Truth Table Invariant Cylindrical Algebraic Decomposition by Regular Chains

Russell Bradford[1], Changbo Chen[2], James H. Davenport[1], Matthew England[1], Marc Moreno Maza[3] and David Wilson[1]

[1] University of Bath, Bath, BA2 7AY, UK.
[2] CIGIT, Chinese Academy of Sciences, Chongqing, 400714, China.
[3] University of Western Ontario, London, Ontario, N6A 5B7, Canada.
{R.Bradford, J.H.Davenport, M.England, D.J.Wilson}@bath.ac.uk,
moreno@csd.uwo.ca, changbo.chen@hotmail.com

Abstract. A new algorithm to compute cylindrical algebraic decompositions (CADs) is presented, building on two recent advances. Firstly, the output is truth table invariant (a TTICAD) meaning given formulae have constant truth value on each cell of the decomposition. Secondly, the computation uses regular chains theory to first build a cylindrical decomposition of complex space (CCD) incrementally by polynomial. Significant modification of the regular chains technology was used to achieve the more sophisticated invariance criteria. Experimental results on an implementation in the `RegularChains` Library for Maple verify that combining these advances gives an algorithm superior to its individual components and competitive with the state of the art.

Keywords: cylindrical algebraic decomposition; equational constraint; regular chains; triangular decomposition

1 Introduction

A *cylindrical algebraic decomposition* (CAD) is a collection of cells such that: they do not intersect and their union describes all of $\mathbb{R}^n$; they are arranged *cylindrically*, meaning the projections of any pair of cells are either equal or disjoint; and, each can be described using a finite sequence of polynomial relations.

CAD was introduced by Collins in [15] to solve quantifier elimination problems, and this remains an important application of CAD (including the new work presented here). Other applications include epidemic modelling [8], parametric optimisation [21], theorem proving [24], robot motion planning [26] and reasoning with multi-valued functions and their branch cuts [17]. CAD has complexity doubly exponential in the number of variables. While for some applications there now exist algorithms with better complexity (see for example [5]), CAD implementations remain the best general purpose approach for many.

In this paper we present a new CAD algorithm which combines two recent advances in CAD theory: the technique of producing CADs via regular chains in complex space [14], and the idea of producing CADs closely aligned to the structure of given logical formulae [2]. The introduction continues by reminding the reader of CAD theory and these advances.

1.1 Background on CAD

We work with polynomials in ordered variables $\boldsymbol{x} = x_1 \prec \ldots \prec x_n$. The *main variable* of a polynomial (mvar) is the greatest variable present with respect to the ordering. Denote by QFF a *quantifier free Tarski formula*: a Boolean combination $(\wedge, \vee, \neg)$ of statements $f_i \, \sigma \, 0$ where $\sigma \in \{=, >, <\}$ and the f_i are polynomials. CAD was developed as a tool for the problem of quantifier elimination over the reals: given a quantified Tarski formula

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

(where $Q_i \in \{\forall, \exists\}$ and F is a QFF), produce an equivalent QFF $\psi(x_1, \ldots, x_k)$. Collins proposed to build a CAD of $\mathbb{R}^n$ which is *sign-invariant*, so each $f_i \in F$ is either positive, negative or zero on each cell. Then ψ is the disjunction of the defining formulae of those cells $c \in \mathbb{R}^k$ where Ψ is true, which given sign-invariance, requires us to only test one *sample point* per cell.

Collins' algorithm works by first *projecting* the problem into decreasing real dimensions and then *lifting* to build CADs of increasing dimension. Important developments range from improved projection operators [23] to the use of certified numerics when lifting [27] [22]. See for example [2] for a fuller discussion.

1.2 Truth table invariant CAD

One important development is the use of *equational constraints* (ECs), which are equations logically implied by a formula. These may be given explicitly as in $(f = 0) \wedge \varphi$, or implicitly as $f_1 f_2 = 0$ is by $(f_1 = 0 \wedge \varphi_1) \vee (f_2 = 0 \wedge \varphi_2)$.

In [23] McCallum developed the theory of a reduced operator for the first projection, so that the CAD produced was sign-invariant for the polynomial defining a given EC, and then sign-invariant for other polynomials only when the EC is satisfied. Extensions of this to make use of more than one EC have been developed (see for example [9]) while in [3] it was shown how McCallum's theory could allow for further savings in the lifting phase.

The CADs produced are no longer sign-invariant for polynomials but instead *truth-invariant* for a formula. Truth-invariance was defined in [6] where sign-invariant CADs were refined to maintain it. We consider a related definition.

Definition 1 ([2]). *Let $\Phi = \{\phi_i\}_{i=1}^t$ be a list of QFFs. A CAD is* Truth Table Invariant *for Φ (a TTICAD) if on each cell every ϕ_i has constant Boolean value.*

In [2] an algorithm to build TTICADs when each ϕ_i has an EC was derived by extending McCallum's theory [23] to define a reduced projection operator. Implementations in Maple showed this offered great savings in both CAD size and computation time when compared to the sign-invariant theory. In [3] this theory has been extended to work on arbitrary ϕ_i, with savings if at least one has an EC. Note that there are two distinct reasons to build a TTICAD:

1. *As a tool to build a truth-invariant CAD:* If a parent formula ϕ^* is built from $\{\phi_i\}$ then any TTICAD for $\{\phi_i\}$ is also truth-invariant for ϕ^*.
   A TTICAD may be the best truth-invariant CAD, or at least the best we can compute. Note that the TTICAD theory allows for more savings than the use of [23] with an implicit EC built as the product of ECs from ϕ_i [2].

2. *When truth table invariance is required:* There are applications which provide a list of formulae but no parent formula. For example, decomposing complex space according to a set of branch cuts for the purpose of algebraic simplification [1] [25] [20]. When the branch cuts can be expressed as semi-algebraic systems a TTICAD provides exactly the required decomposition.

1.3 CAD by regular chains

Recently, a radically different method to build CADs has been investigated. Instead of projecting and lifting the problem is moved to complex space where the theory of triangular decomposition by regular chains is used to build a *complex cylindrical decomposition* (CCD), a decomposition of $\mathbb{C}^n$ such that each cell is cylindrical. This is encoded as a tree data structure, with each path through the tree describing the end leaf as a solution of a regular system [29].

This was first proposed in [14] to build a sign-invariant CAD. Techniques developed for comprehensive triangular decomposition [11] were used to build a sign-invariant decomposition of $\mathbb{C}^n$ which was then refined to a CCD. Finally, real root isolation is applied to refine further to a CAD of $\mathbb{R}^n$. The computation of the CCD may be viewed as an enhanced projection phase since gcds of pairs of polynomials are calculated as well as resultants. The extra work used here makes the second phase, which may be compared to lifting, less expensive. The main advantage is the use of case distinction in the second phase, so that the zeros of polynomials not relevant in a particular branch are not isolated there.

The construction of the CCD was improved in [13]. The former approach built a decomposition for the input in one step using existing algorithms. The latter approach proceeds incrementally by polynomial, each time using purpose-built algorithms to refine an existing tree whilst maintaining cylindricity. Experimental results showed that the latter approach is much quicker, with its implementation in Maple's `RegularChains` library now competing with existing state of the art CAD algorithms: Qepcad [7] and Mathematica [28]. One reason for this improvement is the ability of the new algorithm to recycle subresultant calculations, an idea introduced and detailed in [12] for the purpose of decomposing polynomial systems into regular chains incrementally.

Another benefit of the incremental approach is that it allows for simplification when constructing a CAD in the presence of ECs. Instead of working with polynomials, the algorithm can be modified to work with relations. Then branches in which an EC is not satisfied may be truncated, offering the possibility of a reduction in both computation time and output size. In [13] it was shown that using this optimization allowed the algorithm to process examples which Mathematica and Qepcad could not.

1.4 Contribution and outline

In Section 2 we present our new algorithms. Our aim is to combine the savings from an invariance criteria closer to the underlying application, with the savings offered by the case distinction of the regular chains approach. It requires

adapting the existing algorithms for the regular chains approach so that they refine branches of the tree data structure only when necessary for truth-table invariance, and so that branches are truncated only when appropriate to do so.

We implemented our work in the `RegularChains` library for MAPLE. In Section 3 we qualitatively compare our new algorithm to our previous work and in Section 4 we present experimental results comparing it to the state of the art. Finally, in Section 5 we give our conclusions and ideas for future work.

2 Algorithm

2.1 Constructing a complex cylindrical tree

Let $\boldsymbol{x} = x_1 \prec \cdots \prec x_n$ be a sequence of ordered variables. We will construct TTICADs of $\mathbb{R}^n$ for a semi-algebraic system *sas* (Definition 5). However, to achieve this we first build CCDs of $\mathbb{C}^n$ with respect to a complex system.

Definition 2. *Let $F = \{p_1, \ldots, p_s\}$ be a finite set from $\mathbb{Q}[\boldsymbol{x}]$, $G \subseteq F$ and $\sigma_i \in \{=, \neq\}$. Then we define a* complex system *(denoted by cs) as a set*

$$\{p_i \ \sigma_i \ 0 \mid p_i \in G\} \cup \{p_i \mid p_i \in F \setminus G\}.$$

The complex systems we work with will be defined in accordance with a semi-algebraic counterpart (see Definition 5). For $p \in \mathbb{Q}[\boldsymbol{x}]$ we denote the zero set of p in $\mathbb{C}^n$ by $Z_{\mathbb{C}}(p)$, or $Z_{\mathbb{C}}(p = 0)$, and its complement by $Z_{\mathbb{C}}(p \neq 0)$.

We compute CCDs as trees, following [14, 13]. Throughout let T be a rooted tree with each node of depth i a polynomial constraint of type either, "any x_i" (with zero set defined as $\mathbb{C}^n$), or $p = 0$, or $p \neq 0$ (where $p \in \mathbb{Q}[x_1, \ldots, x_i]$). For any i denote the induced subtree of T with depth i by T_i. Let Γ be a path of T and define its zero set $Z_{\mathbb{C}}(\Gamma)$ as the intersection of zero sets of its nodes. The zero set of T, denoted $Z_{\mathbb{C}}(T)$, is defined as the union of zero sets of its paths.

Definition 3. *T is a* complete complex cylindrical tree *(complete CCT) of $\mathbb{Q}[\boldsymbol{x}]$ if it satisfies recursively:*

1. *If $n = 1$: either T has one leaf "any x_1", or it has $s+1$ ($s \geq 1$) leaves $p_1 = 0, \ldots, p_s = 0, \prod_{i=1}^{s} p_i \neq 0$, where $p_i \in \mathbb{Q}[x_1]$ are squarefree and coprime.*
2. *The induced subtree T_{n-1} is a complete CCT.*
3. *For any given path Γ of T_{n-1}, either its leaf V has only one child "any x_n", or V has $s+1$ ($s \geq 1$) children $p_1 = 0, \ldots, p_s = 0, \prod_{i=1}^{s} p_i \neq 0$, where $p_1, \ldots, p_s \in \mathbb{Q}[\mathbf{x}]$ are squarefree and coprime satisfying:*

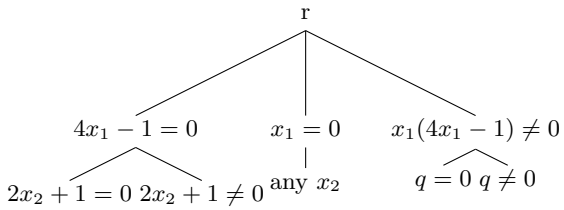
3a. *for any $\alpha \in Z_{\mathbb{C}}(\Gamma)$, none of $\mathrm{lc}(p_j, x_n)$, $j = 1, \ldots, s$, vanishes at α, and*

3b. *$p_1(\alpha, x_n), \ldots, p_s(\alpha, x_n)$ are squarefree and coprime.*

The set $\{Z_{\mathbb{C}}(\Gamma) \mid \Gamma \text{ is a path of } T\}$ is called the *complex cylindrical decomposition* (CCD) of $\mathbb{C}^n$ associated with T: condition (3b) assures that it is a decomposition. Note that for a complete CCT we have $Z_{\mathbb{C}}(T) = \mathbb{C}^n$. A proper subtree rooted at the root node of T of depth n is called a *partial CCT* of $\mathbb{Q}[\boldsymbol{x}]$. We use CCT to refer to either a complete or partial CCT. We call a complex cylindrical tree T an *initial tree* if T has only one path and T is complete.

Definition 4. *Let T be a CCT of $\mathbb{C}^n$ and Γ a path of T. A polynomial $p \in \mathbb{Q}[\boldsymbol{x}]$ is* sign invariant *on Γ if either $Z_{\mathbb{C}}(\Gamma) \cap Z_{\mathbb{C}}(p) = \emptyset$ or $Z_{\mathbb{C}}(p) \supseteq Z_{\mathbb{C}}(\Gamma)$. A constraint $p = 0$ or $p \neq 0$ is* truth-invariant *on Γ if p is sign-invariant on Γ. A complex system cs is* truth-invariant *on Γ if the conjunction of the constraints in cs is truth-invariant on Γ, and each polynomial in cs is sign-invariant on Γ.*

Example 1. Let $q := (x_2^2 + x_2 + x_1)$ and $p := x_1 q$. The following tree is a CCT such that p is sign-invariant (and $p = 0$ is truth invariant) on each path.



We introduce Algorithm 1 to produce truth-table invariant CCTs, and new sub-algorithms 2 and 3. It also uses `IntersectPath` and `NextPathToDo` from [13]. `IntersectPath` takes: a CCT T; a path Γ; and p, either a polynomial or constraint. When a polynomial it refines T so p is sign-invariant above each path from Γ (still satisfying Definition 3). When a constraint it refines so the constraint is true, possibly truncating branches if there can be no solution. This necessitates the housekeeping algorithm `MakeComplete` which restores to a complete CCT by simply adding missing siblings (if any) to every node. `NextPathToDo` simply returns the next incomplete path Γ of T.

Proposition 1. *Algorithm 1 satisfies its specification.*

Proof. It suffices to show that Algorithm 2 is as specified. First observe that Algorithm 3 just recursively calls `IntersectPath` on constraints and so its correctness follows from that of `IntersectPath`. (When called on ECs `IntersectPath` may return a partial tree and so `MakeComplete` must be used in line 6).

Algorithm 2 is clearly correct is its base cases, namely (line 2), (line 5) and (line 9). It also clearly terminates since the input of each recursive call has less constraints. For each path C of the refined Γ, by induction, it is sufficient to show that cs is truth-invariant on C. If $p \neq 0$ on C, then cs is false on C. If $p = 0$ on C, then the truth of cs is invariant since it is completely determined by the truth of $cs' := cs \setminus \{p = 0\}$, invariant on C by induction.

Algorithm 1: TTICCD(L)

Input: A list L of complex systems of $\mathbb{Q}[\boldsymbol{x}]$.
Output: A complete CCT T with each $cs \in L$ truth-invariant on each path.
1 Create the initial CCT T and let Γ be its path;
2 IntersectLCS(L, Γ, T);

Algorithm 2: IntersectLCS(L, Γ, T)

Input: A CCT T of $\mathbb{Q}[\boldsymbol{x}]$. A path Γ of T. A list of complex systems L.
Output: Refinements of Γ and T such that T is complete, and $cs \in L$ is truth-invariant above each path of Γ.

```
1  if L = ∅ then
2  |  return
3  else if |L| = 1 then
4  |  Let cs be the only complex system;
5  |  IntersectPolySet(cs, Γ, T);
6  |  MakeComplete(T);
7  else if no cs ∈ L has an equational constraint then
8  |  Let F be the set of polynomials appearing in L;
9  |  IntersectPolySet(F, Γ, T)
10 else
11 |  Let cs be a complex system of L with an EC denoted p = 0;
12 |  IntersectPath(p, Γ, T) // Γ may become a tree
13 |  while C := NextPathToDo(Γ) ≠ ∅ do
14 |  |  if p = 0 on C then
15 |  |  |  cs' := cs \ {p = 0};
16 |  |  |  IntersectLCS(L \ {cs} ∪ {cs'}, C, T)
17 |  |  else
18 |  |  |  IntersectLCS(L \ {cs}, C, T)
```

Algorithm 3: IntersectPolySet(F, Γ, T)

Input: A CCT T, a path Γ and a set F of polynomials (constraints).
Output: T is refined and Γ becomes a subtree. Each polynomial (constraint) of F is sign (truth)-invariant above each path of Γ.

```
1 if F = ∅ then return;
2 Let p ∈ F; F' := F \ {p};
3 IntersectPath(p, Γ, T) // Γ may become a tree
4 if F' ≠ ∅ then
5 |  while C := NextPathToDo(Γ) ≠ ∅ do
6 |  |  IntersectPolySet(F', C, T);
```

2.2 Illustrating the computational flow

Consider using Algorithm 1 on input of the form

$$L = [cs_1, cs_2] := [\{f_1 = 0, g_1 \neq 0\}, \{f_2 = 0, g_2 \neq 0\}].$$

Algorithm 1 constructs the initial tree and passes to Algorithm 2. We enter the fourth branch of the conditional, let $p = f_1$, and refine to a sign invariant CCT for f_1. This makes a case distinction between $f_1 = 0$ and $f_1 \neq 0$. On the branch $f_1 \neq 0$, we recursively call IntersectLCS on $[cs_2]$ which then passes directly to IntersectPolySet. On the branch $f_1 = 0$, we recursively call IntersectLCS on $[\{g_1 \neq$

$0\}, \{f_2 = 0, g_2 \neq 0\}]$. This time $p = f_2$ and a case discussion is made between $f_2 = 0$ and $f_2 \neq 0$. On the branch $f_2 \neq 0$, we end up calling $\mathsf{IntersectPolySet}(g_1 \neq 0)$ while on the branch $f_2 = 0$ we call $\mathsf{IntersectLCS}$ on $[\{g_1 \neq 0\}, \{g_2 \neq 0\}]$, which reduces to $\mathsf{IntersectPolySet}(g_1, g_2)$. The case discussion is summarised by:

$$\begin{cases} f_1 = 0 : \begin{cases} f_2 = 0 : g_1, g_2 \\ f_2 \neq 0 : g_1 \neq 0 \end{cases} \\ f_1 \neq 0 : f_2 = 0, g_2 \neq 0 \end{cases} .$$

2.3 Refining to a TTICAD

We now discuss how Section 2.1 can be extended from CCDs to CADs.

Definition 5. *A* semi-algebraic system *of* $\mathbb{Q}[\boldsymbol{x}]$ *(sas) is a set of constraints* $\{p_i \ \sigma_i \ 0\}$ *where each* $\sigma_i \in \{=, >, \geq, \neq\}$ *and each* $p_i \in \mathbb{Q}[\boldsymbol{x}]$. *A* corresponding complex system *is formed by replacing all* $p_i > 0$ *by* $p_i \neq 0$ *and all* $p_i \geq 0$ *by* p_i.

A sas is truth-invariant *on a cell if the conjunction of its constraints is.*

Note that the ECs of an sas are still identified as ECs of the corresponding cs. Algorithm 4 produces a TTICAD of $\mathbb{R}^n$ for a sequence of semi-algebraic systems.

Algorithm 4: $\mathsf{RC-TTICAD(L)}$

Input: A list L of semi-algebraic systems of $\mathbb{Q}[\boldsymbol{x}]$.
Output: A CAD such that each $sas \in L$ is truth-invariant on each cell.

1 Set L' to be the list of corresponding complex systems ;
2 $\mathcal{D} := \mathsf{TTICCD}(L')$;
3 `MakeSemiAlgebraic`$(\mathcal{D}, n)$

Proposition 2. *Algorithm 4 satisfies its specification.*

Proof. Algorithm 4 starts by building the corresponding *cs* for each *sas* in the input. It uses Algorithm 1 to form a CCD truth-invariant for each of these and then the algorithm `MakeSemiAlgebraic` introduced in [14] to move to a CAD. `MakeSemiAlgebraic` takes a CCD $\mathcal{D}$ and outputs a CAD $\mathcal{E}$ such that for each element $d \in \mathcal{D}$ the set $d \cap \mathbb{R}^n$ is a union of cells in $\mathcal{E}$. Hence $\mathcal{E}$ is still truth-invariant for each $cs \in L'$. It is also a TTICAD for L, (as to change sign from positive to negative would mean going through zero and thus changing cell). The correctness of Algorithm 4 hence follows from the correctness of its sub-algorithms.

The output of Algorithm 4 is a TTICAD for the formula defined by each semi-algebraic system (the conjunction of the individual constraints of that system). To consider formulae with disjunctions we must first use disjunctive normal form and then construct semi-algebraic systems for each conjunctive clause.

3 Comparison with prior work

We now compare qualitatively to our previous work. Quantitative experiments and a comparison with competing CAD implementations follows in Section 4.

3.1 Comparing with sign-invariant CAD by regular chains

Algorithm 4 uses work from [13] but obtains savings when building the complex tree by ensuring only truth-table invariance. To demonstrate this we compare diagrams representing the number of times a constraint is considered when building a CCD for a complex system.

Definition 6. *Let cs be a complex system. We define the* complete (resp. partial) combination diagram *for cs, denoted by $\Delta_0(cs)$ (resp. $\Delta_1(cs)$), recursively:*

- *If $cs = \emptyset$, then $\Delta_i(cs)$ $(i = 0, 1)$ is defined to be null.*
- *If cs has any ECs then select one, ψ (defined by a polynomial f), and define*

$$\Delta_0(cs) := \begin{cases} f = 0 & \Delta_0(cs \setminus \{\psi\}) \\ f \neq 0 & \Delta_0(cs \setminus \{\psi\}) \end{cases},$$

$$\Delta_1(cs) := \begin{cases} f = 0 & \Delta_1(cs \setminus \{\psi\}) \\ f \neq 0 & \end{cases}.$$

- *Otherwise select a constraint ψ (which is either of the form $f \neq 0$, or f) and for $i = 0, 1$ define*

$$\Delta_i(cs) := \begin{cases} f = 0 & \Delta_i(cs \setminus \{\psi\}) \\ f \neq 0 & \Delta_i(cs \setminus \{\psi\}) \end{cases}.$$

The combination diagrams illustrate the combinations of relations that must be analysed by our algorithms, with the partial diagram relating to Algorithm 1 and the complete diagram the sign-invariant algorithm from [13].

Lemma 3. *Assume that the complex system cs has s ECs and t constraints of other types. Then the number of constraints appearing in $\Delta_0(cs)$ is $2^{s+t+1} - 2$, and the number appearing in $\Delta_1(cs)$ is $2(2^t + s) - 2$.*

Proof. The diagram $\Delta_0(cs)$ is a full binary tree with depth $s + t$. Hence the number of constraints appearing is the geometric series $\sum_{i=1}^{s+t} 2^i = 2^{s+t+1} - 2$.

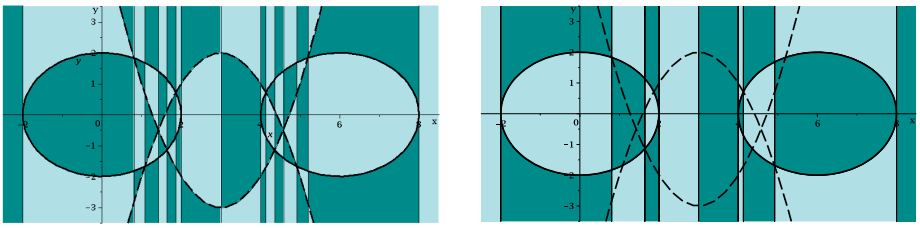
$\Delta_1(cs)$ will start with a binary tree for the ECs, with only one branch continuing at each depth, and thus involves $2s$ constraints. The full binary tree for the other constraints is added to the final branch, giving a total of $2^{t+1} + 2s - 2$.

Definition 7. *Let L be a list of complex systems. We define the* complete (resp. partial) combination diagram *of L, denoted by $\Delta_0(L)$ (resp. $\Delta_1(L)$) recursively: If $L = \emptyset$, then $\Delta_i(L)$, $i = 0, 1$, is null. Otherwise let cs be the first element of L. Then $\Delta_i(L)$ is obtained by appending $\Delta_i(L \setminus \{cs\})$ to each leaf node of $\Delta_i(cs)$.*

Theorem 4. *Let L be a list of r complex systems. Assume each $cs \in L$ has s ECs and t constraints of other types. Then the number of constraints appearing in $\Delta_0(L)$ is $2^{r(s+t)+1} - 2$ and the number of constraints appearing in $\Delta_1(L)$ is $N(r) = 2(s + 2^t)^r - 2$.*

Proof. The number of constraints in $\Delta_0(L)$ again follows from the geometric series. For $\Delta_1(L)$ we proceed with induction on r. The case $r = 1$ is given by Lemma 3, so now assume $N(r - 1) = 2(s + 2^t)^{r-1} - 2$.

Fig. 1. The left is a sign-invariant CAD, and the right a TTICAD, for (1).



The result for r will follow from $C(r) = C(1) + (s + 2^t)C(r-1)$. To conclude this identity consider the diagram for the first $cs \in L$. To extend this to $\Delta_1(L)$ we need to append $\Delta_1(L \setminus cs)$ to each end node. There are s such nodes for cases where an EC was not satisfied and a further 2^t from the cases where all ECs were (and non-equational constraints were included).

We now give an example to demonstrate these savings.

Example 2. Assume ordering $x \prec y$ and consider

$$\begin{aligned} f_1 &:= x^2 + y^2 - 4, & g_1 &:= (x-3)^2 - (y+3), & \phi_1 &:= f_1 = 0 \wedge g_1 < 0, \\ f_2 &:= (x-6)^2 + y^2 - 4, & g_2 &:= (x-3)^2 + (y-2), & \phi_2 &:= f_2 = 0 \wedge g_2 < 0. \end{aligned} \quad (1)$$

The polynomials are graphed in Figure 1 where the solid circles are the f_i and the dashed parabola the g_i. To study the truth of the formulae $\{\phi_1, \phi_2\}$ we could create a sign-invariant CAD. Both the incremental regular chains technology of [13] and QEPCAD [7] do this with 231 cells. The 72 full dimensional cells are visualised on the left of Figure 1, (with the cylinders on each end actually split into three full dimensional cells out of the view).

Alternatively we may build a TTICAD using Algorithm 4 to obtain only 63 cells, 22 of which have full dimension as visualised on the right of Figure 1. By comparing the figures we see that the differences begin in the CAD of the real line, with the sign-invariant case splitting into 31 cells compared to 19. The points identified on the real line each align with a feature of the polynomials. Note that the TTICAD identifies the intersections of f_i and g_j only when $i = j$, and that no features of the inequalities are identified away from the ECs.

3.2 Comparing with TTICAD by projection and lifting

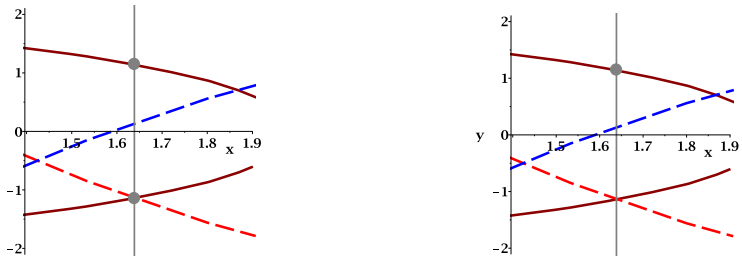
We now compare Algorithm 4 with the TTICADs obtained by projection and lifting in [2]. We identify three main benefits which we demonstrate by example.

(I) Algorithm 4 can achieve cell savings from case distinction.

Example 3. Algorithm 4 produces a TTICAD for (1) with 63 cells compared to a TTICAD of 67 cells from the projection and lifting algorithm in [2]. The full-dimensional cells are identical and so the image on the right of Figure 1 is an accurate visualisation of both. To see the difference we must compare lower

dimensional cells. Figure 2 compares the lifting to $\mathbb{R}^2$ over a cell on the real line aligned with an intersection of f_1 and g_1. The left concerns the algorithm in [2] and the right Algorithm 4. The former isolates both the y-coordinates where $f_1 = 0$ while the latter only one (the single point over the cell where ϕ_1 is true).

Fig. 2. Comparing TTICADs for (1). The left uses [2] and the right Algorithm 4.



If we modified the problem so the inequalities in (1) were not strict then ϕ_1 becomes true at both points and Algorithm 4 outputs the same TTICAD as [2]. Unlike [2], the type of the non-ECs affects the behaviour of Algorithm 4.

(II) Algorithm 4 can take advantage of more than one EC per clause.

Example 4. We assume $x \prec y$ and consider

$$\begin{aligned} &f_1 := x^2 + y^2 - 1, \qquad h := y^2 - \tfrac{x}{2}, \qquad g_1 := xy - \tfrac{1}{4} \\ &f_2 := (x-4)^2 + (y-1)^2 - 1 \quad g_2 := (x-4)(y-1) - \tfrac{1}{4}, \\ &\phi_1 := h = 0 \wedge f_1 = 0 \wedge g_1 < 0, \ \phi_2 := f_2 = 0 \wedge g_2 < 0. \end{aligned} \tag{2}$$

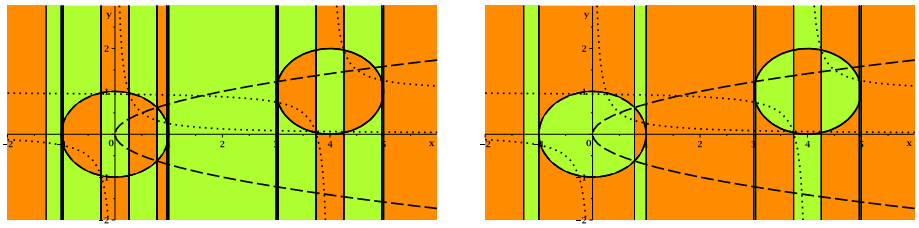
The polynomials are graphed in Figure 3 where the dashed curves are f_1 and h, the solid curve is f_2 and the dotted curves are g_1 and g_2. A TTICAD produced by Algorithm 4 has 69 cells and is visualised on the right of Figure 3 while a TTICAD produced by projection and lifting has 117 cells and is visualised on the left. This time the differences are manifested in the full-dimensional cells.

The algorithm from [2] works with a single designated EC in each QFF (in this case we chose f_1) and so treats h in the same way as g_1. This means for example that all the intersections of h or g_1 with f_1 are identified. By comparison, Algorithm 4 would only identify the intersection of g_1 with an EC if this occurred at a point where both f_1 and h were satisfied (does not occur here). For comparison, a sign-invariant CAD using QEPCAD or [13] has 611 cells.

When using [2] in Example 4 we needed to designate either f_1 or h as the EC. Choosing f_1 gave 117 cells while choosing h gives 163 cells. Algorithm 4 also offers a similar choice: what order should the systems be considered and what order the components within? Currently these choices are made informally. Processing f_1 first gives 69 cells but other choice can decrease this to 65 or increase it to 145. Making these choices intelligently is an important topic for future work.

(III) Algorithm 4 will succeed given sufficient time and memory.

Fig. 3. TTICAD for (2). The left uses [2] and the right Algorithm 4.



This contrasts with the theory of reduced projection operators used in [2], where input must satisfy be *well-oriented* (meaning that certain projection polynomials cannot be nullified when lifting over a cell with respect to them).

Example 5. Consider the identity $\sqrt{z}\sqrt{w} = \sqrt{zw}$ over $\mathbb{C}^2$. We analyse its truth by decomposing according to the branch cuts and testing each cell at its sample point. Letting $z = x + \mathrm{iy}, \mathrm{w} = \mathrm{u} + \mathrm{iv}$ we see that branch cuts occur when

$$(y = 0 \wedge x < 0) \vee (v = 0 \wedge u < 0) \vee (yu + xv = 0 \wedge xu - yv < 0).$$

We can obtain a truth-invariant CAD for this formula by building the TTICAD for the three disjoined clauses. With ordering $v \prec u \prec x \prec y$ Algorithm 4 does this using 97 cells, while the projection and lifting approach identifies the input as not well-oriented. The failure is triggered by $yu + xv$ being nullified over a cell where $u = x = 0$ and $v < 0$.

4 Experimental Results

We present experimental results obtained on a Linux desktop (3.1GHz Intel processor, 8.0Gb total memory). We tested on 52 different examples, with a representative subset of these detailed in Table 1. Maple code for all examples is stored at: `www.cs.bath.ac.uk/~djw42/RCTTICADexamples.txt`.

One set of problems was taken from CAD papers [10] [2] and a second from system solving papers [11] [13]. The polynomials from the problems were placed into different logical formulations: disjunctions in which every clause had an EC (indicated by †) and disjunctions in which only some do (indicated by ††). A third set was generated by branch cuts of algebraic relations: addition formulae for elementary functions and examples previously studied in the literature.

Each problem had a declared variable ordering (with n the number of variables). For each experiment a CAD was produced with the time taken (in seconds) and number of cells (cell count) given. The first is an obvious metric and the second crucial for applications acting on each cell. T/O indicates a time out (set at 30 minutes), FAIL a failure due to theoretical reasons such as input not being well-oriented (see [23] [2]) and Err an unexpected error.

We start by comparing our new algorithm with our previous work (all implemented and tested in development Maple) by considering the first five algorithms in Table 1. RC-TTICAD refers to Algorithm 4, PL-TTICAD the algorithm from [2], PL-CAD an implementation of CAD with McCallum projection, RC-Inc-CAD the algorithm from [13] and RC-Rec-CAD the algorithm from [14].

Those starting RC are part of the `RegularChains` library and those starting PL the `ProjectionCAD` package [19]. RC-Rec-CAD is a modification of the algorithm distributed with MAPLE 17; the construction of the CCD is the same but the conversion to a CAD has been improved. Algorithms RC-TTICAD and RC-Rec-CAD are currently being integrated in the `RegularChains` library, to be downloaded from `www.regularchains.org`.

We see that RC-TTICAD never gives higher cell counts than any of our previous work and that in general the TTICAD theories allow for cell counts an order of magnitude lower. RC-TTICAD is usually the quickest in some cases offering vast speed-ups. It is also important to note that there are many examples where PL-TTICAD has a theoretical failure but for which RC-TTICAD will complete (see point (III) in Section 3.2). Further, these failures largely occurred in the examples from branch cut analysis, a key application of TTICAD.

The results allow us to conclude that our new algorithm successfully combines the good features of our previous approaches, giving an approach superior to either. We now compare with competing CAD implementations, detailed in the last four columns of Table 1: MATHEMATICA [28] (V9 graphical interface); QEPCAD-B [7] (with options `+N500000000 +L200000`, initialization included in the timings and implicit EC declared when present); the REDUCE package REDLOG [18] (2010 Free CSL Version); and the MAPLE package SYNRAC [22].

As reported in [2], the TTICAD theory allows for lower cell counts than QEPCAD even when manually declaring an EC. We found that both SYNRAC and REDLOG failed for many examples, (with SYNRAC returning unexpected error messages and REDLOG hanging producing no output or messages). When computations succeed there were examples for which REDLOG had a lower cell count than RC-TTICAD due to their use of partial lifting techniques, but this was not the case in general. We note that we were using the most current public version of SYNRAC which has since been replaced by a superior development version, (to which we do not have access) and that REDLOG is mostly focused on the virtual substitution approach to quantifier elimination but that we only tested the CAD command.

MATHEMATICA is the quickest in general, often impressively so. However, the output there is not a CAD but a formula with a cylindrical structure [28] (hence cell counts are not available). Such a formula is sufficient for many applications (such as quantifier elimination) but not for others (such as algebraic simplification by branch cut decomposition). Further, there are examples for which RC-TTICAD completes but MATHEMATICA times out. MATHEMATICA's output only references the CAD cells for which the input formula is true. Our implementation can be modified to do this and in some cases this can lead to significant time savings; we will investigate this further in a later publication.

Finally, note that the TTICAD theory allows those algorithms to change with the logical structure of a problem. For example, Solotareff† is simpler than Solotareff†† (the later has an inequality where the former has an equation). A smaller TTICAD can hence be produced, while the sign-invariant algorithms give the same output.

5 Conclusions and further work

We presented a new algorithm to compute CADs. It combined the benefits of case distinction from regular chains construction (meaning more efficient computation and no well-orientedness conditions leading to theoretical failure), with the reduced information requirements of truth-table invariance. We demonstrated its benefit over our previous work and its competitiveness with the state of the art in CAD computation. However, there are still many questions to be considered:

- Can we use heuristics to make choices such as what variable ordering to use? This has been shown to be useful for other CAD algorithms [18] [4].
- Can we make educated choices for the order systems and constraints are analysed by the algorithm? Example 4 shows that this could be beneficial.
- Can we modify the algorithm for the case of providing truth invariant CADs for a formula in disjunctive normal form? In this case we could cease refinement in the complex tree once a branch is known to be true.
- Can we combine with other theory such as partial CAD [16] or cylindrical algebraic sub-decompositions [30]?

Acknowledgements

Supported by EPSRC grant: EP/J003247/1 and NSFC grant: 11301524.

References

1. R. Bradford and J.H. Davenport. Towards better simplification of elementary functions. In *Proc. ISSAC '02*, pp 16–22. ACM, 2002.
2. R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Cylindrical algebraic decompositions for boolean combinations. In *Proc. ISSAC '13*, pp 125–132. ACM, 2013.
3. R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Truth table invariant cylindrical algebraic decomposition. *Preprint at:* `http://opus.bath.ac.uk/38146/`, 2014.
4. R. Bradford, J.H. Davenport, M. England, and D. Wilson. Optimising problem formulations for cylindrical algebraic decomposition. In *Intelligent Computer Mathematics*, (LNCS vol. 7961), pp 19–34. Springer Berlin Heidelberg, 2013.
5. S. Basu, R. Pollack, and M.F. Roy. Algorithms in Real Algebraic Geometry. Springer, 1996.
6. C.W. Brown. Simplification of truth-invariant cylindrical algebraic decompositions. In *Proc. ISSAC '98*, pp 295–301. ACM, 1998.
7. C.W. Brown. An overview of QEPCAD B: A program for computing with semi-algebraic sets using CADs. *SIGSAM Bulletin*, 37(4):97–108, ACM, 2003.
8. C.W. Brown, M. El Kahoui, D. Novotni, and A. Weber. Algorithmic methods for investigating equilibria in epidemic modelling. *J. Symb. Comp.*, 41:1157–1173, 2006.
9. C.W. Brown and S. McCallum. On using bi-equational constraints in CAD construction. In *Proc. ISSAC '05*, pp 76–83. ACM, 2005.
10. B. Buchberger and H. Hong. Speeding up quantifier elimination by Gröbner bases. Technical report, 91-06. RISC, Johannes Kepler University, 1991.

11. C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. Comprehensive triangular decomposition. In *Computer Algebra in Scientific Computing*, (LNCS vol. 4770), pp 73–101. Springer Berlin Heidelberg, 2007.
12. C. Chen and M. Moreno Maza. Algorithms for computing triangular decomposition of polynomial systems. *J. Symb. Comp.*, 47(6):610–642, 2012.
13. C. Chen and M. Moreno Maza. An incremental algorithm for computing cylindrical algebraic decompositions. *Proc. ASCM '12*, (to appear, Springer). Preprint: arXiv:1210.5543, 2012.
14. C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *Proc. ISSAC '09*, pp 95–102. ACM, 2009.
15. G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conference on Automata Theory and Formal Languages*, pp 134–183. Springer-Verlag, 1975.
16. G.E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comp.*, 12:299–328, 1991.
17. J.H. Davenport, R. Bradford, M. England, and D. Wilson. Program verification in the presence of complex numbers, functions with branch cuts etc. In *Proc. SYNASC '12*, pp 83–88. IEEE, 2012.
18. A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. In *Proc. ISSAC '04*, pp 111–118. ACM, 2004.
19. M. England. An implementation of CAD in Maple utilising problem formulation, equational constraints and truth-table invariance. *Uni. of Bath Dept. Computer Science Tech. Report series*, 2013-04. Available at `http://opus.bath.ac.uk/35636/`, 2013.
20. M. England, R. Bradford, J.H. Davenport, and D. Wilson. Understanding branch cuts of expressions. In *Intelligent Computer Mathematics* (LNCS vol. 7961), pp 136–151. Springer Berlin Heidelberg, 2013.
21. I.A. Fotiou, P.A. Parrilo, and M. Morari. Nonlinear parametric optimization using cylindrical algebraic decomposition. In *Proc. Decision and Control, European Control Conference '05*, pp 3735–3740, 2005.
22. H. Iwane, H. Yanami, H. Anai, and K. Yokoyama. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In *Proc. SNC '09*, pp 55–64, 2009.
23. S. McCallum. On projection in CAD-based quantifier elimination with equational constraint. In *Proc. ISSAC '99*, pp 145–149. ACM, 1999.
24. L.C. Paulson. Metitarski: Past and future. In L. Beringer and A. Felty, editors, *Interactive Theorem Proving*, (LNCS vol. 7406), pp 1–10. Springer, 2012.
25. N. Phisanbut, R.J. Bradford, and J.H. Davenport. Geometry of branch cuts. *ACM Communications in Computer Algebra*, 44(3):132–135, 2010.
26. J.T. Schwartz and M. Sharir. On the "Piano-Movers" Problem: II. General techniques for computing topological properties of real algebraic manifolds. *Adv. Appl. Math.*, 4:298–351, 1983.
27. A. Strzeboński. Cylindrical algebraic decomposition using validated numerics. *J. Symb. Comp.*, 41(9):1021–1038, 2006.
28. A. Strzeboński. Computation with semialgebraic sets represented by cylindrical algebraic formulas. In *Proc. ISSAC '10*, pp 61–68. ACM, 2010.
29. D. Wang. Computing triangular systems and regular systems. *J. Symb. Comp.*, 30(2):221–236, 2000.
30. D. Wilson, R. Bradford, J.H. Davenport, and M. England. Cylindrical algebraic sub-decompositions. *Preprint at:* `http://opus.bath.ac.uk/38148/`, 2014.

Table 1. Comparing our new algorithm to our previous work and competing CAD implementations.

		RC-TTICAD		RC-Inc-CAD		RC-Rec-CAD		PL-TTICAD		PL-CAD		Mathematica	Qepcad			SyNRAC		Redlog	
Problem	n	Cells	Time	Cells	Time	Cells	Time	Cells	Time	Cells	Time	Time	Cells	True	Time	Cells	Time	Cells	Time
Intersection†	3	541	1.0	3723	12.0	3723	19.0	579	3.5	3723	29.5	0.1	3723	721	4.9	3723	12.8	Err	—
Ellipse†	5	71231	317.1	81183	544.9	81193	786.8	FAIL	—	FAIL	—	11.2	500609	94816	275.3	Err	—	Err	—
Solotareff†	4	2849	8.8	54037	209.1	54037	539.0	FAIL	—	54037	407.6	0.1	16603	333	5.2	Err	—	3353	8.6
Solotareff††	4	8329	21.4	54037	226.9	54037	573.4	FAIL	—	54037	414.3	0.1	16603	751	5.3	Err	—	8367	13.6
2D Ex†	2	97	0.2	317	1.0	317	2.6	105	0.6	317	1.8	0.0	249	36	4.8	317	1.1	305	0.9
2D Ex††	2	183	0.4	317	1.1	317	2.6	183	1.1	317	1.8	0.0	317	55	4.6	317	1.2	293	0.9
3D Ex†	3	109	3.5	3497	63.1	3525	1165.7	109	2.9	5493	142.8	0.1	739	116	5.4	—	T/O	Err	—
MontesS10	7	3643	19.1	3643	28.3	3643	26.6	—	T/O	—	T/O	T/O	—	—	T/O	—	T/O	Err	—
Wang 93	5	507	44.4	507	49.1	507	46.9	—	T/O	—	T/O	897.1	FAIL	—	—	Err	—	Err	—
Rose†	3	3069	200.9	7075	498.8	7075	477.1	—	T/O	—	T/O	T/O	FAIL	—	—	—	T/O	Err	—
genLinSyst-3-2†	11	222821	3087.5	—	T/O	—	T/O	FAIL	—	FAIL	—	T/O	FAIL	—	—	Err	—	Err	—
BC-Kahan	2	55	0.2	409	2.4	409	4.9	55	0.2	409	2.4	0.1	261	29	4.8	409	1.5	Err	—
BC-Arcsin	2	57	0.1	225	0.9	225	1.9	57	0.2	225	0.9	0.0	225	26	4.8	225	0.7	161	2.4
BC-Sqrt	4	97	0.2	113	0.5	113	1.3	FAIL	—	113	0.6	0.0	105	15	4.7	105	0.4	73	0.0
BC-Arctan	4	211	3.5	—	T/O	—	T/O	FAIL	—	—	T/O	T/O	—	—	T/O	Err	—	—	T/O
BC-Arctanh	4	211	3.5	—	T/O	—	T/O	FAIL	—	—	T/O	T/O	—	—	T/O	Err	—	—	T/O
BC-Phisanbut-1	4	325	0.8	389	1.8	389	5.8	FAIL	—	389	3.6	0.1	377	53	4.8	389	2.0	217	0.2
BC-Phisanbut-4	4	543	1.6	2007	13.6	2065	21.5	FAIL	—	51763	932.5	11.9	51763	6560	8.6	Err	—	Err	—