

# Cylindrical Algebraic Decompositions With Monotone Cells

Hollie

2024-07-25



# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Background</b>	<b>15</b>
2.1	Semialgebraic sets and monotone cells . . . . .	15
2.2	Cylindrical algebraic decomposition . . . . .	21
<b>3</b>	<b>Smooth Stratification</b>	<b>35</b>
3.1	Implementation Details . . . . .	47
<b>4</b>	<b>Quasi-affine cells</b>	<b>63</b>
4.1	Computing the smooth two-dimensional locus of $V$ . . . . .	64
4.2	Ensuring that every cell is the graph of a quasi-affine map . . . .	65
4.3	Correctness and Complexity . . . . .	68
<b>5</b>	<b>Monotone Cells</b>	<b>71</b>
5.1	Two-dimensional monotone sector cells . . . . .	73
5.2	Two-dimensional monotone sections . . . . .	81
5.3	Implementation Details . . . . .	86
5.4	Correctness and Complexity . . . . .	88
5.5	Computing the refinements . . . . .	89
<b>6</b>	<b>Frontier Condition</b>	<b>91</b>
6.1	Construction of a CAD with frontier condition, due to ?, Theorem 3.20 . . . . .	92
6.2	Lazard’s method for dimension not greater than 3 . . . . .	94
6.3	Generalisation of Lazard . . . . .	102
<b>7</b>	<b>A novel algorithm for the obtaining the frontier condition on a cylindrical decomposition of <math>\mathbb{R}^n</math></b>	<b>111</b>
7.1	Statement of the Problem . . . . .	113
7.2	Theory . . . . .	113
7.3	Main Result . . . . .	116
7.4	Generalisations and further work . . . . .	125

<b>8 Implementation And Testing</b>	<b>127</b>
8.1 Implementation on top of QEPCAD . . . . .	127
8.2 Test Cases . . . . .	158

# 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 or 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). ? 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 solutions of a system of multivariate polynomial equations. ? also presented an algorithm, based on Macaulay'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. ? 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 (?), 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 [1]. He then proved the theorem in [2] whose planned publication in 1939 was delayed due to the war. [1] 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. [1] 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 [3]. 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 [4]). 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. [5] employs the critical point method to the emptiness problem, obtaining a bound singly exponential in the number of variables and [6] proved that determining the dimension of a semialgebraic set is  $\text{NP}_{\mathbb{R}}$ -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 [7] 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. [8] 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, ? 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 (?). 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. ? 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 ?, 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, ? employed support vector machines and ? 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, ? 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. ? 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 ?) Lazard’s projection (see ?) have been proposed, each offering different properties. Another optimisation is to construct a “partial CAD”, proposed by ?. 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 on 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., ? 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, [?] 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 [?] 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, [?] 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, [?] 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 (?). 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. [?] 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 2.21). For example, consider the Whitney umbrella, the set of points in  $\mathbb{R}^3$  which satisfy the equation  $f = 0$  where

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

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,  $f$  “blow- 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

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

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. [?] presents an algorithm which produces such a decomposition, in the potential presence of these blow-up points, for dimension  $\leq 3$ . 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. ? 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. ? 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, ? 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 (7.3), we write a basic semialgebraic set of the form

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

as

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

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

? present a constructive proof that, given a family  $V := (V_1, \dots, V_k)$  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 ? 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. ? or ?, 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 ? is discussed by ? (Section 2.19), who introduces the concept of a Van-den-Dries regular cell.

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

All two-dimensional Van-den-Dries regular cells are topologically regular, so if  $\dim(V_1, \dots, V_k)$ , the decomposition described by ? will consist of topologically regular cells. The result of ? 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 ? will be generalised to sets of arbitrary dimension. In particular, a novel algorithm for constructing a cylindrical decomposition, compatible with a semialgebraic set and such 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., (?, §57)).* % TODO define elementary

*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 ? 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 ?, 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. ? 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. (?) 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~7.4.1), 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~??, 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.

? 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.(?) ? 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 ? 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?

### 1.0.1 Structure of this thesis

Chapter 2 gives some necessary definitions, then presents some useful results from ?. The CAD is then formally defined, and the flow of the algorithm, following the description given by ? is outlined. Chapter 3 presents the smooth stratification algorithm from ?, 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.

Chapters 4, 5 and 6.1 concern the construction from ?, Theorem 3.20.

**Proposition 1.1.** (?, Theorem 3.20)

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

The main contribution is an algorithm for constructin the cylindrical decomposition described in ?, Theorem 3.20, for the semialgebraic case.

**Theorem 1.1.** Let

$$F_1, \dots, F_k$$

be quantifier-free Boolean formulas containing  $s$  different polynomials  $\mathbf{F} = \{f_1, \dots, f_s\} \in \mathbb{Z}[x_1, \dots, x_n]$ , 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  $F_1, \dots, F_k$  as input, which constructs a cylindrical algebraic decomposition  $\mathcal{D}$  of  $\mathbb{R}^n$  satisfying the frontier condition, and monotone with respect to each  $V_1, \dots, V_k$ . This algorithm has complexity ??? (at least that of CAD)

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

**Theorem 1.2.** *Let*

$$F_1, \dots, F_k$$

*be quantifier-free Boolean formulas containing  $s$  different polynomials  $\mathbf{F} = \{f_1, \dots, f_s\} \in \mathbb{Z}[x_1, \dots, x_n]$ , 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  $F_1, \dots, F_k$  as input, which constructs an  $\mathbf{F}$ -invariant CAD  $\mathcal{D}$  of  $\mathbb{R}^n$ , which is obviously compatible with each  $V_1, \dots, V_k$ , such that each cell  $C \subset V_1 \cup \dots \cup V_k$  is a smooth manifold and the graph of a quasi-affine map. 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.*

Chapter 5 discusses, following the proofs of ?, 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  $V_1, \dots, V_k$ . An algorithm for constructing this refinement, starting with the polynomials which have constant sign on cells of  $\mathcal{D}$ , is presented.

**Theorem 1.3.** *Let*

$$F_1, \dots, F_k$$

*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  $\mathbf{F} = \{f_1, \dots, f_s\}$  be a set of polynomials in  $\mathbb{Z}[x_1, \dots, x_n]$  of maximum degree  $d$  and  $\mathcal{D}$  be an  $\mathbf{F}$ -invariant CAD of  $\mathbb{R}^n$  compatible with each  $V_1, \dots, V_k$  and such that each cell  $C \subset V_1 \cup \dots \cup V_k$  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  $V_1, \dots, V_k$ . This algorithm has complexity*

Finally, Chapter 6 describes how to complete the construction described in ?, 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 ?, 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  $V_1 \cup \dots \cup V_k$  is the union of some cells of  $\mathcal{D}''$ . In Section

6.2, an algorithm, due to ?, 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 ?, 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 1.4.** *Let  $\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$  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  $V_1, \dots, V_k$  ( $\dim(V_i) \leq 2, 1 \leq i \leq k$ ).*

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

*All of these algorithms are presented as psuedo-code. Chapter 8 discusses the implementation of this algorithm on top of Brown's QEPCAD (?) and presents some test cases.*

*The final contribution, in Chapter 7, 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  $(i_1, \dots, i_n) \in \{0, 1\}^n$  associated with each cell in a cylindrical decomposition. The following is proved.*

**Theorem 1.5.** *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  $\mathbb{R}[x_1, \dots, x_n]$ . 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  $(sd)^{O(1)n^2n}$ . This is also an upper bound on the number of cells in  $\mathcal{D}$ , number of polynomials defining cells and their degrees.*

*Finally, Chapter ?? 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 structures. 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  $\mathbb{R}$ .*

**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  $f \in \mathbb{R}[x_1, \dots, x_n]$ , 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  $Y \subset \mathbb{R}^m$ . A mapping  $f : X \rightarrow Y$  is called semialgebraic if its graph

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

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 2.3.** A quantifier-free Boolean formula is obtained by the following rules:

1. If  $f \in \mathbb{R}[x_1, \dots, x_n]$  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 2.4.** A first-order Boolean formula is obtained by rules 1 and 2 from Definition 2.3 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.*

**Theorem 2.1.** (?) Let  $S \subset \mathbb{R}^{n+1}$  and  $\text{proj}_{\mathbb{R}^n} : \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n$  be the projection map onto the first  $n$  co-ordinates. Then  $\text{proj}_{\mathbb{R}^n}(S) \subset \mathbb{R}^n$  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.** (Algorithm 14.21)

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

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

*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  $K[y_1, \dots, y_k, x_1, \dots, x_n]$ . Note that variables  $y_1, \dots, y_k$  are partitioned into  $\omega$  “blocks”  $Y_1, \dots, Y_\omega$  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  $\omega$  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*

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



where each  $f_{i,j,n}$  is a polynomial in  $K[x_1, \dots, x_n]$  and  $*_{i,j,\ell} \in \{<, =, >\}$  determines its sign. Bounds on this formula are as follows -  $I \leq s^{(k_1+1)(k_2+1)\dots(k_\omega+1)(n+1)} d^{O(k_1)O(k_2)\dots O(k_\omega)}$ , -  $J_i \leq s^{(k_1+1)(k_2+1)\dots(k_\omega+1)} d^{O(k_1)O(k_2)\dots O(k_\omega)}$ , -  $L_{i,j} \leq d^{O(k_1)O(k_2)\dots O(k_\omega)}$ , - polynomials  $f_{i,j,\ell}$  have maximum dergee  $d^{O(k_1)O(k_2)\dots O(k_\omega)}$ .

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

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

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 7.1).

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

**Proposition 2.2.** *Every semialgebraic subset of  $\mathbb{R}^n$  is the (not necessarily disjoint) union of finitely many semialgebraic subsets of the kind*

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

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

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 2.5.** 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  $\text{proj} : R^{n+1} \rightarrow R^n$  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 2.6.** A map  $f : A \rightarrow R^m$ , 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  $f : R^n \rightarrow R$  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 2.7.** (?, 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

$$f_1, \dots, f_r$$

in  $G$  satisfying the differential equations

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

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

A function  $f(\mathbf{x}) = P(\mathbf{x}, f_1(\mathbf{x}), \dots, f_r(\mathbf{x}))$ , where  $P(\mathbf{x}, f_1(\mathbf{x}), \dots, f_r(\mathbf{x}))$  is a polynomial of degree not greater than  $\beta \geq 1$  is called a Pfaffian function of order  $r$  and degree  $(\alpha, \beta)$ . Note that  $f$  is only defined in the domain  $G$ , where all functions  $f_1, \dots, f_r$  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 2.8.** (?, 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  $\text{cl}(X)$  is contained in  $G$ .

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

**Definition 2.9.** (?, Definition 2.8) A set  $X \subset \mathbb{R}^n$  is called sub-Pfaffian in an open domain  $G \subset \mathbb{R}^n$  if

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

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

**Definition 2.10.** (?, 2.9) Let  $I^k := [-1, 1]^k$  be the closed cube in an open domain  $G \subset \mathbb{R}^k$ .  $X \subset I^n$  is called restricted sub-Pfaffian if  $X := \text{proj}_{\mathbb{R}^n}(Y)$ , where  $Y \subset I^{n+m}$  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 (?). The restriction to the closed cube ensures that the restricted sub-Pfaffian sets are definable in an O-minimal structure. Furthermore, ? 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, semialgebraic sets satisfy the compliment theorem.*

**Definition 2.11.** (?, Definition 2.11)

Consider a semi-Pfaffian set

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

where  $f_{ij}$  and  $g_{ij}$  are Pfaffian functions of order  $r$  and dergee  $(\alpha, \beta)$  defined in an open domain  $G$  The format of  $X$  is a tuple

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

where  $N \geq (I_1 + J_1) + \dots + (I_M + J_M)$ .

### 2.1.2 Monotone cells

*We now present, closely following ?, some useful properties of definable sets. These properties can also be applied to semialgebraic sets, being a particular case of definable sets.*

**Definition 2.12.** (?, Definition 2.1) Let  $L_{j,c} := \{(x_1, \dots, x_n) \mid x_j = c\}$  for some  $1 \leq j \leq n$  and  $c \in \mathbb{R}$ . Each intersection of the kind

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

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

**Definition 2.13.** (?, Definition 2.2) Let  $\mathbf{f} = (f_1, \dots, f_k) : X \rightarrow \mathbb{R}^k$  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  $\text{proj}_L|_Y$  is injective if and only if  $\text{proj}_L(Y)$  is  $n$ -dimensional.

**Definition 2.14.** (?, Definition 2.3) Let  $\mathbf{f}: X \rightarrow \mathbb{R}^k$  be a bounded, continuous, quasi-affine 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 2.15.** (?, Notation 2.4)

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

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

let  $W$  be the linear subspace of  $\mathbb{R}^n$  where all coordinates in  $I$  are equal to zero. We write  $\text{span}\{x_{j_1}, \dots, x_{j_m}\}$  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  $\text{span}\{x_1, \dots, x_k\}$ .

**Definition 2.16.** (?, Definition 2.10) Let  $X \subset \text{span}\{x_1, \dots, x_n\}$  be a monotone cell and  $\mathbf{f}: X \rightarrow \text{span}\{y_1, \dots, y_k\}$  be a continuous map having graph  $Y$ . If  $Y$  is a monotone cell, then  $\mathbf{f}$  is called monotone on  $X$ .

**Definition 2.17.** (?, Definition 2.5)  $Y \subset \text{span}\{x_1, \dots, x_n\}$  is called a monotone cell if it is the graph of a monotone map  $\mathbf{f}: X \rightarrow H$  where  $H \subset \text{span}\{x_1, \dots, x_n\}$  and  $X \subset \text{span}\{x_1, \dots, x_n\} \setminus H$ .

**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  $\phi$  on  $\Delta$ . 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 (?, Example 4.3).

**Proposition 2.3.** (?, Theorem 1) Every monotone cell is a topologically regular cell.

The following properties of monotone cells are used in the proof of ?, Theorem 3.20.

**Corollary 2.1.** (?, 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  $\text{fr}(Y) \subset \text{fr}(X)$ . Then  $X \setminus Y$  is a disjoint union of two monotone cells.

**Proposition 2.4.** (*?, Theorem 10*) Let  $X \subset \mathbb{R}^n$  be a monotone cell. Then  $\text{proj}_L(X)$ , for any affine coordinate subspace  $L \subset \mathbb{R}^n$  is also a monotone cell.

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

We finally revise the definition of Big O notation, following *?, Definition 6.1*.

**Definition 2.18.** (*?, Definition 6.1*) Let  $f, g, h: \mathbb{N}^\ell \rightarrow \mathbb{N}$  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). *?* 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  $\mathbf{F} \subset \mathbb{R}[x_1, \dots, x_n]$  has a constant sign on every cylindrical cell in the decomposition.

Before describing Collins' algorithm, we will first define, closely following *?*, a cylindrical cell and cylindrical decomposition. Note that *?* work in the broader class of definable sets.

**Definition 2.19.** 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*

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

- 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:  $(a, b)$ ,  $(-\infty, b)$ ,  $(a, \infty)$ ,  $(-\infty, \infty)$  where  $a, b \in \mathbb{R}$ .
- When  $n > 1$ , suppose that  $(i_1, \dots, i_{n-1})$ -cells are already defined and let  $C'$  be one of these cells. An  $(i_1, \dots, i_{n-1}, 0)$ -cell (section cell) is the graph of a continuous definable function  $f: C' \rightarrow \mathbb{R}$ . An  $(i_1, \dots, i_{n-1}, 1)$ -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  $f, g : C' \rightarrow \mathbb{R}$  are continuous definable functions such that  $f(\mathbf{x}) < g(\mathbf{x})$  for all  $\mathbf{x} \in C'$ .

**Definition 2.20.** Let  $S \subset \mathbb{R}^n$  be a definable subset of  $\mathbb{R}^n$  and define the *frontier* of  $S$

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

where  $\text{cl}(S)$  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  $\text{fr}(C)$  into three subsets: the top ( $C_T$ ), bottom ( $C_B$ ) and side-wall ( $C_W$ ).

First let  $C$  be a sector  $(i_1, \dots, i_{n-1}, 1)$ -cell and let  $C' := \text{proj}_{\mathbb{R}^{n-1}}(C)$ . If  $C$  is bounded from below by a continuous function  $f : C' \rightarrow \mathbb{R}$ , then  $C_B$  is the graph of  $f$ . Similarly, if  $C$  is bounded from above by the graph of a continuous function  $g : C' \rightarrow \mathbb{R}$ , 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  $(i_1, \dots, i_k, 0, \dots, 0)$ -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 2.21 and Example 7.1).

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 2.21.** (?, Definition 9.2)

Let  $\varphi : W \rightarrow \mathbb{R}$  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  $\Phi$  if

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

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 ?, Lemma 3.14).

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

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

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

We now define the cylindrical decomposition.

**Definition 2.22.** 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  $D$  be a partition of  $\mathbb{R}^n$  into cylindrical cells. Define  $D'$  to be the set of all projections  $C' := \text{proj}_{\mathbb{R}^{n-1}}(C)$  for all  $C$  in  $D$ .  $D$  is a cylindrical decomposition of  $\mathbb{R}^n$  if  $D'$  is a cylindrical decomposition of  $\mathbb{R}^{n-1}$ . We call  $D'$  the \*decomposition of  $\mathbb{R}^{n-1}$  induced by  $D$ .

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

**Definition 2.23.** Let  $S \subset \mathbb{R}^n$  be a definable set. A cylindrical decomposition  $D$  is compatible with  $S$  if every cell  $C$  of  $D$  is either a subset of  $S$  or disjoint from  $S$ .

**Definition 2.24.** A cylindrical decomposition  $E$  is called a refinement of  $D$  if  $E$  is compatible with every cell of  $D$ . In other words, every cell of  $D$  is a union of cells in the refinement  $E$ .

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

We now present some useful results proved by ? relating to desirable properties of cylindrical decompositions.

**Definition 2.25.** 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 2.26.** Let  $D$  be a cylindrical decomposition of  $\mathbb{R}^n$  and  $C$  be a cell of  $D$ .  $C$  satisfies the frontier condition in  $D$  if  $\text{fr}(C)$  is a union of cells of  $D$  of smaller dimension. If it is unambiguous that  $C$  is a cell of  $D$ , then we just say that  $C$  satisfies the frontier condition. If every cell in  $D$  satisfies the frontier condition, then the cylindrical decomposition  $D$  satisfies the frontier condition.

We saw in Example ??*exm:top-bottom-not-cylindrical*) 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  $D$ , 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 @??*exm:side-wall*) 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.,  $\text{cl}(V'') \cap I = v$ ). The tangent at the origin of  $c := \text{proj}_{\mathbb{R}^2}(V'')$  is  $1/2$ . The pre-image of the projection map  $W' := \text{proj}_{\mathbb{R}^2}^{-1}|_{\text{cl}(W)}(c)$  would satisfy the condition  $\text{cl}(W') \cap I = v'$ , 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 which the side-walls of  $V$  and  $W$  are single 1-dimensional cells cannot exist.

**Proposition 2.5.** (*?, Lemma 3.3*) Let  $C$  be an  $(i_1, \dots, i_{k-1}, 0, i_{k+1}, \dots, i_n)$ -cell. Then  $C' := \text{proj}_{\text{span}\{x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n\}}(C)$  is a cylindrical  $(i_1, \dots, i_{k-1}, i_{k+1}, \dots, i_n)$ -cell and  $C$  is the graph of a monotone map  $f : C' \rightarrow \text{span}\{x_k\}$ .



**Proposition 2.6.** (*?, 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  $W \subset \text{fr}(C)$  has exactly two connected components, each of which is either a point or a closed curve interval.*

### 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 ? 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 2.27.** Let  $\mathbf{F} := (f_1, \dots, f_s) \subset \mathbb{Z}[x_1, \dots, x_n]$  be a set of polynomials with integer coefficients. A cylindrical algebraic decomposition  $D$  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., ?) 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  $\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$  be a set of polynomials with integer coefficients. We will now describe the algorithm proposed by ?, closely following ?, for constructing an  $\mathbf{F}$ -invariant CAD of  $\mathbb{R}^n$ .*

*Collins' algorithm uses the idea of projections 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 crosses one another. This second property is called delineability.*

*Let us begin the description of the algorithm by considering a single polynomial  $f \in \mathbb{Z}[x_1, \dots, x_n]$ . 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 2.7.** (*?, Proposition 2.16*) Let  $f \in \mathbb{Z}[x_1, \dots, x_n]$  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

$$\psi_1, \dots, \psi_\ell : C' \rightarrow \mathbb{R}$$

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

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

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  $z_1, \dots, z_k$  be the distinct roots of  $f(\mathbf{c}, x_n)$  with multiplicities  $m_1, \dots, m_k$ . Choose  $\varepsilon > 0$  small enough that the open discs  $D(z_i, \varepsilon)z_i \in \mathbb{C}$  (center  $z_i$ , radius  $\varepsilon$ ) 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  $d = m_1 + \dots + m_k$  complex roots counted with multiplicities, it follows that each  $D(z_i, \varepsilon)$  contains exactly one root, denoted  $\zeta_i$ , of  $f(\mathbf{b}, x_n)$  with multiplicity  $m_i$ . If  $z_i$  is real, then  $\zeta_i$  is real, otherwise the complex conjugate of  $\zeta_i$  would be another root of  $f(\mathbf{b}, x_n)$  in the disc  $D(z_i, \varepsilon)$ . If  $z_i$  is complex then  $\zeta_i$  is also complex, since the conjugation of every point in  $D(z_i, \varepsilon)$  lies outside  $D(z_i, \varepsilon)$ , forming another disc  $D(\bar{z}_i, \varepsilon)$ . 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  $\ell$ . Define  $\psi_i(\mathbf{x}) : C' \rightarrow \mathbb{R}$  to be the continuous (by making  $\varepsilon$  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  $\psi_i$  has constant multiplicity. Observe that the graph of each  $\psi_i$  can be expressed by a first-order Boolean formula, using existential quantifiers to express each of the  $\ell$  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 function  $\psi_i$  is a semialgebraic set.  $\square$

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

**Proposition 2.8.** (*?, Proposition 2.18*) Let  $f, g \in \mathbb{Z}[x_1, \dots, x_n]$ ,  $C' \subset \mathbb{R}^{n-1}$  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  $\phi, \psi : C' \rightarrow \mathbb{R}$  be continuous semialgebraic functions such that  $f(\mathbf{x}, \phi(\mathbf{x})) = 0$  and  $g(\mathbf{x}, \psi(\mathbf{x})) = 0$  for all  $\mathbf{x} \in C'$ . If there exists  $\mathbf{c} \in C'$  such that  $\phi(\mathbf{c}) = \psi(\mathbf{c})$ , then  $\phi(\mathbf{x}) = \psi(\mathbf{x})$  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  $z_1 = \phi(\mathbf{c}) = \psi(\mathbf{c}), \dots, z_k$  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  $\min(m_1, p_1) + \dots + \min(m_k, p_k)$  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  $\min(m_1, p_1) + \dots + \min(m_k, p_k)$ , 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  $\phi(\mathbf{b}) = \psi(\mathbf{b})$ . Since  $C'$  is connected, this equality holds for all  $\mathbf{x} \in C'$ .  $\square$

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

$$\begin{aligned} f(x_n) &= a_d x_n^d + \dots + a_1 x_n + a_0 \\ g(x_n) &= b_e x_n^e + \dots + b_1 x_n + b_0. \end{aligned}$$

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

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

**Definition 2.29.** Let  $f, g \in \mathbb{Z}[x_1, \dots, x_{n-1}][x_n]$  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  $f, g \in \mathbb{Z}[x_1, \dots, x_{n-1}][x_n]$  and  $X \subset \mathbb{Z}[x_1, \dots, x_n]$ .

- $f$  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  $\mathbb{Z}[x_1, \dots, x_{n-1}]$ , we need to take the principal subresultant coefficients of the reductum of  $f$  (or  $g$ ). (? , pp36)

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

**Definition 2.30.** Consider  $f \in \mathbb{Z}[x_1, \dots, x_n]$  as a univariate polynomial in  $x_n$ . I.e.,

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

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

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

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

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

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

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

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

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

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

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

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

$$\psi_1, \dots, \psi_\ell : C' \rightarrow \mathbb{R}$$

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

**Proposition 2.9.** (?. Algorithm 11.2) Let  $\mathbf{F} := f_1, \dots, f_s$  be set of polynomials in  $\mathbb{Z}[x_1, \dots, x_n]$  with maximum degree  $d$ . There is an algorithm, taking  $\mathbf{F}$  as input, which produces an  $\mathbf{F}$ -invariant CAD  $D$  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  $D$ , number of polynomials defining cells and their degrees.

Observe that these bounds are doubly exponential in the number of variables. Indeed, ? 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 trath invariant or partial CAD.

**Proposition 2.10.** Let  $S_1, \dots, S_k$  be a finite collection of semialgebraic subsets of  $\mathbb{R}^n$  defined by quantifier-free Boolean formulas  $F_1, \dots, F_k$  respectively. Together, these formulas include  $s$  different polynomials in  $\mathbb{Z}[x_1, \dots, x_n]$  with maximum degree  $d$ . There is an algorithm, taking  $\{F_1, \dots, F_k\}$  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 @??prp:collins) but, in practice, more efficient algorithms exist (see e.g., ? and ?).

*Proof.* Let  $f_1, \dots, f_s \in \mathbb{R}[x_1, \dots, x_n]$  be the polynomials from formulas  $F_1, \dots, F_k$  and let  $\mathcal{E}'$  be the CAD of  $\mathbb{R}^n$  produced by Proposition 2.9 with  $f_1, \dots, f_s$  as input. Since  $F_i$  for each  $1 \leq i \leq k$  provides a set of sign conditions on polynomials  $f_1, \dots, f_s$ ,  $\mathcal{E}'$  is a refinement of the CAD produced by this algorithm.

Truth values of  $F_1, \dots, F_k$  on each cell can easily be computed from the CAD produced by Proposition 2.9 by considering the signs of  $f_1, \dots, f_s$  on each cell, which we computed in the construction of  $\mathcal{E}'$ . Observe that this does not change the asymptotic complexity or bounds in Propoosition ??.

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

**Corollary 2.2.** *Let  $S \subset \mathbb{R}^n$  be a semialgebraic set defined by a quantifier-free Boolean formula  $F$  containing  $s$  different polynomials in  $\mathbb{R}[x_1, \dots, x_n]$ , 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 2.9.*

*Proof.* Immediate, by applying Proposition 2.2 to  $\{F\}$ .  $\square$

The construction described in Proposition 2.10 is somewhat naive. Constructions which are far more efficient in practice are presented in e.g., ?. An alternative approach is presented by ?. While very useful in practice, these constructions are unable to lower the upper bound from Proposition 2.9 due to some extreme cases, e.g., when the input formulas define a sign-invariant CAD (see ?, 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, ? 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  $\text{proj}(\mathbf{F})$ . ? then proved that a similar construction is possible in dimension 3. ? 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 2.31.** (?, 6.1) Let  $\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$  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  $f \in \text{prim}(\mathbf{F})$ ,  $f(\mathbf{x}, y) = 0$  for all  $y \in \mathbb{R}$  on a finite number of points  $\mathbf{x} \in \mathbb{R}^{n-1}$ . (?, condition WO1)
- $\text{proj}(\mathbf{F})$  is well-oriented. (?, condition WO2)

? 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 (?).

Let  $\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$  be a well-oriented set of polynomials and  $D$  be an  $\mathbf{F}$ -invariant CAD constructed using the McCallum projection operator (?). Then, by ? Theorems 2.2.3 and 2.2.4, every cell  $C$  of  $D$  is an analytic submanifold of  $\mathbb{R}^n$ . Informally, if  $\dim(C) = k$ , then  $C$  is a non-empty subset of  $\mathbb{R}^n$  which “looks locally like  $\mathbb{R}^k$ ” (see Definition 2.32 for the precise definition).

**Definition 2.32.**  $S \subset \mathbb{R}^n$  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.

**Proposition 2.11.** *Let  $S \subset \mathbb{R}^n$  be a semialgebraic set defined by a quantifier-free Boolean formula constructed from the set of polynomials  $\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$ , such that  $\mathbf{F}$  is well-oriented. Then a CAD  $D$  of  $\mathbb{R}^n$  compatible with  $S$  and such that every cell  $C$  of  $D$  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, ? completed the proof that Lazard's projection operator is valid. He 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 Proposition 2.11 further:

**Proposition 2.12.** *Let  $S \subset \mathbb{R}^n$  be a semialgebraic set defined by a quantifier-free Boolean formula constructed from the set of polynomials  $\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$ . Then a CAD  $D$  of  $\mathbb{R}^n$  compatible with  $S$  and such that every cell  $C$  of  $D$  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 2.7 allows us to pass from a projection polynomial  $f \in \mathbb{Z}[x_1, \dots, x_k]$  to the continuous semialgebraic functions  $\phi : C' \rightarrow \mathbb{R}$ , where  $C' \subset \mathbb{R}^{n-1}$  is a cylindrical cell, from Definition 2.19 which define some of the section cells in the cylinder  $C' \times \mathbb{R}$ . The construction described in ?, Theorem 3.20 (and related lemmas) works with these continuous semialgebraic functions. However, we are given only the projection polynomials  $f$  of which  $\phi$  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  $\phi : C' \rightarrow \mathbb{R}$ , 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,  $\phi$  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  $C' := \text{proj}_{\mathbb{R}^1}(C) = \{x \in \mathbb{R} \mid x > 0\}$ . We can write

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

such that  $C$  is the graph of  $\phi|_{C'}$ . However, it's clear that  $\phi$  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  $\phi$  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  $f \in \mathbb{Z}[x_1, \dots, x_n]$  and  $C' := \text{proj}_{\mathbb{R}^{n-1}}(C)$  satisfy the conditions in Proposition ??prp:coste-polynomial-to-cell). 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  $C' \times \mathbb{R}$ , including 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 (?). I.e., a root of a polynomial may be defined by sign conditions on its derivatives. Thus, let

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

where  $g_1, \dots, g_k$  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 Proposition 2.10. Indeed, *QEPCAD-B*, being designed primarily to perform quantifier elimination, frequently constructs a “partial” CAD of  $\mathbb{R}^n$  (?). We will force the construction of the sign-invariant CAD and use McCallum's projection operator so as to obtain smooth cells (see Proposition 2.11).

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 will suffice. Indeed, this is a common requirement and Maple's 'RegularChains::CylindricalAlgebraicDecompose' function provides output in this representation (?). 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* (?, ?). The extended language includes the rules of first-order Boolean formulas (see Definition 2.3) along with a means of representing “the  $i$ -th root of polynomial  $f$ ”. We find this representation a little inconvenient (TODO why?) 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

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



*\*\* TODO: draw a picture, label cells by  $C\_1$  and  $C\_2$ . \*\* The two distinct cells  $C_1$  and  $C_2$  having sign  $x > 0, x - y^2 = 0$ . 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. ? proposed an algorithm for constructing a solution formula where 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 ? 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 2.33.** Let  $C_1$  and  $C_2$  be distinct cells in a CAD  $D$  of  $\mathbb{R}^n$ . Each cell  $C$  is assigned a truth value  $T_C \in \{\text{True}, \text{False}, \text{undet}\}$  and a list  $S_C = (S_1, \dots, S_k)$  where  $S_i = (s_{i,1}, \dots, s_{i,s_i}, 1 \leq i \leq k$  is the list of signs of level- $i$  (factorised) projection polynomials  $f_1, \dots, f_{i_s} \in \mathbb{Z}[x_1, \dots, x_i]$ , with  $s_{i,j} \in \{-1, 0, 1\}$ .  $(C_1, C_2)$  is called a conflicting pair in  $D$  if  $T_{C_1}, T_{C_2} \in \{\text{True}, \text{False}\}$  and  $T_{C_1} = T_{C_2}$  and  $S_{C_1} = S_{C_2}$ .

*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 Boolean 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.*

*In 1957, ? 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 (?). 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, ? 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. ? 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 ?, Section 6) will be presented.*

**Definition 3.1.** (?, 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

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

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 assume any other desirable properties of the manifolds  $X_k$ . E.g., the closure of a stratum may not be a union of strata.

### 3.0.1 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 needed to define all the strata is based on the format of the input set. As every polynomial has a finite number of partial derivatives, this bound will simply be the degree of the polynomial in the semialgebraic case.

We will need the following notation for partial derivatives and the partial differential operator defined by ?.

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

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

**Definition 3.3.** (?, 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

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

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

We can now present the algorithm from ?, Theorem 2.

Let

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


---

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

**Input:**

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

**Output:**

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

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

---

Proceed by induction on  $i_k$ .

- Base case:  $i_k = n$ .
  - Return  $X_{x+1} = (\emptyset, \dots, \emptyset)$ .
- Otherwise
  - Initialise
    - $X' := X = \{f_1 = 0, \dots, f_s = 0, g_1 > 0, \dots, g_t > 0\}$ ,
    - $F := \{((0, \dots, 0, 1), f_1), \dots, ((0, \dots, 0, s), f_s)\}$  (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  $\mathbf{i}, \mathbf{j} \in \mathbb{Z}_{\geq 0}^n$ ,
    - let  $\mathbf{i} < \mathbf{j}$  mean that  $\mathbf{i}$  is lexicographically less than  $\mathbf{j}$
    - and  $\mathbf{i} \preceq \mathbf{j}$  denote  $\mathbf{i} < \mathbf{j}$  or  $\mathbf{i} = \mathbf{j}$ .
  - We will iterate over each index  $\mathbf{m} = (i_n, \dots, i_{k+1}, j) \in \mathbb{Z}_{\geq 0}^{n-k+1}$  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  $F$  will be computed.
  - For each  $f \in F$ , 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

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

where  $f_j$  is the polynomial in  $F$  with index  $(0, \dots, 0, j)$ , i.e., the  $j$ -th input polynomial.

- In practice, let  $h_{k+1}$  be the polynomial in  $F$  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

$$(X_{k+2}^{\mathbf{m}}, \dots, X_n^{\mathbf{m}}) := \text{Stratify}(k+1, F, (s, : \mathbf{h}), (i_\ell : \mathbf{i}), G)$$

where  $x : L$  denotes the **cons** operator. I.g., add  $x$  to the beginning of list  $L$ .

- Let  $\text{Empty}(X, s)$  be a subroutine returning true if  $s \in \mathbb{Z}[x_1, \dots, x_n]$  is identically zero on the set  $X \subset \mathbb{R}^n$ .
- If  $\text{Empty}(X, s)$ , we are finished.
- Return

$$X_{k+1} = (X_{k+1}, X_{k+1}, \dots, X_n).$$

where  $X_i, k+2 \leq i \leq n$  is the union of all strata  $X_i^{\mathbf{m}}$  computed in the recursive calls.

- Otherwise,
  - Let
    - $X' := \{\mathbf{x} \in \mathbb{R}^n \mid s(\mathbf{x}) = 0\}$ ,
    - $F := F \cup \{((0, \dots, 0, i_\ell, \dots, i_{k+1}, j), s)\}$ ,
    - $F := F : s$  (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  $F \subset \mathbb{Z}[x_1, \dots, x_n]$  and  $G \subset \mathbb{Z}[x_1, \dots, z_n]$ , compute

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

to obtain a set of strata  $X' = (X_1, \dots, X_n)$ . 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  $X = (\emptyset, X_1, \dots, x_n)$  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  $X_1 \cup \dots \cup X_n = X$ . Note, in the case that  $X_0 = X$ , the smooth stratification algorithm provides no information about the dimension of  $X$ .

### 3.0.2 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 ?, Theorem 1, is to check whether every derivative computed at step  $k + 1$  vanishes on the set

$$U_k \cap \{x_1 = \text{const} \mid i < i_1\}$$

where  $i_k \in \{1, \dots, n\}$  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  $\text{Empty}(S)$ , 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  $\text{Empty}(S)$  as a quantifier elimination problem

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

This problem can be decided using the singly-exponential quantifier elimination algorithm from [14], Algorithm 14.21, which has complexity singly exponential in the number of variables. In practice, quantifier elimination can be performed using CAD. [14] propose an algorithm for determining whether a semialgebraic set is empty, which has complexity  $(sd)^{O(n)}$ , for a semialgebraic set defined by  $s$  polynomials having maximum degree  $d$  in  $n$  variables.

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 strata  $Y_k$ . 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  $X' \cap Y_1 = \emptyset$ . A similar situation may arise, even with nonconstant functions  $s_k$ . For example, for the input set

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

the derivative for index  $(0, 1)$  is  $\partial(x^2 y^2)/\partial x_1 = 2xy^2$ , 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  $(0, \dots, i_\ell - 1, i_{\ell-1}, \dots, i_{k+1}, j)$ . The following observation allows us to quickly find this polynomial, avoiding a search in  $G$  or unnecessary recomputation of derivatives. Suppose we are considering index

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

and want to find the derivative with index

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

The set  $G$  includes polynomials computed in previous rounds of induction and those with index  $\mathbf{i} < \mathbf{m}$ , since we proceed in ascending lexicographical order. Hence  $G$  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  $f_j \in \mathbb{Z}[x_1, \dots, x_3]$  with  $M_j = (1, 2, 2)$  and consider the lexicographically ordered list of indices

$$\begin{aligned} L_1 &:= (0, 0, 0), (0, 0, 1), (0, 0, 2), (0, 1, 0), (0, 1, 1), (0, 1, 2), (1, 0, 0), (1, 0, 1) \dots \\ L_2 &:= (0, 0, 0), (0, 0, 1), (0, 0, 0), (0, 0, 1), (0, 0, 2), (0, 0, 0), (0, 0, 1) \dots \end{aligned}$$

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

- Initialise  $G_j = ((0, \dots, 0), f_j)$ . For  $\mathbf{m} = (0, \dots, 0, 1)$ , set  $G'_j = G_j$ .
- Now suppose that  $G_j$  contains all polynomials with index  $\prec \mathbf{m}$ .
  - If  $\mathbf{m} = (0, \dots, 0, 1, 0, \dots, 0)$ , then  $\mathbf{j} = (0, \dots, 0)$  and we set  $G'_j$  to the head of  $G_j$ .
  - Otherwise, let  $G'_j$  be the tail of  $G'_j$ .

The “index chasing” method works efficiently in *SACLIB*, which uses linked lists. However, we do need to append new derivatives to the end of  $G_j$ , which takes  $O(n)$  steps. To avoid this, we keep a pointer to the last-but-one element in the list, i.e., suppose  $G_j = (a_1, (\dots, (a_{r-1}, (a_r, (\text{NIL}))) \dots)$  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  $(a_r, (b, (\text{NIL})))$ .

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,

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

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*( $k, \dots$ ), with the matrix for  $k = 0$  being empty (i.e.,  $\mathbf{h} = \mathbf{i} = ()$  in the partial differential operator).

### 3.0.3 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  $\text{span}\{x_1, x_2\}$ . Note that the algorithm imposes an order on polynomials. Here we take  $f_1 := z$  and  $f_2 = x^2 - y^2$ , with  $f_1$  and  $f_2$  being assigned indices  $(0, 0, 0, 1)$  and  $(0, 0, 0, 2)$  respectively. The algorithm proceeds as follows*

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

$$(0, 0, 1, 1) \quad h_1 = f_1 = x_3 \quad (3.2)$$

$$s_1 = \partial f_1 / \partial x_1 = 0 \quad (3.3)$$

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

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

$$s_1 = \partial f_2 / \partial x_1 = 2x_1 \quad (3.6)$$

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

$$U_1 := \{\mathbf{x} \in \mathbb{R}^3 \mid x_1^2 - x_2^2 = 0, 2x_1 \neq 0\} \quad (3.8)$$

$$\text{Not open, proceed by induction} \quad (3.9)$$

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

$$(0, 1, 1) \quad h_2 = f_1 \quad (3.11)$$

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

$$(0, 1, 2) \quad h_2 = f_2 \quad (3.13)$$

$$s_2 = 0 \text{ since } h_1 = h_2 \quad (3.14)$$

$$(1, 0, 1) \quad h_2 = f_1 \quad (3.15)$$

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

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

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

$$\text{Note } Y_1 = Y_2 = U_2 \quad (3.19)$$

$$Y_2 \text{ is smooth and has codimension 2} \quad (3.20)$$

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

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

$$\text{Observe } X' = (0, 0, 0) \quad (3.23)$$

$$(0, 0, 2, 1) \quad h_1 = 0, s_1 = 0 \quad (3.24)$$

$$(0, 0, 2, 2) \quad h_1 = 2x_1 \quad (3.25)$$

$$s_1 = 2 \quad (3.26)$$

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

$$U_1 := \{\mathbf{x} \in \mathbb{R}^3 \mid 2x_1 = 0, 2 \neq 0\} \quad (3.28)$$

$$\text{Not open, proceed by induction} \quad (3.29)$$

$$k = 1, h_1 = 2x, i_1 = 1, X'' := Y_1, F = \{f_1 = x_3, f_2 = x_1^2 - x_2^2, f_3 = 2x_1\} \quad (3.30)$$

$$(0, 1, 1) \quad h_2 = f_1 = x_3 \quad (3.31)$$

$$s_2 = 0 \quad (3.32)$$

$$(0, 1, 2) \quad h_2 = f_2 = x_1^2 - x_2^2 \quad (3.33)$$

$$s_2 = \det \begin{pmatrix} \partial h_1 / \partial x_1 & \partial h_1 / \partial x_2 \\ \partial h_2 / \partial x_1 & \partial h_2 / \partial x_2 \end{pmatrix} = \det \begin{pmatrix} 2 & 0 \\ 2x_1 & 2x_2 \end{pmatrix} = 4x_2 \quad (3.34)$$

$$Y_2 := \{\mathbf{x} \in X'' \mid 4x_2 \neq 0\} = \emptyset \quad (3.35)$$

$$(0, 1, 3) \quad h_2 = f_3 = 2x_1 \quad (3.36)$$

$$s_2 = 0 \quad (3.37)$$

$$(1, 0, 1) \quad h_2 = f_1 = x_3 \quad (3.38)$$

As expected, the algorithm outputs the set of strata

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

### 3.0.4 Determining the codimension and basic formula for each stratum

?, 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.

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

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

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

$$k = 1, h_1 = f_1, i_1 = 2, X'' := Y_1 \dots$$

No partial derivatives can be computed.

Return.  $Y_1$  is smooth.

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  $(1, 1, 1)$ .

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

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,  $f_1 := x_1 x_2$  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. ?, Theorem 1 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  $X_k$ .

### 3.0.4.1 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

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

of codimension  $k$ , where the function  $h_k := s_k^{(0, \dots, 0, m_\ell-1, m_{\ell-1}, \dots, m_1, j)}$ . According to ?, 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  $s_1, \dots, s_{k-1}$  which appear in the definition of  $Y_{k-1} \supset Y_k$  should also be taken into account. Begin with a set

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

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

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

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.

### 3.0.5 From Basic Algebraic sets to Arbitrary Semialgebraic sets

?, 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

$$f_1, \dots, f_s.$$

Consider each “sign set”

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

where  $*, 1 \leq i \leq s \in \{<, >, =\}$ .  $S$  can be partitioned into a finite union of some of the  $s^3$  sign sets. If we apply the smooth stratification algorithm to each  $S_{(*_1, \dots, *_s)} \subset S$ , 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  $S$ .

## 3.1 Implementation Details

This algorithm has been implemented in  $\mathcal{C}$ , using the computer algebra library *SACLIB*. *QEPCAD* is used to perform emptiness checks. 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 3.0.3.

Enter a variable list.

(x,y,z)

Please enter a QEPCAD formula defining a basic semialgebraic set(i.e., conjunction of equations and inequalities)  
[ 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, taking as input the variable list (which specifies the order) and a conjunction of polynomial equations, inequations and inequalities (written as a QEPCAD 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  $L = (f_1, \dots, f_k) \subset \mathbb{Z}[x_1, \dots, x_r]$ ,
- Word  $Ineqs = (g_1, \dots, g_\ell) \subset \mathbb{Z}[x_1, \dots, x_r]$ ,
- Word  $V = (v_1, \dots, v_r)$  is a variable list,



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

**Output:**

- Word  $S$ : set of smooth strata  $(x_1, \dots, x_r)$ , such that  $X_i$  contains strata of codimension  $i$  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  $\{g_1 > 0, \dots, g_\ell > 0\}$ ,
- Word  $k$   $1 \leq k \leq r$  is the codimension currently beind worked with,
- Word  $np$ : number of polynomials,
- Word  $Fs$   $((f_1, \deg(f_1)), \dots, (f_{np}, \deg(f_{np})))$  is the list of input polynomials,
- Word  $Is$  is a partial index  $(0, i_1, \dots, i_{k-1})$ ,
- Word  $Hs$  is the set of polynomials  $(h_{i_1}, \dots, h_{i_{k-1}})$ ,
- Word  $Qs$  is a list of polynomials  $\subset \mathbb{Z}[x_1, \dots, x_n]$ ,
- 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  $Gs \subset \mathbb{Z}[x_1, \dots, x_r]$  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  $k \leq j \leq r$ ).

```

1 Word strat_helper(Word r, Word V, Word Ineqs, Word k, Word np, Word Fs, Word Is, Word
2 {
3     // add constraints  $x_1 = 0, \dots, x_{\{i1 - 1\}} = 0$ 
4     Word i0 = FIRST(Is); // number of variables considered so far
5
6     // base case for no polynomials?
7     if (np == 0 || i0 >= r) {
8         *strat_count_ = 0;
9         return NIL;
10    }
11
12    // list of H polynomials (rev order) without the last (zero) one.
13    Word Hs1 = RED(CINV(Hs));
14
15    // set up return value
16    Word Gs1 = NIL; // list of all differentials computed in this round, to return
17    Word Gs2 = NIL; // list of differentials produced during induction
18    Word Gs3 = NIL; // store Gs on current round
19
20    // set up working array

```

```

21 Word g_count = np; // how many differentials computed so far, index in Gs
22 Word Gs = NIL; // list of all differentials computed in this round, working set
23
24 // metadata
25 Word Ms[np]; // number of steps before maximum differentiation variable (i1) should be incren
26 Word Dvs[np]; // list of current differentiation variable (i1)
27 Word Backup[np]; // first polynomial in the list
28 Word Chase[np]; // first element is h1
29 Word ChaseIndex[np]; // chase array index of polynomial h1
30 Word Append[np]; // pointer to last but one element in list, for quick appending.
31
32 Word p_index = 0; // initial polynomial index, ranges over 0 <= p_index < np
33
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 in
55 Word count = 0; // number of derivatives computed so far, scalar value of (m_{i0 + 1}, ..., m
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 differen
60     if (p_index == np) {
61         p_index = 0;
62         n_finished = 0;
63
64         ++count;
65     }

```

```

66      Word v = Dvs[p_index]; // differentiation variable
67      Word m = FIRST(Ms[p_index]);
68
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
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
115 // - if Q is constant, Y1 must be the last candidate, since the next one includes a tr
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, J
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 equation 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  $(i_r, \dots, i_{k+1}, j)$ ,  $1 \leq j \leq np$ .
- Lines 57-100 keeps track of the current polynomial  $P$  and “differentiation variable”  $v \in x_{i_{k+1}}, \dots, i_r$ .
- 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  $\ell$ , and returns the  $(k \times k)$ -matrix obtained from  $A$  by appending column  $(\partial h_1 / \partial x_\ell, \dots, \partial h_{k-1} / \partial x_\ell)^T$  and row  $(\partial f / \partial x_{i_1}, \dots, \partial f / \partial x_{i_{k-1}}, \partial f / \partial \ell)$ . 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 computed, 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 3.0.2.

**Input:**

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

- *Word Gs*  $\subset \mathbb{Z}[x_1, \dots, x_r]$ ,
- *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)
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 more 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
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  $(x_1, \dots, x_r)$ ,
- Word  $Fs = (f_1, \dots, f_s) \subset \mathbb{Z}[x_1, \dots, x_r]$ ,
- Word  $Gs = (h_1, \dots, h_t) \subset \mathbb{Z}[x_1, \dots, x_r]$ ,
- Word *Ineqs*: list of atomic QEPCAD formulas  $\{g_1 > 0\}, \dots, \{g_\ell > 0\}$

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

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

**Output:**

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

```

1 Word ISEMPTY(Word r, Word V, Word Fs, Word Gs, Word Ineqs)
2 {
3     Word P, Ct, Cf;
4
5     // start with the list of inequalities
6     Word F = Ineqs;
7
8     // add equations
9     while (Fs != NIL) {
10         ADV(Fs, &P, &Fs);
11
12         F = COMP(LIST4(EQOP, P, r, NIL), F);

```



```

13     }
14
15     // and inequations
16     while (Gs != NIL) {
17         ADV(Gs, &P, &Gs);
18
19         F = COMP(LIST4(NEOP, P, r, NIL), F);
20     }
21
22     // complete formula by adding the inequation and conjunction
23     F = COMP(ANDOP, F);
24
25     // re-initialise qepcad before each run
26     QepcadCls Q;
27     INITSYS();
28
29     // set input formula
30     Q.SETINPUTFORMULA(V, LIST4(r, r, NIL, F));
31     Q.PRQFF();
32     Q.CADautoConst();
33
34     // special case: trivially false
35     if (Q.GVPC == 0) {
36         return true;
37     }
38
39     LISTOFCWTV(Q.GVPC, &Ct, &Cf);
40
41     /* compute cad */
42     return Ct == NIL;
43 }

```

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.

### 3.1.1 Complexity

We first briefly summarise the semi-Pfaffian case, as presented by ?. The algorithm from ?, Theorem 1 takes as input a semi-Pfaffian set with the format  $(r, N, \alpha, \beta, n)$  (see Definition 2.11) and has a complexity upper bound of

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

where

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

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

Now consider this algorithm applied to the semialgebraic set

$$X := \{\mathbf{x} \in \mathbb{R}^n \mid f_1 = 0, \dots, f_k = 0, g_1 > 0, \dots, g_\ell > 0\}, \quad (3.45)$$

where all  $s := k + \ell$  functions  $f_i, g_j$  are polynomials with maximum degree  $d$ . The upper complexity bound for the algorithm without Oracle is

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

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

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

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

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

These bounds follow immediately from ?. However, these results are theoretical. We will now analyse the complexity of the algorithm presented in Section 3.0.1, which performs emptiness checking on candidate strata.

There are  $s^n (d+1)^{1^{O(n)}}$  candidate strata  $Y_k$ , and the algorithm determines whether each of these is empty, in order to determine 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 complexity analysis purposes, we will use a theoretical algorithm, having singly exponential complexity in the number of variables. ? propose an algorithm which takes a semialgebraic set 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 the set. The algorithm can be used to determine the emptiness of a semialgebraic set 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

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

which is asymptotically the same as

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

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

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

which dominates the complexity of stratification, leading to an overall complexity upper bound of

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

### 3.1.2 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 + 1
}
```

```
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 [?], Theorem 3.18, for constructing a CAD whose two-, one-, and zero-dimensional cells are the graphs of quasi-affine maps.*

**Proposition 4.1.** (*[?], Theorem 3.19*)

*Let  $V_1, \dots, V_k$  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 every  $V_i$  such that every cylindrical cell contained in  $V := V_1 \cup \dots \cup V_k$  is the graph of a quasi-affine map.*

*The following will be proved.*

**Theorem 4.1.** *Let*

$$F_1, \dots, F_k$$

*be quantifier-free Boolean formulas containing  $s$  different polynomials  $\mathbf{F} = \{f_1, \dots, f_s\} \in \mathbb{Z}[x_1, \dots, x_n]$ , 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  $F_1, \dots, F_k$  as input, which constructs an  $\mathbf{F}$ -invariant CAD  $\mathcal{D}$  of  $\mathbb{R}^n$ , which is obviously compatible with each  $V_1, \dots, V_k$ , such that each cell  $C \subset V_1 \cup \dots \cup V_k$  is a smooth manifold and the graph of a quasi-affine map. 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 ?, Theorem 3.19 begins by introducing  $W$ , the smooth 2-dimensional locus of  $V = V_1 \cup \dots \cup V_k$ . 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 @??sec:smooth-strat). In order to apply the algorithm, consider the sign sets  $S_{(*_1, \dots, *_s)} \subset \mathbb{R}^n$  of polynomials  $f_1, \dots, f_s$  appearing in the formula  $F$ , as described in Section 3.0.5. Observe that a finite number of these sign-sets  $S_1, \dots, S_m$  form a partition of  $V$ . For each  $S_i \subset V$ , apply the smooth stratification algorithm to obtain a family  $(X_{i,1}, \dots, X_{i,n})$  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 of 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 ?, 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  $1 \leq i \leq k$ , we may assume that blow-up points will have dimension at most 0. Hence, the McCallum projection operator will result in smooth cells being produced. Since each input set  $V_i$  has dimension at most two,  $\text{cl}((V_1))$  will only contain zero-dimensional blow-up points.

Let  $\mathbf{F} \subset \mathbb{R}^n$  be the set of polynomials appearing in the quantifier-free Boolean formulas defining  $V$  and use the McCallum projection operator to construct an  $F$ -invariant CAD,  $E$ , of  $\mathbb{R}^n$ . Since cells of a CAD are disjoint by construction and smooth by the use of the McCallum projection operator,  $E$  forms a smooth stratification of  $V$ . Thus, we can take as  $W$  the (separate) sets of 2- and 1-dimensional cells of  $E$  which are contained in  $V$ . Let  $C$  be one of these cells, and suppose it has index  $(i_1, \dots, i_n) \in \{0, 1\}^n$ . Since  $\dim(C) = 2$ ,  $\sum_{1 \leq i \leq n} i_i = 2$ . Let  $i_{j_1} = i_{j_2} = 1$  for  $j_1 \neq j_2$  and all other elements of the index be equal to zero. Then  $\text{proj}_{\mathbb{R}^k}(C)$ , for  $k \neq j_1, k \neq j_2$ , is the root of a projection polynomial  $f_k \in \mathbb{Z}[x_1, \dots, x_k]$ . 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  $V \supset C$ . 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 decomposing a set into smooth manifolds. ?, Theorem 3.7 gives a method (which can be turned into an algorithm) for efficiently listing all these sign sets and ?, 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 preferable to many calls to the CAD algorithm, which would be required for the stratification-based approach. In addition, ? 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 cell  $C$  is the graph of a quasi-affine map if, for each affine coordinate subspace  $L \subset \mathbb{R}^n$ , the restriction of the projection map  $\text{proj}_L|_C$  is if and only if the image under this map is  $n$ -dimensional (see Definition 2.13). In order to satisfy this property, ?, Lemma 3.19 states that the CAD  $D$  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  $\text{span}\{x_i, x_j\}$  and  $\text{span}\{x_i\}$ .

Let  $W'_k$  be one of the strata 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

$$J_{i_1, \dots, i_\ell} := \begin{pmatrix} \partial f_1 / \partial x_{i_1} & \dots & \partial f_1 / \partial x_{i_\ell} \\ \vdots & & \vdots \\ \partial f_k / \partial x_{i_1} & \dots & \partial f_k / \partial x_{i_\ell} \end{pmatrix},$$

where  $\{i_1, \dots, i_\ell\} \subset \{1, \dots, n\}$ . Let  $\{j_1, \dots, j_{n-\ell}\} = \{1, \dots, n\} \setminus \{i_1, \dots, i_\ell\}$ . Regular points of the projection of  $W'$  onto  $\text{span}\{j_1, \dots, j_{n-\ell}\}$  are those at which the matrix  $J_{i_1, \dots, i_\ell}$  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  $J_{\{1, \dots, n\} \setminus \{i, j\}}$  does not have full rank.

Note that, for  $W'_{i,j}$ , the corresponding  $J_{\{1, \dots, n\} \setminus \{i, j\}}$  will be a  $((n - 2) \times (n - 2))$ -matrix, so we simply need to find the points at which  $\det(J_{\{1, \dots, n\} \setminus \{i, j\}}) = 0$ . On the other hand,  $J_{\{1, \dots, n\} \setminus \{i\}}$  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  $J_{\{1, \dots, n\} \setminus \{i\}}$ ,  $i \in \{1, \dots, n\}$  are the  $((n-2) \times (n-2))$ -jacobian matrices  $J_{\{1, \dots, n\} \setminus \{i, j\}}$ ,  $j \in \{1, \dots, n\} \setminus \{i\}$ . 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

$$J_{\{1, \dots, n\} \setminus \{i, j\}} \text{ for all } i \neq j,$$

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 Jacobian matrices  $J_{\{1, \dots, n\} \setminus \{i\}}$  for all  $1 \leq i \leq n$ , for the  $n-1$  polynomials defining  $W'_1$ . Critical points exist when 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.

---

**Input:**

$$(F_1, \dots, F_k)$$

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

$$V = V_1 \cup \dots \cup V_k$$

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

**Output:**

$$(D)$$

-  $D$ : an  $\mathbf{F}'$ -invariant CAD of  $\mathbb{R}^n$  compatible with each  $V_1, \dots, V_k$  such that each cell  $C \subset V$  is the graph of a quasi-affine map, where  $\mathbf{F}' \subset \mathbb{Z}[x_1, \dots, x_n]$  contains all polynomials from formulas  $F_1, \dots, F_k$  and, possibly, some additional polynomials which are zero at singlura points of  $V_1 \cup \dots \cup V_k$  and the critical points of projections of  $V_1 \cup \dots \cup V_k$  onto one- and two- dimensional coordinate subspace.

---

#### 4.2. ENSURING THAT EVERY CELL IS THE GRAPH OF A QUASI-AFFINE MAP 67

- Let

$$\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$$

be the set of polynomials appearing in the formulas  $F_1, \dots, F_k$ .

- Compute  $E$ , a CAD of  $\mathbb{R}^n$  compatible with  $V$ , using the McCallum projection operator.
- Let  $A = (A_1, \dots, A_n)$  be the family of projection polynomials for  $E$ , where each  $A_i \subset \mathbb{Z}[x_1, \dots, x_i]$ .
- For each cell  $C_k \subset V$  with index  $(i_1, \dots, i_n) \in \{0, 1\}^n$ , such that  $i_{j_1} = \dots = i_{j_k} = 1$  while all other elements of the index are equal to 0 (observe  $\dim(C_k) = k$ ):
  - Let  $f_1, \dots, f_{n-k}$  be a set of projection polynomials which are equal to 0 on the cell  $C_k$ .
  - For each  $i \in \{1, \dots, n\} \setminus \{j_1, \dots, j_k\}$ , find a polynomial in  $A_i$  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  $f_1, \dots, f_{n-2}$  defining  $C_2$ .
  - Let  $\mathbf{m} = \{j_1, \dots, j_\ell\} = \{1, \dots, n\} \setminus \{i_1, i_2\}$ .
  - Define

$$J_{C_2, \mathbf{m}} := \begin{pmatrix} \partial f_1 / \partial x_{i_1} & \dots & \partial f_1 / \partial x_{i_\ell} \\ \vdots & & \vdots \\ \partial f_{n-2} / \partial x_{i_1} & \dots & \partial f_{n-2} / \partial x_{i_\ell} \end{pmatrix} \quad (4.1)$$

be an  $((n-2) \times (n-2))$  Jacobi matrix.

$\det(J_{C_2, \mathbf{m}}) = 0$  at the critical points of  $\text{proj}_{\text{span}\{i_1, i_2\}}$ .

- For each 1-dimensional cell  $C_1 \subset V$  pair  $1 \leq i_1 < i_2 \leq n$ :
  - Find the set of polynomials  $f_1, \dots, f_{n-1}$  defining  $C_1$ .
  - Let  $\mathbf{m} = \{j_1, \dots, j_\ell\} = \{1, \dots, n\} \setminus \{i_1\}$ .
  - Define

$$J_{C_1, \mathbf{m}} := \begin{pmatrix} \partial f_1 / \partial x_{i_1} & \dots & \partial f_1 / \partial x_{i_\ell} \\ \vdots & & \vdots \\ \partial f_{n-1} / \partial x_{i_1} & \dots & \partial f_{n-1} / \partial x_{i_\ell} \end{pmatrix} \quad (4.2)$$

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

$\det(J_{C_1, \mathbf{m}}) = 0$  at the critical points of  $\text{proj}_{\text{span}\{i_1\}}$ .

- Return the CAD  $D$ , which is sign invariant on

$$\mathbf{F} \cup \bigcup_{C_2 \subset V, \mathbf{m} = \{1, \dots, n\} \setminus \{i_1, i_2\}} \det(J_{C_2, \mathbf{m}}), \quad \bigcup_{C_1 \subset V, \mathbf{n} = \{1, \dots, n\} \setminus \{i_1\}} \det(J_{C_1, \mathbf{n}})$$

for each two and one-dimensional cell  $C_1, C_2 \subset V$  of  $E$  and each pair  $0 \leq i_1 < i_2 \leq n$ .

### 4.3 Correctness and Complexity

Correctness of Theorem 4.1 follows from the proof of ?, 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  $V_1, \dots, V_k$ . By ?, Theorems 2.2.3 and 2.2.4, each cell  $C \subset V$  of  $E$  will be an analytic submanifold, since  $\dim(V_i) \leq 2, 1 \leq i \leq k$ . 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 @ (ref: eq-jacobi-two) and (4.2) respectively. These are the critical points of projections of two- and one- dimensional smooth cylindrical cells  $C \subset V_1, \dots, V_k$ . 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  $V_1 \cup \dots \cup V_k$  onto one- and two-dimensional coordinate subspaces, as required in the proof of ?, 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  $V = V_1 \cup \dots \cup V_k \subset \mathbb{R}^n$ , defined by  $s$  polynomials having maximum degree  $d$ . The smooth stratification step with emptiness checking has complexity

$$(sd)^{2^{O(n)}}.$$

The critical points of two- and one- dimensional strata are then computed. This is done by computing determinants of  $(n-2)$ - and  $(n-1)$ -dimensional Jacobi matrices, containing the  $n-2$  and  $n-1$  polynomial equations which define the strata of codimension 2 and 1 respectively. These polynomial equations have maximum degree  $(d+1)^{1^{O(n)}}$ . 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. Elementary row operations 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  $(n-2)(d+1)^{1^{O(n)}}$  and the  $((n-1) \times (n-1))$ -matrix will be  $(n-1)(d+1)^{1^{O(n)}}$ . These are both asymptotically the same as

$$n(d+1)^{1^{O(n)}}.$$

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  $(n-1) + (n-2) + \dots + 1 = 1/2(n-1)n$ , which is asymptotically the same as  $n^2$ .

A classical CAD, having constant sign on the  $s$  input polynomials plus the  $n + n^2$  Jacobi determinants is then constructed. According to Proposition 2.9 (? , Algorithm 11.2), classical CAD has complexity

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

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

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

which is asymptotically the same as

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

By Proposition 2.9, this is also an upper bound on the number of cells, number of polynomials defining the CAD and their maximum degree.



## Chapter 5

# Monotone Cells

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

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

**Example 5.1.** (? , 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  $\phi$  on  $\Delta$ . 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  $D$  be a CAD of  $\mathbb{R}^n$  and suppose that  $\mathbf{c} = (c_1, \dots, c_k)$  is a 0-dimensional section cell in the decomposition induced by  $D$  on  $\mathbb{R}^k$ . By (? , Remark 3.8), the set of all cells of  $D$  contained in the affine coordinate subspace

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

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

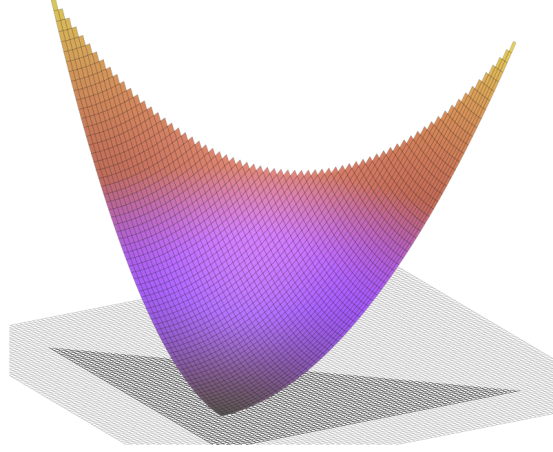


Figure 5.1: Plot of the 2-dimensional quasi-affine cylindrical cell  $Y$  defined in Example `refexm:monotone-cells`,

The construction of monotone cells is based on the following result due to ?.

**Proposition 5.1.** (?, Lemma 3.18)

Let  $X \subset \mathbb{R}^2$  be an open, bounded set and  $f : X \rightarrow \mathbb{R}$  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 ? 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  $D$  and let  $C'' := \text{proj}_{\mathbb{R}^1}(C)$ 
  - If  $\dim(C'') = 0$ , apply the inductive hypothesis to the sub-decomposition of  $D$  above  $C''$ .



- Otherwise,  $C$  has index  $\{1, i_2, \dots, i_n\} \in \{0, 1\}^n$ . Let  $\alpha$  be the smallest among  $\{2, \dots, n\}$  such that  $i_\alpha = 1$ . Since  $\dim(C) = 2$ , only  $i_1$  and  $i_\alpha$  are equal to 1, while all other elements of the index are equal to 0. By ?, Lemma 3.3,  $\text{proj}_{\text{span}\{x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n\}}(C)$ , for all  $i \in \{2, \dots, \alpha - 1, \alpha + 1, \dots, n\}$ , is a cylindrical cell and  $C$  is the graph of a continuous function from this cell to  $\text{span}\{x_i\}$ .

?, Lemma 3.19, which provides the construction of  $D$  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

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

By (?, 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 (?, Lemma 3.11), refinements of this kind preserve the cylindrical structure of  $D$  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 ?, 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  $\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$ . The following will be proved.

**Theorem 5.1.** *Let*

$$F_1, \dots, F_k$$

*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  $\mathbf{F} = \{f_1, \dots, f_s\}$  be a set of polynomials in  $\mathbb{Z}[x_1, \dots, x_n]$  of maximum degree  $d$  and  $\mathcal{D}$  be an  $\mathbf{F}$ -invariant CAD of  $\mathbb{R}^n$  compatible with each  $V_1, \dots, V_k$  and such that each cell  $C \subset V_1 \cup \dots \cup V_k$  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  $V_1, \dots, V_k$ . This algorithm has complexity*

## 5.1 Two-dimensional monotone sector cells

*Let  $C$  be an  $(i_1, \dots, i_n)$ -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  $I(c)$  be the family of these intervals associated with the point  $c$ . Define

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

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  $D$  on  $\mathbb{R}^\alpha$ . Thus, we need a way to translate between  $C'$  and  $X$ .

**Corollary 5.1.** (to Proposition 2.5 (?), Lemma 3.3))

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

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

The above follows from ?, Lemma 3.3 and the property that  $C$  is the graph of a quasi-affine map. An alternative proof is given here.

*Proof.* Since  $C$  is the graph of a quasi-affine map from  $X \rightarrow \text{span}\{x_2, \dots, x_{\alpha-1}\}$ , it follows that the projection map  $\text{proj}_{\text{span}\{x_1, x_\alpha\}}|_C$  is injective. This proves the first claim. The second claim follows by iteratively applying the projection defined in ?, Lemma 3.3 to obtain the 2-dimensional cylindrical sector cell  $X$ . By definition,

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

where  $\varphi, \psi : X' \rightarrow \mathbb{R}$  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, we take  $\varphi(x_1) = -\infty$  (resp.  $\psi(x_1) = \infty$ ) for all  $\mathbf{x} \in X'$ . Functions  $\varphi$  and  $\psi$  are either infinite or finite at every point in  $X'$ , otherwise, new one-dimensional section cells would exist whose projections onto  $\mathbb{R}^1$  form a proper subset of  $X'$ . This would contradict the cylindrical property. It is clear from this definition that  $I(c)$  contains a single open interval  $(\varphi(c), \psi(c))$ , proving the third claim. It follows that  $\gamma$ , which consists of the endpoints of intervals in

$I(c)$ , consists of one-dimensional curve intervals which are the graphs of  $\varphi$  and  $\psi$  (if finite).

The proof of the final claim is completed by defining the functions  $\varphi$  and  $\psi$ .

Since  $C' := \text{proj}_{\mathbb{R}^{\alpha-1}}(C)$  is a  $(1, \dots, 0, 0)$ -cell, it is the graph of a continuous definable map  $\mathbf{h} = (h_1, \dots, h_{\alpha-2}) : X' \rightarrow \mathbb{R}^{\alpha-2}$  and

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

where  $f, g : \mathbb{R}^{\alpha-1} \rightarrow \mathbb{R}$  are continuous definable functions such that  $f(\mathbf{x}) < g(\mathbf{x})$  for all  $\mathbf{x} \in C'$ . Again, if not bounded from above (resp. below),  $f$  (resp.  $g$ ) are taken to be  $-\infty$  (resp.  $\infty$ ) at every point. Observe that the graph of  $f$  (if finite) is the bottom of  $C$  and the graph of  $g$  (if finite) is the top of  $C$ .

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

$$\varphi : C'' \rightarrow \mathbb{R} \qquad \psi : C'' \rightarrow \mathbb{R} \qquad (5.1)$$

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

□

*The refinement is achieved by finding a set of real numbers*

$$(c_1, \dots, c_t)$$

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

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

*is a monotone cell. By (? , Theorem 1.7), this can be achieved by ensuring that*

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

*contains only monotone curve intervals. [? , Theorem 3.18.*

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

$$(c_1, \dots, c_t)$$

*such that the graphs of  $\varphi_{\{c_i < x_1 < c_{i+1}\}}$  and  $\psi_{\{c_i < x_1 < c_{i+1}\}}$  for all  $1 \leq i \leq t-1$  are monotone curve intervals. By ? , Theorem 1.7,  $X \cap \{c_i < x_1 < c_{i+1}\}$  for all  $1 \leq i \leq t-1$  is a semi-monotone set (? , proof of Lemma 3.18). For the graphs of  $\varphi$  and  $\psi$  to be monotone curve intervals, we require that  $\varphi$  and  $\psi$  are either strictly increasing in, strictly decreasing in or independent of  $x_1$ . Thus,  $(c_1, \dots, c_t)$  are the critical points of  $\varphi$  and  $\psi$ . Note that if  $X$  is not bounded from below, then  $\varphi$  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 $\varphi$ and $\psi$

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

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

$$A = (A_1, \dots, A_n), \quad (5.3)$$

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

Let  $C$  be a  $(1, 0, \dots, 0, 1)$ -cell of the decomposition induced by  $D$  on  $\mathbb{R}^\alpha$  such that  $\text{proj}_{\mathbb{R}^\alpha}^{-1}(C)$  is a 2-dimensional cell of  $\mathcal{D}$ . We now describe how to find algebraic numbers  $(c_1, \dots, c_t)$  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  $(1, 0, \dots, 0)$ -cell

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

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

Observe that

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

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

$$g_2 \in A_2, \dots, g_{\alpha-1} \in A_{\alpha-1}$$

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

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

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  $h : C' \rightarrow \mathbb{R}$ . In the first case, there is nothing to do, since  $C_T$  can be thought of as being independent of  $x_\alpha$ . In the second case,  $C_T$  can be written as

$$\{(x_1, \dots, x_\alpha) \mid a < x_1 < b, g_2(x_1, \dots, x_{\alpha-1}) = 0, \dots, g_{\alpha-1}(x_1, \dots, x_{\alpha-1}) = 0, x_\alpha = h(x_1, \dots, x_{\alpha-1})\}. \quad (5.4)$$

By the construction of  $D$  (e.g., using Proposition 2.11)  $C_T$  is  $C^\infty$  smooth, therefore  $h$  is a differentiable function everywhere in  $C'$ .

This representation lends itself naturally to the problem of Lagrange multipliers.

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

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

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

Critical points can be found by solving the system of equations

$$\frac{\partial f}{\partial x_i} - \lambda_1 \frac{\partial g_1}{\partial x_i} - \dots - \lambda_k \frac{\partial g_k}{\partial x_i} = 0, \quad (5.5)$$

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

for  $\lambda_1, \dots, \lambda_k$ .

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

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

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

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

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

Since  $C_T$  is a section cell, there exists a polynomial  $g_\alpha \in A_\alpha$  such that

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

Thus, there exists an algebraic variety

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

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

Note that  $g_\alpha \in \mathbb{Z}[x_1, \dots, x_\alpha]$ , while  $h$  from Equation (5.6) is expected to be a continuous differentiable function from  $\mathbb{R}^{\alpha-1} \rightarrow \mathbb{R}$ . We need to compute derivatives of the implicit  $g_\alpha$ . I.e.,

$$\frac{dx_\alpha}{dx_i} = \frac{\partial g_\alpha / \partial x_i}{\partial g_\alpha / \partial x_\alpha} \quad (5.8)$$

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  $dx_\alpha / dx_i$  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 can be written as

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

Thus, 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.,

$$\det \begin{pmatrix} \frac{\partial g_2}{\partial x_1} & \dots & \frac{\partial g_{\alpha-1}}{\partial x_1} & \frac{\partial g_\alpha}{\partial x_1} \\ \vdots & & & \vdots \\ \frac{\partial g_2}{\partial x_{\alpha-1}} & \dots & \frac{\partial g_{\alpha-1}}{\partial x_{\alpha-1}} & \frac{\partial g_\alpha}{\partial x_{\alpha-1}} \end{pmatrix} = 0. \quad (5.9)$$

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. Observe that this is a semialgebraic set, since  $\{d_T(\mathbf{x}) \neq 0\} \leftrightarrow \mathbb{R}^{\alpha-1} \setminus \{d_T(\mathbf{x}) = 0\}$ . 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 a set of polynomial equations

$$\{g_2, \dots, g_{\alpha-1}, J'_B, J'_T\}, \quad (5.10)$$

and inequalities

$$\{s_1 g_{1,1} > 0, \dots, s_\ell g_{1,\ell} > 0, d_B \neq 0, d_T \neq 0\} \quad (5.11)$$

where  $\{g_{1,1}, \dots, g_{1,\ell}\} \subset \mathbb{Z}[x_1]$  are polynomials from  $A_1$  and  $s_1, \dots, s_\ell \in \{-1, 1\}$  such that

$$x \in X' \leftrightarrow s_1 g_{1,1}(x) > 0 \wedge \dots \wedge s_\ell g_{1,\ell}(x) > 0.$$

We can easily find pairs  $(s_j, g_{1,j}) \in \{-1, 1\} \times A_1$  by considering the sign of the sample point  $c$  of  $X'$  on each polynomial  $g_{1,j} \in A_1$ .

- 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. We want to find  $x_1$ -coordinates of the roots of polynomials from Equation (5.10) subject to the constraints from Equation (5.11).

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

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

where

$$P(x_1, \dots, x_\alpha) = s_1 g_{1,1}(x_1) > 0 \wedge \dots \wedge s_\ell g_{1,\ell}(x_1) > 0 \quad (5.12)$$

$$\wedge g_2(x_1, x_2) = 0 \wedge \dots \wedge g_{\alpha-1}(x_1, \dots, x_{\alpha-1}) = 0 \quad (5.13)$$

$$\wedge d_B(x_1, \dots, x_\alpha) \neq 0 \wedge d_T(x_1, \dots, x_\alpha) \neq 0 \quad (5.14)$$

$$\wedge (J'_B(x_1, \dots, x_{\alpha-1}) = 0 \vee J'_T(x_1, \dots, x_{\alpha-1}) = 0). \quad (5.15)$$

$$(5.16)$$

Computing the projection onto  $\mathbb{R}^1$  of polynomials  $\{g_2, \dots, g_{\alpha-1} J'_B, J'_T\}$  from Equations (5.13) and (5.15), a system

$$F := \{f_1 = 0, \dots, f_r = 0\} \subset \mathbb{Z}[x_1]$$

is obtained, whose real roots include the refinement points  $(c_1, \dots, c_t)$ . Let  $E \subset \mathbb{Z}[x_1]$  be the set of polynomials obtained by computing the projection onto  $\mathbb{R}^1$  of  $\{d_B, d_T\}$  (from Equation (5.14)) and

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

from Equation (5.12).  $(c_1, \dots, c_t)$  can be found by computing the real roots of  $F$  subject to the inequalities  $G$ , then discarding those such that a polynomial in  $E$  is zero.

It is clear that  $(c_1, \dots, c_t)$  lie in an open interval  $X = (a, b) \subset \mathbb{R} \cup \{-\infty, \infty\}$ . Thus, rather than using strict inequalities from Equation (5.12), we could find the roots of  $F$  in the interval  $(a, b)$ .

Each real root will be represented by a pair

$$c_i := (m_i, J_i) \quad (5.17)$$

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  $B$  contains only algebraic numbers and roots will appear in ascending order.

Since this procedure follows the construction described in ?, Theorem 3.18, the following has been proved.

**Proposition 5.3.** *Let  $\mathcal{D}$  be a sign-invariant CAD of  $\mathbb{R}^n$  compatible with each set  $V_1, \dots, V_k \subset \mathbb{R}^n$  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*

$$A = (A_1, \dots, A_n),$$

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

Then there is an algorithm, taking  $\mathcal{D}$  and  $A$  as input, which produces a set of algebraic numbers  $(c_1, \dots, c_t)$  such that  $C \cap \{c_i < x_1 < c_{i+1}\}$  is monotone. Each  $c_i$ ,  $1 \leq i \leq t$  is represented by a pair  $(m_i, J_i) \in \mathbb{Z}[x_1] \times \mathbb{Q}$  (see Equation (5.17)).

### 5.1.2 Working with sub-decompositions above a 0-cell

Note that we are working in a sub-decomposition  $D$  (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 actually in  $\mathbb{Z}[y_1, \dots, y_k][x_1, \dots, x_n]$ . From a polynomial  $f \in \mathbb{Z}[y_1, \dots, y_k][x_1, \dots, x_n]$ , a polynomial  $g := f(\mathbf{c}) \in \mathbb{A}[x_1, \dots, x_n]$  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  $h \in \mathbb{Z}[x_1, \dots, x_n]$  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  $y_1, \dots, y_k$ , there is nothing to do, just set  $h := f$ . Otherwise,  $f$  depends on at least one of  $y_1, \dots, y_k$ .
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  $h \in \mathbb{Z}[x_1, \dots, x_n]$  such that  $g(\mathbf{x}) = 0$  if and only if  $h(\mathbf{x}) = 0$  for all  $\mathbf{x} \in \mathbb{R}^n$ .

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

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

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



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

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

Each coefficient in  $g$  will be an integer number since  $M$  is divisible by all  $b_1, \dots, b_k$  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  $L[x_1, \dots, x_n]$ , viewed as polynomials in  $K[x_1, \dots, x_n, x_\alpha]$ . The norm, in  $L[x_1, \dots, x_n]$  of  $f$  can be found by computing the resultant of  $f$  and  $m$ .

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

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

where  $m_i = x_1^{d_{i,1}} \dots x_n^{d_{i,n}}$  is a monomial in variables  $x_1, \dots, x_n$  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  $\mathbb{Q}[x_1, \dots, x_n, x_\alpha]$  and  $m$  can be viewed as a polynomial in  $\mathbb{Q}[x_1, \dots, x_n, x_\alpha]$  which is independent of all  $x_1, \dots, x_n$ . Compute

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

to obtain a polynomial in  $\mathbb{Q}[x_1, \dots, x_n]$ . 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  $f, m \in K[x_1, \dots, x_n][\alpha]$ , where  $K$  is an algebraically closed field. If at least one of  $f$  and  $m$  is monic, then

$$\text{Res}_\alpha(f, m) \in I \cap R.$$

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  $\text{Res}_\alpha(f, m)$ .

## 5.2 Two-dimensional monotone sections

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

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

for some  $1 \leq i \leq r$ , is a two-dimensional monotone sector cell. Let us return to (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 (3.18), proceeds as follows. Let  $f_j : Y \rightarrow \text{span}\{x_j\}$  be a component of the quasi-affine map  $\mathbf{f} : X \rightarrow \mathbb{R}$ . 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  $x_1, x_\alpha$ . 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  $x_\alpha$ . 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 5.1.** *Let  $Y \subset \mathbb{R}^n$  be the graph of a quasi-affine map*

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

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

*Proof.* Consider a quasi-affine component  $f_j : (a, b) \rightarrow \mathbb{R}$  of the map  $\mathbf{f}$ . By definition,  $f_j$  is either strictly increasing in, strictly decreasing in, or independent of  $x_i$ . Assume for contradiction that  $Y$  is not monotone and the intersection  $Z := Y \cap \{x_i = c\}, c \in \mathbb{R}$  is not connected. Since  $Y$  itself is connected,  $Z$  would consist of two isolated points. It is clear that the projection map  $\text{proj}_{\text{span}\{x_i\}}|_Y$  is not injective and its image would be one-dimensional (if not,  $f_j$  would be constant and equal to  $c$  and  $Z$  would have one connected component).

The proof is completed by applying this argument to each component of the map  $\mathbf{f}$ .  $\square$

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 5.1 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 (3, Theorem 3), if functions

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

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

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

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 5.1 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 necessarily 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

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

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 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

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

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  $C_T \neq \emptyset$ . We must construct subsets  $W \subset Y$  such that  $f_j|_{W_T}$  is a smooth, continuous function. Let

$$A = (A_1, \dots, A_n)$$

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  $g_2, \dots, g_{j-1}$  with each  $g_i \in A_i$ ,  $2 \leq i \leq j-1$ , defined in Equation (5.4), such that  $C'_T \subset V$ . By definition

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

and there exists a polynomial  $g_j \in A_j$  such that  $g_j(\mathbf{x}) = 0$  for all  $\mathbf{x} \in C_T^j$ . Since  $C_T^j$  is either empty or the graph of a continuous function everywhere except, possibly, for 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  $C_T^j$  is one-dimensional, one way to achieve this is by computing a smooth stratification of  $C_T^j$ . 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

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

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 (?). Theoretically speaking, (? , Algorithm 14.21) provides a singly exponential upper bound for quantifier elimination.

Eliminating the  $j-1$  quantifiers, we obtain the desired points  $(b_{i,1}, \dots, b_{i,k_i})$ , where each  $b_{i,j}$  is the  $x_1$ -coordinate of a blow-up point of  $g_j$ . Now we can assume that  $C_T^j \cap \{b_{i,j} < x_1 < b_{i,j+1}\}$  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  $g_2, \dots, g_{j-1}$  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  $s_1 g_{1,1} > 0, \dots, s_1 g_{1,\ell} > 0$  (from Equation (5.12)),  $d_T \neq 0$  and  $d_B$  (from Equation (5.14)) 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

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

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  $g_j \in A_j$  is the polynomial defining  $C_T^j$  and  $g_j \in A_j$  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

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

is obtained.

The union of sets of refinement points

$$\{b_{\alpha+1,1}, \dots, b_{\alpha+1,r_{\alpha+1}}\}, \dots, \{b_{n,1}, \dots, b_{n,r_n}\}$$

forms the refinement of  $C$  into two-dimensional monotone cells.

Altogether, we have proved the following.

**Proposition 5.4.** *Let  $\mathcal{D}$  be a CAD of  $\mathbb{R}^n$  compatible with each set  $V_1, \dots, V_k \subset \mathbb{R}^n$  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*

$$A = (A_1, \dots, A_n),$$

where  $A_k \subset \mathbb{Z}[x_1, \dots, x_k], 1 \leq k \leq n$ , be the projection polynomials defining  $D$ . Let  $C$  be a 2-dimensional section cell of the decomposition induced by  $D$  on  $\mathbb{R}^j$  such that  $\text{proj}_{\mathbb{R}^j}^{-1}(C)$  is a cell of  $\mathcal{D}$  contained in  $V_1, \dots, V_k$  and  $\text{proj}_{\mathbb{R}^{j-1}}(C)$  is monotone.

Then there is an algorithm, taking  $\mathcal{D}$  and  $A$  as input, which produces a set of algebraic numbers  $(c_1, \dots, c_r)$  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 (5.17)).

propositions 5.3 and 5.4 can also be applied to two-dimensional  $(0, \dots, 0, 1, i_{k+1}, \dots, i_n)$ -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

---

**Input:**

$$(E, A)$$

- $E$ : 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  $E$  has a truth-value (true or false attached).
- $A = A_1, \dots, A_{k+n}$  is the family of projection polynomials where each  $A_i \subset \mathbb{Z}[x_1, \dots, x_i]$ .

**Output:**

$$R := \{R_{\mathbf{b}} = \{c_1, \dots, c_t\} \subset \mathbb{A} \mid \mathbf{b} \in \mathbb{R}^{k-1} \text{ is a } (0, \dots, 0)\text{-cell in the decomposition induced by } E \text{ on } \mathbb{R}^k\}$$

such that, for each  $R_{\mathbf{b}}$ , intersecting the sub-cad  $D$  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  $D$  such that each true cell of  $D$  with dimension at most 2 is monotone.

---

First, define a subroutine

$$\text{sub}_{\mathbf{y}}(g) \mid g \in \mathbb{Z}[y_1, \dots, y_m, x_1, \dots, x_n], \mathbf{y} \in \mathbb{A}^m, n > 0,$$

which computes  $h := g(\mathbf{y}) \in \mathbb{A}[x_1, \dots, x_m]$  and returns the normalised polynomial of  $h$ , in  $\mathbb{Z}[x_{m+1}, \dots, x_n]$  (see Section 5.1.2).

It will be convenient to overload this operator and write  $\text{sub}_{\mathbf{y}}(L)$ , where  $L$  is a set (or list) of polynomials to mean  $\{\text{sub}_{\mathbf{y}}(g) \mid g \in L\}$ .

Now define a structure to store refinement polynomials

$$F := \{F_{\mathbf{b}} := \emptyset \mid \mathbf{b} \in \mathbb{R}^{k-1}, k > 0 \text{ is a } (0, \dots, 0)\text{-cell in the decomposition induced by } E \text{ on } \mathbb{R}^{k-1}\}$$

and a similar structure to store refinement points

$$R := \{R_{\mathbf{b}} := \emptyset \mid \mathbf{b} \in \mathbb{R}^{k-1}, k > 0 \text{ is a } (0, \dots, 0)\text{-cell in the decomposition induced by } E \text{ on } \mathbb{R}^{k-1}\}$$

(Note: for each  $(0, \dots, 0)$ -cell  $\mathbf{b}$ ,  $F_{\mathbf{b}}$  is a set of univariate polynomials,  $R_{\mathbf{b}}$  is the set of roots of polynomials  $F_{\mathbf{b}}$ . 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  $E$ .)

Let  $\mathbf{b}$  be a  $(0, \dots, 0)$ -cell of the decomposition induced by  $E$  on  $\mathbb{R}^{k-1}$ ,  $k > 0$ . Let  $D$  be the sub-decomposition of  $\mathbb{R}^n$  of  $E$  above  $\mathbf{b}$ .

Consider each cell  $C$  of  $D$  with truth value true. Let  $C$  have index

$$(0, \dots, 0, 1, i_{k+1}, \dots, i_{k+n})$$

in  $E$ . If  $i_{k+1} + \dots + i_n > 2$ , then  $\dim(C) > 2$  and the algorithm will fail. Otherwise, let  $\alpha$  be the smallest among  $x + 1, \dots, k + n$  such that  $i_{k+1} = \dots = i_{\alpha-1} = 0$  and  $i_{\alpha} = 1$ . Then:

- Let
  - $\mathbf{b} := \text{proj}_{\mathbb{R}^{k-1}}(C)$ ,  
 (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  $\text{sub}_{\mathbf{0}}(g) \equiv g$ .)
  - $C'' \subset \mathbb{A} := \text{proj}_{\mathbb{R}^k}(C)$  ((1)-cell), and let  $(C''_B, C''_T)$  be (0)-cells which are the endpoints of  $C''$ ,
  - $G := \text{sub}_{\mathbf{b}}\{g_i \mid k + 1 \leq i \leq \alpha - 1, g_i \in A_i, g_i(\mathbf{x}) = 0 \ \forall \mathbf{x} \in \text{proj}_{\mathbb{R}^{\alpha-1}}(C)\}$ ,
  - $C' := \text{proj}_{\mathbb{R}^{\alpha}}(C)$ .
- $C'_T$  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  $C_{\delta}$  is not empty, do:
    - Let  $\text{sub}_{\mathbf{b}}(g_j) \in \mathbb{Z}[x_{k+1}, \dots, x_j]$  be such that  $g_{\alpha}(\mathbf{x}) = 0$  for all  $\mathbf{x} \in C'_{\delta}$ .
    - Compute

$$f_{\delta} := \det \begin{pmatrix} \frac{\partial g_{k+1}}{\partial x_k} & \dots & \frac{\partial g_{j-1}}{\partial x_k} & \frac{\partial g_j}{\partial x_k} \\ \vdots & & & \vdots \\ \frac{\partial g_{k+1}}{\partial x_{j-1}} & \dots & \frac{\partial g_{j-1}}{\partial x_{j-1}} & \frac{\partial g_j}{\partial x_{j-1}} \end{pmatrix},$$

from Equation (5.6), using constraints  $g_{k+1}, \dots, g_{\alpha+1}$ , and discarding denominators  $\partial g_{\alpha} / \partial x_{\alpha}$  of the total derivatives in the last column.

- Let

$$H := \{g_{k+1}, \dots, g_{j-1}, f_B, f_T\}$$

be a set of polynomials and compute

$$F_{\mathbf{b}} := F_{\mathbf{b}} \cup \text{proj}_{\mathbb{R}^1}(H) \subset \mathbb{Z}[x_k].$$

- If  $j < n$ , then

- Add  $g_j$  to  $G$ , i.e., let

$$G := G \cup \{g_j\}.$$

- Let  $j := j + 1$ .

- Let

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

$C'$  is a section cell, and its top and bottom may not be graphs of continuous functions. Use QE to compute

...

- For each  $(0, \dots, 0)$ -cell  $\mathbf{b} \in \mathbb{A}^k$ , consider

$$F_{\mathbf{b}} \in F \in \mathbb{Z}[k],$$

compute

$$R_{\mathbf{b}} := (c_1, \dots, c_t)$$

by isolating the real roots of polynomials in  $F_{\mathbf{b}}$ .

- Return the family  $R$  of refinement points.

## 5.4 Correctness and Complexity

The correctness of the algorithm from Theorem 5.1 follows from ?, Theorem 3.20 and Lemma 3.19. 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 ?, Theorem 3.20, where an  $(i_1, 0, \dots, 0, i_\alpha, 0, \dots, 0)$ -cell, where  $i_1 = i_\alpha = 1$ , is considered. By Corollary 5.1 to  $C$ , the two-dimensional subset  $X \subset \text{span}\{x_1, x_\alpha\}$  is obtained. The first part of ?, 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  $(c_1, \dots, c_t) \subset \mathbb{A}$ .  $C$  is the graph of a quasi-affine map  $\mathbf{f} = (f_{i_1}, \dots, f_{i_{n-2}})$  where each component  $f_j : X \rightarrow \text{span}\{x_j\}$  for  $j \in \{1, \dots, n\} \setminus \{1, \alpha\}$  is a quasi-affine function. The second part of ?, Lemma 3.18 describes how to



refine  $X$  such that each function  $f_j$  is monotone. By [?], 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 [?], Lemma 3.18 for  $C$  to be monotone.

### 5.4.1 Complexity

let  $C$  be a  $i_1, \dots, i_\alpha, 0, \dots, 0$  cell with dimension 2. then we compute the determinant from equation  $xx$  at most  $2(n-k)$  times ( $O(n)$ ). as discussed in section  $xx$  we get a polynomial of  $kd$ , where  $d$  is the maximum degree as we discussed in the previous section. Then we find the solutions of a zero-dimensional system of equations. we did this by computing a CAD projection. but we can get a better theoretical bound by asking how many solutions there might be. We can use the first betti number  $b_0$ , which tells us the number of connected components. gabrielov and vorobjov 6.1 gives us some bounds. we have a system of equations, just a conjunction, and their bound tells us  $d(d+1)^n$  bound on betti numbers. the result is asymptotically the same as we had before. and repeating it  $n$  times doesn't make it worse.

this process is repeated for each 2-dimensional cell contained in  $V$ . We saw that the complexity bound is also the bound on the number of cells. total cells, not 2-dimensional. but of course it's a bound on two-dimensional cells too. so we multiply the thing by itself. which is asymptotically the same??

## 5.5 Computing the refinements

We now describe how to refine the CAD  $D$  to be compatible with the refinement points in  $R$ .

For  $0 \leq k \leq n$ , consider each refinement point  $\mathbf{c} = (c_1, \dots, c_{k-1}, c_k) \in R_k$ . It is important to note that refinement points appear in ascending order. Let  $D'$  be the sub-cad of  $\mathbb{R}^{n-k}$  above  $(c_1, \dots, c_k)$ , which is a  $(0, \dots, 0)$ -cell of the CAD induced by  $D$  on  $\mathbb{R}^k$  and let  $D''$  be the CAD induced by  $D'$  on  $\mathbb{R}^{k+1}$ . We want to refine  $D''$  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  $D''$ , compute  $s := \text{sign}(b_{k+1} - c_{k+1})$ . If  $s \leq 0$ , consider the next  $(0)$ -cell in  $D''$ . Otherwise,  $s > 0$  and there is a  $(1)$ -cell  $C$  in  $D''$  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  $(j_1, \dots, j_k, j_{k+1})$ . New cells  $C_2$  and  $C_3$  should have positional indices  $(j_1, \dots, j_k, j_{k+1} + 1)$  and  $(j_1, \dots, j_k, j_{k+1} + 2)$  respectively. For each existing cell with positional index  $(l_1, \dots, l_k, l_{k+1})$  with

$l_k + 1 > j_{k+1}$ , the index should be updated to  $(l_1, \dots, l_k, l_{k+1} + 2)$ . We now need to update the sample points. The sample point of the sector 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,  $c_{k+1}$  is either an element of  $C_1$  (sample point of  $C_1$  is already correct) or  $c_{k+1}$  is an element of  $C_3$  (sample point of  $C_3$  is already correct). 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  $(a_2 + c_{k+1})/2$ . Observe that, if we have updated the sample point of a cell  $C$ , then the sample points of all cells in the sub-CAD above  $C$  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 computing part of the lifting phase recursively on the stack above  $C$ . This involves evaluating each level  $k+1$  projection factor  $f$  of 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 (? , Lemma 3.11).

## Chapter 6

# Frontier Condition

Now suppose we have a CAD  $\mathcal{D}'$  monotone with respect to each bounded semi-algebraic set  $V_1, \dots, V_k$ , 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  $\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$ . 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  $C \subset V_1 \cup \dots \cup V_k$  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  $\text{fr}(C)$  is a union of cells of  $\mathcal{D}''$ , for each cell  $C \subset V_1 \cup \dots \cup V_k$  of  $\mathcal{D}'$ .

We first present a motivating example due to ?.

**Example 6.1.** (?, Example 2.1)

Let  $f := y^2 - x^2z$  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  $\text{fr}(W)$ , which can be expressed as the disjoint union of four semialgebraic sets:

$$\begin{aligned} \text{fr}(W) = & \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, y = x, z = 1\}, \\ & \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, y = -x, z = 1\}, \\ & \{(x, y, z) \in \mathbb{R}^3 \mid x = 1, -1 < y < 1, z = y^2\}, \\ & \{(0, 0, z) \in \mathbb{R}^3 \mid 0 \leq z \leq 1\}. \end{aligned}$$

Due to the blow-up point above the origin,  $\text{fr}(W)$  contains the subset  $B := \{(0, 0, z) \in \mathbb{R}^3 \mid 0 \leq z \leq 1\}$ .

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. ? 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. ? 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  $\text{proj}_{\text{span}\{x, z\}}|_W$  is not injective. The blow-up subset  $B$ , contained in  $\text{fr}(W)$  means that  $W$  is not a topologically regular cell – a weaker property than being monotone. In the procedures described in the following sections, due to ? and ?, 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 ?, Theorem 3.20

*Given the monotone CAD  $\mathcal{D}'$  computed in the algorithm from Section 5.3.1, our final step, according to (?, Theorem 3.20) is to compute a refinement  $\mathcal{D}''$  of  $\mathcal{D}$  such that the frontier  $\text{fr}(C) = \text{cl}(C) - C$  of each 2-dimensional cell  $C \subset V_1 \cup \dots \cup V_k$  is a union of cells of  $\mathcal{D}'$ .*

*The proof of ?, Theorem 3.20 proceeds as follows. For each cell  $C \subset V$ , compute 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  $U$  of  $\text{fr}(C)$  into monotone one-dimensional curves and points, such that  $U$  is compatible with all one-dimensional cells of  $\mathcal{D}'$ . Let  $\mathbf{c} = (c_1, \dots, c_n)$  be one of the one-dimensional components of  $U$ . 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 (?, 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  $T \in U$  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 (?, Theorem 11), are 2-dimensional monotone cylindrical cells. Hence, replacing  $Z$  by  $T$  and the two components of  $Z \setminus T$  makes  $Z$  compatible with  $\text{fr}(C)$  and ensures that  $Z$  is refined into monotone cells.*

## 6.1. CONSTRUCTION OF A CAD WITH FRONTIER CONDITION, DUE TO ?, THEOREM 3.2093

Repeating this process for each 0- and 1-dimensional component of  $U$ , we obtain a refinement of  $D'$  compatible with  $\text{fr}(C)$ . This completes the construction of a cylindrical decomposition satisfying the frontier condition, compatible with and monotone with respect to, each  $V_1, \dots, V_k$ .

The most computationally difficult part of implementing this construction is finding  $\text{fr}(C)$  for each 2-dimensional cell  $C \subset V$ .

**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  $\mathbb{R}[x_1, \dots, x_n]$  having maximum degree  $d$ . There is an algorithm, taking  $F$  as input, which represents the semialgebraic set  $\text{fr}(S)$  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.*

*Proof.* Observe that  $\text{fr}(S)$  can be represented by a first-order Boolean formula

$$\text{fr}(S) = \{\mathbf{x} \in (\mathbb{R}^n \setminus S) \mid \forall \varepsilon > 0 \exists \mathbf{y} \in S (\|\mathbf{x} - \mathbf{y}\| < \varepsilon)\}.$$

Using singly-exponential quantifier elimination algorithm (?, Algorithm 14.21), we represent  $\text{fr}(S)$  as a quantifier-free Boolean formula  $F'$  with the bounds required in the lemma.  $\square$

We see that the “obvious” way of computing the frontier involves solving a quantifier elimination problem with two quantifier alternations. Using an efficient algorithm, e.g., (?, Algorithm 14.21), this procedure has complexity singly exponential in the number of variables.  $\partial C$  is represented by a quantifier-free Boolean formula  $F$ .

First, we construct the family  $U$  of one-dimensional curve intervals  $T$  and the points between them  $\mathbf{c}$ . By Lemma 5.1, 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  $\text{fr}(C)$  will be defined by  $n - 1$  polynomials. We refine  $\text{fr}(C)$  to be compatible with the critical points of  $\text{proj}_{\text{span}\{x_i\}}(D)$ ,  $1 \leq i \leq n$  defined by  $\det(J_{\{1, \dots, n\} \setminus \{i\}}) = 0$ .

A further refinement of each one-dimensional component  $T$  of  $U$  to be compatible with existing cells of  $D'$  may also be required. For each one-dimensional component  $T \in U$  and each one-dimensional cell  $D$  of  $D'$ , 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  $U$  compatible with all cells of the CAD  $D'$ .

Consider each  $T \in U$  which is a subset of a 2-dimensional cell  $Z$ . The refinement can be performed exactly as described by ?, 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  $\mathbf{c} = (c_1, \dots, c_n)$  of  $U$ . Refinements of  $D$  to be compatible with  $\mathbf{c}$  again proceed as described by ?. I.e., Consider  $C' := \text{proj}_{\mathbb{R}^k}(C)$  such that  $C'$  is zero-dimensional and equal to  $(c_1, \dots, c_k)$ , but  $\text{proj}_{\mathbb{R}^{k+1}}(C)$  is one-dimensional. Perform a refinement of the sub-cad above  $(c_1, \dots, c_k)$  by intersecting it with  $\{x_k < c_k\}, \{x_k = c_k\}, \{x_k > c_k\}$  – the refinements defined in ?, Lemma 3.11. Repeat until  $(c_1, \dots, c_n)$  is a cell in the refinement.

Let  $D''$  be the refinement of  $D'$ , compatible with  $\text{fr}(C)$  for all 2-dimensional cells  $C$  of  $D'$ . This CAD is compatible with, and monotone with respect to all  $V_1, \dots, V_k$ , and the frontier of each cell  $C \subset V_1 \cup \dots \cup V_k$  of  $D''$  is the union of some other cells of  $D''$  of smaller dimension.  $D''$  satisfies the frontier condition, so the construction is complete.

We have proved the following.

**Theorem 6.1.** Let  $\mathbf{F} \subset \mathbb{Z}[x_1, \dots, x_n]$  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  $V_1, \dots, V_k$ , 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  $C \subset V_1 \cup \dots \cup V_k$  of  $\mathcal{D}'$  satisfies the frontier condition. This algorithm has complexity...

## 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 avoid calls to a quantifier-elimination algorithm required to compute the frontiers of 2-dimensional cylindrical cells. In the following section, we will explore an alternative way to, based on an algebraic construction described by ?, obtain the frontier condition.

? 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.** (?, 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  $\text{fr}(C)$  is the closure of some  $k - 1$ -dimensional cylindrical cells.
3. A cylindrical cell  $C$  is boundary coherent if  $\text{fr}(C)$  is a union of cells.

*Remark.* (? , 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 2.26). ? call this property closure finiteness.

**Definition 6.2.** (? , Definition 2.7)

A CAD is called “strong” if every cell is boundary smooth, well-bordered and boundary coherent.

**Definition 6.3.** (? , 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  $f \in \mathbf{F} \subset \mathbb{Q}[x_1, \dots, x_n]$  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. A bad cell is a blow-up point of the polynomial  $f$  (see Definition @??def:blow-up-point)).

According to ? , Corollary 4.7, in dimension  $\leq 2$ , 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.** (? , Corollary 5.4)

Every  $\mathbf{F}$ -invariant CAD of  $\mathbb{R}^n$ ,  $n \leq 3$  is well-bordered.

Corollary 6.1 follows from ? , 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  $\mathbf{F}$ -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 ? , Corollary 4.7. ? , Proposition 5.2 depends on the following result.

**Lemma 6.2.** (*?, Proposition 5.2*)

Let  $D$  be a CAD of  $\mathbb{R}^n$  and  $D'$  the decomposition induced by  $D$  on  $\mathbb{R}^{n-1}$ . Let  $C$  be a cell of  $D$  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.

?, Theorem 5.12 describes how to pass from a well-bordered,  $\mathbf{F}$ -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  $V_1, \dots, V_k$ . 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.

?, Section 5.3 describes how to pass from an  $\mathbf{F}$ -invariant CAD of  $\mathbb{R}^3$  such that every cell is well-bordered and boundary smooth to one satisfying the frontier condition. By ?, Theorem 4.4, a cell  $C$  already satisfies the frontier condition if no points in its frontier project on a bad cell (see also ?). Thus, only cells which fail to satisfy this property need to be considered. By ?, Lemma 4.3, in  $\mathbb{R}^3$ , if a bad cell exists, then it has dimension 0.

**Lemma 6.3.** (*?, 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  $\mathbb{Q}[x_1, \dots, x_n]$  in  $\mathbb{R}^{n-1}$  has codimension at least two (see Remark after Definition 2.21).

**Lemma 6.4.** (*?, Theorem 4.4*)



## 6.2. LAZARD'S METHOD FOR DIMENSION NOT GREATER THAN 3 97

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  $D$  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  $f \in \mathbb{Q}[x_1, \dots, x_{n-1}][x_n]$ , 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  $D$  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  $D$ . 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  $\text{fr}(C_B)$  or  $\text{fr}(C_T)$ , or is a sector over a cell  $D \subset \text{fr}(C')$  having dimension less than  $d - 1$ . As  $C'$  is well-bordered,  $D$  belongs to the boundary of some cell  $E \subset \text{fr}(C')$  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  $\text{fr}(C)$  are either homeomorphic to  $C'$  and  $\text{fr}(C')$ , 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.  $\square$

**Lemma 6.5.** (*?, 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  $B := D \cap \text{fr}(C) \neq \emptyset$  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  $\emptyset \neq D \cap \text{fr}(E) \neq D$ .*

*To illustrate this, consider the example of the whitney umbrella  $W := \{x^2 - y^2z = 0\}$ . Let  $C := \{x > 0, y > 0, x^2 - y^2z = 0\} \subset W$  be one of the 2-dimensional topologically regular cells in a CAD of  $\mathbb{R}^3$  compatible with  $W$ . Then case 2 applies, with  $D = \{x = 0, y = 0\}$  and  $E = \{x > 0, y = 0, z = 0\}$ . Taking  $C = E$ , (and the same  $D$ ) we get case 1.*

*Proof.* By the proof of *?*, 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  $D$  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  $\text{fr}(C)$  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  $D$ . 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  $D$  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.

## 6.2. LAZARD'S METHOD FOR DIMENSION NOT GREATER THAN 3 99

- $(0, 0, 1)$ - or  $(0, 0, 0)$ -cells: don't 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 6.1.** (*?, Proposition 5.14*) Let  $f \in \mathbb{Q}[x_1, x_2, x_3]$  be an irreducible polynomial of the kind

$$f = g_d x_3^d + \dots + g_1 x_3 + g_0$$

with each  $g_i \in \mathbb{Q}[x_1, x_2]$ , and

$$\mathbf{p} = (p_1, p_2)$$

be a common root of  $g_d, \dots, g_1, g_0$ . 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  $(x_1, x_2) \rightarrow \mathbf{p}, (x_1, x_2) \neq \mathbf{p}$  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  $g_0, \dots, g_d$  of  $f$  is zero-dimensional and the limits of  $\{f = 0, g = 0\}$  as  $(x_1, x_2) \rightarrow \mathbf{p}$  can be computed as the zeros of  $S$ . □

The proof of *?, Proposition 5.14* depends on taking Zariski closures. There are now defined and a relevant property presented, closely following *?*.

**Proposition 6.2.** (*?, p203, part of Theorem 10*) Let  $I, J \subset K[x_1, \dots, x_n]$  be ideals, then  $\text{cl}(V(I) \cap V(J)) \subset V(I : J^\infty)$ ,

*Proof.* We claim that  $I \subset I : J \subset I : J^\infty \subset I(V(i) \setminus V(J))$ . We have  $I \subset I : J \subset I : J^2 \subset \dots \subset I : J^\infty$  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  $\text{cl}(V(i) \setminus V(J)) = V(I(V(i) \setminus V(J))) \subset V(I : J^\infty)$ .  $\square$

The proof of ?, 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  $\text{fr}(C)$  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 6.2 (?, Theorem 10),

$$\text{cl}(V(\langle f, g \rangle) \setminus V(\langle h \rangle)) \subset S.$$

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)  $\{f_1, \dots, f_k\}$  for  $S$  and solve the system of equations

$$f_1 = 0, \dots, f_k = 0, g = 0, h = 0.$$

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  $D$  be an  $\mathbf{F}$ -invariant CAD of  $\mathbb{R}^n$ ,  $n \leq 3$ . If  $n \leq 2$ ,  $D$  is already strong – well-bordered, boundary smooth and satisfies the frontier condition. In particular, by Lemma 6.3, the frontier condition is already satisfied since there are no bad cells in the decomposition induced by  $\mathcal{D}$  on  $\mathbb{R}^2$  (?, Lemma 4.3). Also by Lemma 6.3, any bad cells in the decomposition induced by  $\mathcal{D}$  on  $\mathbb{R}^2$  are 0-dimensional (?, 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.

---

## 6.2. LAZARD'S METHOD FOR DIMENSION NOT GREATER THAN 3101

(?, Algorithm 5.15)

**Input:**

$$(A, D)$$

- $A = A_1, A_2, A_3$  is the family of projection polynomials, each  $A_k \subset \mathbb{Q}[x_1, \dots, x_k]$ ,
- $D$ : is an  $A$ -invariant CAD of  $\mathbb{R}^3$  such that every cell of  $\mathcal{D}$  is a topologically regular cell.

**Output:**

$$R := \{R_{\mathbf{b}} = \{c_1, \dots, c_t\} \subset \mathbb{A} \mid \mathbf{b} \in \mathbb{R}^2 \text{ is a } (0,0)\text{-cell in the decomposition induced by } D \text{ on } \mathbb{R}^2\}$$

A set of refinement points, as described in Section 5.3.1.

- 
- Let  $R = \emptyset$ .
  - Let  $D'$  be the decomposition induced by  $D$  on  $\mathbb{R}^2$ .
  - For each  $(0,0)$ -cell  $\mathbf{c}$  of  $D'$ , determine whether it is a bad cell.

$\mathbf{b}$  is a bad cell if there exists a polynomial  $f \in A_3$  such that  $f$  has constant sign on a  $(0,0,1)$ -cell  $C$  of  $D$  such that  $\mathbf{p} = \text{proj}_{\mathbb{R}^2}(C)$ .

(Note: this is an easy check – simply look at the signs of polynomials in  $A_3$  on  $C$ . This is part of the information stored about  $C$ .)

- If  $\mathbf{b}$  is bad:
  - Let  $g_1 \in A_1, g_2 \in A_2$  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  $A' := A_1 \cup A_2$ .
  - For each  $g \in A'$ , 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

$$L := \{\text{sub}_{\mathbf{b}}(f') \in \mathbb{Q}[x_3] \mid f' \in S\}$$

and find the real roots

$$(c_1, \dots, c_t)$$

of  $L$ .

- Let

$$R := R \cup \{R_{\mathbf{b}} = (c_1, \dots, c_t)\}.$$

- return  $R$ .

Compute the refinement 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  $\mathbf{f} = (f_1, \dots, f_{n-2}) : X \rightarrow \mathbb{R}^{n-2}$  where  $X \subset \text{span}\{x_i, x_j\}, 1 \leq i < j \leq n$  in a cylindrical  $(1, 1)$ -cell by Proposition 2.5 (? , Lemma 3.3). According to the remark following Definition 2.21, each component  $f_j : X \rightarrow \text{span}\{x_j\}$  of  $\mathbf{f}$  is a continuous definable map at every point in  $\text{cl}(X)$  except for a finite set of isolated points in  $\text{fr}(X) = \text{cl}(X) \setminus X$ . Therefore, it might be natural to ask whether the result from ? 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  $V_1, \dots, V_k$  such that  $\dim(V_i) \leq 2$  for all  $1 \leq i \leq k$ , such that  $\text{cl}(C)$ , for each cell  $C \subset V_1 \cup \dots \cup V_k$  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 of dimension  $\leq 2$  is constructed to be monotone, it is already topologically regular by Proposition 2.3 (? , Theorem 1). Since each cell in a decomposition produced by the algorithm described in Section 5.3.1 is monotone with respect to each  $V_1, \dots, V_k \subset V$ , each cell  $C \subset V$  can be assumed to be topologically regular and, therefore, boundary smooth. The second requirement is that every cell must be well-bordered. While Corollary 6.1 (? , 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, ? , Example 2.11 describes 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 Lemma 6.5 (? , 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  $\text{cl}((C))$  is boundary smooth.

We first extend Corollary 6.1 ( ?, Corollary 5.4) to cells of dimension  $\leq 2$  in a sign-invariant CAD of  $\mathbb{R}^n$ . This can be done by extending ?, Theorem 5.3.

**Theorem 6.2.** 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 ?, 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  $\text{fr}(C')$  and points or intervals projecting on some 0-cells in  $\text{fr}(C')$  by Lemma 6.2 ( ?, Proposition 5.2). We need to prove that these points belong to the boundary of some of the 1-cells contained in  $\text{fr}(C)$ . Let  $\mathbf{c}$  be a 0-cell of  $\mathcal{D}'$  contained in  $\text{fr}(C')$  such that  $\mathbf{c}$  is the projection of exactly one point,  $\mathbf{p}$ , in  $\text{fr}(C)$ . Since  $C'$  is well-bordered (as  $\mathcal{D}'$  is strong), there is a 1-dimensional cell  $C'_1 \subset \text{fr}(C')$  such that  $\mathbf{c}$  is an endpoint. If  $C'_1$  is the projection of a 1-dimensional cell contained in  $\text{fr}(C)$ , we are done – the boundary of this one-dimensional cell consists of some isolating 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  $\infty$ . Therefore  $\mathbf{p}$  is either  $-\infty$  or  $\infty$ , being an endpoint of the interval  $I$  associated to  $C$  by Lemma 6.2 ( ?, Proposition 5.2). This contradicts the assumption that  $\text{proj}_{\mathbb{R}^2}^{-1}(\mathbf{c}) \cap \text{fr}(C)$  contains only  $\mathbf{p}$ .

If  $\dim(C) = 2$  and it is a sector  $((i_1, i_2, 1)$ -cell), then  $\text{fr}(C)$  consists of the two  $(i_1, i_2, 0)$ -cells  $C_B$  and  $C_T$  which project on  $C'$  the side-wall  $W = \text{fr}(C)$  ( $C_B \cup C_T$ ) 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 Lemma 6.2 ( ?, 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 ?, Theorem 4.4, we only need to consider points in  $\text{fr}(C)$  which projects on a bad cell, say  $\mathbf{b}$ , of  $\mathcal{D}'$ . By ?, 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  $B := (\text{fr}(C) \setminus (\text{cl}(C_B) \cup \text{cl}(C_T))) \cap (\mathbf{b} \times \mathbb{R})$ . By Lemma 6.2 ( ?, Proposition 5.2)  $B$  is an open interval. As  $C'$  is well-bordered, there is a 1-dimensional cell  $C'_1 \subset \text{fr}(C')$  such that  $\mathbf{b}$  is an endpoint. The set of points of  $\text{fr}(C)$  which project onto  $C'_1$  are bounded from below and above by points of  $\text{fr}(C_B)$  and  $\text{fr}(C_T)$ , thus they are 1-dimensional section cells. The boundaries of these section cells are included in  $\text{fr}(C_B)$  and  $\text{fr}(C_T)$ . Lemma 6.2 ( ?, 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.  $\square$

**Proposition 6.3.** (*Corollary to ?, Theorem 5.3*)

Let  $\mathcal{D}$  be an  $\mathbf{F}$ -invariant CAD of  $\mathbb{R}^n$  and  $C$  be a cell of  $\mathcal{D}$  of dimension  $\leq 2$  such that the decomposition induced by  $\mathcal{D}$  on  $\mathbb{R}^{n-1}$  is strong about  $\text{proj}_{\mathbb{R}^{n-1}}(C)$ . Then  $C$  is well-bordered.

*Proof.* If  $\dim(C) \leq 2$ , then it is well-bordered by ?, Lemma 5.1. If  $\dim(C) = 2$  and  $C$  is a sector cell, the same argument as in Theorem 6.2 (?, Theorem 5.3) can be applied.

Otherwise,  $\dim(C) = 2$  and  $C$  is a sector cell.  $C' := \text{proj}_{\mathbb{R}^{n-1}}(C)$  is a well-bordered, boundary smooth and boundary coherent 2-dimensional cell in  $\mathcal{D}'$ . The same argument as in Theorem 6.2 (?, Theorem 5.3) can be applied to the cells in  $\text{cl}(C') \times \mathbb{R}$ , which have dimension  $\leq 2$  or are sector cells of dimension 3.  $\square$

By Lemma 6.4, if no cells which intersect  $\text{fr}(C)$  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 Lemma 6.4 only considers cells contained in  $\text{proj}_{\mathbb{R}^{n-1}}(\text{cl}(C)) \times \mathbb{R}$  this property is also true for the weaker property that the induced decomposition is strong about  $\text{proj}_{\mathbb{R}^{n-1}}(C)$ . 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 Lemma 6.5 (?, Lemma 5.13), occurs.

**Lemma 6.6.** (*Extension of Lemma 6.5 (?, Proposition 5.13)*)

Let  $\mathcal{D}$  be a sign-invariant 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  $\text{proj}_{\mathbb{R}^{n-1}}(C)$ . Let  $D$  be a cell of  $\mathcal{D}$  such that  $B := D \cap \text{fr}(C) \neq \emptyset$  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  $\emptyset \neq D \cap \text{fr}(E) \neq D$ .

*Proof.* By the proof of (?, Theorem 4.4),  $D$  will always be a  $(0, \dots, 0, 1)$ -cell in the cylinder above a bad cell  $D'$ .

We consider each case by cell index.

- $(i_1, \dots, i_{n-1}, 1)$ -cell, where  $\sum_{j=1}^{n-1} i_j = 1$ : observe that  $C$  (if bounded) is delineated by some  $(i_1, \dots, i_{n-1}, 0)$ -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  $(0, \dots, 0, 0)$ -cell of  $D$ . Case 2 applies, taking as  $E$  the delineating  $(i_1, \dots, i_{n-1}, 0)$ -cell.



- $(i_1, \dots, i_{n-1}, 0)$ -cells, where  $\sum_{j=1}^{n-1} i_j = 1$ : consider the previous case, letting  $C = E$ .
- $(i_1, \dots, i_{n-1}, 0)$ -cell, where  $\sum_{j=1}^{n-1} i_j = 2$ :  $C$  is necessarily the root of a polynomial  $f \in \mathbb{Q}[x_1, \dots, x_n]$  which vanishes identically on  $D'$ , a  $(0, \dots, 0)$  cell in the decomposition induced by  $D$  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.
- $(0, \dots, 0, 1)$ - or  $(0, \dots, 0, 0)$ -cells: do not satisfy the hypothesis, there is nothing to do.

□

*Remark.* Let  $C$  be a  $(i_1, \dots, i_{n-1}, 0)$ -cell as described in Lemma 6.6 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  $(0, \dots, 0, 1)$ -cell and the cell  $E$ , which has dimension one, is contained in  $\text{cl}(C)$ . Hence,  $E$  must have index either  $(i_1, 0, \dots, 0)$  or  $(0, \dots, 0, i_k, 0, \dots, 0)$ . By Proposition 2.5 (? , Lemma 3.3), the images of  $C$ ,  $D$  and  $E$  under the projection map  $\text{proj}_{\text{span}\{i_1, i_k, i_n\}}$  are cylindrical cells and satisfy the three-dimensional case of Lemma 6.5 (? , Lemma 5.13).

*This implies that we should be able to use a similar technique to that described in Proposition 6.1 (? , 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 ? , Proposition 5.14*)

Let  $f \in \mathbb{Q}[x_1, \dots, x_n]$  be an irreducible polynomial of the kind

$$f = h_d x_n^d + \dots + h_1 x_n + h_0$$

with each  $g_i \in \mathbb{Q}[x_1, \dots, x_{n-1}]$ , and

$$\mathbf{p} = (p_1, \dots, p_{n-1})$$

be a common root of  $g_d, \dots, g_1, g_0$ . Let  $g_1, \dots, g_{n-1} \in \mathbb{Q}[x_1, \dots, x_{n-1}]$  be irreducible, pairwise relatively prime polynomials having  $\mathbf{p}$  as a root. Then the limits of the

common roots of  $f$  and  $g_{i_1}, \dots, g_{i_{n-2}}$  for  $\{i_1, \dots, i_{n-2}\} = \{1, \dots, n-1\}$   $\ell, 1 \leq \ell \leq n-1$  as  $(x_1, x_2) \rightarrow \mathbf{p}, (x_1, x_2) \neq \mathbf{p}$  may be computed as solutions of a zero-dimensional polynomial system.

*Proof.* Since  $f$  is irreducible, it has at least 2 coefficients,  $h_1, h_2 \in \mathbb{Q}[x_1, \dots, x_{n-1}]$ , whose GCD is a constant. Therefore, any ideal generated by these coefficients has codimension at least two and, since polynomials  $g_1, \dots, g_{n-1}$  are irreducible and pairwise relatively prime, the ideal generated by the coefficients of  $f$  and  $n-3$  of the polynomials,  $g_{i_1}, \dots, g_{i_{n-3}}$  has dimension zero. Any ideal containing these polynomials also contains another element,  $g_\ell$ , which is not a multiple of any polynomial in  $\{g_{i_1}, \dots, g_{i_{n-3}}, h_1, h_2\}$ .

The saturation by  $g_\ell$  of the ideal  $\langle f, g_{i_1}, \dots, g_{i_{n-2}} \rangle$  is defined as the ideal

$$S := \langle f, g_{i_1}, \dots, g_{i_{n-2}}, 1 - zg_\ell \rangle \cap \mathbb{Q}[x_1, \dots, x_n].$$

The irreducible components of the zero set of  $S$  are those of the zero set of  $\langle f, g_{i_1}, \dots, g_{i_{n-2}} \rangle$  at which  $g_\ell$  is not identically zero. It follows that the vertical line  $\{x_1 = p_1, \dots, x_{n-1} = p_{n-1}\}$  which is part of the zero set of  $f, g_{i_1}, \dots, g_{i_{n-2}}$  and  $g_\ell$ , 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  $S$ , the coefficients  $h_0, \dots, h_d$  of  $f$  and the polynomials  $g_{i_1}, \dots, g_{i_{n-2}}$  is zero-dimensional and the limits of  $\{f = 0, g_{i_1} = 0, \dots, g_{i_{n-2}} = 0\}$  as  $(x_1, \dots, x_{n-1}) \rightarrow \mathbf{p}$  can be computed as the solutions of  $S$ .  $\square$

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, in 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 6.6 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 ?, Corollary 4.7. For  $k = 3$ , recall that every cell is well-bordered by Corollary 6.1 (?, 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 ?, 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  $\text{proj}_{\mathbb{R}^k}^{-1}(C) \subset V_1 \cup \dots \cup V_k$ . 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 Proposition 6.3 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  $(i_1, \dots, i_k)$ -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  $\varphi : C' \rightarrow \mathbb{R}$ .  $\text{cl}(C)$  is the graph of  $\varphi$  for all but a finite number of isolated blow-up points in  $\text{cl}(C')$  (see Remark after Definition 2.21). Let  $\mathcal{A}_1, \dots, \mathcal{A}_k$  be the set of projection polynomials defining  $\mathcal{D}'$ . By the construction of CAD, these polynomials are irreducible. Let

$$C \subset \{\mathbf{x} \in \mathbb{R}^k \mid g_{j_1}(\mathbf{x}) = 0, \dots, g_{j_{n-2}}(\mathbf{x}) = 0, f(\mathbf{x}) = 0\}$$

where each  $g_i \in \mathcal{A}_i$  for  $j \in \{i_1, \dots, i_{k-1}\} \setminus \{i_{\ell_1}, i_{\ell_2}\}$  and  $f \in \mathcal{A}_k$ . Suppose that there is a  $(0, \dots, 0, 1)$ -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  $g_{j_1}, \dots, g_{j_{n-2}}$  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 ?, Algorithm 5.15 to  $C$  and  $D$ . Usince Conjecture 6.1, compute the saturations by  $g_{\ell_1}$  and  $g_{\ell_2}$ , respectively:

$$I_1 := \langle f, g_{j_1}, \dots, g_{j_{k-3}}, g_{\ell_2}, 1 - zg_{\ell_1} \rangle \cap \mathbb{Z}[x_1, \dots, x_k]$$

and

$$I_2 := \langle f, g_{j_1}, \dots, g_{j_{k-3}}, g_{\ell_1}, 1 - zg_{\ell_2} \rangle \cap \mathbb{Z}[x_1, \dots, x_k],$$

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  $\mathcal{R}_{\mathbf{b}}$  which contains the endpoints of  $D \cap \text{fr}(C)$ . 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 are endpoints of the top and bottom of  $C$ .

Finally, if  $\ell_1 > 1$  then  $C$  is a  $(0, \dots, 0, i_{\ell_1}, \dots, i_k)$ -cell. Apply the method above to the  $(i_{\ell_1+1}, \dots, i_k)$ -cell in the sub-cad above  $\text{proj}_{\mathbb{R}^{i_{\ell_1}-1}}(C)$ .

Compute a refinement of  $\mathcal{D}$  so that it is compatible with the sets of refinement points  $\mathcal{R}_{\mathbf{b}}$ . These are refinements of the kind described in ?, Lemma 3.11, and preserve the cylindrical structure and monotone cells in  $\mathcal{D}$ .  $\mathcal{D}'$  has been refined such that it satisfies the frontier condition, hence it is strong about each cell  $C$  such that  $\text{proj}_{\mathbb{R}^k}^{-1}(C) \subset V_1 \cup \dots \cup V_k$ . This completes the inductive step.

### 6.3.1 Algorithm for lifting above bad points, for CAD compatible with sets of dimension $\leq 2$

Let  $D$  be an  $\mathbf{F}$ -invariant cylindrical algebraic decomposition of  $\mathbb{R}^n$  monotone with respect to each bounded semialgebraic set  $V_1, \dots, V_k$  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  $\text{?}$ , 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  $V_1, \dots, V_k$  for  $3 \leq r \leq n$ .

---

(Extension of  $\text{?}$ , Algorithm 5.15)

**Input:**

$(D, A)$

- $D$ : a CAD of  $\mathbb{R}^k$ , each cell having constant sign on  $\mathcal{A}$ ,  $A$  collection of cells  $\mathcal{C} := \{C_1, \dots, C_m\}$  of  $\mathcal{D}$  are marked as being projections of  $V_1, \dots, V_k$ .
- $A = A_1, \dots, A_k$  is the family of projection polynomials for  $\mathcal{D}$ , each  $A_k \subset \mathbb{Z}[x_1, \dots, x_k]$ .

**Output:**

$R := \{R_{\mathbf{b}} = (c_1, \dots, c_t) \subset \mathbb{A} \mid \mathbf{b} \in \mathbb{R}^{k-1} \text{ is a } (0, \dots, 0)\text{-cell in the decomposition induced by } E \text{ on } \mathbb{R}^k\}$

A set of refinement points, as described in 5.3.1.

- 
- Let  $D'$  be the decomposition induced by  $D$  on  $\mathbb{R}^{n-1}$ .
  - For each  $(i_1, \dots, i_{k-1}, 0)$ -cell  $E \in C$ , 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  $C := \text{proj}_{\text{span}\{i_{\ell_1}, \dots, i_k\}}(E)$  be an  $(i_{\ell_1}, \dots, i_{k-1}, 0)$ -cell in the sub-cad of  $\mathcal{D}$  above  $\mathbf{c}$ . Compute the substitution of polynomials

$$\mathcal{A}'_j := \{\text{sub}_{\mathbf{c}}(f) \in \mathbb{Z}[x_1, \dots, x_j] \mid f \in \mathcal{A}_{i_1 - 1 + j}\}$$

for  $i_{\ell_1} \leq j \leq k$ .

- Find, by examining signs of projection polynomials associated to  $C$ , polynomials

$$(g_{j_1}, \dots, g_{k-3}, f), g_j \in \mathcal{A}'_j, j \in \{\ell_1 + 1, \dots, k-1\} \quad \{\ell_2\}, f \in \mathcal{A}'_k$$

such that  $C \subset \{\mathbf{x} \in \mathbb{R}^{k-\ell_1+1} \mid (g_{j_1}(\mathbf{x}) = 0, \dots, g_{k-3}(\mathbf{x}) = 0, f(\mathbf{x}) = 0)\}$ .

- Determine whether any point on the frontier of  $C$  projects on a bad cell. I.e., check if there is a  $(0, \dots, 0, 1)$ -cell  $D$ , by examining signs of projection polynomials, such that  $f(\mathbf{x}) = 0$  for all  $\mathbf{x} \in D$ .

I.e., all of  $g_{j_1}, \dots, g_{j_{k-3}}, f$  should be equal to zero on  $D$ .

- For each  $D$  projecting on a bad cell, let  $\mathbf{b} := \text{proj}_{\mathbb{R}^{k-1}}(D)$ .

Since  $\mathbf{b}$  is a  $(0, \dots, 0)$ -cell, polynomials  $g_{j_1}, \dots, g_{j_{k-3}}$  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

$$I_1 := \langle f, g_{j_1}, \dots, g_{j_{k-3}}, g_{\ell_2}, 1 - zg_{\ell_1} \rangle \cap \mathbb{Z}[x_1, \dots, x_k]$$

and

$$I_2 := \langle f, g_{j_1}, \dots, g_{j_{k-3}}, g_{\ell_1}, 1 - zg_{\ell_2} \rangle \cap \mathbb{Z}[x_1, \dots, x_k],$$

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 with *lex* order on  $x_{\ell_1}, \dots, x_n, z$ ) and discarding polynomials containing  $z$ .

- Finally, to obtain refinement polynomials for  $D$ , substitute

$$L := \{\text{sub}_{\mathbf{b}}(f) \in \mathbb{Q}[x_k] \mid f \in S_1 \cup S_2\}.$$

- Refinement points are the roots

$$(c_1, \dots, c_t)$$

of polynomials in  $L$ .

- Let

$$R := R \cup \{R_{\mathbf{b}} = (c_1, \dots, c_t)\}.$$

- return  $R$ .

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, 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  $C$ . Proposition 6.3 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 frontier 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 since its top and bottom, which are one-dimensional cells, satisfy the frontier condition.*

## Chapter 7

# A novel algorithm for the obtaining the frontier condition on a cylindrical decomposition of $\mathbb{R}^n$

*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 7.1 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 7.1, reproduced below.

**Example 7.1.** (?, 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 2.19, 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  $\varphi$  at the origin,  $C_B$  is not a cylindrical cell. Note that  $C_B$  does not contain all points at which  $\varphi$  vanishes.

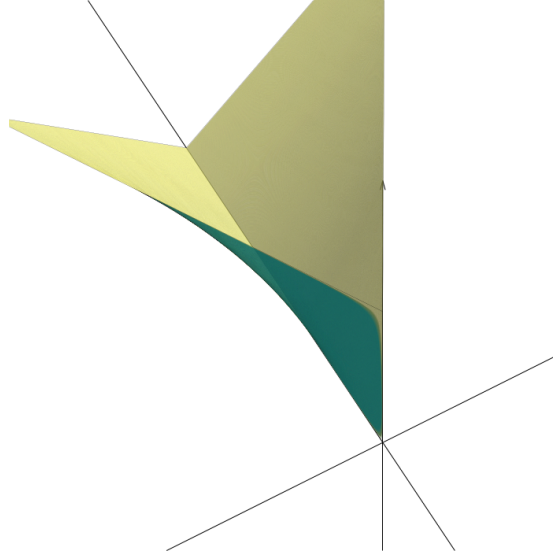


Figure 7.1:  $C$ , with  $\text{fr}(C)$  superimposed, from Example 7.1, demonstrating that  $C_B$  is not the graph of a continuous function.

As discussed above, in relation to Example 7.1, the frontier of a cylindrical cell can be geometrically nontrivial, e.g., it may contain a blow-up.



## 7.1 Statement of the Problem

Semialgebraic sets will usually be represented by Quantifier-Free Boolean formulas (QFFs) (see Definition 2.3). Occasionally, First-Order formulas, a wider class of formulas in which some variables may be bound by quantifiers  $\forall$  and  $\exists$  (see Definition 2.4).

**Theorem 7.1.** *Let  $S \subset \mathbb{R}^n$  be a semialgebraic set defined by a QFF  $F$  with  $s$  different polynomials of maximum degree  $d$  in  $\mathbb{R}[x_1, \dots, x_n]$ . 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  $(sd)^{O(1)n^2}$ . 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 7.1 is understood as a Blum-Shub-Smale (BSS) real numbers machine (?). 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. ?).

*The following result, which is somewhat natural and is also mentioned by ? will be proved. This will allow us to apply Theorem 7.1 directly to semialgebraic sets defined by first-order Boolean formulas with quantifiers.*

**Theorem 7.2.** *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  $\text{proj}_{\mathbb{R}^k}(S)$  and satisfying the frontier condition.*

*This will allow us to apply Theorem 7.1 to semialgebraic sets represented by first-order Boolean formulas (with quantifiers) and obtain a decomposition compatible with  $S$  and satisfying the frontier condition.*

## 7.2 Theory

### 7.2.1 Computing the frontier of a semialgebraic set

We present the classical CAD algorithm due to ?, as formulated in ?, Algorithm 11.2, in 2.9. Given a set of  $s$  different polynomials with maximum degree  $d$  in  $\mathbb{Z}[x_1, \dots, x_n]$ , 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  $\{F_1, \dots, F_k\}$  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 2.9 is obtained, where  $s$  is the number of different polynomials appearing in formulas  $F_1, \dots, F_k$  and  $d$  is their maximum degree. An algorithm for constructing such a CAD is presented by ?, and an alternative approach is presented by ?.

The following algorithm allows us to compute the frontier of  $S$ :  $\text{fr}(S) = \text{cl}(S) \setminus S$ . It is a straightforward application of a singly exponential quantifier elimination algorithm.

**Lemma 7.1.** *Let  $S \subset \mathbb{R}^n$  be a semialgebraic set defined by a QFF  $F$  containing  $s$  different polynomials in  $\mathbb{R}[x_1, \dots, x_n]$  having maximum degree  $d$ . There is an algorithm, taking  $F$  as input, which represents the semialgebraic set  $\text{fr}(S)$  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  $\text{fr}(S)$  can be represented by a first-order Boolean formula

$$\text{fr}(S) = \{\mathbf{x} \in (\mathbb{R}^n \setminus S) \mid \forall \varepsilon > 0 \exists \mathbf{y} \in S (\|\mathbf{x} - \mathbf{y}\| < \varepsilon)\}. \quad (7.1)$$

Using singly-exponential quantifier elimination algorithm (?, Algorithm 14.21), we represent  $\text{fr}(S)$  as a QFF  $F'$  with the bounds required in the lemma.  $\square$

## 7.2.2 Intermediate Results

Recall that, according to Definition 2.19, to each cylindrical cell in a CAD of  $\mathbb{R}^n$  a (multi-)index  $(i_1, \dots, i_n) \in \{0, 1\}^n$  is assigned. Introduce a lexicographic order  $<_{\text{lex}}$  on the set of all cell indices as follows. For any two indices,  $M := (i_1, \dots, i_n)$  and  $N := (j_1, \dots, j_n)$ , we set  $M <_{\text{lex}} N$  iff for the maximal  $k$ ,  $0 \leq k < n$ , such that  $i_1 = j_1, \dots, i_k = j_k$  we have  $i_{k+1} < j_{k+1}$ . If  $M <_{\text{lex}} N$  or  $M = N$ , we write  $M \leq_{\text{lex}} N$ .

**Lemma 7.2.** *Let  $\mathcal{D}$  be a CAD of  $\mathbb{R}^n$  and let  $C$  be a cell of  $\mathcal{D}$  with index  $M$ . Then  $\text{fr}(C)$  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  $M = (i_1, \dots, i_n)$ . The projection  $C' = \text{proj}_{\mathbb{R}^{n-1}}(C)$  is a cell in the induced decomposition  $\mathcal{D}'$  of  $\mathbb{R}^{n-1}$  with index  $(i_1, \dots, i_{n-1})$ . By the induction hypothesis,  $\text{fr}(C')$  is contained in a union of

cells of  $\mathcal{D}'$  with indices  $(j_1, \dots, j_{n-1}) <_{\text{lex}} (i_1, \dots, i_{n-1})$ . I.e., by definition we have

$$i_1 = j_1, \dots, i_{k-1} = j_{k-1} \text{ and } i_k < j_k \text{ for some } 1 \leq k \leq n-1. \quad (7.2)$$

If  $i_n = 0$ , then  $C$  is a section cell. Its frontier  $\text{fr}(C)$  is contained in  $\text{fr}(C') \times \mathbb{R}$  and, therefore, in a union of cells of  $\mathcal{D}$  with indices  $(j_1, \dots, j_{n-1}, j_n)$  for some  $j_n \in \{0, 1\}$ . Since the property in Equation (7.2) still holds, we have shown that  $(j_1, \dots, j_{n-1}, j_n) <_{\text{lex}} M$ .

If  $i_n = 1$ , then  $C$  is a sector cell. In this case,  $\text{fr}(C)$  is contained in  $(\text{fr}(C') \times \mathbb{R}) \cup C_T \cup C_B$ , hence in the union of cells of  $\mathcal{D}$  with indices  $(j_1, \dots, j_{n-1}, j_n)$  for some  $j_n \in \{0, 1\}$ , and two cells,  $C_T$  and  $C_B$ , having the same index  $(i_1, \dots, i_{n-1}, 0) <_{\text{lex}} (i_1, \dots, i_{n-1}, 1)$ . All of these indices are lexicographically less than  $M$ .  $\square$

In the proof of Theorem 7.1, we will need the following technical statement.

**Lemma 7.3.** *Let  $f : X \rightarrow \mathbb{R}$  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  $\text{fr}_F(F \setminus G) = G$ , where  $\text{fr}_F$  denotes the frontier in  $F$ .*

*Proof.* Let  $Y = \text{proj}_{\mathbb{R}^n}(G)$  and observe that  $X = \text{proj}_{\mathbb{R}^n}(F)$  and, since semialgebraic sets are closed under projection, observe that both  $X$  and  $Y$  are semialgebraic. We will prove that  $\text{fr}_X(X \setminus Y) = Y$ . By the definition of frontier, we have

$$\text{fr}_X(X \setminus Y) = \text{cl}_X(X \setminus Y) \setminus (X \setminus Y),$$

where  $\text{cl}_X$  denotes the closure in  $X$ .

We need to prove that  $\text{cl}_X(X \setminus Y) = X$ . Since  $Y$  is a semialgebraic set, it is the union of some basic semialgebraic sets

$$S_i = \{\mathbf{x} \in \mathbb{R}^n \mid f_1(\mathbf{x}) = 0, \dots, f_k(\mathbf{x}) = 0, g_1(\mathbf{x}) > 0, \dots, g_\ell(\mathbf{x}) > 0\} \quad (7.3)$$

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 (7.3). 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

$$\text{cl}_X(X \setminus Z) = X \supset Z \cap X$$

because  $(X \setminus Z) \subset (X \setminus Y)$  implies  $\text{cl}_X(X \setminus Z) \subset \text{cl}_X(X \setminus Y)$ .

When considering  $\text{cl}_X$ , we view  $X$  as a topological space and consider only the points of  $Z$  which are contained in  $X$ .

We prove that

$$Z \cap \text{cl}_X(X \setminus Z) \supset Z \cap X$$

and claim that every  $\mathbf{x} \in X \cap Z$  is a point of  $\text{cl}_X(X \setminus Z)$ . 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  $\text{fr}_F(F \setminus G) = G$ .  $\square$

*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 7.2)

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  $\text{cl}(C')$  is a union of cells of  $\mathcal{D}'$ . Consider the cylinder  $C' \times \mathbb{R}$ .  $\text{cl}(C' \times \mathbb{R})$  can be written as a first-order Boolean formula

$$(\mathbf{x}', x_n) \in \mathbb{R}^n \mid \forall \varepsilon > 0 \exists (\mathbf{y}', y_n) \in C' \times \mathbb{R} (\|(\mathbf{x}', x_n) - (\mathbf{y}', y_n)\| < \varepsilon). \quad (7.4)$$

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  $(\mathbf{x}', x_n) \in \text{cl}(C' \times \mathbb{R})$  belongs to  $\text{cl}(C)$  for some cell  $C$  of  $\mathcal{D}$  in  $C' \times \mathbb{R}$ . Hence,  $\text{cl}(C' \times \mathbb{R})$  is a union of cells of  $\mathcal{D}$ .

It is clear that, if  $(\mathbf{x}', x_n) \in \text{cl}(C' \times \mathbb{R})$ , then  $\text{proj}_{\mathbb{R}^{n-1}}((\mathbf{x}', x_n)) = \mathbf{x}' \in \text{proj}_{\mathbb{R}^{n-1}}(\text{cl}(C' \times \mathbb{R}))$ . Hence  $\text{proj}_{\mathbb{R}^{n-1}}(\text{cl}(C' \times \mathbb{R})) = \text{cl}(C')$  and, by the cylindrical property, since  $\text{cl}(C' \times \mathbb{R})$  is a union of cells of  $\mathcal{D}$ ,  $\text{cl}(C')$  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.  $\square$

## 7.3 Main Result

### 7.3.1 Algorithm Psuedo-Code

*We are now ready to prove the main result. We begin by presenting pseudo-code for the algorithm described in Theorem 7.1. First some basic subroutines are defined.*

- $\mathcal{E} := \mathcal{D} \ \& \ \{F_1, \dots, F_k\}$
- **Input:**  $\mathcal{D}$  is a CAD of  $\mathbb{R}^n$  and  $\{F_1, \dots, F_k\}$  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  $\mathbb{Z}[x_1, \dots, x_n]$ .

- **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  $\{F_1, \dots, F_k\}$  and all cells of  $\mathcal{D}$  using classical CAD algorithm (see Proposition 2.9).
- **Complexity:**  $(sd)^{O(1)^n}$ .
- $G := \text{fr}(F)$ 
  - **Input:**  $F$ , a QFF defining a semialgebraic set  $X \subset \mathbb{R}^n$ . Formula  $F$  contains  $s$  different polynomials of maximum degree  $d$  in  $\mathbb{Z}[x_1, \dots, x_n]$ .
  - **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 7.1).
  - **Complexity:**  $(sd)^{O(n^2)}$ .
- $N := \text{Decrement}(M)$ 
  - **Input:**  $M \in \{0, 1\}^k, k > 0$ .
  - **Output:**  $N \in \{0, 1\}^k$  is the index immediately prior to  $M$  with respect to  $<_{\text{lex}}$ .
  - **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 7.1.

$$\mathcal{D} := \text{CadFC}(F)$$

**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:**

$$(sd)^{O(1)^{n^2}}$$

where  $F$  contains  $s$  different polynomials of maximum degree  $d$  in  $\mathbb{Z}[x_1, \dots, x_n]$ .

- 
- Initialise  $M = (i_1, \dots, i_{n-1}) := (1, \dots, 1) \in \{0, 1\}^{n-1}$ .
  - Let  $\mathcal{E}_M := \mathbf{true} \ \& \ \{\mathbf{F}\}$ , where  $\mathbf{true}$  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  $\leq_{\text{lex}} (i_1, \dots, i_{n-1}, 1)$ .
  - While  $(0, \dots, 0) <_{\text{lex}} M$ , 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  $(i_1, \dots, i_{n-1}, 0)$
- and  $\mathcal{A}_1$  be the family of QFFs defining all cells of  $\mathcal{E}_N$  with index  $(i_1, \dots, i_{n-1}, 1)$ .
- Compute the set of frontiers

$$\mathcal{F}_{N,0} = \{\text{fr}(F_0) \mid F_0 \in \mathcal{A}_0\},$$

$$\mathcal{F}_{N,1} = \{\text{fr}(F_1 \wedge \neg(F_{1,B} \vee F_{1,T})) \mid F_1 \in \mathcal{A}_1, F_{1,B}, F_{1,T} \in \mathcal{A}_0\},$$

$$\mathcal{F}_N = \mathcal{F}_{N,0} \cup \mathcal{F}_{N,1},$$

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  $\mathcal{E}_{(0,\dots,0)}$ .

### 7.3.2 Worked Example

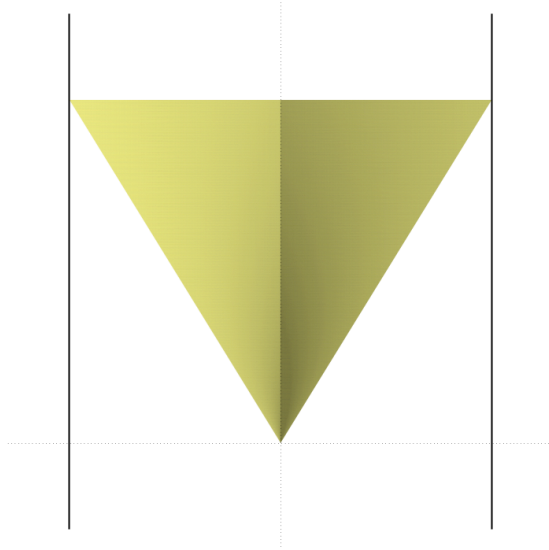


Figure 7.2: Illustrates the initial decomposition constructed in Step 1 (projection onto the  $(x,y)$ -plane).

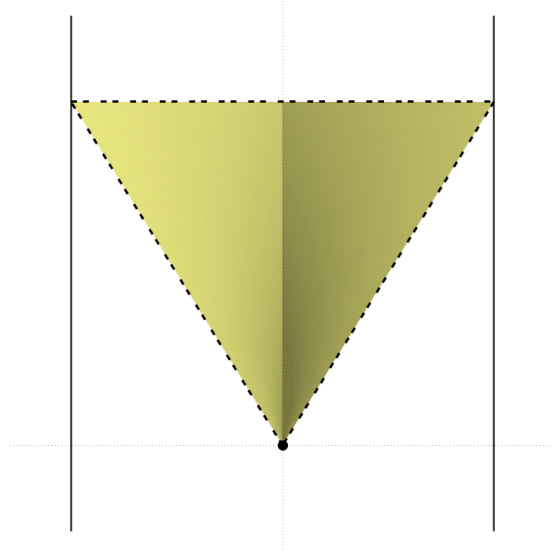


Figure 7.3: Illustrates how  $\text{fr}(S)$  interacts with the cells of  $\mathcal{E}$ . The frontier is indicated by a dashed line, with the blow-up subset above the origin represented by a node.

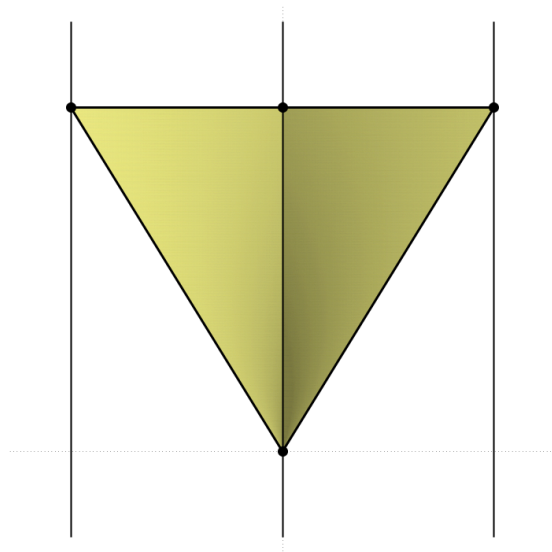


Figure 7.4: Shows cells of  $\mathcal{E}_{(0,1)}$ , the refinement of  $\mathcal{E}$  compatible with  $\text{fr}(S)$  computed in Step 4.

Table 7.1: All cells of the CAD  $\mathcal{E}$ , the initial CAD computed in Section 7.3.2.}

<i>Label</i>	<i>Index</i>	<i>Formula</i>
$C_1$	(1, 1, 1)	$\{x < -1\}$
	(0, 1, 1)	$\{x = -1\}$
$C_2$	(1, 1, 1)	$\{-1 < x < 1, y <  x \}$
	(1, 0, 1)	$\{-1 < x < 1, y =  x \}$
$S$	(1, 1, 1)	$\{-1 < x < 1,  x  < y < 1, z <  x/y \}$
	(1, 1, 0)	$\{-1 < x < 1,  x  < y < 1, z =  x/y \}$
	(1, 1, 1)	$\{-1 < x < 1,  x  < y < 1, z >  x/y \}$
$C_3$	(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)	$\{x > 1\}$

Table 7.2: Cells of the CAD  $\mathcal{E}_{(0,1)}$  computed in step ?? of Section 7.3.2. Note that only cells which are part of  $\text{cl}(S)$  are listed.

<i>Label</i>	<i>Index</i>	<i>Formula</i>
$C'_1$	(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 7.1.

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 7.1 and Figure 7.2). 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

$$\begin{aligned} \text{fr}(S) &= S_B \cup S_T \cup \{(-1, 1, 1), (1, 1, 1)\} \\ \text{where } S_T &= \{-1 < x < 1, y = 1, z = |x|\} \\ \text{and } S_B &= \{-1 < x < 0, y = -x, z = 1\} \\ &\cup \{x = y = 0, 0 \leq z \leq 1\} \\ &\cup \{0 < x < 1, y = x, z = 1\} \end{aligned}$$

(see Figure 7.3). The family  $\mathcal{F}_{(1,0)}$  contains a QFF defining  $X_{(1,0)}$ . Recall (from Example 7.1) 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  $\{x = y = 0, 0 \leq z \leq 1\}$  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  $\text{fr}(S)$  (see Table 7.2 and Figure 7.4). As we will argue in the Correctness section, frontiers  $\text{fr}_S(S')$  and  $\text{fr}_S(S''')$  coincide with  $S''$ , so no further refinements of cells with index  $(1, 1, 1)$  and  $(1, 1, 0)$  are needed.  $X_{(0,1)}$  contains  $\text{fr}(C'_2), \text{fr}(C''_2), \text{fr}(C'_3)$  and  $\text{fr}(C''_3)$ . 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  $\text{fr}(S')$ .

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* (?) and Maple's *CylindricalAlgebraicDecompose* (?) are unable to obtain the frontier condition in general. In particular, when applied to set  $S$  from Example 7.1, both  $\{\text{CylindricalAlgebraicDecompose}\}$  and  $\{\text{QEPCAD B}\}$  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  $B''' \cap \text{fr}(S) \neq \emptyset$ , but  $B''' \not\subset \text{fr}(S)$  so the frontier of  $S$  is not a union of cells in the decomposition.

TODO!! insert the tables.

### 7.3.3 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 2.9 to  $\{F\}$ .

The algorithm computes a sequence of pairs

$$(\mathcal{E}_M, \mathcal{F}_M), \quad (7.5)$$

where  $M = (i_1, \dots, i_{n-1}) \in \{0, 1\}^{n-1}$  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  $\leq_{\text{lex}} (i_1, \dots, i_{n-1}, 1)$ . We will denote this family of cells by  $\mathcal{U}_M$ . In equation (7.5), index  $M$  refers to the pair rather than its elements.

The sequence of pairs is computed recursively, starting with index  $M = (1, 1, \dots, 1)$ , in descending order of indices  $M = (i_1, \dots, i_{n-1})$  with respect to  $<_{\text{lex}}$ . Let the initial pair

$$(\mathcal{E}_{(1, \dots, 1)}, \mathcal{F}_{(1, \dots, 1)}) = (\mathcal{E}, \emptyset).$$

The algorithm will construct the sequence (from right to left):

$$(\mathcal{E}_0, \mathcal{F}_0) <_{\text{lex}} \dots <_{\text{lex}} (\mathcal{E}_N, \mathcal{F}_N) <_{\text{lex}} (\mathcal{E}_M, \mathcal{F}_M) <_{\text{lex}} \dots <_{\text{lex}} (\mathcal{E}, \emptyset), \quad (7.6)$$

where  $\mathbf{0} = (0, \dots, 0)$  and  $\mathcal{F}_0 := \{F \mid F \text{ is a QFF defining } C \in \mathcal{U}_{(0, \dots, 0)}\}$ . Note that the  $<_{\text{lex}}$  notation is naturally extended to elements of the sequence defined in Equation (7.6).

For a given index  $M = (i_1, \dots, i_{n-1})$ , assume that the algorithm has computed the pair  $(\mathcal{E}_M, \mathcal{F}_M)$ . Now we describe how the next pair,  $(\mathcal{E}_N, \mathcal{F}_N)$ , where  $N = (j_1, \dots, j_{n-1})$  is the index immediately prior to  $M$  with respect to  $<_{\text{lex}}$ , is computed. Applying algorithm from Proposition 2.9 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  $(i_1, \dots, i_{n-1}, 0)$  (section cells) and the family  $\mathcal{A}_1$  of cells in  $\mathcal{E}_N$  with indices  $(i_1, \dots, i_{n-1}, 1)$  (sector cells). Compute the set of frontiers

$$X_N := \bigcup_{C \in \mathcal{A}_0} \text{fr}(C) \cup \bigcup_{C \in \mathcal{A}_1} (\text{fr}(C) \cap (C_T \cup C_B)). \quad (7.7)$$

More precisely, apply the algorithm from Lemma 7.1 to each QFF defining a cell  $C \in \mathcal{A}_0$  and to the QFF defining  $C \cap (C_T \cup C_B)$  where  $C \in \mathcal{A}_1$ . Observe that  $C_T, C_B$  are already  $(i_1, \dots, i_{n-1}, 0)$ -cells in  $\mathcal{A}_0$ . According to Lemma 7.2,  $X_N$  is contained in the union of cells in  $\mathcal{E}_N$  with indices  $\leq_{\text{lex}} (j_1, \dots, j_{n-1}, 1) <_{\text{lex}} (i_1, \dots, i_{n-1}, 0)$  (see construction of  $X_{(1,0)}$  in Step 1 of the worked example).

When the algorithm reaches the final pair  $(\mathcal{E}_0, \mathcal{F}_0)$  in the sequence, it computes a CAD  $\mathcal{D}$ , using the algorithm from Proposition 2.9, compatible with  $\mathcal{F}_0$  and every cell of  $\mathcal{E}_0$ . The algorithm terminates by returning  $\mathcal{D}$ .

### 7.3.4 Proof of correctness

*Proof.* (of Theorem 7.1)

Let  $L = (k_1, \dots, k_{n-1})$  be the index immediately prior to  $N = (j_1, \dots, j_{n-1})$  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 (7.7). Suppose that the algorithm has computed the final decomposition  $\mathcal{D}$ , as the refinement of  $\mathcal{E}_0$  compatible with  $\mathcal{F}_0$  in the sequence 7.6. Now construct a new sequence of decompositions (from left to right)

$$\mathcal{E}'_0 <_{\text{lex}} \dots <_{\text{lex}} \mathcal{E}'_L <_{\text{lex}} \mathcal{E}'_N <_{\text{lex}} \mathcal{E}'_M <_{\text{lex}} \dots <_{\text{lex}} \mathcal{E}'_{(1, \dots, 1)} \quad (7.8)$$

where the initial element  $\mathcal{E}'_0$  is the refinement of  $\mathcal{E}_0$  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  $\leq_{\text{lex}} (i_1, \dots, i_{n-1}, 1)$  satisfies the frontier condition.

If  $C$  is a cell in  $\mathcal{E}_N$  with index  $(i_1, \dots, i_{n-1}, 0)$  (section cell), then  $C$  is a union of cells of  $\mathcal{E}_L$  and, hence, of  $\mathcal{E}'_L$ , with indices  $\leq_{\text{lex}} (i_1, \dots, i_{n-1}, 0)$ . Let  $C' \subset C$  be one of the cells in this refinement of  $C$  with index  $(i_1, \dots, i_{n-1}, 0)$  and  $B$  be the union of cells contained in the refinement of  $C$  with indices  $<_{\text{lex}} (i_1, \dots, i_{n-1}, 0)$ . Let  $B' \subset B$  be a cell of  $\mathcal{E}_L$  with index  $(m_1, \dots, m_{n-1}, 0) <_{\text{lex}} (i_1, \dots, i_{n-1}, 0)$ . 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  $m_1 = i_1, \dots, m_{\ell-1} = i_{\ell-1}$  and  $m_\ell = 0, i_\ell = 1$ . This implies that  $\dim(B') = m_1 + \dots + m_{n-1} < i_1 + \dots + i_{n-1} = \dim(C)$  as required. It follows that  $\dim(B) < \dim(C)$  and  $B$  is closed in  $C$ . According to Lemma 7.3,  $\text{fr}_C(C \setminus B) = B$  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  $<_{\text{lex}} (i_1, \dots, i_{n-1}, 0)$ .

By the induction hypothesis, all cells of  $\mathcal{E}'_L$  with index  $<_{\text{lex}} (i_1, \dots, i_{n-1}, 0)$  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  $(i_1, \dots, i_{n-1}, 1)$  (sector cell), then  $C$  is a union of cells of  $\mathcal{E}_L$  with indices  $\leq_{\text{lex}} (i_1, \dots, i_{n-1}, 1)$ . A similar argument to that used for section cells shows that each cell  $C'$  in this union, having index  $(i_1, \dots, i_{n-1}, 1)$ , satisfies the frontier condition in  $\mathcal{E}'_L$ . Note that when using Lemma 7.3 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 7.8 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  $\mathcal{E}'_{(1, \dots, 1)}$ . It follows that every cell of  $\mathcal{D}$  satisfies the frontier condition.  $\square$

### 7.3.5 Complexity

The number of different indices  $(i_1, \dots, i_n)$ , 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 (7.6). In each step, we compute the next pair  $(\mathcal{E}_N, \mathcal{F}_N)$  from  $(\mathcal{E}_M, \mathcal{F}_M)$ , using Proposition 2.9 and Lemma 7.1.

First, Proposition 2.9 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  $(i_1, \dots, i_{n-1}, i_n)$ . The complexity of applying Lemma 7.1 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

$$(s_N d_N)^{O(1)^n} = ((s_M d_M)^{O(1)^n} (s_M d_M)^{O(1)^n})^{O(1)^n}$$

Iterating this  $2^n$  times, with initial parameters  $s_{(1, \dots, 1)} = s$  and  $d_{(1, \dots, 1)} = d$  being the number of input polynomials and their maximum degree, we obtain an overall complexity of

$$(sd)^{O(1)^{n2^n}}.$$

According to Proposition 2.9 and Lemma 7.1, this is also an upper bound on the number of cells in  $\mathcal{D} = \mathcal{E}_0$ , the number of polynomials defining cells and their degrees.

### 7.3.6 A Note on Constructing the Intermediate Decompositions

In the algorithm described in Theorem 7.1, we construct, for each index  $M \in \{0, 1\}^{n-1}$ , a CAD compatible with  $\mathcal{F}_M = \{F_1, \dots, F_k\}$  and all cells of  $\mathcal{E}_M$ . This can be achieved by constructing a CAD, using Proposition 2.9, 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  $\text{cl}(S)$ , which we are not interested in.

Fewer cells may be produced by computing a CAD compatible with  $\{F_1, \dots, F_k, C_1, \dots, C_r\}$  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 7.1. We apply Proposition 2.9 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 [?]. The drawback, of course, is that more polynomials will be produced in the projection phase, which leads to more work during the lifting phase. [?] 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.

## 7.4 Generalisations and further work

### 7.4.1 Pfaffian Functions

Theorem 7.1 can be extended to restricted sub-Pfaffian sets as described by [?]. In particular, it can be used to prove the existence of decompositions with frontier condition compatible with semialgebraic sets defined by  $\{\text{fewnomials}\}$  ([?, 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 7.1 with the result of ([?, Section 5]). In the latter case, the classical CAD algorithm from Proposition 2.9 should be replaced by the main result of [?] (see also ([?, Section 7])). The rest of the proof can be reproduced almost identically. Care must be taken to ensure that Lemma 7.3 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 7.3 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 ? the oracle is needed to decide whether a sub-Pfaffian set is empty or not.

### 7.4.2 Further Work

The complexity upper bound of the algorithm from Theorem 7.1 is significantly worse than the bound for classical CAD algorithm due to ? (and ?). 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 ?. 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 7.1 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 ?.

Finally, since some initial CAD is refined repeatedly, the classical CAD algorithm from Proposition 2.9 could be replaced with an incremental algorithm, E.g., the algorithm presented by ?. 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 8

# Implementation And Testing

### 8.1 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, is reported to work very efficiently (?). Another efficient implementation of CAD is available in Mathematica. However, I opted to use `QEPCAD-B`, a quantifier elimination and cylindrical algebraic decomposition program written in C++ and built on top of the polynomial library `SACLIB`.

`QEPCAD` stand 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  $x_1 < \dots < x_n$ . 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 the cells of the CAD, including their dimension, signs of projection factors and their multiplicities and the sample point associated with the cell. `QEPCAD B` is an extended version of `QEPCAD` developed by ?. 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 polynomial factorising algorithm as well as the possibility of using tools such as Groebner bases. 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 ?, 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 (see ?). For these reasons, QEPCAD B seemed a good choice as a base program.

### 8.1.1 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*  $(v_1, \dots, v_n)$  where  $v_i, 1 \leq i \leq n$  are variable names. Note that this list specifies the variable ordering  $v_1 < \dots < v_n$ .
- *Number of free variables*  $0 \leq k \leq n$  appearing in the first-order formula
- *Prenex formula* a first order Boolean formula  $(Q_1 v_1), \dots, (Q_k v_k)F$ , 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  $\mathbb{Z}[v_1, \dots, v_n]$ ,
  - Suppose that quantifier-free Boolean formulas  $F_1, F_2, \dots, F_k$  are defined, then

$$\neg F_1, F_1 \Rightarrow F_2, F_1 \Leftarrow F_2, F_1 \Leftrightarrow F_2, F_1 \wedge \dots \wedge F_k, F_1 \vee \dots \vee F_k$$

$\$$  are quantifier-free Boolean formulas.

- $(Q_i, v_i), 1 \leq i \leq k$  where

$$Q_i \in \{\exists, \exists_k, \exists_\infty, \forall, \forall_-\}$$

quantify the first  $k$  variables in the list  $(v_1, \dots, v_n)$ . 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  $F$ ,
- $\forall_-$ : all but a finite number of values of  $v_i$  satisfy  $F$ .

#### 2. Normalisation



Let  $(Q_1, v_1), \dots, (Q_k, v_k)F$  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  $f \in \mathbb{Z}[v_1, \dots, v_n]$  is a primitive, irreducible polynomial and  $*$   $\in \{=, \neq, <, \leq, >, \geq\}$ .
- Suppose that normalised formulas  $G_1, \dots, G_k$  are already defined, then

$$G_1 \wedge \dots \wedge G_k \text{ and } G_1 \vee \dots \vee G_k$$

are normalised formulas.

- The family

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

of input polynomials is constructed, where each  $\mathcal{A}_k \subset \mathbb{Z}[v_1, \dots, v_k]$  contains the polynomials appearing in  $G$  with nonzero degree in  $v_k$  but zero degree in all variables  $v_{k+1}, \dots, v_n$ .  $\mathcal{A}_k$  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  $\mathcal{P} := (\mathcal{P}_1, \dots, \mathcal{P}_n)$  and  $\mathcal{F} := (\mathcal{F}_1, \dots, \mathcal{F}_n)$  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.

### 8.1.2 QEPCAD Cells and Sample Points

In this work, we deal mostly with QEPCAD cells rather than the QFF given as output.

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 Solution >
d-nqf-formula y^2 - x^2 + 1 = 0
```

```
Before Solution >
d-input # input polynomials
A_2,1 = input
      = y^2 - x^2 + 1
```

```

Before Solution >
d-proj-pol # projection polynomials
J_1,1= dis(A_2,1)
      = 4 x^2 - 4

```

```

Before Solution >
d-proj-fac # projection factors
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-cell(2,2) # the cell which coincides with the unit circle
----- 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

```

-----

...

A QEPCAD cell  $C$  includes the following information

- **Positional index**  $(j_1, \dots, j_{n-k}), j_i \in \mathbb{N}, 1 \leq i \leq n - k$ , 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**,  $(d_1, \dots, d_m) \in \mathbb{Z}_{\geq 0}^m$  such that  $d_i$  is the degree of the  $i$ -th level  $k$  projection factor after substitution;
- **Multiplicities**,  $((i_1, m_1), \dots, (i_\ell, m_\ell)) \in (\mathbb{N} \times \mathbb{N})^\ell$ , 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)  $(S_1, \dots, S_k)$ , where each  $S_i = (s_1, \dots, s_{m_k}) \in \{1, -1, 0\}^m$  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  $\mathbf{x} = (x_1, \dots, x_k) \in C$ . 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  $\alpha$  as a root of  $M$ ,
- $\mathbf{c} = (x_1, \dots, x_n) \in \mathbb{Q}(\alpha)^n$ .

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 a sample point  $\mathbf{x} \in \mathbb{A}^k$  consists of only rational coordinates, the minimal polynomial  $M := x$  and isolating interval  $I = (0, 0)$ .

For a sample point  $\mathbf{x} = (x_1, \dots, x_{k-1}, x_k) \in \mathbb{A}^k$  with coordinates  $(x_1, \dots, x_{k-1}) \in \mathbb{Q}(\alpha)$  (where at least one of  $x_1, \dots, x_{k-1}$  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  $\mathbf{y} = (x_1, \dots, x_{k-1}) \in \mathbb{Q}(\alpha)$  and  $x_k\beta$  and consists of five elements

$$(M_\alpha, I_\alpha, \mathbf{c}, M_\beta, J_\beta),$$

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  $\beta$  as a root of  $M_\beta$ ,

Since any finite collection of algebraic numbers belongs to an algebraic extension field  $\mathbb{Q}(\alpha)$ , it is always possible to convert a sample point in extended form to primitive form. The QEPCAD function `CONVERT(\mathbf{c}, \mathbf{k})` converts a non-primitive sample point  $\mathbf{c} \in \mathbb{A}^k$  into primitive representation by computing an appropriate  $(M_\gamma, I_\gamma)$  so that  $\mathbf{c} \in \mathbb{Q}(\gamma)$ .

### 8.1.3 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 “bad cells” to satisfy 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  $V = V_1 \cup \dots \cup V_k$  where each  $V_i, 1 \leq i \leq k$  has dimension  $\leq 2$ .

#### 2. Normalisation

Normalise the formula  $F$  and construct the set of input polynomials

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

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

Proceed as described in Section \?(sec:quasi-affine). 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  $\mathcal{D}$ . Each 1- and 2-dimensional cell of  $\mathcal{D}$  is smooth and the graph of a quasi-affine amp (see Section 4.2.1).

### 5. Refinement Phase

- *Monotone Cells*

Following the algorithm described in Section 5.3.1, construct the family of refinement points

$$R := \{R_{\mathbf{b}} = (c_1, \dots, c_t) \subset \mathbb{A} \mid \mathbf{b} \in \mathbb{R}^{k-1} \text{ is a } (0, \dots, 0)\text{-cell in the decomposition induced by } E \text{ and } c_i \in E_i\}$$

and add new polynomials (to  $\mathcal{A}$ , factorised 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 §, 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.

### 8.1.4 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.

```

1 void QepcadCIs::QUASIAFFINE(Word r, Word V, Word F, Word* A_, Word* P_, Word* J_)
2 {
3     Word C, Ps, C1s, C2s, Js = NIL, PIs, PFs, PPs;
4     SmoothOneTwoDim(r, V, F, &C1s, &C2s, &PIs, &PPs, &PFs);
5     Word PF1 = CINV(PFs); // reverse order of proj factors to match cells
6
7     // one-dimensional cells
8     // i represents the projection onto one-dimensional coordinate subspace proj(i)
9     // i.j represents proj(i,j), but we consider all minors with n-2 polynomials
10    while (C1s != NIL) {
11        ADV(C1s, &C, &C1s);
12        Word Ps = ZeroPols(PF1, r, C);
13
14        // proj(i)
15        for (int i = 1; i <= r; ++i) {
16            Word Is = GenerateIndex(r, i, 0);
17            Word J = JACOBI(r, NIL, 0, Ps, Is);
18
19            if (!IPCONST(r, J)) {
20                Js = COMP(J, Js);
21            }
22        }
23    }
24
25    // two-dimensional cells
26    // i.j represent 2-dimensional coordinate subspace proj(i,j), and all minors for proj(i)
27    while (C2s != NIL) {
28        ADV(C2s, &C, &C2s);
29        Word Ps = ZeroPols(PF1, r, C);
30
31        // proj(i,j)
32        for (int i = 1; i < r; ++i) {
33            for (int j = i + 1; j <= r; j++) {
34                Word Is = GenerateIndex(r, i, j);
35                Word J = JACOBI(r, NIL, 0, Ps, Is);
36                if (!IPCONST(r, J)) {
37                    Js = COMP(J, Js);
38                }
39            }
40        }
41    }

```

```

42     }
43
44     // CAD should be sign-invariant on Js. this means it should be compatible with the
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  $V = V_1 \cup \dots \cup V_K \subset \mathbb{R}^r$  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  $V_1, \dots, V_k$ . 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

- $r \in N$ ,
- $f \in \mathbb{Z}[x_1, \dots, x_r]$ ,
- $j \in \mathbb{Z}_{\geq 0}$
- $\text{Hs} = (h_1, \dots, h_k), h_i \in \mathbb{Z}[x_1, \dots, x_r], 1 \leq i \leq k$
- $\text{Is} = (i_1, \dots, i_k) \in \mathbb{Z}^k$ .

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, as usual.

### 8.1.5 Monotone

The function **MONOTONE** is responsible for computing the set of refinement points necessary to construct a CAD monotone with respect to  $V_1, \dots, V_k$ .

- **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.

```

1 Word QepcadCls::MONOTONE(Word* A_, Word* J_, Word D, Word r)
2 {
3     // to store unfactorised refinement polynomials, in the form (Index, Sample, Ps)
4     Word Rs = NIL;
5
6     // consider each true cell in D
7     // note: in practice, it is slower and takes more memory to do a walk of the CAD,
8     Word TrueCells, junk;
9     LISTOFCWTV(D, &TrueCells, &junk);
10    Word Ds = LELTI(D, CHILD); // cells of D, for searching cells
11    Word AA = INV(LCOPY(*A_)); // in reverse order, to match SIGNPF. for later.
12
13    while (TrueCells != NIL) {
14        Word C;
15        ADV(TrueCells, &C, &TrueCells);
16
17        // C has dimension two, then IJ < Ik are the positions in I where the componen
18        Word I = LELTI(C, INDX);
19        Word Ij, Ik;
20        Word d = TwoDimIndex(I, &Ij, &Ik);
21        if (d > 2) {
22            // fail: only implemented for dimension at most 2
23            FAIL("source/ticad/MONOTONE", "cell dimension greater than 2 not supported
24        } else if (d < 2) {
25            // cells with dimension 0 and 1 are already monotone.
26            continue;
27        }
28
29        Word Ij1 = Ij - 1;
30        Word nv = r - Ij1;
31
32        // CO := proj_{j-1}(C) is a 0-dimensional cell (c_1,...,c_{j-1})
33        Word CO, SO, IO;
34        Word ij = LELTI(I, Ij);
35        if (Ij == 1) { // base CAD is D
36            CO = D;
37            SO = LIST3(PMON(1,1), LIST2(0,0), NIL);
38            IO = NIL;
39        } else { // sub-CAD on top of some 0-cell
40            CO = FindByIndex(Ds, I, Ij1, 1);
41            SO = LELTI(CO, SAMPLE);
42            IO = LELTI(CO, INDX);
43        }
44
45        // top and bottom of proj_k(C) are one-dimensional sections by definition.

```

```

46      Word I1 = LCOPY(I);
47
48      // top: (i_1, ..., i_k + 1)
49      SLELTI(I1, Ik, LELTI(I, Ik) + 1);
50      Word CT = FindByIndex(Ds, I1, Ik, 1);
51
52      // bottom: (i_1, ..., i_k - 1)
53      SLELTI(I1, Ik, LELTI(I, Ik) - 1);
54      Word CB = FindByIndex(Ds, I1, Ik, 1);
55
56      // Top and bottom of 1-sector, level J
57      Word COC = LELTI(CO, CHILD);
58      Word IO1 = CCONC(IO, LIST1(ij));
59      Word SOB = NIL, SOT = NIL, Endpoints;
60      if (ij > 1) {
61          GETSAMPLEK(
62              Ij,
63              LELTI(LELTI(COC, ij - 1), SAMPLE),
64              &junk,
65              &SOB
66          );
67
68          SOB = SECOND(SOB);
69      }
70
71      if (ij < LENGTH(COC)) {
72          GETSAMPLEK(
73              Ij,
74              LELTI(LELTI(COC, ij + 1), SAMPLE),
75              &junk,
76              &SOT
77          );
78
79          SOT = SECOND(SOT);
80      }
81
82      Endpoints = LIST2(SOB, SOT);
83
84      // find polynomials in sub-CAD
85      // (note they will be in  $Z[x_i, \dots, x_l]$  after substitution):
86      //  $G_s = g_2, \dots, g_{\{k-1\}}$  define  $\text{proj}_{\{k-1\}}(C)$ .
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, SO, Ij1, nv);
89
90      //  $F_k(f_{\{k,T\}}, f_{\{k,B\}})$  are 0 on CT and CB respectively.

```

```

91     Word FT, FB;
92     if (CT != NIL) FT = FIRST(ZeroPolSub(AA, r, CT, Ik, Ik, S0, Ij1, nv));
93     if (CB != NIL) FB = FIRST(ZeroPolSub(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
96     Word Fs = ZeroPolSub(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  $(i_1, \dots, i_r)$ -cell such that  $i_j = i_k = 1$  and all other elements of the index are equal to zero. - On line 32,  $c_0 = \text{proj}_{\mathbb{R}^{i_j-1}}(C)$ , the zero-cell above which  $C$  is a  $(1, i_{j+1}, \dots, i_r)$ -cell. -  $\text{proj}_{\mathbb{R}^k}(C)$  is a section cell, on lines 50 and 54,  $\text{proj}_{\mathbb{R}^k}(C)_T$  and  $5(C)_{B4}$ , 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 index. - 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

$$Gs := \{g \in \mathbb{Q}[x_j, \dots, x] \mid g(\mathbf{x}) = 0 \forall \mathbf{x} \in C, j < k\}.$$

- Lines 92 and 93 find and substitute polynomials  $f_T$  and  $f_B$  in  $\mathbb{Q}[x_j, \dots, x_k]$  respectively such that  $f_T(\mathbf{x}) = 0 \forall \mathbf{x} \in \text{proj}_{\mathbb{R}^k}(C)_T$  and  $f_B(\mathbf{x}) = 0 \forall \mathbf{x} \in \text{proj}_{\mathbb{R}^k}(C)_T$
- Line 96 finds and substitutes polynomials

$$Fs := \{f \in \mathbb{Q}[x_j, \dots, x] \mid f(\mathbf{x}) = 0 \forall \mathbf{x} \in C, k < r\}.$$

- 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:**

- $G_s, P, F_s$  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);
62 }

```

**LagrangeRefinement** is applied to  $P$  and then to each polynomial in  $F_s$ . On each step, the Jacobi determinant  $Q$ , defined in Equation ??eq:jacobi-det), 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  $G_s$  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 8.1.7.

### 8.1.6 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  $A_s = (\mathcal{A}_k, \dots, \mathcal{A}_r)$ : the projection factor set,

**Output:** - Word  $A$ , Word  $J$ , Word  $RP_s$ : refinement points for bad cells contained

in the sub-CAD above D are added.

```

1 Word QepcadCls::FRONTIER(Word r, Word k, Word D, Word As, Word* A_, Word* J_, Word* RPs_)
2 {
3     Word Ch, TrueCells, junk, C1, C1_B, C1_T, C;
4     Ch = LELTI(D, CHILD);
5
6     // if r < 3, frontier condition is obtained automatically, if no children then nothing to do.
7     if (r < 3 || Ch == NIL) return D;
8
9     ADV(Ch, &C1, &Ch);
10
11     // only one sector, bad cells are not possible.
12     if (Ch != NIL);
13
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  $(j, i_1, \dots, i_{k+1})$ , where  $(i_1, \dots, i_{k+1})$  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  $(0, \dots, 0, m_k, 0, \dots, 0, m_\ell, 0, \dots, 0)$  in the CAD of  $\mathbb{R}^r$  (ambient space) where  $m_k = m_\ell = 1$ .
- **SI** (Line 31) is a list  $(i_{k+1}, \dots, i_r)$  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  $\text{fr}((\text{proj}_{\mathbb{R}^\ell}(C)))$ , may lie in the sub-CADs above the  $(0, \dots, 0)$ -cells  $\text{proj}_{\mathbb{R}^k}(C)_B$  and  $\text{proj}_{\mathbb{R}^k}(C)_T$ . The function **ProcessBadCells** (called on Lines 34 and 35) identifies bad cells and adds refinement points.

#### Input:

- **Word**  $r \in \mathbb{N}$ ,
- **Word**  $C$ : 0-dimensional level- $k$  QEPCAD cell,
- **Word**  $As = (\mathcal{A}_1, \dots, \mathcal{A}_r)$  is the (entire) projection factor set.
- **Word**  $i$ , **Word**  $j \in \mathbb{N}$ : elements of  $(i_1, \dots, i_r)$ , the index of a two-dimensional section cell  $C$ , which are equal to 1.
- **Word**  $S = (j_1, \dots, j_r)$ , indices of projection factors which are equal to zero on  $C$ .

#### Output:

- **RefinedCells**: a note of all cells which require refinement,
- **Word**  $A$ , **Word**  $J$ , **Word**  $RP$ s: 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)
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
45
46     // a bad cell is a (0,...,0,1)-cell of level greater than J, with matching sign which has
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 looks proceeds by induction, recursing on zero-dimensional cells of above  $\mathbf{C}$  whose signature (indices of projection factors which are equal to zero) matches  $\mathbf{S}$ . Note that coordinates  $i$  and  $j$  of  $\mathbf{S}$  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  $(0, \dots, 0, 1)$ -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 two saturation of ideals

$$I_1 := \langle f, g_{j_1}, \dots, g_{j_{k-3}}, g_{\ell_2}, 1 - zg_{\ell_1} \rangle \cap \mathbb{Z}[x_1, \dots, x_k]$$

and

$$I_2 := \langle f, g_{j_1}, \dots, g_{j_{k-3}}, g_{\ell_1}, 1 - zg_{\ell_2} \rangle \cap \mathbb{Z}[x_1, \dots, x_k],$$

as described in Section 6.3. To obtain a refinement polynomial in  $\mathbb{Z}[x_r]$ , the sample point of the bad  $(0, \dots, 0)$ -cell is substituted. **ADDREFINEMENTPOINTS** is then called, taking these polynomials as input, the add the refinement points above this bad cell.

### 8.1.7 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

$$R := \{R_{\mathbf{b}} = \{c_1, \dots, c_t\} \subset \mathbb{A} \mid \mathbf{b} \in \mathbb{R}^{k-1} \text{ is a } (0, \dots, 0)\text{-cell in the decomposition induced by } D \text{ on } \mathbb{R}^k\}.$$

The function **REFINE()** computes refinements of  $\mathcal{D}$  to be compatible with these points. Each refinement point  $(b_1, \dots, b_{k-1}, c) \in \mathbb{A}^k$ , such that  $(b_1, \dots, b_k)$  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

$$(b_1, \dots, b_{k-1}, c), (m_1, \dots, m_{k-1}, m_k)$$

where  $(b_1, \dots, b_{k-1})$  is the sample point, and  $(m_1, \dots, m_{k-1})$  is the positional index of the 0-cell  $\mathbf{b}$  and  $c$  refines the (1)-cell in the sub-CAD of  $\mathcal{D}$  above  $b$  with positional index  $(m_k)$ .  $(b_1, \dots, b_{k-1}, c)$  is stored as a **QEPCAD** sample point. If all of  $(b_1, \dots, b_{k-1})$  are rational, or are elements of the same algebraic extension,  $\mathbb{Q}(\alpha)$ , associated with the sample point of the cell  $\mathbf{b}$ , as  $c$ , then  $(b_1, \dots, b_{k-1}, c)$  is a primitive **QEPCAD** sample point. If  $(b_1, \dots, b_{k-1}) \in \mathbb{Q}(\alpha)$  and

$c \notin \mathbb{Q}(\alpha)$  then  $(c_1, \dots, b_{k-1}, c)$  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` =  $(\mathcal{A}_k, \dots, \mathcal{A}_n)$ : set of refinement points,
- Word `PF` =  $(\mathcal{F}_k, \dots, \mathcal{F}_n)$ : 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  $\mathbf{0}$  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. This means that, if  $C$  is an  $(i_1, \dots, i_k)$ -cell, then  $\text{proj}_{\mathbb{R}^k}^{-1}(C)$  is an  $(i_1, \dots, i_k, 1, \dots, 1)$ -cell of  $\mathcal{D}$ .

Refine proceeds by induction on CAD cells by “walking” the CAD (similar to 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 = LETI(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 = LETI(D, IND);
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, s
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 = REFINES(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 a refinement point has been 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  $(\mathcal{A}_{k+1}, \dots, \mathcal{A}_n)$ , 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.

```

1 Word RefineSubcad(Word k, Word Ch, Word Ps, Word PFs)
2 {
3     Word i1, Ch1, C, i, c, C0;
4     i1 = LELTI(LELTI(FIRST(Ch), INDX), k);
5
6     while (Ps != NIL) {
7         Word PM, PI, J;
8         NextPolynomial(Ps, &PM, &PI, &J, &i, &Ps);
9
10        // find cell with index i.
11        Word j = 0, SOM = NIL, SOI = NIL;
12        Ch1 = Ch, i = i - 1; // we are actually looking fro sector bottom
13        while (i > 0 || Ch1 != NIL) {
14            ADV(Ch1, &C, &Ch1);
15
16            // original cell indices are preserved until the last moment.
17            // we cannot just count in case a cell was refined.
18            j = LELTI(LELTI(C, INDX), k);
19
20            if (j == i) {
21                // C is the cell bottom. get its sample k
22                GETSAMPLEK(-1, LELTI(C, SAMPLE), &SOM, &SOI);
23
24                break;
25            }
26        }
27
28        // first cell in Ch is to be refined, S1M, S1J is the k-th coordinate of the sample point
29        bool refine_after = false; // do we need to refine C3?
30        Ch1 = RefineCell(k, Ch1, PM, PI, SOM, SOI, PFs, &refine_after);
31
32        ADV(RED2(Ch1), &C, &Ch1);
33        // 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(FIRST(PI));
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  $\mathbf{b} = (b_1, \dots, b_{k-1}, c)$  is a QEPCAD sample point such that PI is its isolating interval, PM the minimal polynomial for  $c$  (or  $(c, c)$  if  $c$  is rational) and J is the isolating interval containing  $c$ .  $j$  is the index of the child cell containing  $c$ .
- Suppose that  $C$  is the child with index  $i$ . 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,  $-1$  means the last coordinate.
- `RefineCell`, called on line 32, takes a list of cells,  $(C, C_T, \dots, C_\ell)$ , where  $C$  is the cell to be refined and returns a list  $(C_1, C_2, C_3, C_T, \dots, C_\ell)$ . 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, have

been added. 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 points to the correct cell. 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  $(C, C_T, \dots)$ , 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  $PFs = (\mathcal{F}_k, \dots, \mathcal{F}_n)$  is the set of projection factors, needed to update sample points of children.

**Output:**

- Word  $Ch'$ : refined list of children  $(C_1, C_2, C_3, C_T, \dots)$ , 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 a
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.

### 8.1.8 Obtaining defining formulas for every cell

As discuss  $FD = FRONT$  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  $C \subset V_1, \dots, V_k$  by getting its defining formula. As such, a new command, **d-cell-tarski** has been implemented. This uses Brown's solution formula construction algorithm. 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 basic semialgebraic set using only polynomials from the projection factor set. I.e., every cell  $C$  of  $\mathcal{D}$  can be expressed as a quantifier-free Boolean formula

$$F_1 \vee \dots \vee F_k$$

where  $F_1, \dots, F_k$  are conjunctions of polynomial equations and inequalities defining the cells  $C_1, \dots, C_k$  of  $\mathcal{D}_R$  such that  $C = C_1, \dots, C_k$ . 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 a the QFF

$$(g < 0 \wedge x > 0 \wedge f = 0 \wedge y < 0) \vee (x = 0 \wedge f = 0 \wedge y < 0) \vee (g < 0 \wedge x < 0 \wedge f = 0 \wedge y < 0)$$

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.

### 8.1.9 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, GVLV, 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 ne
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 be
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, GVNFBV, CONC(LIST10(0,0,0,1,0,3,2,4,1,5), LIST1(-1)));
98      else
99          SFCFULLD(GVPC, GVPF, GVPJ, GVNFBV);
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.

### 8.1.10 Additional modifications and Utility Functions

A brief summary of new functions follows.

The following files in the QEPCAD codebase have been modified.

## 8.2 Test Cases

**Example 8.1.** Let  $n = 3$  and consider the unit sphere

$$S := \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 - 1 = 0\}.$$

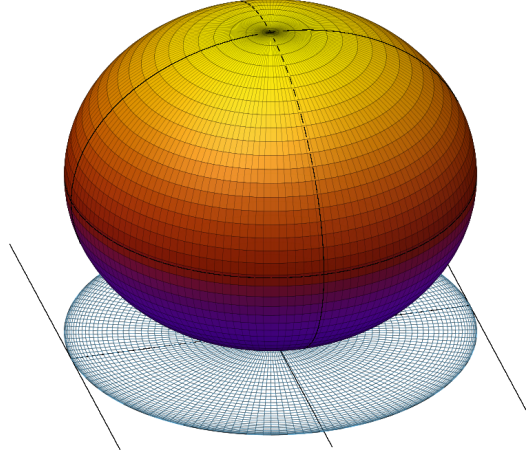


Figure 8.1: Plot of the sphere from Example `refexm:sphere`, showing critical points of projections onto one and two-dimensional coordinate subspaces, and their projections.

- **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.

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<sub>1,1</sub> = fac(J<sub>1,1</sub>) = fac(dis(P<sub>2,1</sub>))  
= x + 1

P<sub>1,2</sub> = fac(J<sub>1,1</sub>) = fac(dis(P<sub>2,1</sub>))  
= x - 1

Q<sub>1,3</sub> = fac(Q<sub>3,2</sub>) = fac(input)  
= x

P<sub>2,1</sub> = fac(J<sub>2,1</sub>) = fac(dis(A<sub>3,1</sub>))  
= y<sup>2</sup> + x<sup>2</sup> - 1

Q<sub>2,2</sub> = fac(Q<sub>3,1</sub>) = fac(input)  
= y

A<sub>3,1</sub> = input  
= z<sup>2</sup> + y<sup>2</sup> + x<sup>2</sup> - 1

Q<sub>3,2</sub> = fac(Q<sub>3,1</sub>) = 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  $4x^2 - 3$  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  $16x^2 - 11$  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  $16x^2 - 11$  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  $4x^2 - 3$  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  $16x^2 - 11$  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  $16x^2 - 11$  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  $4x^2 - 3$  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

$$\begin{aligned} z^2 + y^2 + x^2 - 1 &= 0 \\ z &> 0 \end{aligned}$$

Signs of projection factors (for defining formula) -

Index in RCAD: (4,5,6)

Level 1

$$\begin{aligned} x + 1 &> 0 \\ x - 1 &< 0 \\ x &= 0 \end{aligned}$$

Level 2

$$\begin{aligned} y^2 + x^2 - 1 &< 0 \\ y &> 0 \end{aligned}$$

Level 3

$$\begin{aligned} z^2 + y^2 + x^2 - 1 &= 0 \\ z &> 0 \end{aligned}$$

Sample point -----

The sample point is in a PRIMITIVE representation.

alpha = the unique root of  $16x^2 - 15$  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

$$\begin{aligned} x + 1 &> 0 \\ x - 1 &< 0 \\ x &= 0 \end{aligned}$$

Level 2

$$\begin{aligned} y^2 + x^2 - 1 &< 0 \\ y &> 0 \end{aligned}$$

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  $16x^2 - 15$  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  $4x^2 - 3$  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  $4x^2 - 3$  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  $4x^2 - 3$  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  $16x^2 - 11$  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  $16x^2 - 11$  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  $4x^2 - 3$  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  $4x^2 - 3$  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  $16x^2 - 11$  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  $16x^2 - 11$  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  $4x^2 - 3$  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

$$\begin{aligned} z^2 + y^2 + x^2 - 1 &= 0 \\ z &= 0 \end{aligned}$$

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 8.2, we present a 2-dimensional section cell which is the graph of a quasi-affine map, but is not a monotone cell.

**Example 8.2.** Let  $n = 3$  and consider the 2-dimensional section cell

$$C := \{(x, y, z) \in \mathbb{R}^3 \mid 0 < x < 1, y > 0, x + y^2 < 1, z = x^2 + y^2\}.$$

- **Quasi-affine:**

$C$  is already the graph of a quasi-affine map.

- **Monotone:**

- The  $(1, 1)$ -cell  $C' := \text{proj}_{\mathbb{R}^2}(C)$  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

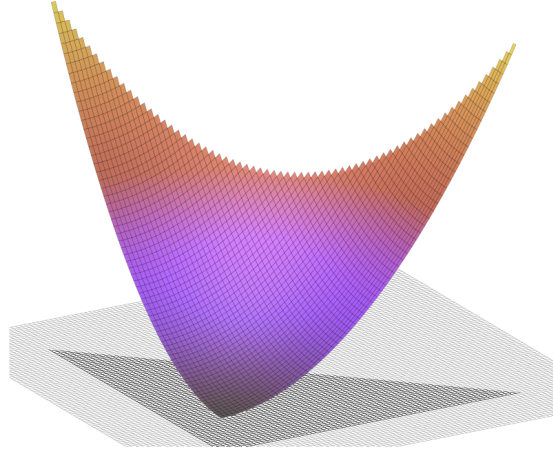


Figure 8.2: Plot of the 2-dimensional cell from Exemplerefxm:qanm.

at which

$$\det \begin{pmatrix} \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \\ \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{pmatrix} = \det \begin{pmatrix} 1 & 1 \\ 2x & 2y \end{pmatrix} = 2x - 2y = x - y.$$

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.

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

$y > 0$

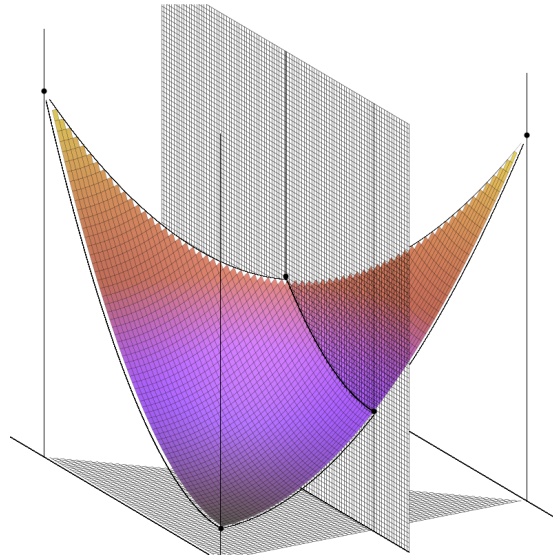


Figure 8.3: Plot of the CAD with monotone cells, compatible with the 2-dimensional cell presented in Example `refexm:qanm`.

$$y + x - 1 < 0$$

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<sub>1,1</sub> = input  
       = x

A<sub>1,2</sub> = fac(J<sub>1,1</sub>) = fac(res(A<sub>2,1</sub>|A<sub>2,2</sub>))  
       = x - 1

M<sub>1,3</sub> = fac(K<sub>1,3</sub>) = fac(input)  
       = 2 x - 1

A<sub>2,1</sub> = input  
       = y

A<sub>2,2</sub> = input  
       = y + x - 1

A<sub>3,1</sub> = input  
       = z - y<sup>2</sup> - x<sup>2</sup>

Before Solution >

d-ref

M<sub>1,1</sub> = 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

$x > 0$

$x - 1 < 0$

$2x - 1 > 0$

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

$x > 0$

$x - 1 < 0$

$2x - 1 > 0$

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  
 $x > 0$   
 $x - 1 < 0$   
 $2x - 1 = 0$   
 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  
 $x > 0$   
 $x - 1 < 0$   
 $2x - 1 = 0$   
 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 >

We now present a few variations of this example, to test various situations. The following Example 8.3 requires an algebraic refinement point.

**Example 8.3.** 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 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 (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 8.4, which follows, demonstrates what happens on sub-CADs above 0-dimensional cells.

**Example 8.4.** 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).

$$\mathbf{b}_1 = 7/4, \quad C_1 = \{(x, y, z) \in \mathbb{R}^3 \mid x > 0, x < 1, y > 0, x + y < 1, z = 7/4x^2 + y^2\}. \quad (8.1)$$

$$\mathbf{b}_2 = -6, \quad C_3 = \{(x, y, z) \in \mathbb{R}^3 \mid x > 0, x < 1, y > 0, x + y < 1, z = -6x^2 + y^2\}. \quad (8.2)$$

$$\mathbf{b}_3 = 6, \quad C_3 = \{(x, y, z) \in \mathbb{R}^3 \mid x > 0, x < 1, y > 0, x + y < 1, z = 6x^2 + y^2\}. \quad (8.3)$$

$C_2$  is already monotone. This can be seen by examining the projections of its top and bottom onto  $\text{span}\{x, z\}$ :

$$C_{2,B} = \{-1 < x < 1, z = -6x^2\}, \quad (8.4)$$

$$C_{2,T} = \{-1 < x < 1, z = -5x - 2x + 1\}. \quad (8.5)$$

$C_1$  and  $C_3$  are of the same kind as the cell presented in Example 8.2 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<sup>2</sup> - w x<sup>2</sup> = 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<sup>2</sup> - w x<sup>2</sup> = 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<sup>2</sup> - w x<sup>2</sup> = 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 8.5.** 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  $8x - 7$  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

$$x > 0$$

$$x - 1 < 0$$

$$5x - 2 > 0$$

Level 3

$$y > 0$$

$$y + x - 1 < 0$$

Level 4

$$z - 2y^2 - 3x^2 = 0$$

Signs of projection factors (for defining formula) -

Index in RCAD: (6,5,3,2)

Level 1

$$w > 0$$

$$w^2 - 2 = 0$$

Level 2

$$x > 0$$

$$x - 1 < 0$$

$$5x - 2 > 0$$

Level 3

$$y > 0$$

$$y + x - 1 < 0$$

Level 4

$$z - 2y^2 - 3x^2 = 0$$

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  $8x - 7$  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  
      $x > 0$   
      $x - 1 < 0$   
      $5x - 2 < 0$   
 Level 3  
      $y > 0$   
      $y + x - 1 < 0$   
 Level 4  
      $z - 2y^2 - 3x^2 = 0$

Signs of projection factors (for defining formula) -

Index in RCAD: (6,3,3,2)  
 Level 1  
      $w > 0$   
      $w^2 - 2 = 0$   
 Level 2  
      $x > 0$   
      $x - 1 < 0$   
      $5x - 2 < 0$   
 Level 3  
      $y > 0$   
      $y + x - 1 < 0$   
 Level 4  
      $z - 2y^2 - 3x^2 = 0$

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 8.6.** 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

$y - x^2 + 2x - 1 < 0$

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

$$y > 0$$

$$y - x^2 + 2x - 1 < 0$$

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  $64x - 17$  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

$$x > 0$$

$$x - 1 < 0$$

$$2x^3 - 6x^2 + 7x - 2 > 0$$

$$8x - 9 < 0$$

Level 3

$$y > 0$$

$$y - x^2 + 2x - 1 < 0$$

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

$$x > 0$$

$$x - 1 < 0$$

$$2x^3 - 6x^2 + 7x - 2 > 0$$

$$8x - 9 < 0$$

Level 3

$$y > 0$$

$$y - x^2 + 2x - 1 < 0$$

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  $64x - 17$  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  
      $x > 0$   
      $x - 1 < 0$   
      $2x^3 - 6x^2 + 7x - 2 = 0$   
      $8x - 9 < 0$   
 Level 3  
      $y > 0$   
      $y - x^2 + 2x - 1 < 0$   
 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  
      $x > 0$   
      $x - 1 < 0$   
      $2x^3 - 6x^2 + 7x - 2 = 0$   
      $8x - 9 < 0$   
 Level 3  
      $y > 0$   
      $y - x^2 + 2x - 1 < 0$   
 Level 4

$$z - y^2 - x^2 = 0$$

Sample point -----

The sample point is in an EXTENDED representation.

alpha = the unique root of  $4x^6 - 24x^5 + 28x^4 + 52x^3 - 35x^2 - 64x - 107$  between  
= 2.1422962953-

Coordinate 1 =  $300/8777 \alpha^5 - 1518/8777 \alpha^4 + 322/8777 \alpha^3 + 5607/8777 \alpha^2$   
= 1.7320508076-

Coordinate 2 =  $-300/8777 \alpha^5 + 1518/8777 \alpha^4 - 322/8777 \alpha^3 - 5607/8777 \alpha^2$   
= 0.4102454877-

Coordinate 3 =  $1/8$   
= 0.1250000000

Coordinate 4 = the unique root of  $262144x^3 - 536576x^2 + 1654976x - 287873$  between  
= the unique root of  $262144x^3 - 536576x^2 + 1654976x - 287873$  between  
= 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  
   $x > 0$   
   $x - 1 < 0$   
   $2x^3 - 6x^2 + 7x - 2 < 0$   
   $8x - 9 < 0$   
Level 3  
   $y > 0$   
   $y - x^2 + 2x - 1 < 0$   
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 8.7.** 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

$$\begin{aligned} w &> 0 \\ w - 1 &< 0 \\ 3w - 1 &= 0 \\ 2w - 1 &< 0 \end{aligned}$$

Level 2

$$\begin{aligned} x &> 0 \\ x + w - 1 &< 0 \end{aligned}$$

Level 3

$$y - x^2 - w^2 = 0$$

Level 4

$$z - x^2 - 2w^2 = 0$$

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

$$\begin{aligned} w &> 0 \\ w - 1 &< 0 \\ 3w - 1 &< 0 \\ 2w - 1 &< 0 \end{aligned}$$

Level 2

$$\begin{aligned} x &> 0 \\ x + w - 1 &< 0 \end{aligned}$$

Level 3

$$y - x^2 - w^2 = 0$$

Level 4

$$z - x^2 - 2w^2 = 0$$

Signs of projection factors (for defining formula) -

Index in RCAD: (3,3,2,2)

Level 1

$$w > 0$$

$$w - 1 < 0$$

$$3w - 1 < 0$$

$$2w - 1 < 0$$

Level 2

$$x > 0$$

$$x + w - 1 < 0$$

Level 3

$$y - x^2 - w^2 = 0$$

Level 4

$$z - x^2 - 2w^2 = 0$$

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 8.8.** 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

$$\begin{aligned} w &> 0 \\ w - 1 &< 0 \\ 2w^3 - 6w^2 + 9w - 2 &> 0 \\ w + 6 &> 0 \\ 3176w - 4353 &< 0 \\ 3w^2 - 8w + 3 &< 0 \end{aligned}$$

Level 2

$$\begin{aligned} x + w - 1 &= 0 \\ x &> 0 \\ x - 3 &< 0 \end{aligned}$$

Level 3

$$\begin{aligned} y &> 0 \\ y + w - 1 &< 0 \\ y^2 - 2w &< 0 \\ xy^2 - 3w &< 0 \\ xy^2 + 2wx - 6w &< 0 \end{aligned}$$

Level 4

$$z - xy^2 - w^2 = 0$$

Level 5

$$t - x^2y^2 - 3w^2 = 0$$

Signs of projection factors (for defining formula) -

Index in RCAD: (5,4,9,2,2)

Level 1

$$\begin{aligned} w &> 0 \\ w - 1 &< 0 \\ 2w^3 - 6w^2 + 9w - 2 &> 0 \\ w + 6 &> 0 \\ 3176w - 4353 &< 0 \\ 3w^2 - 8w + 3 &< 0 \end{aligned}$$

Level 2

$$\begin{aligned} x + w - 1 &= 0 \\ x &> 0 \\ x - 3 &< 0 \end{aligned}$$

Level 3

$$\begin{aligned} y &> 0 \\ y + w - 1 &< 0 \\ y^2 - 2w &< 0 \\ xy^2 - 3w &< 0 \\ xy^2 + 2wx - 6w &< 0 \end{aligned}$$

Level 4

$$z - xy^2 - w^2 = 0$$

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

$$\begin{aligned} w &> 0 \\ w - 1 &< 0 \\ 2 w^3 - 6 w^2 + 9 w - 2 &= 0 \\ w + 6 &> 0 \\ 3176 w - 4353 &> 0 \\ 3 w^2 - 8 w + 3 &> 0 \end{aligned}$$

Level 2

$$\begin{aligned} x + w - 1 &= 0 \\ x &> 0 \\ x - 3 &< 0 \end{aligned}$$

Level 3

$$\begin{aligned} y &> 0 \\ y + w - 1 &< 0 \\ y^2 - 2 w &< 0 \\ x y^2 - 3 w &< 0 \\ x y^2 + 2 w x - 6 w &< 0 \end{aligned}$$

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 >

More examples

- large sub-CAD, maybe  $R^5$
- cells in sub-CAD of different levels
- some variations on the whitney umbrella
- whitney in a sub-CAD
- whitney with multiple refinement points above it