Constructing a Single Open Cell in a Cylindrical Algebraic Decomposition

Christopher W. Brown
Computer Science Department, Stop 9F
United States Naval Academy
572M Holloway Road
Annapolis, MD 21402
wcbrown@usna.edu

ABSTRACT

This paper presents an algorithm that, roughly speaking, constructs a single open cell from a cylindrical algebraic decomposition (CAD). The algorithm takes as input a point and a set of polynomials, and computes a description of an open cylindrical cell containing the point in which the input polynomials have constant non-zero sign, provided the point is sufficiently generic. The paper reports on a few example computations carried out by a test implementation of the algorithm, which demonstrate the functioning of the algorithm and illustrate the sense in which it is more efficient than following the usual "open CAD" approach. Interest in the problem of computing a single cell from a CAD is motivated by a 2012 paper of Jovanovic and de Moura that require solving this problem repeatedly as a key step in NLSAT system. However, the example computations raise the possibility that repeated application of the new method may in fact be more efficient than the usual open CAD approach, both in time and space, for a broad range of problems.

Categories and Subject Descriptors

G.4 [**Mathematics of Computation**]: Mathematical software—*Algorithm design and analysis*

Keywords

cylindrical algebraic decomposition; polynomial inequalities

1. INTRODUCTION

In a 2012 paper [3], Jovanovic and de Moura present NLSAT, a novel algorithm that uses Conflict-Driven Clause Learning (CDCL)-style search to decide the satisfiability of systems of polynomial equalities and inequalities over the real numbers. An essential step in this algorithm is to take an assignment of real values that does not satisfy the original system, and generalize it to a larger region in which all points fail to satisfy the original system. They do this by constructing a single cell in a Cylindrical Algebraic Decomposition (CAD) defined from a set of polynomials — namely the cell that contains the unsatisfying assignment. This problem of constructing a single cell in a CAD efficiently has not, to the best of our knowledge, been addressed in the literature prior to Jovanovic and de Moura's work. The problem is important in light of the success NLSAT has already demonstrated, but also, as will be described later in the paper, it is a problem that has the potential to be important in other contexts as well. This paper takes the first steps towards a dedicated study of this problem, providing an efficient algorithm for constructing and computing with individual *open* cells in CADs, and reporting on the performance of a test implementation of this algorithm. Adapting and extending the algorithm to lower-dimensional cells is an interesting subject for future work.

Roughly speaking, the problem we consider is this: given a set P of polynomials in $\mathbb{R}[x_1, \ldots, x_n]$ and a point $\alpha \in \mathbb{R}^n$, compute the full dimensional cell from the CAD defined by P that contains point α, assuming α is not a zero of any of the polynomials in P. Jovanovic and de Moura gain an advantage over a straight-forward application of CAD to determine the satisfiability of a formula in several ways, but the two that are relevant here are that: 1) the set of polynomials they start with when they build a single CAD cell to generalize conflicting assignment is often smaller than the full set of polynomials in the input formula[1], and 2) in lifting they only lift above the one cell at each level containing the given point α. The fundamental idea behind this paper is that the projection operator can be substantially improved when we restrict ourselves to constructing a single cell containing the point α. In particular, the present paper describes how this can be done when the goal is to construct an open CAD cell containing α. Restricting the problem to open cells simplifies the algorithm, correctness proof, and implementation. However, the approach should be generalizable to constructing cells of lower dimension, and such a generalization is planned as future work.

A precise formulation of the problem appears below. One of the issues that must be addressed in a precise formulation is that smaller projection factor sets generally result in larger CAD cells. Thus, the cell that we construct is often a superset of the cell containing α in the full CAD

[1]A more strictly correct statement of this requires comparing the projection closure of the set of polynomials appearing in the input formula to the set of polynomials they start with in building the single CAD cell.

This paper is authored by an employee(s) of the United States Government and is in the public domain.
ISSAC'13, June 26–29, 2013, Boston, Massachusetts, USA.
ACM 978-1-4503-2059-7/13/06.

constructed in the usual way from the same initial set of polynomials. Although this complicates the description of the problem, it is a good thing! In the context of Jovanovic and de Moura's NLSAT procedure, for instance, a larger cell means a stronger generalization of the conflicting assignment.

1.1 The precise problem

This paper assumes some familiarity with CAD (Cylindrical Algebraic Decomposition) on the part of the reader. In particular, the paper requires the concepts behind and algorithms for constructing an "Open CAD" for a set of polynomials (see for example [4] and [6]), which are actually much simpler than their counterparts for standard CADs. Those unfamiliar with the subject of CAD, may wish to look at the introduction given in [1] which, though quite old, provides a clearly-written, well-explained starting point for the subject. For those familiar with CAD, but not Open CAD, we give a very brief overview.

Generally, CAD construction starts with a set A of input polynomials over some ordered set of variables, $x_1 \prec x_2 \prec \cdots \prec x_n$. The goal is to construct a decomposition of $\mathbb{R}^n$ into cells in which the elements of A have constant sign, and to do it in such a way that the regions in the decomposition have a special "cylindrical" arrangement with respect to the variable ordering. In the case of an *Open CAD* all cells are open sets, and we do not require $\mathbb{R}^n$ to be decomposed in the strict sense, rather we allow some points to be "missed", as long as the set of points in $\mathbb{R}^n$ that are not contained in any cell has measure zero. Usually, CAD construction proceeds in two stages: *projection*, and *lifting*. Projection produces a set $P \subseteq A$, called the projection factor set, that implicitly defines a CAD. Lifting produces an explicit data structure representing the CAD implicitly defined by P. This data structure can be queried to provide a sample point from each cell in the CAD, and a semi-algebraic description of each cell in the CAD.

The projection process is usually defined in terms of a *projection operator*, which eliminates a variable from a set of projection factors, thereby creating new projection factors. The projection operator is applied recursively until all but one variable is eliminated, and the set of input polynomials along with all the polynomials created by applications of the projection operator is the projection factor set. In the case of Open CAD we would use the "Open-McCallum projection" operator, which is defined as follows:[2] If P_k is a set of irreducible polynomials in $\mathbb{R}[x_1, \ldots, x_k]$ of positive degree in x_k, then $\text{ProjMOpen}_k(P_k)$ is the set of irreducible factors of

$$\bigcup_{p \in P_k} \text{ldcf}_{x_k}(p) \cup \bigcup_{p \in P_k} \text{disc}_{x_k}(p) \cup \bigcup_{p,q \in P_k, p \neq q} \text{res}_{x_k}(p, q).$$

The closure of a set of polynomials $P \subset \mathbb{R}[x_1, \ldots, x_n]$ under the Open-McCallum projection is the set P^* computed as follows:

$P^* := P$
for k from n down to 2 do
  $P^* := P^* \cup \text{ProjMOpen}_k(\{p \in P^* | \text{the level of } p \text{ is } k\})$

Given a finite set of input polynomials $P \subset \mathbb{R}[x_1, \ldots, x_n]$, an Open CAD for P is constructed by computing P^*, the closure of P under the Open-McCallum projection, and following the usual CAD lifting procedure with projection factor set P^* with the restriction that lifting is only done over full-dimensional cells, i.e. cells whose level and dimension are equal. The full-dimensional cells of level n comprise the cells of the Open CAD.

We are now ready to give a precise statement of the problem we want to solve:

The Open-Cell Problem Given a set P of polynomials in $\mathbb{R}[x_1, \ldots, x_n]$ and a point $\alpha \in \mathbb{R}^n$, compute a description of an open cylindrical algebraic cell $C \subseteq \mathbb{R}^n$ containing point α and in which the elements of P have constant, non-zero sign such that, if F is the projection closure of P under the Open-McCallum projection, C contains at least one open maximal connected region in which the elements of F have constant non-zero sign. If any element of F is zero at α, FAIL is an acceptable result.

1.2 This paper's contribution

The problem of constructing a single cell of a CAD does not seem to have been considered in the literature prior Jovanovic and de Moura's paper. As described earlier, the approach we take goes beyond what they describe in their paper because it actually reduces the size of the set of projection polynomials, which both speeds up the process of constructing the cell and produces a larger cell — both of which are improvements. The specific contributions of this paper are:

1. OC-CONSTRUCT a simple, efficient algorithm for the Open-Cell problem,
2. a formal proof of correctness for OC-CONSTRUCT, and
3. empirical examples of the operation of OC-CONSTRUCT based on a test implementation.

The remainder of this paper is organized as follows: Section 2 discusses the basic ideas that lead to the algorithms presented later in the paper. An intuitive feeling for these ideas will aid in reading the remainder of the paper. Section 3 describes the OpenCell data structure — which is what the Open-Cell Problem is asking us to produce. Some important properties of this data structure are proved. Section 4 gives the Algorithm OC-CONSTRUCT and proves its correctness. Finally, Section 5 provides some example computations with our test implementation that illustrate how OC-CONSTRUCT works, and why it has the real potential to be the basis for more efficient algorithms to solve some problems than the usual CAD approach can provide.

2. THE BASIC IDEA

The fundamental idea behind the algorithms proposed in this paper is best understood from a simple example. Suppose that we want to construct a single open cell from the point $\alpha = (-1/2, 1/2)$ and input polynomials $P = \{y^2 - 1 - x^2, y - x - 4, y + x - 5\}$. The CAD that would normally be constructed from this set, is shown on the left in Figure 1, with the cell containing $(-1/2, 1/2)$ highlighted.

The three input polynomials appear as the line with positive slope, the line with negative slope and the "hour glass" curve. The vertical lines are defined by the zeros of the polynomials resulting from the projection of the input polynomials.

[2]This description uses the term *level* (see Definition 1).

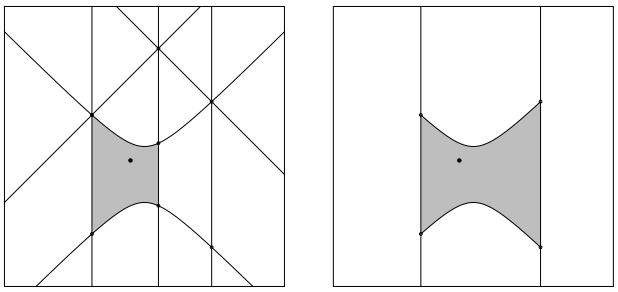

Figure 1: On the left is the full CAD. On the right is the single open cell constructed by OC-Construct. In both plots, the point α is shown by a black dot.

The important observation is that the right boundary of the cell containing α is defined by the intersection of the two non-vertical lines, but that intersection point lies outside of any region containing α in which the signs of the input polynomials stay constant. The algorithm we present recognizes that neither $V(y-x-4)$ nor $V(y+x-5)$ are part of the boundary of the cell containing α we will ultimately construct, and therefore their intersections with one another are irrelevant. Thus the algorithm deduces that the resultant of $y-x-4$ and $y+x-5$ can be left out of the projection. The right side of Figure 1 shows the cell OC-Construct constructs from this input. The cell is larger, and fewer distinct polynomials are required to define its boundaries.

Making this optimization requires us to be able to recognize which polynomials form the boundaries of the cell we are constructing. This is easy to do because we start with the point α that identifies which particular cell we want to construct. Continuing with our example problem, we substitute all but the last coordinate of α into our input polynomials to get univariate polynomials in the main variable – y in this case. We isolate the roots of the three resulting univariate polynomials, and order them and the last coordinate of α. The result is shown in Figure 2.

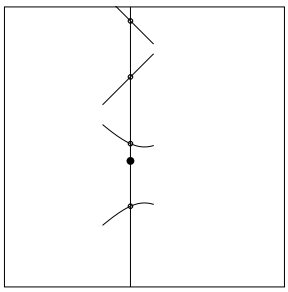

Figure 2: Ordered roots of the input polynomials (circles) and α (filled dot).

The roots immediately above and below the final coordinate of α identify which of the original input polynomials will form the upper and lower boundary of the cell we ultimately construct. (In higher dimensions, we can do the exact same thing — still only ever requiring univariate root isolation.) Generalizing the example, intersections of the varieties of polynomials that do not define the upper and/or lower boundaries of the cell we are trying to construct are irrelevant, and therefore pairwise resultants of those polynomials need not be included in the projection.

3. THE OPENCELL DATA STRUCTURE

Our goal is to construct a representation of a single open cell containing the point α. Before we can give an algorithm for computing this, we must define the representation — we must define the data structure we will use to represent the cell. We call this the *OpenCell data structure*, and in this section we define the structure and prove several important properties about it.

CADs are defined with respect to some variable ordering. In this paper, without loss of generality, we assume the order $x_1 \prec x_2 \prec \cdots \prec x_n$.

Definition 1. *The* level *of a non-constant polynomial $p \in \mathbb{R}[x_1, \ldots, x_n]$ is the largest k such that $deg_{x_k}(p) > 0$.*

Definition 2. *For $p \in \mathbb{R}[x_1, \ldots, x_n]$ and non-negative integer i, we define the* indexed root expression *$root(p, i, x_k)$ at the point $\gamma = (\gamma_1, \ldots, \gamma_n)$ as the ith distinct real root of $p(\gamma_1, \ldots, \gamma_{k-1}, x_k)$ (ordered smallest to largest). If the level of p is not k, if polynomial $p(\gamma_1, \ldots, \gamma_{k-1}, x_k)$ is the zero polynomial or if it has fewer than i distinct real roots, the expression has value* Undef. *Note that this is a more restricted notation than the notation introduced in [2]. We also allow the special indexed root expressions $root(+\infty, 1, x_k)$ and $root(-\infty, 1, x_k)$ to represent positive and negative infinity, respectively. We won't do any arithmetic with these expressions, so those semantics won't be addressed.*

We formally extend the usual relational operators $\{<, >, \leq, \geq, =, \neq\}$ to be false if either the left or right hand sides is $undef$, and we allow the expressions $\text{root}(+\infty, 1, x_k)$ and $\text{root}(-\infty, 1, x_k)$ on the left and right side of the relational operators with the obvious semantics.

Definition 3. *A* RealAlgNum *is a 4-tuple (p, I, t, j) where p is a squarefree univariate polynomial, I is an isolating interval for the jth distinct real root of p ordered from smallest to largest, and t is the trend of p at the root in I, i.e. the sign of p' at the root. If A is a RealAlgNum we adopt the notation $A.p$, $A.I$, $A.t$ and $A.j$ to refer to the four components of the tuple. We refer to the real number that is the $A.j$th distinct real root of $A.p$ as $val(A)$. To simplify the presentation below, we allow $A.p$ to be $+\infty$ or $-\infty$, in which case $val(A)$ is $+\infty$ or $-\infty$, respectively.*

Definition 4. *An* OpenCell Data Structure D containing rational point $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$ *is a list $D[1], \ldots, D[k]$ of 4-tuples (l, L, u, U), where $l \in \mathbb{R}[x_1, \ldots, x_k]$ and L is a RealAlgNum or $l = -\infty$, and $u \in \mathbb{R}[x_1, \ldots, x_k]$ and U is a RealAlgNum or $u = +\infty$, and the following additional requirements hold: Note: $S(D)$ and $F(D)$ are defined below, but used in this definition.*

1. *$D[1], \ldots, D[k-1]$ is an Open Cell Data Structure containing the point $(\alpha_1, \ldots, \alpha_{k-1})$.*
2. *$D[k].l = -\infty$ and $D[k].L = (-\infty, [0, 0], 0, 1)$ or*
   (a) *$D[k].l$ is irreducible and of positive degree in x_k and $D[k].l(\alpha_1, \ldots, \alpha_{k-1}, x_k) = D[k].L.p(x_k)$.*
   (b) *$disc_{x_k}(D[k].l)$ and $ldcf_{x_k}(D[k].l)$ have constant non-zero sign on $S(D[1], \ldots, D[k-1])$.*

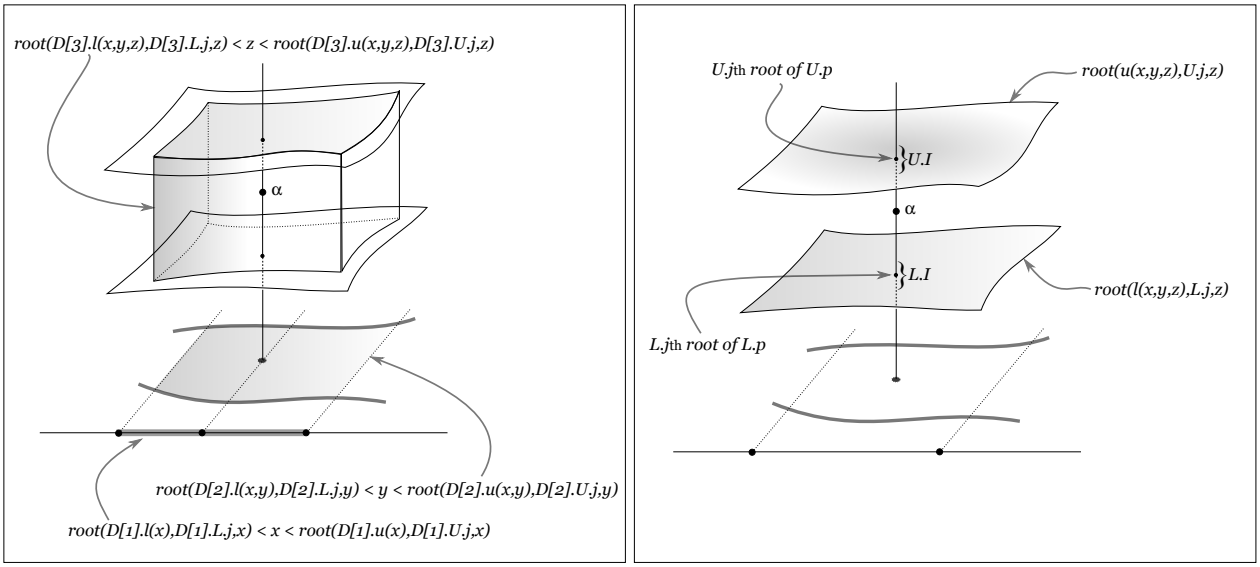

Figure 3: The diagram on the left shows how the elements of an OpenCell data structure D taken together define an open cell. The shaded regions are $S(D[1])$, $S(D[1], D[2])$ and $S(D[1], D[2], D[3])$. The diagram on the right shows how univariate root isolation and the components of the OpenCell data structure identify which sections of which polynomials define the upper and lower boundaries of the open cell defined by the data structure.

   (c) *$D[k].L.p$ has no root in the interval $(val(D[k].L), val(D[k].U))$.*

3. *$D[k].u = +\infty$ and $D[k].U = (+\infty, [0,0], 0, 1)$ or*

   (a) *$D[k].u$ is irreducible and of positive degree in x_k and $D[k].u(\alpha_1, \ldots, \alpha_{k-1}, x_k) = D[k].U.p(x_k)$.*

   (b) *$disc_{x_k}(D[k].u)$ and $ldcf_{x_k}(D[k].u)$ have constant non-zero sign on $S(D[1], \ldots, D[k-1])$.*

   (c) *$D[k].U.p$ has no root in the interval $(val(D[k].L), val(D[k].U))$.*

4. *If $D[k].l \neq -\infty$ and $D[k].u \neq +\infty$, $res_{x_k}(D[k].l, D[k].u)$ has constant non-zero sign on $S(D[1], \ldots, D[k-1])$.*

5. *The following formula is satisfied at point α:*

$$root(D[k].l, D[k].L.j, x_k) < \alpha_k < root(D[k].u, D[k].U.j, x_k)$$

Figure 3 provides two diagrams that illustrate how the OpenCell data structure defines an actual open cell containing point α.

Definition 5. *Let D be an OpenCell data structure containing point $\alpha \in \mathbb{R}^k$. We define $F(D)$ to be true if D is the empty list, and otherwise*

$$F(D[1], \ldots, D[k-1]) \wedge$$
$$root(D[k].l, D[k].L.j, x_k) < x_k < root(D[k].u, D[k].U.j, x_k).$$

We define $S(D)$ as $\mathbb{R}^0$ if D is the empty list, and otherwise

$$S(D) = \{\gamma \in \mathbb{R}^n \mid F(D) \text{ is satisfied at } \gamma\}.$$

Theorem 1. *Let D be an OpenCell data structure containing point $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$. The following properties hold:*

1. *$D[k].l$ and $D[k].u$ are delineable over $S(D[1], \ldots, D[k-1])$.*

2. *$S(D)$ is an open cylindrical subset of $\mathbb{R}^k$ containing α.*

3. *$S(D)$ is the maximal connected region containing α in which $D[1].l, D[1].u, \ldots, D[k].l, D[k].u$ have constant, non-zero sign.*

Proof. We proceed inductively on k. Consider the case $k = 1$. Univariate polynomials are by definition delineable, which takes care of Property 1. When $k = 1$, $S(D)$ is an open interval, which is an open cylindrical subset of $\mathbb{R}^1$. This takes care of Property 2. Finally, when $k = 1$, $S(D)$ is precisely the interval $(\text{val}(D[k].L), \text{val}(D[k].U))$ in which, by definition, neither $D[k].L.p = D[k].l$ nor $D[k].U.p = D[k].u$ are zero. Since the zeros of $D[k].l$ and $D[k].u$ are the endpoints of this interval, it is a maximal connected region in which $D[k].l$ and $D[k].u$ have constant non-zero sign. This takes care of Property 3.

Suppose $k > 1$. We will only consider the case in which $D[k].l \neq -\infty$ and $D[k].u \neq +\infty$, since the other three cases are similar but easier. By induction, all three points hold for the OpenCell data structure $D[1], \ldots, D[k-1]$ containing point $(\alpha_1, \ldots, \alpha_{k-1})$. By definition, the leading coefficients and discriminants of $D[k].l$ and $D[k].u$, as well as their resultant, all have constant non-zero sign in $S(D[1], \ldots, D[k])$, which is an open cylindrical cell and, since by Property 3 its boundaries are semi-algebraic, it is an open cylindrical algebraic cell — in fact, an analytic submanifold. By Theorem 2 of McCallum's paper defining his projection operator ([5]), $D[k].l$ and $D[k].u$ are delineable over $S(D[1], \ldots, D[k])$. This takes care of Property 1. McCallum's Theorem also assures us that the sections $\text{root}(D[k].l, D[k].L.j, x_k)$ and $\text{root}(D[k].u, D[k].U.j, x_k)$ are the graphs of non-intersecting real-valued algebraic functions over $S(D[1], \ldots, D[k])$, so the

fact that (by the definition of the OpenCell data-structure)

$$\text{root}(D[k].l, D[k].L.j, x_k) < \text{root}(D[k].u, D[k].U.j, x_k)$$

at $(\alpha_1, \ldots, \alpha_{k-1})$ implies that the same inequality holds over the entire region $S(D[1], \ldots, D[k-1])$. This proves that $S(D)$ is an open cylindrical subset of $\mathbb{R}^k$ which, since it's obvious that $\alpha \in S(D)$, proves Property 2.

Finally, we prove Property 3. Since we have shown that $S(D)$ is an open cylindrical subset of $\mathbb{R}^k$ containing α, it suffices to show that if a sequence of points in $S(D)$ converges to a point $\gamma \notin S(D)$, then some polynomial in the list $D[1].l, D[1].u, D[2].l, D[2].u, \ldots, D[k].l, D[k].u$ vanishes at γ.

Suppose there is an infinite sequence $\gamma^1, \gamma^2, \ldots$ of points in $S(D)$ that converges to point γ not in $S(D)$. Let π_{k-1} denote the projection of a point in real n-dimensional space, where $n \geq k$, onto $(k-1)$-dimensional space. We have two cases. First, suppose $\pi_{k-1}(\gamma) \notin S(D[1], \ldots, D[k-1])$. Then since $\pi_{k-1}(\gamma^1), \pi_{k-1}(\gamma^2), \ldots$ converges to $\pi_{k-1}(\gamma)$, we are guaranteed by induction that one of $D[1].l$, $D[1].u$, $D[2].l$, $D[2].u$, …, $D[k-1].l$, $D[k-1].u$ is zero at $\pi_{k-1}(\gamma)$. So the second case is $\pi_{k-1}(\gamma) \in S(D[1], \ldots, D[k-1])$. In this case, $\gamma \notin S(D)$ implies that

$$\text{root}(D[k].l, D[k].L.j, x_k) < x_k < \text{root}(D[k].u, D[k].U.j, x_k) \quad (1)$$

fails to hold at γ. Since (1) holds for all points in the sequence $\gamma^1, \gamma^2, \ldots$, this implies that either $D[k].l$ or $D[k].u$ is zero at γ, which completes the proof of Property 3. □

4. THE OC-CONSTRUCT ALGORITHM

In this section we describe $OC-Construct$, a simple, efficient algorithm for the Open-Cell problem. The majority of work is done by a sub-algorithm O-P-Merge which we define first.

Theorem 2. *Algorithm* O-P-Merge *meets its specification.*

Proof. We fix n and proceed inductively on k. Clearly, in Step 1, Fail is returned if $P(\alpha) = 0$ as required by the specification. So we continue under the assumption that $P(\alpha) \neq 0$. As in the proof of Theorem 1, we will assume that $D[k].l \neq -\infty$ and $D[k].u \neq +\infty$, since the proof for cases in which this does not hold is similar but simpler.

Case $k = 0$ and $k = 1$. If $k = 0$, P is a non-zero constant, so returning $D' = D$ meets the specification. When $k = 1$ we skip Steps 4–7. Those steps basically refine the cell defined by D' to ensure the delineability of P, $D[k].l$, and $D[k].u$. Since 1-level polynomials are always delineable, those steps aren't needed. When $k = 1$ the cell represented by D is an open interval. Steps 8–10 orders the roots of P, $D[1].l$ and $D[1].u$ and chooses the nearest root below α_1 as the (possibly new) lower bound, and the nearest root above α_1 as the (possibly new) upper bound. Thus Point 1 of the output requirements is satisfied. When $k = 1$, $F = \{P, D[1].l, D[1].u\}$ and, as discussed above $D'[1].l$ and $D'[1].u$ are taken from this set, so Point 3 of the output requirements is satisfied. It is also clear from the way $D'[1].l$ and $D'[1].u$ are chosen that Point 2 also holds. Finally, we must be sure that D' has the properties required of an OpenCell data structure. All but parts 2c, 3c and 5 of Definition 4 hold trivially, because they deal with properties of $D[1], \ldots, D[k-1]$, which is the empty list when $k = 1$. We choose $D'[1].l, D'[1].L$ to be the nearest root below α_1 and $D'[1].u, D'[1].U$ to be the nearest root above α_1. So $S(D')$ is the interval $(val(D'[1].L), val(D'[1].U))$ and α_1 is in that interval. This shows that part 5 of Definition 4 holds, and since we chose the nearest roots as upper and lower bounds for this interval, parts 2c and 3c hold as well.

Case $k > 0$. Denote by $f_1, \ldots, f_r$ the irreducible factors of the elements of F, in the order in which they are added in Step 7.[3] Each of the f_i's is of level less than k, since they are resultants, discriminants and coefficients, so by induction we may assume that the calls to O-P-Merge return results meeting its specification.

Suppose that $i-1$ iterations of the loop in Step 7 have completed without Fail having been returned, and consider the call O-P-Merge(α, D', f_i) in the ith iteration. Denote by F' the closure under the Open-McCallum projection of

$$\left\{f_i, D'[1].l, D'[1].u, \ldots, D'[k-1].l, D'[k-1].u\right\}.$$

[3]Notice that the order in which these factors are added is not specified by the algorithm. Different choices of order can result in different cells — some larger, some smaller, some more quickly computed, some more slowly. Investigating good orders is a possible avenue for future work.

Algorithm 1 OpenCell–Polynomial Merge (O-P-Merge)

Input:
α : a rational point in $\mathbb{R}^n$, $\alpha = (\alpha_1, \ldots, \alpha_n)$
D : an OpenCell data structure containing α
P : an integral polynomial of level k, where $k < n$

Output:
D' : an OpenCell data structure containing α such that (denote by F is the closure under the Open-McCallum projection of $\{P, D[1].l, D[1].u, \ldots, D[k].l, D[k].u\}$)
1. $S(D') \subseteq S(D)$
2. $S(D')$ contains at least one maximal connected region in which the elements of F have constant, non-zero sign, and
3. $\{D'[1].l, D'[1].u, \ldots, D'[k].l, D'[k].u\} \subseteq F$

-or-
Fail whenever $P(\alpha) = 0$. Outputting Fail is also allowed, but not required, whenever any element of F is zero at α.

1: if $P(\alpha) = 0$ return Fail
2: set D' to a copy of D
3: if $k = 0$ return, if $k = 1$ goto Step 8
4: set $F = \{\text{ldcf}_{x_k}(P), \text{disc}_{x_k}(P)\}$
5: if $D[k].l \neq -\infty$ set $F = F \cup \{\text{res}_{x_k}(P, D[k].l)\}$
6: if $D[k].u \neq +\infty$ set $F = F \cup \{\text{res}_{x_k}(P, D[k].u)\}$
7: for each f from the set of irreducible factors of the elements of F do
  set $R =$ O-P-Merge(α, D', f)
  if $R =$ Fail return Fail, else set $D' = R$
8: isolate the real roots of $P(\alpha_1, \ldots, \alpha_{k-1}, x_k)$ obtaining the ordered list of RealAlgNum's $B_1, \ldots, B_s$, and merge $D[k].L$, $D[k].U$ and α_k into that list.
9: if, for some i, B_i, α_k appear consecutively in the list, set $D'[k].l = P$ and $D'[k].L = B_i$
10: if, for some j, α_k, B_j appear consecutively in the list, set $D'[k].u = P$ and $D'[k].U = B_j$
11: return

Polynomial f_i is a factor of $\text{res}_{x_k}(P, D[k].l)$, $\text{res}_{x_k}(P, D[k].u)$, $\text{disc}_{x_k}(P)$, or $\text{ldcf}_{x_k}(P)$, which are all elements of the Open-McCallum projection of

$$\{P, D[1].l, D[1].u, D[2].l, D[2].u, \ldots, D[k].l, D[k].u\}$$

from which it follows that $F' \subseteq F$.

Suppose FAIL is returned by the recursive call. This means that some element F' vanishes at α, which in turn means that some element of F vanishes at α, in which case the specification allows the result of O-P-Merge(α, D, P) to be FAIL.

Next, suppose that the recursive call does not return FAIL, but instead returns the OpenCell data structure R. By induction, $S(R[1], \ldots, R[k-1]) \subseteq S(D'[1], \ldots, D'[k-1])$ which means that any polynomial that has constant non-zero sign in $S(D'[1], \ldots, D'[k-1])$ also has constant non-zero sign in $S(R[1], \ldots, R[k-1])$.

Assuming that all iterations of the loop at Step 7 are completed without FAIL being returned, we are left after Step 7 with the OpenCell data structure D' such that $\text{disc}_{x_k}(P)$, $\text{ldcf}_{x_k}(P)$, $\text{res}_{x_k}(P, D[k].l)$ and $\text{res}_{x_k}(P, D[k].u)$ all have constant non-zero sign in $S(D'[1], \ldots, D'[k-1])$. Thus, by Theorem 2 of McCallum's paper on projection [5], the zeros of $D[k].l$, $D[k].u$ and P over $S(D'[1], \ldots, D'[k-1])$ are delineable, i.e. their zeros define a finite set of non-intersecting graphs of real-valued algebraic functions over $S(D'[1], \ldots, D'[k-1])$. Steps 8,9,10 simply use this fact to deduce which section of $D[k].l$, $D[k].u$ and P is immediately below α and which section is immediately above α based on which root of $D[k].l(\alpha_1, \ldots, \alpha_{k-1}, x_k)$, $D[k].u(\alpha_1, \ldots, \alpha_{k-1}, x_k)$ and $P(\alpha_1, \ldots, \alpha_{k-1}, x_k)$ is immediately below α_k and which is immediately above. This is, in fact, the usual concept of a "sample point" in CAD. Step 9 sets $D'[k].l$ and $D'[k].L$ to the nearest lower-bound, which guarantees us that

$$\text{root}(D[k].l, D[k].L.j, x_k) \leq \text{root}(D'[k].l, D'[k].L.j, x_k) < \alpha_k.$$

Step 10 sets $D'[k].u$ and $D'[k].U$ to the nearest upper-bound, which guarantees us that

$$\alpha_k < \text{root}(D'[k].u, D'[k].U.j, x_k) \leq \text{root}(D[k].u, D[k].U.j, x_k).$$

Combining these facts with

$$S(D'[1], \ldots, D'[k-1]) \subseteq S(D[1], \ldots, D[k-1])$$

which we know by induction, shows that $S(D') \subseteq S(D)$. Thus Point 1 of the output requirements has been shown to hold. To address Point 3 of the output requirements, we note that we have already shown that

$$\{D'[1].l, D'[1].u, \ldots, D'[k-1].l, D'[k-1].u\} \subseteq F$$

and that $D'[k].l$ and $D'[k].u$ are both elements of $\{D[k].l, D[k].u, P\} \subseteq F$.

At this point we note that the preceding observations also show that the structure D' returned in this case satisfies the five properties enumerated in Definition 4 as requirements of an OpenCell data structure — which, of course, is also a requirement of the algorithm.

Finally, we consider Point 2 of the output requirements. Since $S(D')$ is an open set, it must contain a point at which all elements of F are non-zero in which all elements of F are non-zero. Let T be a maximal connected region containing such a point. The polynomials defining the boundaries of the cell $S(D')$ are all elements of F, from which it follows that $T \subseteq S(D')$. □

Algorithm 2 OpenCell Construct (OC-CONSTRUCT)

Input:

α : a rational point in $\mathbb{R}^n$, $\alpha = (\alpha_1, \ldots, \alpha_n)$

P : a set $\{p_1, \ldots, p_r\}$ of irreducible elements of $\mathbb{R}[x_1, \ldots, x_n]$

Output:

D : an OpenCell data structure containing α such that (denoting by F the closure of P under the Open-McCallum projection operator)

1. $\{D[1].l, D[1].u, \ldots, D[n].l, D[n].u\} \subseteq F$
2. the elements of P have constant, non-zero sign in $S(D)$

-or-

FAIL whenever any element of P is zero at α. Outputting FAIL is also allowed, but not required, whenever any element of F is zero at α.

1: if $P = \emptyset$ and $n = 0$ return the empty list
2: if $P = \emptyset$ and $n > 0$
  set $D' =$ OC-CONSTRUCT $((\alpha_1, \ldots, \alpha_{n-1}), \emptyset)$
  set $B = (-\infty, (-\infty, (0,0), 0, 1), +\infty, (+\infty, (0,0), 0, 1))$
  set $D = D'[1], \ldots, D'[n-1], B$
3: if $P = \{p_r\} \cup \{p_1, \ldots, p_{r-1}\}$, where $r > 0$
  set $D' =$ OC-Construct$(\alpha, \{p_1, \ldots, p_{r-1}\})$
  if $D' =$ FAIL return FAIL
  set $D =$ O-P-MERGE(α, D', p_r)
4: return D

THEOREM 3. *Algorithm* OC-CONSTRUCT *meets its specification.*

This we state without proof, since the correctness of Algorithm OC-CONSTRUCT follows quite directly from the correctness of Algorithm O-P-MERGE.

The Algorithm OC-CONSTRUCT solves the Open-Cell problem. In particular, if it returns an OpenCell data structure D, the formula $F(D)$ describes a cell with the required properties.

5. EXAMPLE COMPUTATIONS

In this section we present a few example computations. These are for the purpose of explaining the algorithm and illustrating where and how it avoids some computations and produces larger — i.e. more general — cells than would be produced computing a complete CAD in the usual way.

This paper does not attempt to provide an exhaustive comparison of the efficiency of OC-CONSTRUCT versus Open-CAD construction. For starters, an investigation of the order in which polynomials are added should be made first. Right now, OC-CONSTRUCT and O-P-MERGE add polynomials in an arbitrary order. Secondly, it is not clear that comparing the two algorithms outside of the context of a particular application is possible. Jovanovic and de Moura's NLSAT algorithm is an example of one such context.

5.1 A 2D problem

Consider the Open-Cell Problem:

$$\left(-\frac{1}{3}, \frac{1}{3}\right), \left\{x^2 + y^2 - 1, 2y^2 - x^2(2x+3), y + \frac{1}{2}x - \frac{1}{2}\right\}.$$

The OC-Construct algorithm constructs a OpenCell data structure for α and x^2+y^2-1, then merges $2y^2-x^2(2x+3)$ with that OpenCell, and then merges the resulting OpenCell with the polynomial $y+1/2x-1/2$. This is illustrated in the sequence of diagrams in Figure 4. The important

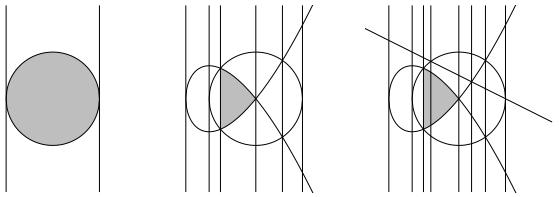

Figure 4: The sequence of OpenCell's constructed by OC-Construct for this example. Each is superimposed on the full CAD that would have been constructed from the polynomials involved at that point. Especially important to note is that the transition from the middle to the right-most OpenCell (i.e. merging $y+1/2x-1/2$) does not change the OpenCell data-structure.

thing to see in this example is that merging $y+1/2x-1/2$ with the current OpenCell in the final step (going from the middle to the right-most OpenCell in Figure 4) does not actually change the OpenCell. In this step it is deduced that $y+1/2x-1/2$ has constant non-zero sign in the current OpenCell, so something is gained, but the OpenCell itself is not refined. Moreover, we did not even have to compute the resultant of x^2+y^2-1 and $y+1/2x-1/2$, which is something the regular CAD algorithm would have constructed given the same polynomials.

Another important observation to be made about this example is that adding the polynomials in a different order makes a difference in which polynomials get computed and which computations can be avoided. Although it is not the case here, examples can be constructed in which the actual cell that gets computed is affected by the order in which the OC-Construct and O-P-Merge add polynomials. Investigating the impact of different criteria for ordering polynomials to add is a direction for future work.

5.2 A 4D example

Next we consider the 4-variable polynomial p

$$b^4t_1^4t_2^4 + 2a^2b^2t_1^4t_2^4 - b^2t_1^4t_2^4 + a^4t_1^4t_2^4 - a^2t_1^4t_2^4 + 2b^4t_1^2t_2^4 + 2a^2b^2t_1^2t_2^4 - 2b^2t_1^2t_2^4 + b^4t_2^4 - b^2t_2^4 + 4ab^3t_1^3t_2^3 + 4a^3bt_1^3t_2^3 + 4ab^3t_1t_2^3 + 2a^2b^2t_1^4t_2^2 + 2a^4t_1^4t_2^2 - 2a^2t_1^4t_2^2 + 6a^2b^2t_1^2t_2^2 + 4a^3bt_1^3t_2 + a^4t_1^4 - a^2t_1^4$$

and variable order $b \prec a \prec t_2 \prec t_1$. This example is for purpose of illustration, so there's no claim that we can extrapolate much from this problem, just that it helps understand what the algorithm does. This polynomial/order was chosen because a) it comes from an application, b) it is a 4-variable problem, which allows us to observe behavior that only occurs when there are repeated projections, and c) this polynomial is small enough that that a full open CAD can be relatively easily computed for it. With a single input polynomial the first projection step performed by OC-Construct is identical to first projection step performed in constructing a complete open CAD. That means there is no obvious reason why fewer projection factors would be computed by OC-Construct — unlike the 2D example from the previous section.

The program QepcadB computes a complete Open-CAD for this problem in 88s — almost all of this time is spent computing and factoring resultants and discriminants. That CAD consists of 824 full-dimensional cells. The projection factor set consists of 23 one-level polynomials, 9 two-level polynomials, 3 three-level polynomials and 1 four-level polynomial.

Consider the Open-Cell Problem instance $\left(\frac{2}{7}, \frac{-3}{11}, \frac{5}{9}, \frac{12}{23}\right), p$. (This point was chosen arbitrarily, and has no special significance.) Our test implementation constructs an Open-Cell data structure for this in .57s, with description

$$\begin{array}{rcccl} -\infty & < & t_1 & < & +\infty \\ 0 & < & t_2 & < & +\infty \\ root(q,1,a) & < & a & < & 0 \\ 0 & < & b & < & 1 \end{array}$$

where $q = a^6 + 3b^2a^4 - 3a^4 + 3b^4a^2 + 21b^2a^2 + 3a^2 + b^6 - 3b^4 + 3b^2 - 1$. Comparing times is of limited meaning at this point, because of the many implementation differences that may obscure algorithmic differences. For example, the test implementation uses Maple to factor and compute resultants and discriminants, while QepcadB uses Singular. Moreover, OC-Construct computes a lot less than QepcadB, because QepcadB computes all open cells, there being no way to instruct it to only compute what is needed to construct the cell with sample point α. However, one meaningful comparison is the number of projection factors that get computed. There are only five projection factors in the OpenCell data structure that the test implementation produces. However, projection factors are constructed and then discarded. Accounting for that, the test implementation computes 10 one-level, 8 two-level, 3 three-level and 1 four-level projection factors. This is substantially less than is required to construct a complete open CAD. It should be noted that the number of four and three-level projection factors is necessarily the same for this example, because there is a single input polynomial.

As previously noted, comparing QepcadB and OC-Construct as in the above example is not terribly meaningful, because QepcadB computes so much less. To better understand how they really compare, we performed the following test. We called the test implementation on polynomial p for the sample points of each of the 824 open cells constructed by QepcadB. This took a combined total of 15s.[4] The results show that the set of all projection factors constructed in the 824 calls to the test implementation consists of 15 one-level polynomials, 9 two-level polynomials, 3 three-level polynomials and 1 four-level polynomial. In other words, eight of the 23 one-level projection factors constructed by the usual Open-McCallum projection were never produced at any point by the 824 calls to the test implementation. Hand-in-hand with this, close inspection of the cells constructed by the test implementation shows that most of them contain multiple open cells from the complete open CAD. In fact, there are only 352 distinct cells constructed, and they combine to cover the same space as the 824 open cells in the complete open CAD. All of this raises

[4]The test implementation memoizes discriminant, resultant and factorization computations, so that time isn't wasted in such a scenario recomputing the same factors.

the possibility that covering $\mathbb{R}^4$ — up to some lower dimensional set — with (possibly overlapping) open cells might be accomplished faster and with fewer cells using an approach based on OC-CONSTRUCT than by computing a complete open CAD in the usual way.

6. CONCLUSION & FUTURE WORK

This paper has considered the *Open-Cell Problem* which, roughly speaking, is the problem of starting with a point α and set P of irreducible polynomials, and computing from them a description of an open cylindrical cell containing α in which the elements of P have constant non-zero sign. Even more roughly speaking, it could be described as constructing a single open cell from a CAD. The algorithm OC-CONSTRUCT, which solves this problem, was introduced and proved correct. Finally, some example computations using a test implementation were described. They demonstrated the algorithm, and illustrated how OC-CONSTRUCT constructs its one cell more efficiently than does applying open CAD techniques naively, and how the cell constructed by OC-CONSTRUCT is often larger ("more general") than the cell from the complete open CAD that contains the point α. The last example computation illustrated why it may be more efficient to apply OC-CONSTRUCT over and over to cover (up to a measure zero set) the input space with open cells than to apply the usual open CAD, which offers some exciting possibilities.

This paper has introduced a new approach to constructing CAD cells — basically allowing us to compute CAD projection "locally", meaning around the input point α. The original motivation to even consider the problem was the paper by Jovanovic and de Moura [3] that introduced NLSAT. A natural starting point for future work building on the present paper would be to try to formulate an "open-NLSAT" and base it around OC-CONSTRUCT. Other applications could also be considered. The approach itself could most likely be improved in a number of ways, the one highlighted here being the order in which polynomials are "merged" with the OpenCell data structure. Indeed, the approach could be formulated in terms of merging OpenCell data structures rather than merging a polynomial and an OpenCell data structure, which would provide even more choices. Finally, this paper has demonstrated that the approach followed by OC-CONSTRUCT works, meaning that we gain both in efficiency and quality of solution. The approach needs to be extended to lower-dimensional cells so that the same gains can be made in those cases. These last two points will be explained in more detail below.

Choices and Merging Step 7 of Algorithm 1 refines the OpenCell data structure D' by iteratively merging the elements of F. The order in which the elements of F are chosen affects the OpenCell resulting from the process. Therefore, one should investigate ordering criteria. In fact, many variations on Algorithm 1 might be considered. An interesting variation is to recast the process as merging two OneCell data structures, rather than a polynomial and a OneCell data structure. In this case, we arrive at a natural divide & conquer algorithm that does a projection step, produces OneCell data structures recursively for each of the resulting projection factors (using the same point for all of them), and then merges all the OneCell data structures together. Preliminary work on this approach shows promise.

Generalizing to Lower-Dimensional Cells Algorithms 1 & 2 return FAIL if point α is a zero of a projection factor. A generalization of this work would have them return a lower-dimensional cell in which projection factors have constant sign, but not necessarily non-zero sign. This necessitates considering a more complicated projection operator, and performing root isolation with algebraic numbers. However, it also offers far more opportunities for improvements in efficiency compared to the standard CAD approach.

Replacing the Usual CAD Approach As already noted, constructing a OneCell data structure for each of the sample points in a CAD can be substantially faster and more memory efficient than constructing a regular CAD of $\mathbb{R}^n$. In many applications it would be possible to replace the usual CAD construction with the construction of many, possibly overlapping, OneCell data structures — as long as it could be demonstrated that $\mathbb{R}^n$ was covered by the union of these cells (at least up to some measure zero set). To make this work, some kind of control must be provided that chooses points and applies OC-CONSTRUCT in such a way that it terminates after a finite number of steps having provably covered $\mathbb{R}^n$, up to a measure zero set. Jovanovic and de Moura approach provides such a control, though it would be interesting to consider other, more geometric rather that logic-oriented ways of doing it.

7. ACKNOWLEDGEMENTS

This work arose out of discussions at the Schloss Dagstuhl Seminar 12462, *Symbolic Methods for Chemical Reaction Networks*, held in November 2012. I would like to thank the organizers of the event and Leibniz Zentrum für Mathematik for a wonderful scientific program. Particularly, I would like to thank Thomas Sturm and Marek Kosta for introducing me to Jovanovic and de Moura's paper, and for helping me understand it during our several discussions at Schloss Dagstuhl along with Carsten Conradi and Andreas Eggers.

8. REFERENCES

[1] ARNON, D. S., COLLINS, G. E., AND MCCALLUM, S. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal on Computing 13*, 4 (1984), 865–877.

[2] BROWN, C. W. *Solution Formula Construction for Truth Invariant CAD's*. PhD thesis, University of Delaware, 1999.

[3] JOVANOVIC, D., AND DE MOURA, L. M. Solving non-linear arithmetic. In *IJCAR* (2012), pp. 339–354.

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

[5] MCCALLUM, S. An improved projection operator for cylindrical algebraic decomposition. In *Quantifier Elimination and Cylindrical Algebraic Decomposition* (1998), B. Caviness and J. Johnson, Eds., Texts and Monographs in Symbolic Computation, Springer-Verlag, Vienna.

[6] STRZEBONSKI, A. Solving systems of strict polynomial inequalities. *Journal of Symbolic Computation 29* (2000), 471–480.