.8 but, in practice, more efficient algorithms exist (see e.g., Collins and Hong (1991) and Bradford et al. (2014)).*

Proof. This construction follows immediately from Proposition 2.8, where truth values of $F_1, \ldots, F_k$ associated to each cell. The complexity upper bound (and bounds on number of cell, polynomials and degrees) is the same as in Proposition 2.8. □

It is also common to construct a CAD compatible with a single semialgebraic set.

Corollary 2.3. *Let $S \subset \mathbb{R}^n$ be a semialgebraic set defined by a quantifier-free Boolean formula F containing s different polynomials in $\mathbb{R}[x_1, \ldots, x_n]$, having maximum degree d. There is an algorithm, taking F as input, which produces a cylindrical decomposition $\mathcal{E}$ of $\mathbb{R}^n$ compatible with S. Complexity, number of cells, number of polynomials and degrees are the same as in Proposition 2.8.*

Proof. Immediate, by applying Corollary 2.3 to $\{F\}$. □

The construction described in Corollary 2.2 is somewhat naive. A far more efficient algorithm are presented in e.g., Collins and Hong (1991). An alternative approach is presented by Bradford et al. (2014). While very useful in practice, these constructions are unable to lower the upper bound from Proposition 2.8 due to some extreme cases, e.g., when the input formulas define a sign-invariant CAD (see Bradford et al. (2014), Section 6.2.1].

2.2.3 The projection operator

Since Collins published his CAD algorithm in 1975, many variations of the projection operator have been proposed. Most of these aim to minimise the number of polynomials appearing in $\text{proj}_{k-1}(\mathbf{F})$, so as to make the algorithm more efficient in practice. For example, Collins (1975) observed that, in dimension 2, if the set $\mathbf{F} \subset \mathbb{Z}[x_1, x_2]$ of input polynomials is squarefree and pairwise relatively prime, then it is sufficient to include only the leading coefficients, resultants and discriminants of pairs of polynomials in $\text{proj}(\mathbf{F})$. McCallum (1988) then proved that a similar construction is possible in dimension 3. McCallum (1998) later improved upon his previous work by extending this result to dimension $n > 3$, as long as the set of input polynomials is well oriented.

Definition 2.32. (McCallum, 1998, 6.1) Let $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$ be a set of polynomials and denote by $\text{prim}(\mathbf{F})$ the set of all primitive parts of $\mathbf{F}$ $\mathbf{F}$ is called well-oriented if, when $n > 1$,

- for each element $f \in \text{prim}(A)$, $f(\mathbf{x}, y) = 0$ for all $y \in \mathbb{R}$ on a finite number of points $\mathbf{x} \in \mathbb{R}^{n-1}$. (McCallum, 1998, condition WO1)
- $\text{proj}(\mathbf{F})$ is well-oriented. (McCallum, 1998, condition WO2)

Brown (2001) then observed that even more polynomials can be discarded, creating the Reduced McCallum Projection Operator. This is commonly used in practice, e.g., in `QEPCAD-B` (Brown, 2003).

Let $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$ be a well-oriented set of polynomials and $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD constructed using the McCallum projection operator (McCallum, 1998). Then, by McCallum (1988) Theorems 2.2.3 and 2.2.4, every cell C of $\mathcal{D}$ is an analytic submanifold of $\mathbb{R}^n$. Informally, if $\dim(C) = k$, thent C is a non-empty subset of $\mathbb{R}^n$ which "looks locally like $\mathbb{R}^k$" (see Definition 2.33 for the precise definition).

Definition 2.33. $S \subset \mathbb{R}^n$ such that $\dim(S) = k$ is an analytic submanifold if, for every point $\mathbf{x} \in S$, there is an analytic coordinate system about $\mathbf{x}$ with respect to which S is locally the intersection of $n - k$ coordinate hyperplanes.

Corollary 2.4. *Let $S \subset \mathbb{R}^n$ be a semialgebraic set defined by a quantifier-free Boolean formula constructed from the set of polynomials $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$, such that $\mathbf{F}$ is well-oriented. Then a CAD $\mathcal{D}$ of $\mathbb{R}^n$ compatible with S and such that*

every cell C of $\mathcal{D}$ is an analytic submanifold of $\mathbb{R}^n$ can be obtained by constructing an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$ using the McCallum projection operator.

More recently, McCallum et al. (2019) completed the proof that Lazard's projection operator is valid. McCallum also proved, without the condition on well-orientedness of input polynomials, that every cell of a sign-invariant CAD constructed using Lazard's projection operator is an analytic submanifold. Thus, we can generalise Corollary 2.4 further:

Let $S \subset \mathbb{R}^n$ be a semialgebraic set defined by a quantifier-free Boolean formula constructed from the set of polynomials $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$. Then a CAD $\mathcal{D}$ of $\mathbb{R}^n$ compatible with S and such that every cell C of $\mathcal{D}$ is an analytic submanifold of $\mathbb{R}^n$ can be obtained by constructing an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$ using the McCallum projection operator.

2.2.4 Projection polynomials vs semialgebraic functions

Recall that Proposition 2.6 allows us to pass from a projection polynomial $f \in \mathbb{Z}[x_1, \ldots, x_k]$ to the continuous semialgebraic functions $\varphi : C' \to \mathbb{R}$, where $C' \subset \mathbb{R}^{n-1}$ is a cylindrical cell, from Definition 2.20 which define some of the section cells in the cylinder $C' \times \mathbb{R}$. The construction described in Basu et al. (2015), Theorem 3.20 (and related lemmas) works with these continuous semialgebraic functions. However, we are given only the projection polynomials f of which φ defines one of the roots.

Let C be a section cell in $\mathbb{R}^n$: a semialgebraic set which is the graph of a continuous semialgebraic function $\varphi : C' \to \mathbb{R}$, where $C' \subset \mathbb{R}^{n-1}$ is a cylindrical cell. A simple example in $\mathbb{R}^2$ demonstrates that, although C is a semialgebraic set, φ may not be a polynomial. Indeed, let

$$C := \{(x, y) \in \mathbb{R}^2 \mid x > 0, y > 0, y = x^2\}$$

be a cylindrical cell in $\mathbb{R}^2$ such that $C' := \mathrm{proj}_{\mathbb{R}^1}(C) = \{x \in \mathbb{R} \mid x > 0\}$. We can write

$$\varphi(x) = +\sqrt{x}$$

such that C is the graph of $\varphi|_{C'}$. However, it's clear that φ is not a polynomial.

In computations, we work exclusively with section cells. When dealing with sector cells, we will consider the graphs of functions defining their top and bottom. Since we will be using `SACLIB` and sometimes `Singular`, which are polynomial libraries, we will not be able to use the functions φ in computations. We will instead use a different representation for a section cell C in $\mathbb{R}^n$. Let C be the root of a polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$ and $C' := \mathrm{proj}_{\mathbb{R}^{n-1}}(C)$ satisfy the conditions in Proposition 2.6. Then

$$C \subset Z := \{(\mathbf{x}, x_n) \in \mathbb{R}^n \mid \mathbf{x} \in C', f(\mathbf{x}, x_n) = 0\} \subset C' \times \mathbb{R}.$$

In particular, Z consists of the roots of f in the cylinder $C' \times \mathbb{R}$, which includes the section cell C. It is clear from the Tarski–Seidenberg theorem that it is

possible to represent C as a first-order Boolean formula. In practice, since C is the root of a polynomial, we may define it using Thom's Lemma (Coste and Roy, 1988). I.e., a root of a polynomial may be defined by sign conditions on its derivatives. Thus, let

$$C = \{(\mathbf{x}, x_n) \in \mathbb{R}^n \mid \mathbf{x} \in C', f(\mathbf{x}, x_n) = 0, g_1(\mathbf{x}, x_n) > 0, \ldots, g_k(\mathbf{x}, x_n) > 0\}$$

where $g_1, \ldots, g_k$ are derivatives of the polynomial f.

2.2.5 CAD cells and their defining formulas

Although it is possible to construct a "full" (sign-invariant) CAD, `QEPCAD-B` provides an implementation of the "truth-invariant" CAD described in Corollary 2.2. Indeed, QEPCAD-B, being designed primarily to perform quantifier elimination, frequently constructs a "partial" CAD of $\mathbb{R}^n$ (Brown, 2003). We will force the construction of the sign-invariant CAD and use McCallum's projection operator so as to obtain smooth cells (see Corollary 2.4).

Once constructed, one may want to examine individual cells of the CAD. In order to do this, we will need a convenient representation for each cell. Depending on the purpose for which the CAD has been constructed, a particular representation may be chosen. For example, if we only need a witness, i.e., some point in the cell, only the sample points are needed. Indeed, this is a common requirement and Maple's `RegularChains::CylindricalAlgebraicDecompose` function provides output in this representation (Chen and Moreno Maza, 2014). On the other hand, one might want to view each cell as a semialgebraic set. In this case, a defining formula will be necessary. Such a representation using only the input and projection polynomials may be impossible and, as such, an "extended language" has been introduced in both Maple and `QEPCAD` (Chen and Moreno Maza, 2014, Brown (2003)). The extended language includes the rules of first-order Boolean formulas (see Definition 2.4) along with a means of representing "the i-th root of polynomial f". We find this representation a little inconvenient and would prefer each cell to be representable by an ordinary first-order Boolean formula.

To see how the signs of projection factors may be insufficient to represent a cylindrical cell, consider the following simple example in $\mathbb{R}^2$. Let $f = x - y^2$ and construct an $\{f\}$-invariant CAD. The projection polynomials

$$P = \{x, x - y^2\}.$$

The two distinct cells C_1 and C_2 have signs $x > 0, x - y^2 = 0$. By Thom's Lemma, it is possible to represent these cells by sign conditions on derivatives of f. In this example, it is clear that another polynomial, y, is needed to distinguish between these cells. Brown (1999) proposed an algorithm for constructing a solution formula if the projection polynomials are not sufficient. The basic idea is that "conflicting pairs", distinct cells having the same signature (signs of projection factors), are identified, then additional derivatives of projection factors are added so that the conflicting pairs can be distinguished.

The algorithm proposed by Brown (1999) is implemented as the "solution formula construction phase" in QEPCAD. A small modification has been made such that this algorithm may be used to construct formulas for individual cells as opposed to the set of points satisfying the input formula. More precisely, the definition of a conflicting pair was slightly modified.

Definition 2.34. Let C_1 and C_2 be distinct cells in a CAD $\mathcal{D}$ of $\mathbb{R}^n$. Each cell C is assigned a truth value $T_C \in \{\texttt{True}, \texttt{False}, \texttt{undet}\}$ and a list $S_C = (S_1, \ldots, S_k)$ where $S_i = (s_{i,1}, \ldots, s_{i,s_i}, 1 \leq i \leq k$ is the list of signs of level-i (factorised) projection polynomials $f_1, \ldots, f_{i_s} \in \mathbb{Z}[x_1, \ldots, x_i]$, with $s_{i,j} \in \{-1, 0, 1\}$. (C_1, C_2) is called a conflicting pair in $\mathcal{D}$ if $T_{C_1}, T_{C_2} \in \{\texttt{True}, \texttt{False}\}$ and $T_{C_1} = T_{C_2}$ and $S_{C_1} = S_{C_2}$ (equality determined component-wise).

For our purposes, the condition on truth values being equal has been dropped so that every cell in the CAD may be distinguished by the signs of its projection polynomials only. In all other aspects, Brown's solution formula construction algorithm may be applied unchanged. In order to represent a cell by a quantifier-free Booleean formula, it may be necessary to split a cell into several disjoint semialgebraic sets. Thus, each cell will be represented in disjunctive normal form, where each literal is a sign condition on a projection polynomial or one of its derivatives.

Chapter 3

Smooth Stratification

Before turning our attention to cylindrical decompositions, we will present another useful algorithm for decomposing a semialgebraic set. A smooth stratification is a finite partition of a set into smooth manifolds. This algorithm will form part of the later work in constructing a CAD with monotone cells, which requires the input sets to first be partitioned into smooth smooth manifolds (see Section 4. However, the algorithm and implementation described in this section are useful in their own right.

In 1957, Whitney (1992) proved that every algebraic variety can be partitioned into a finite set of smooth manifolds, each of which is a semialgebraic set. Lojasiewicz extended this result by proving that every real semianalytic set admits a smooth stratification, such that every stratum is a semianalytic set (Lojasiewicz, 1964). However, the proof was non-constructive and did not provide any information on the class of functions defining the strata. As part of his Complement Theorem, Gabrielov (1996) proved that the strata of a semianalytic set can be defined by functions belonging to the smallest extensions of a family of functions defining the set which is closed under addition, multiplication and taking partial derivatives. One such class of functions is the Pfaffian functions. Gabrielov and Vorobjov (1995) present an algorithm for computing a smooth stratification of a semi-pfaffian set, along with an estimate of its complexity and bounds on the formats of the strata it produces. This algorithm (also summarised in Gabrielov and Vorobjov (2004), Section 6) will be presented.

Definition 3.1. (Gabrielov and Vorobjov, 2004, Definition 6.2) A weak stratification of a semi-Pfaffian set $X \subset \mathbb{R}^n$ is a partition of X into a finite number of non-singular manifolds $X_k, 0 \leq k \leq n$ called strata. Strata need not be connected (or even have a finite number of connected components) and may be empty.

The stratification is called basic if all strata are basic semi-Pfaffian sets which

are effectively nonsingular. I.e., the system of equations and inequalities defining each stratum X_k of codimension k includes a set of k Pfaffian functions

$$h_{i_1}, \ldots, h_{i_k}$$

such that the restriction $h_{i_j}|_{X_i} = 0$ and $dh_{i_j} \neq 0$ at every point $\mathbf{x} \in X_k$, for each $1 \leq j \leq k$.

Note that this definition does not guarantee any other desirable properties of the manifolds X_k. E.g., the closure of a stratum may not be a union of strata.

3.1 Description of the algorithm

Since every semialgebraic set is a semi-Pfaffian set, this algorithm can be applied to the semialgebraic sets with almost no modifications. Since each stratum is defined by sign conditions on partial derivatives of input functions, it is clear that if the input set is semialgebraic, then each stratum will be semialgebraic, too. Furthermore, as emptiness of semialgebraic sets can be decided, e.g., using the cylindrical algebraic decomposition algorithm, an Oracle will not be needed in this variation of the algorithm. The algorithm relies on computing partial derivatives of the functions defining the input set. In the semi-Pfaffian case, the number of partial derivatives which must be considered is bounded by the format of the input set. As every polynomial has a finite number of partial derivatives, the bound for the semialgebraic case will simply be the degree of the polynomial.

We will need the following notation for partial derivatives and the partial differential operator defined by Gabrielov and Vorobjov (1995).

Definition 3.2. (Gabrielov and Vorobjov, 1995, Definition 2) Let $f \in \mathbb{Z}[x_1, \ldots, x_n]$ be a polynomial and let $(m_1, \ldots, m_n) \subset \mathbb{Z}^n_{\geq 0}$ be a multi-index associated to one of its partial derivatives. I.e., we write

$$\partial f^{(m_1,\ldots,m_n)} = \frac{\partial^{m_1} f}{\partial x_1} \cdots \frac{\partial^{m_n} f}{\partial x_n}.$$

Definition 3.3. (Gabrielov and Vorobjov, 1995, Definition 2) We define the partial differential operator $\partial_{\mathbf{h},\mathbf{i},j} f$ (where the argument f is a polynomial) as the determinant

$$\det \begin{pmatrix} \frac{\partial h_1}{\partial x_{i_1}} & \cdots & \frac{\partial h_1}{\partial x_{i_k}} & \frac{\partial h_1}{\partial x_j} \\ & \vdots & & \\ \frac{\partial h_k}{\partial x_{i_1}} & \cdots & \frac{\partial h_k}{\partial x_{i_k}} & \frac{\partial h_k}{\partial x_j} \\ \frac{\partial f}{\partial x_{i_1}} & \cdots & \frac{\partial f}{\partial x_{i_k}} & \frac{\partial f}{\partial x_j} \end{pmatrix}.$$

We write $\partial^m_{\mathbf{h},\mathbf{i},j}$ to mean the m-th iteration of $\partial_{\mathbf{h},\mathbf{i},j}$.

We can now present the algorithm from Gabrielov and Vorobjov (1995), Theorem 2.

Theorem 3.1. *Let*

$$X := \{\mathbf{x} \in \mathbb{R}^n \mid f_1 = 0, \ldots, f_k = 0, g_1 > 0, \ldots, g_\ell > 0\},$$

be a semialgebraic set defined by $s = k + \ell$ different polynomials of maximum degree d. Then there is an algorithm, without oracle, which partitions X into a family

$$\mathcal{X} = (X_0, \ldots, X_n)$$

such that, if X is nonsingular, $X_0 = X$ and all other sets are empty. Otherwise, $X_0 = \emptyset$ and each $X_k, 1 \leq k \leq n$ is a possibly empty, effectively nonsingular stratum of codimension k. This algorithm has complexity

$$3^s \left(s(d+1)\right)^{O(n)^2}.$$

The number of strata does not exceed $s^n(d+1)^2$, and each stratum is defined by at most $s(d+1)^2$ polynomial equations and inequalities of maximum degree $(d+1)^2$.

Let

$$X = \{\mathbf{x} \in \mathbb{R}^n \mid f_1(\mathbf{x}) = 0, \ldots, f_s(\mathbf{x}) = 0, g_1(\mathbf{x}) > 0, \ldots, g_t(\mathbf{x}) > 0\}.$$

Algorithm 1. *Smooth Stratification*

$$\mathcal{X} := \text{Stratify}(\text{k}, \text{F}, \mathbf{h}, \mathbf{i}, \text{G})$$

Input:

- $0 \leq k \leq n$,
- $F = (f_1, \ldots, f_s) \in \mathbb{Z}[x_1, \ldots, z_n]$ *is a list of polynomials,*
- $\mathbf{h} = (h_k, \ldots, h_1, h_0) \in \mathbb{Z}[x_1, \ldots, x_n]$ *such that* $h_0 = 0$,
- $\mathbf{i} = (i_k, \ldots, i_1, i_0) \in \mathbb{Z}_{\geq 0}$ *such that* $i_0 = 0$,
- $G = \{g_1, \ldots, g_t\} \subset \mathbb{Z}[x_1, \ldots, x_n]$.

Output:

$$\mathcal{X} = (X_{k+1}, \ldots, X_n)$$

$X_i, k+1 \leq i \leq n$ is a possibly empty, effectively nonsingular stratum of codimension i. Note that $\mathcal{X}_{n+1} = ()$.

Proceed by induction on i_k.

- *Base case: $i_k = n$.*
  - *Return $\mathcal{X}_{x+1} = (\emptyset, \ldots, \emptyset)$.*

- *Otherwise*
  - *Initialise*
    - $X' := X = \{f_1 = 0, \dots, f_s = 0, g_1 > 0, \dots, g_t > 0\}$,
    - $\mathcal{F} := \{((0, \dots, 0, 1), f_1), \dots, ((0, \dots, 0, s), f_s)\}$ *(the set of input polynomials equipped with initial indices),*
    - $X_{k+1} = \emptyset$ *be the initial set of strata of codimension* $k+1$.
  - *For indices* $\mathbf{i}, \mathbf{j} \in \mathbb{Z}^n_{\geq 0}$,
    - *let* $\mathbf{i} \prec \mathbf{j}$ *mean that* $\mathbf{i}$ *is lexicographically less than* $\mathbf{j}$
    - *and* $\mathbf{i} \preceq \mathbf{j}$ *denote* $\mathbf{i} \prec \mathbf{j}$ *or* $\mathbf{i} = \mathbf{j}$.
  - *We will iterate over each index* $\mathbf{m} = (i_n, \dots, i_{k+1}, j) \in \mathbb{Z}^{n-k+1}_{\geq 0}$ *in ascending lexicographical order, beginning with* $(0, \dots, 0, 1)$.
  - *Partial derivatives* $\partial^{(0,\dots,0,i_{k+1},\dots,i_n)}$ *of functions* f *in* $\mathcal{F}$ *will be computed.*
  - *For each* $f \in \mathcal{F}$, *maximal index*

$$M := (\deg_n(f), \dots, \deg_{k+1}(f)).$$

  - *Each index* $(i_n, \dots, i_{k+1}, j)$ *has the property that*

$$(0, \dots, 1) \preceq (i_n, \dots, i_{k+1}) \preceq M_j.$$

    - *Let* $\mathbf{m} = (0, \dots, 0, i_\ell, \dots, i_{k+1}, j)$ *be one of these indices and define*

$$\mathbf{j} = (0, \dots, 0, i_\ell - 1, i_{\ell-1}, \dots, i_{k+1}, j).$$

    - *Let*

$$s_{k+1} := \partial^{i_\ell}_{(h_1,\dots,h_k),(i_1,\dots,i_k),\ell} \cdots \partial^{i_i+1}_{(h_1,\dots,h_k),(i_1,\dots,i_k),i_k+1} f_j$$

      where f_j *is the polynomial in* $\mathcal{F}$ *with index* $(0, \dots, 0, j)$, *i.e., the* j*-th input polynomial.*
    - *In practice, let* h_{k+1} *be the polynomial in* $\mathcal{F}$ *with index* $\mathbf{i}$ *and compute*

$$s_{k+1} := \partial_{(h_1,\dots,h_k),(i_1,\dots,i_k),i_\ell} h_{k+1}.$$

    - *Define the sets*
      - $Y_{k+1} := \{\mathbf{x} \in X' \mid s_{k+1}(\mathbf{x}) \neq 0\}$ *and*
      - $U_{k+1} := \{\mathbf{x} \in \mathbb{R}^n \mid h_{k+1}(\mathbf{x}) = 0, s_{k+1}(\mathbf{x}) \neq 0, g_1(\mathbf{x}) > 0, \dots, g_t(\mathbf{x}) > 0\}$.
      - *Note*

- *Clearly* $Y_{k+1} \subset X'$ *and* $Y_{k+1} \subset U_{k+1}$
- $h_{k+1}(\mathbf{x}) = 0$ *for all* $\mathbf{x} \in Y_{k+1}$.
- U_{k+1} *is a nonsingular subset of* $\mathbb{R}^n$ *having codimension* $k+1$.

- *If* Y_{k+1} *is an open subset of* U_{k+1}*, then* Y_{k+1} *is smooth and has codimension* $k+1$.
  - *Let* $X_{k+1} := X_{k+1} \cup Y$,
- *Otherwise, proceed by induction.*
  - *Compute*

$$(X^{\mathbf{m}}_{k+2}, \ldots, X^{\mathbf{m}}_{n}) := \text{Stratify}(k+1, F, (s, : \mathbf{h}), (i_\ell : \mathbf{i}), G)$$

    where $x : L$ *denotes the* `cons` *operator. I.e., add* x *to the beginning of list* L.
- *Let* Empty(X, s) *be a subroutine returning* true *if* $s \in \mathbb{Z}[x_1, \ldots, x_n]$ *is identically zero on the set* $X \subset \mathbb{R}^n$.
- *If* Empty(X, s)*, we are finished.*
  - *Return*

$$\mathcal{X}_{k+1} = (X_{k+1}, X_{k+1}, \ldots, X_n).$$

    where $X_i, k+2 \leq i \leq n$ *is the union of all strata* $X^{\mathbf{m}}_i$ *computed in the recursive calls.*
- *Otherwise,*
  - *Let*
    - $X' := \{\mathbf{x} \in \mathbb{R}^n \mid s(\mathbf{x}) = 0\}$,
    - $\mathcal{F} := \mathcal{F} \cup \{((0, \ldots, 0, i_\ell, \ldots, i_{k+1}, j), s)\}$,
  - $F := F : s$ *(i.e., append* s *to the end of list* F*).*
- *Then proceed to the index immediately after* $\mathbf{m}$ *with respect to* $\prec$.
- *The algorithm terminates when maximal index* M_j *for every polynomial* $f_j \in F$ *has been considered.*

To call this algorithm on a semialgebraic set $X \subset \mathbb{R}^n$ defined by equations $F \subset \mathbb{Z}[x_1, \ldots, x_n]$ and $G \subset \mathbb{Z}[x_1, \ldots, z_n]$, compute

$$\mathcal{X}' = \text{Stratify}(0, F, (0), (0), G).$$

to obtain a set of strata $\mathcal{X}' = (X_1, \ldots, X_n)$. If all $X_i = \emptyset, 1 \leq i \leq k$, then we can conclude that every partial derivative vanishes identically on X. Hence, X is a smooth subset of $\mathbb{R}^n$ and we let $X_0 = X$. Otherwise, let $X_0 = \emptyset$

so that $\mathcal{X} = (\emptyset, X_1, \ldots, x_n)$ where each $X_i, 1 \leq i \leq n$ is a smooth subset of X of codimension i, such that each $X_i, X_j, i \neq j$ is pairwise disjoint and $X_1 \cup \ldots \cup X_n = X$. Note, in the case that $X_0 = X$, the smooth stratification algorithm provides no information about the dimension of X.

3.1.1 Subroutines and Algorithm Details

The algorithm employs two subroutines, which were presented in the description as "black-box functions". The first of these is a subroutine which determines whether Y_k is an open subset of U_k. This is the case if, for every point $\mathbf{x}$ in Y_k all points of U_k which are sufficiently close to $\mathbf{x}$ are also in Y_k. This can be expressed as a first-order Boolean formula with two quantifier alternations and its truth value can be determined by performing quantifier elimination. I.e., Y_k is an open subset of U_k if

$$\forall \mathbf{x} \in Y_k \, \exists \varepsilon > 0 \in \mathbb{R} \, \forall \mathbf{y} \in U_k \mid \|\mathbf{x} - \mathbf{y}\| < \varepsilon \rightarrow \mathbf{y} \in Y_k$$

An alternative method, as described in Gabrielov and Vorobjov (1995), Theorem 2, is to check whether every derivative computed at step $k+1$ vanishes on the set

$$U_k \cap \{x_1 = \texttt{const} \mid \texttt{i} < \texttt{i}_1\}$$

where $i_k \in \{1, \ldots, n\}$ is a parameter passed from step k to $k+1$ in the algorithm. This, again, reduces to the emptiness check.

The second of these is Empty(S), where $S \subset \mathbb{R}^n$ is a semialgebraic set represented by a quantifier-free Boolean formula which is a conjunction of polynomial equations, inequations and inequalities. Note that an inequation $\{f(\mathbf{x}) \neq 0\}$ can be rewritten as $\{f(\mathbf{x}) < 0 \vee f(\mathbf{x}) > 0\}$, so it is possible to represent S by two systems of polynomial equations and inequalities, or a Boolean formula containing the conjunction above. This permits various methods for checking emptiness. We may write Empty(S) as a quantifier elimination problem

$$\begin{aligned} \exists \mathbf{x} \in \mathbb{R}^n \mid & f_1(\mathbf{x}) = \ldots = f_s(\mathbf{x}) = 0, \\ & s_1(\mathbf{x}) \neq 0, \ldots, s_k(\mathbf{x}) \neq 0, \\ & g_1(\mathbf{x}) > 0, \ldots, g_t(\mathbf{x}) > 0. \end{aligned}$$

This problem can be decided using the singly-exponential quantifier elimination algorithm from Proposition 2.1 (Basu et al. (2006), Algorithm 14.21), which, with one quantifier alternation, has complexity $(sd)^{O(n)}$ where s is the number of different polynomials defining the set and d their maximum degree – singly exponential in the number of variables. Alternatively, the algorithm presented by Basu et al. (1998), which produces a witness point in every cell in a decomposition of the input set, may be used to decide emptiness. This algorithm also has complexity $(sd)^{O(n)}$. In practice, quantifier elimination can be performed using CAD. This, of course, has complexity doubly exponential in the number of variables: $(sd)^{O(1)^n}$ (see Proposition 2.8).

The algorithm may be implemented as above, but with some small tweaks to simplify the code and make the algorithm work slightly more efficiently. First, observe that not all derivatives are needed to define the candidate stratum Y_k. One obvious situation is when a derivative s_k turns out to be a nonzero constant $c \in \mathbb{A}$. In the formula for Y_k, the inequation $c \neq 0$ will appear, which is obviously true. Since $X' \supset Y_k$ was assumed to be non-empty, we can immediately conclude that Y_k is non-empty and equal to X'. No further derivatives at step k need to be considered, since $X' \setminus Y_1 = \emptyset$. A similar situation may arise, even with nonconstant functions s_k. For example, for the input set

$$X := \{\mathbf{x} \in \mathbb{R}^2 \mid x^2y^2 = 0\},$$

the derivative for index $(0, 1)$ is $\partial(x^2y^2)/\partial x_1 = 2xy^2$, which is equal to zero at every point of X. By discarding these redundant functions, we minimise the number of polynomials defining sets in the emptiness check. Note that we have to keep track of every derivative computed, even the redundant ones, so that we can find the function h_k such that $s_k = \partial_{\mathbf{h},\mathbf{i},j} h_1$.

Now consider the operation of finding the derivative with index $(0, \ldots, 0, i_\ell - 1, i_{\ell-1}, \ldots, i_{k+1}, j)$. The following observation allows us to quickly find this polynomial, avoiding a search in $\mathcal{G}$ or unnecesary recomputation of derivatives. Suppose we are considering index

$$\mathbf{m} = (0, \ldots, 0, i_\ell, i_{\ell-1}, \ldots, i_{k+1}, j)$$

and want to find the derivative with index

$$\mathbf{j} = (0, \ldots, 0, i_\ell - 1, i_{\ell-1}, \ldots, i_{k+1}, j).$$

The set $\mathcal{G}$ includes polynomials computed in previous rounds of induction and those with index $\mathbf{i} \prec \mathbf{m}$, since we proceed in ascending lexicographical order. Hence $\mathcal{G}$ contains the derivative with index $\mathbf{j}$. There is a convenient way to find this derivative. Let us first illustrate the process with an example. Let $f_j \in \mathbb{Z}[x_1, x_2, x_3]$ with $M_j = (1, 2, 2)$ and consider the lexicographically ordered list of indices

$$\begin{array}{lccccccccc} L_1 := & (0,0,0), & (0,0,1), & (0,0,2), & (0,1,0), & (0,1,1), & (0,1,2), & (1,0,0), & (1,0,1) & \ldots \\ L_2 := & & (0,0,0), & (0,0,1), & (0,0,0), & (0,0,1), & (0,0,2), & (0,0,0), & (0,0,1) & \ldots. \end{array}$$

Observe that for indices of the kind $(0, \ldots, 0, 1, 0, \ldots, 0)$ in L_1, the corresponding element of L_2 is $(0, \ldots, 0)$. For an arbitrary index $(0, \ldots, 0, i_\ell, \ldots, i_1)$ in L_1 which appears k elements after $(0, \ldots, 0, i_\ell = 1, 0, \ldots, 0)$, the corresponding element in L_2 appears k elements after $(0, \ldots, 0)$. From this, we can introduce an "index chasing" method as follows. Let $\mathcal{G}_j$ be the (ordered) list of derivatives of polynomial f_j. For each $\mathbf{m}$, we keep a pointer, $\mathcal{G}'_j$, to the element in $\mathcal{G}_j$ with index $\mathbf{j}$.

- Initialise $\mathcal{G}_j = ((0, \ldots, 0), f_j)$. For $\mathbf{m} = (0, \ldots, 0, 1)$, set $\mathcal{G}'_j = \mathcal{G}_j$.
- Now suppose that $\mathcal{G}_j$ contains all polynomials with index $\prec \mathbf{m}$.

- If $\mathbf{m} = (0, \ldots, 0, 1, 0, \ldots, 0)$, then $\mathbf{j} = (0, \ldots, 0)$ and we set $\mathcal{G}'_j$ to the head of $\mathcal{G}_j$.
- Otherwise, let $\mathcal{G}'_j$ be the tail of $\mathcal{G}'_j$.

The "index chasing" method works efficiently in SACLIB, which uses linked lists. However, we do need to append new derivatives to the end of $\mathcal{G}_j$, which takes $O(n)$ steps. To avoid this, we keep a pointer to the last-but-one element in the list, i.e., suppose $G_j = (a_1, (\ldots, (a_{r-1}, (a_r, (\text{NIL}))) \ldots)$ then $G_{j,\text{append}} = (a_{r-1}, (a_r, (\text{NIL})))$. Then, if we wish to append b to the end of G_j, we simply need to set the tail of $G_{j,\text{append}}$ to $(a_r, (b, (NIL)))$.

We can also optimise the construction of the matrix associated to the partial differential operator $\partial_{\mathbf{h},\mathbf{i},j} f$. In this matrix,

$$\begin{pmatrix} \frac{\partial h_1}{\partial x_{i_1}} & \cdots & \frac{\partial h_1}{\partial x_{i_k}} & \frac{\partial h_1}{\partial x_j} \\ & \vdots & & \\ \frac{\partial h_k}{\partial x_{i_1}} & \cdots & \frac{\partial h_k}{\partial x_{i_k}} & \frac{\partial h_k}{\partial x_j} \\ \frac{\partial f}{\partial x_{i_1}} & \cdots & \frac{\partial f}{\partial x_{i_k}} & \frac{\partial f}{\partial x_j} \end{pmatrix},$$

only the last row depends on f and only the last column depends on j. Thus, we can save the $(k \times k)$-submatrix consisting of the first k rows and columns and only append the new row and column. This matrix can be passed as an argument to Stratify$(k, \ldots)$, with the matrix for $k = 0$ being empty (i.e., $\mathbf{h} = \mathbf{i} = ()$ in the partial differential operator).

3.2 Worked example

We illustrate the smooth stratification algorithm with a worked example in $\mathbb{R}^3$. Consider

$$\{x_3 = 0, x_1^2 - x_2^2 = 0\} \subset \mathbb{R}^3.$$

This set is contained in the plane $\{x_3 = 0\}$ and consists of the two intersecting straight lines $\{x_1 - x_2 = 0\}$ and $\{x_1 + x_2 = 0\}$ in span$\{x_1, x_2\}$. Note that the algorithm imposes an order on polynomials. Here we take $f_1 := z$ and $f_2 = x^2 - y^2$, wiht f_1 and f_2 being assigned indices $(0, 0, 0, 1)$ and $(0, 0, 0, 2)$ respectively. The algorithm proceeds as follows

$$k = 0, X' := \left\{\mathbf{x} \in \mathbb{R}^3 \mid x_3 = 0, x_1^2 - x_2^2 = 0\right\} \tag{3.1}$$

$$(0,0,1,1)\ \ h_1 = f_1 = x_3 \tag{3.2}$$

$$s_1 = \partial f_1/\partial x_1 = 0 \tag{3.3}$$

$$Y_1 := \{\mathbf{x} \in X' \mid 0 \neq 0\} = \emptyset \tag{3.4}$$

$$(0,0,1,2)\ \ h_1 = f_2 = x_1^2 - x_2^2 \tag{3.5}$$

$$s_1 = \partial f_2/\partial x_1 = 2x_1 \tag{3.6}$$

$$Y_1 := \{\mathbf{x} \in X' \mid 2x_1 \neq 0\} \tag{3.7}$$

$$U_1 := \left\{\mathbf{x} \in \mathbb{R}^3 \mid x_1^2 - x_2^2 = 0, 2x_1 \neq 0\right\} \tag{3.8}$$

$$\text{Not open, proceed by induction} \tag{3.9}$$

$$k = 1, h_1 = f_2, i_1 = 1, X'' := Y_1, F = \left\{f_1 = x_3, f_2 = x_1^2 - x_2^2\right\} \tag{3.10}$$

$$(0,1,1)\ \ h_2 = f_1 \tag{3.11}$$

$$s_2 = 0 \text{ since } f_1 \text{ doesn't depend on } x_1, x_2 \tag{3.12}$$

$$(0,1,2)\ \ h_2 = f_2 \tag{3.13}$$

$$s_2 = 0 \text{ snce } h_1 = h_2 \tag{3.14}$$

$$(1,0,1)\ \ h_2 = f_1 \tag{3.15}$$

$$s_2 = \det \begin{pmatrix} \partial h_1/\partial x_1 & \partial h_1/\partial x_3 \\ \partial h_2/\partial x_1 & \partial h_2/\partial x_3 \end{pmatrix} = \det \begin{pmatrix} 2x_1 & 0 \\ 0 & 1 \end{pmatrix} = 2x_1 \tag{3.16}$$

$$Y_2 := \{\mathbf{x} \in X'' \mid 2x_1 \neq 0\} \tag{3.17}$$

$$U_2 := \left\{\mathbf{x} \in \mathbb{R}^3 \mid x_1^2 - x_2^2 = 0, x_3 = 0, 2x_1 \neq 0\right\} \tag{3.18}$$

$$\text{Note } Y_1 = Y_2 = U_2 \tag{3.19}$$

$$Y_2 \text{ is smooth and has codimension 2} \tag{3.20}$$

$$\text{since } X'' := \{\mathbf{x} \in X'' \mid 2x_1 = 0\} = \emptyset, \text{ we return to } k = 0 \tag{3.21}$$

$$k = 0, X' := \left\{\mathbf{x} \in \mathbb{R}^3 \mid x_3 = 0, x_1^2 - x_2^2 = 0, 2x_1 = 0\right\} \tag{3.22}$$

$$\text{Obsrve } X' = (0,0,0) \tag{3.23}$$

$$(0,0,2,1)\ \ h_1 = 0, s_1 = 0 \tag{3.24}$$

$$(0,0,2,2)\ \ h_1 = 2x_1 \tag{3.25}$$

$$s_1 = 2 \tag{3.26}$$

$$Y_1 := \{\mathbf{x} \in X' \mid 2 \neq 0\} = X' \tag{3.27}$$

$$U_1 := \left\{\mathbf{x} \in \mathbb{R}^3 \mid 2x_1 = 0, 2 \neq 0\right\} \tag{3.28}$$

$$\text{Not open, proceed by induction} \tag{3.29}$$

$$\tag{3.30}$$

$$k = 1, h_1 = 2x, i_1 = 1, X'' := Y_1, F = \left\{ f_1 = x_3, f_2 = x_1^2 - x_2^2, f_3 = 2x_1 \right\} \tag{3.31}$$

$$(0,1,1)\ \ h_2 = f_1 = x_3 \tag{3.32}$$

$$s_2 = 0 \tag{3.33}$$

$$(0,1,2)\ \ h_2 = f_2 = x_1^2 - x_2^2 \tag{3.34}$$

$$s_2 = \det \begin{pmatrix} \partial h_1/\partial x_1 & \partial h_1/\partial x_2 \\ \partial h_2/\partial x_1 & \partial h_2/\partial x_2 \end{pmatrix} = \det \begin{pmatrix} 2 & 0 \\ 2x_1 & 2x_2 \end{pmatrix} = 4x_2 \tag{3.35}$$

$$Y_2 := \{ \mathbf{x} \in X'' \mid 4x_2 \neq 0 \} = \emptyset \tag{3.36}$$

$$(0,1,3)\ \ h_2 = f_3 = 2x_1 \tag{3.37}$$

$$s_2 = 0 \tag{3.38}$$

$$(1,0,1)\ \ h_2 = f_1 = x_3 \tag{3.39}$$

$$s_2 = \det \begin{pmatrix} \partial h_1/\partial x_1 & \partial h_1/\partial x_3 \\ \partial h_2/\partial x_1 & \partial h_2/\partial x_3 \end{pmatrix} = \det \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = 2 \tag{3.40}$$

$$Y_2 := \{ \mathbf{x} \in X'' \mid 2 \neq 0 \} = X'' \tag{3.41}$$

$$U_2 := \left\{ \mathbf{x} \in \mathbb{R}^3 \mid 2x_1 = 0, x_3 = 0 \right\} \tag{3.42}$$

$$\text{Since } i_2 = 3 \text{ it is impossible to compute} \tag{3.43}$$

$$\text{further derivatives.} \tag{3.44}$$

$$\text{We conclude that} \tag{3.45}$$

$$Y_2 \text{ is smooth and has codimension 3.} \tag{3.46}$$

As expected, the algorithm outputs the set of strata

$$\begin{aligned} \mathcal{X}_1 &:= \emptyset \\ \mathcal{X}_2 &:= \left\{ x_1^2 - x_2^2 = 0, x_3 = 0, 2x_1 \neq 0 \right\} \\ \mathcal{X}_3 &:= \{(0,0,0)\} \end{aligned}$$

3.3 Optimisation of the Output

3.3.1 Determining the codimension and basic formula for each stratum

Gabrielov and Vorobjov (1995), Theorem 2 assert that each stratum of codimension k can be represented by a conjunction containing k equations. I.e., each stratum is basic. A naive implementation of the procedure leads to a non-basic representation as shown in the following example

Example 3.1. Consider the semialgebraic set

$$S := \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1 x_2 = 0\}$$

Apply the algorithm.

$$\begin{aligned}
k = 0, X' &:= \left\{\mathbf{x} \in \mathbb{R}^2 \mid f_1 := x_1 x_2 = 0\right\} \\
(0,1,1)\, h_1 &= f_1 = x_1 x_2 \\
s_1 = \partial f_1/\partial x_1 &= x_2 \\
Y_1 &:= \{\mathbf{x} \in X' \mid y \neq 0\} \\
U_1 &:= \left\{\mathbf{x} \in \mathbb{R}^2 \mid x_1 x_2 = 0, x_2 \neq 0\right\} \\
&\text{Proceed by induction} \\
k = 1, h_1 = f_1, i_1 = 1, X'' &= Y_1, F_1 = \{f_1\} \\
(1,1)\, , h_2 &= f_1 \\
s_2 &= \begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_1/\partial x_2 \end{pmatrix} = 0 \\
&\text{since both columns are equal.} \\
\text{Return. } Y_1 &\text{ is smooth.}
\end{aligned}$$

Observe that the formula for Y_1 contains one equation, $h_1 = f_1$. Hence Y_1 is a smooth stratum in basic representation and we can conclude that it has codimension 1.

For the next nonzero derivative, with index $(1,0,1)$, we get

$$\begin{aligned}
k = 0, X' &:= \left\{\mathbf{x} \in \mathbb{R}^2 \mid x_1 x_2 = 0, y = 0\right\} \\
(1,0,1)\ \ h_1 &= f_1 = x_1 x_2 \\
s_1 = \partial f_1/\partial x_2 &= x_1 \\
Y_1 &:= \{\mathbf{x} \in X' \mid x_1 \neq 0\} \\
U_1 &:= \left\{\mathbf{x} \in \mathbb{R}^2 \mid x_1 x_2 = 0, x_1 \neq 0\right\} \\
&\text{Proceed by indection.} \\
k = 1, h_1 = f_1, i_1 = 2, X'' &= Y_1 \ldots \\
&\text{No partial derivatives can be computed.} \\
\text{Return. } Y_1 &\text{ is smooth.}
\end{aligned}$$

Notice that the inductive step returns immediately. This is because $i_1 = 2$, so it is not possible to take further partial derivatives with respect to $j > i_1$. In the formula for

$$Y_1 := \{\mathbf{x} \in \mathbb{R}^2 \mid f_1 := x_1 x_2 = 0, h_1 := x_2 = 0, x_1 \neq 0\},$$

two different equations, f_1 and h_1 appear. However, Y_1 is not in basic representation. Indeed, the equation $f_1 = 0$ is redundant, since f_1 is equal to zero for all points at which h_1 is equal to zero. Thus f_1 can be discarded, and we see

that only one equation, h_1, is required to represent Y_1 in basic form. We can conclude that Y_1 is smooth and has codimension 1.

Finally, consider the derivative with index $(1,1,1)$.

$$\begin{aligned} k = 0, X' :=& \left\{\mathbf{x} \in \mathbb{R}^2 \mid x_1 x_2 = 0, x_2 = 0, x_1 = 0\right\} \\ (1,1,1)\ \ h_1 =& s_1^{(0,1,1)} = x_2 \\ s_1 = \partial f_1/\partial x_2 =& 1 \\ Y_1 :=& \{\mathbf{x} \in X' \mid 1 \neq 0\} \\ U_1 :=& \left\{\mathbf{x} \in \mathbb{R}^2 \mid x_2 = 0, 1 \neq 0\right\} \end{aligned}$$

Again, the inductive step returns immediately because $i_1 = 2$, but this time we can see that Y_1 is not an open subset of U_1. As mentioned above, the algorithm is unable to determine this without solving a quantifier elimination problem. Instead, we may be able to determine the codimension by looking at the basic representation of

$$Y_1 := \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1 x_2 = 0, x_1 = 0, x_2 = 0\}.$$

(Note that the constraint $1 \neq 0$ is dropped since it is trivial and adds no information.) We see that three equations appear in the definition of Y_1, but the first one, $f_1 := x_1 y_2$ is redundant again. Thus, the two equations $x_2 = 0, x_1 = 0$ define Y_1 and we can conclude that Y_1 is smooth and has codimension 2.

This example shows two things. Firstly, not all functions are required to represent the smooth stratum and secondly k, the current step of induction, gives a lower bound on the codimension of strata. Gabrielov and Vorobjov (1995), Theorem 2 states that

$$\det \begin{pmatrix} \partial h_1/\partial x_{i_1} & \dots & \partial h_1/\partial x_{i_k} \\ \vdots & & \vdots \\ \partial h_k/\partial x_{i_1} & \dots & \partial h_k/\partial x_{i_k} \end{pmatrix} \neq 0$$

at every point of a stratum X_k of codimension k which includes the functions $h_1, \dots, h_k$ in its definition. As we saw in the example, other functions vanish at every point of X_k. We need to find the "basic" set of k functions needed to define X_k.

3.3.2 Discarding redundant equations

Let $F \subset \mathbb{Z}[x_1, \dots, x_n]$ be a set of polynomials, indexed by $(m_{i_1}, \dots, m_1, j)$ and, for $\mathbf{m} = (0, \dots, 0, m_\ell, \dots, m_1, j), \ell > i_1$ let

$$s_k^{m_n,\dots,m_1,j} = \partial_{\mathbf{h},\mathbf{i},n}^{m_n} \partial_{\mathbf{h},\mathbf{i},i_1+1}^{m_{i_1+1}} f$$

be the partial derivative computed at index $\mathbf{m}$, where $f \in F$ has index $(m_1, \dots, m_1, j)$ be one of the partial derivatives computed during the algorithm. Denote by G the set of derivatives with index lexicographically less than

$\mathbf{m}$. $Y_k \subset Y_{k-1}$ is defined such that every function in G is equal to zero while the function $h_k^{\mathbf{m}}$ is not. Define the smooth manifold

$$U_k := \{\mathbf{x} \in \mathbb{R}^n \mid h_1(\mathbf{x}) = 0, \ldots, h_{k-1}(\mathbf{x}) = 0, h_k(\mathbf{x}) = 0, s_k^{\mathbf{m}}(\mathbf{x}) \neq 0\}.$$

of codimension k, where the function $h_k := s_k^{(0,\ldots,0,m_\ell-1,m_{\ell-1},\ldots,m_1,j)}$. According to Gabrielov and Vorobjov (1995), Theorem 2, Y_k is smooth and has codimension k if it is an open subset of U_k. In this case, only k of the polynomials will be needed to define Y_k. Note that inequations $s_1, \ldots, s_{k-1}$ which appear in the definition of $Y_{k-1} \supset Y_k$ should also be taken into account. Begin with a set

$$Y' := \{\mathbf{x} \in \mathbb{R}^n \mid h_1(\mathbf{x}) = 0, \ldots, h_{k-1}(\mathbf{x}) = 0, s_1(\mathbf{x}) \neq 0, \ldots, s_(\mathbf{x}) \neq 0, s_k(\mathbf{x}) \neq 0\} \subset Y_k.$$

Consider each function $g \in G$ in reverse lexicographical order. If $g(\mathbf{x}) = 0$ at every point $\mathbf{x} \in Y'$ then it is redundant and should be discarded. Otherwise, g should be included in the defining equations for Y_k. Since Y_k has codimension k, and Y' is already an open, basic set of codimension $k-1$, only one more polynomial should be needed. If Y_k is not an open subset of U_k, then more polynomials will be required to define it, and the number of polynomials gives the codimension.

A polynomial $g \in \mathbb{Z}[x_1, \ldots, x_n]$ is redundant in the definition of the semialgebraic set $S \subset \mathbb{R}^n$ if the following formula is true

$$\exists x_1, \ldots, x_n, a \in \mathbb{R}^{n+1} \mid (x_1, \ldots, x_n) \in S, a \neq 0, g(x_1, \ldots, x_n) = a.$$

This problem can also be formulated using the emptiness check. g is redundant in the definition of S if the semialgebraic set

$$\{\mathbf{x} \in S \mid g(\mathbf{x}) \neq 0\}$$

is non-empty.

3.3.3 From Basic Algebraic sets to Arbitrary Semialgebraic sets

Gabrielov and Vorobjov (1995), Corollary 1 asserts that the algorithm can be applied to arbitrary semi-pfaffian sets and, therefore, without modification, to arbitrary semialgebraic sets. Let $S \subset \mathbb{R}^n$ be a semialgebraic set represented by a quantifier-free Boolean formula containing polynomials

$$f_1, \ldots, f_s.$$

Consider each "sign set"

$$S_{(*_1,\ldots,*_s)} := \{\mathbf{x} \in \mathbb{R}^n \mid f_1(\mathbf{x}) *_1 0, \ldots, f_s(\mathbf{x}) *_s 0\}$$

where $*_i, 1 \leq i \leq s \in \{<, >, =\}$. S can be partitioned into a finite union of some of the s^3 sign sets. If we apply the smooth stratification algorithm to each $S_{(*_1,\ldots,*_s)} \subset S$, and take the union of strata, we will obtain a smooth stratification of the set S. Indeed, since no two sign sets have non-empty intersection, it is clear that no two strata intersect, and since a finite number of the sign-sets form a partition of S, the smooth strata of these sign-sets also forms a partition of S.

3.3.4 Implementation in C

This algorithm has been implemented in `C`, using the computer algebra library `SACLIB`. `QEPCAD` is used to perform emptyness checks using cylindrical algebraic decomposition. Below is an example of the program's output, given the basic semialgebraic set

$$\{z = 0, x^2 - y^2 = 0\} \subset \mathbb{R}^3,$$

discussed in Section 3.2.

```
Enter a variable list.
Please enter a QEPCAD formula defining a basic semialgebraic set.
(x,y,z)
[ z = 0 /\ x^2 - y^2 = 0 ].

Stratification computed. Please select an option.
  1. print all polynomials
  2. print all strata
  0. exit

1
All polynomials generated by the smooth stratification algorithm:
2 x
-y^2 + x^2
z
1
2 x
z
-y^2 + x^2

Stratification computed. Please select an option.
  1. print all polynomials
  2. print all strata
  0. exit
2
List of all strata.
Strata of codimension 1

Strata of codimension 2
X_(2,1):
  { z == 0
  , -y^2 + x^2 == 0
  , 2 x /= 0
  , 2 x /= 0
  }

Strata of codimension 3
```

```
X_(3,1):
  { -y^2 + x^2 == 0
  , z == 0
  , 2 x == 0
  }
```

The program works in a similar way to `QEPCAD` (see Section 7.1.1), taking as input the variable list (which specifies the order) and a conjunction of polynomial equations, inequations and inequalities (written as a QEPCAD "prenex" formula). The stratification is then computed and there is a basic REPL which allows the user to see information about the polynomials and strata produced.

Exerpts of the code are now presented and discussed. The entrypoint for the algorithm is `stratify`.

Input:

- `Word r` $\in \mathbb{Z}$ (ambient dimension),
- `Word L` $= (f_1, \ldots, f_k) \subset \mathbb{Z}[x_1, \ldots, x_r]$,
- `Word Ineqs` $= (g_1, \ldots, g_\ell) \subset \mathbb{Z}[x_1, \ldots, x_r]$,
- `Word V` $= (v_1, \ldots, v_r)$ is a variable list,

such that $X = \{f_1 = 0, \ldots, f_k = 0, g_1 > 0, \ldots, g_\ell = 0\}$ is the basic semialgebraic set to be stratified.

Output:

- `Word S`: set of smooth strata $(\mathcal{X}_1, \ldots, \mathcal{X}_r)$, such that $\mathcal{X}_i$ contains strata of comidemnios i for $1 \leq i \leq r$, or `NIL` if X is a smooth subset.

```
1  Word stratify(Word r, Word L, Word Ineqs, Word V, Word *S_)
2  {
3      // initialise strata S
4      *S_ = NIL;
5      int i = 0;
6      while (i < r) {
7          *S_ = COMP(NIL, *S_);
8          ++i;
9      }
10
11     // initialise the inequalities as a QEPCAD formula
12     Word D, P;
13     Word F = NIL;
14     while (Ineqs != NIL) {
15         ADV(Ineqs, &P, &Ineqs);
16
17         F = COMP(LIST4(GTOP, P, r, NIL), F);
18     }
19
```

```
20      // initialise the input set of polynomials with their degrees
21      Word Fs = NIL, s = 0;
22      while (L != NIL) {
23          ADV(L, &P, &L);
24          ++s;
25
26          D = DEG(r, P);
27          Fs = COMP(LIST2(P, D), Fs);
28      }
29
30      // initial i0 = FIRST(I1) = 0. h0 = FIRST(Hs) = 0, Minor is
        ↪  the empty matrix
31      int strata_appended;
32      Word Gs = strat_helper(r, V, F, 1, s, Fs, LIST1(0), LIST1(0),
    ↪  NIL, NIL, &strata_appended, S_);
33
34      if (strata_appended == 0) {
35          // X is smooth
36          *S_ = NIL;
37      }
38
39      return Gs;
40  }
```

This function sets up the data structures required by `stratify_helper`, the recursive function which performs stratification of some input set $X \subset \mathbb{R}^n$.

Input:

- `Word r` $\in \mathbb{N}$, `Word V`: variable list, Word Ineqs‘: qepcad formula of inequalities $\{g_1 > 0, \ldots, g_\ell > 0\}$,
- `Word k` $1 \leq k \leq r$ is the codimension currently beind worked with,
- `Word np`: number of polynomials,
- `Word Fs` $((f_1, \deg(f_1)), \ldots, (f_{\text{np}}, \deg(f_{\text{np}})))$ is the list of imput polynomials,
- `Word Is` is a partial index $(0, , i_1, \ldots, i_{k-1})$,
- `Word Hs` is the set of polynomials $(h_{i_1}, \ldots, h_{i_{k-1}})$,
- `Word Qs` is a list of polynomials $\subset \mathbb{Z}[x_1, \ldots, x_n]$,
- `Word Minor` is a $((k-1) \times (k-1))$-matrix $A = |a_{ij}| = \partial h_i / partial x_j$ for $h_i \in$ Hs, j $\in$ Is.

such that the input set X is defined by equations `Fs` and `Hs`, inequalities `Ineqs` and inequations `Qs`.

Output:

- `Word Gs` $\subset \mathbb{Z}[x_1, \ldots, x_r]$ is the set of all partial differentials (for steps $k \leq j \leq r$),
- `int strat_count_` is the number of strata appended (for steps $k \leq j \leq r$),

- `Word S` is the set of strata (for steps $k \leq j \leq r$).

```
Word strat_helper(Word r, Word V, Word Ineqs, Word k, Word np,
↪  Word Fs, Word Is, Word Hs, Word Qs, Word Minor, int
↪  *strat_count_, Word *S_)
{
    // add constraints x_1 = 0, ..., x_{i1 - 1}  0
    Word i0 = FIRST(Is); // number of variables considered so far

    // base case for no polynomials?
    if (np == 0 || i0 >= r) {
        *strat_count_ = 0;
        return NIL;
    }

    // list of H polynomials (rev order) without the last (zero)
    ↪  one.
    Word Hs1 = RED(CINV(Hs));

    // set up return value
    Word Gs1 = NIL; // list of all differentials computed in this
↪  round, to return
    Word Gs2 = NIL; // list of differentials produced during
↪  induction
    Word Gs3 = NIL; // store Gs on current round

    // set up working array
    Word g_count = np; // how many differentials computed so far,
↪  index in Gs
    Word Gs = NIL; // list of all differentials computed in this
↪  round, working set

    // metadata
    Word Ms[np]; // number of steps before maximum
↪  differentiation variable (i1) should be incremented
    Word Dvs[np]; // list of current differentiation variable
↪  (i1)
    Word Backup[np]; // first polynomial in the list
    Word Chase[np]; // first element is h1
    Word ChaseIndex[np]; // chase array index of polynomial h1
    Word Append[np]; // pointer to last but one element in list,
↪  for quick appending.

    Word p_index = 0; // initial polynomial index, ranges over 0
↪  <= p_index < np

```

```
34      // initialise metadata for each input polynomial
35      while (p_index < np) {
36          Word F1;
37          ADV(Fs, &F1, &Fs);
38
39          Word D = REDI(SECOND(F1), i0);
40          Ms[p_index] = COMP(1, LCOPY(D)); // neet do copy as we
    ↪   modify later
41          Dvs[p_index] = i0;
42          Backup[p_index] = F1;
43          Chase[p_index] = F1;
44          ChaseIndex[p_index] = 0;
45          Append[p_index] = RED(F1);
46          Gs = COMP(LCOPY(F1), Gs);
47          Gs3 = COMP(FIRST(F1), Gs3);
48
49          // increment index
50          ++p_index;
51      }
52
53      // main loop, consider each index (j, m_{i0 + 1}, ..., m_r)
        ↪   in lex order
54      Word n_finished = 0; // each polynomial has a different max
    ↪   index, keep track of how many indices are maxed out
55      Word count = 0; // number of derivatives computed so far,
    ↪   scalar value of (m_{i0 + 1}, ..., m_r)
56
57      int strat_count = 0;
58      while (n_finished < np) { // stop once differential index for
        ↪   every polynomial is maxed
59          // reached the end of polynomial list, cycle back to
            ↪   beginning and consider next differential index I
60          if (p_index == np) {
61              p_index = 0;
62              n_finished = 0;
63
64              ++count;
65          }
66
67          Word v = Dvs[p_index]; // differentiation variable
68          Word m = FIRST(Ms[p_index]);
69
70          // update variable v and chaser list
71          if (count >= m && v == r) { // rollover, but we're
            ↪   finished
```

```
72              ++n_finished; // this polynomial is done.
73              ++p_index; // next polynomial
74
75              continue; // skip it
76          } else if (count >= m) { // rollover - increment
    ↪  differentiation variable
77              ++v; // next variable ...
78              Dvs[p_index] = v; // ... and store
79              Chase[p_index] = Backup[p_index];
80              ChaseIndex[p_index] = 0;
81
82              // calculate next m
83              Word M1 = RED(Ms[p_index]);
84              Word d = FIRST(M1);
85              SFIRST(M1, d * m);
86              Ms[p_index] = M1;
87
88              // degree zero - no derivatives taken for this
                ↪  variable. next iteration will increment the
                ↪  variable.
89              if (d == 1) continue;
90          }
91
92          // next polynomial
93          Word F1 = Append[p_index];
94
95          // compute s_k =
            ↪  partial_{(h_1,...,h_{k-1}),(i_1,...,i_{k-1}),v} h_k
96          // get h_k and its degree
97          Word P;
98          ADV(Chase[p_index], &P, &Chase[p_index]);
99
100         // construct jacobi matrix using h1 = P and i1 = v
101         Word Jacobi = JacobiFromMinor(r, P, v, Hs, Is, Minor);
102
103         // compute partial differential, determinant of jacobi
            ↪  matrix
104         // note that Q may be a constant, we need to preserve it
105         Word Q = MAIPDE(r, Jacobi); // next derivative is the
    ↪  jacobi determinant
106         Word Qdeg = DEG(r,Q);
107         bool q_const = IPCONST(r, Q);
108         Word Q1 = LIST2(Q, Qdeg);
109         Word Qs1 = Qs;
110         if (!q_const) Qs1 = COMP(Q, Qs1); // because c /= 0 is
            ↪  trivially true
```

```
111
112         // candidate stratum Y1 on which Q /= 0
113         //   - 0 /= 0 is trivially false, so we immediately
            ↪  conclude that it's empty
114         //   - const /= 0 is trivially true, so assuming the
            ↪  algebraic set Gs3 is non-empty, Y1 is trivially
            ↪  non-empty
115         //   - if Q is constant, Y1 must be the last candidate,
            ↪  since the next one includes a trivially false
            ↪  equation
116         if (Q != 0 && (q_const || !ISEMPTY(r, V, Gs3, Qs1,
            ↪  Ineqs))) {
117             // Gs2 contains derivatives computed during recursion
118             int strata_appended;
119             Gs2 = strat_helper(r, V, Ineqs, k + 1, g_count, Gs,
    ↪  COMP(v, Is), COMP(P, Hs), Qs1, Jacobi, &strata_appended, S_);
120             Gs1 = CONC(Gs1, Gs2);
121
122             // determine if all derivatives at step k+1 vanish on
                ↪  the set Y1
123             strat_count += strata_appended;
124
125             // append stratum if none were appended during
                ↪  induction
126             if (strata_appended == 0) {
127                 Word Y, k1;
128                 construct_stratum_basic(k, r, V, Hs1, Q, Qs, Gs3,
    ↪  Ineqs, &k1, &Y);
129                 append_stratum(S_, k1, Y);
130                 ++strat_count;
131             }
132
133             Gs3 = COMP(Q, Gs3);
134             Gs = COMP(Q1, Gs);
135             ++g_count;
136
137             if (q_const) {
138                 // the next candidate will include the trivially
                    ↪  false eequation const = 0. stop.
139                 break;
140             }
141         }
142
143         // append derivative to Fs, preserving zeroes
144         SRED(F1, Q1);
```

```
145         Append[p_index] = RED2(F1);
146
147         // next polynomial please.
148         Chase[p_index] = RED(Chase[p_index]);
149         ChaseIndex[p_index] = ChaseIndex[p_index] + 1;
150         ++p_index;
151     }
152
153     Gs1 = CONC(Gs1, Gs3);
154     *strat_count_ = strat_count;
155     return Gs1;
156 }
```

- For the base case, where there are no polynomials, or index $i_k \geq r$, (Line 7), there is nothing to do. No strata are appended and no polynomials computed.
- Prior to Line 57, initialisation is done. Metadata for each polynomial in `Fs` is stored.
- The main loop, beginning on Line 57, computes partial derivatives in ascending lexicographical order of index $(i_r, \ldots, i_{k+1}, j)$, $1 \leq j \leq \text{np}$.
- Lines 57-100 keeps track of the current polynomial `P` and "differentiation variable" `v` $\in x_{i_k+1}, \ldots, i_r$.
- Line 101 calls the function `JacobiFromMinor`, which takes a $((k-1) \times (k-1))$-matrix $A = |a_{ij}| = \partial h_i / \partial x_j$, a polynomial f and index ℓ, and returns the $(k \times k)$-matrix obtained from A by appending column $(\partial h_1/\partial x_\ell, \ldots, \partial h_{k-1}/\partial x_\ell)^T$ and row $(\partial f/\partial x_{i_1}, \ldots, \partial f/\partial x_{i_{k-1}}, \partial f/\partial \ell)$. I.e., it constructs the matrix from the partial differential operator (Definition 3.2) and saves recomputing the whole matrix of partial derivatives. Note that, when $k = 1$, `JacobiFromMinor` returns the one-element matrix $\partial f/\partial \ell$.
- On line 105, the determinant of the matrix computed on Line 101 is calculated, for the partial differential operator.
- Line 116 determines if the candidate stratum is empty (see comment on Lines 112-115).
- If non-empty, we proceed by induction, calling `strat_helper` recursively (for $k+1$, appending index `v` and polynomial `P`, and passing the $(k \times k$-matrix as the `Minor`). The number of strata of codimension $> k$ which were added to `S` is returned. If no strata were appended, all the partial derivatives vanish identically on the candidate, and we can conclude that it is smooth and has codimension k.
- Line 128 represents the candidate stratum as a basic semialgebraic set containing k equations. The function `construct_stratum_basic` is discussed shortly.
- The rest of the function is responsible for updating the metadata and keeping track of the list of polynomials `Gs1` to be returned.

The function `construct_stratum_basic` is responsible for representing a candidate stratum as a basic semialgebraic set defined using k polynomial equations. It discards redundant polynomials as described in Section 3.1.1.

Input:

- `Word r` $\in \mathbb{N}$: ambient dimension,
- `Word k` $0 \leq k \leq r$: codimension of candidate stratum,
- `Word V`: variable list,
- `Word Hs` $= (h_{i_1}, \ldots, h_{i_{k-1}}) \subset \mathbb{Z}[x_1, \ldots, x_r]$,
- `Word Q` $\in \mathbb{Z}[x_1, \ldots, x_r]$,
- `Word Qs` $\subset \mathbb{Z}[x_1, \ldots, x_r]$,
- `Word Gs` $\subset \mathbb{Z}[x_1, \ldots, x_r]$,
- `Word Ineqs`: a QEPCAD formula of inequalities,

such that `Hs`, `Qs`, `Q`, `Gs` and `Ineqs` define a candidate stratum Y_k. All of `Hs` and `Gs` are equal to zero on Y, all of `Q` and `Qs` are not equal to zero on Y and the formula `Ineqs` defines a semialgebraic set containing Y.

Output=

- `Word k1` $\geq$ `k` is the actual codimension of Y,
- `Word F` is the stratum Y defined by `k1` equations and the inequations and inequalities from `Qs`, `Q` and `Ineqs`

```
1  void construct_stratum_basic(Word k, Word r, Word V, Word Hs,
   ↪  Word Q, Word Qs, Word Gs, Word Ineqs, Word *k1_, Word *Y_)
2  {
3      Word L, P, i;
4      Word Eqs = NIL, Ineqats = NIL;
5
6      // we may have Q = const, if so don't include it.
7      if (!IPCONST(r, Q)) {
8          Qs = COMP(Q, Qs);
9      }
10
11     // add Qs and Hs to the definition of stratum
12     L = Hs, i = k - 1;
13     while (L != NIL) {
14         // equation h = 0
15         ADV(L, &P, &L);
16         Eqs = COMP(P, Eqs);
17
18        --i;
19     }
20
21     L = Qs;
22     while (L != NIL) {
```

```
23          // inequation s /= 0
24          ADV(L, &P, &L);
25          Ineqats = COMP(P, Ineqats);
26      }
27
28      // we start with functions (h_1,...,h_{k-1}) and then
        ↪  determine which ones from Gs are required.
29      Word k1 = k - 1; // store the codimension.
30
31      // attempt to add mroe polynomials from the list Gs of
        ↪  candidate functions.
32      while (Gs != NIL) {
33          ADV(Gs, &P, &Gs);
34
35          if (!ISEMPTY(r, V, Hs, COMP(P, Qs), Ineqs)) {
36              // if this set is non-empty, then there is a point at
                ↪  which P /= 0, thus P = 0 is necessary
37              Hs = COMP(P, Hs);
38              Eqs = COMP(P, Eqs);
39
40              ++k1; // addition of a new polynomial increases the
↪   codimension
41          }
42      }
43
44      // assign to return
45      *k1_ = k1;
46      *Y_ = LIST2(Eqs, Ineqats);
47  }
```

The idea is to use emptiness checking to find the minimal number, `k1` of equations needed to define the stratum Y as a basic semialgebraic set. `k1` is the codimension of Y.

The function `ISEMPTY` performs emptiness checking.

Input:

- `Word r` $\in \mathbb{N}$,
- `Word V`: variable list $(x_1, \ldots, x_r)$,
- `Word Fs` $= (f_1, \ldots, f_s) \subset \mathbb{Z}[x_1, \ldots, x_r]$,
- `Word Gs` $= (h_1, \ldots, h_t) \subset \mathbb{Z}[x_1, \ldots, x_r]$,
- `Word Ineqs`: list of atomic QEPCAD formulas $\{g_1 > 0\}, \ldots, \{g_\ell > 0\}$

such that the input set $S \subset \mathbb{R}^r$ can be defined by the QFF

$$F = \{f_1 = 0, \ldots, f_s = 0, h_1 \neq 0, \ldots, h_t \neq 0, g_1 > 0, \ldots, g_\ell > 0\}$$

Output:

- `true` if S is empty, `false` otherwise.

```
Word ISEMPTY(Word r, Word V, Word Fs, Word Gs, Word Ineqs)
{
    Word P, Ct, Cf;

    // start with the list of inequalities
    Word F = Ineqs;

    // add equations
    while (Fs != NIL) {
        ADV(Fs, &P, &Fs);

        F = COMP(LIST4(EQOP, P, r, NIL), F);
    }

    // and inequations
    while (Gs != NIL) {
        ADV(Gs, &P, &Gs);

        F = COMP(LIST4(NEOP, P, r, NIL), F);
    }

    // complete formula by adding the inequation and conujnction
    F = COMP(ANDOP, F);

    // re-initialise qepcad before each run
    QepcadCls Q;
    INITSYS();

    // set input formula
    Q.SETINPUTFORMULA(V,LIST4(r, r, NIL, F));
    Q.PRDQFF();
    Q.CADautoConst();

    // special case: trivially false
    if (Q.GVPC == 0) {
        return true;
    }

    LISTOFCWTV(Q.GVPC, &Ct, &Cf);

    /* compute cad */
    return Ct == NIL;
```

```
}
```

The function constructs F, the defining formula for S and uses QEPCAD to construct a CAD of $\mathbb{R}^r$ compatible with S. If S is empty, then this CAD will contain no true cells, so the function checks this condition.

3.4 Complexity Analysis

We first briefly summarise the semi-Pfaffian case, as presented by Gabrielov and Vorobjov (1995). The algorithm from Gabrielov and Vorobjov (1995), Theorem 2 takes as input a semi-Pfaffian set with the format (r, N, α, β, n) (see Definition 2.12) and has a complexity upper bound of

$$3^N N^{n+1} \mathcal{B}$$

where

$$\mathcal{B} = (\alpha + \beta + 1)^{(r+1)^{O(n)}}.$$

Without the Oracle, the total number of strata, some of which may be empty, does not exceed $N^{n+r}\mathcal{B}$, each of which has format $(r, N\mathcal{B}, \alpha, \mathcal{B}, n)$. If the Oracle, used for checking emptiness, is allowed, then the algorithm has complexity not exceeding $N^{n+r}\mathcal{B}$.

Now consider this algorithm applied to the semialgebraic set

$$X := \{\mathbf{x} \in \mathbb{R}^n \mid f_1 = 0, \ldots, f_k = 0, g_1 > 0, \ldots, g_\ell > 0\}, \tag{3.47}$$

where all $s := k + \ell$ functions f_i, g_j are polynomials with maximum degree d – Pfaffian functions of order 0 and degree (α, β). The upper complexity bound for the algorithm without Oracle is

$$3^s s^{n+1} (d+1)^2.$$

The number of strata, some of which may be empty, does not exceed

$$s^n (d+1)^2.$$

Each stratum is defined by at most $s(d+1)^2$ polynomial equations and inequalities of maximum degree $(d+1)^2$. If the Oracle is allowed, no empty strata will be produced and the algorithm has complexity not exceeding

$$s^{n+1} (d+1)^2.$$

These bounds follow immediately from Gabrielov and Vorobjov (1995), Theorem 6.2. We will now analyse the complexity of the algorithm presented in Section 3.1, which performs emptiness checking on candidate strata.

There are $s^n(d+1)^2$ candidate strata Y_k, and the algorithm determines whether each of them is empty, in order to decide whether all partial differential equations

vanish identically on the input set X. In the implementation, the check is done using CAD (a call to `QEPCAD-B`). For the purposes of complexity analysis, we will use a theoretical algorithm, having singly exponential complexity in the number of variables. Basu et al. (1998) propose an algorithm which takes a semialgebraic set $S \subset \mathbb{R}^n$ defined by s polynomials with maximum degree d in $n < s$ variables and produces a "witness" – a point contained in every cell in a decomposition of S. The algorithm can be used to determine the emptiness of S and requires $s(s/n)^n d^{O(n)}$ arithmetic operations. This is asymptotically the same as

$$(sd)^{O(n)}.$$

Using the bounds for the number and degrees of polynomials defining strata, each emptiness check costs

$$\left(s(d+1)^2(d+1)^2\right)^{O(n)} = \left(s(d+1)^4\right)^{O(n)},$$

which is asymptotically the same as

$$\left(s(d+1)\right)^{O(n)}$$

This operation must be performed for all of the $s^n(d+1)^2$ candidate strata, resulting in a total complexity for emptiness checking of

$$\left(s(d+1)\right)^{O(n)^2}.$$

The total complexity of stratification, given by the sum of the complexity of applying the algorithm and performing the emptiness checks, is

$$3^s s^{n+1}(d+1)^2 + \left(s(d+1)\right)^{O(n)^2}.$$

Since $s^{n+1}(d+1)^2$ is bounded from above by $\left(s(d+1)\right)^{O(n)^2}$, we get a total complexity upper bound of

$$3^s \left(s(d+1)\right)^{O(n)^2}.$$

3.5 Test Cases

Some test cases are now presented

1. - **Input:** $(x, y), \{xy = 0\}$
   - **Output:**

```
X_(1,1):
  { y == 0
  , x /= 0
  }
X_(1,2):
```

```
  { x y == 0
  , y /= 0
  }

Strata of codimension 2
X_(2,1):
  { y == 0
  , x == 0
  }
```

2. • **Input:** $(x, y), \{x^5y^4 = 0\}$

   • **Output:**

```
Strata of codimension 1
X_(1,1):
  { 120 y^4 == 0
  , 24 x^5 /= 0
  }
X_(1,2):
  { x^5 y^4 == 0
  , 120 y^4 /= 0
  }

Strata of codimension 2
X_(2,1):
  { 120 y^4 == 0
  , 24 x^5 == 0
  }
```

3. • **Input:** $(x, y)x^2 + y^2 = 1$

   • **Output:**

```
Strata of codimension 1
X_(1,1):
  { y^2 + x^2 - 1 == 0
  , 2 x /= 0
  }

Strata of codimension 2
X_(2,1):
  { y^2 + x^2 - 1 == 0
  , 2 x == 0
  , 4 y /= 0
  }
```

4. • **Input:** $(x, y)(x^2 + y^2 - 1)(x^2 - y^2) = 0$ *(unit circle plus a cross)*

   • **Output:**

```
Strata of codimension 1
X_(1,1):
  { y^4 + 2 x^2 y^2 - y^2 + x^4 - x^2 == 0
  , 4 x y^2 + 4 x^3 - 2 x /= 0
  }

Strata of codimension 2
X_(2,1):
  { 16 y^5 + 32 x^2 y^3 - 16 y^3 + 16 x^4 y - 16 x^2 y + 4 y == 0
  , 4 x y^2 + 4 x^3 - 2 x == 0
  , 4 y^2 + 12 x^2 - 2 /= 0
  , 320 y^6 + 832 x^2 y^4 - 352 y^4 + 704 x^4 y^2 - 576 x^2 y^2 + 112 y^2 + 19
  }
X_(2,2):
  { y^4 + 2 x^2 y^2 - y^2 + x^4 - x^2 == 0
  , 4 x y^2 + 4 x^3 - 2 x == 0
  , 4 y^2 + 12 x^2 - 2 /= 0
  , 16 y^5 + 32 x^2 y^3 - 16 y^3 + 16 x^4 y - 16 x^2 y + 4 y /= 0
  }
```

5. • **Input:** $(x, y, z), xyz = 0$

   • **Output:**

```
Strata of codimension 1
X_(1,1):
  { z == 0
  , x y /= 0
  }
X_(1,2):
  { y z == 0
  , x z /= 0
  }
X_(1,3):
  { x y z == 0
  , y z /= 0
  }

Strata of codimension 2
X_(2,1):
  { z == 0
  , x y == 0
  , y /= 0
  }
X_(2,2):
  { y z == 0
  , x z == 0
```

```
  , z /= 0
  }

Strata of codimension 3
```

6. • **Input:** $(x, y, z)x^2 + y^2 + z^2 = 0$

   • **Output:**

```
Strata of codimension 1

Strata of codimension 2

Strata of codimension 3
X_(3,1):
  { z^2 + y^2 + x^2 == 0
  , 4 y == 0
  , 2 x == 0
  }
```

<!—

7. • **Input:** $(x, y)x^2 + y^2 = 1$

   • **Output:** –>

Chapter 4

Quasi-affine cells

In this section, we develop an algorithm, based on Basu et al. (2015), Theorem 3.19, for constructing a CAD whose two-, one-, and zero-dimensional cells are the graphs of quasi-affine maps. This is the first step to constructing a monotone cylindrical decomposition, as described in Basu et al. (2015), Theorem 3.20.

Proposition 4.1. *(Basu et al., 2015, Theorem 3.19)*

Let $V_1, \ldots, V_k$ be bounded definable subsets in $\mathbb{R}^n$ with $\dim(V_i) \leq 2$ for each $1 \leq i \leq k$. Then there is a cylindrical cell decomposition of $\mathbb{R}^n$, compatible with each V_i such that every cylindrical cell contained in $V := V_1 \cup \ldots \cup V_k$ is the graph of a quasi-affine map.

The following will be proved.

Theorem 4.1. *Let*

$$F_1, \ldots, F_k$$

be quantifier-free Boolean formulas containing s different polynomials $\mathbf{F} = \{f_1, \ldots, f_s\} \subset \mathbb{Z}[x_1, \ldots, x_n]$, having maximum degree d, such that $F_i, 1 \leq i \leq k$ defines a bounded semialgebraic set $V_i \subset \mathbb{R}^n$ such that $\dim(V_i) \leq 2$.

Then there is an algorithm, taking $F_1, \ldots, F_k$ as input, which constructs an $\mathbf{F}$-invariant CAD $\mathcal{D}$ of $\mathbb{R}^n$ such that each cell $C \subset V_1 \cup \ldots \cup V_k$ is a smooth manifold and the graph of a quasi-affine map. $\mathcal{D}$ is obviously compatible with each $V_1, \ldots, V_K$. This algorithm has complexity

$$(s\,(d+1))^{2^{O(n)}},$$

which is also an upper bound on the number of cells in the CAD, number of polynomials and their degrees.

4.1 Computing the smooth two-dimensional locus of V

The proof of Basu et al. (2015), Theorem 3.19 begins by introducing W, the smooth 2-dimensional locus of $V = V_1 \cup \ldots \cup V_k$. Since V is a semialgebraic set, it can be represented by a quantifier-free Boolean formula F. W can be computed using the smooth stratification algorithm described in 3. In order to apply the algorithm, consider the sign sets $S_{(*_1,\ldots,*_s)} \subset \mathbb{R}^n$ of polynomials $f_1, \ldots, f_s$ appearing in the formula F, as described in Section 3.3.3. Observe that a finite number of these sign-sets $S_1, \ldots, S_m$ form a partition of V. For each $S_i \subset V$, apply the smooth stratification algorithm to obtain a family $(X_{i,1}, \ldots X_{i,n})$ of strata, where each $X_{i,j}$ is a smooth subset of S_i of codimension j. We have $\dim(S_i) \leq 2$, since each of the sets in the union V has dimension at most 2. Hence, every stratum $X_{i,j}$ of codimension $j < n-2$ will be empty. We obtain the smooth two-dimensional locus of V by taking the union of strata of codimension $n-2$. For each stratum, it is important to keep the set of $n-2$ polynomials defining it, as these will be needed in the next step. We will need the strata of codimension $n-1$, as each one-dimensional cell must also be the graph of a quasi-affine map.

4.1.1 An alternative method for computing the smooth two-dimensional locus

Recall from Section 2.2.1) that smooth cells can be guaranteed by choosing a certain projection operator. More precisely, according to McCallum (1988), Theorems 2.2.3 and 2.2.4, if there is a finite number of blow-up points, then the McCallum projection operator will result in a CAD where all cells are analytic submanifolds. Since $\dim(V_i) \leq 2$ for all $1 \leq i \leq k$, blow-up points in $\mathrm{cl}\,(()V_i)$ have dimension at most 0 (see the Remark after Definition 2.22). Hence, the McCallum projection operator is sufficient to produce smooth cells.

Let $\mathbf{F} \subset \mathbb{R}^n$ be the set of different polynomials appearing in the quantifier-free Boolean formulas defining V and use the McCallum projection operator to construct an F-invariant CAD, $\mathcal{E}$, of $\mathbb{R}^n$. Since cells of a CAD are disjoint by construction and smooth by the use of the McCallum projection operator, $\mathcal{E}$ forms a smooth stratification of V. Thus, we can take as W the (separate) sets of 2- and 1-dimensional cells of $\mathcal{E}$ which are contained in V. Recall, from Definition 2.20, that each cylindrical cell has a corresponding multi-index $(i_1, \ldots, i_n) \in \{0, 1\}^n$. Let $C \subset V$ be a 2-dimensional be a cell of $\mathcal{E}$, and suppose it has index $(i_1, \ldots, i_n)$. Since $\dim(C) = 2$, $\sum_{1 \leq i \leq n} = 2$. Let $i_{j_1}, = i_{j_2} = 1$ for $j_1 \neq j_2$ while all other elements of the index be equal to zero. Then $\mathrm{proj}_{\mathbb{R}^k}(C)$, for $k \neq j_1, k \neq j_2$, is the root of a projection polynomial $f_k \in \mathbb{Z}[x_1, \ldots, x_k]$. Hence, we have, $n-2$ projection polynomials at which every $\mathbf{x} \in C$ is equal to zero. The polynomials can be determined by using the set of signs of projection factors and used to define an algebraic variety $Y \supset C$. Note that we need some additional inequalities to define C properly, but these are not needed to compute the critical points

of projections in the next section. Similarly for 1-dimensional cells, only one element of the index will be equal to 1, while all other elements are equal to 0. It follows that we can find $n-1$ polynomials, in a similar way, which are equal to zero on each one-dimensional cell.

It is possible to compute W using either technique. Theoretically speaking, using the smooth stratification algorithm seems cleaner, as it is the correct tool for the job. In addition, the bound on number of strata is singly exponential, while the bound on number of CAD cells is doubly exponential. An estimate for the number of cells of strata having a certain dimension is not currently known, but it seems likely that the number of strata will be less than the number of CAD cells. Gabrielov and Vorobjov (2004), Theorem 3.7 gives a method (which can be turned into an algorithm) for efficiently listing all sign sets associated with a set of polynomials and Basu et al. (2006), Algorithm 14.21 describes how quantifier elimination can be performed in singly exponential time. However, in the implementation CAD is used to perform the emptiness check. Thus, the second option seems more appealing because a single recomputation of the CAD is preferrable to many calls to the CAD algorithm, which would be required for the stratification-based approach in the form that it has been implemented. In addition, Kremer and Ábrahám (2020) propose an algorithm, along with its implementation, for refining a CAD to be compatible with new polynomials. This gives us hope that we may be able to avoid a full re-computation. In conclusion, the second option, where we compute an initial CAD consisting of smooth cells, is more tractable in practice.

4.2 Ensuring that every cell is the graph of a quasi-affine map

We now discuss how to obtain quasi-affine cells. A k-dimensional cell C is the graph of a quasi-affine map if, for each affine coordinate subspace $L \subset \mathbb{R}^n$, the image of the projection map $\mathrm{proj}_L(C)$ is k-dimensional if and only if the map $\mathrm{proj}_L|_C$ is injective (see Definition 2.14). In order to satisfy this property, Basu et al. (2015), Lemma 3.19 states that the CAD $\mathcal{D}$ also needs to be compatible with the critical points, W_i and $W_{i,j}$, of projections of W onto one and two-dimensional coordinate subspaces $\mathrm{span}\{x_i, x_j\}$ and $\mathrm{span}\{x_i\}$.

Let W'_k be one of the strata (or smooth cylindrical cells) computed in the previous section, with $k \in \{1, 2\}$. Since W'_k has codimension $n-k$ and is a basic semialgebraic set, its formula contains $n-k$ polynomial equations and, possibly, some inequalities. Consider the Jacobi matrix

$$J_{i_1,\ldots,i_\ell} := \begin{pmatrix} \partial f_1/\partial x_{i_1} & \ldots & \partial f_1/\partial x_{i_\ell} \\ \vdots & & \vdots \\ \partial f_k/\partial x_{i_1} & \ldots & \partial f_{n-k}/\partial x_{i_\ell} \end{pmatrix},$$

where $\{i_1, \ldots, i_\ell\} \subset \{1, \ldots, n\}$. Let $\{j_1, \ldots, j_{n-\ell}\} = \{1, \ldots, n\} \setminus \{i_1, \ldots, i_\ell\}$.

Regular points of the projection of W' onto $\text{span}\{j_1, \ldots, j_{n-\ell}\}$ are those at which the matrix $J_{i_1,\ldots,i_\ell}$ has full rank. Thus, we can compute the critical points $W'_{i,j}$ of $\text{proj}_{\text{span}\{x_i,x_j\}}$ by finding the points at which the Jacobi matrix $J_{\{1,\ldots,n\}\setminus\{i,j\}}$ does not have full rank.

Note that, for $W'_{i,j}$, the corresponding $J_{\{1,\ldots,n\}\setminus\{i,j\}}$ will be a $((n-2)\times(n-2))$-matrix, so we simply need to find the points at which $\det(J_{\{1\ldots,n\}\setminus\{i,j\}}) = 0$. On the other hand, $J_{\{1,\ldots,n\}\setminus\{i\}}$ will be an $((n-2)\times(n-1))$-matrix. It will fail to have full rank if the determinants of all of the $((n-2)\times(n-2))$-minors is zero. Observe that the minors of $J_{\{1,\ldots,n\}\setminus\{i\}}, i \in \{1,\ldots,n\}$ are the $((n-2)\times(n-2))$-jacobi matrices $J_{\{1,\ldots,n\}\setminus\{i,j\}}, j \in \{1,\ldots,n\}\setminus\{i\}$. I.e., they will coincide with the square matrices associated with $W'_{i,j}$. Thus, we need to make the CAD have constant sign on the determinants of each of the matrices

$$J_{\{1,\ldots,n\}\setminus\{i,j\}} \text{ for all } 1 \le i < j \le n,$$

for all two-dimensional smooth strata (or cells) of V, whose definition includes $n-2$ polynomial equations.

Now consider the critical points Z of $\text{proj}_{\text{span}\{x_i,x_j\}}(W'_1)$, where W'_1 is a smooth one-dimensional stratum. If $\dim(Z) = 1$, then every point of $\text{proj}_{\text{span}\{x_i,x_j\}}(W'_1)$ is critical as W'_1 is smooth. Note that z may contain zero-dimensional critical points of $\text{proj}_{\text{span}\{x_i\}}$ and $\text{proj}_{\text{span}\{x_j\}}$. Otherwise, $\dim(Z) = 0$, so Z is a collection of isolated points (c_i, c_j). c_i is also a critical point of $\text{proj}_{\text{span}\{x_i\}}(W'_1)$ and c_j is also a critical point of $\text{proj}_{\text{span}\{x_j\}}(W'_1)$. Therefore, it is sufficient to consider only critical points of projections onto one-dimensional coordinate subspaces. Hence, we only consider Jacobi matrices $J_{\{1,\ldots,n\}\setminus\{i\}}$ for all $1 \le i \le n$, for the $n-1$ polynomials defining W'_1. Critical points are those at which their determinants are equal to zero.

4.2.1 Computing the quasi-affine CAD

We now present pseudo-code for the quasi-affine part of the algorithm.

Algorithm 2. *Quasi-Affine CAD*

Input:

$$(F_1, \ldots, F_k)$$

- $F_1, \ldots, F_k$*: a quantifier-free Boolean formula defining*

$$V = V_1 \cup \ldots \cup V_k$$

where $V \subset \mathbb{R}^n$ *and* $\dim(V_i) \le 2, 1 \le i \le k$.

Output:

$$(\mathcal{D})$$

- $\mathcal{D}$*: an* $\mathbf{F}'$*-invariant CAD of* $\mathbb{R}^n$ *compatible with each* $V_1, \ldots, V_k$ *such that each cell* $C \subset V$ *is the graph of a quasi-affine map, where* $\mathbf{F}' \subset \mathbb{Z}[x_1, \ldots, x_n]$

contains all polynomials from formulas $F_1, \ldots, F_k$ and, possibly, some additional polynomials which are zero at singular points of $V_1 \cup \ldots, \cup V_k$ and the critical points of projections of $Y_1 \cup \ldots \cup V_k$ onto one- and two- dimensional coordinate subspaces.

- *Let*
$$\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$$
be the set of polynomials appearing in the formulas $F_1, \ldots, F_k$.
- *Compute $\mathcal{E}$, an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$ (obviously compatible with each $V_1, \ldots, V_k$) using the McCallum projection operator.*
- *Let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ be the family of projection polynomials for $\mathcal{E}$, where each $\mathcal{A}_k \subset \mathbb{Z}[x_1, \ldots, x_k]$.*
- *For each cell $C_k \subset V$ with index $(i_1, \ldots, i_n) \in \{0,1\}^n$, such that $i_{j_1} = \ldots = i_{j_k} = 1$ while all other elements of the index are equal to 0 (since $\dim(V) \leq 2$, $k \leq 2$ and $\dim(C_k) = k$):*
  - *Let $f_1, \ldots, f_{n-k}$ be a set of projection polynomials, where $f_i \in \mathcal{A}_i$, which are equal to 0 on the cell C_k.*
  - *For each $i \in \{1, \ldots, n\} \setminus \{j_1, \ldots, j_k\}$, find a polynomial in $\mathcal{A}_i$ which is 0 at every point $\mathbf{x} \in C_k$. This can be done by checking the set of signs of projection factors.*
- *For each 2-dimensional cell $C_2 \subset V$ and pair $1 \leq i_1 < i_2 \leq n$:*
  - *Find the set of polynomials $f_1, \ldots, f_{n-2}$ defining C_2 as described above.*
  - *Let $\mathbf{m} = \{j_1, \ldots, j_{n-2}\} = \{1 \ldots, n\} \setminus \{i_1, i_2\}$.*
  - *Define*
$$J_{C_2,\mathbf{m}} := \begin{pmatrix} \partial f_1/\partial x_{j_1} & \ldots & \partial f_1/\partial x_{j_{n-2}} \\ \vdots & & \vdots \\ \partial f_{n-2}/\partial x_{j_1} & \ldots & \partial f_{n-2}/\partial x_{j_{n-2}} \end{pmatrix}, \tag{4.1}$$
an $((n-2) \times (n-2))$ Jacobi matrix.

  $\det(J_{C_2,\mathbf{m}}) = 0$ *at the critical points of* $\mathrm{proj}_{\mathrm{span}\{i_1,i_2\}}$.
- *For each 1-dimensional cell $C_1 \subset V$ and each $1 \leq i \leq n$:*
  - *Find the set of polynomials $f_1, \ldots, f_{n-1}$ defining C_1.*
  - *Let $\mathbf{m} = \{j_1, \ldots, j_{n-1}\} = \{1 \ldots, n\} \setminus \{i\}$.*
  - *Define*
$$J_{C_1,\mathbf{m}} := \begin{pmatrix} \partial f_1/\partial x_{j_1} & \ldots & \partial f_1/\partial x_{j_{n-1}} \\ \vdots & & \vdots \\ \partial f_{n-1}/\partial x_{j_1} & \ldots & \partial f_{n-1}/\partial x_{j_{n-1}} \end{pmatrix}, \tag{4.2}$$

an $((n-1)\times(n-1))$ *Jacobi matrix.*

$\det(J_{C_1,\mathbf{m}}) = 0$ *at the critical points of* $\mathrm{proj}_{\mathrm{span}\{i\}}$.

- *Return the CAD* $\mathcal{D}$*, which is sign invariant on*

$$\begin{aligned}\mathbf{F}' = & \quad \mathbf{F} \\ \cup & \bigcup_{C_2\subset V,\dim(C_2)=2,\mathbf{m}=\{1,\ldots,n\}\setminus\{i_1,i_2\},1\le i_1<i_2\le n} \det\left(J_{C_2,\mathbf{m}}\right) \\ \cup & \bigcup_{C_1\subset V,\dim(C_1)=1,\mathbf{m}=\{1,\ldots,n\}\setminus\{i\},1\le i\le n} \det\left(J_{C_1,\mathbf{m}}\right).\end{aligned}$$

4.3 Correctness and Complexity

Correctness of Theorem 4.1 follows from the proof of Basu et al. (2015), Lemma 3.19. Let $\mathcal{E}$ be the CAD computed by the algorithm from Section 4.2.1, using the McCallum projection operator, in which each cell has constant sign on the polynomials defining sets $V_1,\ldots,V_k$. By McCallum (1988), Theorems 2.2.3 and 2.2.4, each cell $C\subset V$ of $\mathcal{E}$ will be an analytic submanifold, since $\dim(V_i)\le 2, 1\le i\le k$. The algorithm then makes the CAD compatible with the determinants of matrices $J_{C_2,\mathbf{m}}$ and $J_{C_1,\mathbf{m}}$, defined in Equations (4.1) and (4.2) respectively. These are the critical points of projections of two- and one- dimensional smooth cylindrical cells $C\subset V_1,\ldots,V_k$. Thus, $\mathcal{D}$, the CAD output by the algorithm from Section 4.2.1 is compatible with the critical points of projections of the smooth two-dimensional locus of $V_1\cup\ldots\cup V_k$ onte one- and two-dimensional coordinate subspaces, as required in the proof of Basu et al. (2015), Lemma 3.19.

4.3.1 Complexity Analysis

Let us examine the complexity by supposing that the smooth stratification algorithm is used to find the smooth two-dimensional locus of $V=V_1\cup\ldots\cup V_k\subset\mathbb{R}^n$, defined by s polynomials having maximum degree d. By the result from Section 3.4, the smooth stratification step with emptiness checking has complexity

$$3^s\left(s(d+1)\right)^{O(n)^2}$$

The critical points of one- and two- dimensional strata are then computed. This is done by computing determinants of $(n-1)$- and $(n-2)$-dimensional Jacobi matrices, containing the $n-1$ and $n-2$ polynomial equations which define the strata of codimension 1 and 2 respectively. These polynomials have maximum degree $(d+1)^2$. In order to estimate the maximum degree of polynomials appearing in the Jacobi determinant of a $(k\times k)$-matrix, suppose that elementary row operations – swapping two rows, multiplying a row by a number and adding one row to another – will be used to convert the matrix into upper triangular form, then the elements on the main diagonal will be multiplied. Note that the

complexity of performing these row operations to transform the matrix into row echelon form has complexity $O\left(n^2\right)$. Elementary row operationns do not change the upper bound on degree. For the determinant, multiplying k polynomials of maximum degree d will result in a polynomial of maximum degree kd. Thus, the maximum degree of the determinant of the $((n-2)\times(n-2))$-matrix will be $(n-2)(d+1)^2$ and the $((n-1)\times(n-1))$-matrix will be $(n-1)(d+1)^2$. These are both bounded from above by

$$n(d+1)^2,$$

For the critical points of $\text{proj}_{\text{span}\{x_i\}}, 1 \leq i \leq n$, there will be n Jacobi determinants, and for $\text{proj}_{\text{span}\{x_i,x_j\}}, 1 \leq i < j \leq n$, there will be $(n-1)+(n-2)+\cdots+1 = \frac{1}{2}(n-1)n$, which is asymptotically the same as n^2. Accordirg to Section 3.4, the smooth stratification algorithm will produce at most $s^n(d+1)^2$. The bound on the number of strata of each codimension is not known, but the total number of strata is, of course, an upper bound. Therefore, there will be at most $n\left(s^n(d+1)^2\right)$ additional polynomials.

A classical CAD, having constant sign on the s input polynomials along with the $n\left(s^n(d+1)^2\right)$ Jacobi determinants, which have maximum degree $n(d+1)^2$, is then constructed. According to Proposition 2.8 (Basu et al. (2006), Algorithm 11.2), classical CAD has complexity

$$\left(s'd'\right)^{O(1)^n},$$

where s' is the number of input polynomials and d' their maximum degree. Using the bounds obtained from the application of smooth stratification, we get

$$\left(\left(s+n\left(s^n(d+1)^2\right)\right)n\left(d+1\right)^2\right)^{O(1)^n},$$

Which can be simplified to

$$\left(ns^n\left(d+1\right)\right)^{O(1)^n}.$$

Since the expression $n^{O(1)^n}$ can be simplified to $2^{O(1)^n}$ we get an upper complexity bound, for constructing a quasi-affine CAD, of

$$\left(s(d+1)\right)^{O(1)^n}.$$

By Proposition 2.8, this is also an upper bound on the number of cells, number of polynomials defining the CAD and their maximum degree.

Adding the stratification step, we get a total complexity of

$$3^s\left(s(d+1)\right)^{O(n)^2}+\left(s(d+1)\right)^{O(1)^n},$$

which is bounded from above by the doubly exponential CAD computation, leading to a complexity upper bound of

$$\left(s(d+1)\right)^{O(1)^n}.$$

Chapter 5

Monotone Cells

So far, we have constructed an **F**-invariant CAD $\mathcal{D}$ of $\mathbb{R}^n$, compatible with a family of bounded semialgebraic sets $V_1, \ldots, V_k$ of dimension at most two, in which every cell $C \subset V_1 \cup \ldots \cup V_k$ is a smooth manifold and the graph of a quasi-affine map. We now return to (Basu et al., 2015, Theorem 3.20) and discuss how to obtain monotone cells, starting with this quasi-affine decomposition.

We first present a motivating example, of a 2-dimensional cylindrical cell which is quasi-affine but not monotone.

Example 5.1. (Basu et al., 2015, Example 3.2)

Let

$$\Delta := \{(x, y) \in \mathbb{R}^2 \mid 0 < x < 1, y > 0, x + y < 1\} \subset \mathbb{R}^2$$

and

$$\varphi(x, y) = x^2 + y^2.$$

Let $Y \subset \mathbb{R}^3$ be the graph of function ϕ on Δ. Observe that Y is a cylindrical cell and $\varphi|_\Delta$ is a quasi-affine map, since it is strictly increasing in both x and y. However, $y \cap \{z = 3/4\}$ is not connected, hence $\phi|_\Delta$ is not a monotone function and Y is not a monotone cell.

Definition 5.1. Let $\mathcal{D}$ be a CAD of $\mathbb{R}^n$ and suppose that $\mathbf{c} = (c_1, \ldots, c_k)$ is a 0-dimensional section cell in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^k$. By (Basu et al., 2015, Remark 3.8), the set of all cells of $\mathcal{D}$ contained in the affine coordinate subspace

$$\{x_1 = c_1, \ldots, x_k = c_k\}$$

forms a cylindrical decomposition of $\mathbb{R}^{n-k}$. This decomposition will be called the *sub-decomposition* of $\mathcal{D}$ above c, and $\mathbf{c}$ the *base cell* of the sub-decomposition (or sub-CAD).

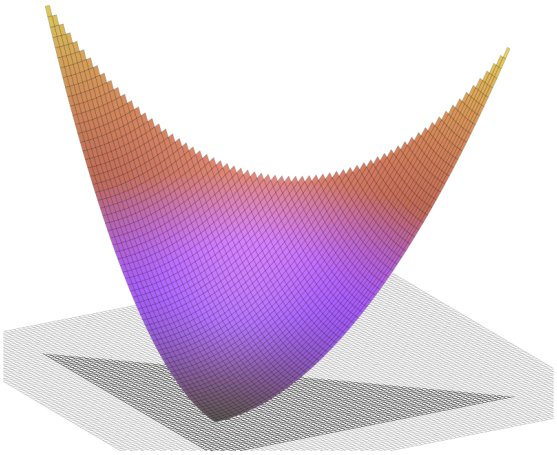

Figure 5.1: Plot of the 2-dimensional quasi-affine cylindrical cell Y defined in Example
refexm:monotone-cells,

The construction of monotone cells is based on the following result due to Basu et al. (2015).

Proposition 5.1. *(Basu et al., 2015, Lemma 3.18)*

Let $X \subset \mathbb{R}^2$ be an open, bounded set and $f : X \to \mathbb{R}$ be a quasi-affine function. Then there is a cylindrical decomposition of $\mathbb{R}^2$ compatible with X, obtained by intersecting X with straight lines

$$\{x_1 = c\}$$

and half-planes

$$\{x_1 < c\}, \{x_1 > c\}$$

such that the restriction $f|_B$ for every cell $B \subset X$ is a monotone function.

We now summarise the construction presented by Basu et al. (2015) in the proof of Theorem 3.20. Their proof proceeds by induction on n.

- The base case, $n = 1$ is straightforward, since (0) and (1)-cells are already monotone.
- When $n > 1$, consider each two-dimensional cell C of $\mathcal{D}$ and let $C'' := \text{proj}_{\mathbb{R}^1}(C)$

- If $\dim(C'') = 0$, apply the inductive hypothesis to the sub-decomposition of $\mathcal{D}$ above C''.
- Otherwise, C has index $\{1, i_2, \ldots, i_n\} \in \{0,1\}^n$. Let α be the smallest among $\{2, \ldots, n\}$ such that $i_\alpha = 1$. Since $\dim(C) = 2$, only i_1 and i_α are equal to 1, while all other elements of the index are equal to 0. By Basu et al. (2015), Lemma 3.3, $\text{proj}_{\text{span}\{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n\}}(C)$, for all $i \in \{2, \ldots, \alpha-1, \alpha+1, \ldots, n\}$, is a cylindrical cell and C is the graph of a continuous function from this cell to $\text{span}\{x_i\}$.

  Basu et al. (2015), Lemma 3.19, which provides the construction of $\mathcal{D}$ in the previous section, gives us a more useful representation of C. Let $X := \text{proj}_{\text{span}\{x_1,x_\alpha\}}(C)$ be a 2-dimensional sector cell. C can be viewed as the graph of a quasi-affine map

  $$\mathbf{f} = (f_1, \ldots, f_{n-2}) : X \to \text{span}\{x_2, \ldots, x_{\alpha-1}, x_{\alpha+1}, \ldots, x_n\}.$$

  By (Basu et al., 2015, Lemma 3.18), there exists a refinement of the CAD of $\text{span}\{x_1, x_\alpha\}$, obtained by intersecting X with straight lines and half-planes of the kind

  $$\{x_1 < c\}, \{x_1 = c\}, \{x_1 > c\}$$

  where $c \in \mathbb{R}$, such that X is a union of monotone sector cells B and $f_j|_B$ is a monotone function. By (Basu et al., 2015, Lemma 3.11), refinements of this kind preserve the cylindrical structure of $\mathcal{D}$ and do not destroy existing monotone cells. In the semialgebraic case, we will see that the refinement points c will always be algebraic numbers, which makes implementation using computer algebra systems possible.

In the following sections, we will apply the construction given in the proof of Basu et al. (2015), Theorems 3.20 and 3.18 to develop an algorithm for finding the set of refinement points in the semialgebraic case, starting with the cad $\mathcal{D}$ and its set of input polynomials $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$. The following will be proved.

Theorem 5.1. *Let*

$$F_1, \ldots, F_k$$

be quantifier-free Boolean formulas such that $F_i, 1 \leq i \leq k$ defines a bounded semialgebraic set $V_i \subset \mathbb{R}^n$ such that $\dim(V_i) \leq 2$. Let $\mathbf{F} = \{f_1, \ldots, f_s\}$ be a set of polynomials in $\mathbb{Z}[x_1, \ldots, x_n]$ of maximum degree d and $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$ compatible with each $V_1, \ldots, V_k$ and such that each cell $C \subset V_1 \cup \ldots \cup V_k$ is a smooth manifold and the graph of a quasi-affine map.

Then there is an algorithm, taking $\mathcal{D}$ as input, which produces a refinement $\mathcal{D}'$ of $\mathcal{D}$ monotone with respect to each $V_1, \ldots, V_k$. This algorithm has complexity

$$(s\,(d+1))^{O(1)^n},$$

which is also an upper bound on the number of cells in the CAD, number of polynomials and their degrees.

5.1 Two-dimensional monotone sector cells

Let C be an $(i_1, \ldots, i_n)$-cell such that $i_1 = i_\alpha = 1$ and all other elements of the index are equal to 0. The proof of Proposition 5.1 proceeds in two stages: first, $X \subset \text{span}\{x_1, x_\alpha\}$ is refined such that each $Y \subset X$ is a monotone 2-dimensional sector cell. Consider the intersection $X \cap \{x_1 = c\}$ for all $c \in \mathbb{R}$. This intersection is a finite union of pairwise disjoint intervals. Let $\mathcal{I}(c)$ be the family of these intervals associated with the point c. Define

$$\gamma := \{(x_1, x_\alpha) \in \text{span}\{x_1, x_\alpha\} \mid x_\alpha \text{ is an endpoint of an interval in } \mathcal{I}(x_1)\}.$$

Although the cell X under consideration is a 2-dimensional cylindrical cell in $\text{span}\{x_1, x_\alpha\}$, we actually need to refine the 2-dimensional cylindrical sector cell $C' := \text{proj}_{\mathbb{R}^\alpha}(C)$ in the cylindrical decomposition induced by $\mathcal{D}$ on $\mathbb{R}^\alpha$. Thus, we need a way to translate between C' and X.

Corollary 5.1. *(to Proposition 2.4 (Basu et al. (2015), Lemma 3.3))*

Let $C \subset \mathbb{R}^\alpha$ be a $(1, 0, \ldots, 0, 1)$-cell, $X := \text{proj}_{\text{span}\{x_1, x_\alpha\}}(C)$ and $X' := \text{proj}_{\mathbb{R}^1}(C)$. Suppose we have the family $\mathcal{I}(c), c \in X'$ and γ be the set of one-dimensional curves associated with C. Then

1. *X is a $(1, 1)$-cell.*
2. *The projection map $\text{proj}_{\text{span}\{x_1, x_\alpha\}}|_C$ is injective.*
3. *For each $c \in X'$, the family $\mathcal{I}(c)$ contains a single open interval.*
4. *γ consists of zero, one or two disjoint curve intervals which coincide with the top and bottom of X. Furthermore, the connected components of γ can be defined as the graphs of continuous definable functions from X' to $\mathbb{R}$, using the functions defining the cell C.*

Corollary 5.1 follows from Basu et al. (2015), Lemma 3.3 and the property that C is the graph of a quasi-affine map. A proof is presented below, which explicitly defines the functions which send points in X' to points in each of the connected components of γ.

Proof. Property (1) is proved by repeatedly applying the projection defined in Proposition 2.4 (Basu et al. (2015), Lemma 3.3) to C to obtain X. Property (2) follows immediately from the fact that X is 2-dimensional and C is the graph of a quasi-affine map.

Since X is a 2-dimensional sector cell, by definition it is the semialgebraic set

$$\{(x_1, t) \in \mathbb{R}^2 \mid x_1 \in X', \varphi(x_1) < t < \psi(x_1)\}$$

where $\varphi, \psi : X' \to \mathbb{R}$ are continuous definable functions such that $\varphi(x_1) < \psi(x_1)$ for all $x_1 \in X'$. Note that X may not be bounded from below (resp. above). In

this case, φ (resp. ψ) are omitted from the definition of X, as in Definition 2.20. It is clear that $\mathcal{I}(c)$ contains a single open interval $(\varphi(c), \psi(c))$, proving (3). It follows that γ, which consists of the endpoints of intervals in $\mathcal{I}(c)$, consists of one-dimensional curve intervals which are the graphs of φ and ψ (if they exist).

We now prove (4) by defining φ and ψ using the definition of cylindrical cells. Since $C' := \text{proj}_{\mathbb{R}^{\alpha-1}}(C)$ is a $(1, \ldots, 0, 0)$-cell, it is the graph of a continuous definable map $\mathbf{h} = (h_1, \ldots, h_{\alpha-2}) : X' \to \mathbb{R}^{\alpha-2}$ and

$$C = \{(\mathbf{x}, t) \mid \mathbf{x} \in C', f(\mathbf{x}) < t < g(\mathbf{x})\}$$

where $f, g : \mathbb{R}^{\alpha-1} \to \mathbb{R}$ are continuous definable functions such that $f(\mathbf{x}) < g(\mathbf{x})$ for all $\mathbf{x} \in C'$. Again, if not bounded from below (resp. above), f (resp. g) are omitted.

Finally, we write C in terms of $x_1 \in X'$, f, g and $\mathbf{h}$, we obtain the functions

$$\varphi : C'' \to \mathbb{R} \qquad \psi : C'' \to \mathbb{R} \tag{5.1}$$

$$\varphi(x_1) = f(x_1, \mathbf{h}(x_1)) \qquad \psi(x_1) = g(x_1, \mathbf{h}(x_1)) \tag{5.2}$$

□

The refinement is achieved by finding a set of real numbers

$$(c_1, \ldots, c_t)$$

such that, for each $1 \leq i < t$,

$$X \cap \{c_i < x_1 < c_{i+1}\}$$

is a monotone cell. By (Basu et al., 2013, Theorem 1.7), this can be achieved by ensuring that

$$\gamma \cap \{c_i < x_1 < c_{i+1}\}$$

contains only monotone curve intervals. [Basu et al. (2015, Theorem 3.18).

Applying Corollary 5.1 to C, we obtain continuous definable functions $\varphi, \psi : X' \to \mathbb{R}$ defining the bottom, X_B, and top, X_T, of X respectively. Let

$$(c_1, \ldots, c_t)$$

such that the graphs of $\varphi|_{\{c_i < x_1 < c_{i+1}\}}$ and $\psi|_{\{c_i < x_1 < c_{i+1}\}}$ for all $1 \leq i < t$ are monotone curve intervals. By Basu et al. (2015), Theorem 1.7, $X \cap \{c_i < x_1 < c_{i+1}\}$ for all $1 \leq i < t$ is a semi-monotone set (Basu et al., 2015, proof of Lemma 3.18). For the graphs of φ and ψ to be monotone curve intervals, we require that φ and ψ are either strictly increasing in, strictly decreasing in or independent of x_1. Thus, $(c_1, \ldots, c_t)$ are the critical points of φ and ψ. Note that if X is not bounded from below, then φ does not exist and does not have a graph. In this case, it is clear that no refinement of X_B is required.

5.1.1 Finding the critical points of functions phi and psi

We now discuss how to find these critical points starting with the polynomials defining cells in $\mathcal{D}$.

Let $\mathcal{D}$ be a CAD of $\mathbb{R}^n$ compatible with each set $V_1, \ldots, V_k \subset \mathbb{R}^n$ such that every 2-dimensional cell contained in $V = V_1 \cup \ldots \cup V_k$ is the graph of a quasi-affine map. Let

$$\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n), \tag{5.3}$$

where $\mathcal{A}_k \subset \mathbb{Z}[x_1, \ldots, x_k], 1 \leq k \leq n$, be the projection polynomials defining $\mathcal{D}$. I.e., polynomials in $\mathcal{A}_k$ have constant sign on every cell in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^k$.

Let C be a $(1, 0, \ldots, 0, 1)$-cell of the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^\alpha$ such that ${\text{proj}_{\mathbb{R}^\alpha}}^{-1}(C)$ is a 2-dimensional cell of $\mathcal{D}$ contained in V. We now describe how to find the numbers $(c_1, \ldots, c_t)$ such that $C \cap \{c_i < x_1 < c_{i+1}\}$ is monotone.

For convenience, we write $g(\mathbf{x}) = 0$ for some $g \in A_k$ and $\mathbf{x} \in \mathbb{R}^n$, such that $k < n$, if $g(\text{proj}_{\mathbb{R}^k}(\mathbf{x})) = 0$. Consider the $(1, 0, \ldots, 0)$-cell

$$C' := \text{proj}_{\mathbb{R}^{\alpha-1}}(C)$$

of the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{\alpha-1}$.

Observe that

$$\text{proj}_{\mathbb{R}^1}(C) = (a, b) \subset \mathbb{R}$$

and, since C' is a section cell, there exist polynomials

$$g_2 \in \mathcal{A}_2, \ldots, g_{\alpha-1} \in \mathcal{A}_{\alpha-1}$$

such that $g_i(\mathbf{x}) = 0$ for all $1 \leq i \leq \alpha - 1$ and $\mathbf{x} \in (C')$. Therefore, there exists a one-dimensional algebraic variety

$$V' := \{g_2(\mathbf{x}) = 0, \ldots, g_{\alpha-1}(\mathbf{x}) = 0\}$$

such that $C' \subset V'$.

Now consider the top of C, denoted by C_T. By definition, if C is not bounded from above, then C_T does not exist. Otherwise, C_T is the graph of a continuous definable function $h : C' \to \mathbb{R}$. In the first case, there is nothing to do. In the second case, C_T can be written as

$$\{\mathbf{x} \in \mathbb{R}^\alpha \mid a < x_1 < b, g_2(\mathbf{x}) = 0, \ldots, g_{\alpha-1}(\mathbf{x}) = 0, x_\alpha = h(\mathbf{x})\}, \tag{5.4}$$

where $\mathbf{x}$ is shorthand for $(x_1, \ldots, x_\alpha)$. By the construction of $\mathcal{D}$ (e.g., using Corollary 2.4) C_T is C^∞ smooth, therefore h is a differentiable function everywhere in C'.

This representation lends itself naturally to the problem of optimisation with constraints – Lagrange multipliers.

Proposition 5.2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ and $g_1, \ldots, g_k : \mathbb{R}^n \to \mathbb{R}$ be continuous functions with continuous first partial derivatives. Local maxima and minima of f, subject to the constraints $g_1 = 0, \ldots, g_k = 0$ can be found by computing the critical points of the function*

$$\mathcal{L}(\mathbf{x}, \lambda_1, \ldots, \lambda_k) = f(\mathbf{x}) - \lambda_1 g_1(\mathbf{x}) - \cdots - \lambda_k g_k(\mathbf{x}),$$

where $\mathbf{x} \in \mathbb{R}^n$ and $(\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^k$ are new variables.

Critical points can be found by solving the system of equations

$$\frac{\partial f}{\partial x_i} - \lambda_1 \frac{\partial g_1}{\partial x_i} - \cdots - \lambda_k \frac{\partial g_k}{\partial x_i} = 0, \tag{5.5}$$

where $1 \leq i \leq n-1$,

for $\lambda_1, \ldots, \lambda_k$.

Remark. Using variables $(x_1, \ldots, x_{\alpha-1})$, constraints $g_2, \ldots, g_{\alpha-1}$ and function h from Equation (5.4), the system from Equation (5.5) can be written in matrix form as follows.

$$\begin{pmatrix} \frac{\partial g_2}{\partial x_1} & \cdots & \frac{\partial g_{\alpha-1}}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial g_2}{\partial x_{\alpha-1}} & \cdots & \frac{\partial g_{\alpha-1}}{\partial x_{\alpha-1}} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_{\alpha-2} \end{pmatrix} = \begin{pmatrix} \frac{\partial h}{\partial x_1} \\ \vdots \\ \frac{\partial h}{\partial x_{\alpha-1}} \end{pmatrix}$$

It is clear that, if a solution exists, then the last column $\left(\frac{\partial h}{\partial x_1}, \ldots, \frac{\partial h}{\partial x_{\alpha-1}}\right)^T$ will be a linear combination of the first $\alpha - 1$ columns (in $g_2, \ldots, g_{\alpha-1}$). Hence, optimal values for $(x_1, \ldots, x_{\alpha-1})$ exist and satisfy the equation

$$\det \begin{pmatrix} \frac{\partial g_2}{\partial x_1} & \cdots & \frac{\partial g_{\alpha-1}}{\partial x_1} & \frac{\partial h}{\partial x_1} \\ \vdots & & & \vdots \\ \frac{\partial g_2}{\partial x_{\alpha-1}} & \cdots & \frac{\partial g_{\alpha-1}}{\partial x_{\alpha-1}} & \frac{\partial h}{\partial x_{\alpha-1}} \end{pmatrix} = 0. \tag{5.6}$$

Equation (5.6) gives a direct formula to solve Equation (5.5) for $x_1, \ldots, x_{\alpha-1}$. Refinement points $(c_1, \ldots, c_t)$ are the x_1-coordinates of these solutions.

Since C_T is a section cell, there exists a polynomial $g_\alpha \in \mathcal{A}_\alpha$ such that

$$C_T := \{\mathbf{x} \in \mathbb{R}^\alpha \mid \mathrm{proj}_{\mathbb{R}^{\alpha-1}}(\mathbf{x}) \in C', g_\alpha(\mathbf{x}) = 0\}.$$

Thus, there exists an algebraic variety

$$V := \{g_2(\mathbf{x}) = 0, \ldots, g_{\alpha-1}(\mathbf{x}) = 0, g_\alpha(\mathbf{x}) = 0\} \tag{5.7}$$

such that $C_T \subset V$, where $g_2, \ldots, g_{\alpha-1}$ are the same polynomials appearing in Equation (5.4).

Note that $g_\alpha \in \mathbb{Z}[x_1, \ldots, x_\alpha]$ – it is multivariate a polynomial – while h from Equation (5.6) is expected to be a continuous differentiable function from $\mathbb{R}^{\alpha-1} \to \mathbb{R}$. In order to use the above construction, we have to be careful only to consider points at which g_α is differentiable. In other words, we need to compute derivatives of the implicit function g_α:

$$\frac{dx_\alpha}{dx_i} = \frac{\partial g_\alpha / \partial x_i}{\partial g_\alpha / \partial x_\alpha} \tag{5.8}$$

for $1 \leq i \leq \alpha - 1$. Let J_T be the determinant of the matrix of partial derivatives from Equation (5.6) with each $\partial h/\partial x_i, 1 \leq i \leq \alpha - 1$ replaced with dx_α/dx_i from Equation (5.8). Observe that the denominator of $\partial h/\partial x_i$, for all $1 \leq i \leq \alpha - 1$, is equal to

$$d_T := \partial g_\alpha / \partial x_\alpha,$$

so the last column of the matrix, in terms of g_α is actually

$$\frac{1}{d_T} \left(\partial g_\alpha / x_1, \ldots, \partial g_\alpha / x_{\alpha-1}\right)^T.$$

This results in the determinant

$$J_T = \frac{1}{d_T} J'_T,$$

where J'_T is the matrix from Equation (5.6) with the $\partial h/\partial x_i$ replaced with $\partial g_\alpha / \partial x_i$ for $1 \leq i \leq \alpha - 1$. I.e.,

$$J'_T = \det \begin{pmatrix} \frac{\partial g_2}{\partial x_1} & \cdots & \frac{\partial g_{\alpha-1}}{\partial x_1} & \frac{\partial g_\alpha}{\partial x_1} \\ \vdots & & & \vdots \\ \frac{\partial g_2}{\partial x_{\alpha-1}} & \cdots & \frac{\partial g_{\alpha-1}}{\partial x_{\alpha-1}} & \frac{\partial g_\alpha}{\partial x_{\alpha-1}} \end{pmatrix} = 0. \tag{5.9}$$

Observe that J_T is a rational function: a fraction in which the numerator and denominator are both polynomials. $\mathbf{x} \in \mathbb{R}^{\alpha-1}$ is a solution of this rational function if and only if $J'_T(\mathbf{x}) = 0$ and $d_T(\mathbf{x}) \neq 0$. Thus, solutions of the system

$$\{J'_T = 0, d_T \neq 0\}$$

should be found. This is clearly a semialgebraic set. Also note that, when $\deg(d_T) = 0$, d_T is a constant and therefore has no solutions. Thus, the condition $d_T = 0$ can be dropped in this case.

Repeating this process to obtain J'_B and d_B, if sector cell C is bounded from below, we have two systems of polynomial equations

$$\{g_2, \ldots, g_{\alpha-1}, J'_B\}, \tag{5.10}$$

$$\{g_2, \ldots, g_{\alpha-1}, J'_T\} \tag{5.11}$$

and inequalities

$$\{s_1 g_{1,1} > 0, \ldots, s_\ell g_{1,\ell} > 0, d_B \neq 0, d_T \neq 0\} \tag{5.12}$$

where $\{g_{1,1}, \ldots, g_{1,\ell}\} \subset \mathbb{Z}[x_1]$ are polynomials from $\mathcal{A}_1$ and $s_1, \ldots, s_\ell \in \{-1, 1\}$ such that

$$X' = (a, b) = \{x \in \mathbb{R} \mid s_1 g_{1,1}(x) > 0, \ldots, s_\ell g_{1,\ell}(x) > 0\}.$$

We can easily find pairs $(s_j, g_{1,j}) \in \{-1, 1\} \times \mathcal{A}_1$ by considering the sign of the sample point c of X' on each polynomial $g_{1,j} \in \mathcal{A}_1$.

- If $g_{1,j} > 0$, then $s_j = 1$.
- If $g_{1.j}(c) < 0$, then $s_j = -1$.
- The final case, $g_{1,j}(c) = 0$, is impossible since an irreducible univariate polynomial cannot be zero over an interval.

Perhaps an easier way, if the information available, is to examine the signs of polynomials from $\mathcal{A}$ on the cell X'. Clearly, if the sign on g_j is positive, $s_i = 1$ and if the sign is negative $s_i = -1$.

For the refinement points $(c_1, \ldots, c_t)$, we want to find x_1-coordinates of the roots of polynomials from Equation (5.11) subject to the constraints from Equation (5.12).

CAD can, of course, be used to do this, since we are really solving the quantifier elimination problem

$$\exists x_2, \ldots, x_{\alpha-1}, x_\alpha \mid P(x_1, \ldots, x_\alpha)$$

where

$$\begin{aligned} P(x_1, \ldots, x_\alpha) = {} & s_1 g_{1,1}(x_1) > 0 \wedge \ldots \wedge s_\ell g_{1,\ell}(x_1) > 0 && (5.13)\\ & \wedge\ g_2(x_1, x_2) = 0 \wedge \ldots \wedge g_{\alpha-1}(x_1, \ldots, x_{\alpha-1}) = 0 && (5.14)\\ & \wedge\ d_B(x_1, \ldots, x_\alpha) \neq 0 \wedge d_T(x_1, \ldots, x_\alpha) \neq 0 && (5.15)\\ & \wedge\ (J'_B(x_1, \ldots, x_{\alpha-1}) = 0 \vee J'_T(x_1, \ldots, x_{\alpha-1}) = 0). && (5.16) \end{aligned}$$

Computing the projection onto $\mathbb{R}^1$ of polynomials $\{g_2, \ldots, g_{\alpha-1} J'_B, J'_T\}$ from Equations (5.14) and (5.16), a system

$$\mathcal{F} := \{f_1 = 0, \ldots, f_r = 0\} \subset \mathbb{Z}[x_1] \tag{5.17}$$

is obtained, whose real roots include the refinement points $(c_1, \ldots, c_t)$. Let $\mathcal{E} \subset \mathbb{Z}[x_1]$ be the set of polynomials obtained by computing the projection onto $\mathbb{R}^1$ of $\{d_B, d_T\}$ (from Equation (5.15)) and

$$\mathcal{G} := \{s_1 g_{1,1}, \ldots, s_\ell g_{1,\ell}\}$$

from Equation (5.13). $(c_1, \ldots, c_t)$ can be found by computing the real roots of $\mathcal{F}$ subject to the inequalities $\mathcal{G}$, then discarding those such that a polynomial in $\mathcal{E}$ is zero.

It is clear that $(c_1, \ldots, c_t)$ lie in an open interval $X' = (a, b) \subset \mathbb{R} \cup \{-\infty, \infty\}$, where a and b (if finite) are section cells. Thus, rather than using strict inequalities from Equation (5.13), we could find the roots of $\mathcal{F}$ in the interval (a, b).

Each real root will be represented by a pair

$$c_i := (m_i, J_i) \tag{5.18}$$

where c_i is the unique root of the polynomial $m_i \in \mathbb{Z}[x_1]$ in the (left-open right-closed) isolating interval J_i. Note that, being solutions of polynomials, every $(c_1, \ldots, c_t)$ contains only algebraic numbers and roots will appear in ascending order.

Since this procedure follows the construction described in Basu et al. (2015), Theorem 3.18, the following has been proved.

Lemma 5.1. *Let $\mathcal{D}$ be a sign-invariant CAD of $\mathbb{R}^n$ compatible with each set $V_1, \ldots, V_k \subset \mathbb{R}^n$ with $\dim(V_i) \leq 2$ and such that each cell $C \subset V_i$ is the graph of a quasi-affine map, for $1 \leq i \leq k$. Let*

$$\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n),$$

where $\mathcal{A}_k \subset \mathbb{Z}[x_1, \ldots, x_k], 1 \leq k \leq n$, be the projection polynomials defining $\mathcal{D}$. Let C be a $(1, 0, \ldots, 0, 1)$-cell of the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^\alpha$ such that ${\text{proj}_{\mathbb{R}^\alpha}}^{-1}(C)$ is a cell of $\mathcal{D}$ contained in $V_1, \ldots, V_k$.

Then there is an algorithm, taking $\mathcal{D}$ and $\mathcal{A}$ as input, which produces a set of algebraic numbers $(c_1, \ldots, c_t)$ such that $C \cap \{c_i < x_1 < c_{i+1}\}$ is monotone. Each $c_i, 1 \leq i < t$ is represented by a pair $(m_i, J_i) \in \mathbb{Z}[x_1] \times \mathbb{Q}$ (see Equation (5.18).

5.1.2 Working with sub-decompositions above a 0-cell

In the previous section, we considered $(1, 0, \ldots, 0, 1, 0, \ldots, 0)$-cells However, we will actually be working in a sub-decomposition $\mathcal{D}$ (of $\mathbb{R}^{k+n}$) above a 0-cell $\mathbf{c}$ of $\mathbb{R}^k$ ($k \geq 0$). Therefore the polynomials appearing in Equation (5.7) are elements of $\mathbb{Z}[y_1, \ldots, y_k][x_1, \ldots, x_n]$. From a polynomial $f \in \mathbb{Z}[y_1, \ldots, y_k][x_1, \ldots, x_n]$, a polynomial $g := f(\mathbf{c}) \in \mathbb{A}[x_1, \ldots, x_n]$ can be obtained by evaluating f at $\mathbf{c}$. Evaluated polynomials g define the sub-cad of $\mathbb{R}^n$ above $\mathbf{c}$ because $\mathbf{c}$ is a 0-cell.

Note that, since $\mathbf{c}$ is an algebraic number, g has algebraic coefficients. Since roots of g are algebraic numbers, there exists a polynomial $h \in \mathbb{Z}[x_1, \ldots, x_n]$ such that $g(\mathbf{x}) = 0$ if and only if $h(\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathbb{R}^n$. A method for obtaining the polynomial h with integer coefficients is now described.

1. If f is independent of all $y_1, \ldots, y_k$, there is nothing to do, just set $h := f$. Otherwise, f depends on at least one of $y_1, \ldots, y_k$.

2. If $\mathbf{c}$ is rational, then $g = f(\mathbf{c})$ has coefficients in $\mathbb{Q}$. There exists a non-zero $a \in \mathbb{Z}$ such that $h := ag$ has integer coefficients and the same roots as g.

3. Otherwise, $\mathbf{c}$ is not rational. Then $f(\mathbf{c})$ has coefficients in $\mathbb{A}$. There exists an algorithm, which takes g as input and returns a polynomial $h \in \mathbb{Z}[x_1, \ldots, x_n]$ such that $g(\mathbf{x}) = 0$ if and only if $h(\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathbb{R}^n$.

Remark. Let $f \in \mathbb{Q}[x_1, \ldots, x_n]$ be a polynomial such that

$$f(x_1, \ldots, x_n) = \frac{a_1}{b_1} m_1 + \cdots + \frac{a_k}{b_k} m_k$$

where $m_i = x_1^{d_{i,1}} \cdots x_n^{d_{i,n}}$ for $1 \leq i \leq k$ is a monomial in $x_1, \ldots, x_n$.

Let $M := \operatorname{lcm}(b_1, \ldots, b_k)$ be the least common multiple of denominators and construct

$$g(x_1, \ldots, x_n) = Mf(x_1, \ldots, x_n) = \frac{Ma_1}{b_1} m_1 + \cdots + \frac{Ma_k}{b_k} m_k.$$

Each coefficient in g will be an integer number since M is divisible by all $b_1, \ldots, b_k$ and polynomials f and g have the same roots.

Definition 5.2. Let K be a field and L a finite (algebraic) extension of K. The field L can be thought of as a finite dimensional vector space over K.

Consider f and m be polynomials in $L[x_1, \ldots, x_n]$, viewed as polynomials in $K[x_1, \ldots, x_n, x_\alpha]$. The norm, in $L[x_1, \ldots, x_n]$ of f can be found by computing the resultant of f and m.

Remark. Let $f \in \mathbb{A}[x_1, \ldots, x_n]$ be a polynomial such that

$$f(x_1, \ldots, x_n) = a_1 m_1 + \cdots + a_k m_k$$

where $m_i = x_1^{d_{i,1}} \cdots x_n^{d_{i,n}}$ is a monomial in variables $x_1, \ldots, x_n$ and a_i is an element of the algebraic extension of the rational numbers $\mathbb{Q}(\alpha)$ for $1 \leq i \leq k$.

Let $m \in \mathbb{Z}[\alpha]$ be the minimal polynomial for $\mathbb{Q}(\alpha)$. f can be written as a polynomial in $\mathbb{Q}[x_1, \ldots, x_n, x_\alpha]$ and m can be viewed as a polynomial in $\mathbb{Q}[x_1, \ldots, x_n, x_\alpha]$ which is independent of all $x_1, \ldots, x_n$. Compute

$$g := \operatorname{Res}_\alpha(f, m)$$

to obtain a polynomial in $\mathbb{Q}[x_1, \ldots, x_n]$. The property that $f(\mathbf{x}) = 0$ if and only if $g(\mathbf{x}) = 0$ follows from a property of resultants:

> Let $I = \langle f, m \rangle$ be the ideal generated by polynomials $f, m \in K[x_1, \ldots, x_n][\alpha]$, where K is an algebraically closed field. If at least one of f and m is monic, then
>
> $$\operatorname{Res}_\alpha(f, m) \in I \cap R.$$

It follows that $\mathbf{x} \in K^n$ is a common zero of the elements of $I \cap R$ if and only if it is a zero of $\operatorname{Res}_\alpha(f, m)$.

5.2 Two-dimensional monotone sections

Let $X := \text{proj}_{\text{span}\{x_1, x_\alpha\}}(C)$ and $(c_1, \ldots, c_r)$ be the refinement points computed in the previous section such that

$$Y := \{c_i < x_1 < c_{x+1}\} \cap X,$$

for some $1 \leq i \leq r$, is a two-dimensional monotone sector cell. Let us return to (Basu et al., 2015, Theorem 3.18) to further refine Y such that every two-dimensional section cell is a monotone cell.

The procedure for computing the refinement of Y such that f_j is a monotone function, described in the proof of Basu et al. (2015), Theorem 3.18, proceeds as follows. Let $f_j : Y \to \text{span}\{x_j\}$ be a component of the quasi-affine map $\mathbf{f} : X \to \mathbb{R}$. Since $\mathbf{f}$ is quasi-affine, each component of $\mathbf{f}$ is quasi-affine and the restriction of component f_j to Y is quasi-affine since $Y \subset X$. Hence, f_j is either strictly increasing in, strictly decreasing in, or independent of each variable x_1, x_α. In addition, the restriction $f|_{Y \cap \{x_\alpha = c\}}$ for any $c \in \mathbb{R}$ is either strictly increasing in, strictly decreasing in, or independent of x_α. We will compute a refinement of Y such that $f_j|_Y$ is monotone, each $B \subset Y$ in the refinement. As before, only refinements of the kind

$$\{x_1 < c\}, \{x_1 = c\}, \{x_1 > c\},$$

$c \in \mathbb{R}$, which preserves both the cylindrical structure and existing monotone cells will be performed.

5.2.1 Case 1: $2 \leq j \leq \alpha - 1$

Lemma 5.2. *Let $Y \subset \mathbb{R}^n$ be the graph of a quasi-affine map*

$$\mathbf{f} : (a, b) \to \mathbb{R}^{n-1}$$

such that $(a, b) \subset \mathbb{R}$. Then Y is a monotone cell.

Proof. Assume for contradiction that Y is not monotone. Then intersection $Z := Y \cap \{x_i = c\}$, for some $1 \leq i \leq n$ and $c \in \mathbb{R}$, is not connected. Z cannot contain an interval, since Y is connected, one-dimensional and the graph of a continuous map. Hence, Z consists of more than one isolated point, which implies that $\text{proj}_{\text{span}\{i\}}|_Y$ is not injective, but its image is one-dimensional. This contradicts the assumption that Y is the graph of a quasi-affine map. □

Suppose that $2 \leq j \leq \alpha - 1$. Since

$$\text{proj}_{\text{span}\{x_j, x_\alpha\}}(C)$$

is one-dimensional, connected and the graph of a quasi-affine map, it follows from Lemma 5.2 that $\text{proj}_{\text{span}\{x_j, x_\alpha\}}(C)$ is a monotone cell and $f_j|_Y$ is already a monotone function. No refinement is needed.

5.2.2 Case 2: $\alpha + 1 \leq j \leq n$

Now suppose that $\alpha+1 \leq j \leq n$. By (Basu et al., 2013, Theorem 3), if functions

$$\inf_{x_\alpha} f_j : \text{span}\{x_1\} \to \text{span}\{x_j\} \text{ and } \sup_{x_\alpha} f_j : \text{span}\{x_1\} \to \text{span}\{x_j\}$$

are monotone, then f_j itself is monotone. Hence, we need to find refinement points

$$\{b_1, \ldots, b_r\} \subset (c_i, c_{i+1})$$

such that, for each $1 \leq \ell < r$, the restrictions of both $\inf_{x_\alpha} f_j$ and $\sup_{x_\alpha} f_j$ to $B := \{b_\ell < x_1 < b_{\ell+1}\}$ are monotone.

Consider

$$Z := \text{proj}_{\text{span}\{x_1, x_j\}}(C).$$

If $\dim(Z) < 2$, then it is clear that $\dim(Z) = 1$, because

$$Y' := \text{proj}_{\mathbb{R}^1}(C)$$

is one-dimensional. It follows that $\inf_{x_\alpha} f_j$ and $\sup_{x_\alpha} f_j$ coincide. Since Z is one-dimensional, connected and the graph of a quasi-affine map, Lemma 5.2 implies that f_j is already monotone and no refinement is needed.

On the other hand, if $\dim(Z) = 2$, since $\mathbf{f}$ is quasi-affine, $\text{proj}_{\text{span}\{x_1,x_j\}}|_C$ is injective. Let Z_T be the graph of $\sup_{x_\alpha} f_j$. Despite the notation, Z_T is not necesarily the top of Z. In particular, Z_T may have non-empty intersection with Z. Since semialgebraic sets are closed under intersection and projection, there exist points

$$\{b_1, \ldots, b_k\} \subset (c_i, c_{i+1})$$

such that for all $1 \leq i < k$

$$Z_T^{b_i, b_{i+1}} := Z_T \cap \{b_i < x_1 < b_{i+1}\}$$

is either a subset of Z or is disjoint from Z. First suppose that $Z_T^{b_i,b_{i+1}} \subset Z$ for some $1 \leq i < k$. In this case, Z is not bounded from above, so $\sup_{x_\alpha} f_j|_{\{b_i<x_1<b_{i+1}\}}$ is independent of x_1 and is therefore already monotone. On the other hand, suppose that $Z_T^{b_i,b_{i+1}} \cap Z = \emptyset$. In this case, $Z_T^{b_i,b_{i+1}} \subset \text{proj}_{\text{span}\{x_1,x_j\}}(C_T)$. If there exists a point $b \in (b_i, b_{i+1})$ such that $\text{proj}_{\text{span}\{x_1,x_j\}}(C_T) \cap \{x_1 = b\} \cap Z \neq \emptyset$, then it is clear that $C_T \cap \{b_i < x_1 < b_{i+1}\}$ is not the graph of a continuous function. I.e., $\text{proj}_{\text{span}\{x_1,x_\alpha,x_j\}}(C_T)$ contains an open interval above $(b, y) \in Y_T$. Since $\dim(C) = 2$, all these blow-up points (b, y) have dimension 0 and therefore there is a finite number of them. Repeating this process for $\alpha + 1 \leq j \leq n$, it follows that there exist points

$$\{b_{i,1}, \ldots, b_{i,k_i}\} \subset (b_i, b_{i+1})$$

such that, for all $1 \leq j < k_i$, $C_T \cap \{b_{i,j} < x_1 < b_{i,j+1}\}$ is the graph of a continuous map. In particular, $C_T \cap \{b_{i,j} < x_i < b_{i,j+1}\}$ is the graph of $\mathbf{f}|_{Y_T \cap \{b_{i,j}<x_1<b_{i,j+1}\}}$.

Apply the same argument to $\inf_{x_\alpha} f_j$ (for C_B) so that, C_T and C_B, if non-empty, are split into one-dimensional continuous curves which are graphs of continuous functions, and the intervals above 0-dimensional blow-up points. Once this property has been satisfied, we are left with a similar construction of optimisation with constraints as in Section 5.1.1.

5.2.3 Critical points of the top and bottom of C

We now discuss how to perform this refinement on the CAD structure. Let us assume that $C_T \neq \emptyset$, otherwise no refinement is needed. We must construct subsets $W \subset Y$ such that $f_j|_{W_T}$ is a smooth, continuous function. Let

$$\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$$

be the projection factor set as defined in Equation (5.3) and

$$C' := \text{proj}_{\mathbb{R}^{j-1}}(C), C^j := \text{proj}_{\mathbb{R}^j}(C),$$

Begin with $j := \alpha + 1$ so that C' is a cylindrical sector cell and there exists an algebraic variety V containing polynomials $g_2, \ldots, g_{j-1}$ with each $g_i \in \mathcal{A}_i, 2 \leq i \leq j-1$, defined in Equation (5.4), such that $C'_T \subset V$. By definition

$$C^j_T = \text{cl}\left(C^j\right) \cap (C'_T \times \mathbb{R})$$

and there exists a polynomial $g_j \in \mathcal{A}_j$ such that $g_j(\mathbf{x}) = 0$ for all $\mathbf{x} \in C^j_T$. Since C^j_T is either empty or the graph of a continuous function everywhere except, possibly, over a finite number of blow-up points, we need to find x_1-coordinates of points in C'_T such that g_j vanishes over an open interval. Since C^j_T is one-dimensional, one way to achieve this is by computing a smooth stratification of C^j_T. If the projection of a one-dimensional stratum onto x_1 is a 0-dimensional point $b \in \mathbb{R}$, then b is the x_1-coordinate of a blow-up point.

Alternatively, since the refinement points form a 0-dimensional semialgebraic subset of $\mathbb{R}$, this set can be represented as a first-order Boolean formula

$$\exists x_2, \ldots, x_{j-1} \exists_\infty x_j ((x_1, \ldots, x_{j-1}) \in C'_T, g_j(x_1, \ldots, g_j) = 0)$$

where $\exists_\infty x$ is a special quantifier defined in `QEPCAD` meaning "there exist infinitely many values of x". Using $\exists_\infty$, some of the costly computations with algebraic numbers can be avoided, thereby improving efficiency in practice (Brown and Hong, 2004). Theoretically speaking, (Basu et al., 2006, Algorithm 14.21) provides a singly exponential upper bound for quantifier elimination.

Eliminating the $j-1$ quantifiers, we obtain the desired points $(b_{i,1}, \ldots, b_{i,k_i})$, where each $b_{i,j}$ is the x_1-coordinate of a blow-up point of g_j. Now we can assume that $C^j_T \cap \{b_{i,j} < x_1 < b_{i,j+1}\}$ is the graph of a continuous function. More precisely, the implicit function g_j can be considered as a function from $C'_T \cap \{b_{i,j} < x_1 < b_{i,j+1}\}$ to $\mathbb{R}$. We have a very similar construction as in Section

5.1.1. That is, polynomials $g_2, \ldots, g_{j-1}$ will provide the constraints in the first $j-2$ columns of the matrix in Equation (5.6), while $\partial h/\partial x_i, 1 \leq i \leq j-1$ in the last column should be replaced with $\partial g_j/\partial x_i$ Denote this polynomial by J_T, and repeat with C_B^j if appropriate. Inequalities $s_1 g_{1,1} > 0, \ldots, s_1 g_{1,\ell} > 0$ (from Equation (5.13)), $d_T \neq 0$ and d_B (from Equation (5.15)) are obtained and solved for x_1 in a very similar way as in Section 5.1.1. We obtain a set of refinement points

$$\{b_{j,1}, \ldots, b_{j,r_j}\} \subset (c_i, c_{i+1})$$

such that each of C_T^j and C_B^j is either empty or the graph of a continuous, monotone function.

The process is completed by induction on j. Suppose that the refinement has been performed for $\alpha + 1 \leq j$. If $j = n$ we are done, and C is now monotone. Otherwise, we proceed to $j+1$. By the induction hypothesis, we know that $C_T^j \cap \{b_{j-1,i} < x_1 < b_{j-1,i+1}\}, 1 \leq i < r_j$ is either empty or the graph of a continuous (monotone) function. If $C_T^j = \emptyset$, there is nothing to do as C_T^{j+1} is empty, too, and the process will terminate. Otherwise, the new constraint $g_j \in \mathcal{A}_j$ is the polynomial defining C_T^j and $g_j \in \mathcal{A}_j$ can be found such that $g_j(\mathbf{x}) = 0$ for all $\mathbf{x} \in C_T^{j+1}$. Perform the same computation described above, to refine C_T^{j+1} such that one-dimensional components are graphs of continuous, monotone functions. Repeat with C_B^{j+1}. As before, a set of refinement points

$$\{b_{j+1,1}, \ldots, b_{j+1,r_{j+1}}\}$$

is obtained.

The union of sets of refinement points

$$\{b_{\alpha+1,1}, \ldots, b_{\alpha+1,r_{\alpha+1}}\} \cup \ldots \cup \{b_{n,1}, \ldots, b_{n,r_n}\}$$

forms the refinement of C into two-dimensional monotone cells.

Altogether, we have proved the following.

Lemma 5.3. *Let $\mathcal{D}$ be a CAD of $\mathbb{R}^n$ compatible with each set $V_1, \ldots, V_k \subset \mathbb{R}^n$ with $\dim(V_i) \leq 2$ and such that each cell $C \subset V_i$ is the graph of a quasi-affine map, for $1 \leq i \leq k$. Let*

$$\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n),$$

where $\mathcal{A}_k \subset \mathbb{Z}[x_1, \ldots, x_k], 1 \leq k \leq n$, be the projection polynomials defining $\mathcal{D}$. Let C be a 2-dimensional section cell of the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^j$ such that $\mathrm{proj}_{\mathbb{R}^j}{}^{-1}(C)$ is a cell of $\mathcal{D}$ contained in $V_1, \ldots, V_k$ and $\mathrm{proj}_{\mathbb{R}^{j-1}}(C)$ is monotone.

Then there is an algorithm, taking $\mathcal{D}$ and $\mathcal{A}$ as input, which produces a set of algebraic numbers $(c_1, \ldots, c_r)$ such that $C \cap \{c_i < x_1 < c_{i+1}\}$ is monotone. Each $c_i, 1 \leq i \leq r$ is represented by a pair $(m_i, J_i) \in \mathbb{Z}[x_1] \times \mathbb{Q}$ (see Equation (5.18).

Lemmas 5.1 and 5.3 can also be applied to two-dimensional $(0, \ldots, 0, 1, i_{k+1}, \ldots, i_n)$-cells by working in a sub-cad of $\mathbb{R}^{n-k+1}$ above a zero-cell, as described in Section 5.1.2.

5.3 Implementation Details

We now present psuedocode for the monotone part of the algorithm.

5.3.1 Algorithm

Algorithm 3. *Monotone CAD*

Input:

$$(\mathcal{E}, \mathcal{A})$$

- $\mathcal{E}$*: a cylindrical algebraic decomposition of* $\mathbb{R}^{k+n}$*, such that each cell is the graph of a quasi-affine map. Each cell* C *of* $\mathcal{E}$ *has a truth-value (*true *or* false *attached.*
- $\mathcal{A} = \mathcal{A}_1, \ldots, \mathcal{A}_{k+n}$ *is the family of projection polynomials where each* $\mathcal{A}_i \subset \mathbb{Z}[x_1, \ldots, x_i]$.

Output:

$$\mathcal{R} := \{\mathcal{R}_{\mathbf{b}} = \{c_1, \ldots, c_t\} \subset \mathbb{A} \mid \mathbf{b} \in \mathbb{R}^{k-1}\}$$

where $\mathbf{b} \in \mathbb{R}^{k-1}$ *is a* $(0, \ldots, 0)$*-cell in the decomposition induced by* $\mathcal{E}$ *on* $\mathbb{R}^k$*, such that, for each* $\mathcal{R}_{\mathbf{b}}$*, intersecting the sub-cad* $\mathcal{D}$ *above* $\mathbf{b}$ *with straight lines and half-planes of the kind* $\{x_1 < c_i\}, \{x_1 = c_i\}, \{x_1 > c_i\}, 1 \leq i \leq t$ *results in a refinement of* $\mathcal{D}$ *such that each true cell of* $\mathcal{D}$ *with dimension at most* 2 *is monotone.*

First, define a subroutine

$$\mathrm{sub}_{\mathbf{y}}(g) \mid g \in \mathbb{Z}[y_1, \ldots, y_m, x_1, \ldots, x_n], \mathbf{y} \in \mathbb{A}^m, n > 0,$$

which computes $h := g(\mathbf{y}) \in \mathbb{A}[x_1, \ldots, x_m]$ *and returns the normalised polynomial of* h*, in* $\mathbb{Z}[x_{m+1}, \ldots, x_n]$ *(see Section 5.1.2).*

It will be convenient to overload this operator and write $\mathrm{sub}_{\mathbf{y}}(L)$*, where* L *is a set (or list) of polynomials to mean* $\{\mathrm{sub}_{\mathbf{y}}(g) \mid g \in L\}$.

Now define a structure to store refinement polynomials

$$\mathcal{F} := \{\mathcal{F}_{\mathbf{b}} := \emptyset \mid \mathbf{b} \in \mathbb{R}^{k-1}, k > 0 \text{ is a } (0, \ldots, 0)\text{-cell in the decomposition induced by } \mathcal{E} \text{ on } \mathbb{R}^{k-1}\}$$

and a similar structure to store refinement points

$$\mathcal{R} := \{\mathcal{R}_{\mathbf{b}} := \emptyset \mid \mathbf{b} \in \mathbb{R}^{k-1}, k > 0 \text{ is a } (0, \ldots, 0)\text{-cell in the decomposition induced by } \mathcal{E} \text{ on } \mathbb{R}^{k-1}\}$$

(Note: for each $(0, \ldots, 0)$-cell $\mathbf{b}$, $\mathcal{F}_{\mathbf{b}}$ is a set of univariate polynomials, $\mathcal{R}_{\mathbf{b}}$ is the set of roots of polynomials $\mathcal{F}_{\mathbf{b}}$. When $k = 1$, then the CAD of $\mathbb{R}^0$ consisting of

the unique point of $\mathbb{R}^0$ is used – practically speaking, we work directly with the CAD $\mathcal{E}$.)

Let $\mathbf{b}$ *be a* $(0,\ldots,0)$*-cell of the decomposition induced by* $\mathcal{E}$ *on* $\mathbb{R}^{k-1}, k>0$*. Let* $\mathcal{D}$ *be the sub-decomposition of* $\mathbb{R}^n$ *of* $\mathcal{E}$ *above* $\mathbf{b}$*.*

Consider each cell C *of* $\mathcal{D}$ *with truth value* true*. Let* C *have index*

$$(0,\ldots,0,1,i_{k+1},\ldots,i_{k+n})$$

in $\mathcal{E}$*. If* $i_{k+1}+\cdots+i_n>2$*, then* $\dim(C)>2$ *and the algorithm will fail. Otherwise, let* α *be the smallest among* $x+1,\ldots,k+n$ *such that* $i_{k+1}=\ldots=i_{\alpha-1}=0$ *and* $i_\alpha=1$*. Then:*

- *Let*
  - $\mathbf{b}:=\mathrm{proj}_{\mathbb{R}^{k-1}}(C)$,

    (Note: observe that $\mathbf{b}\in A^{k-1}$. If $k=1$, then $\mathbf{b}$ is the unique cell, $\mathbf{0}$, in the CAD of $\mathbb{R}^0$. In this case, it is clear that $\mathrm{sub}_{\mathbf{0}}(g)\equiv g$.)
  - $C''\subset\mathbb{A}:=\mathrm{proj}_{\mathbb{R}^k}(C)$ *((1)-cell), and let* (C''_B,C''_T) *be* (0)*-cells which are the endpoints of* C''*,*
  - $\mathcal{G}:=\mathrm{sub}_{\mathbf{b}}\{g_i \mid k+1\le i\le\alpha-1, g_i\in\mathcal{A}_i, g_i(\mathbf{x})=0\ \forall\mathbf{x}\in\mathrm{proj}_{\mathbb{R}^{\alpha-1}}(C)\}$,
  - $C':=\mathrm{proj}_{\mathbb{R}^\alpha}(C)$.
- C'_T *and* C'_B *be the top and bottom, respectively,* C'*.*

*(*Note: If C' is not bounded from above (resp. below) by a continuous definable function, i.e., there is no section above (resp. below) C', then let $C'_T:=\emptyset$ (resp C'_B) and skip any computations involving C'_T (resp C'_B).)

- *Let* $j=\alpha$ *and do:*
  - ***(*)** For* $\delta\in\{B,T\}$*, if* C_δ *is not empty, do:*
    - *Let* $\mathrm{sub}_{\mathbf{b}}(\mathrm{g_j})\in\mathbb{Z}[\mathrm{x_{k+1},\ldots,x_j}]$ *be such that* $g_\alpha(\mathbf{x})=0$ *for all* $\mathbf{x}\in C'_\delta$*.*
    - *Compute*

$$f_\delta:=\det\begin{pmatrix}\frac{\partial g_{k+1}}{\partial x_k} & \cdots & \frac{\partial g_{j-1}}{\partial x_k} & \frac{\partial g_j}{\partial x_k}\\ \vdots & & & \vdots\\ \frac{\partial g_{k+1}}{\partial x_{j-1}} & \cdots & \frac{\partial g_{j-1}}{\partial x_{j-1}} & \frac{\partial g_j}{\partial x_{j-1}}\end{pmatrix},$$

      from Equation (5.6)*, using constraints* $g_{k+1},\ldots,g_{\alpha+1}$*, and discarding denominators* $\partial g_\alpha/\partial x_\alpha$ *of the total derivatives in the last column.*

- *Let*
$$\mathcal{H} := \{g_{k+1}, \ldots, g_{j-1}, f_B, f_T\}$$
be a set of polynomials and compute
$$\mathcal{F}_{\mathbf{b}} := \mathcal{F}_{\mathbf{b}} \cup \mathrm{proj}_{\mathbb{R}^1}(\mathcal{H}) \subset \mathbb{Z}[x_k].$$
- *If $j < n$, then*
  - *Add g_j to $\mathcal{G}$, i.e., let*
  $$\mathcal{G} := \mathcal{G} \cup \{g_j\}.$$
  - *Let $j := j + 1$.*
  - *Let*
  $$C' := \mathrm{proj}_{\mathbb{R}^j}(C)$$
  C' is a section cell, and its top and bottom may not be graphs of continuous functions. Use QE to compute

  ...

- *For each $(0, \ldots, 0)$-cell $\mathbf{b} \in \mathbb{A}^k$, consider*
$$\mathcal{F}_{\mathbf{b}} \in \mathcal{F} \in \mathbb{Z}[k],$$
compute
$$\mathcal{R}_{\mathbf{b}} := (c_1, \ldots, c_t)$$
by isolating the real roots of polynomials in $\mathcal{F}_{\mathbf{b}}$.
- *Return the family $\mathcal{R}$ of refinement points.*

5.4 Correctness and Complexity

The correctness of the algorithm from Theorem 5.1 follows from Basu et al. (2015), Theorem 3.20 and Lemma 3.19, and is proved in Lemmas 5.1 and 5.3. The algorithm described in Section 5.3.1 considers each 2-dimensional cell in a sub-cad $\mathcal{E}$ of $\mathbb{R}^n$ above a 0-dimensional cell $\mathbf{b} \in \mathbb{R}^{k-1}$. This is exactly the inductive step of Basu et al. (2015), Theorem 3.20, where an $(i_1, 0, \ldots, 0, i_\alpha, 0, \ldots, 0)$-cell, where $i_1 = i_\alpha = 1$, is considered. By Corollary 5.1 to C, the two-dimensional subset $X \subset \mathrm{span}\{x_1, x_\alpha\}$ is obtained. The first part of Basu et al. (2015), Lemma 3.18 describes how X should be refined into two-dimensional semi-monotone sets. The construction described in Section 4 explains how to obtain the set of refinement points $(c_1, \ldots, c_t) \subset \mathbb{A}$. C is the graph of a quasi-affine map $\mathbf{f} = (f_{i_1}, \ldots, f_{i_{n-2}})$ where each component $f_j : X \to \mathrm{span}\{x_j\}$ for $j \in \{1, \ldots, n\} \setminus \{1, \alpha\}$ is a quasi-affine function. The second part of Basu et al. (2015), Lemma 3.18 describes

how to refine X such that each function f_j is monotone. By Basu et al. (2015), Lemma 3.16, if every component f_i is monotone on a subset in the refinement of X, then the map $\mathbf{f}$ itself is monotone on that subset of X. The construction described in Section 5.2 describes how to find the refinement points to ensure that $\mathbf{f}$ is a monotone map. Thus, the construction in this section satisfies the properties required in the proof of Basu et al. (2015), Lemma 3.18 for C to be monotone.

5.4.1 Complexity Analysis

Let C be a $(0, \ldots, 0, i_j, 0, \ldots, 0, i_k, 0 \ldots, 0)$-cell, with $i_j = i_k = 1$. The Jacobi determinant, defined in Equation (5.9), is computed for all $k \leq \ell \leq n$. As discussed in Section 4.3.1, the determinant of a $(k \times k)$-matrix of polynomials with maximum degree d is a polynomial of maximum degree kd. Also from Section 4.3.1, the complexity of computing the CAD with quasi-affine cells is

$$(s(d+1))^{O(1)^n},$$

which, according to Proposition 2.8, is also an upper bound on the number of polynomials, their degree and the number of cells in this CAD. Thus, the degree of the polynomial defined in Equation (5.6) is at most $n \cdot (s(d+1))^{O(1)^n}$, which is asymptotically the same as $(s(d+1))^{O(1)^n}$.

We then compute the x_1-coordinates of the solutions to the 0-dimensional system of equations (defined in Equations (5.11)), subject to the inequalities (defined in Equation (5.12)). In the implementation, CAD projection is used to compute a family of polynomials $\mathcal{F}$, defined in Equation (5.17), such that the required refinement points contain their roots. As we are well aware, the number of polynomials and their degrees will be doubly exponential in n. However, a better bound on the number of refinement points can be obtained. Equation (5.11) defines two systems of $\alpha - 1$ equations in $\alpha - 1$ unknowns and both systems have a finite number of solutions. A generalisation of Bezout's Theorem (see Masser and Wüstholz (1983)) states that, over an algebraically closed field, if a system of n polynomial equations, having degrees $d_1, \ldots, d_n$, in n unknowns has a finite number of solutions, then this number, counted with multiplicities, is at most the product $d_1 \cdots d_n$. This will be an upper bound on the number of different real solutions to the system of polynomials. Since we only have an upper bound on the maximum degree d of polynomials appearing in the systems from Equation (5.11), then there are at most d^n distinct real solutions to the system. An even tighter bound can be obtained by considering the number of connected components of the algebraic set defined by this system of equations. Indeed, the number of connected components of a definable set is given by the 0-th Betti number, b_0, of the set. Gabrielov and Vorobjov (2009) Section 6.1 present some bounds on the total Betti number – sum of Betti numbers of all orders – $\mathtt{b}(S)$ of a semialgebraic set $S \subset \mathbb{R}^n$. For a semialgebraic set $S \subset \mathbb{R}^n$ defined by a conjunction of any number of polynomials having maximum degree d, $\mathtt{b}(S) \leq d(2d-1)^{n-1}$ (Gabrielov and Vorobjov, 2009, Section 6.1 (a)). This

bound is sufficient for our situation because the inequalities define a subset of these roots. Using the upper bound on the degrees of polynomials defining the CAD and the determinant of the matrix, along with the bound on the number of roots given by Bezout's theorem, we get a bound for the number of refinement points of

$$(s(d+1))^{n\cdot O(1)^n}.$$

This is asymptotically the same as

$$(s(d+1))^{O(1)^n}.$$

For the cell C, this process is repeated at most $2(n-2)$ times, and then repeated for each 2-dimensional cell in the CAD contained in the input set V. There are at most

$$(s(d+1))^{O(1)^n}$$

cells in total, which is obviously an upper bound on the cells of dimension 2. Thus, we get an upper bound on the number of refinement points of

$$\left((s(d+1))^{O(1)^n}\right)^2$$

which is asymptotically the same as

$$(s(d+1))^{O(1)^n}.$$

Thus, we obtain a bound on the total number of refinement points, which is asymptotically the same as the bound on the number of cells in the CAD containing quasi-affine cells. This gives us an upper bound on the complexity, number of cells, number of polynomials and degrees defining a CAD which is monotone with respect to all $V_1, \ldots, V_k$.

5.5 Computing the refinements

We now describe how to refine the CAD $\mathcal{D}$ to be compatible with the refinement points in $\mathcal{R}$.

For $0 \leq k \leq n$, consider each refinement point $\mathbf{c} = (c_1, \ldots, c_{k-1}, c_k) \in \mathcal{R}_k$. It is important to note that refinement points appear in ascending order. Let $\mathcal{D}'$ be the sub-cad of $\mathbb{R}^{n-k}$ above $(c_1, \ldots, c_k)$, which is a $(0, \ldots, 0)$-cell of the CAD induced by $\mathcal{D}$ on $\mathbb{R}^k$ and let $\mathcal{D}''$ be the CAD induced by $\mathcal{D}'$ on $\mathbb{R}^{k+1}$. We want to refine $\mathcal{D}''$ such that $\mathbf{c}$ is a new (0)-cell. We need only to consider coordinate $k+1$ of $\mathbf{c}$. Denote it by c_{k+1}. For each (0)-cell b_{k+1} of $\mathcal{D}''$, compute $s := \text{sign}(b_{k+1} - c_{k+1})$. If $s \leq 0$, consider the next (0)-cell in $\mathcal{D}''$. Otherwise, $s > 0$ and there is a (1)-cell C in $\mathcal{D}''$ such that $c_{k+1} \in C$. This is the cell we need to refine. Make two new copies of C, so that we have three identical cells C_1, C_2, C_3. Recall that, to each cell belongs a sample point and possibly some other information. E.g., in QEPCAD, each cell has a set of signs of projection factors, which should remain

unchanged, and a positional index which describes where in the CAD the cell can be found. Let C_1 have positional index $(j_1, \ldots, j_k, j_{k+1})$. New cells C_2 and C_3 should have positional indices $(j_1, \ldots, j_k, j_{k+1} + 1$ and $(j_1, \ldots, j_k, j_{k+1} + 2)$ respectively. For each existing cell with positional index $(l_1, \ldots, l_k, l_{k+1})$ with $l_k + 1 > j_{k+1}$, the index should be updated to $(l_1, \ldots, l_k, l_{k+1} + 2)$. We now need to update the sample points. The sample point of the section cell C_2 should be $\mathbf{c}$. This might turn out to be the case by chance. In this case, there is nothing to be done with C_2 and the sample points of sector cells C_1 and C_3 are incorrect. Otherwise, $\mathbf{c}$ is either greater than the sample point of C_1, in which case the sample point of C_1 is already correct, or $\mathbf{c}$ is less than C_3, in which case the sample point of C_3 is already correct. I.e., in every case, exactly two of the sample points of the cells C_1, C_2 and C_3 must be updated. Let C be the open interval (a_1, a_2). If the sample point of C_1 needs to be updated, then set it to $(a_1 + c_{k+1})/2$ and if the sample point of C_3 needs updating, then set it to $(c_{k+1} + a_2)/2$. Observe that, if we have updated the sample point of a cell C, then the sample points of every cell which projects on C are now incorrect. Since polynomials are delineable over the original cell, then they will be delineable over the smaller cells in the refinement, so we simply need to run the sample point computation part of the lifting phase recursively on the stack above C. This involves evaluating each level $k + 1$ projection factor f at the new sample point $\mathbf{b}$ and finding the roots of the (possibly algebraic) univariate polynomial $f(\mathbf{b}, x_n)$.

Repeating this process with each refinement point in $\mathbf{R}_k$, we obtain the refinement described in (Basu et al., 2015, Lemma 3.11).

Chapter 6

Frontier Condition

Suppose we have a CAD $\mathcal{D}'$ monotone with respect to each bounded semialgebraic set $V_1, \ldots, V_k$, such that $\dim(V_i) \leq 2, 1 \leq i \leq k$. The algorithm from Section 5.3.1 produces such a CAD, which is also sign-invariant on a set of polynomials $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$. It is clear that $\mathbf{F}$ should contain the polynomials in defining formulas for sets V_i, along with some additional polynomials which ensure that each cell $C \subset V_1 \cup \ldots \cup V_k$ is monotone. In this section, we describe how to refine $\mathcal{D}'$ in order that it satisfies the frontier condition. A refinement, $\mathcal{D}''$ of $\mathcal{D}'$ satisfies the frontier condition if $\mathrm{fr}\,(C)$ is a union of cells of $\mathcal{D}''$, for each cell $C \subset V_1 \cup \ldots \cup V_k$ of $\mathcal{D}''$ (Definition 2.27).

We first present a motivating example due to Davenport et al. (2020).

Example 6.1. (Davenport et al., 2020, Example 2.1)

Let $f := y^2 - x^2 z$ and observe that $f(0, 0, z) = 0, z \in \mathbb{R}$, i.e., f "blows up" above the origin. Let $\Delta := \{(x, y) \in \mathbb{R}^2 \mid 0 < x < 1, -x < y < x\} \subset \mathbb{R}^2$ and let

$$W := \{(x, y, z) \in \mathbb{R}^3 \mid (x, y) \in \Delta, f(x, y, z) = 0\}.$$

Consider $\mathrm{fr}\,(W)$, which can be expressed as the disjoint union of four basic semialgebraic sets:

$$\begin{aligned}
\mathrm{fr}\,(W) = &\{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, y = x, z = 1\} && (6.1)\\
&\cup \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, y = -x, z = 1\} && (6.2)\\
&\cup \{(x, y, z) \in \mathbb{R}^3 \mid x = 1, -1 < y < 1, z = y^2\} && (6.3)\\
&\cup \{(0, 0, z) \in \mathbb{R}^3 \mid 0 \leq z \leq 1\}. && (6.4)
\end{aligned}$$

Due to the blow-up point above the origin, $\mathrm{fr}\,(W)$ contains the subset $B := \{(0, 0, z) \in \mathbb{R}^3 \mid 0 \leq z \leq 1\}$ (from Equation (6.4)).

A CAD having constant sign on the polynomials

$$\{x, x-1, x+y, x-y, y^2-x^2z\}$$

defining W will not contain the blow-up subset B, because two additional polynomials, z and $z-1$, are needed to define it.

It is clear from Example 6.1 that a sign-invariant CAD may not always be sufficient to construct a CAD with frontier condition. Schwartz and Sharir (1983) proved that it is always possible to construct a sign-invariant CAD which satisfies the frontier condition if a linear rotation of coordinates is permitted. Davenport et al. (2020) also proved that a sign-invariant CAD will always satisfy the frontier condition if polynomials containing blow-up points are not allowed. In the following sections, we discuss what to do when a rotation of coordinates is not permitted and when blow-up points may be present.

Note that W, from Example 6.1, is a $(1,1,0)$-cell, but it is not monotone. In fact, W is not even the graph of a quasi-affine map, since $\text{proj}_{\text{span}\{x,z\}}(W)$ is two-dimensional, but the projection map $\text{proj}_{\text{span}\{x,z\}}|_W$ is not injective. The blow-up subset B, contained in $\text{fr}\,(W)$ means that W is not a topologically regular cell, since $\text{fr}\,(()W)$ is not homeomorphic to a circle. In the procedures described in the following sections, due to Basu et al. (2015) and Lazard (2010), the cells on which the frontier condition is to be satisfied must be topologically regular cells.

6.1 Construction of a CAD with frontier condition due to Basu et al. (2015)

Given the monotone CAD $\mathcal{D}'$ computed in the algorithm from Section 5.3.1, our final step, according to (Basu et al., 2015, Theorem 3.20) is to compute a refinement $\mathcal{D}''$ of $\mathcal{D}'$ such that the frontier $\text{fr}\,(C) = \text{cl}\,(C) \setminus C$ of every cell $C \subset V = V_1 \cup \ldots, \cup V_k$ is a union of cells of $\mathcal{D}''$.

The construction from Basu et al. (2015), Theorem 3.20 proceeds as follows. For each 2-dimensional cell $C \subset V$, consider the semialgebraic set $\text{fr}\,(C)$. Since C is monotone, we can assume that $\text{fr}\,(C)$ is homeomorphic to a circle. Construct a partition $\mathcal{U}$ of $\text{fr}\,(C)$ into monotone one-dimensional curves and points, such that $\mathcal{U}$ is compatible with all one-dimensional cells of $\mathcal{D}'$. Let $\mathbf{c} = (c_1, \ldots, c_n)$ be one of the one-dimensional components of $\mathcal{U}$. If c_1 is not a 0-dimensional cell of the decomposition induced by $\mathcal{D}'$ on $\mathbb{R}^1$, then perform a refinement of the kind $x_1 < c_1, x_1 = c_1, x_1 > c_1$. Recall that these refinements, according to (Basu et al., 2015, Lemma 3.11), preserve the cylindrical structure and monotone cells. Otherwise, c_1 is a 0-cell and we apply the same argument to the sub-CAD of $\mathcal{D}'$ above c_1.

Now let $T \in \mathcal{U}$ be one of the one-dimensional components. If T is not an existing 1-dimensional cell of $\mathcal{D}'$, then it is a subset of a 2-dimensional cell Z of $\mathcal{D}'$.

In fact, it divides Z into two cells $Z \setminus T$, both of which, according to (Basu et al., 2013, Theorem 11), are 2-dimensional monotone cylindrical cells. Hence, replacing Z by T and the two monotone connected components of $Z \setminus T$ makes Z compatible with $\mathrm{fr}\,(C)$, while the refinement of Z consists of only monotone cells.

Repeating this process for each 0- and 1-dimensional component of $\mathcal{U}$, we obtain a refinement of $\mathcal{D}'$ compatible with $\mathrm{fr}\,(C)$. This completes the construction of a cylindrical decomposition $\mathcal{D}''$ satisfying the frontier condition and monotone with respect to each bounded semialgebraic set of dimension at most two $V_1, \ldots, V_k$.

The most computationally difficult part of implementing this construction is computing $\mathrm{fr}\,(C)$.

Lemma 6.1. *Let $S \subset \mathbb{R}^n$ be a semialgebraic set defined by a quantifier-free Boolean formula F containing s different polynomials in $\mathbb{R}[x_1, \ldots, x_n]$ having maximum degree d. There is an algorithm, taking F as input, which represents the semialgebraic set $\mathrm{fr}\,(S)$ by a quantifier-free Boolean formula F' with complexity*

$$(sd)^{O(n^2)}.$$

This is also an upper bound on the number of polynomials in F' and their degrees.

Lemma 8.1 is proved and discussed in more detail in Section 8.2.1. The "obvious" way of computing the frontier involves solving a quantifier elimination problem with two quantifier alternations. Using an efficient algorithm, e.g., (Basu et al., 2006, Algorithm 14.21), this procedure has complexity singly exponential in the number of variables. ∂C is represented by a quantifier-free Boolean formula F.

Once $\mathrm{fr}\,(()C)$ has been computed, we first construct the family $\mathcal{U}$ of one-dimensional curve intervals T and the points between them $\mathbf{c}$. By Lemma 5.2, one-dimensional quasi-affine cells are always monotone. Thus, we may apply the method described in Section 4 to obtain Jacobi matrices whose determinants are zero at the critical points of projections onto one and two-dimensional coordinate subspaces. If CAD with the McCallum projection operator is used to solve the QE problem, then each smooth 1-dimensional cell D in $\mathrm{fr}\,(C)$ will be defined by $n-1$ polynomials. We refine $\mathrm{fr}\,(C)$ to be compatible with the critical points of $\mathrm{proj}_{\mathrm{span}\{x_i\}}(D), 1 \leq i \leq n$ defined by $\det(J_{\{1,\ldots,n\}\setminus\{i\}}) = 0$.

A further refinement of each one-dimensional component T of $\mathcal{U}$ to be compatible with existing cells of $\mathcal{D}'$ may also be required. For each one-dimensional component $T \in \mathcal{U}$ and each one-dimensional cell D of $\mathcal{D}'$, consider $E := T \cap D$ such that $E \neq \emptyset$. If $E \subset D$, then T should be split into one-dimensional components E and $\int(T \setminus E)$ and the zero-dimensional endpoints of E. This makes $\mathcal{U}$ compatible with all cells of the CAD $\mathcal{D}'$.

Consider each $T \in \mathcal{U}$ which is a subset of a 2-dimensional cell Z. The refinement can be performed exactly as described by Basu et al. (2015), Theorem 3.20. I.e.,

partition Z into $Z_1 \cup T \cup Z_2$ where $Z_{1,B} = Z_B$ and $Z_{1,T} = T$ and $Z_{1,B} = Z_T$ and $Z_{1,T} = Z_T$.

Now consider the zero-dimensional components $\mathbf{c} = (c_1, \ldots, c_n)$ of $\mathcal{U}$. Refinements of $\mathcal{D}$ to be compatible with $\mathbf{c}$ again proceed as described by Basu et al. (2015). I.e., Consider $C' := \text{proj}_{\mathbb{R}^k}(C)$ such that C' is zero-dimensional and equal to $(c_1, \ldots, c_k)$, but $\text{proj}_{\mathbb{R}^{k+1}}(C)$ is one-dimensional. Perform a refinement of the sub-cad above $(c_1, \ldots, c_k)$ by intersecting it with $\{x_k < c_k\}, \{x_k = c_k\}, \{x_k > c_k\}$ – the refinements defined in Basu et al. (2015), Lemma 3.11. Repeat until $(c_1, \ldots, c_n)$ is a cell in the refinement.

Let $\mathcal{D}''$ be the refinement of $\mathcal{D}'$, compatible with $\text{fr}(C)$ for all 2-dimensional cells C of $\mathcal{D}'$. This CAD is compatible with, and monotone with respect to all $V_1, \ldots, V_k$, and the frontier of each cell $C \subset V_1 \cup \ldots V_k$ of $\mathcal{D}''$ is the union of some other cells of $\mathcal{D}''$ of smaller dimension. $\mathcal{D}''$ satisfies the frontier condition, so the construction is complete.

We have proved that the following algorithm exists, and will discuss its complexity later.

Theorem 6.1. *Let $\mathbf{F} \subset \mathbb{Z}[x_1, \ldots, x_n]$ be a set of s polynomials with maximum degree d. Let $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$, compatible with some bounded definable semialgebraic sets $V_1, \ldots, V_k$, which $\dim(V_i) \leq 2$ and such that each cell $C \subset V_i$ is monotone.*

Then there is an algorithm, taking $\mathcal{D}$ as input, which produces a refinement $\mathcal{D}'$ of $\mathcal{D}$ such that each cell $C \subset V_1 \cup \ldots \cup V_k$ of $\mathcal{D}'$ satisfies the frontier condition. This algorithm has complexity

$$(s\,(d+1))^{2^{O(n)}},$$

which is also an upper bound on the number of cells in the CAD, number of polynomials and their degrees.

6.2 Lazard's method for dimension not greater than 3

The method described in Section 6.1 is relatively straightforward as it relies on techniques we have used before (obtaining quasi-affine subsets and performing refinements). However, we may want to find an alternative method, so as to avoid the QE required to compute the frontiers of 2-dimensional cylindrical cells. In the following section, we will explore an alternative method to obtain the frontier condition, based on an algebraic construction described by Lazard (2010).

Lazard (2010) presents an algorithm for constructing an $\mathbf{F}$-invariant cylindrical algebraic decomposition of $\mathbb{R}^n, n \leq 3$ such that every cell is a topologically regular

cell and the frontier condition is satisfied. Lazard refers to a decomposition having these properties as a "strong" decomposition.

Definition 6.1. (Lazard, 2010, Definition 2.7)

1. A cylindrical cell C is boundary smooth if the intersection with C of every small open ball centred on the boundary of C is connected
2. A k-dimensional cylindrical cell is well-bordered if $\mathrm{fr}\,(C)$ is the closure of some $k-1$-dimensional cylindrical cells.
3. A cylindrical cell C is boundary coherent if $\mathrm{fr}\,(C)$ is a union of cells.

Remark. (Lazard, 2010, Definition 2.7)

These properties can be extended from cells to the whole CAD. I.e., a CAD $\mathcal{D}$ is called well-bordered (resp. boundary coherent, boundary smooth) if every cell in $\mathcal{D}$ is well-bordered (resp. boundary coherent, boundary smooth).

Remark. If a cylindrical cell satisfies Property 2 of Definition 6.1 (boundary smooth) then it is a topologically regular cell.

Property 3 of Definition 6.1 (boundary coherent) is the frontier condition (see Definition 2.27). Davenport et al. (2020) call this property closure finiteness.

Definition 6.2. (Lazard, 2010, Definition 2.7)

A CAD is called "strong" if every cell is boundary smooth, well-bordered and boundary coherent.

Definition 6.3. (Lazard, 2010, Definition 4.1)

Let $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$ and $\mathcal{D}'$ be the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{n-1}$. A cell C' of $\mathcal{D}'$ is called "bad" if there is a polynomial $f \in \mathbf{F} \subset \mathbb{Q}[x_1, \ldots, x_n]$ such that f is primitive and has positive degree in x_n and $f(\mathbf{x})$, for all $\mathbf{x} \in C'$ vanishes over an open interval. Every point $\mathbf{x}$ in the bad cell C' is a blow-up point of the polynomial f (see Definition 2.22).

According to Lazard (2010), Corollary 4.7, in dimension ≤ 2, there is nothing to do as Collins' algorithm already produces these "strong" decompositions. For a cylindrical decomposition of $\mathbb{R}^3$, the process proceeds in three steps:

1. Obtain well-bordered cells
2. Obtain boundary smooth cells
3. "Lift above bad points" to satisfy the frontier condition.

First, we must ensure that every cell is well-bordered. The following result shows that this is already the case for sign-invariant decompositions of $\mathbb{R}^3$.

Corollary 6.1. *(Lazard, 2010, Corollary 5.4)*

Every $\mathbf{F}$-invariant CAD of $\mathbb{R}^n, n \leq 3$ is well-bordered.

Corollary 6.1 follows from Lazard (2010), Lemma 5.1 and Theorem 5.3. The former asserts that every 0- and 1-dimensional cell in a CAD of arbitrary dimension is well-bordered. The latter asserts that any cell in an **F**-invariant CAD $\mathcal{D}$ of $\mathbb{R}^3$, such that the decomposition $\mathcal{D}'$ induced by $\mathcal{D}$ on $\mathbb{R}^2$ is strong, is well-bordered. Note that $\mathcal{D}'$ is a $\text{proj}_{\mathbb{R}^2}(\mathbf{F})$-invariant CAD of $\mathbb{R}^2$ and is always strong by Lazard (2010), Corollary 4.7. Lazard (2010), Proposition 5.2 depends on the following result.

Proposition 6.1. *(Lazard, 2010, Proposition 5,2)*

Let $\mathcal{D}$ be a CAD of $\mathbb{R}^n$ and $\mathcal{D}'$ the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{n-1}$. Let C be a cell of $\mathcal{D}$ and $C' := \text{proj}_{\mathbb{R}^{n-1}}(C)$ such that C' is well-bordered. If $p \in \text{fr}(C')$, then $I := (p \times \mathbb{R}) \cap \text{fr}(C)$ is connected. I is therefore a point or a closed segment, either bounded or unbounded. Furthermore, if C is a sector cell, then the set of points of I which do not belong to the boundaries of the section cells delineating C is either empty or an open interval.

Lazard (2010), Theorem 5.12 describes how to pass from a well-bordered, **F**-invariant CAD $\mathcal{D}$ of $\mathbb{R}^3$ to one in which every cell is boundary smooth. This is done by refining $\mathcal{D}$ such that it has constant sign on some new polynomials. This refinement contains only well-bordered, topologically regular cells. The construction of a CAD with monotone cells described in Section 5 also satisfies these properties, if we limit ourselves to $n \leq 3$. Indeed, the algorithm from Section 4.2.1 returns a CAD with constant sign on $\mathbf{F}'$, the set of input polynomials and the additional polynomials added to ensure that every cell is smooth and the graph of a quasi-affine map. Hence, every cell in this CAD is well-bordered. The algorithm for obtaining monotone cells, described in Section 5.3.1 requires us to compute refinements of the kind $\{x_1 < c\}, \{x_1 = c\}, \{x_1 > c\}$ of a (1)-cell C' in the sub-decomposition above a 0-dimensional cell. This refinement is also applied to every cell C in this sub-decomposition such that $C' := \text{proj}_{\mathbb{R}^1}(C)$. Thus, the refined CAD is sign invariant on $\mathbf{F}'$ and the cells C projecting on C' are sign-invariant with respect to polynomials $x_1 - c$, defining each refinement point c. Cells which were not refined are either already monotone or lie outside of the input sets $V_1, \ldots, V_k$. As a result, we are able to apply Lazard's method for lifting with bad points to the CAD produced by the algorithm from Section 5.3.1.

Lazard (2010), Section 5.3 describes how to pass from an **F**-invariant CAD of $\mathbb{R}^3$ such that every cell is well-bordered and boundary smooth to one satisfying the frontier condition. By Lazard (2010), Theorem 4.4, a cell C already satisfies the frontier condition if no points in its frontier project on a bad cell (see also Davenport et al. (2020)). Thus, only cells which fail to satisfy this property need to be considered. By Lazard (2010), Lemma 4.3, in $\mathbb{R}^3$, if a bad cell exists, then it has dimension 0.

Proposition 6.2. *(Lazard, 2010, Lemma 4.3)*

A sign-invariant CAD of $\mathbb{R}^2$ contains no bad cells.

For an $\mathbf{F} \subset \mathbb{Q}[x_1, x_2, x_3]$-invariant CAD $\mathcal{D}$ of $\mathbb{R}^3$, if the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^2$ contains a bad cell, then it is a $(0,0)$-cell.

This follows from the fact that, in $\mathbb{R}^n$, a blow-up point of a polynomial in $\mathbb{Q}[x_1, \ldots, x_n]$ in $\mathbb{R}^{n-1}$ has codimension at least two (see Remark aftor Definition 2.22).

Proposition 6.3. *(Lazard, 2010, Theorem 4.4)*

Let $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n$ and C be a cell of $\mathcal{D}$ with $C' := \text{proj}_{\mathbb{R}^{n-1}}(C)$. If the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{n-1}$ is strong and none of C' and any cell D' such that $D' \cap \text{fr}\,(C') \neq \emptyset$ is bad, C is well-bordered, boundary smooth and $\text{fr}\,(C)$ is a union of some cells of $\mathcal{D}$.

Proof. This result follows from the following fact about roots of multivariate polynomials: given a polynomial $f \in \mathbb{Q}[x_1, \ldots, x_{n-1}][x_n]$, i.e., considered as a univariate polynomial with polynomial coefficients, its complex roots depend continuously on the parameters in the region of the parameter space where the polynomial does not vanish identically. In fact, this is only true in the projective closure of $\mathbb{C}$ as some roots may pass through the point at infinity if the leading coefficient is zero.

First suppose that C is a section cell. I.e., it is a root of some polynomial f and $\text{fr}\,(C)$ is defined as the limit of this root over the boundary of C'. Thus $\text{proj}_{\mathbb{R}^{n-1}}$ is a continuous, injective map from $\text{fr}\,(C)$ to $\text{fr}\,(C')$. This follows from the assumption that the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{n-1}$ is strong and that if the root defining the section C is infinite over some cell $C'' \subset \text{fr}\,(C')$, then it is also infinite over $\text{fr}\,(C'')$.

Now suppose that C is a sector cell. If nonempty, the top C_T and bottom C_B of C are section cells. We complete the proof by describing $W := \text{fr}\,(C) \setminus (C_T \cup C_B)$. We will only consider the case where $C \cap \{\mathbf{x} \times \mathbb{R}\}$ is an open interval bounded from below by a point in C_B and above by a point in C_T.

Let $\dim(C) = d + 1$. $\text{fr}\,(C)$ consists of

- the cells in $\text{fr}\,(C_B)$ and $\text{fr}\,(C_T)$, which have dimension $< d$ and are contained in $\text{fr}\,(C') \times \mathbb{R}$,
- the intervals (which may be empty) bounded by the points in $\text{fr}\,(C_B)$ and $\text{fr}\,(C_T)$, contained in $C' \times \mathbb{R}$. Since C_B and C_T are section cells, they are roots of polynomials f and g and, if finite, the points in $\text{fr}\,(C_B)$ and $\text{fr}\,(C_T)$ are also roots of f and g (respectively). Therefore, these intervals are contained in some cells of $\mathcal{D}$. More precisely, the cells with dimension d contained in $\text{fr}\,(C') \times \mathbb{R}$ are exactly those lying over a cell $C'' \subset \text{fr}\,(C')$ with dimension $d - 1$.

A cell of dimension $< d$ either belongs to $\mathrm{fr}\,(C_B)$ or $\mathrm{fr}\,(C_T)$, or is a sector over a cell $D \subset \mathrm{fr}\,(C')$ having dimension less than $d-1$. As C' is well-bordered, D belongs to the boundary of some cell $E \subset \mathrm{fr}\,(C')$ having dimension $d-1$. Thus, the sector above D is contained in the boundary of the sector above E, which has dimension d

This proves that, if the decomposition induced by D on $\mathbb{R}^{n-1}$ is strong, then C is well-bordered and also satisfies the frontier condition. In addition, this proves that C is a topologically regular cell as C and $\mathrm{fr}\,(C)$ are either homeomorphic to C' and $\mathrm{fr}\,(C')$, if C is a sector cell, or to an open cylinder, either bounded or unbounded, and its boundary. It is clear that an open cylinder contained in $C' \times \mathbb{R}$, where C' is topologically regular, is also topologically regular.

The other cases are deduced by omitting the cells defined from C_B and C_T as appropriate. □

Proposition 6.4. *(Lazard, 2010, Proposition 5.13)*

Let $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD of $\mathbb{R}^3$ and C be a well-bordered, boundary smooth cell of $\mathcal{D}$. Suppose that D is a cell of $\mathcal{D}$ such that $B := D \cap \mathrm{fr}\,(C) \neq \emptyset$ and $B \neq D$. Then either

1. *C is a 1-dimensional section cell whose endpoint, B, projects on a bad cell, or*
2. *there exists a cell E of $\mathcal{D}$ such that $\dim(E) < \dim(C)$ and $\emptyset \neq D \cap \mathrm{fr}\,(E) \neq D$.*

Proof. By the proof of Lazard (2010), Theorem 4.4, D will always be a $(0,0,1)$-cell in the cylinder above a bad cell D'.

Consider each case for C by cell index.

- $(1,1,1)$-cell: since C is well-bordered, its boundary is the closure of some 2-dimensional cells of $\mathcal{D}$ and B is contained in the union of the boundaries of the 2-cells. Thus, case 2 applies and we can choose an appropriate 2-dimensional cell in $\mathrm{fr}\,(C)$ for E.
- $(1,0,1)$- or $(0,1,1)$-cell, we write $(i,j,1)$-cell where $i+j=1$: observe that C (if bounded) is delineated by some $(i,j,0)$-cells (one-dimensional curve intervals). Take B to be the one-dimensional interval delineated by the endpoints of these cells. Since $B \neq C$, at least one of these endpoints is not a $(0,0,0)$-cell of $\mathcal{D}$. Case 2 applies, taking as E the delineating $(i,j,0)$-cell.
- $(1,0,0)$- or $(0,1,0)$-cells: consider the previous case, letting $C = E$.
- $(1,1,0)$-cell: these are necessarily the root of a polynomial $f \in \mathbb{Q}[x_1, x_2, x_3]$ which vanishes identically on D', a $(0,0)$ cell in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^2$. Let V_r be a small neighbourhood (with radius $r > 0$) centred around D' and consider V_r', the cylinder above the boundary of V_r. Since C is a monotone cell, the maximum and minimum values of x_3 in this

intersection are reached on the boundary of C, for sufficiently small r. The limits, as r tends to 0, of this maximum and minimum clearly belong to the boundary of B, if these limits do not tend to infinity. Note that both the maximum and minimum cannot be infinite, since this would imply that either B is empty, or coincides with D. Therefore, the non-infinite limits are the endpoints of some 1-dimensional cell belonging to the boundary of C. Case 2 applies, taking E to be this 1-dimensional cell.

- $(0, 0, 1)$- or $(0, 0, 0)$-cells: do not satisfy the hypothesis, there is nothing to do.

□

The following result shows how $(0, 0, 1)$-cells which project on a bad point can be refined to obtain the frontier condition.

Proposition 6.5. *(Lazard, 2010, Proposition 5.14)*

Let $f \in \mathbb{Q}[x_1, x_2, x_3]$ be an irreducible polynomial of the kind

$$f = g_d x_3^d + \ldots + g_1 x_3 + g_0$$

with each $g_i \in \mathbb{Q}[x_1, x_2]$, and

$$\mathbf{p} = (p_1, p_2)$$

be a common root of $g_d, \ldots, g_1, g_0$. Let $g \in \mathbb{Q}[x_1, x_2]$ be an irreducible polynomial having $\mathbf{p}$ as a root. The limits of the common roots of f and g as $(x_1, x_2) \to \mathbf{p}, (x_1, x_2) \neq \mathbf{p}$ may be computed as solutions of a zero-dimensional polynomial system.

Proof. Since f is irreducible, it has at least two coefficients, $g_i, g_j \in \mathbb{Q}[x_1, x_2]$, whose GCD is a constant. Therefore, any ideal generated by these coefficients has dimension zero. Any ideal containing these coefficients also contains another element, h, which is not a multiple of g.

The saturation by h of the ideal $\langle f, g\rangle$ is defined as the ideal

$$S := \langle f, g, 1 - zh\rangle \cap \mathbb{Q}[x_1, x_2, x_3].$$

The irreducible components of the zero set of S are those of the zero set of $\langle f, g\rangle$ at which h is not identically zero. It follows that the vertical line $\{x_1 = p_1, x_2 = p_2\}$ which is part of the zero set of f ,g and h, is not contained in the zero set of S. Meanwhile, the zero set of S contains the limit points we are interested in finding.

Therefore, the ideal generated by S and the coefficients $g_0, \ldots, g_d$ of f is zero-dimensional and the limits of $\{f = 0, g = 0\}$ as $(x_1, x_2) \to \mathbf{p}$ can be computed as the zeros of S. □

The proof of Lazard (2010), Proposition 5.14 depends on taking Zariski closures. These are now defined and a relevant property presented, closely following Cox et al. (2013).

Proposition 6.6. *(Cox et al., 2013, p203, part of Theorem 10)*

Let $I, J \subset K[x_1, \ldots, x_n]$ *be ideals, then* $\mathrm{cl}\,(V(I) \setminus V(J)) \subset V(I : J^\infty)$,

Proof. We claim that $I \subset I : J \subset I : J^\infty \subset I(V(i0) \setminus V(J))$. We have $I \subset I : J \subset I : J^2 \subset \cdots \subset I : J^\infty$ by the ascending chain condition. Suppose that $f \in I : J\infty$ and $a \in V(I) \setminus V(J)$. Then $fg^N \in I$ for all $g \in J$ and a suitable (large enough) $N \geq 0$. Since $a \in V(I)$, $f(a)g^N(a) = 0$ (for all $g \in J$ and large enough N). Since $a \notin V(J)$, there exists $f \in J$ and $N \geq 0$ such that $g(a) \neq 0$. Hence $f(a) = 0$ for all $a \in V(I) \setminus V(J)$. We have $f \in I(V(I) \setminus V(J))$. Transitivity of $\subset$ proves the claim. Since V reverses inclusion, we have $= \mathrm{cl}\,(V(i) \setminus V(J)) = V(I(V(i) \setminus V(J))) \subset V(I : J^\infty)$. □

The proof of Lazard (2010), Proposition 5.14 begins with an irreducible polynomial $f \in \mathbb{Q}[x_1, x_2, x_3]$, such that there exists a $(1, 1, 0)$-cell $C \subset \{\mathbf{x} \in \mathbb{R}^3 \mid f(\mathbf{x}) = 0\}$, a point $\mathbf{p} = (p_1, p_2) \in \mathbb{R}^2$ such that $\mathbf{p}$ is a blow-up point of f, and $g \in \mathbb{Q}[x_1, x_2]$ such that $g(\mathbf{p}) = 0$. Consider the variety $V(\langle f\rangle + \langle g\rangle) = V(f) \cap V(g)$, which contains some of the one-dimensional cells which form part of $\mathrm{fr}\,(C)$ and the blow-up subset $B \subset \mathbf{p} \times \mathbb{R}$. The saturation by h of I,

$$S := \langle f, g, 1 - zh\rangle \cap \mathbb{Q}[x_1, x_2, x_3]$$

is constructed. By Proposition 6.6 (Cox et al. (2013), Theorem 10),

$$\mathrm{cl}\,(V(\langle f, g\rangle) \setminus V(\langle h\rangle)) \subset S.$$

Since our blow-up point $\mathbf{p}$ is 0-dimensional, computing this Zariski closure "fills in the hole" left by removing the blow-up subset from $V(\langle f, g\rangle)$. Thus, we have the limit points of the cell C as it approaches the blow-up subset B. In order to find the refinement points above $\mathbf{p}$, we compute a generating set (Groebner basis) $\{f_1, \ldots, f_k\}$ for S and solve the system of equations

$$f_1 = 0, \ldots, f_k = 0, g = 0, h = 0.$$

Note that this procedure only works if the blow-up points have dimension zero. Otherwise, the endpoints of 1-dimensional cells in the frontier of C will not be sufficient to satisfy the frontier condition.

6.2.1 Lazard's Algorithm for lifting with bad points in $\mathbb{R}^3$

Let $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD of $\mathbb{R}^n, n \leq 3$. If $n \leq 2$, $\mathcal{D}$ is already strong – well-bordered, boundary smooth and satisfies the frontier condition. In particular, by Proposition 6.2, the frontier condition is already satisfied since there are no bad

cells in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^2$ (Lazard, 2010, Lemma 4.3). Also by Proposition 6.2, any bad cells in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^2$ are 0-dimensional (Lazard, 2010, Lemma 4.3). Therefore, the frontier condition is satisfied everywhere except in the neighbourhoods above bad cells in the induced decomposition of $\mathbb{R}^2$. The following algorithm is a special method for lifting above a bad cell.

Algorithm 4. *Lazard Lifting With Bad Points in* $\mathbb{R}^3$

(Lazard, 2010, Algorithm 5.15)

Input:

$$(\mathcal{A}, \mathcal{D})$$

- $\mathcal{A} = \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ *is the family of projection polynomials, each* $\mathcal{A}_k \subset \mathbb{Q}[x_1, \ldots, x_k]$,
- $\mathcal{D}$*: is an* $\mathcal{A}$*-invariant CAD of* $\mathbb{R}^3$ *such that every cell of* $\mathcal{D}$ *is a topologically regular cell.*

Output:

$$\mathcal{R} := \{\mathcal{R}_{\mathbf{b}} = \{c_1, \ldots, c_t\} \subset \mathbb{A}\}$$

where $\mathbf{b} \in \mathbb{R}^2$ *is a* $(0,0)$*-cell in the decomposition induced by* $\mathcal{D}$ *on* $\mathbb{R}^2$*.*

A set of refinement points, as described in Section 5.3.1.

- *Let* $\mathcal{R} = \emptyset$*.*
- *Let* $\mathcal{D}'$ *be the decomposition induced by* $\mathcal{D}$ *on* $\mathbb{R}^2$*.*
- *For each* $(0,0)$*-cell* $\mathbf{b}$ *of* $\mathcal{D}'$*, determine whether it is a bad cell.*

  $\mathbf{b}$ *is a bad cell if there exists a polynomial* $f \in \mathcal{A}_3$ *such that* f *has constant sign on a* $(0,0,1)$*-cell* C *of* $\mathcal{D}$ *such that* $\mathbf{b} = \mathrm{proj}_{\mathbb{R}^2}(C)$*.*

  (Note: this is an easy check – simply look at the signs of polynomials in $\mathcal{A}_3$ *on* C*. This is part of the information stored about* C*.)*

- *If* $\mathbf{b}$ *is bad:*
  - *Let* $g_1 \in \mathcal{A}_1, g_2 \in \mathcal{A}_2$ *such that* $g_1(\mathbf{b}) = 0$ *and* $g_2(\mathbf{b}) = 0$ *(find* g_1 *and* g_2 *using signs of projection polynomials on* $\mathbf{b}$*).*
  - *Let* $\mathcal{A}' := \mathcal{A}_1 \cup \mathcal{A}_2$*.*
  - *For each* $g \in \mathcal{A}'$*, do*
    - *If* $g \in \mathbb{Q}[x_1]$*, then* $h := g_2$*. Otherwise,* $g \in \mathbb{Q}[x_1, x_2]$ *and* $h := g_1$*.*
    - *Define the saturation*

      $$I := \langle f, g, 1 - zh \rangle \cap \mathbb{Q}[x_1, x_2, x_3],$$

      where z *is a new variable.*

- *Compute a generator system S, i.e., a Groebner basis for the ideal I.*
- *Compute*
$$L := \{\mathrm{sub}_{\mathbf{b}}(f') \in \mathbb{Q}[x_3] \mid f' \in S\}$$
and find the real roots
$$(c_1, \ldots, c_t)$$
of L.
- *Let*
$$\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}_{\mathbf{b}} = (c_1, \ldots, c_t)\}.$$

- *return $\mathcal{R}$.*

Compute the refinements, given by $\mathcal{R}$, of sections above bad cells using the method described in Section 5.5.

6.3 Generalisation of Lazard

Every 2-dimensional cylindrical cell in a CAD of $\mathbb{R}^n$ is the graph of a continuous definable map $\mathbf{f} = (f_1, \ldots, f_{n-2}) : X \to \mathbb{R}^{n-2}$ where $X \subset \mathrm{span}\{x_i, x_j\}, 1 \leq i < j \leq n$ is a cylindrical $(1,1)$-cell by Proposition 2.4 (Basu et al. (2013), Lemma 3.3). According to the remark following Definition 2.22, each component $f_j : X \to \mathrm{span}\{x_j\}$ of $\mathbf{f}$ is a continuous definable map at every point in $\mathrm{cl}\,(X)$ except, perhaps, for a finite set of isolated points in $\mathrm{fr}\,(X) = \mathrm{cl}\,(X) \backslash X$. Therefore, it might be natural to ask whether the result from Lazard (2010) for lifting with bad points can be generalised to cylindrical decompositions of $\mathbb{R}^n, n > 3$ compatible with semialgebraic sets of dimension at most 2. Our goal is to construct a CAD $\mathcal{D}$ of $\mathbb{R}^n$, monotone with respect to each set $V_1, \ldots, V_k$ such that $\dim(V_i) \leq 2$ for all $1 \leq i \leq k$, such that $\mathrm{cl}\,(C)$, for each cell $C \subset V_1 \cup \ldots \cup V_k$ of $\mathcal{D}$ is a union of some cells of $\mathcal{D}$. Note that this method is not claimed to work for all cells of $\mathcal{D}$, in particular, those of dimension > 2.

In Section 6.2, we applied Lazard's method for lifting with bad cells to CADs of $\mathbb{R}^3$ output by the algorithm described in Section 5.3.1. Lazard's algorithm required the input CAD to have two properties. First, each cell must be boundary smooth. Since each cell $C \subset V = V_1 \cup \ldots \cup V_k$ is constructed to be monotone, it is already topologically regular by Proposition 2.2 (Basu et al. (2013), Theorem 1), and is therefore already boundary smooth. The second requirement is that every cell must be well-bordered. While Corollary 6.1 (Lazard (2010), Corollary 5.4) asserts that any sign-invariant CAD of $\mathbb{R}^3$ is well-bordered, this property cannot be extended to CADs of arbitrary dimension. Indeed, Lazard (2010), Example 2.11 describes a 3-dimensional cylindrical cell in $\mathbb{R}^4$ which cannot be well-bordered since its frontier consists of a single half-line. This condition is necessary to get the configuration of cells described in Proposition 6.4 (Lazard (2010), Lemma 5.13), which is required to apply the algorithm for lifting with

bad points. Therefore, it seems unlikely that this method could be generalised to cells of dimension > 2.

Definition 6.4. Let $\mathcal{D}$ be a CAD of $\mathbb{R}^n$ and C be a cell of $\mathcal{D}$. $\mathcal{D}$ will be called strong about C if C is well-bordered, boundary coherent and every cell contained in $\mathrm{cl}\,(C)$ is boundary smooth.

We first extend Corollary 6.1 (Lazard (2010), Corollary 5.4) to cells of dimension ≤ 2 in a sign-invariant CAD of $\mathbb{R}^n$. This can be done by extending Lazard (2010), Theorem 5.3.

Proposition 6.7. *(Lazard, 2010, Theorem 5.3)*

Let $\mathcal{D}$ be an $\mathbf{F}$-invariant CAD of $\mathbb{R}^3$ such that the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^2$ is strong. Then every cell C of $\mathcal{D}$ is well-bordered.

Proof. If $\dim(C) < 2$, then C is well-bordered by Lazard (2010), Lemma 5.1.

If $\dim(C) = 2$ and C is a $(1,1,0)$-cell, then its boundary consists of some section cells which project onto one-dimensional cells in $\mathcal{D}$ and which are contained in $\mathrm{fr}\,(C')$ and points or intervals projecting on some 0-cells in $\mathrm{fr}\,(C')$ by Proposition 6.1 (Lazard (2010), Proposition 5.2). We need to prove that these points belong to the boundary of some of the 1-cells contained in $\mathrm{fr}\,(C)$. Let $\mathbf{c}$ be a 0-cell of $\mathcal{D}'$ contained in $\mathrm{fr}\,(C')$ such that $\mathbf{c}$ is the projection of exactly one point, $\mathbf{p}$, in $\mathrm{fr}\,(C)$. Since C' is well-bordered (as $\mathcal{D}'$ is strong), there is a 1-dimensional cell $C_1' \subset \mathrm{fr}\,(C')$ such that $\mathbf{c}$ is an endpoint. If C_1' is the projection of a 1-dimensional cell contained in $\mathrm{fr}\,(C)$, we are done – the boundary of this one-dimensional cell consists of some isolated points including $\mathbf{p}$. C_1' is not the projection of a section cell, then the limits of the sequences of points in C, whose projections tend to C_1', tend to $-\infty$ or ∞. Therefore $\mathbf{p}$ is either $-\infty$ or ∞, being an endpoint of the interval I associated to C by Proposition 6.1 (Lazard (2010), Proposition 5.2). This contradicts the assumption that ${\mathrm{proj}_{\mathbb{R}^2}}^{-1}(\mathbf{c}) \cap \mathrm{fr}\,(C)$ contains only $\mathbf{p}$.

If $\dim(C) = 2$ and it is a sector ($(i_1, i_2, 1)$-cell), then $\mathrm{fr}\,(C)$ consists of the two $(i_1, i_2, 0)$-cells C_B and C_T which project on C' the side-wall $W = \mathrm{fr}\,(C) \backslash (C_B \cup C_T)$ which consists of points or intervals projecting on the endpoints of C'. If W contains an isolated point above an endpoint $\mathbf{c}$ of C'. Then Proposition 6.1 (Lazard (2010), Proposition 5.2) asserts that it belongs to the boundary of a section bounding C from below or above (i.e., C_B or C_T), and we are done.

Finally, suppose that C is a $(1,1,1)$-cell. By the proof of Lazard (2010), Theorem 4.4, we only need to consider points in $\mathrm{fr}\,(C)$ which projects on a bad cell, say $\mathbf{b}$, of $\mathcal{D}'$. By Lazard (2010), Lemma 4.3 $\dim(\mathbf{b}) = 0$. C_B and C_T are $(1,1,0)$-cells, therefore we only need to consider the blow-up subsets $B := (\mathrm{fr}\,(C) \setminus (\mathrm{cl}\,(C_B) \cup \mathrm{cl}\,(C_T))) \cap (\mathbf{b} \times \mathbb{R})$. By Proposition 6.1 (Lazard (2010), Proposition 5.2) B is an open interval. As C' is well-bordered, there is a 1-dimensional cell $C_1' \subset \mathrm{fr}\,(C')$ such that $\mathbf{b}$ is an endpoint. The set of points of

$\mathrm{fr}(C)$ which project onto C_1' are bounded from below and above by points of $\mathrm{fr}(C_B)$ and $\mathrm{fr}(C_T)$, thus they are 1-dimensional section cells. The boundaries of these section cells are included in $\mathrm{fr}(C_B)$ and $\mathrm{fr}(C_T)$. Proposition 6.1 (Lazard (2010), Proposition 5.2) implies that these sections which project onto C_1' define a (finite and non empty) 2-dimensional sector cell which projects onto C_1' whose boundary contains B.

As all cases have been considered, this completes the proof. □

Corollary 6.2. *(Corollary to Lazard (2010), Theorem 5.3)*

Let $\mathcal{D}$ be an **F***-invariant CAD of $\mathbb{R}^n$ and C be a cell of $\mathcal{D}$ of dimension ≤ 2 such that the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{n-1}$ is strong about $\mathrm{proj}_{\mathbb{R}^{n-1}}(C)$. Then C is well-bordered.*

Proof. If $\dim(C) \leq 2$, then it is well-bordered by Lazard (2010), Lemma 5.1. If $\dim(C) = 2$ and C is a sector cell, the same argument as in Proposition 6.7 (Lazard (2010), Theorem 5.3) can be applied.

Otherwise, $\dim(C) = 2$ and C is a sector cell. $C' := \mathrm{proj}_{\mathbb{R}^{n-1}}(C)$ is a well-bordered, boundary smooth and boundary coherent 2-dimensional cell in $\mathcal{D}'$. The same argument as in Proposition 6.7 (Lazard (2010), Theorem 5.3) can be applied to the cells in $\mathrm{cl}(C') \times \mathbb{R}$, which have dimension ≤ 2 or are sector cells of dimension 3. □

By Proposition 6.3, if no cells which intersect $\mathrm{fr}(C)$ project on a bad cell and the induced decomposition is strong, C is well-bordered, boundary smooth and satisfies the frontier condition. Since the proof of Proposition 6.3 only considers cells contained in $\mathrm{proj}_{\mathbb{R}^{n-1}}(\mathrm{cl}(C)) \times \mathbb{R}$ this property is also true for the weaker condition that the induced decomposition is strong about $\mathrm{proj}_{\mathbb{R}^{n-1}}(C)$. Thus, we need to look at what happens in the neighbourhood of bad cells (blow-up points) only. If there is a blow-up point, then we claim that the following situation, similar to that described in Proposition 6.4 (Lazard (2010), Lemma 5.13), occurs.

Lemma 6.2. *(Extension of Proposition 6.4 (Lazard, 2010, Proposition 5.13))*

Let $\mathcal{D}$ be a sign-invarient decomposition of $\mathbb{R}^n$ and C be a cell of $\mathcal{D}$ of dimension at most two, such that the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{n-1}$ is strong about $\mathrm{proj}_{\mathbb{R}^{n-1}}(C)$. Let D be a cell of $\mathcal{D}$ such that $B := D \cap \mathrm{fr}(C) \neq \emptyset$ and $B \neq D$. Then either

1. *C is a 1-dimensional section cell whose endpoint, B, projects on a bad cell, or*
2. *there exists a cell E of $\mathcal{D}$ such that $\dim(E) < \dim(C)$ and $\emptyset \neq D \cap \mathrm{fr}(E) \neq D$.*

Proof. By the proof of (Lazard, 2010, Theorem 4.4), D will always be a $(0, \ldots, 0, 1)$-cell in the cylinder above a bad cell D'.

We consider each case by cell index.

- $(i_1, \ldots, i_{n-1}, 1)$-cell, where $\sum_{j=1}^{n-1} i_j = 1$: observe that C (if bounded) is delineated by some $(i_1, \ldots, i_{n-1}, 0)$-cells (one-dimensional curve intervals). Take B to be the one-dimensional interval delineated by the endpoints of these curves. Since $B \neq C$, at least one of these endpoints is not a $(0, \ldots, 0, 0)$-cell of $\mathcal{D}$. Case 2 applies, taking as E the delineating $(i_1, \ldots, i_{n-1}, 0)$-cell.
- $(i_1, \ldots, i_{n-1}, 0)$-cells, where $\sum_{j=1}^{n-1} i_j = 1$: consider the previous case, letting $C = E$.
- $(i_1, \ldots, i_{n-1}, 0)$-cell, where $\sum_{j=1}^{n-1} i_j = 2$: C is necessarily the root of a polynomial $f \in \mathbb{Q}[x_1, \ldots, x_n]$ which vanishes identically on D', a $(0, \ldots, 0)$ cell in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{n-1}$. Let V_r be a small neighbourhood (with radius $r > 0$) centred around D' and consider V'_r, the cylinder above the boundary of V_r. Since C is a monotone cell, the maximum and minimum values of x_n in this intersection are obtained on the boundary of C, for sufficiently small r. The limits, as r tends to 0, of this maximum and minimum clearly belong to the boundary of B, if these limits do not tend to infinity. Note that both the maximum and minimum cannot be infinite, since this would imply that either B is empty, or coincides with D. Therefore, the non-infinite limits are the endpoints of some 1-dimensional cell belonging to the boundary of C. Case 2 applies, taking E to be this 1-dimensional cell.
- $(0, \ldots, 0, 1)$- or $(0, \ldots, 0, 0)$-cells: do not satisfy the hypothesis, there is nothing to do.

□

Remark. Let C be a $(i_1, \ldots, i_{n-1}, 0)$-cell as described in Lemma 6.2 such that $i_1 = i_k = 1$ for some $1 \leq k \leq n$ and all other elements of the index are equal to zero. The cell D, which projects on a bad-cell is necessarily a $(0, \ldots, 0, 1)$-cell and the cell E, which has dimension one, is contained in $\mathrm{cl}\,(C)$. Hence, E must have index either $(i_1, 0 \ldots, 0)$ or $(0, \ldots, 0, i_k, 0, \ldots, 0)$. By Proposition 2.4 (Basu et al. (2015), Lemma 3.3), the images of C, D and E under the projection map $\mathrm{proj}_{\mathrm{span}\{i_1, i_k, i_n\}}$ are cylindrical cells and satisfy Proposition 6.4 (Lazard (2010), Lemma 5.13).

This implies that we should be able to use a similar technique to that described in Proposition 6.5 (Lazard (2010), Proposition 5.14) to lift above bad cells and satisfy the frontier condition on two-dimensional section cells in decompositions of arbitrary dimension.

Conjecture 6.1. *(Extension of Lazard (2010), Proposition 5.14)*

Let $f \in \mathbb{Q}[x_1, \ldots, x_n]$ be an irreducible polynomial of the kind

$$f = h_d x_n^d + \ldots + h_1 x_n + h_0$$

with each $h_i \in \mathbb{Q}[x_1, \ldots, x_{n-1}]$*, and*

$$\mathbf{p} = (p_1, \ldots, p_{n-1})$$

be a common root of $h_d, \ldots, h_1, h_0$*. Let* $g_1, \ldots, g_{n-2} \in \mathbb{Q}[x_1, \ldots, x_{n-1}]$ *be irreducible, pairwise relatively prime polynomials having* $\mathbf{p}$ *as a root. Then the limits of the common roots of* f *and* $g_{i_1}, \ldots, g_{i_{n-2}}$ *for* $\{i_1, \ldots, i_{n-2}\} = \{1, \ldots, n-1\} \setminus \ell, 1 \leq \ell \leq n-1$ *as* $(x_1, \ldots, x_{n-1}) \to \mathbf{p}, (x_1, \ldots, x_{n-1}) \neq \mathbf{p}$ *may be computed as the solutions of a system of polynomial equations having dimension zero.*

Proof. Since f is irreducible, it has at least 2 coefficients, $h_1, h_2 \in \mathbb{Q}[x_1, \ldots, x_{n-1}]$, whose GCD is a constant. Therefore, any ideal generated by these coefficients has codimension at least two and, since polynomials $g_1, \ldots, g_{n-1}$ are irreducible and pairwise relatively prime, the ideal generated by the coefficients of f and the polynomials, $g_{i_1}, \ldots, g_{i_{n-2}}$ has dimension zero. Any ideal containing these polynomials also contains another element, g_ℓ, which is not a multiple of any of the $g_{i_1}, \ldots, g_{i_{n-2}}$'s.

The saturation by g_ℓ of the ideal $\langle f, g_{i_1}, \ldots, g_{i_{n-2}} \rangle$ is defined as the ideal

$$S := \langle f, g_{i_1}, \ldots, g_{i_{n-2}}, 1 - z g_\ell \rangle \cap \mathbb{Q}[x_1, \ldots, x_n].$$

The irreducible components of the zero set of S are those of the zero set of $\langle f, g_{i_1}, \ldots, i_{n-2} \rangle$ at which g_ℓ is not identically zero. It follows that the vertical line $\{x_1 = p_1, \ldots, x_{n-1} = p_{n-1}\}$ which is part of the zero set of f, $g_{i_1}, \ldots, g_{i_{n-2}}$ and g_ℓ, is not contained in the zero set of S. Meanwhile, the zero set of S contains the limit points we want to find.

Therefore, the ideal generated by S, the coefficients $h_0, \ldots, h_d$ of f and the polynomials $g_{i_1}, \ldots, g_{i_{n-2}}$ is zero-dimensional and the limits of $\{f = 0, g_{i_1} = 0, \ldots, g_{i_{n-n}} = 0\}$ as $(x_1, \ldots, x_{n-1}) \to \mathbf{p}$ can be computed as the solutions of S. □

Let $\mathcal{D}$ be an $\mathbf{F}$ invariant CAD of $\mathbb{R}^n$ as constructed by the algorithm in Section 5.3.1. Recall that, for the previous results, we require the decomposition induced on $\mathbb{R}^{n-1}$ to be strong about the projections of some cells of $\mathcal{D}$. Therefore, we need to proceed by induction on decompositions induced by $\mathcal{D}$ on $\mathbb{R}^k, k \geq 1$ to be able to apply Lemma 6.2 and Conjecture 6.1.

For $k \leq 2$, we have a $\text{proj}_{\mathbb{R}^k}(\mathbf{F})$-invariant CAD which already satisfies the frontier condition by Lazard (2010), Corollary 4.7. For $k = 3$, recall that every cell is well-bordered by Corollary 6.1 (Lazard (2010), Corollary 5.4). We already described how to construct a $\text{proj}_{\mathbb{R}^3}(\mathbf{F})$-invariant CAD of $\mathbb{R}^3$ satisfying the frontier condition using Lazard (2010), Algorithm 5.15.

For $k > 3$ let $\mathcal{D}'$ be the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^k$ and let C be a cell of $\mathcal{D}'$ such that ${\text{proj}_{\mathbb{R}^k}}^{-1}(C) \subset V_1 \cup \ldots \cup V_k$. By the induction hypothesis, the decomposition induced by $\mathcal{D}'$ on $\mathbb{R}^{k-1}$ is strong about $\text{proj}_{\mathbb{R}^{k-1}}(C)$. C is

topologically regular since the projection of a monotone cell is monotone and C is well-bordered by Corollary 6.2 since $\dim(C) \leq 2$. If $\dim(C) < 2$, C already satisfies the frontier condition since it is well-bordered. If $\dim(C') = 2$, then C is an $(i_1, \ldots, i_k)$-cell such that $i_{\ell_1} = i_{\ell_2} = 1$ and all other elements of the index are equal to zero.

First suppose that $\ell_1 = 1$. If $\ell_2 < k$ then C is a section cell. Let $C' = \text{proj}_{\mathbb{R}^{k-1}}(C)$. C' satisfies the frontier condition by the induction hypothesis and C is the graph of a continuous definable function $\varphi : C' \to \mathbb{R}$. $\text{cl}\,(C)$ is the graph of φ for all but, possibly, a finite number of isolated blow-up points in $\text{cl}\,(C')$ (see Remark after Definition 2.22). Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be the set of projection polynomials defining $\mathcal{D}'$. By the construction of CAD, these polynomials are irreducible. Let

$$C \subset \{\mathbf{x} \in \mathbb{R}^k \mid g_{j_1}(\mathbf{x}) = 0, \ldots, g_{j_{k-2}}(\mathbf{x}) = 0, f(\mathbf{x}) = 0\}$$

where each $g_i \in \mathcal{A}_i$ for $i \in \{j_1, \ldots, j_{k-2}\} = \{1, \ldots, k-1\} \setminus \{\ell_1, \ell_2\}$ and $f \in \mathcal{A}_k$. Suppose that there is a $(0, \ldots, 0, 1)$-cell D such that $D \cap \text{fr}\,(C)$ and $f(\mathbf{x}) = 0$ for all $\mathbf{x} \in D$. I.e., $\mathbf{b} := \text{proj}_{\mathbb{R}^{k-1}}(D)$ is a bad cell and is a common root of $g_{j_1}, \ldots, g_{j_{k-2}}$ and some $g_{\ell_1} \in \mathcal{A}_{\ell_1}, g_{\ell_2} \in \mathcal{A}_{\ell_2}$. Apply a method similar to that described in Lazard (2010), Algorithm 5.15 to C and D – using Conjecture 6.1, compute the saturations by g_{ℓ_1} and g_{ℓ_2}, respectively:

$$I_1 := \langle f, g_{j_1}, \ldots, g_{j_{k-2}}, g_{\ell_2}, 1 - z g_{\ell_1} \rangle \cap \mathbb{Z}[x_1, \ldots, x_k] \tag{6.5}$$

and

$$I_2 := \langle f, g_{j_1}, \ldots, g_{j_{k-2}}, g_{\ell_1}, 1 - z g_{\ell_2} \rangle \cap \mathbb{Z}[x_1, \ldots, x_k], \tag{6.6}$$

where z is a new variable. Let S_1, S_2 be Groebner bases for I_1, I_2 respectively. The union of zero-dimensional solutions to S_1, S_2 forms the set of refinement points $\mathcal{R}_{\mathbf{b}}$ whiche contains the endpoints of the open interval $D \cap \text{fr}\,(C)$, By considering each bad cell in $\text{fr}\,(()\text{proj}_{\mathbb{R}^{k-1}}(C))$, $\mathcal{D}'$ can be refined such that C satisfies the frontier condition.

On the other hand, if $\ell_2 = k$, C is a sector cell. In this case, C already satisfies the frontier condition since its top and bottom, which are one-dimensional section cells, already satisfy the frontier condition and its side-wall consists of at most two connected components – points or closed intervals, whose endpoints coincide with the endpoints of the top and bottom of C.

Finaly, if $\ell_1 > 1$ then C is a $(0, \ldots, 0, i_{\ell_1}, \ldots, i_k)$-cell. Apply the method described above to the $(i_{\ell_1+1}, \ldots, i_k)$-cell in the sub-cad above $\text{proj}_{\mathbb{R}^{i_\ell - 1}}(C)$.

Compute a refinement of $\mathcal{D}$ so that it is compatible with the set of refinement points $\mathcal{R}_{\mathbf{b}}$. These are refinements of the kind described in Basu et al. (2015), Lemma 3.11, and preserve the cylindrical structure and monotone cells in $\mathcal{D}$. By repeating this process for each bad cell $\mathbf{b}$, $\mathcal{D}'$ is refined such that it satisfies the frontier condition, hence it is strong about each cell C such that $\text{proj}_{\mathbb{R}^k}{}^{-1}(C) \subset V_1 \cup \ldots, \cup V_k$. This completes the inductive step and, by induction, a cylindrical decomposition monotone with respect to each set $V_1, \ldots, V_k$ and satisfying the frontier condition has been constructed.

6.3.1 Algorithm for lifting above bad points, for CAD compatible with sets of dimension ≤ 2

Let $\mathcal{D}$ be an **F**-invariant cylindrical algebraic decomposition of $\mathbb{R}^n$ monotone with respect to each bounded semialgebraic set $V_1, \ldots, V_k$ such that $\dim(V_i) \leq 2, 1, \leq i \leq k$ (e.g., the output produced by the algorithm from Section 5.3.1). We now extend Lazard (2010), Algorithm 5.15 to CAD of this kind. First, prepare $\mathcal{D}$ by marking each cell $C' := \text{proj}_{\mathbb{R}^r}(C)$, such that C' is a two-dimensional section cell in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^r$ such that C is a cell of $\mathcal{D}$ contained in $V_1, \ldots, V_k$ for $3 \leq r \leq n$.

Algorithm 5. *Extension of Lazard's Lifting With Bad Points, for 2-dimensional cells in $\mathbb{R}^n$*

(Extension of Lazard, 2010, Algorithm 5.15)

Input:

$$(\mathcal{D}, \mathcal{A})$$

- *$\mathcal{D}$: a CAD of $\mathbb{R}^k$, each cell having constant sign on $\mathcal{A}$, A collection of cells $\mathcal{C} := \{C_1, \ldots, C_m\}$ of $\mathcal{D}$ are marked as being projections of $V_1, \ldots, V_k$.*
- *$\mathcal{A} = \mathcal{A}_1, \ldots, \mathcal{A}_k$ is the family of projection polynomials for $\mathcal{D}$, each $\mathcal{A}_k \subset \mathbb{Z}[x_1, \ldots, x_k]$.*

Output:

$$\mathcal{R} := \{\mathcal{R}_{\mathbf{b}} = \{c_1, \ldots, c_t\} \subset \mathbb{A}\}$$

where $\mathbf{b} \in \mathbb{R}^{k-1}$ is a $(0, \ldots, 0)$-cell in the decomposition induced by $\mathcal{E}$ on $\mathbb{R}^k$.

A set of refinement points, as described in Section 5.3.1.

- *Let $\mathcal{D}'$ be the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{n-1}$.*
- *For each $(i_1, \ldots, i_{k-1}, 0)$-cell $E \in \mathcal{C}$, let $i_{\ell_1} = i_{\ell_2} = 1$, while all other elements in the index are equal to zero.*
  - *Let $\mathbf{c} := \text{proj}_{\mathbb{R}^{j_{\ell_1}-1}}(E)$ and compute the set of polynomials for the sub-cad of $\mathcal{D}$ above $\mathbf{c}$.*

    I.e., Let $C := \text{proj}_{\text{span}\{i_\ell, \ldots, i_k\}}(E)$ be an $(i_{\ell_1}, \ldots, i_{k-1}, 0)$-cell in the sub-cad of $\mathcal{D}$ above $\mathbf{c}$. Compute the substitution of polynomials

    $$\mathcal{A'}_j := \{\text{sub}_{\mathbf{c}}(f) \in \mathbb{Z}[x_{\ell_1}, \ldots, x_j] \mid f \in \mathcal{A}_{i_{\ell_1}-1+j}\}$$

    for $i_{\ell_1} \leq j \leq k$.
  - *Find, by examining signs of projection polynomials associated to C, polynomials*

    $$(g_{j_1}, \ldots, g_{k-3}, f), g_j \in \mathcal{A'}_j, j \in \{\ell_1 + 1, \ldots, k-1\} \setminus \{\ell_2\}, f \in \mathcal{A'}_k$$

such that $C \subset \{\mathbf{x} \in \mathbb{R}^{k-\ell_1+1} \mid (g_{j_1}(\mathbf{x}) = 0, \ldots, g_{k-3}(\mathbf{x}) = 0, f(\mathbf{x}) = 0\}$.

- *Determine whether any point on the frontier of* C *projects on a bad cell. I.e., check if there is a* $(0, \ldots, 0, 1)$*-cell* D*, by examining signs of projection polynomials, such that* $f(\mathbf{x}) = 0$ *for all* $\mathbf{x} \in D$.

  I.e., all of $g_{j_1}, \ldots, g_{j_{k-3}}, f$ *should be equal to zero on* D.

- *For each* D *projecting on a bad cell, let* $\mathbf{b} := \mathrm{proj}_{\mathbb{R}^{k-1}}(D)$.

  Since $\mathbf{b}$ *is a* $(0, \ldots, 0)$*-cell, polynomials* $g_{j_1}, \ldots, g_{j_{k-3}}$ *are equal to zero on* $\mathbf{b}$ *and there are two more polynomials* $g_{\ell_1} \in \mathcal{A}'_{\ell_1}, g_{\ell_2} \in \mathcal{A}'_{\ell_2}$ *which are equal to zero on* $\mathbf{b}$.

  - *Compute saturations*

    $$I_1 := \langle f, g_{j_1}, \ldots, g_{j_{k-3}}, g_{\ell_2}, 1 - z g_{\ell_1} \rangle \cap \mathbb{Z}[x_1, \ldots, x_k]$$

    and

    $$I_2 := \langle f, g_{j_1}, \ldots, g_{j_{k-3}}, g_{\ell_1}, 1 - z g_{\ell_2} \rangle \cap \mathbb{Z}[x_1, \ldots, x_k],$$

    where z *is a new variable.*

  - *Compute generator systems* S_1, S_2 *for* I_1, I_2 *respectively.*

    This may be done by computing a Groebner basis eliminating z.

  - *Finally, to obtain refinement polynomials for* D*, substitute*

    $$L := \{\mathrm{sub}_{\mathbf{b}}(f) \in \mathbb{Q}[x_k] \mid f \in S_1 \cup S_2\}.$$

  - *Refinement points are the roots*

    $$(c_1, \ldots, c_t)$$

    of polynomials in L.

  - *Let*

    $$\mathcal{R} := \mathcal{R} \cup \{\mathcal{R}_{\mathbf{b}} = (c_1, \ldots, c_t)\}.$$

- *return* $\mathcal{R}$.

Apply this procedure to each decomposition, for $3 \leq k \leq n$ induced by $\mathcal{D}$ on $\mathbb{R}^k$ and compute the refinement of sections above bad cells using the same method described in Section 5.5 to obtain a refinement $\mathcal{E}$ of $\mathcal{D}$. Note that, since refinement points are algebraic numbers splitting (1)-cells in induced decompositions, Basu et al. (2015), Lemma 3.11 asserts that the cylindrical structure and monotone cells are preserved in $\mathcal{E}$.

6.3.2 Correctness and Complexity

Since zero- and one-dimensional cells already satisfy the frontier condition, it suffices to consider cells of dimension two. Let $\mathcal{D}$ be a CAD of $\mathbb{R}^k$ and C be a two-dimensional section cell of $\mathcal{D}$ such that the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{k-1}$ is strong about C. Corollary 6.2 asserts that C is well-bordered, so its frontier consists of the closure of some one-dimensional cells of $\mathcal{D}$. The frontier condition fails in the region of sector cells of $\mathcal{D}$ above 0-dimensional bad cells in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{k-1}$. The procedure described above explains how to obtain some new polynomials which allow these one-dimensional sectors to be refined such that they are compatible with the frontier of C. By performing this refinement, C is made to satisfy the forntier condition.

If C is a 2-dimensional sector cell, then its top and bottom are 1-dimensional sector cells of $\mathcal{D}$ and the side-wall of C is contained in cylinders $\mathbf{c} \times \mathbb{R}$, where $\mathbf{c}$ is a zero-dimensional cell which is an endpoint of $\text{proj}_{\mathbb{R}^{k-1}}(C)$. It follows that C satisfies the frontier condition sits its top and bottom, which are one-dimensional cells, satisfy the frontier condition.

6.3.3 Complexity analysis

We will see, in Section 8, that there is an algorithm for obtaining the frontier condition on a CAD compatible with a semialgebraic set of arbitrary dimension. This algorithm has complexity $(sd)^{O(1)^{n2^n}}$ – triply exponential in the number of variables. However, in the particular case of semialgebraic sets having dimension at most two, the complexity may be lower. This is now discussed.

Let us first discuss complexity in relation to the procedure described by Basu et al. (2015) in Theorem 3.20. Given the family $V_1, \ldots, V_k$, $\text{fr}\,(C)$ is computed for each 2-dimensional cell $C \subset V_1 \cup \ldots \cup V_k$ According to Lemma 8.1, there is an algorithm, which uses Basu et al. (2006) Algorithm 14.21 as its main subroutine, for computing $\text{fr}\,(C)$, having complexity

$$(sd)^{O(n^2)},$$

where C is defined by s different polynomials of maximum degree d. C has constant sign on all polynomials defining the CAD, so, using the bounds discussed in Sections 4.3.1 and 5.4.1, the complexity upper bound for computing $\text{fr}\,(C)$ is

$$\left((s(d+1))^{O(1)^n} \cdot (s(d+1))^{O(1)^n}\right)^{O(n^2)},$$

which is asymptotically the same as

$$(s(d+1))^{O(1)^n}..$$

Note that, in some cases, only some of the $(s(d+1))^{O(1)^n}$ polynomials are required to define the cell C. This is, according to Lemma 8.1, an upper bound on the number of polynomials and their degrees in the QFF defining $\text{fr}\,(C)$. The

procedure requires us to partition fr (C) into points and monotone curve intervals. In order to do this, a similar process as that used to obtain monotone cells (see Section 5) can be used. This will be repeated for each of the 2-dimensional cells contained in $V_1 \cup \ldots \cup V_k$. As such, the overall complexity will be the same as that presented in Section 5.4.1. I.e.,

$$(s(d+1))^{O(1)^n}.$$

Now let us consider the extension of Lazard (2010), discussed in Section 6.3. For each 1-dimensional sector cell which projects on a 0-dimensional bad cell, the saturation of ideals, defined in Equations (6.5) and (6.6) are computed. There will be at most n polynomials, in $n+1$ variables, appearing in these ideals. The saturation of these ideals is computed by finding a Groebner basis for them. According to Lakshman (1991), if the ideal $I = \langle f_1, \ldots, f_k \rangle$ where $f_1, \ldots, f_k$ are polynomials in n variables, then a Groebner basis for I can be computed in time singly exponential in n. However, in our case, only the saturation of the ideal over the real numbers is zero-dimensional, rather than the original ideal over complex numbers before eliminating the new variable z. Therefore, the complexity of computing a Groebner basis is doubly exponential in the number of variables – 2^{2^n} (Mayr, 1997). In the worst case, we may be considering a $(1, 1, 0, \ldots, 0)$-cell, part of whose frontier projects on a bad $(0, 0)$-cell in the induced decomposition of $\mathbb{R}^2$. It follows that at most $n-2$ Groebner basis computations will be required. Thus, the total complexity for lifting above a bad cell is at most

$$n \cdot 2^{2^n} = 2^{n2^n}.$$

The bound on the number of bad cells is unknown, but it is obviously bounded from above by the total number of cells in the induced decomposition of $\mathbb{R}^{n-1}$. I.e., $(s(d+1))^{2^{O(n-1)}}$. It follows that the complexity of the algorithm arising from the generalisation of Lazard is

$$(s(d+1))^{O(1)^{n-1}} \cdot 2^{n2^n},$$

which is bounded from above by

$$(s(d+1))^{O(1)^n}.$$

Both approaches – Basu et al. (2015) and the generalisation of Lazard (2010) – have the same complexity upper bound.

To summarise, each of the three stages – quasi-affine, monotone and frontier condition – of the algorithm for computing a CAD which is monotone with respect to all $V_1, \ldots, V_k$, such that $\dim(V_i) \leq 2$ for all $1 \leq i \leq k$, and satisfying the frontier condition each has complexity $(s(d+1))^{2^{O(n)}}$. Thus, the overall complexity of this algorithm is

$$(s(d+1))^{O(1)^n}.$$

By Proposition 2.8, this is also an upper bound on the number of cells, number of polynomials and their degrees defining this CAD.

Recall that the complexity of computing a "classical" CAD (from Proposition 2.8) is

$$(sd)^{O(1)^n}.$$

Therefore, the monotone CAD algorithm has, as expected, a slightly higher complexity than that of the classical algorithm. However, the complexity is not significantly worse, still being doubly exponential in the number of variables. This discussion completes the proof of Theorem 1.2.

Chapter 7

Implementation And Testing

7.1 Implementation on top of QEPCAD

There are several well-known implementations of CAD. For example, Maple's `CylindricalAlgebraicDecompose`, which uses regular chains instead of the classical projection and lifting algorithm of Collins (1975), is reported to work very efficiently (Chen and Moreno Maza, 2014). Another efficient implementation of CAD is available in Mathematica. However, `QEPCAD-B`, a quantifier elimination and cylindrical algebraic decomposition program written in C++ and built on top of the polynomial library SACLIB, was chosen as the starting point for the monotone CAD algorithm.

`QEPCAD` stands for Quantifier Elimination by Partial Cylindrical Algebraic Decomposition. It takes as input a first-order Boolean formula F, containing k free variables and $n-k$ variables bound by quantifiers, along with a variable ordering $x_1 < \cdots < x_n$. It then constructs a CAD compatible with the set $S \subset \mathbb{R}^k$ defined by F and returns a quantifier-free Boolean formula defining S. As well as the output formula, the user is able to examine the polynomials generated in the projection phase and view information about each cell of the CAD, including their dimension, signs of projection factors, multiplicities of roots of polynomials and the sample point associated with the cell. `QEPCAD B` is an extended version of `QEPCAD` developed by Brown (2003). As well as $\forall$ and $\exists$, `QEPCAD B` introduces some special quantifiers which allow the user to specify "exists exactly k", "exists infinitely many" and "for all but finitely many", allowing the problem to be solved in a more efficient way. `QEPCAD B` is also able to plot and determine cell adjacencies for decompositions of $\mathbb{R}^2$. An extended language for output formulas, which uses "i-th root of polynomial f", somewhat similar to the output given by Maple's `CylindricalAlgebraicDecompose`, is also included.

`QEPCAD B` and `SACLIB` are open-source, meaning it is easy to extend their functionality. The computer algebra library `Singular` is also integrated, which gives

the developer access to more efficient algorithms, e.g., for polynomial factorisation. Integration with `Singular` also gives the developer access to additional computer algebra tools, such as Groebner bases, which are leveraged for the first time in the implementation of Lazard Lifting algorithm. `QEPCAD B` by default provides the output as a quantifier-free (Tarski) formula, which is the most convenient representation for our purposes. As shown by Collins (1975), it is not always possible to obtain such formulas using only the polynomials produced in the projection phase. For this reason, Maple's `CylindricalAlgebraicDecompose` is only able to produce formulas in the extended language. `QEPCAD B` implements an algorithm to add additional defining polynomials (Brown, 1999). A small modification to this algorithm allows every cell in the CAD to be defined by a quantifier-free Boolean formula (see Section 2.2.5). This was the main realon for choosing `QEPCAD B`, along with the fact that `QEPCAD B` is open-source which allows the algorithm to easily be modified.

7.1.1 Description of QEPCAD B

The `QEPCAD B` algorithm proceeds in five stages:

1. **Input**

   The user is prompted to enter the following information:

   - *Informal problem description*, for the user's benefit
   - *Variable list* $(v_1, \ldots, v_n)$ where $v_i, 1 \leq i \leq n$ are variable names. Note that this list specifies the variable ordering $v_1 < \cdots < v_n$.
   - *Number of free variables* $0 \leq k \leq n$ appearing in the first-order formula (free variables are those not bound by quantifiers)
   - *Prenex formula* a first order Boolean formula $(Q_1 v_1), \ldots, (Q_k v_k) F$, where F is a quantifier-free Boolean formula defined by induction such that
     - atoms are polynomial equations, inequations, strict or non-strict inequalities, with polynomials in $\mathbb{Z}[v_1, \ldots, v_n]$,
     - Suppose that quantifier-free Boolean formulas $F_1, F_2, \ldots, F_k$ are defined, then

       $$\neg F_1,\ F_1 \Rightarrow F_2,\ F_1 \Leftarrow F_2,\ F_1 \Leftrightarrow F_2,\ F_1 \wedge \ldots \wedge F_k,\ F_1 \vee \ldots \vee F_k$$

       are quantifier-free Boolean formulas.
     - $(Q_i, v_i), 1 \leq i \leq k$ where

       $$Q_i \in \{\exists, \exists_k, \exists_\infty, \forall, \forall_\neg\}$$

       quantify the first k variables in the list $(v_1, \ldots, v_n)$. Quantifiers $\forall, \exists$ are defined as expected. New quantifiers are defined as follows
     - $\exists_k v_i$: there are exactly $k \in \mathbb{N}$ values of v_i satisfying F.
     - $\exists_\infty v_i$: there are infinitely many values of v_i satisfying F,
     - $\forall_\neg$: all but a finite number of values of v_i satisfy F.

2. **Normalisation**

   Let $(Q_1, v_1), \ldots, (Q_k, v_k)F$ be the input formula. The normalisation phase uses logical equivalences and polynomial factorisation to transform F into a "normalised" formula G, defined by induction as follows:

   - atoms are of the form $f * 0$ where $f \in \mathbb{Z}[v_1, \ldots, v_n]$ is a primitive, irreducible polynomial and $* \in \{=, \neq, <, \leq, >, \geq\}$.
   - Suppose that normalised formulas $G_1, \ldots, G_k$ are already defined, then
     $$G_1 \wedge \cdots \wedge G_k \text{ and } G_1 \vee \cdots \vee G_k$$
     are normalised formulas.
   - The family
     $$\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$$
     of input polynomials is constructed, where each $\mathcal{A}_k \subset \mathbb{Z}[v_1, \ldots, v_k]$ contains the polynomials appearing in G with nonzero degree in v_k but zero degree in all variables $v_{k+1}, \ldots, v_n$. $\mathcal{A}_k$ is called the set of level-k input polynomials.

3. **Projection**

   The projection phase of CAD is then performed. The user is able to specify the projection operator to be used at each level. Implemented operators are

   - Collins' projection,
   - McCallum's projection,
   - Hong's projection,
   - Lazard's projection,
   - "Partial" reduced McCallum projection (only adds the leading coefficient if it can be proved that lifting is never performed over a cell on which the polynomial vanishes identically),
   - McCallum's projection excluding leading coefficients.

   By default, the *partial reduced McCallum* projection operator is used. It is noted in the documentation that McCallum's projection operator is preferred, as it produces fewer projection polynomials, but can sometimes fail during the lifting phase. In this case, an error is printed and the user is advised to use Hong's projection operator, which is guaranteed to succeed, but produces more polynomials.

   Families $\mathcal{P} := (\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and $\mathcal{F} := (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ are constructed, where $\mathcal{P}_k$ is the set of level-k projection polynomials, and $\mathcal{F}_k$, containing factorised elements of $\mathcal{A}_k \cup \mathcal{P}_k$ is the set of level-k "projection factors". Elements of $\mathcal{F}$ are used to construct the CAD in the next step.

4. **Lifting** (stack construction)

By default, `QEPCAD B` constructs a CAD which is compatible with the set defined by the input formula, Before the lifting phase is started, the user may specify `full-cad`, which constructs a CAD with constant sign on all projection factors.

A CAD $\mathcal{D}$ of $\mathbb{R}^{n-k}$ (free variable space) is constructed such that the union of true cells is the set defined by the input formula. Truth values are either explicitly assigned to cells using the input formula or propagated down to free variable space using the quantifiers appearing in the input formula.

5. **Solution formula construction**

   Given the CAD $\mathcal{D}$, whose union of true cells coincides with the set defined by the input formula, a quantifier-free Boolean formula defining this set is produced. `QEPCAD B` attempts to use the projection factors in this formula, but sometimes additional polynomials (derivatives of projection factors) must be added in order to define the required set.

7.1.2 QEPCAD Cells and Sample Points

Since the main advantage of the monotone CAD algorithm is to produce a CAD with cells having desirable topological properties, we are mostly interested in examining individual CAD cells rather than QEPCAD's output formula.

Consider a simple example of a CAD compatible with the unit circle in $\mathbb{R}^2$, defined by the quantifier-free Boolean formula

$$F = \{x^2 + y^2 - 1 = 0\}.$$

Below is an excerpt from QEPCAD's output.

```
============================================================
Enter an informal description  between '[' and ']':
[ Unit Circle ]
Enter a variable list:
(x,y)
Enter the number of free variables:
2
Enter a prenex formula:
[ x^2 + y^2 = 1 ].

============================================================

Before Normalization >
g

Before Projection (y) >
g
```

```
Before Choice >
g

Before Solution >
d-proj-fac
P_1,1  = fac(J_1,1) = fac(dis(A_2,1))
       = x + 1

P_1,2  = fac(J_1,1) = fac(dis(A_2,1))
       = x - 1


A_2,1  = input
       = y^2 + x^2 - 1



Before Solution >
d-true-cells
---------- Information about the cell (4,2) ----------

Level                       : 2
Dimension                   : 0
Number of children          : 0
Truth value                 : T    by trial evaluation.
Degrees after substitution  : Not known yet or No polynomial.
Multiplicities              : ((1,2))
Signs of Projection Factors
Level 1  : (+,0)
Level 2  : (0)
----------   Sample point  ----------
The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1
             = 1.0000000000
Coordinate 2 = 0
             = 0.0000000000


-----------------------------------------------------
---------- Information about the cell (3,4) ----------
```

```
Level                       : 2
Dimension                   : 1
Number of children          : 0
Truth value                 : T    by trial evaluation.
Degrees after substitution  : Not known yet or No polynomial.
Multiplicities              : ((1,1))
Signs of Projection Factors
Level 1  : (+,-)
Level 2  : (0)
----------   Sample point  ----------
The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = 1
             = 1.0000000000


-----------------------------------------------------
---------- Information about the cell (3,2) ----------

Level                       : 2
Dimension                   : 1
Number of children          : 0
Truth value                 : T    by trial evaluation.
Degrees after substitution  : Not known yet or No polynomial.
Multiplicities              : ((1,1))
Signs of Projection Factors
Level 1  : (+,-)
Level 2  : (0)
----------   Sample point  ----------
The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = -1
             = -1.0000000000


-----------------------------------------------------
```

```
---------- Information about the cell (2,2) ----------

Level                       : 2
Dimension                   : 0
Number of children          : 0
Truth value                 : T    by trial evaluation.
Degrees after substitution  : Not known yet or No polynomial.
Multiplicities              : ((1,2))
Signs of Projection Factors
Level 1  : (0,-)
Level 2  : (0)
----------   Sample point  ----------
The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = -1
             = -1.0000000000
Coordinate 2 = 0
             = 0.0000000000


--------------------------------------------------

Before Solution >
finish

An equivalent quantifier-free formula:

y^2 + x^2 - 1 = 0


====================  The End  ======================

--------------------------------------------------------------------------------
0 Garbage collections, 0 Cells and 0 Arrays reclaimed, in 0 milliseconds.
488853 Cells in AVAIL, 500000 Cells in SPACE.

System time: 9 milliseconds.
System time after the initialization: 3 milliseconds.
--------------------------------------------------------------------------------
```

A QEPCAD cell C includes the following information

- **Positional index** $(j_1, \ldots, j_{n-k}), j_i \in \mathbb{N}, 1 \leq i \leq n-k$, indicates the position of this cell in the CAD;

- **Level** $k \in \mathbb{N}$, the level of the cell;
- **Dimension** $d \in \mathbb{Z}_{\geq 0}$, the dimension of the cell;
- **Number of children**, where a child is a CAD cell D of level $k+1$ such that $\text{proj}_{\mathbb{R}^k}(D) = C$;
- **Truth value** (and how it was determined), $t \in \{\text{TRUE}, \text{FALSE}, \text{UNDET}\}$, truth values may be determined by trial evaluation, propagation or equational constraints;
- **Degrees after substitution**, $(d_1, \ldots, d_m) \in \mathbb{Z}_{\geq 0}^m$ such that d_i is the degree of the i-th level k projection factor after substitution;
- **Multiplicities**, $((i_1, m_1), \ldots, (i_\ell, m_\ell)) \in (\mathbb{N} \times \mathbb{N})^\ell$, where m_j is the multiplicity of the root of the i_j-th substituted level-k projection factor;
- **Signs of Projection Factors** (for each level) $(S_1, \ldots, S_k)$, where each $S_i = (s_1, \ldots, s_{m_k}) \in \{1, -1, 0\}^m$ indicates the signs of level-$(k-i+1)$ projection factors
- **Sample Point** $\mathbf{x} \in \mathbb{A}^k$.

A sample point of a cell C is an algebraic number $\mathbf{x} = (x_1, \ldots, x_k) \in C$. This may be represented in one of two forms in QEPCAD. A primitive sample point $\mathbf{x}$ is an element of the algebraic extension field $\mathbb{Q}(\alpha)$ and is represented by the triple

$$(M, I, \mathbf{c})$$

where

- $M \in \mathbb{Z}[x]$ is a primitive, irreducible polynomial having $\alpha \in \mathbb{A}$ as a root,
- $I \subset \mathbb{Q}$ is an isolating interval for α as a root of M,
- $\mathbf{c} = (x_1, \ldots, x_n) \in \mathbb{Q}(\alpha)^n$.

The `SACLIB` representation for rational and algebraic numbers is used. A rational number $x = a/b$ is stored as a two-element list (a, b). An element $x \in \mathbb{Q}(\alpha)$ is stored as a two-element list $(q, g) \in \mathbb{Q} \times \mathbb{Z}[\alpha]$, where $x = q \cdot g(\alpha)$. If $x \in \mathbb{Q}(\alpha)$ is rational, then it is stored as $x = x \cdot \alpha^0 = x \cdot 1$. Likewise, if the sample point $\mathbf{x} \in \mathbb{Q}^k$, i.e., it only contains rational numbers, the minimal polynomial $M := x$ and isolating interval $I = (0, 0)$.

For a sample point $\mathbf{x} = (x_1, \ldots, x_{k-1}, x_k) \in \mathbb{A}^k$ with coordinates $(x_1, \ldots, x_{k-1}) \in \mathbb{Q}(\alpha)$ (where at least one of $x_1, \ldots, x_{k-1}$ is irrational) and irrational $x_k \notin \mathbb{Q}(\alpha)$, the sample point is stored in extended (non-primitive) form. This representation splits $\mathbf{x}$ into $\mathbf{y} = (x_1, \ldots, x_{k-1}) \in \mathbb{Q}(\alpha)$ and $x_k = \beta$ and is stored as the quintuple

$$(M_\alpha, I_\alpha, \mathbf{c}, M_\beta, J_\beta),$$

where

- $M_\alpha, I_\alpha, \mathbf{c}$ represents $\mathbf{y} \in \mathbb{Q}(\alpha)$ as described in the primitive representation
- $M_\beta \in \mathbb{Q}(\alpha)[y]$ is a primitive, irreducible polynomial having $\beta \in \mathbb{A}$ as a root,
- $I_\beta \subset \mathbb{Q}$ is an isolating interval for β as a root of M_β,

Since any finite set of algebraic numbers belongs to a single algebraic extension field $\mathbb{Q}(\gamma)$, it is always possible to convert a sample point in extended form to primitive form. The QEPCAD function CONVERT($\mathbf{c}, k$) converts a non-primitive sample point $\mathbf{c} \in \mathbb{A}^k$ into primitive representation by computing an appropriate (M_γ, I_γ) so that $\mathbf{c} \in \mathbb{Q}(\gamma)$.

7.1.3 Modifications to QEPCAD

The QEPCAD algorithm has been modified to include construction of quasi-affine cells, refinement of the CAD to produce monotone cells, and refinement of sectors above "bad cells" to obtain the frontier condition. A new option, `mct <y / n>` has been added, to enable or disable (respectively) the construction of a monotone CAD. The algorithm now proceeds as follows.

1. Input

   Let F be a Prenex formula defining a semialgebraic set $V = V_1 \cup \ldots V_k$ where each $V_i, 1 \leq i \leq k$ has dimension ≤ 2.

2. Normalisation

   Normalise the formula F and construct the set of input polynomials

   $$\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$$

   as described above.

   If the user enters `mct y`, monotone cells will be constructed. This also enables `full-cad`, so that the CAD will be sign-invariant on $\mathcal{A}$.

3. **Projection and Quasi-affine cells**

   If `mct` is enabled, proceed as described in Section 4. Every one- and two- dimensional cell will be smooth and polynomials which are zero at the critical points of projections to two- and one-dimensional coordinate subspaces will be added to $\mathcal{A}$. As the projections of polynomials in $\mathcal{A}$ are computed as part of the algorithm in Section 4.2.1, this stage also constructs the projection polynomials $\mathcal{P}$ and projection factor set $\mathcal{F}$. In order to ensure smooth and quasi-affine cells, new polynomials and their projections are included in the projection factor set $\mathcal{F}$.

   ~~Projection~~ (has already been completed)

4. **Lifting**

   Constructs an $\mathcal{A}$-invariant CAD $\mathcal{D}$. If `mct` is enabled, each cell $C \subset V_1 \cup \ldots \cup V_k$ of $\mathcal{D}$ will be the graph of a quasi-affine map, due to the polynomials added in the previous step.

5. **Refinement Phase**

- *Monotone Cells*

  Following the algorithm described in Section 5.3.1, construct the family of refinement points

  $$\mathcal{R} := \{\mathcal{R}_{\mathbf{b}} = (c_1, \ldots, c_t) \subset \mathbb{A} \mid \mathbf{b} \in \mathbb{R}^{k-1}\}$$

  where is a $(0, \ldots, 0)$-cell in the decomposition induced by $\mathcal{E}$ on $\mathbb{R}^k$, and add new polynomials (to $\mathcal{A}$, along with their factors to $\mathcal{F}$) which are equal to zero at these refinement points.

  For each refinement point $c \in \mathbb{A}$ in each family $\mathcal{R}_{\mathbf{b}}$, refine the sub-CAD of $\mathbb{R}^{n-k}$ $\mathcal{D}$ above the 0-cell $\mathbf{b}$ by intersecting it with straight lines and half-planes

  $$\{x_{k+1} < c\}, \{x_{k+1} = c\}, \{x_{k+1} > c\}.$$

  Positional indices and sample points of refined cells are updated.

- *Frontier Condition*

  Applies the method, similar to that described in Lazard (2010), to 0-dimensional "bad cells" (see Section 6.3.1). Groebner bases are constructed using `Singular`, which is already used for faster polynomial factorisation in `QEPCAD B`.

6. Solution formula construction

   Proceeds unchanged – prints the solution formula and exits the program.

   The user may pause here to examine the cells in the refined CAD.

In the forthcoming sections, the new code added to `QEPCAD B` is presented and discussed.

7.1.4 Quasi-affine

The function responsible for constructing a CAD containing quasi-affine cells is `QUASIAFFINE`

- **input**
  - `Word r`: number of variables
  - `Word V`: variable list
  - `Word F`: normalised input formula
- **Output:**
  - `Word* A_`: set of input polynomials
  - `Word* P_`: set of projection polynomials
  - `Word* J_`: set of projection factors

Given the normalised input formula, this function implements a modified version of the projection phase. It returns the polynomials sufficient to construct a CAD

such that each one- and two-dimensional cell contained in the set defined by the formula F is smooth and the graph of a quasi-affine map.

```
void QepcadCls::QUASIAFFINE(Word r, Word V, Word F, Word* A_,
↪  Word* P_, Word* J_)
{
    Word C, Ps, C1s, C2s, Js = NIL, PIs, PFs, PPs;
    SmoothOneTwoDim(r, V, F, &C1s, &C2s, &PIs, &PPs, &PFs);
    Word PF1 = CINV(PFs); // reverse order of proj factors to
↪  match cells

    // one-dimensional cells
    // i represents the projection onto one-dimensional
    ↪  coordinate subspace proj(i)
    // i.j represents proj(i,j), but we consider all minors with
    ↪  n-2 polynomials
    while (C1s != NIL) {
        ADV(C1s, &C, &C1s);
        Word Ps = ZeroPols(PF1, r, C);

        // proj(i)
        for (int i = 1; i <= r; ++i) {
            Word Is = GenerateIndex(r, i,0);
            Word J = JACOBI(r, NIL, 0, Ps, Is);

            if (!IPCONST(r, J)) {
                Js = COMP(J, Js);
            }
        }
    }

    // two-dimensional cells
    // i.j represent 2-dimensional coordinate subspace proj(i,j),
    ↪  and all minors for proj(i)
    while (C2s != NIL) {
        ADV(C2s, &C, &C2s);
        Word Ps = ZeroPols(PF1, r, C);

        // proj(i,j)
        for (int i = 1; i < r; ++i) {
            for (int j = i + 1; j <= r; j++) {
                Word Is = GenerateIndex(r, i,j);
                Word J = JACOBI(r, NIL, 0, Ps, Is);
                if (!IPCONST(r, J)) {
                    Js = COMP(J, Js);
                }
```

```
39
40              }
41          }
42      }
43
44      // CAD should be sign-invariant on Js. this means it should
        ↪  be compatible with their projections, too.
45      // we add projections here to save recomputing the projection
        ↪  for entire CAD.
46
47      // assign the pointers.
48      *A_ = PIs; // input polynomials
49      *P_ = PFs; // projection factors
50      *J_ = PPs; // projection polynomials
51
52      // add Js...
53      // Ps1 contains QEPCAD polynomials, Fs1 contains its factors.
54      Word Ps1, PsQ, Js1, Fs1;
55      ProcessPolynomials(r, Js, &Ps1, &PsQ, &Js1, &Fs1);
56      ADDPOLS(Ps1, r, LFS("Q"), J_);
57      ADDPOLS(Fs1, r, LFS("Q"), P_);
58
59      // and its projections
60      while (r > 1) {
61          Js = ProjMcxUtil(r, PsQ);
62          Js = CONC(Js, Js1); // polynomials with 0 degree in x_r
63          --r;
64
65          ProcessPolynomials(r, Js, &Ps1, &PsQ, &Js1, &Fs1);
66          ADDPOLS(Ps1, r, LFS("Q"), J_);
67          ADDPOLS(Fs1, r, LFS("Q"), P_);
68      }
69  }
```

- On line 4, the function `SmoothOneTwoDim` is called. It constructs a CAD $\mathcal{E}$ of $\mathbb{R}^r$, compatible with the set $V = V_1 \cup \ldots \cup V_K \subset \mathbb{R}^r$ defined by the input formula F using the McCallum projection operator so that every one- and two-dimensional cell of $\mathcal{E}$ contained in V is a smooth manifold. In fact, $\mathcal{E}$ is sign-invariant on the polynomials in F, so $\mathcal{E}$ is compatible with each $V_1, \ldots, V_k$. It returns
  - `C1s`: set of one-dimensional cells of $\mathcal{E}$ contained in V,
  - `C2s`: set of two-dimensional cells of $\mathcal{E}$ contained in V,
  - `PIs`: set of input polynomials defining $\mathcal{E}$ (the polynomials in F),
  - `Pps`: set of projection polynomials defining $\mathcal{E}$,
  - `Pfs`: set of projection factors (factorised projection polynomials)

defining $\mathcal{E}$.

- Lines 10-23 compute the critical points of projections of 1-dimensional cells in `C1s` onto one-dimensional coordinate subspaces $\text{span}\{x_i\}, 1 \leq i \leq r$.
- Lines 37-42 compute the critical points of projections of 2-dimensional cells an `C2s` onto two-dimensional coordinate subspaces $\text{span}\{x_i, x_j\}, 1 \leq i < j \leq n$.
- The `JACOBI(r, f, j, Hs, Is)` function computes a Jacobi determinant

$$\det \begin{pmatrix} \frac{\partial h_1}{\partial x_{i_1}} & \cdots & \frac{\partial h_1}{\partial x_{i_k}} & \frac{\partial h_1}{\partial x_j} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial h_k}{\partial x_{i_1}} & \cdots & \frac{\partial h_k}{\partial x_{i_k}} & \frac{\partial h_k}{\partial x_j} \\ \frac{\partial f}{\partial x_{i_1}} & \cdots & \frac{\partial f}{\partial x_{i_k}} & \frac{\partial f}{\partial x_j} \end{pmatrix}$$

  with parameters

  - $r \in N$,
  - $f \in \mathbb{Z}[x_1, \ldots, x_r]$,
  - $j \in \mathbb{Z}_{\geq 0}$
  - $\text{Hs} = (\text{h}_1, \ldots, \text{h}_\text{k}), \text{h}_\text{i} \in \mathbb{Z}[\text{x}_1, \ldots, \text{x}_\text{r}], 1 \leq \text{i} \leq \text{k}$
  - $\text{Is} = (\text{i}_1, \ldots, \text{i}_\text{k}) \in \mathbb{Z}^\text{k}$.

  This is similar to the partial differential operator from Definition 3.2, except that if $j = 0$, the last row and column is omitted from the matrix.
- The rest of the function sets up the input polynomials, projection polynomials and projection factors necessary to define a sign-invariant CAD compatible with V and critical points of projections of one- and two-dimensional cells onto one- and two-dimensional coordinate subspaces. Projections of the new polynomials computed in the previous two steps are computed and added to the projection polynomial and projection factor sets. The lifting phase proceeds, using these polynomials, us usual.

7.1.5 Monotone

The function `MONOTONE` is responsible for computing the set of refinement points necessary to construct a CAD monotone with respect to $V_1, \ldots, V_k$.

- **Input:**
  - `Word r`: $r \in N$,
  - `Word D`: sign-invariant CAD of $\mathbb{R}^r$,
  - `Word A`: set of projection polynomials for `D`,
  - `Word J`: projection factor set for `D`.
- **Output:**

- `Word R`: set of refinement points,
- Refinement polynomials are also added to `A` and `J`.

An excerpt of the code is presented below.

```
Word QepcadCls::MONOTONE(Word* A_, Word* J_, Word D, Word r)
{
    // to store unfactorised refinement polynomials, in the form
    ↪  (Index, Sample, Ps)
    Word Rs = NIL;

    // consider each true cell in D
    // note: in practice, it is slower and takes more memory to
    ↪  do a walk of the CAD, saving polynomials along the way.
    Word TrueCells, junk;
    LISTOFCWTV(D, &TrueCells, &junk);
    Word Ds = LELTI(D, CHILD); // cells of D, for searching cells
    Word AA = INV(LCOPY(*A_)); // in reverse order, to match
↪   SIGNPF. for later.

    while (TrueCells != NIL) {
        Word C;
        ADV(TrueCells, &C, &TrueCells);

        // C has dimension two, then IJ < Ik are the positions in
        ↪  I where the component is equal to 1. otherwise skip
        Word I = LELTI(C, INDX);
        Word Ij, Ik;
        Word d = TwoDimIndex(I, &Ij, &Ik);
        if (d > 2) {
            // fail: only implemented for dimension at most 2
            FAIL("source/ticad/MONOTONE", "cell dimension greater
↪   than 2 not supported.");
        } else if (d < 2) {
            // cells with dimension 0 and 1 are already monotone.
            continue;
        }

        Word Ij1 = Ij - 1;
        Word nv = r - Ij1;

        // C0 := proj_{j-1}(C) is a 0-dimensional cell
        ↪  (c_1,...,c_{j-1})
        Word C0, S0, I0;
        Word ij = LELTI(I, Ij);
        if (Ij == 1) { // base CAD is D
```

```
            C0 = D;
            S0 = LIST3(PMON(1,1), LIST2(0,0), NIL);
            I0 = NIL;
        } else { // sub-CAD on top of some 0-cell
            C0 = FindByIndex(Ds, I, Ij1, 1);
            S0 = LELTI(C0, SAMPLE);
            I0 = LELTI(C0, INDX);
        }

        // top and bottom of proj_k(C) are one-dimensional
        ↪  sections by definition.
        Word I1 = LCOPY(I);

        // top: (i_1,...,i_k + 1)
        SLELTI(I1, Ik, LELTI(I, Ik) + 1);
        Word CT = FindByIndex(Ds, I1, Ik, 1);

        // bottom: (i_1,...,i_k - 1)
        SLELTI(I1, Ik, LELTI(I, Ik) - 1);
        Word CB = FindByIndex(Ds, I1, Ik, 1);

        // Top and bottom of 1-sector, level J
        Word C0C = LELTI(C0, CHILD);
        Word I01 = CCONC(I0, LIST1(ij));
        Word S0B = NIL, S0T = NIL, Endpoints;
        if (ij > 1) {
            GETSAMPLEK(
                Ij,
                LELTI(LELTI(C0C, ij - 1), SAMPLE),
                &junk,
                &S0B
            );

            S0B = SECOND(S0B);
        }

        if (ij < LENGTH(C0C)) {
            GETSAMPLEK(
                Ij,
                LELTI(LELTI(C0C, ij + 1), SAMPLE),
                &junk,
                &S0T
            );

            S0T = SECOND(S0T);
```

```
80          }
81
82          Endpoints = LIST2(S0B, S0T);
83
84          // find polynomials in sub-CAD
85          // (note they will be in Z[x_i,...,x_l] after
            ↪  substitution):
86          // Gs = g_2,...,g_{k-1} define proj_{k-1}(C).
87          // TODO can we save the polynomials, maybe using cell
            ↪  position, to save duplicating work?
88          Word Gs = ZeroPolsSub(AA, r, C, Ij + 1, Ik - 1, S0, Ij1,
    ↪  nv);
89
90          // Fk (f_{k,T}, f_{k,B}) are 0 on CT and CB respectively.
91          Word FT, FB;
92          if (CT != NIL) FT = FIRST(ZeroPolsSub(AA, r, CT, Ik, Ik,
            ↪  S0, Ij1, nv));
93          if (CB != NIL) FB = FIRST(ZeroPolsSub(AA, r, CB, Ik, Ik,
            ↪  S0, Ij1, nv));
94
95          // Fs = (f_{K+1},...,f_n) is a map from proj_{k}(C) to
            ↪  R^{n-k}, of which C is the graph.
96          Word Fs = ZeroPolsSub(AA, r, C, Ik + 1, r, S0, Ij1, nv);
97
98          // perform refinement
99          if (CT != NIL) {
100             STOREPOLYNOMIALS(Refinement(nv, Gs, FT, Fs), I01, S0,
    ↪  Endpoints, &Rs);
101         }
102
103         if (CB != NIL) {
104             STOREPOLYNOMIALS(Refinement(nv, Gs, FB, Fs), I01, S0,
    ↪  Endpoints, &Rs);
105         }
106     }
107
108     return REFINEMENTPOINTS(r, Rs, A_, J_);
109 }
```

The function considers each 2-dimensional cell C of $\mathcal{E}$ such that $C \subset V$. C is an $(i_1, \ldots, i_r)$-cell such that $i_j = i_k = 1$ and all other elements of the index are equal to zero.

- On line 32, $\mathbf{c}_0 = \text{proj}_{\mathbb{R}^{i_j - 1}}(C)$, the zero-cell above which C is a $(1, i_{j+1}, \ldots, i_r)$-cell.
- $\text{proj}_{\mathbb{R}^k}(C)$ is a section cell, on lines 50 and 54, $\text{proj}_{\mathbb{R}^k}(C)_T$ and $\text{proj}_{\mathbb{R}^k}(C)_B$,

the section cells defining the top and bottom of this sector, respectively, are retrieved from the decomposition induced by $\mathcal{E}$ on $\mathbb{R}^k$ by their positional indices.

- Line 82 retrieves the endpoints of the 1-dimensional sector $\text{proj}_{\mathbb{R}^j}(C) = (a, b)$.
- Line 88 retrieves and substitutes the polynomials

$$\text{Gs} := \{g_\ell \in \mathbb{Q}[x_j, \ldots, x_\ell] \mid g_\ell(\mathbf{x}) = 0, \mathbf{x} \in C, j < \ell < k\}.$$

- Lines 92 and 93 find and substitute polynomials f_T and f_B in $\mathbb{Q}[x_j, \ldots, f_k$ respectively such that $f_T(\mathbf{x}) = 0 \forall \mathbf{x} \in \text{proj}_{\mathbb{R}^k}(C)_T$ and $f_T(\mathbf{x}) = 0 \forall \mathbf{x} \in \text{proj}_{\mathbb{R}^k}(C)_T$
- Line 96 finds and substitutes polynomials

$$\text{Fs} := \{f_\ell \in \mathbb{Q}[x_j, \ldots, x_\ell] \mid f_\ell(\mathbf{x}) = 0 \forall \mathbf{x} \in C, k < \ell < r\}.$$

- The remainder of the function computes the refinements of $\text{proj}_{\text{span}\{k\}}(C)_T$ and $\text{proj}_{\text{span}\{k\}}(C)_T$, if they exist, such that C is monotone.
- `STOREPOLYNOMIALS` adds refinement polynomials to the projection polynomial and factor sets and `REFONIMENTPOINTS` uses the factorised refinement polynomials to compute the set of refinement points.

The function `Refinement` is responsible for computing the refinement polynomials using an iterative application of the method of Lagrange multipliers.

- **Input:**
  - `Gs`, `P`, `Fs` as defined above, for $\text{proj}_{\mathbb{R}^k}(C)_T$ or $\text{proj}_{\mathbb{R}^k}(C)_B$.
- **Output:**
  - a set of univariate polynomials in $\mathbb{Q}[x_j]$ such that C is monotone.

```
1  Word Refinement(Word r, Word Gs, Word P, Word Fs)
2  {
3      Word Rs, Q, i, k, Is;
4
5      // generate sequence Is = (1,...,k-1)
6      i = 1;
7      k = LENGTH(Gs) + i;
8      Is = NIL;
9      while (i < k) {
10         Is = COMP(i, Is);
11         ++i;
12     }
13
14     Rs = NIL; // refinement polynomials
15
16     // semi-monotone: critical points of P subject to Gs
17     if (Gs != NIL) {
18         Rs = CONC(LagrangeRefinement(r, P, k, Gs, Is), Rs);
```

```
19      }
20
21      // monotone: each F_i subject to Gs, F_1,...F_{i-1} for each
        ↪  i
22      while (Fs != NIL) {
23          ADV(Fs, &Q, &Fs);
24          Gs = COMP(P, Gs);
25          Is = COMP(k, Is);
26          P = Q;
27          ++k;
28
29          // monotone, index k: critical points of Q subject to Gs
30          Rs = CONC(LagrangeRefinement(r, Q, k, Gs, Is), Rs);
31      }
32
33      // solve for x_1
34      return Rs;
35  }
36
37  Word LagrangeRefinement(Word r, Word f, Word i, Word Gs, Word Is)
38  {
39      Word Q = JACOBI(r, f, i, Gs, Is);
40
41      // check for zero polynomial. no solutions
42      if (Q == 0) return NIL;
43
44      // factorise
45      Word junk, Qs, Q1;
46      IPFACDB(r ,Q, &junk, &junk, &Qs);
47      while (Qs != NIL) {
48          ADV(Qs, &Q1, &Qs);
49
50          Gs = COMP(SECOND(Q1), Gs);
51      }
52
53      // simplify Gs by constructing a Groebner basis is supported
54      // this step makes solving the jacobi determinant for x_1 a
        ↪  lot quicker in practice.
55      if (GVCAP->supports("GROEBNER")) {
56          Gs = GVCAP->GROEBNER(Gs, NIL, r);
57      }
58
59      // find solution in x_1 by projection
60      // Gs now includes factors of the jacobi determinant
61      return ProjSolve(r, Gs);
```

```
}
```

`LagrangeRefinement` is applied to `P` and then to each polynomial in `Fs`. On each step, the Jacobi determinant Q, defined in Equation (5.6), is computed. Using CAD projection, a set of univariate polynomials defining the x_1-coordinates of the solutions to $Q = 0$ is then returned. Since the degree of Q may be high, it was noticed that computing a Groebner basis, using Singular, for Q and `Gs` significantly sped up the computation of the univariate polynomials (from several seconds to less than a second).

Once the set of refinement points has been computed, the CAD is refined using the method described in Section 7.1.7.

7.1.6 Frontier Condition

The frontier condition is obtained by using the generalisation of Lazard's method for lifting with bad points, as described in section 6.3, the entry point for computing a CAD with frontier condition is `FRONTIER`. This function identifies bad zero-dimensional cells and adds refinement points, in the same format as that used in the construction of monotone cells.

Input:

- `Word r` $\in \mathbb{N}$,
- `Word k` $\in \mathbb{N}, k \leq r$ the level of the cell `D` $+1$,
- `Word D`: \$00 $-$ *dimensionallevel*$-$(k-1)\$ QEPCAD cell,
- `Word As` $= (\mathcal{A}_k, \ldots, \mathcal{A}_r)$: the projection factor set,

Output:

- `Word A`, `Word J`, `Word RPs`: refinement points for bad cells contained in the sub-CAD above `D` are added.

```
Word QepcadCls::FRONTIER(Word r, Word k, Word D, Word As, Word*
↪  A_, Word* J_, Word* RPs_)
{
    Word Ch, TrueCells, junk, C1, C1_B, C1_T, C;
    Ch = LELTI(D, CHILD);

    // if r < 3, frontier condition is obtained automatically, if
    ↪  no children then nothing to do.
    if (r < 3 || Ch == NIL) return D;

    ADV(Ch, &C1, &Ch);

    // only one sector, bad cells are not possible.
    if (Ch != NIL);

```

```
14      Word RefinedCells = NIL;
15      C1_B = NIL, C1_T = NIL;
16      while (Ch != NIL) {
17          // Ch = (top, next sector, ...)
18          // cells will be taken in pairs.
19          ADV(Ch, &C1_T, &Ch);
20          FRONTIER(r, k+1, C1_T, As, A_, J_, RPs_);
21
22          // get true (1,...)-cells
23          LISTOFCWTV(C1, &TrueCells, &junk);
24          while (TrueCells != NIL) {
25              Word d, j, SI;
26              ADV(TrueCells, &C, &TrueCells);
27
28              d = TwoDimIndex(LELTI(C, INDX), &junk, &j);
29              if (d != 2 || j == r) continue;
30
31              // find indices of polynomials which are zero on C.
32              SI = REDI(SignatureIndex(LELTI(C, SIGNPF)), k);
33
34              ProcessBadCells(r, C1_B, As, k, j, SI, &RefinedCells,
    ↪  A_, J_, RPs_);
35              ProcessBadCells(r, C1_T, As, k, j, SI, &RefinedCells,
    ↪  A_, J_, RPs_);
36          }
37
38          // next sector.
39          ADV(Ch, &C1, &Ch);
40          C1_B = C1_T; // one sector's top is the next sector's
    ↪  bottom.
41      }
42
43      return D;
44  }
```

- Lines 6-12 determine if bad cells are possible. I.e., if the dimension of the sub-CAD above `D` is greater than 3 and if there is more than one child cell, since bad cells can only occur above 0-cells.
- A note is kept of cells which were already refined (Line 14). This information is stored in the form $(j, i_1, \ldots, i_{k+1})$, where $(i_1, \ldots, i_{k+1})$ is the positional index of the cell C to be refined and j the index of the level-$(k+1)$ polynomial which vanishes identically over C. This prevents the program computing the same refinement points multiple times, as the same bad cell may lie in the projection of the boundary of multiple 2-dimensional cells.
- The loop (Lines 24-36) considers each 2-dimensional section cell C in this

sub-CAD, with index $(0,\ldots,0,m_k,0,\ldots,0,m_\ell,0,\ldots,0)$ in the CAD of $\mathbb{R}^r$ (ambient space) where $m_k = m_\ell = 1$.

- `SI` (Line 31) is a list $(i_{k+1},\ldots,i_r)$ where each $i_j \in \mathbb{Z}$ is equal to the index of the first polynomial in $\mathcal{A}_j$ which is equal to zero on C, or -1 if no such polynomial exists. In particular, $i_k = i_\ell = 1$.
- Bad cells, in $\mathrm{fr}\,(\mathrm{proj}_{\mathbb{R}^\ell}(C))$, may lie in the sub-CADs above the $(0,\ldots,0)$-cells $\mathrm{proj}_{\mathbb{R}^k}(C)_B$ and $\mathrm{proj}_{\mathbb{R}^k}(C)_T$. The function `ProcessBadCells` (called on Lines 34 and 35) identifies bad cells and adds refinement points. This function is described next.

Input:

- `Word r` $\in \mathbb{N}$,
- `Word C`: 0-dimensional level-k QEPCAD cell,
- `Word As` $= (\mathcal{A}_1,\ldots,\mathcal{A}_r)$ is the (entire) projection factor set.
- `Word i`, `Word j` $\in \mathbb{N}$: elements of $(i_1,\ldots,i_r)$, the index of a two-dimensional section cell C, which are equal to 1.
- `Word S` $= (j_1,\ldots,j_r)$, indices of projection factors which are equal to zero an C.

Output:

- `RefinedCells`: a note of all cells which require refinement,
- `Word A`, `Word J`, `Word RPs`: refinement points for bad cells contained in the sub-CAD above `D` are added.

```
1  void ProcessBadCells(Word r, Word C, Word As, Word i, Word j,
   ↪  Word S, Word *RefinedCells_, Word *A_, Word *J_, Word* RPs_)
2  {
3      if (C == NIL) return; // base case, nothing to do
4
5      Word s;
6      ADV(S, &s, &S);
7
8      Word Ch = LELTI(C, CHILD);
9      if (Ch == NIL) return;
10
11     Word C1 = NIL, C2 = NIL, JT = NIL, JB = NIL;
12     Word level = LELTI(C, LEVEL) + 1; // level of children
13     Word sample = LELTI(C, SAMPLE);
14     bool section = false;
15     while (true) {
16         Word s1, SC1;
17         C1 = C2;
18         JB = JT;
19         section = !section;
20
21         // loop exit check. reached end of list.
```

```
22          if (Ch == NIL && C1 == NIL) break;
23
24          if (Ch != NIL) {
25              ADV(Ch, &C2, &Ch);
26          } else {
27              C2 = NIL;
28          }
29
30          if (!section && C2 != NIL) {
31              Word SM, SJ;
32              GETSAMPLEK(level, LELTI(C2, SAMPLE), &SM, &SJ);
33              JT = RNQ(RNSUM(FIRST(SJ), SECOND(SJ)), RNINT(2));
34          } else if (!section) {
35              JT = NIL;
36          }
37
38          if (C1 == NIL) continue;
39
40          SC1 = LELTI(C1, SIGNPF);
41          s1 = IndexOfFirstZero(FIRST(SC1));
42
43          Word Idx = LELTI(C1, INDX);
44          Word I1x = COMP(s1, Idx); // bit like a hash, polynomial
    ↪  plus cell index to indicate that a cell has been refined
45
46          // a bad cell is a (0,...,0,1)-cell of level greater than
            ↪  J, with matching sign which has not yet been refined
47          if (!section && level > j && s == s1 && LSRCH(I1x,
            ↪  *RefinedCells_) == 0) {
48              Word RP = LazardLifting(
49                  level,
50                  sample,
51                  As,
52                  COMP(s1, SignatureIndex(RED(SC1))),
53                  i,
54                  j
55              );
56
57              ADDREFINEMENTPOINTS(Idx, sample, RP, LIST2(JB, JT),
    ↪  A_, J_, RPs_);
58              *RefinedCells_ = COMP(I1x, *RefinedCells_);
59          }
60
61          if (section && (level == j || s == s1)) {
62              ProcessBadCells(r, C1, As, i, j, S, RefinedCells_,
    ↪  A_, J_, RPs_);
```

```
63          }
64      }
65  }
```

This function proceeds by induction, recursing on zero-dimensional cells above `C` whose signature (indices of projection factors which are equal to zero) matches `S`. Note that coordinates i and j of `S` are equal to -1, so any zero-dimensional cells at those levels are candidates for being bad cells. Lines 45-59 process possible bad cells. A bad cell is a $(0, ..., 0, 1)$-cell of level greater than j, which has not yet been refined, and such that a polynomial vanishes identically on it. The function `LazardLifting` uses `Singular` to compute the Groebner basis for the saturations of two ideals

$$I_1 := \langle f, g_{j_1}, \ldots, g_{j_{k-3}}, g_{\ell_2}, 1 - z g_{\ell_1} \rangle \cap \mathbb{Z}[x_1, \ldots, x_k]$$

and

$$I_2 := \langle f, g_{j_1}, \ldots, g_{j_{k-3}}, g_{\ell_1}, 1 - z g_{\ell_2} \rangle \cap \mathbb{Z}[x_1, \ldots, x_k],$$

as described in Section 6.3. To obtain a refinement polynomial in $\mathbb{Z}[x_r]$, the sample point of the bad $(0, \ldots, 0)$-cell is substituted. `ADDREFINEMENTPOINTS` is then called, taking these polynomials as input, the add the refinement points above this bad cell.

7.1.7 Refinement

Given the CAD $\mathcal{D}$ of $\mathbb{R}^n$ constructed in the lifting phase, both `MONOTONE()` and `FRONTIER()` produces a set of refinement points

$$\mathcal{R} := \{\mathcal{R}_{\mathbf{b}} = \{c_1, \ldots, c_t\} \subset \mathbb{A} \mid \mathbf{b} \in \mathbb{R}^{k-1}\}.$$

where $\mathbf{b} \in \mathbb{R}^{k-1}$ is a $(0, \ldots, 0)$-cell in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^k$. The function `REFINE()` computes refinements of $\mathcal{D}$ to be compatible with these points. Each refinement point $(b_1, \ldots, b_{k-1}, c) \in \mathbb{A}^k$, such that $(b_1, \ldots, b_k)$ is a 0-dimensional cell, $\mathbf{b}$, in the CAD induced by $\mathcal{D}$ on $\mathbb{R}^{k-1}$ is stored separately, is stored separately as a pair

$$((b_1, \ldots, b_{k-1}, c), (m_1, \ldots, m_{k-1}, m_k))$$

where $(b_1, \ldots, b_{k-1})$ is the sample point, and $(m_1, \ldots, m_{k-1})$ is the positional index of the 0-cell $\mathbf{b}$ and c refines the (1)-cell in the sub-CAD of $\mathcal{D}$ above $\lfloor$ with positional index (m_k). $(b_1, \ldots, b_{k-1}, c)$ is stored as a QEPCAD sample point. If all of $(b_1, \ldots, b_{k-1})$ are rational, or are elements of the same algebraic extension, $\mathbb{Q}(\alpha)$, as the sample point of the cell $\mathbf{b}$, as c, then $(b_1, \ldots, b_{k-1}, c)$ is a QEPCAD sample point in the primitive representation. If $(b_1, \ldots, b_{k-1}) \in \mathbb{Q}(\alpha)$ and $c \notin \mathbb{Q}(\alpha)$ then $(c_1, \ldots, b_{k-1}, c)$ is a QEPCAD sample point stored in extended representation. The refinement point c is a root of a polynomial $f \in \mathbb{Q}[x_k]$, computed by `MONOTONE()` or `FRONTIER()`. This polynomial is stored in the projection polynomial set, and its factors in the projection factor set.

The function `REFINE` computes the refinements.

Input:

- `Word k` $\in \mathbb{N}$,
- `Word D` a cad of $\mathbb{R}^{k-1}$,
- `Word A` $= (\mathcal{A}_k, \ldots, \mathcal{A}_n)$: set of refinement points,
- `Word PF` $= (\mathcal{F}_k, \ldots, \mathcal{F}_n)$: set of projection factors.

Output:

- `Word D'`, a refinement of D compatible with the refinement points in $\mathcal{A}_k$.

QEPCAD represents a CAD $\mathcal{D}$ of $\mathbb{R}^n$ as a tree of cells. The root is the unique cell **0** in the CAD of $\mathbb{R}^0$ and its children are the cells of the CAD induced by $\mathcal{D}$ on $\mathbb{R}^1$. As one might expect, given a QEPCAD cell C in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^k$, its children are the cells of the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{k+1}$ which project on C. QEPCAD constructs a "partial CAD". This means that there may be cells of level $k < n$ with no children. In other words, if C is an $(i_1, \ldots, i_k)$-cell in the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^k$ with no children, then ${\text{proj}_{\mathbb{R}^k}}^{-1}(C)$ is an $(i_1, \ldots, i_k, 1, \ldots, 1)$-cell of $\mathcal{D}$.

Refine proceeds by induction on CAD cells by "walking" the CAD (in the same way as a depth-first-search), identifying cells which need refinement by checking the level-k refinement points, computing this refinement and returning the refined CAD.

```
1  Word QepcadCls::REFINE(Word k, Word D, Word A, Word PF)
2  {
3      // no children to refine.
4      Word Ch = LELTI(D, CHILD);
5      if (Ch == NIL) {
6          return D;
7      }
8
9      Word k1 = k-1;
10     Word A1;
11     ADV(A, &A1, &A); // deconstruct A. A1 is the set of level k+1
   ↪   polys
12
13     // find the new PO_REFINE polynomials.
14     Word Ps = NIL, I = LELTI(D, INDX);
15     while (A1 != NIL) {
16         Word P;
17         ADV(A1, &P, &A1);
18
19         Word J = LELTI(P, PO_REFINEMENT);
20         if (FIRST(J) != -1 && EQUALK(k1, J, I)) { // list
           ↪   equality check not needed, since same cell index
           ↪   pointer is used
```

```
21              Ps = COMP(COMP(LELTI(J, k1 + 1), LELTI(P, PO_POLY)),
    ↪  Ps);
22
23              // mark the refinement point as "used"
24              SLELTI(P, PO_REFINEMENT, COMP(-1, J));
25          }
26      }
27
28      // do refinement if the list of Ps is non-empty
29      if (Ps != NIL) {
30          Ch = RefineSubcad(k, Ch, Ps, PF);
31          SLELTI(D, CHILD, Ch);
32      }
33
34      // no more refinement polynomials
35      if (A == NIL) {
36          return D;
37      }
38
39      // walk the CAD, sections only.
40      Word C, junk;
41      ADV(Ch, &junk, &Ch);
42      PF = RED(PF);
43      while (Ch != NIL) {
44          ADV2(Ch, &C, &junk, &Ch);
45
46          C = REFINE(k+1, C, A, PF);
47      }
48
49      return D;
50  }
```

- Lines 13-26 identify the refinement polynomials. Note that -1 is appended to the cell index (stored in `PO_REFINEMENT`) to indicate that the refinement by that point has been completed, marking the point as "used".
- Line 31 computes the refinement of the children projecting on `D` by calling `RefineSubcad()`.
- Line 40 onwards completes the depth-first-search walk of the (refined) children.

The function `RefineSubcad` does all the heavy lifting.

Input:

- `Word k` $\in \mathbb{N}$
- `Word Ch`: list of children – level-k QEPCAD cells, in ascending order of sample points, which project on a single level-$k-1$ QEPCAD cell,

- `Word Ps`: list of refinement points (in ascending order) of the children,
- `Word PFs`: projection factor set $(\mathcal{A}_{k+1}, \ldots, \mathcal{A}_{\setminus})$, used for recomputing sample points.

Output:

- `Word Ch'`: `Ch`, refined to be compatible with refinement points `Ps`, indices, sample points and data for all child cells has been updated.

```
Word RefineSubcad(Word k, Word Ch, Word Ps, Word PFs)
{
    Word i1, Ch1, C, i, c, C0;
    i1 = LELTI(LELTI(FIRST(Ch), INDX), k);

    while (Ps != NIL) {
        Word PM, PI, J;
        NextPolynomial(Ps, &PM, &PI, &J, &i, &Ps);

        // find cell with index i.
        Word j = 0, S0M = NIL, S0I = NIL;
        Ch1 = Ch, i = i - 1;  // we are actually looking fro
↪   sector bottom
        while (i > 0 || Ch1 != NIL) {
            ADV(Ch1, &C, &Ch1);

            // original cell indices are preserved until the last
            ↪  moment.
            // we cannot just count in case a cell was refined.
            j = LELTI(LELTI(C, INDX), k);

            if (j == i) {
                // C is the cell bottom. get its sample k
                GETSAMPLEK(-1, LELTI(C, SAMPLE), &S0M, &S0I);

                break;
            }
        }

        // first cell in Ch is to be refined, S1M, S1J is the
        ↪  k-th coordinate of the sample point of C_B
        bool refine_after = false; // do we need to refine C3?
        Ch1 = RefineCell(k, Ch1, PM, PI, S0M, S0I, PFs,
↪   &refine_after);

        ADV(RED2(Ch1), &C, &Ch1);
        // now FIRST(Ch) is the top of C, if C is bounded from
        ↪  above
```

```
34
35          // might be that we need to refine C3
36          if (!refine_after) {
37              // don't forget to add missing signpfs
38              ADDSIGNPF(k, C, FIRST(PFs));
39
40              continue;
41          }
42
43          if (Ch1 == NIL) { // not bounded from above. easy!
44              c = RNSUM(SECOND(PI), RNINT(1));
45          } else {
46              GETSAMPLEK(-1, LELTI(FIRST(Ch1), SAMPLE), &SOM,
    ↪  &SOI);
47              c = RNQ(RNSUM(SECOND(PI), FIRST(SOI)), RNINT(2));
48              RNWRITE(SECOND(PI)); SWRITE(" ");
    ↪  RNWRITE(FIRST(SOI)); SWRITE(" ");  RNWRITE(c);
49          }
50
51          SETSAMPLE(C, PMON(1,1), LIST1(c), RED(PFs));
52          ADDSIGNPF(k, C, FIRST(PFs));
53      }
54
55      // finally update indices.
56      i = i1 - 1, Ch1 = Ch;
57      while (Ch1 != NIL) {
58          ++i;
59          ADV(Ch1, &C, &Ch1);
60
61          SETINDEXK(C, k, i);
62      }
63
64      return Ch;
65  }
```

- `NextPolynomial()`, called on, e.g., Line 9, is a helper function for retrieving the next refinement point. Recall that $\mathbf{b} = (b_1, \ldots, b_{k-1}, c)$ is a QEPCAD sample point such that `PI` is its isolating interval and `PM` the minimal polynomial for c. If c is rational, $I = (c, c)$. `J` is the isolating interval containing $c \in \mathbb{A}$. `j` is the index of the child cell which requires refinement.
- Suppose that C is the cell with index j. Then `SOM` and `SOI`, set on line 24, are the minimal polynomial and isolating interval, respectively, for the sample point of C_B. The function `GETSAMPLEK` is a helper function responsible for retrieving this information. The first argument indicates the coordinate to retrieve, with -1 being shorthand for last coordinate.

- `RefineCell`, called on line 32, takes a list of cells, $(C, C_T, \ldots, C_\ell)$, where C is the cell to be refined and returns a list $(C_1, C_2, C_3, C_T, \ldots, C_\ell)$. The sample points are updated for C_1 and C_2. The flag `refine_after` is set to `true` if the sample point of C_3, which is still equal to the sample point of C at this point, is incorrect.
- Some new projection factors, whose roots are the refinement points, were added to the projection factor set, but cell signatures have not yet been updated to reflect this. `ADDSIGNPF` (called on lines 40 and 54) is responsible for ensuring that the signs of these polynomials are attached to refined cells. If a cell is not refined, this information will not be needed.
- Lines 45-53 update the sample point of C_3, if needed.
- During the loop in which cells are refined (Lines 7-55), the positional indices for refined cells are not updated. This is so that the index attached to refinement points still refers to the correct positional index. Lines 57-64 updates the positional indices for all children, using the helper function `SETINDEXK`.

The function `RefineCell` is now presented.

Input:

- `Word k$` $\in \mathbb{N}$,
- `Word Cs` list of cells $(C, C_T, \ldots)$, such that C is the (1)-cell to be refined.
- `Word PM`: minimal polynomial defining refinement point c (x if rational),
- `Word PI`: isolating interval for refinement point,
- `Word c`,
- `Word PFs` $= (\mathcal{F}_k, \ldots, \mathcal{F}_n)$ is the set of projection factors, needed to update sample points of children.

Output:

- `Word Ch'`: refined list of children $(C_1, C_2, C_3, C_T, \ldots)$, such that C is refined into three cells C_1, C_2, C_3,
- `bool rc`: whether the refinement caused the sample point of the (1)-cell C_3 to be incorrect.

```
1  Word RefineCell(Word k, Word Cs, Word PM, Word PI, Word c, Word
   ↪  PFs, bool* rc)
2  {
3      Word Cs2 = Cs;
4
5      // split projection factors
6      Word PF1;
7      ADV(PFs, &PF1, &PFs);
8
9      // Let C = (a,b). C becomes (a,s), C2 becomes new cell s and
       ↪  C3 new cell (s,b)
10     Word C1;
```

```
11      ADV(Cs, &C1, &Cs);
12      Word C2 = LDCOPY(C1);
13      Word C3 = LDCOPY(C1);
14
15      SWRITE("Refine cell "); LWRITE(LELTI(C1, INDX));
    ↪   SWRITE("\n");
16
17      // update sample
18      // we will need to update only two of the cells, as the
        ↪  existing sample will be correct for one of them
19      Word SQ, SJ;
20      GETSAMPLEK(-1, LELTI(C1, SAMPLE), &SQ, &SJ);
21
22      Word sign = COMPARE(SQ, &SJ, PM, &PI);
23      // -1: C1 is correct, 0: C2 is correct, +1: C3 is correct.
24
25      if (sign != -1) { // need to update C1 ...
26          // we need a rational number in between the bottom C1 and
            ↪  the refinement point.
27          Word c;
28          if (SOM == NIL) { // not bounded from below. easy!
29              c = RNSUM(FIRST(PI), RNINT(-1));
30          } else {
31              c = RNQ(RNSUM(SECOND(SOI), FIRST(PI)), RNINT(2));
32          }
33
34          SETSAMPLE(C1, PMON(1,1), LIST1(c), PFs);
35      }
36
37      if (sign != 0) { // need to update C2 ...
38          // to the new "refinement point" given
39          SETSAMPLE(C2, PM, PI, PFs);
40      }
41
42      // add missing signs of projection factors
43      ADDSIGNPF(k, C1, PF1);
44      ADDSIGNPF(k, C2, PF1);
45
46      // we may will need to update the sample of C3, but this is
        ↪  done later.
47      *rc = sign != 1;
48
49      // append new cells
50      SRED(Cs2, COMP2(C2, C3, Cs));
51      return Cs2;
```

```
52 }
```

This function lets $C = C_1$ and inserts two copies C_1, C_2 of C into the list of children. It then updates sample points, if needed (the sample point of C is correct for exactly one of C_1, C_2 or C_3), and adds missing signs of projection factors whose roots are the refinement points. The comments explain how the function works. The helper function `LDCOPY` (list "deep" copy) makes a copy of each element in a list, proceeding by induction if the element is a list. This is required as C_2, C_3 should not contain any pointers to data in C. The functions `SETINDEXK`, `GETSAMPLEK` and `SETSAMPLEK` were discussed in relation to `RefineSubcad`.

7.1.8 Obtaining defining formulas for every cell

As discussed in Section 2.2.5, the projection factors may not be sufficient to produce a quantifier-free Boolean formula for every cell C in a CAD. We would like to be able to examine every cell $C \subset V_1, \ldots, V_k$ using its defining formula. As such, a new command, `d-cell-tarski` has been implemented. This uses Brown's solution formula construction algorithm for guaranteed solution formula construction. Given a CAD $\mathcal{D}$, the algorithm builds an RCad $\mathcal{D}_R$, which is a refinement of $\mathcal{D}$ such that each cell can be expressed as a disjoint union of basic semialgebraic sets using only polynomials from the projection factor set. In most cases, each cell can be written as a basic semialgebraic set (conjunction of sign conditions on polynomials), but occasionally, a cell C of $\mathcal{D}$ is refined in $\mathcal{D}_R$. In both cases, every cell C of $\mathcal{D}$ can be expressed as a quantifier-free Boolean formula

$$F_1 \vee \ldots \vee F_k$$

where $F_1, \ldots, F_k$ are conjunctions of polynomial equations and inequalities defining the cells $C_1, \ldots, C_k$ of $\mathcal{D}_R$ such that $C = C_1 \cup \ldots \cup C_k$. The RCad is constructed when `d-cell-tarski` is used and then it is cached. If refinements to the original CAD are performed, e.g., because of `MONOTONE` or `FRONTIER`, the RCad is invalidated and must be reconstructed (see Line 77 of the `QEPCAD` function).

To demonstrate `d-cell-tarski`, construct a CAD compatible with

$$V := \{x = y^2 - 2y\}.$$

QEPCAD produces the following projection factors

```
d-proj-fac
P_1,1  = fac(J_1,1) = fac(dis(A_2,1))
       = 27 x^2 - 32


A_2,1  = input
       = y^3 - 2 y - x
```

Let $a, b \in \mathbb{A}$ be the unique roots of $27x^2-32$ in the intervals $(-1115/1024, -557/512)$ and $(557/512, 1115/1024)$ respectively. The cylinder $(a, b) \times \mathbb{R}$ contains three true cells, on which $x = y^3 - 2y$ is equal to zero. It is not possible to represent these three cells as QFFs containing only the projection factors, although their positional indices and sample points may be used to distinguish them. We show the output of `d-cell-tarski` for the true cell with positional index $(3, 2)$.

```
d-cell-tarski(3,2)
---------- Information about the cell (3,2)

Dimension (1,0) 1----------


Signs of projection factors -------------------------

Level 1
  27 x^2 - 32 < 0
Level 2
  y^3 - 2 y - x = 0


Signs of (guaranteed definable ) projection factors

*** Initialising the RCAD. ***

This cell consists of 3 cells in the RCAD.

Index in RCAD: (5,2)
Level 1
  27 x^2 - 32 < 0
  x > 0
Level 2
  y^3 - 2 y - x = 0
  3 y^2 - 2 > 0
  y < 0

Index in RCAD: (4,2)
Level 1
  27 x^2 - 32 < 0
  x = 0
Level 2
  y^3 - 2 y - x = 0
  3 y^2 - 2 > 0
  y < 0

Index in RCAD: (3,2)
```

```
Level 1
  27 x^2 - 32 < 0
  x < 0
Level 2
  y^3 - 2 y - x = 0
  3 y^2 - 2 > 0
  y < 0


Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between -2 and -1
      = -1.4142135624-

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = alpha
             = -1.4142135624-


--------------------------------------------------
```

The cell C has been refined into three cells in the RCad, and it can be represented by the QFF

$$\begin{aligned}&(g < 0 \ \wedge \ x > 0 \ \wedge \ f = 0 \ \wedge \ y < 0) \vee \\ &(x = 0 \ \wedge \ f = 0 \ \wedge \ y < 0) \vee \\ &(g < 0 \ \wedge \ x < 0 \ \wedge \ f = 0 \ \wedge \ y < 0)\end{aligned}$$

where $g := 27x^2 - 32$ and $f := y^3 - 2y - x$. Note that the polynomial $3y^2 - 2$ is greater than zero at all points in C, so it can be omitted from the formula.

7.1.9 The overall algorithm

The function `QEPCAD` is responsible for running the QEPCAD algorithm. The modified version of the function is presented below.

```
1 void QepcadCls::QEPCAD(Word Fs, Word *t_, Word *F_e_, Word *F_n_,
  ↪  Word *F_s_)
2 {
3     Word A,D,F,F_e,F_n,F_s,Fh,J,P,Q,Ths,f,i,r,t, T;
4     /* hide Ths,i,t; */
5     Word cL,**cC,cr,ce,ci,*cT,cj,cs,cl,ct; /* Chris variables. */
6     Word Cs,Ps,Qs,Pps,Cps,Qps,SF; /* Chris variables. */
7     char c1,c2; /* Chris variables. */
```

```
8
9   Step1: /* Normalize. */
10      FIRST4(Fs,&r,&f,&Q,&Fh);
11      /*Int*/ PCNSTEP = 1;
12      /*Int*/ if (INTERACT()) USERINT(LFS("Before
        ↪  Normalization"),'a');
13      /*Int*/ if (PCCONTINUE == TRUE) { goto Return; }
14      /*Int*/ Ths = ACLOCK();
15      F = NORMQFF(Fh);
16      if (GVUA != NIL) GVNA = NORMQFF(GVUA);
17      /*Int*/ Ths = ACLOCK() - Ths;
18      /*Int*/ TMNORMQFF = Ths;
19      /*Int*/ GVNQFF = F;
20      //       if (TYPEQFF(F) != UNDET) { t = EQU; F_e = F; goto
        ↪  Return; }
21      /*Int*/ GVREFL = NIL;
22      /*Int*/ GVTD = NIL;
23
24  Step2: /* Projection. */
25      if (GVUA != NIL) F = LIST3(ANDOP,GVNA,F);
26      A = EXTRACT(r,F);
27      if (GVUA != NIL) {
28          GVNA = SECOND(F);
29          F = THIRD(F);
30      }
31
32      /*Int*/ for (i = 1; i <= r; i++) NMNIP[i] =
        ↪  LENGTH(LELTI(A,i));
33      /*Int*/ GVPF = LLCOPY(A);
34      /*Int*/ GVNIP = A;
35      /*Int*/ GVLV = r;
36      /*Int*/ PCNSTEP = 1;
37
38      /*Int if (INTERACT()) USERINT(LFS("After
        ↪  Normalization"),'A'); */
39      /*Int PCNSTEP = 1; */
40
41      // project and add jacobi determinants for quasi-affine cells
42      if (PCMCT == 'y') {
43          // note that quasi-affine does projection, too.
44          /*Int*/     USERINT(LFS("Before Projection
            ↪  (quasi-affine)"),'b');
45          QUASIAFFINE(r, GVVL, F, &A, &P, &J);
46          GVNIP = A;
47          GVPF = P;
```

```
48          GVPJ = J;
49      } else { // standard case
50          PROJECT(r,A,&P,&J);
51      }
52
53      /*Int*/ if (PCCONTINUE == TRUE) { goto Return; }
54
55  Step3: /* Truth-invariant CAD. */
56      /*Int*/ NMFPF = 0;
57      /*Int*/ for (i=1; i<=f; i++) NMFPF=NMFPF+LENGTH(LELTI(P,i));
58      /*Int*/ PCNSTEP = 1;
59      D = TICAD(Q,F,f,P,A);
60
61  Step5: /* Monotone cells, if needed */
62      if (PCMCT == 'y') {
63          /*Int*/ GVPC = D;
64          /*Int*/ PCNSTEP = 1;
65          /*Int*/ if (INTERACT()) USERINT(LFS("Before Refinement
            ↪  For Monotone Cells"),'m');
66          /*Int*/ if (PCCONTINUE == TRUE) { goto Return; }
67          /*Int*/ Ths = ACLOCK();
68
69          // add extra polynomials for [semi]-monotone cells and
            ↪  recompute the cad if needed
70          Word RPs = MONOTONE(&P, &J, D, r);
71          GVREFL = RPs;
72
73          // refine the CAD
74          D = REFINE(1, D, GVREFL, P);
75
76          // if we cached an ESPCAD, it will no longer be valid. If
            ↪  CAD was projection definable before, it will be still
77          // be projection definable now.
78          if (GVTD != NIL && FIRST(GVTD) == 0) {
79              SWRITE("*** Invalidating cached ESPCAD. ***\n");
80          }
81
82          // frontier condition
83          FRONTIER(f, D, P, &P, &J, &RPs);
84          GVREFL = RPs;
85          D = REFINE(1, D, GVREFL, P);
86
87          /*Int*/ if (PCCONTINUE == TRUE) { goto Return; }
88      }
89
```

```
90  Step6: /* Solution. */
91      /*Int*/ GVPC = D;
92      /*Int*/ PCNSTEP = 1;
93      /*Int*/ if (INTERACT()) USERINT(LFS("Before Solution"),'e');
94      /*Int*/ if (PCCONTINUE == TRUE) { goto Return; }
95      T = ACLOCK();
96      if (!PCMZERROR)
97
    ↪   SFC3(GVPC,GVPF,GVPJ,GVNFV,CCONC(LIST10(0,0,0,1,0,3,2,4,1,5),LIST1(-1)));
98      else
99          SFCFULLD(GVPC,GVPF,GVPJ,GVNFV);
100     T = ACLOCK() - T;
101     TMSFCONST = COMP(T,TMSFCONST);
102
103 Return: /* Prepare for return. */
104     *t_ = t;
105     *F_e_ = F_e;
106     *F_n_ = F_n;
107     *F_s_ = F_s;
108     return;
109 }
```

- The variable `PCMCT` indicates that a monotone CAD should be computed if it is equal to 1, and runs the standard algorithm otherwise.
- The first modification occurs on Lines 41-51, where standard projection is replaced by quasi-affine projection. The function `QUASIAFFINE` is called and the input polynomials, projection polynomials and projection factors are updated as necessary.
- Lines 55-61 remain unchanged, and construct a truth-invariant CAD. If monotone cells are to be produced, the CAD will have constant sign on the projection factors, as the option `full-cad` will be enabled.
- Lines 61-88 define a new step, where refinements for monotone cells and frontier condition are performed. Lines 70-71 compute the refinement points for monotone cells, by calling `MONOTONE` and Line 74 calls `REFINE` to compute the refinement. Line 84 calls `FRONTIER` to complete the refinement points above bad cells and Line 85 calls computes the refinement.

7.1.10 Additional modifications and Utility Functions

A brief summary of new and modified functions is now presented.

7.1.10.1 Modifications

- `extensions/sfext/addpol/CFLCELLLIST.c`: the function which identifies conflicting pairs of cells for solution formula construction. This function is now used to ensure that every cell can be defined by a Tarski formula. As

such, it takes a `flag` which indicates that a conflicting pair of cells should not depend on their truth value.

- `extensions/sfext/addpol/CLEAN_BIGLOOP.c`: modified to pass the `flag` to `CFLCELLLIST()`.
- `extensions/sfext/addpol/STRIPPED_BIGLOOP.c`: modified to pass the `flag` to `CFLCELLLIST()`.
- `extensions/sfext/sfcons/SFC3.c`: passes the `flag` to `CFLCELLLIST()`.
- `source/db/CAPolicy.h`: defines the function `GROEBNER`, which uses `Singular` to compute a Groebner basis.
- `source/db/CAServer.h`: defines the function `GROEBNER`, which uses `Singular` to compute a Groebner basis.
- `source/db/SINGULAR.c`: implements the function `GROEBNER`, which uses `Singular` to compute a Groebner basis.
- `source/db/SINGULAR.h`: defines the function `GROEBNER`, which uses `Singular` to compute a Groebner basis.
- `source/db/SingSacPolicy.h`: defines the function `GROEBNER`, which uses `Singular` to compute a Groebner basis.
- `source/db/SingularPolicy.h`: defines the function `GROEBNER`, which uses `Singular` to compute a Groebner basis.
- `source/main/data.c`: defines and initialises `PCMCT`, the flag to indicate that a monotone CAD should be constructed.
- `source/main/qepcadcls.h`:
  - Defines `GVREFL` and `GVTD`, list of refinement points and the CAD in which every cell can be defined by sings of projection factors, respectively,
  - Defines the new `QepcadCls::*` functions.
- `source/qepcad.h`: defines new non-`QepcadCls::*` functions.
- `source/proj/EXTRACTS.c`: responsible for extracting projecion polynomials. `ADD2A` has been refactored and renamed `ADDPOL`.
- `source/userint/USERINT.c`: adds new user interface commands.
- `source/qepcad.help`: documents new commands, such as `d-cell-*-t` and `pcmct <y/n>`.

7.1.10.2 New files

- `source/io/CELLWRT.c`: writes a cell as a Tarski formula, implementing the command `d-cell-t`.

- `extensions/sfext/crcads/ALLCELLWRITET.c`: writes all cells as Tarski formulas, implementing the command `d-all-cells-t`.
- `extensions/sfext/crcads/FALSECELLWRITET.c`: writes false all cells as Tarski formulas, implementing the command `d-false-cells-t`.
- `extensions/sfext/crcads/TRUECELLWRITET.c`: writes true all cells as Tarski formulas, implementing the command `d-true-cells-t`.
- `extensions/sfext/sfcons/SFC3f.c`: writes true all cells as Tarski formulas, implementing the command `d-true-cells-t`.
- `extensions/sfext/sfcons/SFC4.c`: writes true all cells as Tarski formulas, implementing the command `d-true-cells-t`.
- `source/io/CELLIPLLDWR.c`: helper function for writing cells as Tarski formulas, writes $f * 0$, where f is a polynomial and $* \in \{<, =, >\}$.
- `source/userint/PRDCT.c`: process `d-cell-t`.
- `source/userint/PRMCT.c`: process `mct <y/n>`.
- `source/util/ADDPOL.c`: refactoring of `ADD2A`.
- `source/util/ADDPOLS.c`: bulk call to `ADDPOL`.
- `source/proj/ProjMcxUtil.c`: Stripped down version of McCallum projection, used as a utility function, e.g., to compute the x_1-coordinates of refinement points in `MONOTONE()`.
- `source/proj/QUASIAFFINE.c`: computes a quasi-affine CAD, as discussed above.
- `source/ticad/MONOTONE.c`: finds the set of refinement points required for each 2-dimensional cell in the CAD to be monotone.
- `source/ticad/FRONTIER.c`: implements the function `FRONTIER()`, responsible for refining the CAD such that it satisfies teh frontier condition.
- `source/ticad/LazardLifting.c`: helper function for `FRONTIER()`, responsible for implementing the extension of Lazard's algorithm for lifting with bad points.
- `source/ticad/REFINE.c`:: responsible for computing the refinements, defined in `GVREFL`.
- `source/util/ADDREFINEMENTPOINTS.c`: adds refinement points (see above).
- `source/util/GETSAMPLEK.c`: retrieves k-th coordinate of a cell's sample point.
- `source/util/IPFRPmod.c`: the original "integral polynomial from rational polynomial" assumes that the input is in rational representation but

always has integer coefficients. This version converts an arbitrary rational polynomial into an integral polynomial.

- `source/util/JACOBI.c`: given a list of polynomials and variable indices, construct a Jacobi matrix. source/util/LDCOPY.c‘: list “deep” (recursive) copy.
- `source/util/LEVELIDX.c`: helper function. Given a positional index for a cell C with index $(i_1, \ldots, i_n)$, return $(j_1, \ldots, j_k)$, where $j \leq n$ such that $i_{j_\ell} = 1, 1 \leq \ell \leq k$.
- `source/util/PADDVS.c`: append variables. Given $f \in \mathbb{Z}[x_1, \ldots, x_k]$, output f as a polynomial in $\mathbb{Z}[x_1, \ldots, x_k, y_1, \ldots, y_\ell]$.
- `source/util/PPREPVS.c`: prepend variables. Given $f \in \mathbb{Z}[x_1, \ldots, x_k]$, output f as a polynomial in $\mathbb{Z}[y_1, \ldots, y_\ell, x_1, \ldots, x_k]$.
- `source/util/ROOTS.c`: convenience function for univariate root fonding. Uses `saclib`'s `IPFSFB`, which computes a squaref-free basis.
- `source/util/SUBSTITUTE.c`: takes a polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$ and a sample point $\mathbf{b} \in \mathbb{A}^k$, compute $f(\mathbf{b}) \in \mathbb{Z}[x_{k+1}, \ldots, x_n]$.
- `source/util/TwoDimensionalIndex.c`: Given a cell's positional index, which corresponds to the $(i_1, \ldots, i_k) \in \{0, 1\}^n$, return j, k such that $i_j = i_k = 1$. If $(i_1, \ldots, i_n)$ does not have exactly two elements equal to 1, the function returns an error.

Note that some other files show changes, when using git diff, but these are just formatting and whitespace changes which made the code more readable.

7.2 Test Cases

Example 7.1. Let $n = 3$ and consider the unit sphere

$$S := \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 - 1 = 0\}.$$

- **Quasi-affine**:

  critical points of

  - $\text{proj}_{\text{span}\{x,y\}} : z = 0$
  - $\text{proj}_{\text{span}\{x,z\}} : y = 0$
  - $\text{proj}_{\text{span}\{y,z\}} : x = 0$

- **Monotone**:

  each of the 8 2-dimensional section cells are already monotone

- **Frontier Condition**:

  No blow-up points, already satisfied.

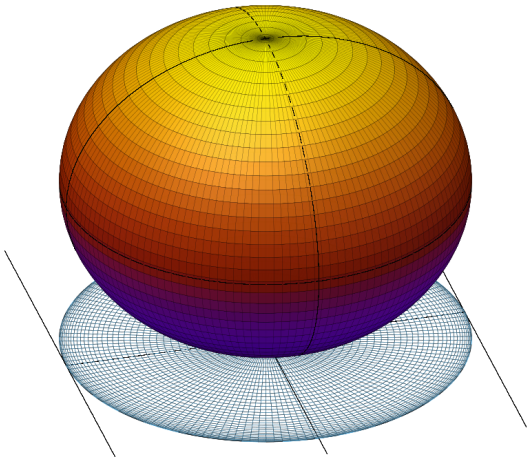

Figure 7.1: Plot of the sphere from Example refexm:sphere, showing critical points of projections onto one and two-dimensional coordinate subspaces, and their projections.

Below is the set of projection factors and true cells produced by QEPCAD. In the projection factor set, factors of input polynomials are labelled by `A`, factorised polynomials output by CAD projection are labelled by `P` and polynomials which ensure the CAD contains quasi-affine cells are labelled by `Q`.

```
Before Solution >
d-proj-fac
P_1,1  = fac(J_1,1) = fac(dis(P_2,1))
       = x + 1

P_1,2  = fac(J_1,1) = fac(dis(P_2,1))
       = x - 1

Q_1,3  = fac(Q_3,2) = fac(input)
       = x


P_2,1  = fac(J_2,1) = fac(dis(A_3,1))
       = y^2 + x^2 - 1

Q_2,2  = fac(Q_3,1) = fac(input)
       = y
```

```
A_3,1  = input
       = z^2 + y^2 + x^2 - 1

Q_3,2  = fac(Q_3,1) = fac(input)
       = z



Before Solution >
d-true-cells-t
---------- Information about the cell (6,2,2) (Dimension (0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 = 0
  x > 0
Level 2
  y^2 + x^2 - 1 = 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

*** Initialising the RCAD. ***

Signs of projection factors (for defining formula) -

Index in RCAD: (6,2,2)
Level 1
  x + 1 > 0
  x - 1 = 0
  x > 0
Level 2
  y^2 + x^2 - 1 = 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Sample point ---------------------------------------

The sample point is in a PRIMITIVE representation.
```

```
alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1
             = 1.0000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000


-----------------------------------------------------
---------- Information about the cell (5,6,2) (Dimension (1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 = 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,6,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 = 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Sample point ---------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of 4 x^2 - 3 between 1/2 and 1
```

```
      = 0.8660254038-

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = alpha
             = 0.8660254038-
Coordinate 3 = the unique root of x between 0 and 0
             = the unique root of x between 0 and 0
             = 0.0000000000


-----------------------------------------------------
---------- Information about the cell (5,5,6) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,5,6)
Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 11 between 1/2 and 1
```

```
      = 0.8291561976-

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = alpha
             = 0.8291561976-


----------------------------------------------------
---------- Information about the cell (5,5,2) (Dimension (1,1,0) (2))

Signs of projection factors -------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,5,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 11 between -1 and -1/2
      = -0.8291561976-
```

```
Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = alpha
             = -0.8291561976-


-----------------------------------------------------
---------- Information about the cell (5,4,6) (Dimension (1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,4,6)
Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Sample point ---------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 4 x^2 - 3 between 1/2 and 1
      = 0.8660254038-
```

```
Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = alpha
             = 0.8660254038-


--------------------------------------------------
---------- Information about the cell (5,4,2) (Dimension (1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,4,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 4 x^2 - 3 between -1 and -1/2
      = -0.8660254038-

Coordinate 1 = 1/2
```

```
             = 0.5000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = alpha
             = -0.8660254038-


------------------------------------------------------
---------- Information about the cell (5,3,6) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,3,6)
Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Sample point ---------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 11 between 1/2 and 1
      = 0.8291561976-

Coordinate 1 = 1/2
             = 0.5000000000
```

```
Coordinate 2 = -1/4
             = -0.2500000000
Coordinate 3 = alpha
             = 0.8291561976-


-----------------------------------------------------
---------- Information about the cell (5,3,2) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,3,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 11 between -1 and -1/2
      = -0.8291561976-

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = -1/4
```

```
             = -0.2500000000
Coordinate 3 = alpha
             = -0.8291561976-


-----------------------------------------------------
---------- Information about the cell (5,2,2) (Dimension (1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 = 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,2,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x > 0
Level 2
  y^2 + x^2 - 1 = 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of 4 x^2 - 3 between -1 and -1/2
      = -0.8660254038-

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = alpha
             = -0.8660254038-
```

```
Coordinate 3 = the unique root of x between 0 and 0
             = the unique root of x between 0 and 0
             = 0.0000000000


--------------------------------------------------
---------- Information about the cell (4,6,2) (Dimension (0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 = 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,6,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 = 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = 1
             = 1.0000000000
```

```
Coordinate 3 = 0
             = 0.0000000000


-----------------------------------------------------
---------- Information about the cell (4,5,6) (Dimension (0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,5,6)
Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 15 between 1/2 and 1
      = 0.9682458366-

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = alpha
```

```
             = 0.9682458366-


-----------------------------------------------------
---------- Information about the cell (4,5,2) (Dimension (0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,5,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 15 between -1 and -1/2
      = -0.9682458366-

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = alpha
             = -0.9682458366-
```

```
-----------------------------------------------------
---------- Information about the cell (4,4,6) (Dimension (0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,4,6)
Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 1
             = 1.0000000000
```

```
--------------------------------------------------
---------- Information about the cell (4,4,2) (Dimension (0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,4,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = -1
             = -1.0000000000
```

```
------------------------------------------------------
---------- Information about the cell (4,3,6) (Dimension (0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,3,6)
Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 4 x^2 - 3 between 1/2 and 1
      = 0.8660254038-

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = -1/2
             = -0.5000000000
Coordinate 3 = alpha
             = 0.8660254038-


------------------------------------------------------
```

```
---------- Information about the cell (4,3,2) (Dimension (0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,3,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Sample point ---------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 4 x^2 - 3 between -1 and -1/2
      = -0.8660254038-

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = -1/2
             = -0.5000000000
Coordinate 3 = alpha
             = -0.8660254038-


--------------------------------------------------
---------- Information about the cell (4,2,2) (Dimension (0,0,0) (0))
```

```
Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 = 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,2,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x = 0
Level 2
  y^2 + x^2 - 1 = 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = -1
             = -1.0000000000
Coordinate 3 = 0
             = 0.0000000000


------------------------------------------------------
---------- Information about the cell (3,6,2) (Dimension (1,0,0) (1))
```

```
Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 = 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,6,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 = 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of 4 x^2 - 3 between 1/2 and 1
      = 0.8660254038-

Coordinate 1 = -1/2
             = -0.5000000000
Coordinate 2 = alpha
             = 0.8660254038-
Coordinate 3 = the unique root of x between 0 and 0
             = the unique root of x between 0 and 0
             = 0.0000000000


--------------------------------------------------
---------- Information about the cell (3,5,6) (Dimension (1,1,0) (2))
```

```
Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,5,6)
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 11 between 1/2 and 1
      = 0.8291561976-

Coordinate 1 = -1/2
             = -0.5000000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = alpha
             = 0.8291561976-


------------------------------------------------------
---------- Information about the cell (3,5,2) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------
```

```
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,5,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y > 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 11 between -1 and -1/2
      = -0.8291561976-

Coordinate 1 = -1/2
             = -0.5000000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = alpha
             = -0.8291561976-


--------------------------------------------------
---------- Information about the cell (3,4,6) (Dimension (1,0,0) (1))

Signs of projection factors ------------------------
```

```
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,4,6)
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 4 x^2 - 3 between 1/2 and 1
      = 0.8660254038-

Coordinate 1 = -1/2
             = -0.5000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = alpha
             = 0.8660254038-


------------------------------------------------------
---------- Information about the cell (3,4,2) (Dimension (1,0,0) (1))

Signs of projection factors ------------------------

Level 1
```

```
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,4,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 4 x^2 - 3 between -1 and -1/2
      = -0.8660254038-

Coordinate 1 = -1/2
             = -0.5000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = alpha
             = -0.8660254038-


--------------------------------------------------
---------- Information about the cell (3,3,6) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
```

```
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,3,6)
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 11 between 1/2 and 1
      = 0.8291561976-

Coordinate 1 = -1/2
             = -0.5000000000
Coordinate 2 = -1/4
             = -0.2500000000
Coordinate 3 = alpha
             = 0.8291561976-


------------------------------------------------------
---------- Information about the cell (3,3,2) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
```

```
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,3,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 < 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of 16 x^2 - 11 between -1 and -1/2
      = -0.8291561976-

Coordinate 1 = -1/2
             = -0.5000000000
Coordinate 2 = -1/4
             = -0.2500000000
Coordinate 3 = alpha
             = -0.8291561976-


--------------------------------------------------
---------- Information about the cell (3,2,2) (Dimension (1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
```

```
Level 2
  y^2 + x^2 - 1 = 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,2,2)
Level 1
  x + 1 > 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 = 0
  y < 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of 4 x^2 - 3 between -1 and -1/2
      = -0.8660254038-

Coordinate 1 = -1/2
             = -0.5000000000
Coordinate 2 = alpha
             = -0.8660254038-
Coordinate 3 = the unique root of x between 0 and 0
             = the unique root of x between 0 and 0
             = 0.0000000000


--------------------------------------------------
---------- Information about the cell (2,2,2) (Dimension (0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  x + 1 = 0
  x - 1 < 0
  x < 0
```

```
Level 2
  y^2 + x^2 - 1 = 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,2)
Level 1
  x + 1 = 0
  x - 1 < 0
  x < 0
Level 2
  y^2 + x^2 - 1 = 0
  y = 0
Level 3
  z^2 + y^2 + x^2 - 1 = 0
  z = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = -1
             = -1.0000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000


--------------------------------------------------
```

The sphere is decomposed into 26 cylindrical cells: 8 2-dimensional cells, 12 1-dimensional cells and 6 0-dimensional cells.

In Example 7.2, we present a 2-dimensional section cell which is the graph of a quasi-affine map, but is not a monotone cell.

Example 7.2. Let $n = 3$ and consider the 2-dimensional section cell

$$C := \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, y > 0, x + y^2 < 1, z = x^2 + y^2\}.$$

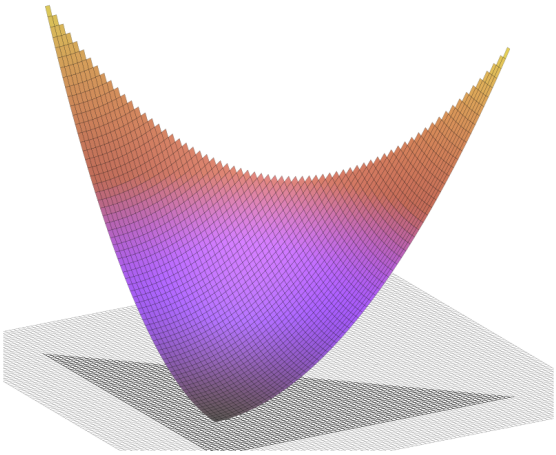

Figure 7.2: Plot of the 2-dimensional cell from Examplerefexm:qanm.

- **Quasi-affine**:

  C is already the graph of a quasi-affine map.

- **Monotone**:

  - The $(1,1)$-cell $C' := \mathrm{proj}_{\mathbb{R}^2}(C)$ is already monotone, since its bottom $C'_B = \{-1 < x < 1, y = 0\}$ and top $C'_T = \{-1 < x < 1, y = 1 - x\}$ are independent of and strictly decreasing in x respectively.
  - $C_B = \{-1 < x < 1, y = 0, z = x^2\}$ is already monotone,
  - $C_T = \{-1 < x < 1, y = 1 - x, z = x^2 + y^2\}$ is not monotone, since it intersects the plane $\{z = c\}, 1/2 < c < 1$ in two points. The algorithm will find the critical points of $f := z - y^2 - x^2 = 0$ subject to $g := y + x - 1 = 0$ in the interval $0 < x < 1$. These are the points at which

    $$\det \begin{pmatrix} \frac{\partial g}{x} & \frac{\partial g}{y} \\ \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{pmatrix} = \det \begin{pmatrix} 1 & 1 \\ 2x & 2y \end{pmatrix} = 2x - 2y = x - y.$$

    Since $x - y = 0$ and $x + y - 1$ intersect at $x = 1/2$, the refinement polynomial $2x - y = 0$ is produced.

- **Frontier Condition**:

  No blow-up points, already satisfied.

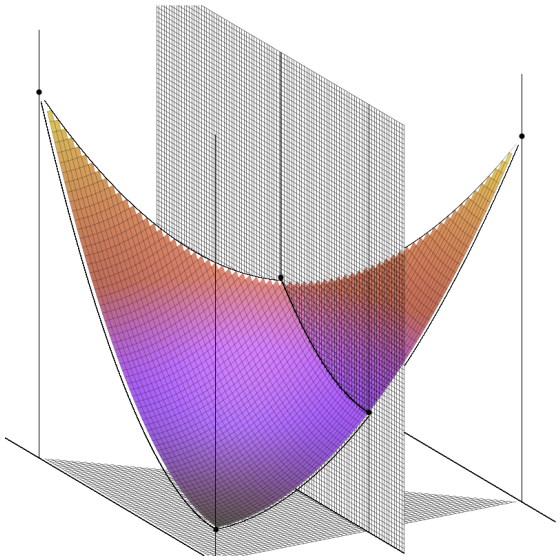

Figure 7.3: Plot of the CAD with monotone cells, compatible with the 2-dimensional cell presented in Example
refexm:qanm.

QEPCAD output is presented below. Note the projection factor M_1_3 defines the refinement point $x = 1/2$.

```
Before Refinement For Monotone Cells >
d-true-cells-t
---------- Information about the cell (3,3,2) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 < 0
Level 2
  y > 0
  y + x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

*** Initialising the RCAD. ***

Signs of projection factors (for defining formula) -

Index in RCAD: (3,3,2)
```

```
Level 1
  x > 0
  x - 1 < 0
Level 2
  y > 0
  y + x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 1/8
             = 0.1250000000
Coordinate 3 = 17/64
             = 0.2656250000


--------------------------------------------------

Before Refinement For Monotone Cells >
g
Before Solution >
d-proj-fac
A_1,1  = input
       = x

A_1,2  = fac(J_1,1) = fac(res(A_2,1|A_2,2))
       = x - 1

M_1,3  = fac(K_1,3) = fac(input)
       = 2 x - 1


A_2,1  = input
       = y

A_2,2  = input
       = y + x - 1
```

```
A_3,1  = input
       = z - y^2 - x^2



Before Solution >
d-ref
M_1,1  = input  *** Refinement of cell (3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000

Coordinate 1 = 1/2
             = 0.5000




Before Solution >
d-true-cells-t
---------- Information about the cell (5,3,2) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 < 0
  2 x - 1 > 0
Level 2
  y > 0
  y + x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,3,2)
Level 1
  x > 0
  x - 1 < 0
  2 x - 1 > 0
Level 2
  y > 0
```

```
  y + x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 3/4
             = 0.7500000000
Coordinate 2 = 1/16
             = 0.0625000000
Coordinate 3 = 145/256
             = 0.5664062500


------------------------------------------------------
---------- Information about the cell (4,3,2) (Dimension (0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 < 0
  2 x - 1 = 0
Level 2
  y > 0
  y + x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,3,2)
Level 1
  x > 0
  x - 1 < 0
  2 x - 1 = 0
Level 2
  y > 0
  y + x - 1 < 0
Level 3
  z - y^2 - x^2 = 0
```

```
Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 1/8
             = 0.1250000000
Coordinate 3 = 17/64
             = 0.2656250000


--------------------------------------------------
---------- Information about the cell (3,3,2) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 < 0
  2 x - 1 < 0
Level 2
  y > 0
  y + x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,3,2)
Level 1
  x > 0
  x - 1 < 0
  2 x - 1 < 0
Level 2
  y > 0
  y + x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

Sample point ----------------------------------------
```

```
The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/4
             = 0.2500000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = 1/8
             = 0.1250000000


-----------------------------------------------------

Before Solution >
```

Example 6.1 (Davenport et al. (2020), Example 2.1) illustrated a CAD which fails to satisfy the frontier condition due to a blow-up point. The cell shown in the example is part of the Whitney umbrella, and, unlike the examples considered so far, its frontier is not homeomorphic to a circle.

Example 7.3. (Davenport et al., 2020, Example 2.1)

Let $n = 3$ and consider the $(1, 1, 0)$-cell

$$C = \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, -x < y < x, y^2 - x^2 z = 0\}.$$

Quasi-affine**:

critical points of - $\text{proj}_{\text{span}\{x,y\}} : z = 0$ - $\text{proj}_{\text{span}\{x,z\}} : y = 0$ - $\text{proj}_{\text{span}\{y,z\}} : x = 0$ - **Monotone**:

C is partitioned, in the quasi-affine step, into three cells

$$C_1 = \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, -x < y < 0, y^2 - x^2 z = 0\}, \tag{7.1}$$
$$C_2 = \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, y = 0, y^2 - x^2 z = 0\}, \tag{7.2}$$
$$C_3 = \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, 0 < y < x, y^2 - x^2 z = 0\}, \tag{7.3}$$
$$\tag{7.4}$$

each of which is already monotone.

- **Frontier Condition**:

  there is one bad point, $\mathbf{b} = (0, 0)$ in the decomposition induced on $\mathbb{R}^2$, above which the polynomial $y^2 - x^2 z$ vanishes identically. The CAD

contains three cells projecting on **b**:

$$B_1 = \{(0,0,z) \in \mathbb{R}^3 \mid z < 0\}, \tag{7.5}$$
$$B_2 = \{(0,0,z) \in \mathbb{R}^3 \mid z = 0\}, \tag{7.6}$$
$$B_3 = \{(0,0,z) \in \mathbb{R}^3 \mid z > 0\}. \tag{7.7}$$
$$\tag{7.8}$$

This CAD does not satisfy the frontier condition. Indeed, we have

$$B := B_3 \cap \mathrm{fr}\,(()C_1) = B_3 \cap \mathrm{fr}\,(()C_2)$$

which is not a cell of the CAD. According to Proposition 6.4 (Lazard (2010), Proposition 5.13), the following situation should occur:

> there exists a cell E of $\mathcal{D}$ such that $\dim(E) < \dim(C_1) = \dim((C_2)$ and $\emptyset \neq B_3 \cap \mathrm{fr}\,(E) \neq B_3$.

For C_1, the cell E in question is the $(1,0,0)$-cell $\{0 < x < 1, x + y = 0, z = 1\}$, and for C_3, E is the $(1,0,0)$-cell $\{0 < x < 1, x - y = 0, z = 1\}$ In both cases, $\mathrm{fr}\,(()E) \cap B_3 = (0,0,1)$ – the required refinement point.

Lazard (2010), Algorithm 5.15 is then applied, resulting in the computation of the following saturations:

$$I_1 := \{y^2 - x^2 z, x, 1 - yt\} \cap \mathbb{Z}[x,y,z], \tag{7.9}$$
$$I_2 := \{y^2 - x^2 z, y, 1 - xt\} \cap \mathbb{Z}[x,y,z], \tag{7.10}$$
$$I_3 := \{y^2 - x^2 z, x, 1 - (x+y)t\} \cap \mathbb{Z}[x,y,z], \tag{7.11}$$
$$I_4 := \{y^2 - x^2 z, x + y, 1 - xt\} \cap \mathbb{Z}[x,y,z], \tag{7.12}$$
$$I_5 := \{y^2 - x^2 z, x, 1 - (x-y)t\} \cap \mathbb{Z}[x,y,z], \tag{7.13}$$
$$I_6 := \{y^2 - x^2 z, x - y, 1 - xt\} \cap \mathbb{Z}[x,y,z], \tag{7.14}$$
$$\tag{7.15}$$

where t is a new variable.

The Groebner basis for I_2 contains the polynomial z, which defines the refinement point $(0,0,0)$, which is already a cell in the CAD. The Groebner bases for I_4 and I_6 both contain $z-1$, defining the refinement point $(0,0,1)$, which was identified by Proposition 6.4.

Part of the QEPCAD output is now presented. In particular the projection factors, refinement points and the collection of cells which comprises the frontier of C.

```
Before Solution >
d-proj-fac
A_1,1  = fac2(J_2,1) = fac2(ldcf(A_3,1))
       = fac(J_1,1) = fac(res(A_2,1|A_2,2))
```

```
       = fac(J_1,2) = fac(res(A_2,1|P_2,3))
       = fac(J_1,3) = fac(res(A_2,2|P_2,3))
       = x

A_1,2  = input
       = x - 1


A_2,1  = input
       = y + x

A_2,2  = input
       = y - x

P_2,3  = fac2(J_2,2) = fac2(ldcf(red(A_3,1)))
       = y


A_3,1  = input
       = x^2 z - y^2

Q_3,2  = fac(Q_3,1) = fac(input)
       = z

M_3,3  = fac(K_3,3) = fac(input)
       = z - 1



Before Solution >
d-ref


M_3,1  = input  *** Refinement of cell (2,2,3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000

Coordinate 1 = 0
             = 0.0000
Coordinate 2 = 0
             = 0.0000
Coordinate 3 = 1
             = 1.0000
```

```
Before Solution >
d-cell-t(3,2,2)---------- Information about the cell (3,2,2) (Dimension (1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 < 0
Level 2
  y + x = 0
  y - x < 0
  y < 0
Level 3
  x^2 z - y^2 < 0
  z = 0

*** Initialising the RCAD. ***

Signs of projection factors (for defining formula) -

Index in RCAD: (3,2,2)
Level 1
  x > 0
  x - 1 < 0
Level 2
  y + x = 0
  y - x < 0
  y < 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Sample point ---------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = -1/2
             = -0.5000000000
Coordinate 3 = 0
```

```
             = 0.0000000000


-----------------------------------------------------

Before Solution >
d-cell-t(3,6,2)---------- Information about the cell (3,6,2) (Dimension (1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 < 0
Level 2
  y + x > 0
  y - x = 0
  y > 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,6,2)
Level 1
  x > 0
  x - 1 < 0
Level 2
  y + x > 0
  y - x = 0
  y > 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 1/2
             = 0.5000000000
```

```
Coordinate 3 = 0
             = 0.0000000000


-----------------------------------------------------

Before Solution >
d-cell-t(4,2,2)---------- Information about the cell (4,2,2) (Dimension (0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x = 0
  y - x < 0
  y < 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,2,2)
Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x = 0
  y - x < 0
  y < 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1
             = 1.0000000000
Coordinate 2 = -1
```

```
             = -1.0000000000
Coordinate 3 = 0
             = 0.0000000000


-----------------------------------------------------

Before Solution >
d-cell-t(4,3,2)---------- Information about the cell (4,3,2) (Dimension (0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x > 0
  y - x < 0
  y < 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,3,2)
Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x > 0
  y - x < 0
  y < 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Sample point ---------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1
             = 1.0000000000
```

```
Coordinate 2 = -1/2
             = -0.5000000000
Coordinate 3 = 0
             = 0.0000000000


-----------------------------------------------------

Before Solution >
d-cell-t(4,4,2)---------- Information about the cell (4,4,2) (Dimension (0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x > 0
  y - x < 0
  y = 0
Level 3
  x^2 z - y^2 = 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,4,2)
Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x > 0
  y - x < 0
  y = 0
Level 3
  x^2 z - y^2 = 0
  z = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1
```

```
             = 1.0000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000


--------------------------------------------------

Before Solution >
d-cell-t(4,5,2)---------- Information about the cell (4,5,2) (Dimension (0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x > 0
  y - x < 0
  y > 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,5,2)
Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x > 0
  y - x < 0
  y > 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000
```

```
Coordinate 1 = 1
             = 1.0000000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = 0
             = 0.0000000000


--------------------------------------------------

Before Solution >
d-cell-t(4,6,2)---------- Information about the cell (4,6,2) (Dimension (0,0,0) (0))

Signs of projection factors -------------------------

Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x > 0
  y - x = 0
  y > 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,6,2)
Level 1
  x > 0
  x - 1 = 0
Level 2
  y + x > 0
  y - x = 0
  y > 0
Level 3
  x^2 z - y^2 < 0
  z = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000
```

```
Coordinate 1 = 1
             = 1.0000000000
Coordinate 2 = 1
             = 1.0000000000
Coordinate 3 = 0
             = 0.0000000000


-----------------------------------------------------

Before Solution >
d-cell-t(2,2,2)---------- Information about the cell (2,2,2) (Dimension (0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  x = 0
  x - 1 < 0
Level 2
  y + x = 0
  y - x = 0
  y = 0
Level 3
  x^2 z - y^2 = 0
  z = 0
  z - 1 < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,2)
Level 1
  x = 0
  x - 1 < 0
Level 2
  y + x = 0
  y - x = 0
  y = 0
Level 3
  x^2 z - y^2 = 0
  z = 0
  z - 1 < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.
```

```
alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000


--------------------------------------------------

Before Solution >
d-cell-t(2,2,3)---------- Information about the cell (2,2,3) (Dimension (0,0,1) (1))

Signs of projection factors ------------------------

Level 1
  x = 0
  x - 1 < 0
Level 2
  y + x = 0
  y - x = 0
  y = 0
Level 3
  x^2 z - y^2 = 0
  z > 0
  z - 1 < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,3)
Level 1
  x = 0
  x - 1 < 0
Level 2
  y + x = 0
  y - x = 0
  y = 0
Level 3
  x^2 z - y^2 = 0
  z > 0
  z - 1 < 0
```

```
Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 1/2
             = 0.5000000000


--------------------------------------------------

Before Solution >
d-cell-t(2,2,4)---------- Information about the cell (2,2,4) (Dimension (0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  x = 0
  x - 1 < 0
Level 2
  y + x = 0
  y - x = 0
  y = 0
Level 3
  x^2 z - y^2 = 0
  z > 0
  z - 1 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,4)
Level 1
  x = 0
  x - 1 < 0
Level 2
  y + x = 0
  y - x = 0
  y = 0
Level 3
  x^2 z - y^2 = 0
```

```
  z > 0
  z - 1 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 0
             = 0.0000000000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 1
             = 1.0000000000


--------------------------------------------------
```

Chapter 8

A novel algorithm for obtaining the frontier condition on a cylindrical decomposition of n-dimensional space

So far, we have been concerned only with cylindrical algebraic decompositions compatible with subsets of $\mathbb{R}^n$ having dimension at most two. We will now present a novel algorithm for obtaining the frontier condition on a cylindrical algebraic decomposition of $\mathbb{R}^n$ compatible with a set of arbitrary dimension.

We begin by presenting some examples of cylindrical cells and their frontiers. Example 8.1 provides motivation for this novel algorithm. Many cylindrical section cells are well-behaved, with their top and bottom being cylindrical section cells. For example, consider

$$C := \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, f(x, y) > 0, g(x, y) < 0, z = \varphi(x, y)\}$$

where $f(x, y) = y, g(x, y) = x + y - 1$ and $\varphi(x, y) = x^2 + y^2$. The projection of C onto $\mathbb{R}^2$ is a cylindrical sector cell

$$C' = \{(x, y) \in \mathbb{R}^2 \mid 0 < x < 1, f(x, y) > 0, g(x, y) < 0\}$$

with the bottom, C'_B, of C' being the graph of f and the top, C'_T, of C' being the graph of g. It is clear that $\varphi|_{C'_B}$ and $\varphi|_{C'_T}$ is a continuous function, hence C_B and C_T are cylindrical section cells. Sometimes the bottom and top may fail to exist. This can happen if a cell is not bounded from below or above, or if

the function of which a section cell is the graph tends to infinity. For example, the section cell

$$C = \{(x, y, z) \mid x > 0, y > 0, yz - 1 = 0\}$$

has no top or bottom. Firstly, note that the projection, C', of C onto $\mathbb{R}^2$ is not bounded from above, hence C_T does not exist. Secondly, at every point of $C'_B = \{(x, y) \in \mathbb{R}^2 \mid x > 0, y = 0\}$, $yz - 1 = 0$ is not defined as the z-coordinate of C tends to infinity as y approaches zero. A more interesting situation arises when there is a blow-up in the top or bottom of a section cell, as shown in Example 8.1, reproduced below.

Example 8.1. (Basu et al., 2015, Example 3.2) Let

$$C' = \{(x, y) \in \mathbb{R}^2 \mid -1 < x < 1, |x| < y < 1\}$$

be a cylindrical sector cell in $\mathbb{R}^2$ and C be the graph of $\varphi|_{C'}$, where

$$\varphi(x, y) = |x/y|.$$

C is a cylindrical section cell in $\mathbb{R}^3$, according to Definition 2.20, despite the fact that $\varphi|_{C'}$ is not analytic. Consider the bottom of C

$$C_B = \{(x, y, z) \in \mathbb{R}^3 \mid -1 < x < 1, x \neq 0, y = |x|, z = 1\} \cup \{(0, 0, z) \mid 0 \leq z \leq 1\}.$$

Due to the blow-up of φ at the origin, C_B is not a cylindrical cell. Note that C_B does not contain all points at which φ vanishes.

As discussed above, in relation to Example 8.1, the frontier of a cylindrical cell can be geometrically nontrivial, e.g., it may contain a blow-up.

8.1 Statement of the Problem

Semialgebraic sets will usually be represented by Quantifier-Free Boolean formulas (QFFs) (see Definition 2.4). Occasionally, First-Order formulas, a wider class of formulas in which some variables may be bound by quantifiers $\forall$ and $\exists$ (see Definition 2.5).

Theorem 8.1. *Let $S \subset \mathbb{R}^n$ be a semialgebraic set defined by a QFF F with s different polynomials of maximum degree d in $\mathbb{R}[x_1, \ldots, x_n]$. There is an algorithm, taking F as input, which outputs a cylindrical decomposition $\mathcal{D}$ of $\mathbb{R}^n$ compatible with S and satisfying the frontier condition. The complexity of this algorithm is $(sd)^{O(1)^{n2^n}}$. This is also an upper bound on the number of cells in $\mathcal{D}$, number of polynomials defining cells and their degrees.*

Remark. The algorithm from Theorem 8.1 is understood as a Blum-Shub-Smale (BSS) real numbers machine (Blum et al., 1998). A similar statement is also true for the Turing machine model, in which case the complexity bound depends, in addition, polynomially on maximal bit-size of coefficients (cf. Collins (1975)).

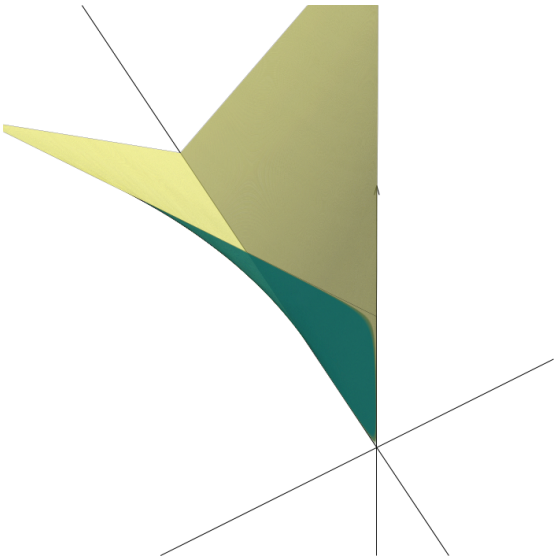

Figure 8.1: C, with $\mathrm{fr}\,(C)$ superimposed, from Example refexm:top-bottom-not-cylindrical, demonstrating that C_B is not the graph of a continuous function.

The following result, which is somewhat natural and is also mentioned by Basu et al. (2015) will be proved. This will allow us to apply Theorem 8.1 directly to semialgebraic sets defined by first-order Boolean formulas with quantifiers.

Theorem 8.2. *Let $\mathcal{D}$ be a CAD of $\mathbb{R}^n$ compatible with a semialgebraic set $S \subset \mathbb{R}^n$ and satisfying the frontier condition. Then the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^k$ for all $1 \leq k < n$ is a cylindrical decomposition of $\mathbb{R}^k$ compatible with $\mathrm{proj}_{\mathbb{R}^k}(S)$ and satisfying the frontier condition.*

This will allow us to apply Theorem 8.1 to semialgebraic sets represented by first-order Boolean formulas (with quantifiers) and obtain a decomposition compatible with S and satisfying the frontier condition.

8.2 Theory

8.2.1 Computing the frontier of a semialgebraic set

We presentd the classical CAD algorithm due to Collins (1975), as formulated in Basu et al. (2006), Algorithm 11.2, in 2.8. Given a set of s different polynomials with maximum degree d in $\mathbb{Z}[x_1, \ldots, x_n$, this algorithm has complexity

$$(sd)^{O(1)^n}.$$

This is also an upper bound on the number of polynomials, their maximum degree, and the number of cells in the CAD it produces. Several variants of the classical CAD algorithm exist. The most convenient version for our situation takes as input a set $\{F_1, \ldots, F_k\}$ of QFF and returns a CAD compatible with each set $S_i \subset \mathbb{R}^n$ represented by the formula F_i, $1 \leq i \leq k$. The same bounds on complexity, number of cells, number of polynomials and degrees as obtained in Proposition 2.8 is obtained, where s is the number of different polynomials appearing in formulas $F_1, \ldots, F_k$ and d is their maximum degree. An algorithm for constructing such a CAD is presented by Collins and Hong (1991), and an alternative approach is presented by Bradford et al. (2014).

The following algorithm allows us to compute the frontier of S: $\operatorname{fr}(S) = \operatorname{cl}(S) \setminus S$ It is a straightforward application of a singly exponential quantifier elimination algorithm.

Lemma 8.1. *Let $S \subset \mathbb{R}^n$ be a semialgebraic set defined by a QFF F containing s different polynomials in $\mathbb{R}[x_1, \ldots, x_n]$ having maximum degree d. There is an algorithm, taking F as input, which represents the semialgebraic set $\operatorname{fr}(S)$ by a QFF F' with complexity $(sd)^{O(n^2)}$. This is also an upper bound on the number of polynomials in F' and their degrees.*

Proof. Observe that $\operatorname{fr}(S)$ can be represented by a first-order Boolean formula

$$\operatorname{fr}(S) = \{\mathbf{x} \in (\mathbb{R}^n \setminus S) \mid \forall \varepsilon > 0 \, \exists \mathbf{y} \in S \, (\|\mathbf{x} - \mathbf{y}\| < \varepsilon)\}. \tag{8.1}$$

Using singly-exponential quantifier elimination algorithm (Basu et al., 2006, Algorithm 14.21), we represent $\operatorname{fr}(S)$ as a QFF F' with the bounds required in the lemma. □

8.2.2 Intermediate Results

Recall that, according to Definition 2.20, to each cylindrical cell in a CAD of $\mathbb{R}^n$ a (multi-)index $(i_1, \ldots, i_n) \in \{0, 1\}^n$ is assigned. Introduce a lexicographic order $<_{\text{lex}}$ on the set of all cell indices as follows. For any two indices, $M := (i_1, \ldots, i_n)$ and $N := (j_1, \ldots, j_n)$, we set $M <_{\text{lex}} N$ iff for the maximal k, $0 \leq k < n$, such that $i_1 = j_1, \ldots, i_k = j_k$ we have $i_{k+1} < j_{k+1}$. If $M <_{\text{lex}} N$ or $M = N$, we write $M \leq_{\text{lex}} N$.

Lemma 8.2. *Let $\mathcal{D}$ be a CAD of $\mathbb{R}^n$ and let C be a cell of $\mathcal{D}$ with index M. Then $\operatorname{fr}(C)$ is contained in a union of cells of $\mathcal{D}$ with indices lexicographically less than M.*

Proof. Proceed by induction on n.

When $n = 1$, the cell C either has index $M = (0)$ or $M = (1)$. If $M = (0)$ then C is a single point and its frontier is empty. If $M = (1)$, C is an open interval. By the definition of CAD, endpoints of C are (0)-cells.

Now let $n > 1$ and let C have index $M = (i_1, \ldots, i_n)$. The projection $C' = \mathrm{proj}_{\mathbb{R}^{n-1}}(C)$ is a cell in the induced decomposition $\mathcal{D}'$ of $\mathbb{R}^{n-1}$ with index $(i_1, \ldots, i_{n-1})$. By the induction hypothesis, $\mathrm{fr}\,(C')$ is contained in a union of cells of $\mathcal{D}'$ with indices $(j_1, \ldots, j_{n-1}) <_{\mathrm{lex}} (i_1, \ldots, i_{n-1})$. I.e., by definition we have

$$i_1 = j_1, \ldots, i_{k-1} = j_{k-1} \text{ and } i_k < j_k \text{ for some } 1 \leq k \leq n-1. \tag{8.2}$$

If $i_n = 0$, then C is a section cell. Its frontier $\mathrm{fr}\,(C)$ is contained in $\mathrm{fr}\,(C') \times \mathbb{R}$ and, therefore, in a union of cells of $\mathcal{D}$ with indices $(j_1, \ldots, j_{n-1}, j_n)$ for some $j_n \in \{0, 1\}$. Since the property in Equation (8.2) still holds, we have shown that $(j_1, \ldots, j_{n-1}, j_n) <_{\mathrm{lex}} M$.

If $i_n = 1$, then C is a sector cell. In this case, $\mathrm{fr}\,(C)$ is contained in $(\mathrm{fr}\,(C') \times \mathbb{R}) \cup C_T \cup C_B$, hence in the union of cells of $\mathcal{D}$ with indices $(j_1, \ldots, j_{n-1}, j_n)$ for some $j_n \in \{0, 1\}$, and two cells, C_T and C_B, having the same index $(i_1, \ldots, i_{n-1}, 0) <_{\mathrm{lex}} (i_1, \ldots, i_{n-1}, 1)$. All of these indices are lexicographically less than M. □

In the proof of Theorem 8.1, we will need the following technical statement.

Lemma 8.3. *Let $f : X \to \mathbb{R}$ be a continuous definable function on a set $X \subset \mathbb{R}^n$ having the graph $F \subset \mathbb{R}^{n+1}$, such that F is a semialgebraic set. Let G be a semialgebraic subset, closed in F, of dimension less than $\dim(F) = n$. Then $\mathrm{fr}_F(F \setminus G) = G$, where fr_F denotes the frontier in F.*

Proof. Let $Y = \mathrm{proj}_{\mathbb{R}^n}(G)$ and observe that $X = \mathrm{proj}_{\mathbb{R}^n}(F)$ and, since semialgebraic sets are closed under projection, observe that both X and Y are semialgebraic. We will prove that $\mathrm{fr}_X(X \setminus Y) = Y$. By the definition of frontier, we have

$$\mathrm{fr}_X(X \setminus Y) = \mathrm{cl}_X(X \setminus Y) \setminus (X \setminus Y),$$

where cl_X denotes the closure in X.

We need to prove that $\mathrm{cl}_X(X \setminus Y) = X$. Since Y is a semialgebraic set, it is the union of some basic semialgebraic sets

$$S_i = \{\mathbf{x} \in \mathbb{R}^n \mid f_1(\mathbf{x}) = 0, \ldots, f_k(\mathbf{x}) = 0, g_1(\mathbf{x}) > 0, \ldots, g_\ell(\mathbf{x}) > 0\} \tag{8.3}$$

of dimension $< n$. Each S_i is contained in a hypersurface $V_i = V(f_i)$, where f_i is one of the polynomial equations defining S_i in Equation (8.3). Let $Z = \bigcup_i V_i = \bigcup_i V(f_i) = V(\prod f_i)$ be a closed set which contains Y. We may replace Y with Z, and prove that

$$\mathrm{cl}_X(X \setminus Z) = X \supset Z \cap X$$

because $(X \setminus Z) \subset (X \setminus Y)$ implies $\mathrm{cl}_X(X \setminus Z) \subset \mathrm{cl}_X(X \setminus Y)$.

When considering cl_X, we view X as a topological space and consider only the points of Z which are contained in X.

We prove that

$$Z \cap \mathrm{cl}_X(X \setminus Z) \supset Z \cap X$$

and claim that every $\mathbf{x} \in X \cap Z$ is a point of $\mathrm{cl}_X(X \setminus Z)$. I.e., since we view X as a topological space, even if X is not an open set, by the definition of closure, there is a small real number $r > 0$ such that the open ball $B(\mathbf{x}, r)$ is contained in $X \setminus Z$. Assume for contradiction that this is false. Then $B(\mathbf{x}, r) \cap (X \setminus Z) = \emptyset$. It follows that $B(\mathbf{x}, r)$ must be contained in Z, but this is impossible because a set of dimension $< n$ cannot contain an n-dimensional open ball.

Finally, since f is a continuous function, and therefore Y and X are homeomorphic to G and F respectively, we obtain the required formula $\mathrm{fr}_F(F \setminus G) = G$. □

We now prove that the decomposition induced on $\mathbb{R}^k, 1 \leq k \leq n-1$ by any CAD satisfying the frontier condition also satisfies the frontier condition.

Proof. (of Theorem 8.2)

Let $\mathcal{D}$ be the cylindrical decomposition of $\mathbb{R}^n$ which satisfies the frontier condition and let $\mathcal{D}'$ be the decomposition induced by $\mathcal{D}$ on $\mathbb{R}^{n-1}$. Let C' be a cell of $\mathcal{D}'$. We claim that C' satisfies the frontier condition and prove that $\mathrm{cl}\,(C')$ is a union of cells of $\mathcal{D}'$. Consider the cylinder $C' \times \mathbb{R}$. $\mathrm{cl}\,(C' \times \mathbb{R})$ can be written as a first-order Boolean formula

$$(\mathbf{x}', x_n) \in \mathbb{R}^n \mid \forall \varepsilon > 0 \, \exists (\mathbf{y}', y_n) \in C' \times \mathbb{R} \, (\|(\mathbf{x}', x_n) - (\mathbf{y}', y_n)\| < \varepsilon). \tag{8.4}$$

Since $C' \times \mathbb{R}$ is a union of cells of $\mathcal{D}$ and $\mathcal{D}$ satisfies the frontier condition, it follows that every point $(\mathbf{x}', x_n) \in \mathrm{cl}\,(C' \times \mathbb{R})$ belongs to $\mathrm{cl}\,(C)$ for some cell C of $\mathcal{D}$ in $C' \times \mathbb{R}$. Hence, $\mathrm{cl}\,(C' \times \mathbb{R})$ is a union of cells of $\mathcal{D}$.

It is clear that, if $(\mathbf{x}', x_n) \in \mathrm{cl}\,(C' \times \mathbb{R})$, then $\mathrm{proj}_{\mathbb{R}^{n-1}}((\mathbf{x}', x_n)) = \mathbf{x}' \in \mathrm{proj}_{\mathbb{R}^{n-1}}(\mathrm{cl}\,(C' \times \mathbb{R}))$. Hence $\mathrm{proj}_{\mathbb{R}^{n-1}}(\mathrm{cl}\,(C' \times \mathbb{R})) = \mathrm{cl}\,(C')$ and, by the cylindrical property, since $\mathrm{cl}\,(C' \times \mathbb{R})$ is a union of cells of $\mathcal{D}$, $\mathrm{cl}\,(C')$ is a union of cells of $\mathcal{D}'$.

The proof is completed, for decompositions induced by $\mathcal{D}$ on $\mathbb{R}^k, 1 \leq k \leq n-1$, by iterating this argument for successive projections. □

8.3 Main Result

8.3.1 Algorithm Psuedo-Code

We are now ready to prove the main result. We begin by presenting pseudo-code for the algorithm described in Theorem 8.1. First some basic subroutines are defined.

- $\mathcal{E} := \mathcal{D}$ & $\{F_1, \ldots, F_k\}$

  - **Input:** $\mathcal{D}$ is a CAD of $\mathbb{R}^n$ and $\{F_1, \ldots, F_k\}$ is a set of QFFs. Each $F_i, 1 \leq i \leq k$ defines a semialgebraic set $S_i \subset \mathbb{R}^n$. Formulas F_i and those defining cells of $\mathcal{D}$ contain s different polynomials of maximum degree d in $\mathbb{Z}[x_1, \ldots, x_n]$.
  - **Output:** $\mathcal{E}$ is a refinement of $\mathcal{D}$ compatible with each of the sets S_i.
  - **Description:** Construct a "classical" CAD of $\mathbb{R}^n$, compatible with $\{F_1, \ldots, F_k\}$ and all cells of $\mathcal{D}$ using classical CAD algorithm (see Proposition 2.8).
  - **Complexity:** $(sd)^{O(1)^n}$.
- $G := \mathrm{fr}\,(F)$
  - **Input:** F, a QFF defining a semialgebraic set $X \subset \mathbb{R}^n$. Formula F contains s different polynomials of maximum degree d in $\mathbb{Z}[x_1, \ldots, x_n]$.
  - **Output:** G, a QFF defining a semialgebraic set $Y \subset \mathbb{R}^n$ such that Y is the frontier of X.
  - **Description:** Represent the frontier of X as a first-order formula and apply quantifier elimination to obtain G (see Lemma 8.1).
  - **Complexity:** $(sd)^{O(n^2)}$.
- $N :=$ Decrement(M)
  - **Input:** $M \in \{0,1\}^k, k > 0$.
  - **Output:** $N \in \{0,1\}^k$ is the index immediately prior to M with respect to $<_{\mathrm{lex}}$.
  - **Description:** Straightforward. E.g., view M as an unsigned integer in binary and subtract 1.
  - **Complexity:** $O(1)$.

We now present the algorithm from Theorem 8.1.

Algorithm 6. *CadFC*

$$\mathcal{D} := \mathrm{CadFC}(F)$$

Input: *F: a QFF representing a semialgebraic set $S \subset \mathbb{R}^n$.*

Output: *A CAD $\mathcal{D}$ of $\mathbb{R}^n$ compatible with S and satisfying the frontier condition.*

Complexity:

$$(sd)^{O(1)^{n2^n}}$$

where F contains s different polynomials of maximum degree d in $\mathbb{Z}[x_1, \ldots, x_n]$.

- *Initialise $M = (i_1, \ldots, i_{n-1}) := (1, \ldots, 1) \in \{0,1\}^{n-1}$.*
- *Let $\mathcal{E}_M :=$ **true** & $\{\mathbf{F}\}$, where **true** is the trivial CAD of $\mathbb{R}^n$, be the initial CAD.*
- *Define the initial pair $(\mathcal{E}_M, \mathcal{F}_M) := (\mathcal{E}, \emptyset)$.*

  $\mathcal{E}_M$ is a refinement of $\mathcal{E}$ and $\mathcal{F}_M$ is a set of QFFs, each defining a semialgebraic set contained in a union of cells of $\mathcal{E}_M$ with index $\leq_{\mathrm{lex}} (i_1, \ldots, i_{n-1}, 1)$.

- *While* $(0,\ldots,0) <_{\text{lex}} M$*, do*
  - *Let* $N := \text{Decrement}(M)$.
  - *Compute* $\mathcal{E}_N := \mathcal{E}_M$ & $\mathcal{F}_M$.
  - *Let* $\mathcal{A}_0$ *be the family of QFFs defining all cells of* $\mathcal{E}_N$ *with index* $(i_1,\ldots,i_{n-1},0)$
  - *and* $\mathcal{A}_1$ *be the family of QFFs defining all cells of* $\mathcal{E}_N$ *with index* $(i_1,\ldots,i_{n-1},1)$.
  - *Compute the set of frontiers*

$$\begin{aligned}\mathcal{F}_{N,0} &= \{\text{fr}\,(F_0) \mid F_0 \in \mathcal{A}_0\},\\ \mathcal{F}_{N,1} &= \{\text{fr}\,(F_1 \wedge \neg(F_{1,B} \vee F_{1,T})) \mid F_1 \in \mathcal{A}_1, F_{1,B}, F_{1,T} \in \mathcal{A}_0\},\\ \mathcal{F}_N &= \mathcal{F}_{N,0} \cup \mathcal{F}_{N,1},\end{aligned}$$

    where $F_{1,B}$ *(resp.* $F_{1,T}$*) is a QFF defining the bottom (resp. top) of the sector cell defined by* F_1*, or* **false** *if the bottom (resp. top) does not exist.*
  - *Let* $M := N$.
- *Return* $\mathcal{E}_{(0,\ldots,0)}$.

8.3.2 Worked Example

Table 8.1: All cells of the CAD $\mathcal{E}$, the initial CAD computed in Section 8.3.2.}

Label	Index	Formula
	$(1,1,1)$	$\{x < -1\}$
C_1	$(0,1,1)$	$\{x = -1\}$
	$(1,1,1)$	$\{-1 < x < 1, y < \lvert x\rvert\}$
C_2	$(1,0,1)$	$\{-1 < x < 1, y = \lvert x\rvert\}$
	$(1,1,1)$	$\{-1 < x < 1, \lvert x\rvert < y < 1, z < \lvert x/y\rvert\}$
S	$(1,1,0)$	$\{-1 < x < 1, \lvert x\rvert < y < 1, z = \lvert x/y\rvert\}$
	$(1,1,1)$	$\{-1 < x < 1, \lvert x\rvert < y < 1, z > \lvert x/y\rvert\}$
C_3	$(1,0,1)$	$\{-1 < x < 1, y = 1\}$
	$(1,1,1)$	$\{-1 < x < 1, y > 1\}$
C_4	$(0,1,1)$	$\{x = 1\}$
	$(1,1,1)$	$\{x > 1\}$

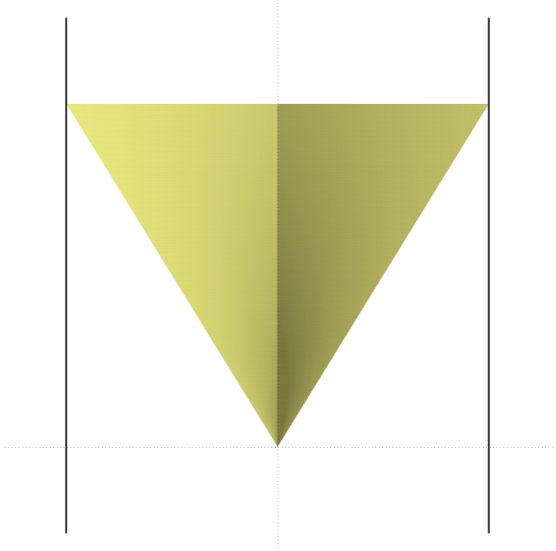

Figure 8.2: Illustrates the initial decomposition constructed in Step 1 (projection onto the (x, y)-plane).

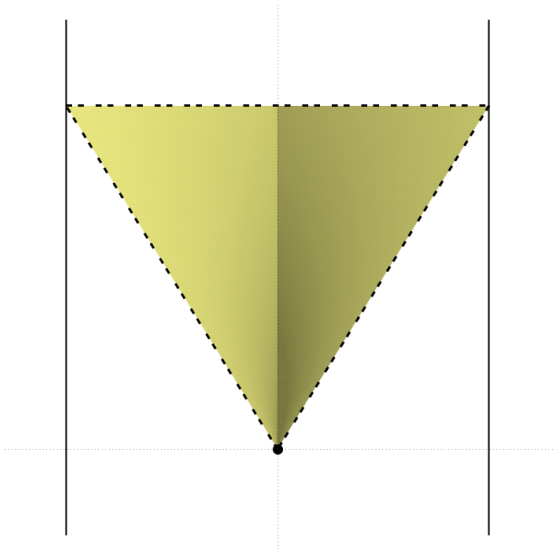

Figure 8.3: Illustrates how $\mathrm{fr}\,(S)$ interacts with the cells of $\mathcal{E}$. The frontier is indicated by a dashed line, with the blow-up subset above the origin represented by a node.

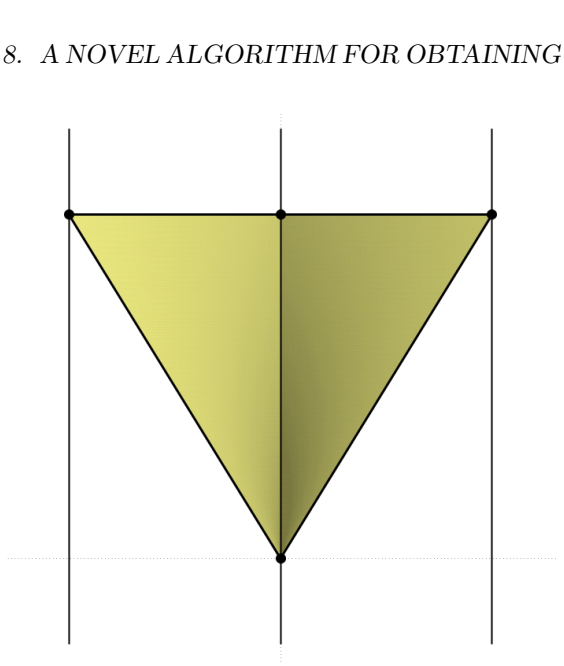

Figure 8.4: Shows cells of $\mathcal{E}_{(0,1)}$, the refinement of $\mathcal{E}$ compatible with $\mathrm{fr}\,(S)$ computed in Step 4.

Table 8.2: Cells of the CAD $\mathcal{E}_{(0,1)}$ computed in step 3 of Section 8.3.2. Note that only cells which are part of $\mathrm{cl}\,(S)$ are listed.

Label	Index	Formula
C'_1	$(0,0,0)$	$\{x=-1, y=1, z=1\}$
C'_2	$(1,0,0)$	$\{-1<x<0, y=-x, z=1\}$
$C''_{2,1}$	$(0,0,0)$	$\{x=y=0, z=0\}$
$C''_{2,2}$	$(0,0,1)$	$\{x=y=0, 0<z<1\}$
$C''_{2,3}$	$(0,0,0)$	$\{x=y=0, z=1\}$
C'''_2	$(1,0,0)$	$\{0<x<1, y=x, z=1\}$
S'	$(1,1,0)$	$\{-1<x<0, -x<y<1, z=-x/y\}$
S''	$(0,1,0)$	$\{x=0, 0<y<1, z=0\}$
S'''	$(1,1,0)$	$\{0<x<1, x<y<1, z=x/y\}$
C'_3	$(1,0,0)$	$\{-1<x<0, y=1, z=-x\}$
C''_3	$(0,0,0)$	$\{x=0, y=1, z=0\}$
C'''_3	$(1,0,0)$	$\{0<x<1, y=1, z=x\}$
C'_4	$(0,0,0)$	$\{x=1, y=1, z=1\}$

We demonstrate the CadFC algorithm by applying it to the set

$$S := \{-1<x<1, |x|<y<1, z=|x/y|\}$$

from Example 8.1.

1. Let F be a QFF defining S and compute $\mathcal{E} := \mathbf{true}\ \&\ \{F\}$, a CAD of $\mathbb{R}^3$ compatible with S (see Table 8.1 and Figure 8.2). Observe that S is a single 2-dimensional cylindrical $(1,1,0)$-cell of $\mathcal{E}$. The initial pair

$$(\mathcal{E}_{(1,1)}, \mathcal{F}_{(1,1)}) = (\mathcal{E}, \emptyset).$$

2. We compute the next pair $(\mathcal{E}_{(1,0)}, \mathcal{F}_{(1,0)})$.

$$\mathcal{E}_{(1,0)} := \mathcal{E}_{(1,1)}\ \&\ \mathcal{F}_{(1,1)} = \mathcal{E}\ \&\ \emptyset = \mathcal{E}.$$

   Denote by $X_{(1,0)}$ the set of frontiers of cells of $\mathcal{E}_{(1,0)}$ with indices $(1,1,1)$ and $(1,1,0)$. This set contains

$$\begin{aligned} \mathrm{fr}\,(S) &= S_B \cup S_T \cup \{(-1,1,1),(1,1,1)\} \\ \text{where } S_T &= \{-1 < x < 1, y = 1, z = |x|\} \\ \text{and } S_B &= \{-1 < x < 0, y = -x, z = 1\} \\ &\cup \{x = y = 0, 0 \le z \le 1\} \\ &\cup \{0 < x < 1, y = x, z = 1\} \end{aligned}$$

   (see Figure 8.3). The family $\mathcal{F}_{(1,0)}$ contains a QFF defining $X_{(1,0)}$. Recall (from Example 8.1) that S_B cannot be a cylindrical cell as it is not the graph of a continuous function.

3. Compute the next pair $(\mathcal{E}_{(0,1)}, \mathcal{F}_{(0,1)})$ as follows. Let

$$\mathcal{E}_{(0,1)} := \mathcal{E}_{(1,0)}\ \&\ \mathcal{F}_{(1,0)}.$$

   The blow-up subset $\{x = y = 0, 0 \le z \le 1\}$ in $X_{(1,0)}$ results in a refinement of the decomposition induced by $\mathcal{E}_{(1,0)}$ on $\mathbb{R}^1$ such that it includes the cells $\{-1 < x < 0\}, (0), \{0 < x < 1\}$. Thus $\mathcal{E}_{(0,1)}$ includes a cell $S'' = \{x = 0, 0 < y < 1, z = 0\}$ and S is split into three cells S', S'', S''' with indices $(1,1,0), (1,0,0)$ and $(1,1,0)$ respectively. Cells C_1, C_2, C_3 and C_4 of $\mathcal{E}_{(1,0)} = \mathcal{E}$ are also refined in this step so that they are compatible with $\mathrm{fr}\,(S)$ (see Table 8.2 and Figure 8.4). As we will argue in the Correctness section, frontiers $\mathrm{fr}_S(S')$ and $\mathrm{fr}_S(S''')$ coincide with S'', so no further refinements of cells with index $(1,1,1)$ and $(1,1,0)$ are needed. $X_{(0,1)}$ contains $\mathrm{fr}\,(C_2'), \mathrm{fr}\,(C_2'''), \mathrm{fr}\,(C_3')$ and $\mathrm{fr}\,(C_3''')$. Thus, $\mathcal{F}_{(0,1)}$ contains QFFs defining $X_{(0,1)}$.

4. In this particular case, $X_{(0,1)}$ is already a union of cells of $\mathcal{E}_{(0,1)}$, so $\mathcal{E}_{(0,0)} = \mathcal{E}_{(0,1)}$ as no refinement is needed. $X_{(0,0)}$ contains $\mathrm{fr}\,(S')$.

5. $X_{(0,0)}$ is already a union of cells of $\mathcal{E}_{(0,0)}$, so no further refinement of $\mathcal{E}_{(0,0)} = \mathcal{E}_{(0,1)}$ is needed. The algorithm terminates and returns $\mathcal{D} := \mathcal{E}_{(0,1)}$.

Well-known implementations of cylindrical algebraic decomposition such as `QEPCAD B` (Brown, 2003) and Maple's `CylindricalAlgebraicDecompose` (Chen and Moreno Maza, 2014) are unable to obtain the frontier condition in general.

In particular, when applied to set S from Example 8.1, both 'CylindricalAlgebraicDecompose' and 'QEPCAD B' create three cells above the origin: $B' = \{x = y = 0, z < 0\}$, $B'' = \{x = y = 0, z = 0\}$ and $B''' = \{x = y = 0, z > 0\}$. Observe that $B''' \cap \mathrm{fr}\,(S) \neq \emptyset$, but $B''' \not\subset \mathrm{fr}\,(S)$ so the frontier of S is not a union of cells in the decomposition.

TODO!! insert the tables.

8.3.3 Mathematical Description

The algorithm takes as input a quantifier-free Boolean formula F representing a semialgebraic set $S \subset \mathbb{R}^n$. Begin by computing a CAD $\mathcal{E}$ compatible with S, by applying the algorithm from Proposition 2.8 to $\{F\}$.

The algorithm computes a sequence of pairs

$$(\mathcal{E}_M,\ \mathcal{F}_M), \tag{8.5}$$

where $M = (i_1, \ldots, i_{n-1}) \in \{0,1\}^{n-1}$ is an index, $\mathcal{E}_M$ is a CAD of $\mathbb{R}^n$ which is a refinement of $\mathcal{E}$ and $\mathcal{F}_M$ is a family of quantifier-free Boolean formulas, each defining a semialgebraic set which is contained in the union of all cells of $\mathcal{E}_M$ with indices $\leq_{\mathrm{lex}} (i_1, \ldots, i_{n-1}, 1)$. We will denote this family of cells by $\mathcal{U}_M$. In equation (8.5), index M refers to the pair rather than its elements.

The sequence of pairs is computed recursively, starting with index $M = (1, 1, \ldots, 1)$, in descending order of indices $M = (i_1, \ldots, i_{n-1})$ with respect to $<_{\mathrm{lex}}$. Let the initial pair

$$(\mathcal{E}_{(1,\ldots,1)},\ \mathcal{F}_{(1,\ldots,1)}) = (\mathcal{E},\ \emptyset).$$

The algorithm will construct the sequence (from right to left):

$$(\mathcal{E}_{\mathbf{0}}, \mathcal{F}_{\mathbf{0}}) <_{\mathrm{lex}} \cdots <_{\mathrm{lex}} (\mathcal{E}_N, \mathcal{F}_N) <_{\mathrm{lex}} (\mathcal{E}_M, \mathcal{F}_M) <_{\mathrm{lex}} \cdots <_{\mathrm{lex}} (\mathcal{E}, \emptyset), \tag{8.6}$$

where $\mathbf{0} = (0, \ldots, 0)$ and $\mathcal{F}_{\mathbf{0}} := \{F \mid F \text{ is a QFF defining } C \in \mathcal{U}_{(0,\ldots,0)}\}$. Note that the $<_{\mathrm{lex}}$ notation is naturally extended to elements of the sequence defined in Equation (8.6).

For a given index $M = (i_1, \ldots, i_{n-1})$, assume that the algorithm has computed the pair $(\mathcal{E}_M,\ \mathbf{F}_M)$. Now we describe how the next pair, $(\mathcal{E}_N,\ \mathcal{F}_N)$, where $N = (j_1, \ldots, j_{n-1})$ is the index immediately prior to M with respect to $<_{\mathrm{lex}}$, is computed. Applying algorithm from Proposition 2.8 to $\mathcal{E}_M$ and $\mathcal{F}_M$, we obtain a CAD $\mathcal{E}_N$ of $\mathbb{R}^n$ compatible with every cell of $\mathcal{E}_M$ and each set defined by QFFs in $\mathcal{F}_M$ (see construction of $\mathcal{E}_{(0,1)}$ in Step 2 of the worked example). Consider the family $\mathcal{A}_0$ of cells in $\mathcal{E}_N$ with indices $(i_1, \ldots, i_{n-1}, 0)$ (section cells) and the family $\mathcal{A}_1$ of cells in $\mathcal{E}_N$ with indices $(i_1, \ldots, i_{n-1}, 1)$ (sector cells). Compute the set of frontiers

$$X_N := \bigcup_{C \in \mathcal{A}_0} \mathrm{fr}\,(C) \cup \bigcup_{C \in \mathcal{A}_1} (\mathrm{fr}\,(C) \setminus (C_T \cup C_B)). \tag{8.7}$$

More precisely, apply the algorithm from Lemma 8.1 to each QFF defining a cell $C \in \mathcal{A}_0$ and to the QFF defining $C \setminus (C_T \cup C_B)$ where $C \in \mathcal{A}_1$. Observe that C_T, C_B are already $(i_1, \ldots, i_{n-1}, 0)$-cells in $\mathcal{A}_0$. According to Lemma 8.2, X_N is contained in the union of cells in $\mathcal{E}_N$ with indices $\leq_{\text{lex}} (j_1, \ldots, j_{n-1}, 1) <_{\text{lex}} (i_1, \ldots, i_{n-1}, 0)$ (see construction of $X_{(1,0)}$ in Step 1 of the worked example).

When the algorithm reaches the final pair $(\mathcal{E}_{\mathbf{0}}, \mathcal{F}_{\mathbf{0}})$ in the sequence, it computes a CAD $\mathcal{D}$, using the algorithm from Proposition 2.8, compatible with $\mathcal{F}_{\mathbf{0}}$ and every cell of $\mathcal{E}_{\mathbf{0}}$. The algorithm terminates by returning $\mathcal{D}$.

8.3.4 Proof of correctness

Proof. (of Theorem 8.1)

Let $L = (k_1, \ldots, k_{n-1})$ be the index immediately prior to $N = (j_1, \ldots, j_{n-1})$ with respect to $<_{\text{lex}}$. The algorithm computes $\mathcal{E}_L$ as the refinement of $\mathcal{E}_N$ compatible with each set represented by a formula in $\mathcal{F}_N$. This is the set X_N defined in Equation (8.7). Suppose that the algorithm has computed the final decomposition $\mathcal{D}$, as the refinement of $\mathcal{E}_{\mathbf{0}}$ compatible with $\mathcal{F}_{\mathbf{0}}$ in the sequence 8.6. Now construct a new sequence of decompositions (from left to right)

$$\mathcal{E}'_{\mathbf{0}} <_{\text{lex}} \cdots <_{\text{lex}} \mathcal{E}'_L <_{\text{lex}} \mathcal{E}'_N <_{\text{lex}} \mathcal{E}'_M <_{\text{lex}} \cdots <_{\text{lex}} \mathcal{E}'_{(1,\ldots,1)} \tag{8.8}$$

where the initial element $\mathcal{E}'_{\mathbf{0}}$ is the refinement of $\mathcal{E}_{\mathbf{0}}$ compatible with each cell of $\mathcal{D}$ and each $\mathcal{E}'_I$ is the refinement of $\mathcal{E}_I$ compatible with all cells of $\mathcal{E}'_J$, where J is the index immediately prior to I with respect to $<_{\text{lex}}$. We want to prove that every cell in $\mathcal{E}'_L$ with index $\leq_{\text{lex}} (i_1, \ldots, i_{n-1}, 1)$ satisfies the frontier condition.

If C is a cell in $\mathcal{E}_N$ with index $(i_1, \ldots, i_{n-1}, 0)$ (section cell), then C is a union of cells of $\mathcal{E}_L$ and, hence, of $\mathcal{E}'_L$, with indices $\leq_{\text{lex}} (i_1, \ldots, i_{n-1}, 0)$. Let $C' \subset C$ be one of the cells in this refinement of C with index $(i_1, \ldots, i_{n-1}, 0)$ and B be the union of cells contained in the refinement of C with indices $<_{\text{lex}} (i_1, \ldots, i_{n-1}, 0)$. Let $B' \subset B$ be a cell of $\mathcal{E}_L$ with index $(m_1, \ldots, m_{n-1}, 0) <_{\text{lex}} (i_1, \ldots, i_{n-1}, 0)$. We claim that $\dim(B') < \dim(C)$. Indeed, on the one hand, if $i_k = 0$, for some $1 \leq k \leq n-1$, then $m_k = 0$ as well. Otherwise, $\text{proj}_{\mathbb{R}^k}(C)$ is a section cell which must contain $\text{proj}_{\mathbb{R}^k}(B')$, which is a sector cell, but this is impossible. On the other hand, by the definition of $<_{\text{lex}}$, there exists some $1 \leq \ell \leq n-1$ such that $m_1 = i_1, \ldots, m_{\ell-1} = i_{\ell-1}$ and $m_\ell = 0, i_\ell = 1$. This implies that $\dim(B') = m_1 + \cdots + m_{n-1} < i_1 + \cdots + m_{n-1} = \dim(C)$ as required. It follows that $\dim(B) < \dim(C)$ and B is closed in C. According to Lemma 8.3, $\text{fr}_C(C \setminus B) = B$ where $\text{fr}_X(Y)$ denotes the frontier of Y in X. Hence, $\text{fr}_C(C') \subset B$. On the other hand, $\text{fr}\,(C') \setminus \text{fr}_C(C')$ is a subset of $\text{fr}\,(C)$, which is a union of cells of $\mathcal{E}_L$ (and of the refinement $\mathcal{E}'_L$) with index $<_{\text{lex}} (i_1, \ldots, i_{n-1}, 0)$.

By the induction hypothesis, all cells of $\mathcal{E}'_L$ with index $<_{\text{lex}} (i_1, \ldots, i_{n-1}, 0)$ satisfy the frontier condition. In particular, $\text{fr}\,(C)$ and $\text{fr}\,(B) \subset \text{fr}\,(C)$ is a union of cells of $\mathcal{E}'_L$. It follows from the cylindrical structure of $\mathcal{E}'_L$ that C' satisfies the frontier condition.

If C is a cell in $\mathcal{E}_N$ with index $(i_1, \ldots, i_{n-1}, 1)$ (sector cell), then C is a union of cells of $\mathcal{E}_L$ with indices $\leq_{\text{lex}} (i_1, \ldots, i_{n-1}, 1)$. A similar argument to that used for section cells shows that each cell C' in this union, having index $(i_1, \ldots, i_{n-1}, 1)$, satisfies the frontier condition in $\mathcal{E}'_L$. Note that when using Lemma 8.3 we consider sector cell C as a graph of a constant function over itself.

Finally, each decomposition $\mathcal{E}'_I$, $I \in \{0,1\}^{n-1}$, in the sequence 8.8 coincides with $\mathcal{D}$. Indeed, $\mathcal{D}$ is a refinement of $\mathcal{E}_I$ and $\mathcal{E}'_I$ is a refinement of $\mathcal{E}_I$ compatible with all cells of $\mathcal{D}$. In other words, no refinement of $\mathcal{D}$ is required to obtain $\mathcal{E}'_{(1,\ldots,1)}$. It follows that every cell of $\mathcal{D}$ satisfies the frontier condition. □

8.3.5 Complexity

The number of different indices $(i_1, \ldots, i_n)$, where each $i_k \in \{0,1\}$, is 2^n. Therefore, the algorithm makes $O(2^n)$ "steps' ': computing successive pairs $(\mathcal{E}_M, \mathcal{F}_M)$ in the sequence (8.6). In each step, we compute the next pair $(\mathcal{E}_N,\ \mathcal{F}_N)$ from $(\mathcal{E}_M,\ \mathcal{F}_M)$, using Proposition 2.8 and Lemma 8.1.

First, Proposition 2.8 is applied to compute $\mathcal{E}_N$, the CAD of $\mathbb{R}^n$ compatible with all cells of $\mathcal{E}_M$ and the sets defined by formulas in $\mathcal{F}_M$. The complexity of this operation depends on the number of polynomials, s_M, and their maximum degree, d_M, defining $(\mathcal{E}_M,\ \mathcal{F}_M)$. The complexity of this operation is

$$(s_M d_M)^{O(1)^n}.$$

Since this is also an upper bound on the number of cells, number of polynomials and their degrees defining $\mathcal{E}_N$, let

$$s_N := (s_M d_M)^{O(1)^n} \text{ and } d_N := (s_M d_M)^{O(1)^n}.$$

We now compute the set of frontiers $\mathcal{F}_N$ of cells of $\mathcal{E}_M$ with index $(i_1, \ldots, i_{n-1}, i_n)$. The complexity of applying Lemma 8.1 is $(s_M d_M)^{n^2 O(1)^n}$, which is asymptotically the same as $(s_M d_M)^{O(1)^n}$. Thus, the overall complexity of this "step" is again

$$(s_M d_M)^{O(1)^n}.$$

The overall complexity is obtained by iterating this process for each index. E.g., the complexity of passing from $(\mathcal{E}_M\, \mathcal{F}_M)$ to $(\mathcal{E}_L\, \mathcal{F}_L)$, where $M <_{\text{lex}} N <_{\text{lex}} L$ are subsequent indices in the lexicographical order, would be

$$(s_N d_N)^{O(1)^n} = \left((s_M d_M)^{O(1)^n} (s_M d_M)^{O(1)^n}\right)^{O(1)^n}$$

Iterating this 2^n times, with initial parameters $s_{(1,\ldots,1)} = s$ and $d_{(1,\ldots,1)} = d$ being the number of input polynomials and their maximum degree, we obtain an overall complexity of

$$(sd)^{O(1)^{n 2^n}}.$$

According to Proposition 2.8 and Lemma 8.1, this is also an upper bound on the number of cells in $\mathcal{D} = \mathcal{E}_{\mathbf{0}}$, the number of polynomials defining cells and their degrees.

8.3.6 A Note on Constructing the Intermediate Decompositions

In the algorithm described in Theorem 8.1, we construct, for each index $M \in \{0,1\}^{n-1}$, a CAD compatible with $\mathcal{F}_M = \{F_1, \ldots, F_k\}$ and all cells of $\mathcal{E}_M$. This can be achieved by constructing a CAD, using Proposition 2.8, such that each cell has constant sign on the polynomials in formulas $\mathcal{F}_M$ and all cells of $\mathcal{E}_M$. Such a CAD is clearly compatible with sets defined by formulas in $\mathcal{F}_M$ and all cells of $\mathcal{E}_M$. However, it may include some cells, outside $\mathrm{cl}\,(S)$, which we are not interested in.

Fewer cells may be produced by computing a CAD compatible with $\{F_1, \ldots, F_k, C_1, \ldots, C_r\}$ where $C_i, 1 \leq i \leq r$ is a cell of $\mathcal{E}$ such that $\bigcup_{1 \leq i \leq r} C_i = \mathbb{R}^n$. This CAD is obviously compatible with all cells of $\mathcal{E}_M$ and is compatible with the required sets.

Both options have the same asymptotic complexity. Therefore, the choice of CAD subroutine does not change the complexity bound obtained in the proof of Theorem 8.1. We apply Proposition 2.8 to compute a CAD compatible with the input set $S \subset \mathbb{R}^n$ for the initial decomposition $\mathcal{E}$ The refinements are computed by applying a variant of the classical CAD algorithm which constructs a decomposition compatible with a family of sets.

In the algorithm description, we pass from semialgebraic sets (and CAD cells) to quantifier-free Boolean formulas. This is clearly possible by Thom's Lemma. However, the signs of polynomials on which each CAD cell has constant sign may not be sufficient to properly define every cell in the CAD. Signs of polynomials defining the CAD will be sufficient to define every cell if Collins' augmented projection is used {Collins (1975)]. The drawback, of course, is that more polynomials will be produced in the projection phase, which leads to more work during the lifting phase. Brown (1999) also presents an algorithm that ensures every cell can be represented by a formula containing projection polynomials and, possibly, a small number of derivatives. This may be preferable since fewer extra polynomials are needed.

8.4 Generalisations and further work

8.4.1 Pfaffian Functions

Theorem 8.1 can be extended to restricted sub-Pfaffian sets as described by Gabrielov and Vorobjov (2004). In particular, it can be used to prove the existence of decompositions with frontier condition compatible with semialgebraic sets defined by *fewnomials* (Gabrielov and Vorobjov, 2004, Section 2.6), whose structure is destroyed by a change of coordinates, and obtain a bound on the number of cells in these decompositions.

The idea of the proof is still valid. Only a modification to the subroutines for computing the frontier, and for constructing a classical CAD, is needed. For the

former, replace the quantifier elimination algorithm in Lemma 8.1 with the result of (Gabrielov and Vorobjov, 2004, Section 5). In the latter case, the classical CAD algorithm from Proposition 2.8 should be replaced by the main result of Gabrielov and Vorobjov (2001) (see also (Gabrielov and Vorobjov, 2004, Section 7)). The rest of the proof can be reproduced almost identically. Care must be taken to ensure that Lemma 8.3 is still applicable to restricted sub-Pfaffian sets. The projection of a sub-Pfaffian set is a semi-Pfaffian set, which admits the representation as a finite union of basic semi-Pfaffian sets. In fact, basic sets can be defined using any functions definable in $\mathbb{R}^n$ and it is clear that any finite union of sets defined by conjunctions of strict or non-strict inequalities is the finite union of basic sets. The representation of Z as the union of hypersurfaces containing basic sets is also possible, and the argument that open balls cannot be contained in a set of smaller dimension is geometric and still holds. Thus, the proof of Lemma 8.3 can be reproduced almost identically, too.

Note that it is not currently clear how this result can be implemented since in the algorithm for cylindrical decomposition by Gabrielov and Vorobjov (2001) the oracle is needed to decide whether a sub-Pfaffian set is empty or not.

8.4.2 Further Wrok

The complexity upper bound of the algorithm from Theorem 8.1 is significantly worse than the bound for classical CAD algorithm due to Collins (1975) (and Wüthrich (1976)). This is caused by the parameter of its recursive loop: the index of a cylindrical cell, which is exponential in the ambient dimension. It is difficult to see another parameter that could make the recursion significantly shorter. Thus, if any progress is to be made towards a better asymptotic complexity, the method used may need to be based on a completely different ideas. On the other hand, there is a lower bound on complexity of the classical CAD algorithm due to Davenport and Heintz (1988). However, it is not yet known whether the lower bound for CAD with frontier condition is greater than that of the classical CAD. Another strand of research could be to explore this question, possibly by attempting to raise this lower bound for CAD with frontier condition.

Another improvement in the algorithm might come from a different subroutine for computing the frontier of a cylindrical cell. The subroutine from Lemma 8.1 works for any semialgebraic set and does not take advantage of its cylindrical structure. We may be able to exploit this structure by factorising the equational part of the formula representing a cell into irreducible components, generalising a method due to Lazard (2010).

Finally, since some initial CAD is refined repeatedly, the classical CAD algorithm from Proposition 2.8 could be replaced with an incremental algorithm, E.g., the algorithm presented by Kremer and Ábrahám (2020). It is not clear whether substituting the CAD algorithm will reduce the complexity bound. The set X_N is still computed at each step, using a singly exponential algorithm, and the initial CAD may need to be refined at every step and this refinement may result in the

incremental algorithm backtracking all the way to the decomposition induced on $\mathbb{R}^1$. However, this change would be very useful in practice as unnecessary CAD re-computations could be avoided, E.g., if $\mathcal{E}_N$ is already compatible with the set X_N, the algorithm in its current form will compute the refinement even though it is not needed.

Chapter 9

Conclusions and Further Work

9.1 Conclusions

This thesis makes three main contributions:

1. an implementation of a smooth stratification algorithm, based on the work of Gabrielov and Vorobjov (1995);
2. a concrete algorithm and implementation for constructing a CAD, monotone with respect to semialgebraic sets $V_1, \ldots, V_k \subset \mathbb{R}^n$ such that $\dim(V_i) \leq 2$, and satisfying the frontier condition;
3. and a novel algorithm for constructing a CAD, satisfying the frontier condition and compatible with a semialgebraic set $S \subset \mathbb{R}^n$ of arbitrary dimension.

Complexity results for each of the algorithms are presented, and the first two algorithms have been implemented in C, on top of `saclib`.

9.1.1 Smooth stratification of a semialgebraic set

Chapter 3 presents the algorithm from Gabrielov and Vorobjov (1995), Theorem 1 which computes a smooth stratification of a semi-Pfaffian set $X \subset \mathbb{R}^n$ and then specialises it to the semialgebraic case, where the Oracle is not required, since it is always possible to check whether a semialgebraic set is empty. The main contribution of this section is a concrete algorithm and its implementation on top of `saclib`, which uses `QEPCAD-B` to determine whether candidate strata are empty. The implementation also ensures that each strata is defined "nicely", i.e., every stratum of codimension k is defined by exactly k polynomial equations and, possibly, some inequalities. Complexity bounds, based on Gabrielov and Vorobjov (1995) and a singly exponential algorithm for deciding emptiness due

to Basu et al. (1998) are presented, including the bound on the number of strata, along with the maximum degree and number of polynomials required to define each stratum.

9.1.2 An algorithm to construct monotone CAD compatible with a family of semialgebraic sets of dimension at most 2

Basu et al. (2015), Theorem 3.20 gives a constructive proof that there exists a CAD, monotone with respect to a collection of bounded definable sets $V_1, \ldots, V_k$, each of which has dimension at most 2, and satisfying the frontier condition. The main contribution of Chapters 4, 5 and 6 is the design and implementation of an algorithm, based on Basu et al. (2015), Theorem 3.20, for constructing such a CAD. Following the construction due to Basu et al. (2015), the algorithm proceeds in three steps.

First, a CAD, compatible with each of $V_1, \ldots, V_k$ and such that each cell $C \subset V_1 \cup \ldots \cup V_k$ is the graph of a quasi-affine map, following Basu et al. (2015) Lemma 3.19, is constructed. This lemma first instructs us to consider W, the smooth 2-dimensional locus of $V_1 \cup \ldots \cup V_k$. This can be done by applying the smooth stratification algorithm, as presented in Section 3, and collecting the strata of codimension $n-2$. It was then observed that, according to McCallum (1998) Theorems 2.2.3 and 2.2.4, if McCallum's projection operator is used and there is only a finite number of blow-up points, smooth cells are obtained automatically. Basu et al. (2015), Lemma 3.19 then tells us to construct a CAD which is compatible with all $V_1, \ldots, V_k$ and the critical points of the projections of W onto one- and two-dimensional coordinate subspaces. This can be done by ensuring that the CAD has constant sign on all polynomials defining the sets $V_1, \ldots, V_k$, along with some additional polynomials which arise from computing the determinants of Jacobi matrices defining either the (smooth) 2-dimensional cells, or 2-dimensional strata. This completes the construction described in Basu et al. (2015), Lemma 3.19. Pseudo-code is presented and a complexity bound of $(s(d+1))^{O(1)^n}$, which is slightly worse than classical CAD, is obtained.

Next, Chapter 5 describes how some simple refinements of the quasi-affine CAD can be performed such that each cell $C \subset V_1 \cup \ldots \cup V_k$ is monotone. These refinements, described in Basu et al. (2015), Lemma 3.11, are of the kind $\{x_1 = c\}, \{x_1 > c\}, \{x_1 < c\}$, where $c \in \mathbb{R}$. Since the real roots of polynomials are always algebraic numbers, in the semialgebraic case, c is always an element of $\mathbb{A}$. These refinements are easy to perform, as they simply split (1)-cells, and, by Basu et al. (2015) Lemma 3.11, this type of refinement preserves the cylindrical structure along with existing monotone cells. Basu et al. (2015), Lemma 3.18, defines these refinement points as those which ensure that the top and bottom of every 2-dimensional cell contained in $V_1 \cup \ldots V_k$ is the graph of a continuous, definable, monotone map. The main contribution of this Chapter is the application of Lagrange Multipliers to find these refinement points. This

process relies on computing determinants of square matrices of polynomials and finding the first coordinate of their real root – straightforward tools which are readily available in almost every computer algebra system. Again, pseudo-code is presented and a complexity result, which is the same as that presented in Chapter 4, is obtained.

Finally, Chapter 6 explains how to refine the CAD such that it satisfies the frontier condition. The first approach is a direct translation of the last part of Basu et al. (2015), Theorem 3.20. I.e., the frontiers of 2-dimensional cells are computed and then partitioned into points and monotone curve intervals. The latter part of this construction, partitioning a one-dimensional curve into monotone pieces – is straightforward, as it has already been done when constructing monotone cells in the previous chapter. However, the task of computing the frontier of a semialgebraic set is not as easy. Lemma 8.1 describes how to do this using quantifier elimination, and asserts that it has complexity singly exponential in the number of variables. However, the algorithm presented in (Basu et al., 2006, Algorithm 14.21) is theoretical in nature and, in practice, we would have to resort to doubly exponential CAD to solve the QE problem. Endeavouring to avoid this, we turned to Lazard (2010), who presents an algorithm for computing a CAD of $\mathbb{R}^3$ with frontier condition. This algorithm relies on a CAD, containing only topologically regular cells, and having constant sign on a set of polynomials to be given. Since this has already been done (in Chapters 4 and 5), Lazard's algorithm can be applied directly in dimension 3. Furthermore, if 2-dimensional cells are contained in a sub-CAD of $\mathbb{R}^3$ above a 0-dimensional cell, hen Lazard's algorithm can, again, be applied directly to the sub-CAD. The main contribution of Chapter 6 is the observation that a similar situation to that described by Lazard exists in the special case of 2-dimensional cells in a CAD of $\mathbb{R}^n$ and the suggestion of an extension of Lazard's method to this situation. This is the approach taken in the implementation, since it is more tractable than computing the frontier of every cell. Pseudo-code for the extension of Lazard's algorithm and a complexity analysis for both Gabrielov and Vorobjov's approach and Lazard's approach is performed. The complexity, for both approaches, is $(s(d+1))^{O(1)^n}$, which is the overall complexity of constructing the CAD with the properties from Basu et al. (2015), Theorem 3.20.

Thus, a concrete description of an algorithm to construct the cylindrical decomposition, in the semialgebraic case, described in Basu et al. (2015), Theorem 3.20 is given, which has complexity only slightly worse than classical CAD. Chapter 7 is concerned with the implementation details of the this algorithm, along with presenting some test cases which give an exposition of its three stages: quasi-affine, monotone and frontier condition.

9.1.3 A Novel algorithm for computing a CAD with frontier condition, compatible with a semialgebraic set of arbitrary dimension

The final contribution is an algorithm, having ELEMENTARY complexity, for constructing a CAD, compatible with a semialgebraic set of arbitrary dimension and satisfying the frontier condition. This algorithm removes the restriction imposed by Davenport et al. (2020), as blow-ups are permitted, and does not require the rotation of coordinates proposed by Schwartz and Sharir (1983). It was not clear how this could be achieved using the usual induction – on the ambient dimension – so a novel approach was needed. The algorithm relies on a recursion on the lexicographical order of cell index $(i_1, \ldots, i_n) \in \{0, 1\}^n$. A mathematical description, proof of correctness, complexity analysis and pseudo-code are presented for this algorithm, which has complexity doubly exponential in the number of variables. This work was presented in extended abstract form at CASC 2023 and is currently under review for publication in the Journal of Symbolic Computation. This algorithm has not yet been implemented due to time constraints, but it is clear from the pseudo-code that it could be done using the tools we have been using throughout this thesis.

9.2 Further Work

Another fundamental construction in real algebraic geometry is triangulation – which partitions a set into a collection of simplices (a simplicial complex). The main result of Basu et al. (2015) relates to definable monotone families. A family $\{S_\delta\}_{\delta>0}$ is called monotone if sets S_δ are monotone increasing as $\delta \to 0$, i.e., $S_\delta \subset S_\eta$ for all sufficiently small $\delta > \eta > 0$. It is often useful to approximate definable sets $K \subset \mathbb{R}^n$ by definable monotone families S_δ. However, when blow-up points are present, these families can exibit a complex behaviour. Basu et al. (2015) prove that there exists a triangulation of every definable set $K \subset \mathbb{R}^n$, having dimension at most two, such that in each 2-simplex, the family belongs to one of five types, corresponding to one of five possible lexicographically monotone Boolean functions in two variables. The proof of the classification in Basu et al. (2015) is non-constructive. A natural progression of this project would be to design and implement an efficient algorithm computing a triangulation for the input $(K, \{S_\delta\})$. An essential step in the computation of this triangulation is the construction of a CAD with monotone cells and satisfying the frontier condition. Furthermore, a linear change of coordinates, as suggested by, e.g., Schwartz and Sharir (1983), is not allowed. Therefore, the monotone CAD algorithm developed in this thesis will be needed as a subroutine for this algorithm.

It is conjectured that this classification can be extended to n-dimensional sets K. Another direction of research would be to prove this conjecture by generalising the results of Basu et al. (2015), and design an algorithm for computing such a triangulation in the case of sets with arbitrary dimension. The first step would be to generalise the monotone CAD algorithm to sets with higher dimension. It

is not currently clear how this can be done, but one might want to start with sets of dimension 3. In Basu et al. (rint) an algorithm-like procedure was proposed for constructing a decomposition of a 3-dimensional K into topologically regular cylindrical cells which may not form an overall cylindrical decomposition, as cells may be cylindrical with respect to different orderings of variables. The first step in solving this problem may be to implement this procedure or, possibly, strengthen it to obtain a decomposition which is cylindrical with respect to the given coordinate ordering. The results in Section 8 make some strides towards the generalisation, allowing us to obtain the frontier condition in CADs compatible with sets of any dimension.

In addition, there are open questions, both theoretical and practical, relating to the novel algorithm for obtaining the frontier condition, presented in Section 8. For example, one might try to tighten the triply exponential complexity bound, by exploring whether it could be lowered, or by proving that the bound on CAD with frontier condition is higher than the doubly exponential bound obtained by Davenport and Heintz (1988). Practically speaking, the algorithm presented in Section 8 should be implemented, following the suggestions made in Section 8.4.2, and tested with some examples of small dimension.

Finally, the implementation of the monotone CAD algorithm is a proof of concept and, as such, there is plenty of room for improvement where efficiency is concerned. For example, in order to compute the x_1-coordinates of the solutions to a system of multivariate polynomial equations. CAD projection is used. It is well-known that a doubly exponential number of polynomials may result. It may be more efficient to use other tools, e.g., Groebner bases, to compute these x_1-coordinates.

Appendix A: Test Cases

We first present a few variations of Example 7.2, to test various situations. The following Example 9.1 requires an algebraic refinement point.

Example 9.1. Let $n = 3$ and consider the 2-dimensional section cell

$$C := \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, y > 0, y - x^2 + 2x < 1, z = x^2 + y^2\}.$$

```
Before Solution >
d-proj-fac
A_1,1  = input
       = x

A_1,2  = fac2(J_1,1) = fac2(res(A_2,1|A_2,2))
       = x - 1

M_1,3  = fac(K_1,3) = fac(input)
       = 2 x^3 - 6 x^2 + 7 x - 2

M_1,4  = fac(K_1,6) = fac(input)
       = 8 x - 9


A_2,1  = input
       = y

A_2,2  = input
       = y - x^2 + 2 x - 1


A_3,1  = input
       = z - y^2 - x^2
```

```
Before Solution >
d-ref
M_1,1  = input  *** Refinement of cell (3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of 2 x^3 - 6 x^2 + 7 x - 2 between 1/4 and 1/2
      = 0.4102+

Coordinate 1 = alpha
             = 0.4102+




Before Solution >
d-true-cells-t
---------- Information about the cell (5,3,2) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 > 0
  8 x - 9 < 0
Level 2
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

*** Initialising the RCAD. ***

Signs of projection factors (for defining formula) -

Index in RCAD: (5,3,2)
Level 1
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 > 0
  8 x - 9 < 0
Level 2
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 3
```

```
  z - y^2 - x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 1/16
             = 0.0625000000
Coordinate 3 = 65/256
             = 0.2539062500


------------------------------------------------------
---------- Information about the cell (4,3,2) (Dimension (0,1,0) (1))

Signs of projection factors -------------------------

Level 1
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 = 0
  8 x - 9 > 0
Level 2
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,3,2)
Level 1
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 = 0
  8 x - 9 > 0
Level 2
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 3
  z - y^2 - x^2 = 0
```

```
Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of 2 x^3 - 6 x^2 + 7 x - 2 between 3/8 and 7/16
      = 0.4102454877-

Coordinate 1 = alpha
             = 0.4102454877-
Coordinate 2 = 1/8
             = 0.1250000000
Coordinate 3 = the unique root of 262144 x^3 - 536576 x^2 + 1654976 x - 287873 between
             = the unique root of 262144 x^3 - 536576 x^2 + 1654976 x - 287873 between
             = 0.1839263602-


--------------------------------------------------
---------- Information about the cell (3,3,2) (Dimension (1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 < 0
  8 x - 9 < 0
Level 2
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 3
  z - y^2 - x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,3,2)
Level 1
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 < 0
  8 x - 9 < 0
Level 2
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 3
  z - y^2 - x^2 = 0
```

```
Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 3/16
             = 0.1875000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = 25/256
             = 0.0976562500


------------------------------------------------

Before Solution >
```

Note that two refinement polynomials are produced, but only one refinement point. `M_1_4` lies outside of $-1 < x < 1$, so it is ignored.

Example 9.2, which follows, demonstrates what happens on sub-CADs above 0-dimensional cells.

Example 9.2. Let $n = 4$ and consider the semialgebraic set defined by the QFF

$$F := ((4w = 7 \vee w^2 = 36) \wedge x > 0 \wedge x < 1 \wedge y > 0 \wedge x + y < 1 \wedge z = wx^2 + y^2).$$

It has three connected components, C_1, C_2 and C_3, each of which is a cylindrical section cell in a sub-CAD of $\mathbb{R}^3$ above a 0-dimensional cell ($\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ respectively).

$$\mathbf{b}_1 = 7/4, \quad C_1 = \{(x, y, z) \in \mathbb{R}^3 \mid x > 0, x < 1, y > 0, x + y < 1, z = 7/4x^2 + y^2\}. \tag{9.1}$$

$$\mathbf{b}_2 = -6, \quad C_3 = \{(x, y, z) \in \mathbb{R}^3 \mid x > 0, x < 1, y > 0, x + y < 1, z = -6x^2 + y^2\}. \tag{9.2}$$

$$\mathbf{b}_3 = 6, \quad C_3 = \{(x, y, z) \in \mathbb{R}^3 \mid x > 0, x < 1, y > 0, x + y < 1, z = 6x^2 + y^2\}. \tag{9.3}$$

C_2 is already monotone. This can be seen by examining the projections of its top and bottom onto span$\{x, z\}$:

$$C_{2,B} = \{-1 < x < 1, z = -6x^2\}, \tag{9.4}$$

$$C_{2,T} = \{-1 < x < 1, z = -5x - 2x + 1\}. \tag{9.5}$$

C_1 and C_3 are of the same kind as the cell presented in Example 7.2 and require refinement.

```
Before Solution >
d-proj-fac
A_1,1  = input
       = 4 w - 7

A_1,2  = input
       = w - 6

A_1,3  = input
       = w + 6

Q_1,4  = fac(Q_2,2) = fac(input)
       = w


A_2,1  = input
       = x

A_2,2  = fac(J_2,1) = fac(res(A_3,1|A_3,2))
       = x - 1

M_2,3  = fac(K_2,4) = fac(input)
       = 5 x + 1

M_2,4  = fac(K_2,6) = fac(input)
       = 8 x - 1

M_2,5  = fac(K_2,7) = fac(input)
       = 7 x - 1


A_3,1  = input
       = y

A_3,2  = input
       = y + x - 1


A_4,1  = input
       = z - y^2 - w x^2
```

```
Before Solution >
d-ref

M_2,2  = input  *** Refinement of cell (8,3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000

Coordinate 1 = 6
             = 6.0000
Coordinate 2 = 1/7
             = 0.1429-


M_2,1  = input  *** Refinement of cell (6,3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000

Coordinate 1 = 7/4
             = 1.7500
Coordinate 2 = 1/8
             = 0.1250




Before Solution >
d-true-cells-t
---------- Information about the cell (8,5,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  4 w - 7 > 0
  w - 6 = 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 > 0
```

```
  7 x - 1 > 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

*** Initialising the RCAD. ***

Signs of projection factors (for defining formula) -

Index in RCAD: (8,5,3,2)
Level 1
  4 w - 7 > 0
  w - 6 = 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 > 0
  7 x - 1 > 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 6
             = 6.0000000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 7/16
             = 0.4375000000
```

```
-----------------------------------------------------
---------- Information about the cell (8,4,3,2) (Dimension (0,0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  4 w - 7 > 0
  w - 6 = 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 > 0
  7 x - 1 = 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (8,4,3,2)
Level 1
  4 w - 7 > 0
  w - 6 = 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 > 0
  7 x - 1 = 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.
```

```
alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 6
             = 6.0000000000
Coordinate 2 = 1/7
             = 0.1428571429-
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 145/784
             = 0.1849489796-


-----------------------------------------------------
---------- Information about the cell (8,3,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  4 w - 7 > 0
  w - 6 = 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 < 0
  7 x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (8,3,3,2)
Level 1
  4 w - 7 > 0
  w - 6 = 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
```

```
  5 x + 1 > 0
  8 x - 1 < 0
  7 x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 6
             = 6.0000000000
Coordinate 2 = 1/14
             = 0.0714285714+
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 73/784
             = 0.0931122449-


--------------------------------------------------
---------- Information about the cell (6,5,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  4 w - 7 = 0
  w - 6 < 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 > 0
  7 x - 1 > 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
```

```
  z - y^2 - w x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,5,3,2)
Level 1
  4 w - 7 = 0
  w - 6 < 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 > 0
  7 x - 1 > 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 7/4
             = 1.7500000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 11/64
             = 0.1718750000


------------------------------------------------------
---------- Information about the cell (6,4,3,2) (Dimension (0,0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  4 w - 7 = 0
```

```
  w - 6 < 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 = 0
  7 x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,4,3,2)
Level 1
  4 w - 7 = 0
  w - 6 < 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 = 0
  7 x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 7/4
             = 1.7500000000
Coordinate 2 = 1/8
             = 0.1250000000
```

```
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 23/256
             = 0.0898437500


-----------------------------------------------------
---------- Information about the cell (6,3,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors -------------------------

Level 1
  4 w - 7 = 0
  w - 6 < 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 < 0
  7 x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,3,3,2)
Level 1
  4 w - 7 = 0
  w - 6 < 0
  w + 6 > 0
  w > 0
Level 2
  x > 0
  x - 1 < 0
  5 x + 1 > 0
  8 x - 1 < 0
  7 x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
```

```
  z - y^2 - w x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 7/4
             = 1.7500000000
Coordinate 2 = 1/16
             = 0.0625000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 71/1024
             = 0.0693359375


--------------------------------------------------
---------- Information about the cell (2,3,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  4 w - 7 < 0
  w - 6 < 0
  w + 6 = 0
  w < 0
Level 2
  x > 0
  x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,3,3,2)
Level 1
  4 w - 7 < 0
  w - 6 < 0
  w + 6 = 0
  w < 0
```

```
Level 2
  x > 0
  x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - y^2 - w x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = -6
             = -6.0000000000
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = -5/16
             = -0.3125000000


--------------------------------------------------

Before Solution >
```

The following example requires a rational refinement above an algebraic zero-cell.

Example 9.3. Let $n = 4$ and consider the section cell

$$C := \{(w, x, y, z) \in \mathbb{R}^4 \mid w > 0, w^2 = 2, 0 < x < 1, y > 0, x+y < 1, z = 3x^2+2y^2\}.$$

This cell lies in the sub-cad of $\mathbb{R}^3$ above $\sqrt{2}$. `QEPCAD` output is presented below. Note that the sample points remain in extended representation, which is how they were generated in the lifting phase.

```
Before Refinement For Monotone Cells >
d-true-cells-t
---------- Information about the cell (6,3,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
```

```
  w > 0
  w^2 - 2 = 0
Level 2
  x > 0
  x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - 2 y^2 - 3 x^2 = 0

*** Initialising the RCAD. ***

Signs of projection factors (for defining formula) -

Index in RCAD: (6,3,3,2)
Level 1
  w > 0
  w^2 - 2 = 0
Level 2
  x > 0
  x - 1 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - 2 y^2 - 3 x^2 = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 1/2
             = 0.5000000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = the unique root of x - 7/8 between 0 and 2
             = the unique root of 8 x - 7 between 0 and 2
             = 0.8750000000
```

```
-----------------------------------------------------

Before Refinement For Monotone Cells >
g

Before Solution >
d-proj-fac
A_1,1  = input
       = w

A_1,2  = input
       = w^2 - 2


A_2,1  = input
       = x

A_2,2  = fac(J_2,1) = fac(res(A_3,1|A_3,2))
       = x - 1

M_2,3  = fac(K_2,3) = fac(input)
       = 5 x - 2


A_3,1  = input
       = y

A_3,2  = input
       = y + x - 1


A_4,1  = input
       = z - 2 y^2 - 3 x^2



Before Solution >
d-ref

M_2,1  = input  *** Refinement of cell (6,3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142+

Coordinate 1 = alpha
```

```
             = 1.4142+
Coordinate 2 = 2/5
             = 0.4000




Before Solution >
d-true-cells-t
---------- Information about the cell (6,5,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w^2 - 2 = 0
Level 2
  x > 0
  x - 1 < 0
  5 x - 2 > 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - 2 y^2 - 3 x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,5,3,2)
Level 1
  w > 0
  w^2 - 2 = 0
Level 2
  x > 0
  x - 1 < 0
  5 x - 2 > 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - 2 y^2 - 3 x^2 = 0

Sample point ---------------------------------------

The sample point is in an EXTENDED representation.
```

```
alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 1/2
             = 0.5000000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = the unique root of x - 7/8 between 0 and 2
             = the unique root of 8 x - 7 between 0 and 2
             = 0.8750000000


-----------------------------------------------------
---------- Information about the cell (6,4,3,2) (Dimension (0,0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  w > 0
  w^2 - 2 = 0
Level 2
  x > 0
  x - 1 < 0
  5 x - 2 = 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - 2 y^2 - 3 x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,4,3,2)
Level 1
  w > 0
  w^2 - 2 = 0
Level 2
  x > 0
  x - 1 < 0
  5 x - 2 = 0
Level 3
  y > 0
  y + x - 1 < 0
```

```
Level 4
  z - 2 y^2 - 3 x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 2/5
             = 0.4000000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 121/200
             = 0.6050000000


------------------------------------------------------
---------- Information about the cell (6,3,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w^2 - 2 = 0
Level 2
  x > 0
  x - 1 < 0
  5 x - 2 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - 2 y^2 - 3 x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,3,3,2)
Level 1
  w > 0
  w^2 - 2 = 0
Level 2
  x > 0
```

```
  x - 1 < 0
  5 x - 2 < 0
Level 3
  y > 0
  y + x - 1 < 0
Level 4
  z - 2 y^2 - 3 x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 1/5
             = 0.2000000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 49/200
             = 0.2450000000


--------------------------------------------------

Before Solution >
```

We now show a similar example, requiring an algebraic refinement.

Example 9.4. Let $n = 4$ and consider the section cell

$C := \{(w, x, y, z) \in \mathbb{R}^4 \mid w > 0, w^2 = 3, 0 < x < 1, y > 0, y < (x-1)^2, z = 3x^2+2y^2\}$.

Note that the sample points of the refined cells have been converted into primitive representation.

```
Before Refinement For Monotone Cells >
d-true-cells-t
---------- Information about the cell (6,3,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w^2 - 3 = 0
```

```
Level 2
  x > 0
  x - 1 < 0
Level 3
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 4
  z - y^2 - x^2 = 0

*** Initialising the RCAD. ***

Signs of projection factors (for defining formula) -

Index in RCAD: (6,3,3,2)
Level 1
  w > 0
  w^2 - 3 = 0
Level 2
  x > 0
  x - 1 < 0
Level 3
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 4
  z - y^2 - x^2 = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 3 between 1773/1024 and 887/512
      = 1.7320508076-

Coordinate 1 = alpha
             = 1.7320508076-
Coordinate 2 = 1/2
             = 0.5000000000
Coordinate 3 = 1/8
             = 0.1250000000
Coordinate 4 = the unique root of x - 17/64 between 0 and 1
             = the unique root of 64 x - 17 between 0 and 1
             = 0.2656250000


------------------------------------------------
```

```
Before Refinement For Monotone Cells >
g

Before Solution >
d-proj-fac
A_1,1  = input
       = w

A_1,2  = input
       = w^2 - 3


A_2,1  = input
       = x

A_2,2  = fac2(J_2,1) = fac2(res(A_3,1|A_3,2))
       = x - 1

M_2,3  = fac(K_2,3) = fac(input)
       = 2 x^3 - 6 x^2 + 7 x - 2

M_2,4  = fac(K_2,6) = fac(input)
       = 8 x - 9


A_3,1  = input
       = y

A_3,2  = input
       = y - x^2 + 2 x - 1


A_4,1  = input
       = z - y^2 - x^2



Before Solution >
d-ref

M_2,1  = input  *** Refinement of cell (6,3) ***
       = The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 3 between 1773/1024 and 887/512
      = 1.7321-
```

```
Coordinate 1 = alpha
             = 1.7321-
Coordinate 2 = the unique root of 2 x^3 - 6 x^2 + 7 x - 2 between 1/4 and 1/2
             = the unique root of 2 x^3 - 6 x^2 + 7 x - 2 between 1/4 and 1/2
             = 0.4102+
```

```
Before Solution >
d-true-cells-t
---------- Information about the cell (6,5,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w^2 - 3 = 0
Level 2
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 > 0
  8 x - 9 < 0
Level 3
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 4
  z - y^2 - x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,5,3,2)
Level 1
  w > 0
  w^2 - 3 = 0
Level 2
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 > 0
  8 x - 9 < 0
Level 3
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 4
  z - y^2 - x^2 = 0
```

```
Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 3 between 1773/1024 and 887/512
      = 1.7320508076-

Coordinate 1 = alpha
             = 1.7320508076-
Coordinate 2 = 1/2
             = 0.5000000000
Coordinate 3 = 1/8
             = 0.1250000000
Coordinate 4 = the unique root of x - 17/64 between 0 and 1
             = the unique root of 64 x - 17 between 0 and 1
             = 0.2656250000


------------------------------------------------------
---------- Information about the cell (6,4,3,2) (Dimension (0,0,1,0) (1))

Signs of projection factors ------------------------

Level 1
  w > 0
  w^2 - 3 = 0
Level 2
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 = 0
  8 x - 9 < 0
Level 3
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 4
  z - y^2 - x^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,4,3,2)
Level 1
  w > 0
  w^2 - 3 = 0
Level 2
  x > 0
```

```
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 = 0
  8 x - 9 < 0
Level 3
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 4
  z - y^2 - x^2 = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of 4 x^6 - 24 x^5 + 28 x^4 + 52 x^3 - 35 x^2 - 64 x - 107 between 17/8 ar
      = 2.1422962953-

Coordinate 1 = 300/8777 alpha^5 - 1518/8777 alpha^4 + 322/8777 alpha^3 + 5607/8777 alpha^2 + 6683
             = 1.7320508076-
Coordinate 2 = -300/8777 alpha^5 + 1518/8777 alpha^4 - 322/8777 alpha^3 - 5607/8777 alpha^2 + 209
             = 0.4102454877-
Coordinate 3 = 1/8
             = 0.1250000000
Coordinate 4 = the unique root of 262144 x^3 - 536576 x^2 + 1654976 x - 287873 between 0 and 4
             = the unique root of 262144 x^3 - 536576 x^2 + 1654976 x - 287873 between 0 and 4
             = 0.1839263602-


--------------------------------------------------
---------- Information about the cell (6,3,3,2) (Dimension (0,1,1,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w^2 - 3 = 0
Level 2
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 < 0
  8 x - 9 < 0
Level 3
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 4
  z - y^2 - x^2 = 0
```

```
Signs of projection factors (for defining formula) -

Index in RCAD: (6,3,3,2)
Level 1
  w > 0
  w^2 - 3 = 0
Level 2
  x > 0
  x - 1 < 0
  2 x^3 - 6 x^2 + 7 x - 2 < 0
  8 x - 9 < 0
Level 3
  y > 0
  y - x^2 + 2 x - 1 < 0
Level 4
  z - y^2 - x^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 3 between 1773/1024 and 887/512
      = 1.7320508076-

Coordinate 1 = alpha
             = 1.7320508076-
Coordinate 2 = 3/16
             = 0.1875000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 25/256
             = 0.0976562500


--------------------------------------------------

Before Solution >
```

Now consider a $(1,1,0,0)$-cell in $\mathbb{R}^4$ requiring two refinements.

Example 9.5. Let $n = 4$ and consider the section cell

$C := \{(w,x,y,z) \in \mathbb{R}^4 \mid 0 < w < 1, x > 0, w+x < 1, y = w^2+x^2, z = 2w^2+x^2\}$.

```
Before Refinement For Monotone Cells >
d-true-cells-t
---------- Information about the cell (3,3,2,2) (Dimension (1,1,0,0) (2))
```

```
Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

*** Initialising the RCAD. ***

Signs of projection factors (for defining formula) -

Index in RCAD: (3,3,2,2)
Level 1
  w > 0
  w - 1 < 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Sample point ---------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 1/8
             = 0.1250000000
Coordinate 3 = 17/64
             = 0.2656250000
Coordinate 4 = 33/64
             = 0.5156250000
```

```
--------------------------------------------------

Before Refinement For Monotone Cells >
g

Before Solution >
d-proj-fac
A_1,1  = input
       = w

A_1,2  = fac(J_1,1) = fac(res(A_2,1|A_2,2))
       = w - 1

M_1,3  = fac(K_1,4) = fac(input)
       = 3 w - 1

M_1,4  = fac(K_1,5) = fac(input)
       = 2 w - 1


A_2,1  = input
       = x

A_2,2  = input
       = x + w - 1


A_3,1  = input
       = y - x^2 - w^2


A_4,1  = input
       = z - x^2 - 2 w^2



Before Solution >
d-ref
M_1,2  = input  *** Refinement of cell (3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000

Coordinate 1 = 1/2
```

```
             = 0.5000


M_1,1  = input  *** Refinement of cell (3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000

Coordinate 1 = 1/3
             = 0.3333+




Before Solution >
d-true-cells-t
---------- Information about the cell (7,3,2,2) (Dimension (1,1,0,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
  3 w - 1 > 0
  2 w - 1 > 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (7,3,2,2)
Level 1
  w > 0
  w - 1 < 0
  3 w - 1 > 0
  2 w - 1 > 0
Level 2
  x > 0
```

```
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 3/4
             = 0.7500000000
Coordinate 2 = 1/16
             = 0.0625000000
Coordinate 3 = 145/256
             = 0.5664062500
Coordinate 4 = 289/256
             = 1.1289062500


------------------------------------------------------
---------- Information about the cell (6,3,2,2) (Dimension (0,1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
  3 w - 1 > 0
  2 w - 1 = 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,3,2,2)
Level 1
  w > 0
```

```
  w - 1 < 0
  3 w - 1 > 0
  2 w - 1 = 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 1/8
             = 0.1250000000
Coordinate 3 = 17/64
             = 0.2656250000
Coordinate 4 = 33/64
             = 0.5156250000


------------------------------------------------------
---------- Information about the cell (5,3,2,2) (Dimension (1,1,0,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
  3 w - 1 > 0
  2 w - 1 < 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0
```

```
Signs of projection factors (for defining formula) -

Index in RCAD: (5,3,2,2)
Level 1
  w > 0
  w - 1 < 0
  3 w - 1 > 0
  2 w - 1 < 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 5/12
             = 0.4166666667-
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = 17/72
             = 0.2361111111+
Coordinate 4 = 59/144
             = 0.4097222222+


------------------------------------------------------
---------- Information about the cell (4,3,2,2) (Dimension (0,1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
  3 w - 1 = 0
  2 w - 1 < 0
Level 2
  x > 0
  x + w - 1 < 0
```

```
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,3,2,2)
Level 1
  w > 0
  w - 1 < 0
  3 w - 1 = 0
  2 w - 1 < 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/3
             = 0.3333333333+
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = 25/144
             = 0.1736111111+
Coordinate 4 = 41/144
             = 0.2847222222+


--------------------------------------------------
---------- Information about the cell (3,3,2,2) (Dimension (1,1,0,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
```

```
  3 w - 1 < 0
  2 w - 1 < 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,3,2,2)
Level 1
  w > 0
  w - 1 < 0
  3 w - 1 < 0
  2 w - 1 < 0
Level 2
  x > 0
  x + w - 1 < 0
Level 3
  y - x^2 - w^2 = 0
Level 4
  z - x^2 - 2 w^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/6
             = 0.1666666667-
Coordinate 2 = 1/4
             = 0.2500000000
Coordinate 3 = 13/144
             = 0.0902777778-
Coordinate 4 = 17/144
             = 0.1180555556-


--------------------------------------------------

Before Solution >
```

The following example, in $\mathbb{R}^5$ presents a $(1,0,1,0,0)$-cell, which requires two algebraic refinements of the CAD induced on $\mathbb{R}^1$. When running this test, `SACLIB` ran out of memory, so the number of available cells was doubled.

Example 9.6. Let $n = 5$ and consider the $(1,0,1,0,0)$-cell

$C = \{(w,x,y,z) \in \mathbb{R}^4 \mid 0 < w < 1, x = 1-w, y > 0, w+y < 1, z = w^2+xy^2, t = 3w^2+x^2y^2\}$.

```
Before Solution >
d-proj-fac
A_1,1  = input
       = w

A_1,2  = fac(J_2,1) = fac(res(A_3,1|A_3,2))
       = w - 1

M_1,3  = fac(K_1,1) = fac(input)
       = 2 w^3 - 6 w^2 + 9 w - 2

M_1,4  = fac(K_1,2) = fac(input)
       = w + 6

M_1,5  = fac(K_1,4) = fac(input)
       = 3176 w - 4353

M_1,6  = fac(K_1,5) = fac(input)
       = 3 w^2 - 8 w + 3


A_2,1  = input
       = x + w - 1

Q_2,2  = fac(Q_3,1) = fac(input)
       = x

Q_2,3  = fac3(Q_2,2) = fac3(input)
       = x - 3


A_3,1  = input
       = y

A_3,2  = input
       = y + w - 1

Q_3,3  = fac(Q_3,3) = fac(input)
```

```
       = y^2 - 2 w

Q_3,4  = fac(Q_3,4) = fac(input)
       = x y^2 - 3 w

Q_3,5  = fac(Q_3,5) = fac(input)
       = x y^2 + 2 w x - 6 w


A_4,1  = input
       = z - x y^2 - w^2


A_5,1  = input
       = t - x^2 y^2 - 3 w^2



Before Solution >
d-ref
M_1,2  = input  *** Refinement of cell (3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of 3 x^2 - 8 x + 3 between 3/8 and 1/2
      = 0.4514+

Coordinate 1 = alpha
             = 0.4514+


M_1,1  = input  *** Refinement of cell (3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of 2 x^3 - 6 x^2 + 9 x - 2 between 1/4 and 5/16
      = 0.2649-

Coordinate 1 = alpha
             = 0.2649-







Before Solution >
```

```
d-true-cells-t
---------- Information about the cell (7,4,9,2,2) (Dimension (1,0,1,0,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 > 0
  w + 6 > 0
  3176 w - 4353 < 0
  3 w^2 - 8 w + 3 < 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (7,4,9,2,2)
Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 > 0
  w + 6 > 0
  3176 w - 4353 < 0
  3 w^2 - 8 w + 3 < 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
```

```
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/2
             = 0.5000000000
Coordinate 2 = 1/2
             = 0.5000000000
Coordinate 3 = 1/8
             = 0.1250000000
Coordinate 4 = 33/128
             = 0.2578125000
Coordinate 5 = 193/256
             = 0.7539062500


------------------------------------------------------
---------- Information about the cell (6,4,9,2,2) (Dimension (0,0,1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 > 0
  w + 6 > 0
  3176 w - 4353 > 0
  3 w^2 - 8 w + 3 = 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
```

```
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (6,4,9,2,2)
Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 > 0
  w + 6 > 0
  3176 w - 4353 > 0
  3 w^2 - 8 w + 3 = 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of 3 x^2 - 8 x + 3 between 7/16 and 15/32
      = 0.4514162296+

Coordinate 1 = alpha
             = 0.4514162296+
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 1/4
             = 0.2500000000
Coordinate 4 = 8/3 alpha - 1
             = 0.2037766124-
```

```
Coordinate 5 = the unique root of 3 x^2 - 46 x + 27 between 0 and 4
             = the unique root of 3 x^2 - 46 x + 27 between 0 and 4
             = 0.6113298372-


-----------------------------------------------------
---------- Information about the cell (5,4,9,2,2) (Dimension (1,0,1,0,0) (2))

Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 > 0
  w + 6 > 0
  3176 w - 4353 < 0
  3 w^2 - 8 w + 3 < 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (5,4,9,2,2)
Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 > 0
  w + 6 > 0
  3176 w - 4353 < 0
  3 w^2 - 8 w + 3 < 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
```

```
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 39/32
             = 1.2187500000
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 3
             = 3.0000000000
Coordinate 4 = 1521/1024
             = 1.4853515625
Coordinate 5 = 4563/1024
             = 4.4560546875


--------------------------------------------------
---------- Information about the cell (4,4,9,2,2) (Dimension (0,0,1,0,0) (1))

Signs of projection factors ------------------------

Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 = 0
  w + 6 > 0
  3176 w - 4353 > 0
  3 w^2 - 8 w + 3 > 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
```

```
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (4,4,9,2,2)
Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 = 0
  w + 6 > 0
  3176 w - 4353 > 0
  3 w^2 - 8 w + 3 > 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of 2 x^3 - 6 x^2 + 9 x - 2 between 1/4 and 5/16
      = 0.2648607410-

Coordinate 1 = alpha
             = 0.2648607410-
Coordinate 2 = -alpha + 1
```

```
             = 0.7351392590+
Coordinate 3 = 93/256
             = 0.3632812500
Coordinate 4 = alpha^2 - 8649/65536 alpha + 8649/65536
             = 0.1671699415+
Coordinate 5 = the unique root of 1125899906842624 x^3 + 445766065717248 x^2 + 147784030448320512
             = the unique root of 1125899906842624 x^3 + 445766065717248 x^2 + 147784030448320512
             = 0.2817759132-


-----------------------------------------------------
---------- Information about the cell (3,4,9,2,2) (Dimension (1,0,1,0,0) (2))

Signs of projection factors -------------------------

Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 < 0
  w + 6 > 0
  3176 w - 4353 < 0
  3 w^2 - 8 w + 3 > 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (3,4,9,2,2)
Level 1
  w > 0
  w - 1 < 0
  2 w^3 - 6 w^2 + 9 w - 2 < 0
  w + 6 > 0
  3176 w - 4353 < 0
```

```
  3 w^2 - 8 w + 3 > 0
Level 2
  x + w - 1 = 0
  x > 0
  x - 3 < 0
Level 3
  y > 0
  y + w - 1 < 0
  y^2 - 2 w < 0
  x y^2 - 3 w < 0
  x y^2 + 2 w x - 6 w < 0
Level 4
  z - x y^2 - w^2 = 0
Level 5
  t - x^2 y^2 - 3 w^2 = 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000000000

Coordinate 1 = 1/8
             = 0.1250000000
Coordinate 2 = 7/8
             = 0.8750000000
Coordinate 3 = 1/8
             = 0.1250000000
Coordinate 4 = 15/512
             = 0.0292968750
Coordinate 5 = 241/4096
             = 0.0588378906+


--------------------------------------------------

Before Solution >
```

Example 9.7. Let $n = 4$ and consider the semialgebraic set defined by the QFF, containing polynomials in $Z[t, x, y, z]$,

$$F = \{(t = 0 \vee t = 2) \wedge 0 < x < 1 \wedge -x < y < x \wedge y^2 - x^2(z - t) = 0\}.$$

The QFF defines two cylindrical cells in $\mathbb{R}^3$ (equipped with (x, y, z)), similar to those defined in Example 6.1. The following refinement points are obtained.

```
M_4,2  = input  *** Refinement of cell (6,2,2,5) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000

Coordinate 1 = 2
             = 2.0000
Coordinate 2 = 0
             = 0.0000
Coordinate 3 = 0
             = 0.0000
Coordinate 4 = 3
             = 3.0000


M_4,1  = input  *** Refinement of cell (4,2,2,3) ***
       = The sample point is in a PRIMITIVE representation.

alpha = the unique root of x between 0 and 0
      = 0.0000

Coordinate 1 = 0
             = 0.0000
Coordinate 2 = 0
             = 0.0000
Coordinate 3 = 0
             = 0.0000
Coordinate 4 = 1
             = 1.0000
```

Example 9.8. Let $n = 4$ and consider the twe 2-diemnsional subsets of the Whitney whitney umbrella defined by the QFF

$$F = \{(t, x, y, z) \in \mathbb{R}^4 \mid t^2 = 0 \wedge 0 < x < 1 \wedge -x < y < x \wedge y^2 - x^2(z - t) = 0\}.$$

THe following algebraic refinement points are produced.

```
M_4,3  = input  *** Refinement of cell (8,2,2,1) ***
       = The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142+

Coordinate 1 = alpha
             = 1.4142+
```

```
Coordinate 2 = 0
             = 0.0000
Coordinate 3 = 0
             = 0.0000
Coordinate 4 = the unique root of x^2 - 2 x - 1 between -1/2 and -1/4
             = the unique root of x^2 - 2 x - 1 between -1/2 and -1/4
             = -0.4142+


M_4,2  = input  *** Refinement of cell (2,2,2,3) ***
       = The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142+

Coordinate 1 = alpha
             = -1.4142+
Coordinate 2 = 0
             = 0.0000
Coordinate 3 = 0
             = 0.0000
Coordinate 4 = the unique root of x^2 - 2 x - 1 between 0 and 4
             = the unique root of x^2 - 2 x - 1 between 0 and 4
             = 2.4142+


M_4,1  = input  *** Refinement of cell (2,2,2,3) ***
       = The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142+

Coordinate 1 = alpha
             = -1.4142+
Coordinate 2 = 0
             = 0.0000
Coordinate 3 = 0
             = 0.0000
Coordinate 4 = the unique root of x^2 - 2 x - 1 between -1/2 and -1/4
             = the unique root of x^2 - 2 x - 1 between -1/2 and -1/4
             = -0.4142+
```

This results in the following cells above, respectively, $2, 2, 2)$ and $(8, 2, 2)$.

```
d-cell(2,2,2)---------- Information about the cell (2,2,2) ----------

Level                       : 3
```

```
Dimension                   : 0
Number of children          : 7
Truth value                 : F    by trial evaluation.
Degrees after substitution  : (-1,1)
Multiplicities              : ((1,1),(2,1),(3,2),(4,1))
Signs of Projection Factors
Level 1  : (0,-,-)
Level 2  : (0,-)
Level 3  : (0,0,0,0)
----------   Sample point  ----------
The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142135624-

Coordinate 1 = alpha
             = -1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000


--------------------------------------------------

Before Solution >
d-cell-t(2,2,2,1)---------- Information about the cell (2,2,2,1) (Dimension (0,0,0,1) (1))

Signs of projection factors ------------------------

Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t < 0
  z^2 - 2 z - 1 > 0
```

```
Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,2,1)
Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t < 0
  z^2 - 2 z - 1 > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142135624-

Coordinate 1 = alpha
             = -1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = -5
             = -5.0000000000


--------------------------------------------------

Before Solution >
d-cell-t(2,2,2,2)---------- Information about the cell (2,2,2,2) (Dimension (0,0,0,0)

Signs of projection factors ------------------------

Level 1
```

```
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t = 0
  z^2 - 2 z - 1 < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,2,2)
Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t = 0
  z^2 - 2 z - 1 < 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142135624-

Coordinate 1 = alpha
             = -1.4142135624-
Coordinate 2 = 0
```

```
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = the unique root of x - alpha between -4 and 0
             = the unique root of x^2 - 2 between -4 and 0
             = -1.4142135624-


--------------------------------------------------

Before Solution >
d-cell-t(2,2,2,3)---------- Information about the cell (2,2,2,3) (Dimension (0,0,0,1)

Signs of projection factors ------------------------

Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,2,3)
Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
```

```
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142135624-

Coordinate 1 = alpha
             = -1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = -19/16
             = -1.1875000000


--------------------------------------------------

Before Solution >
d-cell-t(2,2,2,4)---------- Information about the cell (2,2,2,4) (Dimension (0,0,0,0) (0))

Signs of projection factors ------------------------

Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 = 0
```

```
Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,2,4)
Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142135624-

Coordinate 1 = alpha
             = -1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = the unique root of x^2 - 2 x - 1 between -1/2 and -1/4
             = the unique root of x^2 - 2 x - 1 between -1/2 and -1/4
             = -0.4142135624-


--------------------------------------------------

Before Solution >
d-cell-t(2,2,2,5)---------- Information about the cell (2,2,2,5) (Dimension (0,0,0,1)

Signs of projection factors ------------------------
```

```
Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,2,5)
Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 < 0

Sample point ---------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142135624-

Coordinate 1 = alpha
             = -1.4142135624-
```

```
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = 1
             = 1.0000000000


--------------------------------------------------

Before Solution >
d-cell-t(2,2,2,6)---------- Information about the cell (2,2,2,6) (Dimension (0,0,0,0)

Signs of projection factors ------------------------

Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,2,6)
Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
```

```
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142135624-

Coordinate 1 = alpha
             = -1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = the unique root of x^2 - 2 x - 1 between 2 and 4
             = the unique root of x^2 - 2 x - 1 between 2 and 4
             = 2.4142135624-


--------------------------------------------------

Before Solution >
d-cell-t(2,2,2,7)---------- Information about the cell (2,2,2,7) (Dimension (0,0,0,1) (1))

Signs of projection factors ------------------------

Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
```

```
  z^2 - 2 z - 1 > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (2,2,2,7)
Level 1
  t^2 - 2 = 0
  t < 0
  t + 1 < 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between -1449/1024 and -181/128
      = -1.4142135624-

Coordinate 1 = alpha
             = -1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = 5
             = 5.0000000000


--------------------------------------------------

Before Solution >
d-cell(8,2,2)---------- Information about the cell (8,2,2) ----------

Level                       : 3
Dimension                   : 0
```

```
Number of children          : 5
Truth value                 : F    by trial evaluation.
Degrees after substitution  : (-1,1)
Multiplicities              : ((1,1),(2,1),(3,2),(4,1))
Signs of Projection Factors
Level 1  : (0,+,+)
Level 2  : (0,-)
Level 3  : (0,0,0,0)
----------   Sample point  ----------
The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000


--------------------------------------------------

Before Solution >
d-cell-t(8,2,2,1)---------- Information about the cell (8,2,2,1) (Dimension (0,0,0,1) (1))

Signs of projection factors -------------------------

Level 1
  t^2 - 2 = 0
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t < 0
  z^2 - 2 z - 1 > 0
```

```
Signs of projection factors (for defining formula) -

Index in RCAD: (8,2,2,1)
Level 1
  t^2 - 2 = 0
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t < 0
  z^2 - 2 z - 1 > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = -1
             = -1.0000000000


--------------------------------------------------

Before Solution >
d-cell-t(8,2,2,2)---------- Information about the cell (8,2,2,2) (Dimension (0,0,0,0)

Signs of projection factors ------------------------

Level 1
  t^2 - 2 = 0
```

```
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t < 0
  z^2 - 2 z - 1 = 0

Signs of projection factors (for defining formula) -

Index in RCAD: (8,2,2,2)
Level 1
  t^2 - 2 = 0
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t < 0
  z^2 - 2 z - 1 = 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
```

```
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = the unique root of x^2 - 2 x - 1 between -1/2 and -1/4
             = the unique root of x^2 - 2 x - 1 between -1/2 and -1/4
             = -0.4142135624-


--------------------------------------------------

Before Solution >
d-cell-t(8,2,2,3)---------- Information about the cell (8,2,2,3) (Dimension (0,0,0,1)

Signs of projection factors ------------------------

Level 1
  t^2 - 2 = 0
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t < 0
  z^2 - 2 z - 1 < 0

Signs of projection factors (for defining formula) -

Index in RCAD: (8,2,2,3)
Level 1
  t^2 - 2 = 0
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
```

```
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t < 0
  z^2 - 2 z - 1 < 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = -1/8
             = -0.1250000000


--------------------------------------------------

Before Solution >
d-cell-t(8,2,2,4)---------- Information about the cell (8,2,2,4) (Dimension (0,0,0,0) (0))

Signs of projection factors -------------------------

Level 1
  t^2 - 2 = 0
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t = 0
  z^2 - 2 z - 1 < 0
```

```
Signs of projection factors (for defining formula) -

Index in RCAD: (8,2,2,4)
Level 1
  t^2 - 2 = 0
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t = 0
  z^2 - 2 z - 1 < 0

Sample point ----------------------------------------

The sample point is in an EXTENDED representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 0
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = the unique root of x - alpha between 0 and 4
             = the unique root of x^2 - 2 between 0 and 4
             = 1.4142135624-


--------------------------------------------------

Before Solution >
d-cell-t(8,2,2,5)---------- Information about the cell (8,2,2,5) (Dimension (0,0,0,1)

Signs of projection factors ------------------------

Level 1
```

```
  t^2 - 2 = 0
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 > 0

Signs of projection factors (for defining formula) -

Index in RCAD: (8,2,2,5)
Level 1
  t^2 - 2 = 0
  t > 0
  t + 1 > 0
Level 2
  x = 0
  x - 1 < 0
Level 3
  y + x = 0
  y - x = 0
  y^2 + t x^2 = 0
  y = 0
Level 4
  x^2 z - y^2 - t x^2 = 0
  z - t > 0
  z^2 - 2 z - 1 > 0

Sample point ----------------------------------------

The sample point is in a PRIMITIVE representation.

alpha = the unique root of x^2 - 2 between 181/128 and 1449/1024
      = 1.4142135624-

Coordinate 1 = alpha
             = 1.4142135624-
Coordinate 2 = 0
```

```
             = 0.0000000000
Coordinate 3 = 0
             = 0.0000000000
Coordinate 4 = 5
             = 5.0000000000
```

More examples

- large sub-CAD, maybe R^5
- cel ls in sub-CAD of different levels
- some variations on the whitney umbrella
  - algebraic refinement point?
  - multiple levels refinement (stacked whitneys)
- whitney in a sub-CAD
- whitney with multiple refinement points above it

3D multi: multi-3.txt. many unnecesary refinements computed resulting from quasi-affine. had to use +N5000000

4D multi: it's similar, but requiring algebraic refinements. this required +N50000000 for saclib and look at how many cells it produced! we should probably associate the quasi polynomials with the true cell so that we don't make a truly tsign-invariant CAD

Bibliography

(2024). saclib: Computations with real algebraic numbers. https://doc.sagemath.org/html/en/reference/spkg/saclib.html. Accessed: 2024-08-09.

Arnon, D. (1979). A cellular decomposition algorithm for semialgebraic sets. In Ng, E. W., editor, *Symbolic and Algebraic Computation*, pages 301–315, Berlin, Heidelberg. Springer Berlin Heidelberg.

Arnon, D. S., Collins, G. E., and McCallum, S. (1984). Cylindrical algebraic decomposition ii: An adjacency algorithm for the plane. In Caviness, B. F. and Johnson, J. R., editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 152–165, Vienna. Springer Vienna.

Arnon, D. S., Collins, G. E., and McCallum, S. (1987). An adjacency algorithm for cylindrical algebraic decompositions of three-dimensional space. In Caviness, B. F., editor, *EUROCAL '85*, pages 246–261, Berlin, Heidelberg. Springer Berlin Heidelberg.

Basu, S., Gabrielov, A., and Vorobjov, N. (2013). Monotone functions and maps. *Revista de la Real Academia de Ciencias Exactas, Fisicas y Naturales. Serie A. Matematicas*, 107(1):5–33.

Basu, S., Gabrielov, A., and Vorobjov, N. (2015). Triangulations of monotone families i: two-dimensional families. *Proceedings of the London Mathematical Society*, 111(5):1013–1051.

Basu, S., Gabrielov, A., and Vorobjov, N. (preprint). Triangulation of monotone families ii: families in r3.

Basu, S., Pollack, R., and Roy, M.-F. (1998). A new algorithm to find a point in every cell defined by a family of polynomials. In Caviness, B. F. and Johnson, J. R., editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 341–350, Vienna. Springer Vienna.

Basu, S., Pollack, R., and Roy, M.-F. (2006). *Algorithms in Real Algebraic Geometry*. Algorithms and Computation in Mathematics, 10. Springer Berlin Heidelberg : Imprint: Springer, Berlin, Heidelberg, 2nd ed. 2006. edition.

Blum, L., Shub, M., and Smale, S. (1998). *Complexity and Real Computation*. Springer New York : Imprint: Springer, New York, NY.

Bradford, R., Chen, C., Davenport, J. H., England, M., Moreno Maza, M., and Wilson, D. (2014). Truth table invariant cylindrical algebraic decomposition by regular chains. In *Computer Algebra in Scientific Computing: 16th International Workshop, CASC 2014, Warsaw, Poland, September 8-12, 2014. Proceedings 16*, pages 44–58. Springer.

Brown, C. and Hong, H. Q. (2004). Quantifier elimination by partial cylindrical algebraic decomposition.

Brown, C. W. (1999). Guaranteed solution formula construction. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, ISSAC '99, page 137–144, New York, NY, USA. Association for Computing Machinery.

Brown, C. W. (2001). Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465.

Brown, C. W. (2003). Qepcad b: A program for computing with semi-algebraic sets using cads. *SIGSAM Bull.*, 37(4):97–108.

Brown, C. W. and Davenport, J. H. (2007). The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proceedings of the 2007 international symposium on Symbolic and algebraic computation*, pages 54–60.

Buchberger, B. (1979). A criterion for detecting unnecessary reductions in the construction of gröbner-bases. In *International Symposium on Symbolic and Algebraic Manipulation*, pages 3–21. Springer.

Chen, C. and Moreno Maza, M. (2014). An incremental algorithm for computing cylindrical algebraic decompositions. In *Computer Mathematics*, pages 199–221. Springer.

Chen, C., Zhu, Z., and Chi, H. (2020). Variable ordering selection for cylindrical algebraic decomposition with artificial neural networks. In Bigatti, A. M., Carette, J., Davenport, J. H., Joswig, M., and de Wolff, T., editors, *Mathematical Software – ICMS 2020*, pages 281–291, Cham. Springer International Publishing.

Collins, G. E. (1975). Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata theory and formal languages*, pages 134–183. Springer.

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

Coste, M. (2000). An introduction to semialgebraic geometry.

Coste, M. and Roy, M. (1988). Thom's lemma, the coding of real algebraic numbers and the computation of the topology of semi-algebraic sets. *Journal of Symbolic Computation*, 5(1):121–129.

Cox, D., Little, J., and OShea, D. (2013). *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*. Springer Science & Business Media.

Davenport, J. H. and Heintz, J. (1988). Real quantifier elimination is doubly exponential. *Journal of symbolic computation*, 5(1):29–35.

Davenport, J. H., Locatelli, A., and Sankaran, G. (2020). Regular cylindrical algebraic decomposition. *Journal of the London Mathematical Society*, 101(1):43–59.

del Río, T. and England, M. (2022). New heuristic to choose a cylindrical algebraic decomposition variable ordering motivated by complexity analysis. In Boulier, F., England, M., Sadykov, T. M., and Vorozhtsov, E. V., editors, *Computer Algebra in Scientific Computing*, pages 300–317, Cham. Springer International Publishing.

Gabrielov, A. (1996). Complements of subanalytic sets and existential formulas for analytic functions. *Inventiones mathematicae*, 125(1):1–12.

Gabrielov, A. and Vorobjov, N. (1995). Complexity of stratification of semi-pfaffian sets. *Discrete & computational geometry*, 14(1):71–91.

Gabrielov, A. and Vorobjov, N. (2001). Complexity of cylindrical decompositions of sub-pfaffian sets. *Journal of Pure and Applied Algebra*, 164(1-2):179–197.

Gabrielov, A. and Vorobjov, N. (2004). Complexity of computations with pfaffian and noetherian functions. *Normal forms, bifurcations and finiteness problems in differential equations*, 137:211–250.

Gabrielov, A. and Vorobjov, N. (2009). Approximation of definable sets by compact families, and upper bounds on homotopy and homology. *Journal of the London Mathematical Society*, 80(1):35–54.

Hong, H. (1990). An improvement of the projection operator in cylindrical algebraic decomposition. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pages 261–264.

Hong, H. (1993). Heuristic search strategies for cylindrical algebraic decomposition. In Calmet, J. and Campbell, J. A., editors, *Artificial Intelligence and Symbolic Mathematical Computing*, Berlin, Heidelberg. Springer Berlin Heidelberg.

Huang, Z., England, M., Wilson, D., Davenport, J. H., Paulson, L. C., and Bridge, J. (2014). Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In Watt, S. M., Davenport, J. H., Sexton, A. P., Sojka, P., and Urban, J.,

editors, *Intelligent Computer Mathematics*, pages 92–107, Cham. Springer International Publishing.

Kleene, S. C. (1952). *Introduction to Metamathematics*. P. Noordhoff N.V., Groningen.

Koiran, P. (1999). The real dimension problem is npr-complete. *Journal of Complexity*, 15(2):227–238.

Kremer, G. and Ábrahám, E. (2020). Fully incremental cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 100:11–37. Symbolic Computation and Satisfiability Checking.

Lakshman, Y. N. (1991). *A Single Exponential Bound on the Complexity of Computing Gröbner Bases of Zero Dimensional Ideals*, pages 227–234. Birkhäuser Boston, Boston, MA.

Lazard, D. (1979). Systems of algebraic equations. In *Symbolic and Algebraic Computation: EUROSM'79, An International Symposium on Symbolic and Algebraic Manipulation, Marseille, France, June 1979 2*, pages 88–94. Springer.

Lazard, D. (2010). Cad and topology of semi-algebraic sets. *Mathematics in Computer Science*, 4(1):93–112.

Lojasiewicz, S. (1964). Triangulation of semi-analytic sets. *Annali della Scuola Normale Superiore di Pisa - Scienze Fisiche e Matematiche*, Ser. 3, 18(4):449–474.

Masser, D. W. and Wüstholz, G. (1983). Fields of large transcendence degree generated by values of elliptic functions. *Inventiones mathematicae*, 72:407–464.

Mayr, E. W. (1997). Some complexity results for polynomial ideals. *Journal of Complexity*, 13(3):303–325.

McCallum, S. (1988). An improved projection operation for cylindrical algebraic decomposition of three-dimensional space. *Journal of Symbolic Computation*, 5(1):141–161.

McCallum, S. (1993). Solving Polynomial Strict Inequalities Using Cylindrical Algebraic Decomposition. *The Computer Journal*, 36(5):432–438.

McCallum, S. (1998). An improved projection operation for cylindrical algebraic decomposition. In Caviness, B. F. and Johnson, J. R., editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 242–268, Vienna. Springer Vienna.

McCallum, S., Parusiński, A., and Paunescu, L. (2019). Validity proof of lazard's method for cad construction. *Journal of Symbolic Computation*, 92:52–69.

Reif, J. H. (1979). Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427. IEEE Computer Society.

Safey El Din, M. (2013). Critical point methods and effective real algebraic geometry: new results and trends. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, pages 5–6.

Schwartz, J. T. and Sharir, M. (1983). On the "piano movers" problem. ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in applied Mathematics*, 4(3):298–350.

Seidenberg, A. (1954). A new decision method for elementary algebra. *Annals of Mathematics*, pages 365–374.

Sierpiński, W. and Tarski, A. (1930). Sur une propriété caractéristique des nombres inaccessibles. *Fundamenta Mathematicae*, 1(15):292–300.

Tarski, A. (1998). A decision method for elementary algebra and geometry. In *Quantifier elimination and cylindrical algebraic decomposition*, pages 24–84. Springer.

Van den Dries, L. (1998). *Tame topology and o-minimal structures*, volume 248. Cambridge university press.

Van den Dries, L. et al. (1988). Alfred tarski's elimination theory for real closed fields. *The journal of symbolic logic.*, 53(1):7–19.

Whitney, H. (1992). Elementary structure of real algebraic varieties. *Hassler Whitney Collected Papers*, pages 456–467.

Wüthrich, H. R. (1976). Ein entscheidungsverfahren für die theorie der reell-abgeschlossenen körper. In *Komplexität von Entscheidungsproblemen Ein Seminar*, Lecture Notes in Computer Science 43, 138162, pages 138–162, Berlin, Heidelberg. Springer.