Truth Table Invariant Cylindrical Algebraic Decomposition

Russell Bradford [a], James H. Davenport [a], Matthew England [a], Scott McCallum [b] and David Wilson [a]

[a] *Department of Computer Science, University of Bath, Bath, BA2 7AY, UK*

[b] *Department of Computing, Macquarie University, NSW 2109, Australia*

Abstract

When using cylindrical algebraic decomposition (CAD) to solve a problem with respect to a set of polynomials, it is often not the signs of those polynomials that are of paramount importance but rather the truth values of certain quantifier free formulae involving them. This observation motivates our article and definition of a Truth Table Invariant CAD (TTICAD).

In ISSAC 2013 the current authors presented an algorithm that can efficiently and directly construct a TTICAD for a list of formulae in which each has an equational constraint. This was achieved by generalising McCallum's theory of reduced projection operators. In this paper we present an extended version of that theory which can be applied to an arbitrary list of formulae, present heuristics to help with the formulation of input to the algorithm, and explain how the theory of reduced projection operators can allow for further improvements in the lifting stage of CAD algorithms, even in the context of a single equational constraint.

The algorithm is implemented fully in Maple and we present promising results from experimentation.

Key words: cylindrical algebraic decomposition, equational constraint
1991 MSC: [2010] 68W30, 03C10

⋆ This work was supported by EPSRC grant EP/J003247/1.

Email addresses: R.J.Bradford@bath.ac.uk (Russell Bradford), J.H.Davenport@bath.ac.uk (James H. Davenport), M.England@bath.ac.uk (Matthew England), Scott.McCallum@mq.edu.au (Scott McCallum), D.J.Wilson@bath.ac.uk (David Wilson).

1. Introduction

A *cylindrical algebraic decomposition* (CAD) is a decomposition of $\mathbb{R}^n$ into cells arranged cylindrically (meaning the projections of any pair of cells are either equal or disjoint) each of which is (semi-)algebraic (describable using polynomial relations). CAD is a key tool in real algebraic geometry, both for the original motivation of solving quantifier elimination problems and a range of other applications that have been found since. These include robot motion planning (Schwartz and Sharir, 1983), parametric optimisation (Fotiou et al., 2005), epidemic modelling (Brown et al., 2006), theorem proving (Paulson, 2012) and programming with complex functions (Davenport et al., 2012).

Traditionally CADs are produced *sign-invariant* to a given set of polynomials, (the signs of the polynomials do not vary on the cells). However, this gives far more information than required for most problems. Usually a more appropriate object is a *truth-invariant* CAD (the truth of a logical formula does not vary on each cell).

In this paper we generalise to define *truth table-invariant* CADs (the truth values of a list of quantifier-free formulae do not vary within cells) and give an algorithm to compute these directly. This can be a tool to efficiently produce a truth-invariant CAD for a larger parent formula (built from the input list), or indeed the required object for solving a problem involving the input list. Examples of both such uses are provided following the formal definition in Section 1.2. We continue the introduction with some background on CAD, before defining our object of study and introducing some examples to demonstrate our ideas which we will return to throughout the paper. We then conclude the introduction by clarifying the contributions and plan of this paper.

1.1. Background on CAD

A *Tarski formula* $F(x_1, \ldots, x_n)$ is a Boolean combination $(\wedge, \vee, \neg, \rightarrow)$ of statements about the signs, $(= 0, > 0, < 0$, but therefore $\neq 0, \geq 0, \leq 0$ as well), of certain integral polynomials $f_i(x_1, \ldots, x_n)$. Such statements may involve the universal or existential quantifiers $(\forall, \exists)$. We use denote by QFF a *quantifier-free Tarski formula.*

Given a quantified Tarski formula

$$Q_{k+1}x_{k+1} \ldots Q_n x_n F(x_1, \ldots, x_n) \tag{1}$$

(where $Q_i \in \{\forall, \exists\}$ and F is a QFF) the *quantifier elimination problem* is to produce $\psi(x_1, \ldots, x_k)$, an equivalent QFF to (1).

Collins developed CAD as a tool for quantifier elimination over the reals. He proposed to decompose $\mathbb{R}^n$ cylindrically such that each cell was sign-invariant for all polynomials f_i occurring in F. Then ψ would be the disjunction of the defining formulae of those cells c_i in $\mathbb{R}^k$ such that (1) was true over the whole of c_i, which due to sign-invariance is the same as saying that (1) is true at any one *sample point* of c_i.

A complete description of Collins' original algorithm is given in Arnon et al. (1984a). The first phase, *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.* These are then used in the second phase, *lifting*, to build the CAD incrementally. 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. The output for each cell consists of *sections* (where a polynomial vanishes) and *sectors*

(the regions between). Together these form a *stack* over the cell, and taking the union of these stacks gives the CAD of $\mathbb{R}^2$. This is repeated until a CAD of $\mathbb{R}^n$ is produced.

To conclude that a CAD produced in this way is sign-invariant we need delineability. A polynomial is *delineable* in a cell if the portion of its zero set in the cell consists of disjoint sections. A set of polynomials are *delineable* in a cell if each is delineable and the sections of different polynomials in the cell are either identical or disjoint. The projection operator used must be defined so that over each cell of a sign-invariant CAD for projection polynomials in r variables, the polynomials in $r+1$ variables are delineable.

The output of this and subsequent CAD algorithms (including the one presented in this paper) depends heavily on the variable ordering. We usually work with polynomials in $\mathbb{Z}[x_1, \ldots, x_n]$ with the variables, $\mathbf{x}$, in ascending order (so we first project with respect to x_n and continue to reach univariate polynomials in x_1). The *main variable* of a polynomial (mvar) is the greatest variable present with respect to the ordering.

Developments to CAD since Collin's original treatment include the following:

- Refinements to the projection operator (Hong, 1990; McCallum, 1988, 1998; Brown, 2003), reducing the number of projection polynomials and hence cells in the CAD.
- Algorithms to identify the adjacency of cells in a CAD (Arnon et al., 1984b, 1988) and following from this the idea of clustering (Arnon, 1988) to minimise the lifting.
- Partial CAD, introduced by Collins and Hong (1991), where the structure of F is used to lift less of the decomposition of $\mathbb{R}^k$ to $\mathbb{R}^n$, if it is sufficient to deduce ψ.
- The theory of equational constraints, (McCallum, 1999, 2001; Brown and McCallum, 2005), also aiming to deduce ψ itself, this time using more efficient projections.
- The complexity theory of CAD (Brown and Davenport, 2007; Davenport and Heintz, 1988) demonstrating that it is doubly exponential in the number of variables.
- The use of certified numerics in the lifting phase to minimise the amount of symbolic computation required (Strzeboński, 2006; Iwane et al., 2009).
- CAD via Triangular Decomposition (Chen et al., 2009b) (which is used for MAPLE's inbuilt CAD command). This algorithm first constructs a decomposition of complex space and then refines to a CAD. A recent improvement (Chen and Maza, 2012), proceeds incrementally (by polynomial) allowing also for the use of equational constraints.

1.2. TTICAD

Brown (1998) defined a *truth-invariant CAD* as one for which a formula had invariant truth value on each cell, constructed by refining sign-invariant CADs whilst maintaining this property. Some of the developments discussed in the previous section such as partial CAD and the use of equational constraints can be viewed as methods to produce truth-invariant CADs directly. We now define a new and related type of CAD, the topic of this paper.

Definition 1. Let $\{\phi_i\}_{i=1}^t$ be a list of QFFs. We say a cylindrical algebraic decomposition $\mathcal{D}$ is a *Truth Table Invariant* CAD for the list (TTICAD) if the Boolean value of each ϕ_i is constant (either true or false) on each cell of $\mathcal{D}$.

A sign-invariant CAD for the set of polynomials occurring in a list of formulae would clearly be a TTICAD for the list. However, we aim to produce an algorithm that will construct smaller TTICADs for many input lists of formulae. We will achieve this using the theory of equational constraints, first suggested by Collins (1998) with the key theory developed by McCallum (1999).

Definition 2. Suppose some quantified formula is given:

$$\phi^* = (Q_{k+1}x_{k+1}) \cdots (Q_n x_n)\phi(\mathbf{x}).$$

where the Q_i are quantifiers and ϕ is quantifier free. An equation $f = 0$ is an *equational constraint* of ϕ^* if $f = 0$ is logically implied by ϕ (the quantifier-free part of ϕ^*). Such a constraint may be either explicit (an atom of the formula) or otherwise implicit.

In Sections 2 and 3 we will develop a theory allowing the construction of TTICADs which can use the presence of equational constraints to reduce the number of cells required in comparison to a sign-invariant CAD. There are two reasons to use this theory.

(1) *As a tool to build a truth-invariant CAD efficiently:* If a parent formula ϕ^* is built from the formulae $\{\phi_i\}$ then any TTICAD for $\{\phi_i\}$ is also a truth-invariant for ϕ^*.

We note that for such a formula a TTICAD may need to contain more cells than a truth-invariant CAD. For example, consider a cell in a truth-invariant CAD for $\phi^* = \phi_1 \vee \phi_2$ within which ϕ_1 is always true. If ϕ_2 changed truth value in such a cell then it would need to be split in order to achieve a TTICAD, but this is unnecessary for a truth-invariant CAD of ϕ^*.

Nevertheless, we find that our TTICAD theory is often able to produce smaller truth-invariant CADs than any other available approach. We demonstrate the savings offered via worked examples introduced in the next subsection.

(2) *When given a problem for which truth table invariance is required:* That is, a problem for which the list of formulae are not derived from a larger parent formula and thus there is no truth-invariant CAD which would suffice.

For example, decomposing complex space according to a set of branch cuts for the purpose of algebraic simplification (Bradford and Davenport, 2002; Phisanbut et al., 2010, etc.). The idea is to represent each branch cut as a semi-algebraic set to give input admissible to CAD, (recent progress on this has been described by England et al. (2013)). Then a TTICAD for the list of formulae these sets define provides the necessary decomposition. Examples 19 and 21 are from this class.

1.3. Worked examples

To demonstrate our ideas we will provide details for the following worked examples. Assume we have the variable ordering $x \prec y$ (meaning 1-dimensional CADs are with respect to x) and consider the following polynomials, graphed in Figure 1.

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

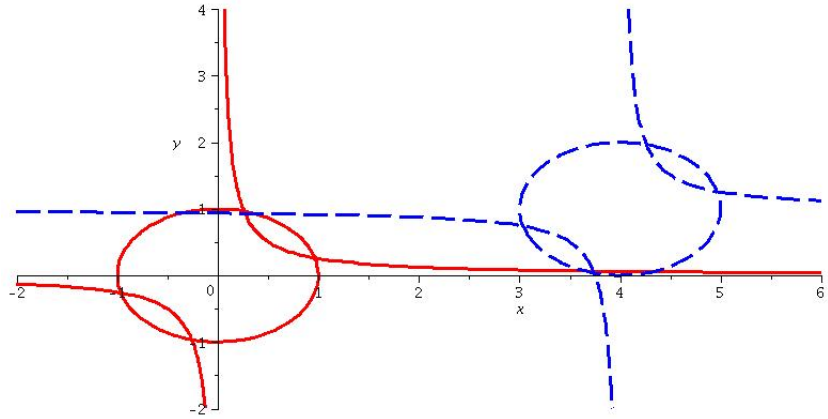
We first wish to find the regions of $\mathbb{R}^2$ where the following formula is true:

$$\Phi := (f_1 = 0 \wedge g_1 < 0) \vee (f_2 = 0 \wedge g_2 < 0)\,. \tag{2}$$

Both QEPCAD (Brown, 2003) and MAPLE 16 (Chen et al., 2009b) produce a sign-invariant CAD for the polynomials with 317 cells.

At first glance it seems that the theory of equational constraints is not applicable here as neither $f_1 = 0$ nor $f_2 = 0$ is logically implied by Φ. However, while there is no explicit equational constraint we can observe that $f_1 f_2 = 0$ is an *implicit* constraint of Φ. Using QEPCAD with this declared gives a CAD with 249 cells. Later, in Section 2.4 we demonstrate how a TTICAD with 105 cells can be produced.

Fig. 1. The polynomials from Section 1.3. The solid lines are f_1 and g_1 while the dashed lines are f_2 and g_2.



We also consider the related problem of identifying where

$$\Psi := (f_1 = 0 \wedge g_1 < 0) \vee (f_2 > 0 \wedge g_2 < 0) \tag{3}$$

is true. As above, we could use a sign-invariant CAD with 317 cells. But this time there is no implicit equational constraint and so the previous approach is not applicable. In Section 2.4 we demonstrate how a TTICAD with 183 cells can be produced.

1.4. Contributions and plan of the paper

In (Bradford et al., 2013a) the authors of the present paper derived an algorithm to efficiently produce TTICADs for a list of formulae in which each had an explicit equational constraint. In the present paper we have extended that work to apply for a general list of formulae, i.e. it is no longer required that each formula has an equational constraint. The new algorithm is able to take advantage of any equational constraints which are present, and in general will produce smaller CADs than the previous theory so long as at least one is present. This means that a TTICAD can now be produced for Ψ in Section 1.3 as well as Φ. This extension is important since if at least one sub-formula does not have an equational constraint then there can be no overall implicit equational constraint meaning that the previous theory of equational constraints is not applicable and hence the comparative benefit of TTICAD is greatly increased.

In Section 2 we develop the theory of a reduced projection operator for producing TTICADs and in Section 3 we present our new algorithm and a proof of its correctness. These sections are structured similarly to Sections 2 and 3 in (Bradford et al., 2013a) but with the theory extended to consider the wider class of problems. In Section 4 we give details of our Maple implementation of this theory. Next in Section 5 we discuss how the theory of reduced projection operators can also be used to make improvements in the lifting phase, which is also true for the theory of a single equational constraint but had not been noted before. Then in Section 6 we discuss the issues of problem formulation for the TTICAD algorithm and present heuristics to assist with this, work that was previously discussed by Bradford et al. (2013b) which looked at the wider issues of formulating problems for CAD. In Section 7 we present experimental results for this implementation, extending the results in (Bradford et al., 2013a) to include a wider set of problems. Finally in Section 8 we give our conclusions and ideas for future work.

2. Projection Operators and their Theorems

2.1. The theory of reduced projection operators

We use two key theorems from McCallum's work on projection and equational constraints. Both theorems use CADs which are not just sign-invariant but have the stronger property of order-invariance. A CAD is *order-invariant* with respect to a set of polynomials if each polynomial has constant order of vanishing within each cell.

Let P be the projection operator defined by McCallum (1988), which extracts coefficients, discriminant and cross resultants from a set of polynomials. We assume the usual trivial simplifications such as removal of constants, exclusion of entries identical to a previous entry (up to constant multiple), and using only the necessary coefficients. Recall that a set $A \subset \mathbb{Z}[\mathbf{x}]$ is an *irreducible basis* if the elements of A are of positive degree in the main variable, irreducible and pairwise relatively prime.

The main theorem underlying the use of P follows.

Theorem 3 (McCallum (1998))**.** *Let A be an irreducible basis in $\mathbb{Z}[\mathbf{x}]$ and let S be a connected submanifold of $\mathbb{R}^{n-1}$. Suppose each element of $P(A)$ is order-invariant in S.*

Then each element of A either vanishes identically on S or is analytic delineable on S, (a slight variant on traditional delineability, see (McCallum, 1998)). Further, the sections of A not identically vanishing are pairwise disjoint, and each element of A not identically vanishing is order-invariant in such sections.

Theorem 3 means that we can use this projection operator in place of that of Collins to produce order-invariant CADs so long as none of the projection polynomials with main variable x_k vanishes on a cell of the CAD of $\mathbb{R}^{k-1}$; a condition that can be checked when lifting.

The main mathematical result underlying the reduction of P in the presence of an equational constraint f is as follows.

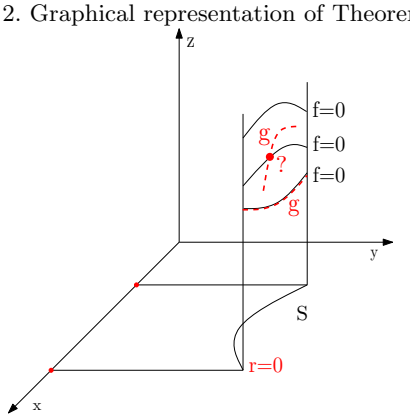
Theorem 4 (McCallum (1999))**.** *Let $f(\mathbf{x}), g(\mathbf{x})$ be integral polynomials with positive degree in x_n, let $r(x_1, \ldots, x_{n-1})$ be their resultant, and suppose $r \neq 0$. Let S be a connected subset of $\mathbb{R}^{n-1}$ such that f is delineable on S and r is order-invariant in S.*

Then g is sign-invariant *in every section of f over S.*

Figure 2 gives a graphical representation of the question answered by Theorem 4. Here we consider polynomials $f(x, y, z)$ and $g(x, y, z)$ of positive degree in z whose resultant r is non-zero, and a connected subset $S \subset \mathbb{R}^2$ in which r is order-invariant. We further suppose that f is delineable on S (noting that Theorem 3 with $n = 3$ and $A = \{f\}$ provides sufficient conditions for this). We ask whether g is sign-invariant in the sections of f over S. Theorem 4 answers this question affirmatively: the real variety of g either aligns with a given section of f exactly (as for the bottom section of f in Figure 2), or has no intersection with such a section (as for the top). The situation at the middle section of f cannot happen.

Theorem 4 thus suggests a reduction of the projection operator P relative to an equational constraint $f = 0$ for the first projection step, i.e. for the first step it is sufficient to take only the $P(f)$ together with the resultants of f with the non-equational constraints. This operator was presented in (McCallum, 1999) along with a generalisation of Theorem 3 (which followed from Theorems 3 and 4) verifying its use. We call CADs produced this way *invariant with respect to an equational constraint.*

Fig. 2. Graphical representation of Theorem 4



2.2. A projection operator for TTICAD

In (McCallum, 1999) the central concept is the reduced projection of a set A of integral polynomials relative to a nonempty subset $E \subseteq A$ (where A is derived from the input polynomials and E from those which are equational constraints). The operator was

$$P_E(A) = P(E) \cup \{\text{res}_{x_n}(f, g) \mid f \in E, g \in A, g \notin E\}, \tag{4}$$

where P is the original McCallum projection operator given by

$$P(A) = \{\text{coeffs}(f), \text{disc}(f), \text{res}_{x_n}(f, g) \mid f, g \in A, f \neq g\}. \tag{5}$$

We will use an extension of this idea to define a projection of a list of sets of polynomials (derived from a list of formulae), some of which may have subsets (derived from equational constraints). For simplicity in (McCallum, 1999) the concept is first defined for the case when A is an irreducible basis and we emulate this approach, generalising for use in other cases by considering contents and irreducible factors of positive degree when describing the algorithm in Section 3.

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 subsets $E_i \subseteq A_i$. Put $A = \bigcup_{i=1}^t A_i$ and $E = \bigcup_{i=1}^t E_i$. Note that we use the convention of uppercase Roman letters for sets of polynomials and calligraphic letters for lists of these.

Definition 5. We define the *reduced projection of* $\mathcal{A}$ *with respect to* $\mathcal{E}$ as

$$P_{\mathcal{E}}(\mathcal{A}) := \bigcup_{i=1}^t P_{E_i}(A_i) \cup \text{RES}^{\times}(\mathcal{E}) \tag{6}$$

where $\text{RES}^{\times}(\mathcal{E})$ is the cross resultant set

$$\text{RES}^{\times}(\mathcal{E}) = \{\text{res}_{x_n}(f, \hat{f}) \mid \exists\, i, j \text{ such that } f \in E_i, \hat{f} \in E_j, i < j, f \neq \hat{f}\}.$$

For later use, we define the set of the polynomials in $P(A)$ but excluded from $P_{\mathcal{E}}(\mathcal{A})$.

Definition 6. We define the *excluded projection polynomials* of (A_i, E_i) by

$$\begin{aligned} \text{ExclP}_{E_i}(A_i) &:= P(A_i) \setminus P_{E_i}(A_i) \\ &= \{\text{coeffs}(g), \text{disc}_{x_n}(g), \text{res}_{x_n}(g, \hat{g}) \mid g, \hat{g} \in A_i \setminus E_i, g \neq \hat{g}\}. \end{aligned} \tag{7}$$

The total set of excluded polynomials from $P(A)$ will not only include all the entries of the $\text{ExclP}_{E_i}(A_i)$ but also the missing cross resultants of polynomials in $A_i \setminus E_i$ with polynomials from $A_j \neq A_i$. However, this larger set is not used in this paper.

The following is an analogue of Theorem 2.3 of (McCallum, 1999) and provides the foundation for Algorithm 1.

Theorem 7. *Let S be a connected submanifold of $\mathbb{R}^{n-1}$. Suppose each element of $P_{\mathcal{E}}(\mathcal{A})$ is order invariant in S. Then each $f \in E$ either vanishes identically on S or is analytically delineable on S, the sections over S of the $f \in E$ which do not vanish identically are pairwise disjoint, and each element $f \in E$ which does not vanish identically is order-invariant in such sections.*

Moreover*, for each i, in $1 \leq i \leq t$ every $g \in A_i \setminus E_i$ is sign-invariant in each section over S of every $f \in E_i$ which does not vanish identically.*

Proof. The crucial observation is that $P(E) \subseteq P_{\mathcal{E}}(\mathcal{A})$. To see this, recall equation (6) and note that we can write

$$P(E) = \bigcup_i P(E_i) \cup \mathrm{RES}^{\times}(\mathcal{E}).$$

We can therefore apply Theorem 3 to the set E and obtain the first three conclusions immediately. There remains the final conclusion to prove. Let i be in the range $1 \leq i \leq t$, let $g \in A_i \setminus E_i$ and let $f \in E_i$; suppose f does not vanish identically on S. Now $\mathrm{res}_{x_n}(f, g) \in P_{\mathcal{E}}(\mathcal{A})$, and so is order-invariant in S by hypothesis. Further, we already concluded that f is delineable. Therefore by Theorem 4, g is sign-invariant in each section of f over S. □

In Section 3 we use Theorem 7 as the key tool for our implementation of TTICAD, so long as no $f \in E$ vanishes identically on the lower dimensional manifold, S. A polynomial f in r variables that vanishes identically at a point $\alpha \in \mathbb{R}^{r-1}$ is said to be *nullified* at α.

The theory of this subsection appears identical to the work in (Bradford et al., 2013a). The difference is in the application of the theory to build a TTICAD algorithm (as described in full in Section 3). We suppose that the input is a list of QFFs, $\{\phi_i\}$, with each A_i defined from the polynomials in each ϕ_i. In (Bradford et al., 2013a) there was an assumption that each of these formulae had a designated equational constraint $f_i = 0$ from which the subsets E_i are defined. This assumption does not apply in this paper. Instead we define E_i to be a basis for $\{f_i\}$ if there is such a designated equational constraint and otherwise define $E_i = A_i$ to ensure information is taken from QFFs which have no equational constraint. That is, to extend our TTICAD algorithm for such cases we needed to treat all the polynomials in QFFs with no equational constraint with the importance otherwise reserved for equational constraints.

2.3. Comparison with using a single implicit equational constraint

It is clear that the reduced projection $P_{\mathcal{E}}(\mathcal{A})$ will lead to fewer (or the same) projection polynomials than using the full projection P. However, a comparison with the existing theory of equational constraints requires a little analysis.

First, we note that the TTICAD theory is applicable to a sequence of formulae while the equational constraints theory is applicable only to a single formula. Hence if the truth value of each QFF is needed then TTICAD is the only option; a truth-invariant CAD for a parent formula will not necessarily suffice. Second we note that even if the sequence do form a parent formula then it must be this parent formula which has an overall equational constraint logically implied (not just each sub-formulae) to apply (McCallum, 1999) while the TTICAD theory is applicable even if this is not the case.

Now we consider the situation where both theories are applicable. Suppose we have a sequence of formulae which belong to a parent formula for which a truth-invariant CAD is desired. Further suppose each of these has an equational constraint and thus the parent formula has an implicit equational constraint, (their product). In the context of Section 1.2 this corresponds to using $\prod_i f_i$ as the equational constraint. Such a CAD is sufficient to solve the problem so it is natural to ask if the TTICAD theory is required.

The implicit equational constraint approach would correspond to using the reduced projection $P_E(A)$ of (McCallum, 1999), with $E = \cup_i E_i$ and $A = \cup_i A_i$. We make the simplifying assumption that A is an irreducible basis. In general $P_{\mathcal{E}}(\mathcal{A})$ will still contain fewer polynomials than $P_E(A)$ since $P_E(A)$ contains all resultants $\text{res}(f, g)$ where $f \in E_i, g \in A_j$ (and $g \notin E$), while $P_{\mathcal{E}}(\mathcal{A})$ contains only those with $i = j$ (and $g \notin E_i$). Thus even in situations where the previous theory applies there is advantage in using the new theory. These savings are made clear when considering the worked examples in the next subsection.

2.4. Worked Examples

In Section 3 we define an algorithm for producing TTICADs. First we illustrate the savings with our worked examples from Section 1.3, which satisfy the simplifying assumptions made in Section 2.2.

First we consider Φ from equation (2). In the notation introduced above we have:

$$A_1 := \{f_1, g_1\}, \qquad E_1 := \{f_1\};$$
$$A_2 := \{f_2, g_2\}, \qquad E_2 := \{f_2\}.$$

We construct the reduced projection sets for each ϕ_i,

$$P_{E_1}(A_1) = \left\{x^2 - 1, x^4 - x^2 + \tfrac{1}{16}\right\},$$
$$P_{E_2}(A_2) = \left\{x^2 - 8x + 15, x^4 - 16x^3 + 95x^2 - 248x + \tfrac{3841}{16}\right\}$$

and the cross-resultant set

$$\text{Res}^{\times}(\mathcal{E}) = \{\text{res}_y(f_1, f_2)\} = \{68x^2 - 272x + 285\}.$$

$P_{\mathcal{E}}(\mathcal{A})$ is then the union of these three sets. In Figure 3 we plot the polynomials (solid curves) and identify the 12 real solutions of $P_{\mathcal{E}}(\mathcal{A})$ (solid vertical lines). We can see the solutions align with the asymptotes of the f_i's and the important intersections (those of f_1 with g_1 and f_2 with g_2).

If we were to instead use a projection operator based on an implicit equational constraint $f_1 f_2 = 0$ then in the notation above we would construct $P_E(A)$ from $A = \{f_1, f_2, g_1, g_2\}$ and $E = \{f_1, f_2\}$. This set provides an extra 4 solutions (the dashed vertical lines) which align with the intersections of f_1 with g_2 and f_2 with g_1. Finally, if we were to consider $P(A)$ then we gain a further 4 solutions (the dotted vertical lines) which align with the intersections of g_1 and g_2 and the asymptotes of the g_i's. In Figure 4 we magnify a region to show explicitly that the point of intersection between f_1 and g_1 is identified by $P_{\mathcal{E}}(\mathcal{A})$, while the intersections of g_2 with both f_1 and g_1 are ignored. The 1-dimensional CAD produced using $P_{\mathcal{E}}(\mathcal{A})$ has 25 cells compared to 33 when using $P_E(A)$ and 41 when using $P(A)$. However, it is important to note that this reduction is amplified after lifting (using Theorem 7 and and Algorithm 1). The 2-dimensional TTICAD has 105 cells and the sign-invariant CAD has 317. Using QEPCAD to build a CAD invariant with respect to the implicit equational constraint we have 249 cells

Fig. 3. The polynomials from Φ in equation (2) along with the roots of $P_{\mathcal{E}}(\mathcal{A})$ (solid lines), $P_E(A)$ (dashed lines) and $P(A)$ (dotted lines).

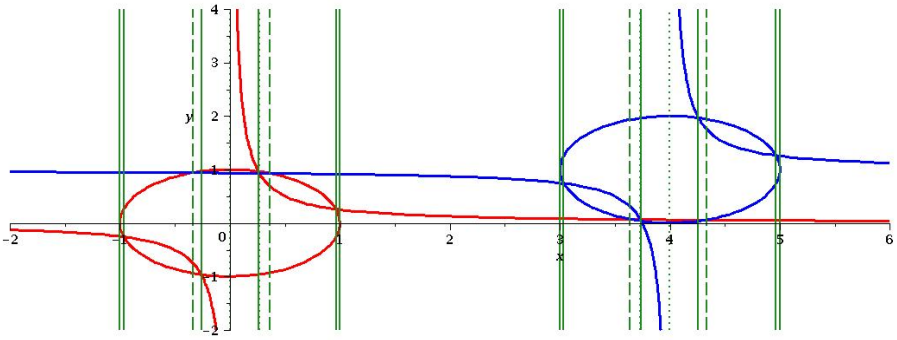


Fig. 4. Magnified region of Figure 3

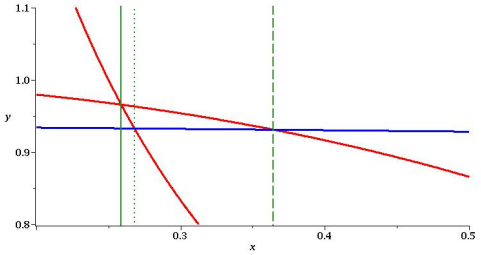


Fig. 5. The polynomials from Ψ in equation (3) along with the roots of $P_{\mathcal{E}}(\mathcal{A})$.

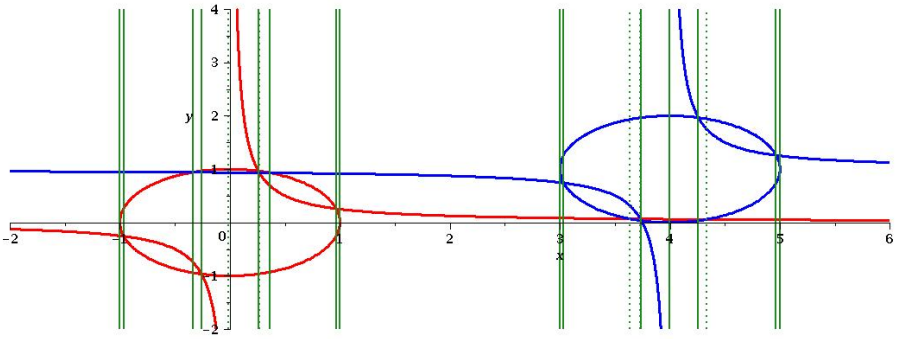
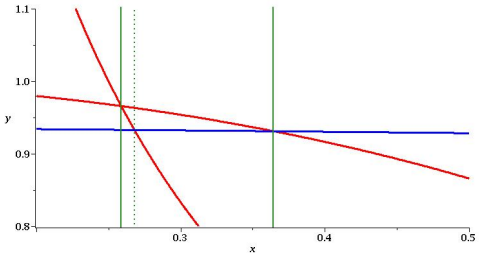


Fig. 6. Magnified region of Figure 5



Next we consider determining the truth of Ψ from equation (3). This time

$$A_1 := \{f_1, g_1\},\ E_1 := \{f_1\},$$
$$A_2 := \{f_2, g_2\},\ E_2 := \{f_2, g_2\},$$

and so $P_{E_1}(A_1)$ is as above but $P_{E_2}(A_2)$ contains an extra polynomial $x-4$ (the coefficient of y in g_2). The cross-resultant set $\mathrm{RES}^{\times}(\mathcal{E})$ also contains an extra polynomial,

$$\mathrm{res}_y(f_1, g_2) = x^4 - 8x^3 + 16x^2 + \tfrac{1}{2}x - \tfrac{31}{16}.$$

These two extra polynomials provide three extra real roots and hence the 1-dimensional CAD produced using $P_{\mathcal{E}}(\mathcal{A})$ this time has 31 cells.

In Figure 5 we again graph the four curves this time with solid vertical lines highlighting the real solutions of $P_{\mathcal{E}}(\mathcal{A})$. By comparing with Figure 3 we see that more points in the CAD of $\mathbb{R}^1$ have been identified for the TTICAD of Ψ than the TTICAD of Φ (15 instead of 12) but that there is still a saving over the sign-invariant CAD (which had 20, the five extra solutions indicated by dotted lines). The lack of an equational constraint in the second clause has meant that the asymptote of g_2 and its intersections with f_1 have been identified. However, note that the intersections of g_1 with f_2 and g_2 and have not been identified. Figure 6 magnifies a region of Figure 5 to make this clearer (in comparison with Figure 4 the dashed line has become solid, while the dotted line remains unidentified by the TTICAD).

Note that we are unable to use the original theory of equational constraints for this problem as there is no polynomial logically implied (either explicitly or implicitly) by Ψ. Hence there are no dashed lines and the comparison is between the sign-invariant CAD with 317 cells or the TTICAD, which for this example has 183 cells.

3. Algorithm

3.1. Description and Proof

We describe carefully Algorithm 1. This will create a TTICAD of $\mathbb{R}^n$ for a list of quantifier free formulae $\{\phi_i\}_{i=1}^t$ in variables $\mathbf{x} = x_1 \prec x_2 \prec \cdots \prec x_n$, where each ϕ_i has at most one designated equational constraint, (although there may be other non-designated equational constraints), $f_i = 0$ of positive degree.

It uses a subalgorithm `CADW`, which was validated by McCallum (1998). The input of `CADW` is: r, a positive integer and A, a set of r-variate integral polynomials. The output is a boolean w which if true is accompanied by an order-invariant CAD for A (represented as a list of indices I and sample points S).

Let A_i be the set of all polynomials occurring in ϕ_i. If ϕ_i has a designated equational constraint then put $E_i = \{f_i\}$ and if not put $E_i = A_i$. Let $\mathcal{A}$ and $\mathcal{E}$ be the lists of the A_i and E_i respectively. Our algorithm effectively defines the reduced projection of $\mathcal{A}$ with respect to $\mathcal{E}$ in terms of the special case of this definition from the previous section. The definition amounts to the following:

$$P_{\mathcal{E}}(\mathcal{A}) := C \cup P_{\mathcal{F}}(\mathcal{B})$$

where C is the set of contents of all the elements of all the A_i, $\mathcal{B}$ is the list $\{B_i\}_{i=1}^t$, such that B_i is the finest squarefree basis for the set $\mathrm{prim}(A_i)$ of primitive parts of elements of A_i which have positive degree, and $\mathcal{F}$ is the list $\{F_i\}_{i=1}^t$, such that F_i is the finest

squarefree basis for prim(E_i). (The reader will notice that this notation and the definition of $P_{\mathcal{E}}(\mathcal{A})$ here is analogous to the work in Section 5 of (McCallum, 1999).)

We shall prove that, provided the input satisfies the condition of well-orientedness given in Definition 9, the output of Algorithm 1 is indeed a TTICAD for $\{\phi_i\}$. We first recall the more general notion of well-orientedness from (McCallum, 1998). The boolean output of `CADW` is false if the input set was not well-oriented in this sense.

Definition 8. A set A of n-variate polynomials is said to be *well oriented* if whenever $n > 1$, every $f \in \text{prim}(A)$ is nullified by at most a finite number of points in $\mathbb{R}^{n-1}$, and (recursively) $P(A)$ is well-oriented.

This condition is required for `CADW` since the validity of this algorithm relies on Theorem 3 which holds only when polynomials do not vanish identically. The conditions allows for a finite number of these nullifications since this indicates a problem on a zero cell, that is a single point. In such cases it is possible to replace the nullified polynomial by a so called *delineating polynomial* which is not nullified and can be used in place to ensure the delineability of the other. The use of these is part of the verified algorithm `CADW` (McCallum, 1998) and they are studied in detail in (Brown, 2005).

We now define our new notion of well-orientedness for the lists of sets $\mathcal{A}$ and $\mathcal{E}$.

Definition 9. 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 $P_{\mathcal{F}}(\mathcal{B})$ is well-oriented in the sense of Definition 8.

It is clear than Algorithm 1 terminates. We now prove that it is correct using the theory developed in Section 2.

Theorem 10. *The output of Algorithm 1 is as specified.*

Proof. We must show that when the input is well-oriented the output is a TTICAD, (each ϕ_i has constant truth value in each cell of $\mathcal{D}$), and **FAIL** otherwise.

If the input was univariate then it is trivially well-oriented. The algorithm will construct a CAD $\mathcal{D}$ of $\mathbb{R}^1$ using the roots of the irreducible factors of the polynomials in E (steps 6 to 7). At each 0-cell all the polynomials in each ϕ_i trivially have constant signs, and hence every ϕ_i has constant truth value. In each 1-cell no equational constraint can change sign and so every ϕ_i has constant truth value $false$, unless there are no equational constraints in any clause. In this second case the algorithm would have constructed a CAD using all the polynomials and hence on each 1-cell no polynomial changes sign and so each clause has constant truth value.

From now on suppose $n > 1$. If $\mathfrak{P} = C \cup P_{\mathcal{F}}(\mathcal{B})$ is not well-oriented in the sense of Definition 8 then `CADW` returns w' as false. In this case the input is not well oriented in the sense of Definition 9 and Algorithm 1 correctly returns **FAIL** in step 17. Otherwise, we have $w' = true$ with I' and S' specifying a CAD, $\mathcal{D}'$, which is order-invariant with respect to $\mathfrak{P}$ (by the correctness of `CADW`, as proved in (McCallum, 1998)). Let c, a submanifold of $\mathbb{R}^{n-1}$, be a cell of $\mathcal{D}'$ and let α be its sample point.

We suppose first that the dimension of c is positive. If any polynomial $f \in E$ vanishes identically on c then the input is not well oriented in the sense of Definition 9 and the algorithm correctly returns **FAIL** at step 24. Otherwise, we know that the input list was certainly well-oriented. Since no polynomial $f \in E$ vanishes then no element of the basis

Algorithm 1: TTICAD Algorithm

Input : A list of quantifier-free formulae $\{\phi_i\}_{i=1}^t$ in variables $x_1, \ldots, x_n$. Each ϕ_i has at most one designated equational constraint $f_i = 0$.

Output: Either • $\mathcal{D}$: A CAD of $\mathbb{R}^n$ (described by lists I and S of cell indices and sample points) which is truth table invariant for the list of input formulae; or • **FAIL**: If $\mathcal{A}$ is not well-oriented with respect to $\mathcal{E}$) (Def 9).

for $i = 1 \ldots t$ **do**
  If there is no designated equational constraint then set $E_i := A_i$ and otherwise set $E_i := \{f_i\}$;
  Compute the finest squarefree basis F_i for prim(E_i);
Set $F \leftarrow \cup_{i=1}^t F_i$;
if $n = 1$ **then**
  Isolate the real roots of the polynomials in F and thus form cell indices and sample points for a CAD of $\mathbb{R}$;
  return I and S for $\mathcal{D}$;
else
  for $i = 1 \ldots t$ **do**
    Extract the set A_i of polynomials in ϕ_i;
    Compute the set C_i of contents of the elements of A_i;
    Compute the set B_i, the finest squarefree basis for prim(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: $w', I', S' :=$ `CADW`$(n-1, \mathfrak{P})$;
  if $w' = false$ **then**
    return FAIL (since $\mathfrak{P}$ is not well oriented) ;
  $I \leftarrow \emptyset$; $S \leftarrow \emptyset$;
  for *each cell* $c \in \mathcal{D}'$ **do**
    $L_c \leftarrow \{\}$;
    for $i = 1, \ldots t$ **do**
      if *any* $f \in E_i$ *is nullified on* c **then**
        if $\dim(c) > 0$ **then**
          return FAIL (since $\{\phi_i\}_{i=1}^t$ is not well oriented) ;
        else
          $L_c \leftarrow L_c \cup B_i$;
      else
        $L_c \leftarrow L_c \cup F_i$;
    Generate a stack over c using L_c: construct cell indices and sample points for the stack over c of the polynomials in L_c, adding them to I and S ;
  return I and S for $\mathcal{D}$;

F vanishes identically on c either. Hence, by Theorem 7, applied with $\mathcal{A} = \mathcal{B}$ and $\mathcal{E} = \mathcal{F}$, each element of F is delineable on c, and the sections over c of the elements of F are pairwise disjoint. Thus the sections and sectors over c of the elements of F comprise a stack Σ over c. Furthermore, the last conclusion of Theorem 7 assures us that, for each i, every element of $B_i \setminus F_i$ is sign-invariant in each section over c of every element of F_i. Let $1 \leq i \leq t$. We shall show that each ϕ_i has constant truth value in both the sections and sectors of Σ.

If ϕ_i has a designated equational constraint then let f_i denote the constraint polynomial; otherwise let f_i denote an arbitrary element of A_i.

Consider first a section σ of Σ. Now f_i is a product of its content $\text{cont}(f_i)$ and some elements of the basis F_i. But $\text{cont}(f_i)$, an element of $\mathfrak{P}$, is sign-invariant (indeed order-invariant) in the whole cylinder $c \times \mathbb{R}$ and hence, in particular, in σ. Moreover all of the elements of F_i are sign-invariant in σ, as was noted previously. Therefore f_i is sign-invariant in σ. If ϕ_i has no constraint (and so f_i denotes an arbitrary element of A_i) then this implies that ϕ_i has constant truth value in σ. So consider from now on the case in which $f_i = 0$ is the designated constraint polynomial of ϕ_i.

If f_i is positive or negative in σ then ϕ_i has constant truth value $false$ in σ. So suppose that $f_i = 0$ throughout σ. It follows that σ must be a section of some element of the basis F_i. Let $g \in A_i \setminus E_i$ be a non-constraint polynomial in A_i. Now, by the definition of B_i, we see g can be written as

$$g = \text{cont}(g) h_1^{p_1} \cdots h_k^{p_k}$$

where $h_j \in B_i, p_j \in \mathbb{N}$. But $\text{cont}(g)$, in $\mathfrak{P}$, is sign-invariant (indeed order-invariant) in the whole cylinder $c \times \mathbb{R}$, and hence in particular in σ. Moreover each h_j is sign-invariant in σ, as was noted previously. Hence g is sign-invariant in σ. (Note that in the case where g does not have main variable x_n then $g = \text{cont}(g)$ and the conclusion still holds). Since g was an arbitrary element of $A_i \setminus E_i$, it follows that all polynomials in A_i are sign-invariant in σ, hence that ϕ_i has constant truth value in σ.

Next consider a sector σ of the stack Σ, and notice that at least one such sector exists. As observed above, $\text{cont}(f_i)$ is sign-invariant in c, and f_i does not vanish identically on c. Hence $\text{cont}(f_i)$ is non-zero throughout c. Moreover each element of the basis F_i is delineable on c. Hence f_i is nullified by no point of c. It follows from this that the algorithm does not return **FAIL** during the lifting phase. It follows also that $f_i \neq 0$ throughout σ. Hence ϕ_i has constant truth value $false$ in σ.

It remains to consider the case in which the dimension of c is 0. In this case the roots of the polynomials in the lifting set L_c constructed by the algorithm determine a stack Σ over c. Each ϕ_i trivially has constant truth value in each section (0-cell) of this stack, and the same can routinely be shown for each sector (1-cell) of this stack. □

3.2. TTICAD via the ResCAD Set

In Algorithm 1 the lifting stage (steps 18 to 29) varies according to whether any $f \in E$ is nullified. When this does not occur there is an alternative implementation of TTICAD which would be simple to introduce into existing CAD algorithms.

Define the *ResCAD Set* of $\{\phi_i\}$ as

$$\mathcal{R}(\{\phi_i\}) = E \cup \bigcup_{i=1}^{t} \{\text{res}_{x_n}(f, g) \mid f \in E_i, g \in A_i, g \notin E_i\}.$$

Theorem 11. *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 $E_i \subseteq A_i$. Then we have*

$$P(\mathcal{R}(\{\phi_i\})) = P_{\mathcal{E}}(\mathcal{A}).$$

The proof is straightforward and so omitted here.

Corollary 12. *If no $f \in E$ is nullified by a point in $\mathbb{R}^{n-1}$ then inputting $\mathcal{R}(\{\phi_i\})$ into any algorithm which produces a sign-invariant CAD using McCallum's projection operator, will result in the TTICAD for $\{\phi_i\}$ produced by Algorithm 1.*

Hence Corollary 12 gives us a simple way to compute TTICADs using existing CAD implementations based on McCallum's approach, such as QEPCAD. However, this cannot be applied as widely as Algorithm 1.

4. Our implementation in Maple

There are various implementations of CAD already available including: MATHEMATICA (Strzeboński, 2006, 2010), QEPCAD (Brown, 2003), `Redlog` (part of the REDUCE system) (Seidl and Sturm, 2003), `RegularChains` (Chen et al., 2009b) (in MAPLE), and `SyNRAC` (Yanami and Anai, 2006) (third party package for MAPLE). However, none of these can (currently) be used to build CADs which guarantee order-invariance, a property required for proving the correctness of our TTICAD algorithm. Hence we have constructed our own CAD implementation in order to obtain experimental results for our ideas.

4.1. ProjectionCAD

Our implementation is a third party MAPLE package which we call `ProjectionCAD`. It gathers together algorithms for producing CADs via projection and lifting to complement the existing CAD commands in MAPLE which use an alternative approach based on the theory of regular chains and triangular decomposition.

All the projection operators discussed in Section 2 have been implemented and so `ProjectionCAD` can produce CADs which are sign-invariant, order-invariant, invariant with respect to a declared equational constraint, and truth table invariant. Stack generation (step 29 in Algorithm 1) is achieved by making use of existing commands in MAPLE from the `RegularChains` package. The `RegularChains` command for stack generation is described fully in Section 5.2 of (Chen et al., 2009b). Stack generation in `ProjectionCAD` uses this, but first processes the input in order to satisfy the assumptions that algorithm has. Namely that the polynomials are co-prime and square-free when evaluated on the cell, (*separate above the cell* in the language of regular chains). This is achieved using other commands from the `RegularChains` library.

Utilising the `RegularChains` code like this means that `ProjectionCAD` can represent and present CADs in the same way. In particular this allows for easy comparison of CADs from the different implementations, the use of existing commands for studying the CADs and the ability to display CADs to the user in the easy to understand `piecewise` representation (Chen et al., 2009a).

A simple example of using the code is shown in Figure 7. The output is as displayed in a MAPLE worksheet, except that the sample points have been replaced by SP for brevity.

Fig. 7. Using `ProjectionCAD` in MAPLE to build a sign-invariant CAD for the unit circle.

```
> f := x^2+y^2-1:
> cad := CADFull([f], vars, method=McCallum, output=piecewise);
```

$$\begin{cases} SP & & x < -1 \\ \begin{cases} SP & y < 0 & \\ SP & y = 0 & x = -1 \\ SP & 0 < y & \end{cases} \\ \begin{cases} SP & y < -\sqrt{-x^2+1} & \\ SP & y = -\sqrt{-x^2+1} & \\ SP & And\left(-\sqrt{-x^2+1} < y, y < \sqrt{-x^2+1}\right) & And\left(-1 < x, x < 1\right) \\ SP & y = +\sqrt{-x^2+1} & \\ SP & \sqrt{-x^2+1} < y & \end{cases} \\ \begin{cases} SP & y < 0 & \\ SP & y = 0 & x = 1 \\ SP & 0 < y & \end{cases} \\ SP & & 1 < x \end{cases}$$

```
> CADNumCellsInPiecewise(cad);
```

13

Unlike QEPCAD, `ProjectionCAD` has an implementation of delineating polynomials (actually the minimal delineating polynomials discussed by Brown (2005)), which means it can solve certain problems not admissible to QEPCAD (see (England, 2013a) for details). It is hence the only implementation that can reproduce the theoretical algorithm `CADW`.

Other notable features of `ProjectionCAD` include commands to present the different formulations of problems for the algorithms and heuristics to help choose between these. For more details on `ProjectionCAD` and the algorithms implemented within see the technical reports (England, 2013a,b). The webpages hosting these also contain the MAPLE code described which may be downloaded for free along with an introductory worksheet demonstrating the functionality. However, to run the code users will need a version of MAPLE with the `RegularChains` CAD implementation (version 16 and up).

4.2. Minimising failure of TTICAD

Algorithm 1 was kept simple to aid readability and understanding. Our implementation does make some refinements not presented formally in Algorithm 1. Most of these are trivial, such as removing constants from the set of projection polynomials or when taking coefficients in order of degree, stopping if the ones already included can be shown not to vanish simultaneously. It is known that the well-orientedness conditions can often be overly cautious. In (Brown, 2005), several cases where non-well oriented input can still lead to an order-invariant CAD are discussed. Similarly here, we can sometimes allow the nullification of an equational constraint on a positive dimensional cell.

Lemma 13. *Let f_i be an equational constraint which vanishes identically on a cell $c \in \mathcal{D}'$ constructed during Algorithm 1. If all polynomials in* $\text{ExclP}_{E_i}(A_i)$ *are constant on c then any $g \in A_i \setminus E_i$ will be delineable over c.*

Proof. Suppose first that A_i and E_i satisfy the simplifying conditions from Section 2.2. Rearranging (7) we see

$$P(A_i) = P_{E_i}(A_i) \cup \text{ExclP}_{E_i}(A_i).$$

However, given the conditions of the lemma, this is equivalent (after the removal of constants which do not affect CAD construction) to $P_{E_i}(A_i)$ on c. So here $P(A_i)$ is a subset of $P_{\mathcal{E}}(\mathcal{A})$ and we can conclude by Theorem 3 that all elements of A_i vanish identically on c or are delineable over c.

In the more general case we can still draw the same conclusion because $P(A_i) = C_i \cup P_{F_i}(B_i) \cup \text{ExclP}_{F_i}(B_i) \subseteq \mathfrak{P}$. □

Hence Lemma 13 allows us to extend Algorithm 1 to deal safely with such cases. Although we cannot conclude sign-invariance we can conclude delineability and so instead of returning failure we can proceed by extending the lifting set L_c to the full set of polynomials in such cases (similar to the case of nullification on a cell of dimension zero dealt with in step 26 of Algorithm 1). In particular, this allows for equational constraints f_i which do not have main variable x_n. Our implementation makes use of this.

Note that the widening of the lifting step here (and also in the case of the zero dimensional cell) is for the generation of the stack over a single cell. The extension is only performed for the necessary cells thus minimising the cell count while maximising the success of the algorithm, as demonstrated in Example 14. Of course, since a polynomial cannot be nullified everywhere such case distinction will certainly decrease the amount of lifting.

Example 14. Consider the polynomials

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

the single formula $f = 0 \wedge g < 0 \wedge h < 0$ and assume the variable ordering $x \prec y \prec z \prec w$. Using the `ProjectionCAD` package we can build a TTICAD with 467 cells for this formula. The induced CAD of $\mathbb{R}^3$, D, 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 fixed to $0, 4$ or belonging to the three intervals splitting at these points.

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

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 occurs because h is included in the lifting set only for the five cells of D where f was nullified.

In theory, we could go further and allow this extension to apply when the polynomials in $\text{ExclP}_{E_i}(A_i)$ are not necessarily all constant, but have no real roots within the cell c. However, identifying such cases would, in general, require answering a separate quantifier elimination question, which may not be trivial, and so this has not yet been implemented.

5. Utilising the existing theory for improved lifting

Consider the case when the input to Algorithm 1 is a single QFF $\{\phi\}$ with a declared equational constraint. In this case the reduced projection operator $P_{\mathcal{E}}(\mathcal{A})$ produces the same polynomials as the operator $P_E(A)$ defined in (McCallum, 1999) and so one may expect the TTICAD produced to be the same as the CAD invariant with respect to an equational constraint produced by an implementation of (McCallum, 1999) such as QEPCAD. In practice this is not the case. The main reason is that Algorithm 1 makes use of the equational constraints theory in the lifting stage as well as the projection phase.

McCallum (1999) discussed how the theory of a reduced projection operator would improve the projection phase of CAD, by creating fewer projection polynomials. The only modification to the lifting phase of Collins' CAD algorithm described was the need to check the well-orientedness condition of Definition 8, (as is also the case when using the original McCallum operator for producing sign-invariant CADs).

In this section we note two subtleties in the lifting phase of Algorithm 1 which result in efficiencies that could be replicated for use with the original theory of equational constraints. In fact, the `ProjectionCAD` package (England, 2013b) discussed in Section 4.1 now has commands for building CADs invariant with respect to a single equational constraint which make use of these efficiencies.

5.1. A finer check for well-orientedness

Theorem 2.3 of (McCallum, 1999) justifies the use of $P_E(A)$, the original reduced projection operator. The proof uses Theorem 3 to conclude a CAD is sign-invariant for the equational constraint and Theorem 4 to conclude that the other constraints are sign-invariant in the cells which are sections of f. The conditions of Theorems 3 mean that this result holds only when the equational constraint and the other projection polynomials found by recursively applying P) have a finite number of nullification points. Theorem 4 requires that the resultants of the equational constraint with the other constraints have no nullification points which is guaranteed by the input satisfying Definition 8, the condition used in (McCallum, 1999). However, this also requires that all projection polynomials, including the non-equational constraints themselves, have no nullification points and hence is stronger than required. In Algorithm 1, step 22 only checks for nullification of the polynomials in E_i (in this case meaning only the equational constraint). Hence this algorithm is checking the necessary conditions but not whether the non-equational constraints (in the main variable) are nullified.

Example 15. Assume the variable ordering $x \prec y \prec z \prec w$ and 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 this using a sign-invariant CAD with 557 cells but it is more efficient to make use of the equational constraint. Our implementation of Algorithm 1 produces a CAD with 165 cells while declaring the equational constraint in QEPCAD results in a CAD with 221 cells (the higher number due to issues discussed in subsection 5.2). QEPCAD also returns an error message

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

indicating the output may not be correct. The error message was triggered by the nullification of g when $x = y = 0$ which does not invalidate the theory. It seems QEPCAD is checking for nullification of all projection polynomials when equational constraints are declared leading to unnecessary errors.

5.2. Smaller lifting sets

Traditionally in CAD algorithms the projection phase identifies a set of projection polynomials, which are then used in the lifting phase to create the stacks. However when making use of equational constraints we can actually 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 by taking their resultant with the equational constraint. However, these polynomials are not then (usually) required for the lifting since Theorem 4 can (usually) be used to conclude that they are sign-invariant in the cells produced by lifting the equational constraints.

Note that 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 14 which are not in the main variable. The lower dimensional CAD D constructed in step 15 is guaranteed to be sign-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 to build stacks with respect to f.

The following two examples (and Example 19 later) demonstrate these efficiencies.

Example 16. Consider from Section 1.3 the circle f_1, hyperbola g_1 and sub-formula $\phi_1 := f_1 = 0 \wedge g_1 < 0$. Building a sign-invariant CAD for these polynomials uses 83 cells with the induced CAD of $\mathbb{R}$ identifying 7 points. Declaring the equational constraint in QEPCAD results in a CAD with 69 cells while using our implementation of Algorithm 1 produces a CAD with 53 cells. Both implementations give the same induced CAD of $\mathbb{R}$ identifying 6 points 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_1 changes sign. The splitting is not necessary for a CAD invariant with respect to an equational constraint since f_1 is non-zero for all $x < -2$.

Example 17. Now consider all four polynomials from Section 1.3 and the formula Φ from equation (2). In Section 2.4 we reported that a TTICAD could be built with 105 cells compared to a CAD with 249 cells built invariant with respect to the implicit equational constraint $f_1 f_2 = 0$ using QEPCAD. The improved projection resulted in the induced CAD of $\mathbb{R}$ identifying 12 points rather than 16.

We now observe that some of the cell savings was actually down to using smaller sets of lifting polynomials. We may simulate the projection with respect to the implicit equational constraint via Algorithm 1 by inputting a set consisting of the single formula

$$\Phi' = f_1 f_2 = 0 \wedge \Phi$$

(note that logically $\Phi = \Phi'$). The implementation in `ProjectionCAD` would then produce a CAD with 145 cells. So we may conclude that improved lifting allowed for a saving of 104 cells and improved projection a further saving of 40 cells.

In this example 72% of the cell saving came from improved lifting and only 28% from improved projection but we should not conclude from this that the former is more important. The improved lifting is applicable for the final lift (from a CAD of $\mathbb{R}^{n-1}$ to one of $\mathbb{R}^n$) while the improved projection applies to the first projection (from polynomials in n

variables to those with $n-1$). Hence the saving from improved projection get magnified throughout the rest of the algorithm and so as the number of variables in a problem increases we should expect to see its importance increase.

Example 18. We consider a simple 3d generalisation of the previous example. Let

$$\Phi^{3d} = \left(x^2+y^2+z^2-1=0 \wedge xyz-\tfrac{1}{4}<0\right) \\ \vee \left((x-4)^2+(y-1)^2+(z-2)^2-1=0 \wedge (x-4)(y-1)(z-2)-\tfrac{1}{4}<0\right)$$

and assume variable ordering $x \prec y \prec z$. Using Algorithm 1 on the two QFFs joined by disjunction gives a CAD with 109 cells while declaring the implicit equational constraint in QEPCAD gives 739 cells. Using Algorithm 1 on the single formula conjuncted with the implicit equational constraint gave a CAD with 353 cells. So in this case the improved lifting saves 386 cells and the improved projection a further 244 cells.

Hence moving to going from 2 to 3 variables has increased the proportion of the saving from improved projection from 28% to 39%. Further, we note that the new projection theory allows for CADs to be produced for examples where no implicit equational constraint exists meaning the benefit of Algorithm 1 is greatly enhanced (see Section 7.3).

6. Formulating a Problem for TTICAD

Algorithm 1 will produce a TTICAD for a given list of QFFs. However, before using the algorithm various choices may be required which can have significant effects on the output of the algorithm. We examine some of these possibilities here, noting that much of this analysis was first discussed in (Bradford et al., 2013b).

6.1. Variable ordering

Note that Algorithm 1 is implicitly assuming that the variables used are subject to an ordering. As with all CAD algorithms, the variable ordering will have a large effect on the output, or even whether obtaining output is computationally feasible. Brown and Davenport (2007) proved there are problems where one variable ordering will lead to a CAD with a constant number of cells while another will give a number of cells doubly exponential in the number of variables. The ordering (or some parts of it) may be determined by the origin of the problem. For example, when using a CAD for quantifier elimination the quantified variables must be eliminated first. However, in many cases we may be free to specify the ordering.

Dolzmann et al. (2004) considered the problem of choosing a variable ordering for producing a sign-invariant CAD. They identified a measure of CAD complexity that was correlated to the computation time, number of cells in the CAD and number of leaves in a partial CAD. They identified the *sum of total degrees of all monomials of all projection polynomials*, known as `sotd` and proposed the heuristic of picking the ordering with the lowest `sotd`. Although the best known heuristic, `sotd` does not always pick the ideal ordering as demonstrated by experiments in (Dolzmann et al., 2004; Bradford et al., 2013b) and Example 19.

In (Bradford et al., 2013b) we investigated the shortcoming of `sotd`, namely that it measures properties of the problem in the algebraic closure and thus misses some features specific to the real geometry. We suggested a new measure, the *number of distinct*

real roots of the univariate projection factors, or `ndrr` which could compensate for this shortcoming. Unfortunately, `ndrr` has its own shortcomings, such as not admitting a greedy algorithm for picking variables one at a time. Hence we recommended using the measures in tandem for choosing variable orderings.

It was noted by Phisanbut (2011) that a class of problems particularly unsuitable for `sotd` is choosing between coupled variables (the real and imaginary parts of a complex variable). These are used when constructing a CAD to decompose the domain according to their branch cuts, one of the main applications of the TTICAD theory. We use examples from this class in this section.

Example 19. Consider $f = \sqrt{z^2+1}$ where $z \in \mathbb{C}$. The square root function has a branch cut along the negative real axis and so f has branch cuts when

$$\mathcal{R} = \Re(z^2+1) = x^2 - y^2 + 1 < 0 \quad \text{and} \quad \mathcal{I} = \Im(z^2+1) = 2xy = 0,$$

where x, y are coupled real variables such that $z = x + \mathrm{i}y$. Suppose we use Algorithm 1 to build a TTICAD. For either variable ordering the discriminant of the equational constraint is a constant and its coefficients monomials of degree one. We have

$$\mathrm{res}_y(\mathcal{R}, \mathcal{I}) = 4x^2(x^2+1), \quad \mathrm{res}_x(\mathcal{R}, \mathcal{I}) = 4(1-y^2)y^2,$$

which have the same `sotd` but different `ndrr` (as the latter has two real roots and the former none). A TTICAD using $x \prec y$ has 13 cells and one with $y \prec x$ has 21 cells.

The first two plots in Figure 6.1 are visualisations of the two TTICADs just described. Here the solid curve is the equational constraint $\mathcal{I}$ and the dashed curve the polynomial $\mathcal{R}$. The dotted lines indicate the stacks above the isolated roots on the real line. Each circle indicates a cell with the number next to it the dimension. The final plot in Figure 6.1 is the imaginary part of $f = \sqrt{z^2+1}$ which demonstrates that f has branch cuts on the imaginary axis above 1 and below -1.

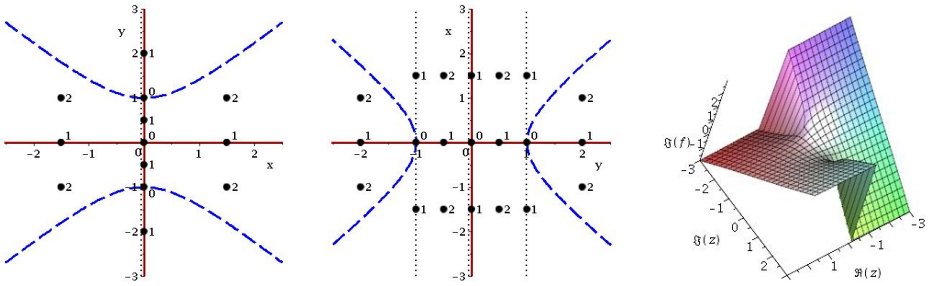


Fig. 8. Studying the branch cuts of $f = \sqrt{z^2+1}$.

Remark 20. In Example 19 the TTICAD theory has allowed us to ignore the hyperbola except for when it intersects the y-axis: all that is needed for defining the branch cuts. Although the theory of equational constraints would have allowed this, we require the improved lifting described in Section 5 to take advantage of it. In the next example the branch cuts require more than one QFF to describe them and so the theory of equational constraints could only be used with the product of all equational constraints from each QFF declared implicitly. In this case there are savings from the improved projection theory of TTICAD as well, as we described in Section 2.3.

So for choosing the variable ordering of a TTICAD we propose using the measures `sotd` and `ndrr` together, for example by using the latter to break ties in the former. It is important that they be applied specifically to the projection polynomials for the TTICAD algorithm, rather than any other CAD algorithm as the best choice for variable ordering may differ depending on the algorithm as in the next example.

Example 21. A classic example within the theory of branch cut calculation and algebraic simplification is that of Kahan's teardrop, from (Kahan, 1987). Kahan considers a fluid mechanics problem leading to the relation

$$2\text{arccosh}\left(\frac{3+2z}{3}\right) - \text{arccosh}\left(\frac{5z+12}{3(z+4)}\right) = 2\text{arccosh}\left(2(z+3)\sqrt{\frac{z+3}{27(z+4)}}\right). \quad (8)$$

He notes that it is true for all values of z in the complex plane except for a small teardrop shaped region over the negative real axis, as demonstrated by the plot of the imaginary part of the difference of the two sides on the left of Figure 9.

Recent work described in (England et al., 2013) allows for the systematic identification of semi-algebraic formula to describe branch cuts, identifying the curves in the plot on the right of Figure 9. These are described by 7 pairs of equations and inequalities. Using `ProjectionCAD`, a sign-invariant CAD for these polynomials has 409 cells using $x \prec y$ and 1143 with $y \prec x$ while a TTICAD has 55 cells using $x \prec y$ and 35 with $y \prec x$

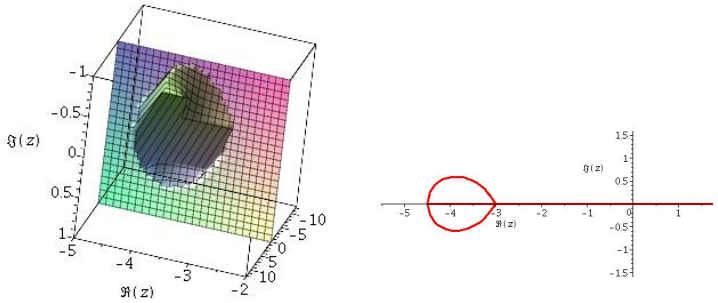


Fig. 9. Plots relating to equation (8) from Example 21.

6.2. Equational constraint designation and logical formulation

If any QFF has more than one equational constraint present then a choice must be made as to which is designated for use in the algorithm, (the others would then be treated as any other constraint). As with choosing a variable ordering, the two different projection sets could be calculated and the measures `sotd` and `ndrr` taken and used as heuristics, picking the choice that leads to the lowest values.

But this situation actually offers further choice for problem formulation than the designation. If ϕ_i had two equational constraints then it would be admissible to split this into two QFFs $\phi_{i,1}, \phi_{i,2}$ with one equational constraint assigned to each and the other constraints partitioned between them in any manner. (Admissible because any TTICAD for $\phi_{i,1}, \phi_{i,2}$ is also a TTICAD for ϕ_i.)

This is a generalisation of the following observation: given a formula ϕ with two equational constraints a CAD could be constructed using either the traditional theory of equational constraints or the TTICAD algorithm applied to two QFFs. On the surface it is not clear why the latter option would ever be chosen since it would certainly lead to more projection polynomials after the first projection. However, a specific equational constraint may have a comparatively large number of intersections with another constraint, in which case, while separating these into different QFFs would likely increase the number of projection polynomials it may still reduce the number of cells in the CAD (since the resultants taken would be less complicated leading to fewer projection factors at subsequent steps).

An example of a problem which could be tackled using the theory of equational constraints alone, but for which it is beneficial to split into two QFFs and tackle with TTICAD is the following.

Example 22. Assume $x \prec y$ and consider again $\Phi := f_1 = 0 \wedge g_1 > 0 \wedge f_2 = 0 \wedge g_2 < 0$ but this time with polynomials below. These are plotted in Figure 10 where the solid curve is f_1, the solid line g_1, the dashed curve f_2 and the dashed line g_2.

$$f_1 := (y-1) - x^3 + x^2 + x, \qquad g_1 := y - \tfrac{x}{4} + \tfrac{1}{2},$$
$$f_2 := (-y-1) - x^3 + x^2 + x, \qquad g_2 := -y - \tfrac{x}{4} + \tfrac{1}{2},$$

First, if we use the theory of equational constraints (with either f_1 or f_2 designated) then a CAD is constructed which identifies all the intersections except for g_1 with g_2 This is visualised by the plot on the left while the plot on the right relates to a TTICAD with two QFFs. In this case only three 0-cells are identified, with the intersections of g_2 with f_1 and g_1 with f_2 ignored. The TTICAD has 31 cells while the CADs produced using equational constraints both have 39 cells. Both `sotd` and `ndrr` identify the smaller CAD.

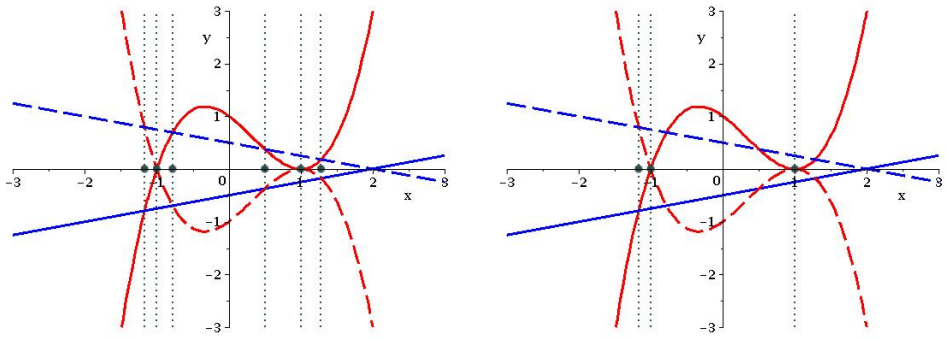


Fig. 10. Plots visualising the CADs described for the example in Section 6.

As suggested by the example we suggest using the measures `sotd` and `ndrr` applied to the set of projection polynomials as heuristics for picking an approach.

We may also consider whether to combine any QFFs if only a single truth-invariant formula is required. If the QFFs are related by conjunction then it would probably be beneficial. Formulae joined by disjunction could be combined only if they share an equational constraint, (so the new QFF has an overall one). Upon inspection of the projection operators, we see such a merger would not change the set of projection polynomials.

When considering the logical formulation the number of possibilities increases quickly. Hence we propose a method for TTICAD QFF formulation, making the choices one QFF at a time. Given a list $\{\hat{\phi}_i\}$ of QFFs:

(1) Take the disjunction of the QFFs and put that formula into disjunctive normal form, $\bigvee \hat{\phi}_i$ so that each $\hat{\phi}_i$ is a conjunction of atomic formulae.

(2) Consider each $\hat{\phi}_i$ in turn and let m_i be the number of equational constraints.
- If $m_i = 0$ then no designation is possible (and E_i will be the full set A_i).
- If $m_i = 1$ then the sole equational constraint is designated trivially.
- If $m_i > 1$ then we consider all the possible partitions of the formula in $\hat{\phi}_i$ into sub QFFs with at least one equational constraint each, and all the different designations of equational constraint within those sub-QFFs with more than one. Choose a partition and designation for this clause according to the heuristics based on `sotd` and `ndrr` applied to the projections polynomials from the clause.

(3) Input the list of new QFFs, $\{\phi_i\}$ to TTICAD.

6.3. Preconditioning input QFFs

Another option available before using Algorithm 1 is to precondition the input. Buchberger and Hong (1991) conducted experiments to see if Gröbner basis techniques could precondition problems effectively for CAD. They considered replacing any input polynomials which came from equations by a purely lexicographical Gröbner basis for them. In (Wilson et al., 2012b) this idea was considered in greater depth with a larger base of problems tested and the idea extended to include Gröbner reduction on the other polynomials. The preconditioning was shown to be highly beneficial in some cases but not others, but a simple metric was posited and shown to be a good indicator of when preconditioning was useful

In (Bradford et al., 2013b) using Gröbner preconditioning for TTICAD was investigated by performing the Gröbner preconditioning on the individual QFFs. The problem must be suitably complicated for this preconditioning to work: each QFF must have multiple equational constraints amenable to the creation of a Gröbner Basis.

It was demonstrated that for such examples the preconditioning could produce significant reductions. However, again the benefit was not always universal with the measures `sotd` and `ndrr` applied to the projection polynomials again providing good heuristics.

6.4. Summary

In this section we have highlighted some of the choices a user may need to make before using Algorithm 1 (there may be others). The heuristics suggested are all implemented in the `ProjectionCAD` package discussed in Section 4. For the experimental results in Section 7 we use the required variable ordering for a problem if it has one and otherwise test all possible orderings. If there are questions of logical formulation or equational constraint designation we use the heuristics discussed here. No Gröbner precondition was used as the aim is to analyse the TTICAD theory itself.

It is important to note that our heuristics are just that, and as such can be misled by certain examples. Also, while we have considered these issues individually they of course intersect. For example, the TTICAD formulation with two QFFs was the best choice in Example 22 but if we had assumed the other variable ordering then a single QFF is superior. Taken together, all these choices of formulation can become combinatorially overwhelming and so methods to reduce this, such as the greedy algorithm in (Dolzmann et al., 2004) or the method at the end of Section 6.2 are important.

7. Experimental Results

7.1. Description of experiments

Our timings were obtained on a Linux desktop (3.1GHz Intel processor, 8.0Gb total memory) with MAPLE 16 (command line interface), MATHEMATICA 9 (graphical interface) and QEPCAD-B 1.69. For each experiment we produce a CAD and give the time taken and number of cells (cell count). The first is an obvious metric while the second is crucial for applications performing operations on each cell.

For QEPCAD the options `+N500000000` and `+L200000` were provided, the initialization included in the timings and explicit equational constraints declared when present with the product of those from the individual QFFs declared otherwise. In MATHEMATICA the output is not a CAD but a formula constructed from one (Strzeboński, 2010), with the actual CAD not available to the user. Cell counts for the algorithms were provided by the author of the MATHEMATICA code.

TTICADs are calculated using our `ProjectionCAD` implementation described in Section 4. The results in this section are not presented to claim that our implementation is state of the art, but to demonstrate the power of the TTICAD theory over the the conventional theory, and how it can allow even a simple implementation to compete. Hence the cell counts are of most interest.

The time is measured to the nearest tenth of a second, with a time out (T/O) set at 5000 seconds. When **F** occurs it indicates failure due to a theoretical reason such as not well-oriented (in either sense). The occurrence of Err indicates an error in an internal subroutine of MAPLE's `RegularChains` package, used by `ProjectionCAD`. This error is not theoretical but a bug, which should be fixed in the future and is beyond our control.

We started by considering examples originating from (Buchberger and Hong, 1991). However these problems (and most others in the literature) involve conjunctions of conditions, chosen as such to make them amenable to existing technologies. These problems can be tackled using TTICAD, but they do not demonstrate its full strength. Hence we introduce some new examples. The first set, those denoted with a †, are adapted from (Buchberger and Hong, 1991) by turning certain conjunctions into disjunctions. The second set were generated randomly as examples with two QFFs, only one of which has an equational constraint (using random polynomials in 3 variables of degree at most 2).

Two further examples came from the application of branch cut analysis for simplification. We included Example 21 along with the problem induced by considering the validity of the double angle formulae for arcsin. Finally we considered the worked examples from Section 1.3 and the generalisation to three dimensions presented in Example 18. Note that A and B following the problem name indicate different variable orderings. Full details for all examples can be found in the CAD repository (Wilson et al., 2012a) available freely online at `http://opus.bath.ac.uk/29503`.

7.2. Results

We present our results in Table 1. For each problem we give the name used in the repository, n the number of variables, d the maximum degree of polynomials involved and t the number of QFFs used for TTICAD. We then give the time taken and number of cells produced by each algorithm.

We first compare our TTICAD implementation with the sign-invariant CAD generated using `ProjectionCAD` with McCallum's projection operator. Since these use the

same architecture the comparison makes clear the benefits of the TTICAD theory. The experiments confirm the fact that, since each cell of a TTICAD is a superset of cells from a sign-invariant CAD, the cell count for TTICAD will always be less than or equal to that of a sign-invariant CAD produced using the same implementation. Ellipse† A is not well-oriented in the sense of (McCallum, 1998), and so both methods return **FAIL**. Solotareff† A and B are well-oriented in this sense but not in the stronger sense of Definition 9 and hence TTICAD fails while the sign-invariant CADs can be produced. The only example with equal cell counts is Collision† A in which the non-equational constraints were so simple that the projection polynomials were unchanged. Examining the results for the worked examples and the 3d generalisation we start to see the true power of TTICAD. In 3D Example A we see a 759-fold reduction in time and a 50-fold reduction in cell count.

We next compare our implementation of TTICAD with the state of the art in CAD: QEPCAD (Brown, 2003), MAPLE (Chen et al., 2009b) and MATHEMATICA (Strzeboński, 2006, 2010). MATHEMATICA is the quickest, however TTICAD often produces fewer cells. We note that MATHEMATICA's algorithm uses powerful heuristics and so actually used Gröbner bases on the first two problems, causing the cell counts to be so low. When all implementations succeed TTICAD usually produces far fewer cells than QEPCAD or MAPLE, especially impressive given QEPCAD is producing partial CADs for the quantified problems, while TTICAD is only working with the polynomials involved. For Collision† A the TTICAD theory offers no benefit allowing the better optimized alternatives to have a lower cell count.

Reasons for the TTICAD implementation struggling to compete on speed in general are that the MATHEMATICA and QEPCAD algorithms are largely implemented directly in `C`, have had more optimization, and in the case of MATHEMATICA use validated numerics for lifting (Strzeboński, 2006). However, the strong performance in cell counts is very encouraging, both due its importance for applications where CAD is part of a wider algorithm (such as branch cut analysis) and for the potential if TTICAD theory were implemented elsewhere.

7.3. The increased benefit of TTICAD

We finish by demonstrating that the benefit of TTICAD over the existing theory should increase with the number of QFFs and that this benefit is much more pronounced if at least one of these does not have an equational constraint.

Example 23. We consider a family of examples (to which our worked examples belong). Assume $x \prec y$ and for j a non-negative integer define

$$\begin{aligned}
f_{j+1} &:= (x-4j)^2 + (y-j)^2 - 1,\\
g_{j+1} &:= (x-4j) * (y-j) - \tfrac{1}{4},\\
F_{j+1} &:= \{f_k, g_k\}_{k=1\ldots j+1}\\
\Phi_{j+1} &:= \bigvee_{k=1}^{j+1} (f_k = 0 \wedge g_k < 0),\\
\Psi_{j+1} &:= \left(\bigvee_{k=1}^{j} (f_k = 0 \wedge g_k < 0) \right) \vee (f_{j+1} < 0 \wedge g_{j+1} < 0).
\end{aligned}$$

Table 1. Comparing TTICAD to the full CAD built with the same architecture and other CAD algorithms.

Problem		Full-CAD		TTICAD		QEPCAD		MAPLE		MATHEMATICA	
Name	n d t	Time	Cells	Time	Cells	Time	Cells	Time	Cells	Time	Cells
IntersectionA	3 2 1	360.1	3707	1.7	269	4.5	825	—	Err	0.0	3
IntersectionB	3 2 1	332.2	2985	1.5	303	4.5	803	50.2	2795	0.0	3
RandomA	3 3 1	268.5	2093	4.5	435	4.6	1667	23.0	1267	0.1	657
RandomB	3 3 1	442.7	4097	8.1	711	5.4	2857	48.1	1517	0.0	191
Intersection†A	3 2 2	360.1	3707	68.7	575	4.8	3723	—	Err	0.1	601
Intersection†B	3 2 2	332.2	2985	70.0	601	4.7	3001	50.2	2795	0.1	549
Random†A	3 3 2	268.5	2093	223.4	663	4.6	2101	23.0	1267	0.2	808
Random†B	3 3 2	442.7	4097	268.4	1075	142.4	4105	48.1	1517	0.2	1156
Ellipse†A	5 4 2	—	**F**	—	**F**	291.6	500609	1940.1	81193	11.2	80111
Ellipse†B	5 4 2	T/O	—	T/O	—	T/O	—	T/O	—	2911.2	16603131
Solotareff†A	4 3 2	677.6	54037	46.1	**F**	4.9	20307	1014.2	54037	0.1	260
Solotareff†B	4 3 2	2009.2	154527	123.8	**F**	6.3	87469	2951.6	154527	0.1	762
Collision†A	4 4 2	264.6	8387	267.7	8387	5.0	7813	376.4	7895	3.6	7171
Collision†B	4 4 2	—	Err	—	Err	T/O	—	T/O	—	591.5	1234601
KahanA	2 4 7	10.7	409	0.3	55	4.8	261	15.2	409	0.0	72
KahanB	2 4 7	87.9	1143	0.3	39	4.8	1143	154.9	1143	0.1	278
ArcsinA	2 4 4	2.5	225	0.3	57	4.6	225	3.3	225	0.0	175
ArcsinB	2 4 4	6.5	393	0.2	25	4.5	393	7.8	393	0.0	79
2DEx(Φ)A	2 2 2	5.7	317	1.2	105	4.7	249	6.3	317	0.0	24
2DEx(Φ)B	2 2 2	6.1	377	1.5	153	4.5	329	7.2	377	0.0	175
2DEx(Ψ)A	2 2 2	5.7	317	1.6	183	4.9	317	6.3	317	0.053	372
2DEx(Ψ)B	2 2 2	6.1	377	1.9	233	4.8	377	7.2	377	0.070	596
3DExA	3 3 2	3795.8	5453	5.0	109	5.3	739	—	Err	0.1	44
3DExB	3 3 2	3404.7	6413	5.8	153	5.7	1009	—	Err	0.1	135
Random1	3 2 2	16.4	1533	76.8	1533	4.9	1535	25.7	1535	0.18	579
Random2	3 2 2	837.9	7991	132.4	2911	5.2	8023	173.0	8023	0.81	2551
Random3	3 2 2	258.6	8889	98.1	4005	5.3	8913	77.9	5061	0.65	3815
Random4	3 2 2	1442.3	11979	167.1	4035	5.4	12031	258.3	12031	1.3	4339
Random5	3 2 2	310.3	11869	110.7	4905	5.5	11893	104.3	6241	0.90	5041

Then Φ_2 is Φ from equation (2) and Ψ_2 is Ψ from equation (3). Table 2 shows the cell counts for various CADs produced for studying the truth of the formulae. Both Φ_i and Ψ_i may be studied by a sign-invariant CAD for the polynomials F_i, shown in the column marked `CADFull`. The remaining CADs are specific to one formula. Each formula has had a TTICAD constructed using Algorithm 1 on the natural sub-formulae created by the disjunctions, while the Φ_i have also had a CAD constructed using the theory of equational constraints alone. This was simulated by running Algorithm 1 on the single formula declaring the product of the f_is as an equational constraint (column marked `ECCAD`). All the proceeding CADs were constructed with `ProjectionCAD`. For each formula a CAD has also been created with QEPCAD, with the product of f_i declared as an equational constraint for Φ_i.

Table 2. Table detailing the number of cells in CADs constructed to analyse the truth of the formulae from Example 23.

j	Φ_i			F_j	Ψ	
	`ECCAD`	`TTICAD`	QEPCAD	`CADFull`	`TTICAD`	QEPCAD
2	145	105	249	317	183	317
3	237	157	508	695	259	695
4	329	209	849	1241	335	1241
5	421	261	1269	1979	411	1979
6	513	313	1769	2933	487	2933

We see that the size of a sign-invariant CAD is growing much faster than the size of a TTICAD, regardless of whether or not all QFFs have an equational constraint. If they do (the case of the Φ_i) then we see that by declaring the implicit equational constraint the existing theory can also make big savings. However, this is dependent on making use of the improved lifting discussed in Section 5 as can be concluded by comparing the `ECCAD` figures with the QEPCAD figures. In the case where at least one QFF does not have an equational constraint (the case of the Ψ_i) the existing theory of equational constraints cannot be used. So while the comparative benefit over sign-invariant CAD is slightly less, the benefit when comparing with the best available previous theory is much greater.

8. Conclusions

We have defined Truth Table Invariant CADs and by extending the theory of equational constraints have provided an algorithm to construct these efficiently, extending our previous work so that the algorithm may be applied to a general sequence of formulae. The algorithm has been implemented in MAPLE giving promising experimental results. TTICADs in general have much fewer cells than sign-invariant CADs using the same implementation and we showed that this allows even a simple implementation of TTICAD to compete with the state of the art CAD implementations. For many problems the TTICAD theory offers the smallest truth-invariant CAD for a parent formula, and there are also classes of problems for which TTICAD is exactly the desired structure. The benefits of TTICAD increase with the number of QFFs in a problem and is magnified if there is a QFF with no equational constraint (as the previous theory is not applicable here). We hope that these results inspire other implementations of TTICAD.

8.1. Future Work

There is scope for optimizing the algorithm and extending it to allow less restrictive input. Lemma 13 gives one extension that is included in our implementation while other possibilities include removing some of the caution implied by well-orientedness, analogous to (Brown, 2005).

Of course, the implementation of TTICAD used here could be optimised in many ways, but perhaps more desirable would be for TTICAD to be incorporated into existing state of the art CAD implementations. In particular, work is now ongoing on building a TTICAD with the `RegularChains` technology in Maple, already used to build sign-invariant CADs as described in (Chen et al., 2009b). We see several possibilities for the theoretical development of TTICAD:

- Can we apply the theory recursively instead of only at the top level? For example by widening the projection operator to allow enough information to conclude order-invariance, as in (McCallum, 2001).
- Can we define an algorithm to make use of more than one one equational constraint per QFF in order to gain reductions beyond the first projection? For example, can we generalise the theory of bi-equational constraints?
- Can we make use of the ideas behind partial CAD to avoid unnecessary lifting once the truth value of a QFF on a cell is determined?
- Can we implement the lifting algorithm in parallel?
- Can we modify the lifting algorithm to only return those cells required for the application (as indicated by preliminary investigations in (Wilson and England, 2013))?
- Can anything be done when the input is not well oriented?

Acknowledgements

We are grateful to A. Strzeboński for assistance in performing the Mathematica tests and to the anonymous referees of (Bradford et al., 2013a) for their useful comments on TTICAD. We also thank the rest of the Triangular Sets seminar at Bath (A. Locatelli, G. Sankaran and N. Vorobjov) for their input, and the team at Western University (C. Chen, M. Moreno Maza, R. Xiao and Y. Xie) for access to their Maple code and helpful discussions.

References

Arnon, D., 1988. A cluster-based cylindrical algebraic decomposition algorithm. Journal of Symbolic Computation 5 (1-2), 189–212.

Arnon, D., Collins, G., McCallum, S., 1984a. Cylindrical algebraic decomposition I: The basic algorithm. SIAM J, Comput. 13, 865–877.

Arnon, D., Collins, G., McCallum, S., 1984b. Cylindrical algebraic decomposition II: An adjacency algorithm for the plane. SIAM J, Comput. 13, 878–889.

Arnon, D., Collins, G., McCallum, S., 1988. An adjacency algorithm for cylindrical algebraic decompositions of three-dimensional space. J. Symb. Comput. 5 (1/2), 163–187.

Bradford, R., Davenport, J., 2002. Towards better simplification of elementary functions. In: Proceedings of the 2002 international symposium on symbolic and algebraic computation. ISSAC '02. ACM, pp. 16–22.

Bradford, R., Davenport, J., England, M., McCallum, S., Wilson, D., 2013a. Cylindrical algebraic decompositions for boolean combinations. In: Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation. ISSAC '13. ACM, pp. 125–132.

Bradford, R., Davenport, J., England, M., Wilson, D., 2013b. Optimising problem formulations for cylindrical algebraic decomposition. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (Eds.), Intelligent Computer Mathematics. Vol. 7961 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 19–34.

Brown, C., 1998. Simplification of truth-invariant cylindrical algebraic decompositions. In: Proceedings of the 1998 international symposium on Symbolic and algebraic computation. ISSAC '98. ACM, pp. 295–301.

Brown, C., 2003. An overview of QEPCAD B: a tool for real quantifier elimination and formula simplification. Journal of Japan Society for Symbolic and Algebraic Computation 10 (1), 13–22.

Brown, C., 2005. The McCallum projection, lifting, and order-invariance. Tech. rep., U.S. Naval Academy, Computer Science Department.

Brown, C., Davenport, J., 2007. The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proceedings of the 2007 international symposium on Symbolic and algebraic computation. ISSAC '07. ACM, pp. 54–60.

Brown, C., Kahoui, M. E., Novotni, D., Weber, A., 2006. Algorithmic methods for investigating equilibria in epidemic modelling. J. Symbolic Computation 41, 1157–1173.

Brown, C., McCallum, S., 2005. On using bi-equational constraints in CAD construction. In: Proceedings of the 2005 international symposium on Symbolic and algebraic computation. ISSAC '05. ACM, pp. 76–83.

Buchberger, B., Hong, H., 1991. Speeding up quantifier elimination by Gröbner bases. Tech. rep., 91-06. RISC, Johannes Kepler University.

Chen, C., Davenport, J., May, J., Maza, M. M., Xia, B., Xiao, R., Xie, Y., 2009a. User interface design for geometrical decomposition algorithms in Maple. In: Proceedings of Mathematical User-Interface. p. 12pp.

Chen, C., Maza, M. M., 2012. An incremental algorithm for computing cylindrical algebraic decompositions. Preprint: arXiv:1210.5543v1.

Chen, C., Maza, M. M., Xia, B., Yang, L., 2009b. Computing cylindrical algebraic decomposition via triangular decomposition. In: Proceedings of the 2009 international symposium on Symbolic and algebraic computation. ISSAC '09. ACM, pp. 95–102.

Collins, G., 1998. Quantifier elimination by cylindrical algebraic decomposition – 20 years of progress. In: Caviness, B., Johnson, J. (Eds.), Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts & Monographs in Symbolic Computation. Springer-Verlag, pp. 8–23.

Collins, G., Hong, H., 1991. Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. 12, 299–328.

Davenport, J., Bradford, R., England, M., Wilson, D., 2012. 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. IEEE, pp. 83–88.

Davenport, J., Heintz, J., 1988. Real quantifier elimination is doubly exponential. J. Symb. Comput. 5 (1-2), 29–35.

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

England, M., 2013a. 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/`.

England, M., 2013b. An implementation of CAD in maple utilising problem formulation, equational constraints and truth-table invariance. Department of Computer Science Technical Report series 2013-04, University of Bath. Available at `http://opus.bath.ac.uk/35636/`.

England, M., Bradford, R., Davenport, J., Wilson, D., 2013. Understanding branch cuts of expressions. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (Eds.), Intelligent Computer Mathematics. Vol. 7961 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 136–151.

Fotiou, I., Parrilo, P., Morari, M., 2005. Nonlinear parametric optimization using cylindrical algebraic decomposition. In: Decision and Control, 2005 European Control Conference. CDC-ECC '05. pp. 3735–3740.

Hong, H., 1990. An improvement of the projection operator in cylindrical algebraic decomposition. In: Proceedings of the international symposium on Symbolic and algebraic computation. ISSAC '90. ACM, pp. 261–264.

Iwane, H., Yanami, H., Anai, H., Yokoyama, K., 2009. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In: Proceedings of the 2009 conference on Symbolic Numeric Computation. SNC '09. pp. 55–64.

Kahan, W., 1987. Branch cuts for complex elementary functions. In: Iserles, A., Powell, M. (Eds.), Proceedings The State of Art in Numerical Analysis. Clarendon Press, pp. 165–211.

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

McCallum, S., 1998. An improved projection operation for cylindrical algebraic decomposition. In: Caviness, B., Johnson, J. (Eds.), Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts & Monographs in Symbolic Computation. Springer-Verlag, pp. 242–268.

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

McCallum, S., 2001. On propagation of equational constraints in CAD-based quantifier elimination. In: Proceedings of the 2001 international symposium on Symbolic and algebraic computation. ISSAC '01. ACM, pp. 223–231.

Paulson, L., 2012. Metitarski: Past and future. In: Beringer, L., Felty, A. (Eds.), Interactive Theorem Proving. Vol. 7406 of Lecture Notes in Computer Science. Springer, pp. 1–10.

Phisanbut, N., 2011. Practical simplification of elementary functions using cylindrical algebraic decomposition. Ph.D. thesis, University of Bath.

Phisanbut, N., Bradford, R., Davenport, J., 2010. Geometry of branch cuts. ACM Communications in Computer Algebra 44 (3), 132–135.

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

Seidl, A., Sturm, T., 2003. A generic projection operator for partial cylindrical algebraic decomposition. In: Proceedings of the 2003 international symposium on Symbolic and algebraic computation. ISSAC '03. ACM, pp. 240–247.

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

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

Wilson, D., Bradford, R., Davenport, J., 2012a. A repository for CAD examples. ACM Communications in Computer Algebra 46 (3), 67–69.

Wilson, D., Bradford, R., Davenport, J., 2012b. Speeding up cylindrical algebraic decomposition by Gröbner bases. In: Jeuring, J., Campbell, J., Carette, J., Reis, G., Sojka, P., Wenzel, M., Sorge, V. (Eds.), Intelligent Computer Mathematics. Vol. 7362 of Lecture Notes in Computer Science. Springer, pp. 280–294.

Wilson, D., England, M., 2013. ayered cylindrical algebraic decomposition. Department of Computer Science Technical Report series 2013-05, University of Bath. Available at `http://opus.bath.ac.uk/36712/`.

Yanami, H., Anai, H., 2006. Development of SyNRAC. In: Proceedings of the 6th international conference on Computational Science: Part II. (LNCS vol 3992). ICCS '06. pp. 462–469.