Department of Computer Science

Technical Report

An implementation of CAD in Maple utilising problem formulation, equational constraints and truth-table invariance

Matthew England

Technical Report 2013-04 **June 2013**

ISSN 1740-9497

Copyright ©June 2013 by the authors.

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

An implementation of CAD in Maple utilising problem formulation, equational constraints and truth-table invariance

Matthew England

Department of Computer Science, University of Bath, Bath, UK
M.England@bath.ac.uk

Abstract

Cylindrical algebraic decomposition (CAD) is an important tool for the investigation of semi-algebraic sets, with applications within algebraic geometry and beyond. We recently reported on a new implementation of CAD in Maple which implemented the original algorithm of Collins and the subsequent improvement to projection by McCallum. Our implementation was in contrast to Maple's in-built CAD command, based on a quite separate theory. Although initially developed as an investigative tool to compare the algorithms, we found and reported that our code offered functionality not currently available in any other existing implementations. One particularly important piece of functionality is the ability to produce order-invariant CADs. This has allowed us to extend the implementation to produce CADs invariant with respect to either equational constraints (ECCADs) or the truth-tables of sequences of formulae (TTICADs). This new functionality is contained in the second release of our code, along with commands to consider problem formulation which can be a major factor in the tractability of a CAD.

In the report we describe the new functionality and some theoretical discoveries it prompted. We describe how the CADs produced using equational constraints are able to take advantage of not just improved projection but also improvements in the lifting phase. We also present an extension to the original TTICAD algorithm which increases both the applicability of TTICAD and its relative benefit over other algorithms. The code and an introductory Maple worksheet / pdf demonstrating the full functionality of the package should accompany this report

This work is supported by EPSRC grant EP/J003247/1.

1 Introduction

This report is the second, following [15], to describe our implementation of cylindrical algebraic decomposition (CAD) in Maple. This report describes the functionality added to the second release of the code. This includes the ability to: produce CADs invariant with respect to equational constraints following [19], produce truth-table invariant CADs (TTICADs) following [3], and derive different formulations of the input with heuristics to pick the best following [4]. The introduction continues with a brief background on CAD and a summary of the findings of [15]. Then in Section 2 we describe the new functionality in more detail.

As with [15], the implementation prompted some theoretical discoveries which are also described in this report. In Section 3 we explain how the use of equational constraints can not only lead to the improved projection described in [19], but also improved lifting. Then in Section 4 we present an extension to the original TTICAD algorithm of [3], allowing it to be applied to a wider variety of input formulae, including formulae which could not be tackled by equational constraints alone. We demonstrate that this increases not just the applicability of TTICAD, but its relative benefit over other algorithms.

This report should be accompanied by the second release of the `ProjectionCAD` code and an introductory Maple worksheet / pdf demonstrating the full functionality of the package.

1.1 Background on CAD

A cylindrical algebraic decomposition (CAD) is a decomposition of $\mathbb{R}^n$ into cells, constructed with respect to an input, usually either polynomials or formulae, in n variables. Each cell describes a semi-algebraic set and the cells are cylindrically arranged, meaning the projection of any two cells is either equal or disjoint. A CAD is sign-invariant if the input polynomials have constant sign on each cell. Such a CAD allows for the solution of many problems defined by the polynomials. Collins provided the definition and first algorithm [9, 1], motivated as a tool for quantifier elimination in real closed fields. Since their discovery they have found many other applications ranging from robot-motion planning [20, 12] to simplification technology [2, 13, etc.]

Collins' algorithm has two phases. The first, *projection*, applies a projection operator repeatedly to a set of polynomials, each time producing another set in one fewer variables. Together these sets contain the *projection polynomials*. The second phase, *lifting*, then builds the CAD incrementally from these polynomials. First $\mathbb{R}$ is decomposed into cells which are points and intervals corresponding to the real roots of the univariate polynomials. Then $\mathbb{R}^2$ is decomposed by repeating the process over each cell using the bivariate polynomials at a sample point of the cell. The output for each cell consists of *sections* of polynomials (where a polynomial vanishes) and *sectors* (the regions between these). Together these form the *stack* over the cell, and taking the union of these stacks gives the CAD of $\mathbb{R}^2$. This process is repeated until a CAD of $\mathbb{R}^n$ is produced. The projection operator must be chosen in order to conclude that the CAD of $\mathbb{R}^n$ produced using sample points in this way is sign-invariant. The output of a CAD algorithm depends on the ordering of the variables. In this paper we usually work with polynomials in $\mathbb{Z}[x_1, \ldots, x_n]$ with the variables, $\mathbf{x}$, listed in ascending order; (so we first project with respect to x_n and so on until we have univariate polynomials in x_1). The *main variable* of a polynomial, $\text{mvar}(f)$, is the greatest variable present with respect to the ordering.

Since Collins published the original algorithm there has been much research into improvements with a summary of the developments over the first twenty years given by [10]. A key area of work was in the definition of projection operators to use in the first phase. McCallum defined an operator [17, 18] which usually produces far fewer polynomials than Collins' and thus an algorithm which uses it produces a CAD with less cells in less time. However, unlike Collins' operator, McCallum's cannot be applied universally. McCallum defined the notion of input which is well-oriented [18] to describe when his operator could be used.

QEPCAD [6] is a dedicated CAD implementation: a command-line program which implements both McCallum's projection operator and Hong's modification of Collins' operator [11]). In 2009 a radically different approach was presented [8] where instead of projection and lifting, a cylindrical decomposition of complex space is first produced which is then refined to a CAD of real space. That algorithm is distributed with MAPLE where it is part of the `RegularChains` library [16, etc.].

1.2 The `ProjectionCAD` package

`ProjectionCAD` is a MAPLE package developed at The University of Bath to implement CAD via projection and lifting. It was initially developed in order to study the differences between the traditional approach to CAD and the new approach in [8] using regular chains. In [15] we described our implementation of both McCallum's and Collin's algorithms to produce sign-invariant CADs, and how we made use of commands from the `RegularChains` library to give output in the same format as [8].

`ProjectionCAD` is a theoretical tool, unoptimised, and so does not compete particularly well on timings for sign-invariant CADs (see [15, Section 4] for a comparison). However, it is currently the only implementation which can produce order-invariant CADs, (CADs with the stronger property that each polynomial has constant order of vanishing on each cell). This feature is essential for the extensions to the package described in Section 2, which in turn allow this implementation to compete with the others.

`ProjectionCAD` is also the only implementation of delineating polynomials [18, 7] which modify Collins' original lifting algorithm to allow McCallum's projection operator to be applied more widely. Without these, the condition of well-orientedness has to be strengthened to thus restricting when the improved projection operator may be used. QEPCAD does not make use of delineating polynomials and so there are examples where `ProjectionCAD` can produce CADs which QEPCAD cannot (see [15, Section 2]. However, QEPCAD has implemented many of the the ideas in [7] in order to minimise the effect of this.

2 New functionality in `ProjectionCAD` V2

In this section we describe the new functionality available in the second release of the `ProjectionCAD` package. This adds to the work in the first release (described in [15]) for producing sign and order-invariant CADs.

2.1 Equational constraints

An **equational constraint** is an equation logically implied by a formula. Their use in CAD is based on the observation that the formula will be false for any cell in the CAD where the equation is not satisfied. Hence, instead of a CAD sign-invariant with respect to all the polynomials, we could use one that is: (a) sign-invariant with respect to the equational constraint; and (b) sign-invariant with respect to the other constraints only in those cells over the sections of the equational constraint. Such a CAD is said to be **invariant with respect to an equational constraint**. This observation was made in [10] with the first detailed approach given by [19] where a new projection operator was presented to implement the idea.

The background theory

Let P be the McCallum projection operator. Informally this is applied to a set of polynomials to produce the coefficients, discriminant and cross resultants (full details of the theory are presented in [17, 18] and the implementation in `ProjectionCAD` is discussed in [15]). We note that most implementations including `ProjectionCAD` make trivial simplifications such as removal of constants, exclusion of polynomials that are identical to a previous entry (up to constant multiple), and only including those coefficients which are necessary for the theory to hold. This operator may be used for CAD provided the input polynomials are **well oriented**. This is defined to mean that every projection polynomial has a finite number of nullification points, i.e. if the main variable is x_k then $f(\alpha, x_k) = 0$ for at most a finite number of $\alpha \in \mathbb{R}^{k-1}$.

Given a problem with an equational constraint f McCallum suggested a reduced projection operator P_f. Informally, this consists of the coefficients and discriminant of f together with the resultant of f taken with each of the other polynomials. Formally, we must take more care leading to an operator P_F where F is a squarefree basis for the primitive part of f (see [19] for the full details of the theory). This improved projection operator is used for the first projection, reverting to P for subsequent projections. To use this new operator in place of the original McCallum operator the polynomials in the CAD algorithm the input must again satisfy the well-orientedness condition

The implementation in `ProjectionCAD`

Algorithm 1 describes our implementation in `ProjectionCAD` which builds a CAD of $\mathbb{R}^n$ with respect to an equational constraint. While the projection phases follows the work of [19] exactly the lifting phase is somewhat different, as discussed in detail in Section 3. The algorithm uses the sub-algorithms: `CADFull` (Algorithm 3 in [15]) to build an order-invariant CAD for the projection polynomials not in the main variable; `LiftingSet` (Algorithm 2) to calculate the set of polynomials to use for the final lift; and `CADGenerateStack` (Algorithm 4 in [15]) to then lift over cells with respect to these polynomials.

It is well documented that CADs invariant with respect to equational constraints usually have fewer cells than those which are sign-invariant for all polynomials. Example 1 gives a simple demonstration of this.

Example 1. Assume the variable ordering $y > x$. Consider the polynomials

$$f = x^2 + y^2 - 4, \qquad \text{and} \qquad g = xy - 1$$

which define the circle and hyperbola in Figure 1, and the problem $f = 0 \wedge g < 0$. We assume the variable ordering $y > x$. The problem could be solved by means of a full sign-invariant CAD for $\{f, g\}$ which `ProjectionCAD` would produce using 83 cells. However, it is more efficient to produce a CAD invariant with respect to the equational constraint f, which using Algorithm 1 in `ProjectionCAD`, has 53 cells.

The induced CADs of the real line have 15 and 13 cells respectively. The difference is that the full CAD identifies the origin on the real line, corresponding to the asymptote of the hyperbola and arising from the

Algorithm 1: ECCAD

Input : • A polynomial $f \in \mathbb{R}[x_n, \ldots, x_1]$. • A set of polynomials $G \subset \mathbb{R}[x_n, \ldots, x_1]$.
Output: Either: • a CAD of $\mathbb{R}^n$, sign-invariant with respect to f and, when $f = 0$, also with respect to G, or; • **FAIL** if the input is not well-oriented.

Set $E := \{f\}$.
Compute F, the finest squarefree basis for the primitive parts of E.
if $n = 1$ **then**
    return The CAD of $\mathbb{R}$ formed by the decomposition of the real line according to the real roots of the polynomials in F.
Set $A := G \cup E$.
Compute the set C of contents of the elements of A.
Compute the set B, the finest squarefree basis for the primitive parts of A.
Construct the projection set $\mathfrak{P} := C \cup P_F(B)$.
Attempt to construct a lower-dimensional CAD by calculating $D := \texttt{CADFull}(\mathfrak{P}, \text{proj} = \text{McCallum}, \text{fcad} = \text{true})$.
if `CADFull` warns about potential failure **then**
    return FAIL.
for *each cell* $c \in \mathcal{D}$ **do**
    Set $L_c := \texttt{LiftingSet}(c, A, E)$.
    if $L_c =$ **FAIL then**
        return FAIL.
    Set $S_c := \texttt{CADGenerateStack}(c, L_c)$.
return $\bigcup_c S_c$.

Algorithm 2: LiftingSet

Input : • A cell c from a CAD of $\mathbb{R}^{n-1}$. • A set of polynomials $A \subset \mathbb{R}[x_n, \ldots, x_1]$.
• A subset of polynomials $E \subset A$.
Output: Either: • a set of polynomials $L \subset \mathbb{R}[x_n, \ldots, x_1]$ to use for lifting over the cell, or;
• **FAIL** if the input is not well-oriented.

Set $L := \{\}$.
if any polynomial in E is nullified on c **then**
    if $\dim(c) > 0$ **then**
        calculate $\text{ExclP}_E(A) := P(A \setminus E) \setminus P_E(A)$.
        if $\text{ExclP}_E(A)$ is empty or contains only constants **then**
            $L := L \cup A_i$.
        else
            return FAIL.
    else
        $L := L \cup A_i$.
else
    $L := L \cup E_i$.

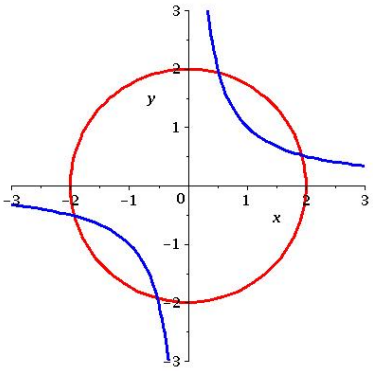

Figure 1: Plots of the curves used in Example 1.

inclusion of the coefficients of g is the projection polynomials. The CAD using the equational constraint only considers the behaviour of g on cells where $f = 0$, (by including the resultant of f and g in the first projection but not the coefficients of g individually) and thus the origin is not identified.

As we discuss further in Section 3, Algorithm 1 actually offers some subtle improvements to the original algorithm described in [19] and implemented in QEPCAD. For the example above this manifests in the cell count of 53 produced by `ProjectionCAD` comparing to a cell count of 69 with QEPCAD as described in Example 4.

2.2 TTICAD

Given a sequence of formulae, a **truth table invariant CAD** (TTICAD) is a CAD such that each formula has constant truth value (true or false) on each cell. The idea of using truth invariance was introduced in [5] for use in simplifying sign-invariant CADs. Of course, a sign-invariant CAD is itself truth-invariant but by focusing on the second property certain cells can be merged to give a smaller CAD. Recently, in [3] an algorithm was presented allowing the efficient construction of TTICADs (without having to first build a sign-invariant CAD).

The background theory

The algorithm presented in [3] makes use of the theory of equational constraints. It assumes that each formula in the sequence has a designated equational constraint (for that formula only). Then a new projection operator was defined which, informally, included the following polynomials.

- For each formula, those obtained by applying McCallum's projection operator for equational constraints (see [19] and Section 2.1 above).
- For each formula, the polynomials obtained by taking the resultant of the designated equational constraint and each of the other polynomials.
- The cross-resultants of the set of designated equational constraints from each formula.

This new projection operator is used for the first projection, with the original McCallum projection operator used for subsequent projections. Constructing the projection set this way ensures that the CAD produced is sign-invariant with respect to equational constraints, and that for cells where an equational constraint is zero, the other polynomials from that formula are sign-invariant.

As with the other projection operators, this one is only valid for use when the input sets satisfy a condition. The well-orientedness condition described above applied to the polynomials in the input would be sufficient, but instead, a finer condition is used, discussed further below.

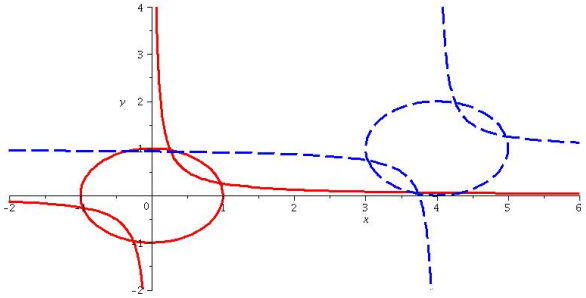

Figure 2: Plots of the curves used in Example 2. The solid circle is f_1, the solid hyperbola g_1, the dashed circle f_2 and the dashed hyperbola g_2.

The implementation in `ProjectionCAD`

In [3] an algorithm was provided and verified for the approach above. In Algorithm 3 we present the algorithm used in `ProjectionCAD`. This extends the algorithm in [3] so that it may be applied to sequences of formulae in which not every formula has a designated equational constraint.

First we formally define the projection operator used. Let $\mathcal{A} = \{A_i\}_{i=1}^t$ be a list of irreducible bases A_i and let $\mathcal{E} = \{E_i\}_{i=1}^t$ be a list of non-empty subsets $E_i \subseteq A_i$. Put $A = \bigcup_{i=1}^t A_i, E = \bigcup_{i=1}^t E_i$. (We use the convention of uppercase Roman letters for sets and calligraphic letters for sequences).

Then define the **reduced projection of $\mathcal{A}$ with respect to $\mathcal{E}$**, denoted by $P_{\mathcal{E}}(\mathcal{A})$, as follows:

$$P_{\mathcal{E}}(\mathcal{A}) := \bigcup_{i=1}^t P_{E_i}(A_i) \cup \mathrm{RES}^{\times}(\mathcal{E})$$

where

$$P_E(A) = P(E) \cup \{\mathrm{res}_{x_n}(f,g) \mid f \in E, g \in A, g \notin E\};$$
$$\mathrm{RES}^{\times}(\mathcal{E}) = \{\mathrm{res}_{x_n}(f,\hat{f}) \mid \exists\, i,j \text{ such that } f \in E_i, \hat{f} \in E_j, i < j, f \neq \hat{f}\}.$$

We say that $\mathcal{A}$ is **well oriented with respect to $\mathcal{E}$** if whenever $n > 1$: every polynomial $f \in E$ is nullified by at most a finite number of points in $\mathbb{R}^{n-1}$, and $P_{\mathcal{E}}(\mathcal{A})$ is well-oriented in the original sense. Algorithm 3 shows how the projection operator may be applied more widely.

The definitions above are the same as [3] but their use in Algorithm 3 differs from their use in the algorithm presented in [3]. This is discussed further in Section 4. Experimental results in [3] show that TTICAD offers huge benefits compared to sign-invariant CADs constructed with the same technology (`ProjectionCAD`). It also demonstrates that the TTICAD theory allows `ProjectionCAD` to compete with the existing state of the art CAD technology.

Example 2. Assume the variable ordering $y > x$. Consider the polynomials:

$$f_1 = x^2 + y^2 - 1 \qquad g_1 = xy - \tfrac{1}{4}$$
$$f_2 = (x-4)^2 + (y-1)^2 - 1 \qquad g_2 = (x-4)(y-1) - \tfrac{1}{4}$$

which are plotted in Figure 2, and the formula

$$\Phi = (f_1 = 0 \wedge g_1 < 0) \vee (f_2 = 0 \wedge g_2 < 0)\,.$$

Using `ProjectionCAD` a full sign-invariant CAD with respect to the polynomials can be constructed using 317 cells while a TTICAD can be produced using only 105 cells. The problem could also be tackled using the theory of equational constraints alone, (by declaring the implicit equational constraint $f_1 f_2 = 0$). Using Algorithm 1 in `ProjectionCAD` produces a CAD using 145 cells. There are more cells than the TTICAD because the projection set will include polynomials relating to the intersection of f_1 with g_2 and f_1 with g_2 which are ignored by the TTICAD. This example was worked through and discussed in detail in [3].

Algorithm 3: TTICAD

Input : A list of quantifier-free formulae $\Phi = \{\phi_i\}_{i=1}^t$ in variables $x_1, \ldots, x_n$. Each ϕ_i may or may not have a designated equational constraint $f_i = 0$.

Output: Either • $\mathcal{D}$: A TTICAD of $\mathbb{R}^n$ for Φ; or • **FAIL**: If Φ is not well oriented

for $i = 1 \ldots t$ **do**
  Extract the set A_i of polynomials in ϕ_i.
  if ϕ_i *has a designated equational constraint* f_i **then**
    set $E_i := \{f_i\}$.
  else
    set $E_i := A_i$.
  Compute the finest squarefree basis F_i for the primitive parts of E_i.
Set $F := \cup_{i=1}^t F_i$.
if $n = 1$ **then**
  return The CAD of $\mathbb{R}$ formed by the decomposition of the real line according to the real roots of the polynomials in F.
for $i = 1 \ldots t$ **do**
  Compute the set C_i of contents of the elements of A_i.
  Compute the set B_i, the finest squarefree basis for the primitive parts of A_i.
Set $C := \cup_{i=1}^t C_i$, $\mathcal{B} := (B_i)_{i=1}^t$ and $\mathcal{F} := (F_i)_{i=1}^t$.
Construct the projection set: $\mathfrak{P} := C \cup P_{\mathcal{F}}(\mathcal{B})$.
Attempt to construct a lower-dimensional CAD by calculating $D :=$ `CADFull`$(\mathfrak{P}, \text{proj} = \text{McCallum}, \text{fcad} = \text{true})$
if `CADFull` warns about potential failure **then**
  return FAIL.
for *each cell* $c \in \mathcal{D}$ **do**
  Set $L_c :=$ `LiftingSet`(c, A_i, E_i).
  if $L_c =$ **FAIL then**
    return FAIL.
  Set $S_c :=$ `CADGenerateStack`(c, L_c).
return $\bigcup_c S_c$.

2.3 Problem formulation and heuristics

The algorithms described above and in [15] can have different results for the same problem depending on how the problem is formulated. For example, which variable ordering is used, which equational constraint is designated, how a formula is broken down into sub-formulae for TTICAD.

In [4] these issues were studied in detail and heuristics were developed for choosing the formulations. The heuristics were based on two measures of CAD complexity which can be applied to the the projection polynomials. The first measure, introduced in [14], is the sum of total degrees of the each monomial in each polynomial, abbreviated to `sotd`. The second introduced in [4] in the number of distinct real roots of the univariate projection polynomials (the number of sections in the induced CAD of $\mathbb{R}$), abbreviated to `ndrr`. Both measures have been implemented in `ProjectionCAD`, the second by first taking a square free basis of the polynomials and then using Maple's implementation of Sturm chains. Heuristic algorithms have also been introduced to consider possible formulations and pick the best based on the values of these measures. The user can specify to use either measure, a combination for successively breaking ties or a weighted average. There is also implementation of the greedy algorithm for choosing variable orderings given in [14], and the ability to specify variable blocks and then let the algorithm pick the best ordering which respects these blocks (as required when using CAD for quantifier elimination).

3 Improved lifting with equational constraints

The implementation of CAD with respect to an equational constraint described in Section 2.1 offers some subtle improvements to the original work of [19] and implementations based on this such as QEPCAD.

As discussed, the approach rests on the use of an improved projection operator P_f to build a CAD, taking the place of either the Collins or original McCallum projection operator which construct full sign-invariant CADs. When defined in [19] the author only discussed how this would improve the projection phase of CAD, by creating fewer projection polynomials. The only modification to the lifting phase described was the need to check the well-orientedness condition, as is also the case if using the original McCallum operator.

Algorithm 1 makes further changes to the lifting phase, which although subtle can have a significant effect on the CADs produced. We note that these ideas are also used in Algorithm 3 for producing TTICADs.

3.1 A finer check for well-orientedness

Theorem 2.3 of [19] is the key result that justifies the use of the improved projection operator for equational constraints. The theorem allows us to confirm that the cells in the outputted CAD which are over the sections of the equational constraint are sign-invariant with respect to the other polynomials. The theorem only holds when the equational constraint has a finite number of nullification points, a condition guaranteed by the input being well-oriented.

However, the standard well-orientedness condition is actually stronger than necessary for this situation. It requires that all projection polynomials have a finite number of nullification points, including the non-equational constraints. Algorithm 1 does not check the condition for every projection polynomial. When using `CADFull` in step 9 the condition is checked for all projection polynomials not in the main variable, while in step 2 of Algorithm 2 the condition is checked for the equational constraint. However, the condition is never checked for the non-equational constraints (in the main variable) as this is not required for the theory of the projection operator to hold. QEPCAD appears to check for nullification of all projection polynomials, returning errors even if the theory does allow for the outputted CAD to have the requested properties. Although it has detailed checks to avoid unnecessary errors [7] Example 3 demonstrates that these false errors may still occur, arising from this situation.

Example 3. Consider the polynomials

$$f = x + y + z + w, \qquad g = zy - x^2 w$$

and the formula $f = 0 \wedge g < 0$. We could analyse the truth of the formula using a full sign-invariant CAD for $\{f, g\}$ which both `ProjectionCAD` and QEPCAD would produce with 557 cells. However, it is far more efficient to make use of the equational constraint. Using Algorithm 1 `ProjectionCAD` produces a CAD with 165 cells. Declaring the equational constraint in QEPCAD results in a CAD with 221 cells (the higher number due to the issues discussed in subsection 3.2). However, QEPCAD also returns an error message

```
Error! Delineating polynomial should be added over cell(2,2)!
```

indicating that the output is not guaranteed to be correct. In fact the output is correct since the the error message was triggered by the nullification of g when $x = y = 0$ which does not invalidate the theory. Thus it is the error message which is incorrect: no delineating polynomial was required.

3.2 Smaller lifting sets

Traditionally in CAD, the projection phase identifies a set of projection polynomials, which are then used in the lifting phase at sample points to create the stacks. However when constructing CADs with respect to equational constraints we can be more efficient by discarding some of the projection polynomials before lifting. The non-equational constraints (in the main variable) are part of the set of projection polynomials, required in order to produce subsequent projection polynomials through their resultant with the equational constraint. However, these polynomials are not then usually required for the lifting.

In Algorithm 1 the projection polynomials are formed from the input polynomials (in the main variable) and the set of polynomials $\mathfrak{P}$ constructed in step 8 which are not in the main variable. The lower dimensional CAD D constructed in step 9 is guaranteed to be sign-invariant (in fact order invariant) for $\mathfrak{P}$. In particular, $\mathfrak{P}$ contains the resultants of the equational constraint with the other constraints and thus D is already decomposing the domain into cells such that the presence of an intersection of f and g is invariant in each cell. Hence for the final lift we need only ensure that f is sign-invariant.

As demonstrated by Examples 4 and 5, using smaller lifting sets can reduce the number of cells in a CAD.

Example 4. Consider again the circle and hyperbola in Figure 1 and the formula $f = 0 \wedge g < 0$ from Example 1. Using Algorithm 1 `ProjectionCAD` produces a CAD invariant with respect to the equational constraint using 53 cells. This may be compared to QEPCAD which after declaring the equational constraint produces a CAD with 69 cells.

Both implementations give the same induced CAD of the real line but QEPCAD uses more cells for the CAD of $\mathbb{R}^2$. In particular, `ProjectionCAD` has a cell where $x < -2$ and y is free while QEPCAD uses three cells, splitting where g changes sign. The splitting is necessary for a sign-invariant CAD but not a CAD with respect to an equational constraint since f is non-zero for all $x < -2$. QEPCAD splits the cell unnecessarily since g is in the set of projection polynomials and thus used in the final lift.

Example 5. Consider again the polynomials and formula from Example 2. Here we explained how the problem could be tackled using an implicit equational constraint. Using Algorithm 1 in `ProjectionCAD` gives a CAD with 145 cells while declaring the equational constraint in QEPCAD results in a CAD with 249 cells.

3.3 Optimising the lifting to reduce cell counts and avoid unnecessary failure

In Subsection 3.2 above we described how smaller lifting sets are used to reduce cell counts. Actually, Algorithm 2 does sometimes use the original larger lifting sets, (in steps 6 and 10). These exceptions occur when an equational constraint is nullified. If this occurs over a zero dimensional cell (stet 6) then we know that the sample point is certainly representative of the cell and hence can proceed to make a stack of cells such that all the polynomials are sign-invariant trivially.

The other exception is implemented in steps 4 to 6 of Algorithm 2. In step 4 we form $\text{ExclP}_E(A) := P(A \setminus E) \setminus P_E(A)$ which is the set of projection polynomials that would have been calculated if using the original McCallum projection operator, but which are ignored by the reduced operator. That is,

$$P(A) = P_E(A) \cup \text{ExclP}_\text{E}(\text{A}).$$

If all the polynomials in $\text{ExclP}_E(A)$ are constant then we can use the theory of the original projection operator in [18] to conclude that the output of the algorithm will be a valid CAD. However, this requires the lifting set to contain all the projection polynomials.

Algorithm 1 uses Algorithm 2 to identify the best lifting set to use in each cell for the final lift. Hence when the lifting set is extended in this way it only effects the necessary cells thus minimising the cell count while maximising the success of the algorithm. Example 6 demonstrates this.

Example 6. Assume the variable ordering $w > z > y > x$. Consider the polynomials

$$f = z + yw, \quad g = yx + 1, \quad h = w(z + 1) + 1,$$

and the formula $f = 0 \wedge g < 0 \wedge h < 0$. Using the `ProjectionCAD` package we can build a full sign-invariant CAD for $\{f, g, h\}$ with 927 cells and a CAD invariant with respect to the equational constraint with 467 cells. The induced CAD of $\mathbb{R}^3$ has 169 cells and on five of these cells the polynomial f is nullified. On these five cells both y and z are zero, with x being either $0, 4$ or on the three intervals splitting at these points create.

In this example $\text{ExclP}_E(A) = \{z + 1\}$ arising from the coefficient of h. We see that this is a constant value of 1 on all five of the cells above. Thus the algorithm is allowed to proceed without error, lifting with respect to all the projection polynomials on these cells.

We note that the lifting set varies from cell to cell in D. For example, the stack over the cell $c_1 \in D$ where $x = y = z = 0$ uses three cells, splitting when $w = -1$. This is required for a CAD invariant with respect to

f since $f = 0$ on c but h changes sign when $w = -1$. Compare this with, for example, the cell $c_2 \in D$ where $x = y = 0$ and $z < -1$. The stack over c_2 has only one cell, with w free. The polynomial h will change sign over this cell, but this is not relevant since f will never be zero. This is achieved by including h in the lifting set only for the five cells of D where f was nullified. We note that for this example QEPCAD can make use of partial CAD trial evaluation techniques to build a CAD invariant with respect to f using only 203 cells.

4 Extensions to truth-table invariant CAD

Algorithm 3 describing the implementation of TTICAD in the `ProjectionCAD` package is an extended version of the algorithm given in [3].

It includes all three of the approaches for improved lifting with equational constraints discussed in Section 3. The first (checking the finer well-orientedness condition) and third (adapting the lifting set as required) were already used in [3] and follow automatically from the main theorem allowing the use of the reduced projection of $\mathcal{A}$ with respect to $\mathcal{E}$, (Theorem 3 in [3]). The second (analysing the excluded polynomials to reduce the risk of failure from not being well-oriented) was also discussed in [3], although it was not included in the algorithm there for simplicity. Lemma 7 in [3] validates its use in the TTICAD case.

However, the most significant extension is the relaxation of the input to allow formulae which do not have any equational constraint. Although the extension is reasonably straightforward, it actually dramatically increases both the applicability and benefit of TTICAD .

4.1 Extending the applicability of TTICAD

Algorithm 3 allows the user to input a sequence of formulae, each of which may or may not have an equational constraint. This is in contrast to the theory in [3] where it was assumed each formula had one. In Algorithm 3 formulae without equational constraints are dealt with by treating all their polynomials with the importance reserved for equational constraints, (step 3 to 6).

The main theorem of [3] validating the use of the projection operator (Theorem 3) requires no more adjustment that the redefinition of E_i (and thus E and $\mathcal{E}$) introduced by these steps. Extra polynomials have been added to the projection set sufficient to allow the conclusion that: (a) the CAD is sign-invariant with respect to all the equational constraints and all the other constraints belonging to a formulae with no equational constraint, and (b) in cells where an equational constraint vanishes the other constraints its formulae are sign invariant.

However, as with Theorem 3 in [3] this theorem holds when the equational constraints are not nullified and thus the well-orientedness condition must be extended to include the redefined $\mathcal{E}$ from the algorithm. This is indeed the case in the implementation since when a clause without an equational constraint has E_i redefined accordingly in step 4 and this set is passed to Algorithm 2 where it is used in the check at step 2.

4.2 Increasing the benefit of TTICAD

In this final section we demonstrate how the extension dramatically increases the benefit of the TTICAD theorem. First we note that in [3] experimental results demonstrated the great benefit of TTICAD over sign-invariant CADs through much smaller cell counts. The benefits increase in proportion to the number of formulae, as in Example 7.

Example 7. We consider a family of examples based on Example 2. Define the f_1, g_1, f_2, g_3 as in that example and consider also

$$f_3 = (x+4)^2 + (y+1)^2 - 1, \qquad g_3 = (x+4)(y+1) - 1/4.$$

Then consider the formulae

$$\Phi_1 = (f_1 = 0 \wedge g_1 < 0)\,, \qquad \Phi_2 = (f_1 = 0 \wedge g_1 < 0) \vee (f_2 = 0 \wedge g_2 < 0)\,,$$
$$\Phi_3 = (f_1 = 0 \wedge g_1 < 0) \vee (f_2 = 0 \wedge g_2 < 0) \vee (f_3 = 0 \wedge g_3 < 0)\,.$$

For each formula we used `ProjectionCAD` to construct a full sign-invariant CAD for the polynomials, a CAD invariant with respect to the implicit equational constraint and a TTICAD, using Algorithm 3 in [15] and Algorithms 1 and 3 from this report respectively. The sequence of formulae for TTICAD is the obvious one (breaking the formula at the disjunctions). We also used QEPCAD to analyse the formula on both its default settings and by declaring the implicit equational constraint manually. Table 1 shows the cell counts from these experiments.

Formula	`ProjectionCAD`			QEPCAD	
	`CADFull`	`ECCAD`	`TTICAD`	`Default`	`Declare`
Φ_1	83	69	53	69	83
Φ_2	317	145	105	317	249
Φ_3	695	237	157	695	509

Table 1: Table detailing the number of cells in CADs constructed using various algorithms to analyse the formulae in Example 7.

As expected, the TTICAD is the superior of these algorithms in terms of cell counts, with its advantage increasing with the number of disjunctions (separate formulae in the sequence inputted to the algorithm. QEPCAD can also use the theory of equational constraints to avoid building a full sign-invariant CAD (although it requires manual input), however, the smaller lifting sets used in `ProjectionCAD` (Subsection 3.2) mean that QEPCAD has higher cell counts.

The benefit of the extended TTICAD over a sign-invariant CAD will now increase in proportion with the number of the formulae *that have an equational constraint*. Hence, for such relaxed input, the benefit over sign-invariant CAD will be slightly less. However, for such input, the theory of equational constraints alone is not applicable (as there is no overall implicit equational constraint). Thus in these cases TTICAD is the only available theory to consider the structure given by the equations, and therefore of much greater importance. Example 8 demonstrates this.

Example 8. Consider again the polynomials from Example 7 and the formulae Φ_1, Φ_2, Φ_3 but this time with $f_1 < 0$ instead of $f_1 = 0$. For these formulae TTICADs can be produced with 83, 183 and 283 cells respectively. Although a little larger than the cell counts in Table 1, these still represent great savings over the full sign-invariant CADs. Further, for these modified formulae the `ECCAD` algorithm and declaring an equational constraint in QEPCAD is not applicable, making the difference between the possible cell counts for each problem much more dramatic.

5 Summary

We have described the new functionality present in the second release of `ProjectionCAD`. This includes an algorithm to produce a CAD invariant with respect to an equational constraint. Similar algorithms are present in other implementations such as QEPCAD, but these usually just include improvements to the projection phase, resulting in a smaller projection set and hence a CAD with less cells. In Section 3 we described how the theory of equational constraints also allows for improvements to the lifting phase which can further reduce the cell counts.

The second release also includes an algorithm to produce truth-table invariant CADs, the only such algorithm to be implemented. This is based on the theory in [3] but in Section 4 we describe how that theory can be extended to increase the applicability of the algorithm to situations where its benefit is greatest.

Throughout this report we have focussed on cell counts as the measure of the success of the implementation. Of course, the actual time taken to compute the CADs is also important and it is the case that QEPCAD and the other state of the art algorithms (`RegularChains` in Maple [8] and Mathematica [21]) are usually quicker to produce sign-invariant CADs. However, as reported in [3] the implementation of TTICAD offers such large reductions in cell counts that it allows `ProjectionCAD` to compete on timings as well as cell

counts. The extension to TTICAD described in this report will further increase the relative performance of TTICAD since it can now be applied to examples where the theory of equational constraints alone is no use.

Finally, we note that the second release of `ProjectionCAD` includes implementations of the work in [4] on heuristics for choosing how to formulate problems for the algorithms. The problem formulation can have a dramatic effect on the computation and future work on `ProjectionCAD` will include automating the use of these heuristics within the main algorithms.

References

[1] D. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM J, Comput.*, 13:865–877, 1984.

[2] R. Bradford and J.H. Davenport. Towards better simplification of elementary functions. In *Proceedings of the 2002 international symposium on symbolic and algebraic computation*, ISSAC '02, pages 16–22. ACM, 2002.

[3] R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Cylindrical algebraic decompositions for boolean combinations. *In Press: Proc. ISSAC 2013. Preprint at* `http://opus.bath.ac.uk/33926/`, 2013.

[4] R. Bradford, M. England, J.H. Davenport, and D. Wilson. Optimising problem formulations for cylindrical algebraic decomposition. *In Press: Proc, CICM-Calculemus 2013.* Preprint at `http://opus.bath.ac.uk/34373/`, 2013.

[5] C.W. Brown. Simplification of truth-invariant cylindrical algebraic decompositions. In *Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, ISSAC '98, pages 295–301. ACM, 1998.

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

[7] C.W. Brown. The McCallum projection, lifting, and order-invariance. Technical report, U.S. Naval Academy, Computer Science Department, 2005.

[8] C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, ISSAC '09, pages 95–102. ACM, 2009.

[9] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages*, pages 134–183. Springer-Verlag, 1975.

[10] G.E. Collins. Quantifier elimination by cylindrical algebraic decomposition – 20 years of progress. In B. Caviness and J. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symbolic Computation, pages 8–23. Springer-Verlag, 1998.

[11] G.E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12:299–328, 1991.

[12] J.H. Davenport. A "Piano-Movers" Problem. *SIGSAM Bull.*, 20(1-2):15–17, 1986.

[13] J.H. Davenport, R. Bradford, M. England, and D. Wilson. Program verification in the presence of complex numbers, functions with branch cuts etc. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC 2012, pages 83–88. IEEE, 2012.

[14] A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. In *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, ISSAC '04, pages 111–118. ACM, 2004.

[15] M. England. An implementation of CAD in Maple utilising McCallum projection. Department of Computer Science Technical Report series 2013-02, University of Bath. Available at `http://opus.bath.ac.uk/33180/`, 2013.

[16] M. Moreno Maza. On triangular decompositions of algebraic varieties. Technical report, NAG Technical Report, 1999.

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

[18] S. McCallum. An improved projection operation for cylindrical algebraic decomposition. In B. Caviness and J. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symbolic Computation, pages 242–268. Springer-Verlag, 1998.

[19] S. McCallum. On projection in CAD-based quantifier elimination with equational constraint. In *Proceedings of the 1999 international symposium on Symbolic and algebraic computation*, ISSAC '99, pages 145–149. ACM, 1999.

[20] J.T. Schwartz and M. Sharir. On the "Piano-Movers" Problem: II. General techniques for computing topological properties of real algebraic manifolds. *Adv. Appl. Math.*, 4:298–351, 1983.

[21] A. Strzeboński. Computation with semialgebraic sets represented by cylindrical algebraic formulas. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 61–68. ACM, 2010.