

Cylindrical Algebraic Decomposition: from Polynomials to Formulae

James Davenport: The University of Bath

Joint work with:

Bath Russell Bradford, Matthew England and
David Wilson

CIGIT Changbo Chen

Macquarie Scott McCallum

Western Ontario Marc Moreno Maza

Rikkyo University: 31 July 2014

Outline

- 1 Introduction
 - Cylindrical Algebraic Decomposition
 - CAD for Boolean Combinations
- 2 Developing TTICAD
 - Motivation
 - New Projection Operator
 - Important Technicalities
- 3 TTICAD in Practice
 - Implementation in MAPLE
 - Experimental Results
 - Beyond equational constraints
 - Regular Chains CAD
- 4 Conclusions etc.
 - Conclusions
 - Bibliography

Cylindrical algebraic decomposition

A **Cylindrical Algebraic Decomposition (CAD)** is a partition of \mathbb{R}^n into cells arranged cylindrically (meaning their projections are either equal or disjoint) such that each cell is defined by a semi-algebraic set.

Defined by Collins who gave an algorithm to produce a **sign-invariant** CAD for a set of polynomials, meaning each polynomial had constant sign on each cell. In some sense, makes the induced geometry of \mathbb{R}^n explicit

Originally motivated for use in quantifier elimination. Have also been applied directly on problems as diverse as algebraic simplification and (at least theoretically) robot motion planning.

Projection and lifting

Collins algorithm has two main phases:

Projection A projection operator is applied repeatedly to the polynomials, each time producing a new set of polynomials in one less variable.

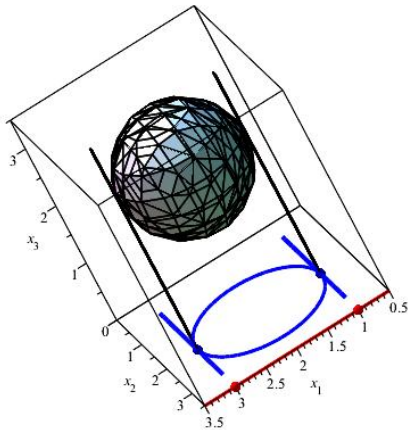
- Lifting**
- A CAD of \mathbb{R} is produced using the roots of the univariate polynomials and intervals between.
 - Over each cell: the bivariate polynomials are evaluated at a sample point, a **stack** is built consisting of **sections** (the roots) and **sectors** (the intervals). Together these are a CAD of \mathbb{R}^2 .

⋮

-
- Repeated until a CAD of \mathbb{R}^n is constructed.

The projection operator is defined so the CAD is sign-invariant.

Projection example



The projection operator applied to the sphere identifies the circle. The projection operator applied to the circle identifies two points on the real line.

Projection and lifting

Collins algorithm has two main phases:

Projection A projection operator is applied repeatedly to the polynomials, each time producing a new set of polynomials in one less variable.

Lifting

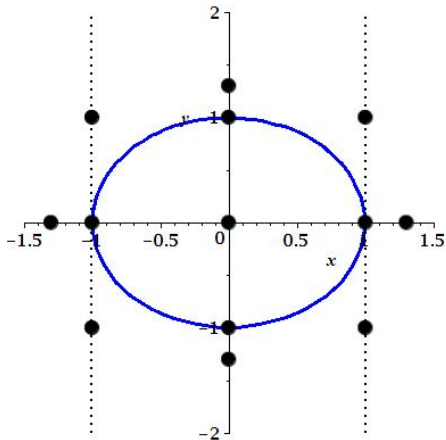
- A CAD of \mathbb{R} is produced using the roots of the univariate polynomials and intervals between.
- Over each cell: the bivariate polynomials are evaluated at a sample point, a **stack** is built consisting of **sections** (the roots) and **sectors** (the intervals). Together these are a CAD of \mathbb{R}^2 .

⋮

- Repeated until a CAD of \mathbb{R}^n is constructed.

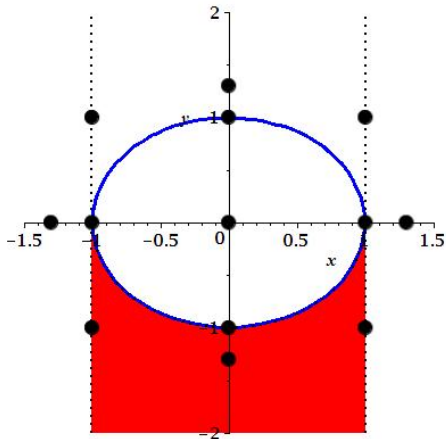
The projection operator is defined so the CAD is sign-invariant.

Lifting example



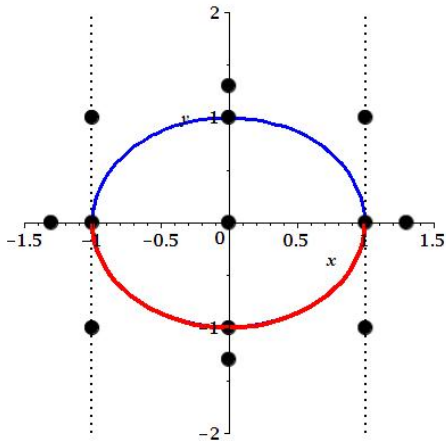
A CAD of \mathbb{R}^2 which is sign-invariant with respect to the circle. Each black dot represents a cell.

Lifting example



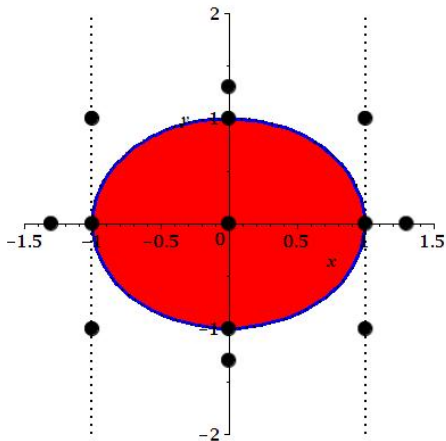
A CAD of \mathbb{R}^2 which is sign-invariant with respect to the circle. Each black dot represents a cell.

Lifting example



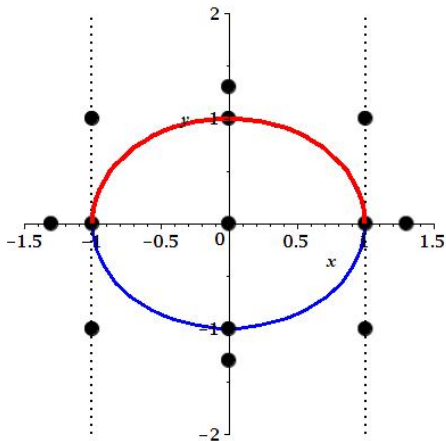
A CAD of \mathbb{R}^2 which is sign-invariant with respect to the circle. Each black dot represents a cell.

Lifting example



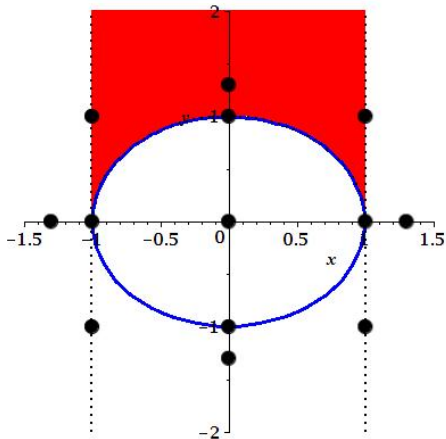
A CAD of \mathbb{R}^2 which is sign-invariant with respect to the circle. Each black dot represents a cell.

Lifting example



A CAD of \mathbb{R}^2 which is sign-invariant with respect to the circle. Each black dot represents a cell.

Lifting example



A CAD of \mathbb{R}^2 which is sign-invariant with respect to the circle. Each black dot represents a cell.

Improvements to CAD

There have been many improvements and extensions to CAD theory including but not limited to:

- Improvements to the sub-algorithms used by Collins.
- New projection operators.
- Results on complexity of CAD.
- CAD tailored to specific problems (notably Virtual Term Substitution).
- Results and algorithms on the adjacency of CAD cells.
- CAD via triangular decomposition (see later).

So how do we project?

(Lifting is in fact relatively straight-forward)

Given polynomials $\mathcal{P}_n = \{p_i\}$ in x_1, \dots, x_n , what should \mathcal{P}_{n-1} be?

Naïve (Doesn't work!) Every $\text{disc}_{x_n}(p_i)$, every $\text{res}_{x_n}(p_i, p_j)$

i.e. where the polynomials fold, or cross: misses lots of "special cases"

[Col75] First enlarge \mathcal{P}_n with all its reducta, then naïve plus the coefficients of \mathcal{P}_n (with respect to x_n) the principal subresultant coefficients from the disc_{x_n} and res_{x_n} calculations

[Hon90] a tidied version of [Col75].

[McC88] Let \mathcal{B}_n be a squarefree basis for the primitive parts of \mathcal{P}_n . Then \mathcal{P}_{n-1} is the contents of \mathcal{P}_n , the coefficients of \mathcal{B}_n and every $\text{disc}_{x_n}(b_i)$, $\text{res}_{x_n}(b_i, b_j)$ from \mathcal{B}_n

[Bro01] Naïve plus leading coefficients (not squarefree!)

Are these projections correct?

[Col75] Yes, and it's relatively straightforward to prove that, over a cell in \mathbb{R}^{n-1} sign-invariant for \mathcal{P}_{n-1} , the polynomials of \mathcal{P}_n do not cross, and define cells sign-invariant for the polynomials of \mathcal{P}_n

[McC88] 52 pages (based on [Zar75]) prove the equivalent statement, but for **order-invariance**, not sign-invariance, provided the polynomials are **well-oriented**, a test that has to be applied during lifting.

But what if they're not known to be well-oriented?

[McC88] suggests adding all partial derivatives

In practice hope for well-oriented, and if it fails use Hong's projection.

[Bro01] Needs well-orientedness and additional checks

What about the complexity?

If the McCallum projection is well-oriented, the complexity is

$$(2d)^{n2^{n+7}} m^{2^{n+4}} l^3 \quad (1)$$

versus the original

$$(2d)^{2^{2n+8}} m^{2^{n+6}} l^3 \quad (??)$$

and in practice the gains in running time can be factors of a thousand, or, more often, the difference between feasibility and infeasibility

“Randomly”, well-orientedness ought to occur with probability 1, but we have a family of “real-world” examples (simplification/branch cuts) where it often fails

Need it be this hard?

The Heintz construction

$$\Phi_k(x_k, y_k) := \left[\begin{array}{l} \exists z_k \forall x_{k-1} y_{k-1} \left[\begin{array}{l} y_{k-1} = y_k \wedge x_{k-1} = z_k \vee y_{k-1} = z_k \wedge x_{k-1} = x_k \\ \Rightarrow \Phi_{k-1}(x_{k-1}, y_{k-1}) \end{array} \right] \end{array} \right]$$

If $\Phi_1 \equiv y_1 = f(x_1)$, then $\Phi_2 \equiv y_2 = f(f(x_2))$,

$\Phi_3 \equiv y_3 = f(f(f(f(x_3))))$

[DH88] shows $\Omega\left(2^{2^{(n-2)/5}}\right)$ (using $y_R + iy_I = (x_R + ix_I)^4$)

[BD07] shows $\Omega\left(2^{2^{(n-1)/3}}\right)$ (using a sawtooth)

Hence doubly exponential is inevitable, but there's a lot of room!

In fact, there are theoretical algorithms which are

singly-exponential in n , but doubly-exponential in the number of

$\exists \forall$ alternations

CAD of a formula

Most applications of CAD relate not just to polynomials, but formulae containing them. A key approach to improving CAD is to take the structure of these formulae into account.

PartialCAD The input is a quantified formula rather than the polynomials within. Stack construction terminates early if the value of the quantified formula on the whole stack is already apparent. **lifting**

CAD with equational constraint The input is a formula and equation logically implied by the formula. The projection operator is modified so that the other polynomials are guaranteed sign invariant only on those cells of the CAD where the equational constraint is satisfied. **projection+**

Truth invariance

A CAD is **truth-invariant** with respect to a formula if the formula has constant truth value on each cell. Such a CAD could in theory be produced using far fewer cells than a CAD sign-invariant for the polynomials involved.

- Brown employed truth invariance to simplify sign-invariant CADs / PartialCADs.
- The use of a reduced projection operator with respect to an equational constraint produces a CAD which is not sign-invariant but truth-invariant.

Truth-table invariance

Given a sequence of quantifier free formulae (QFF) we define a **truth table invariant CAD (TTICAD)** as a CAD such that each formulae has constant truth value on each cell.

We gave [BDE⁺13] an algorithm to construct TTICADs for sequences of formulae which each has an equational constraint.

This:

- will (in general) produce far fewer cells than the sign-invariant CAD for the polynomials involved;
- does not require calculation of the sign-invariant CAD first.

We achieve this by extending the theory of equational constraints.

The algorithm has been implemented in MAPLE and shows promising experimental results.

Simple motivating example

Consider the polynomials:

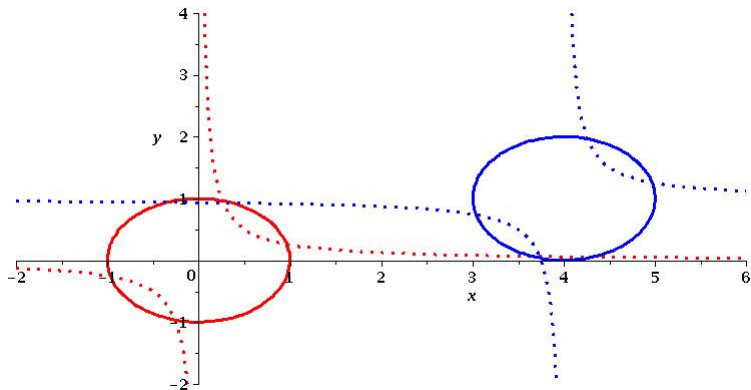
$$\begin{aligned} f_1 &:= x^2 + y^2 - 1 & g_1 &:= xy - \frac{1}{4} \\ f_2 &:= (x - 4)^2 + (y - 1)^2 - 1 & g_2 &:= (x - 4)(y - 1) - \frac{1}{4} \end{aligned}$$

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

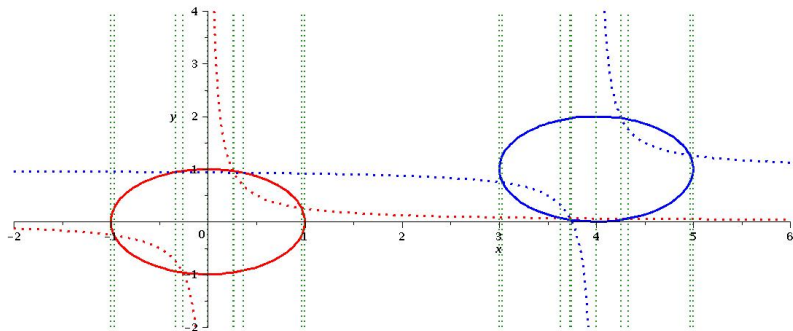
$$\Phi := (f_1 = 0 \wedge g_1 < 0) \vee (f_2 = 0 \wedge g_2 < 0)$$

We could solve the problem using a full sign-invariant CAD for $\{f_1, g_1, f_2, g_2\}$, QEPCAD and MAPLE would both use 317 cells. This identified 20 points on the real line.

Example: graph of polynomials



Example: sign-invariant CAD



All curve intersections identified.

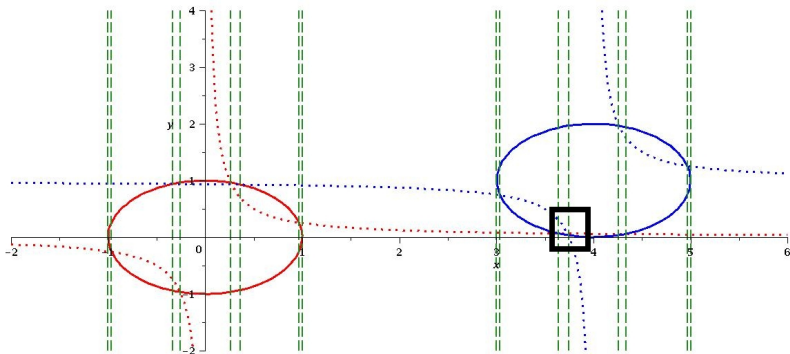
Simple motivating example continued

We could instead employ the theory of equational constraints.

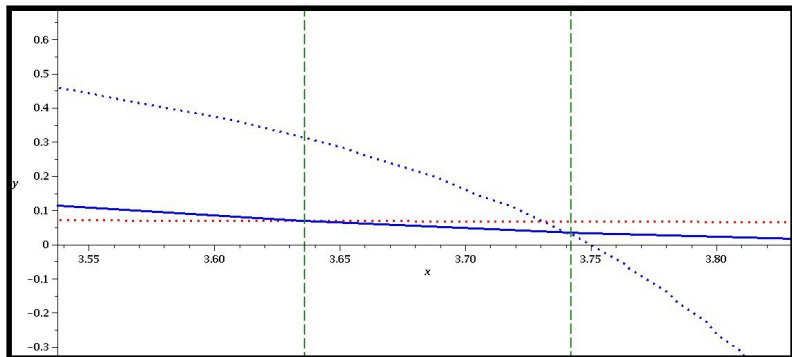
Although Φ has no explicit equational constraint the equation $f_1 f_2 = 0$ is implied implicitly.

Using the functionality in QEPCAD this gives a CAD with 249 cells. This identifies 16 points on the real line.

Example: CAD with equational constraint



Example: CAD with equational constraint



New projection operator for TTICAD

Let $\mathcal{A} = \{A_i\}_{i=1}^t$ be a list of irreducible bases for the polynomials in a sequence of QFFs and $\mathcal{E} = \{E_i\}_{i=1}^t$ non-empty subsets $E_i \subseteq A_i$.

We define the **reduced projection of \mathcal{A} with respect to \mathcal{E}** , as:

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

where

$$P_{E_i}(A_i) = P(E_i) \cup \{\text{res}_{x_n}(e, a)\}_{e \in E_i, a \in A_i \setminus E_i}$$

$$P(A) = \{\text{disc}(a), \text{coeffs}_{x_n}(a), \text{res}_{x_n}(a, b)\}_{a, b \in A}$$

$$\text{Res}^{\times}(\mathcal{E}) = \{\text{res}_{x_n}(e, \hat{e}) \mid \exists i, j : e \in E_i, \hat{e} \in E_j, i < j, e \neq \hat{e}\}$$

Using the operator to build a TTICAD

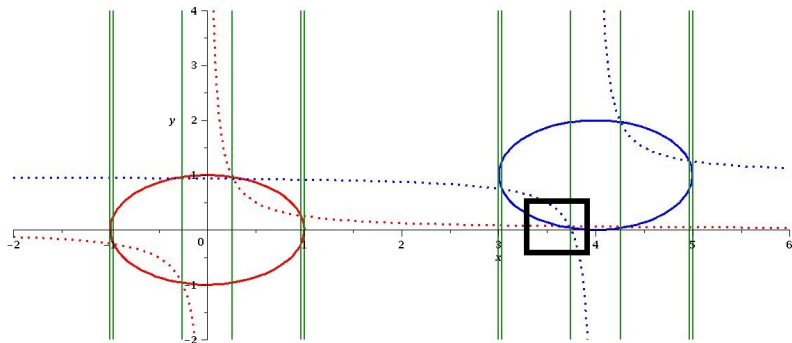
Full technical details of our algorithm to produce a TTICAD of \mathbb{R}^n are given in [BDE⁺13], along with a formal verification.

Key points:

- Apply the reduced projection once to find projection polynomials \mathfrak{P} in $n - 1$ variables.
- Use McCallum's verified algorithm to build a sign-invariant CAD of \mathbb{R}^{n-1} for \mathfrak{P} .
- Perform a final lift with respect to the equational constraints.

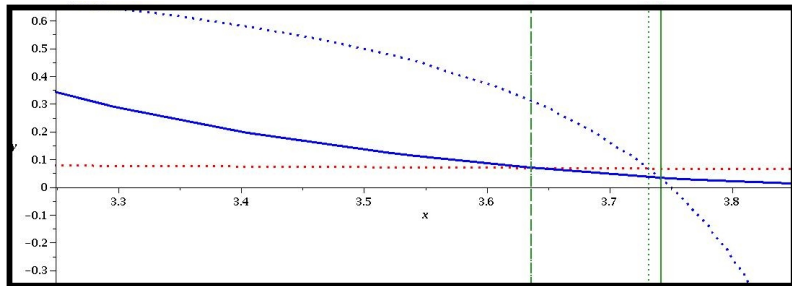
Example: TTICAD

A TTICAD for the motivating example is built with 105 cells (compared to 317 and 249). This identified 12 points on the real line (compared to 20 and 16).



Example: TTICAD

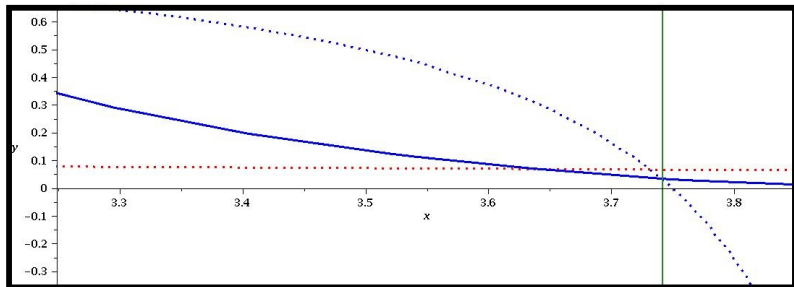
A TTICAD for the motivating example is built with 105 cells (compared to 317 and 249). This identified 12 points on the real line (compared to 20 and 16).



All three CADs together.

Example: TTICAD

A TTICAD for the motivating example is built with 105 cells (compared to 317 and 249). This identified 12 points on the real line (compared to 20 and 16).



TTICAD only

Important technicalities

We highlight a couple of important technicalities:

- 1 We used McCallum's algorithm to produce the CAD of \mathbb{R}^{n-1} as this gives a CAD which is order-invariant.
This stronger condition is required to conclude that the output of our algorithm is a TTICAD.
- 2 McCallum's operator and hence his algorithm are only valid for use when the input is **well-oriented**, (finite number of nullification points for all projection polynomials).
- 3 Hence our new projection operator and algorithm requires a similar condition:

\mathcal{A} is well oriented with respect to \mathcal{E} if the equational constraints have a finite number of nullification points and \mathfrak{P} is well-oriented.

Implementations

There are various existing implementations of CAD including QEPCAD, MAPLE, MATHEMATICA. But none output order-invariant CADs.

We built our own implementation on MAPLE. Developed a package ProjectionCAD for use in MAPLE 16 on. Available to download freely from: <http://opus.bath.ac.uk/35636/>

Can produce CADs sign-invariant (using McCallum or Collins' operators), order invariant, with equational constraint and truth-table invariant. Also provides heuristics for formulation.

Experiments I

First compared our implementation of TTICAD with our implementation of sign-invariant CAD using McCallum's operator.

- TTICAD cell counts and timings usually an order of magnitude lower.
- One example with the same cell count: the equational constraint occurred as a projection factor of the projection set for the other constraints.
- Two examples where a sign-invariant CAD could be constructed while a TTICAD cannot: an equational constraint was nullified.

Experiments II

[BDE⁺13] compared our TTICAD implementation with QEPCAD-B (v1.59), MAPLE (v16) and MATHEMATICA (v9).

- Mathematica certainly the quickest although TTICAD often produces fewer cells. Mathematica produces cylindrical formulae rather than CADs and uses powerful heuristics.
- TTICAD usually produces far fewer cells than QEPCAD or MAPLE, even when QEPCAD produces partial CADs.
- Some examples of theoretical failure for TTICAD where others complete.
- Timings vary according to example. TTICAD competing well with QEPCAD and MAPLE, but usually slower.

Further Developments (submitted)

Can we widen the input specification to allow some QFFs without equational constraint?

YES: By treating all constraints in that QFF with the importance reserved for equational constraints.

Naïvely If a $g_j > 0$ occurs \wedge with $f_i = 0$, the only polynomial involving g_j we need is $\text{res}(f_i, g_j)$.

Generalising our example

Assume $x \prec y$ and for j a non-negative integer define

$$f_{j+1} := (x - 4j)^2 + (y - j)^2 - 1,$$

$$g_{j+1} := (x - 4j) * (y - j) - \frac{1}{4},$$

$$F_{j+1} := \{f_k, g_k\}_{k=1\dots j+1}$$

$$\Phi_{j+1} := \bigvee_{k=1}^{j+1} (f_k = 0 \wedge g_k < 0),$$

$$\Psi_{j+1} := \left(\bigvee_{k=1}^j (f_k = 0 \wedge g_k < 0) \right) \vee (f_{j+1} < 0 \wedge g_{j+1} < 0).$$

So Ψ does not have an implicit equational constraint

Cells Counts

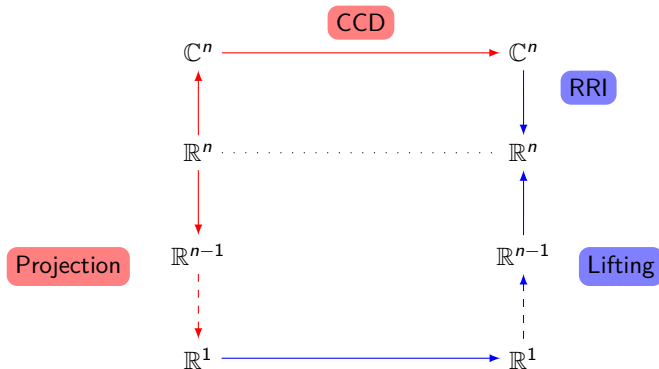
j	Φ_j			F_j	Ψ	
	ECCAD	TTICAD	QEPCAD	CADFull	TTICAD	QEPCAD
2	145	105	249	317	183	317
3	237	157	508	695	259	695
4	329	209	849	1241	335	1241
5	421	261	1269	1979	411	1979
6	513	313	1769	2933	487	2933

ECCAD and TTICAD both seem linear in j , while CADFull is quadratic.

For $j \geq 5$ TTICAD on Ψ even beats ECCAD on Φ .

An alternative approach [CMMXY09, CMM12]

Proceed via the complex numbers,



Do a complex cylindrical decomposition via **Regular Chains**

Example Complex CD

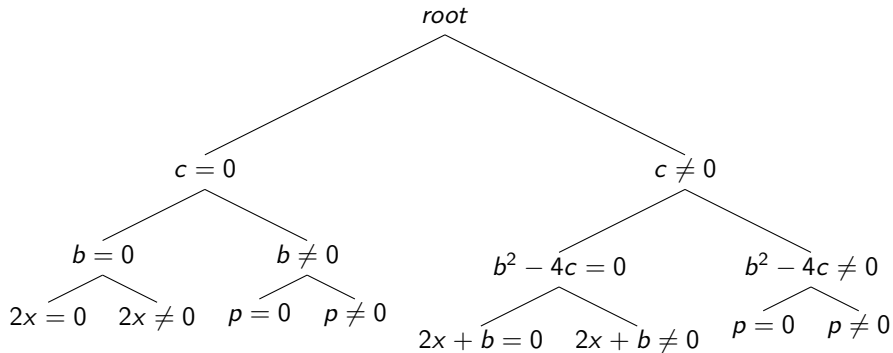


Figure: Complete complex cylindrical tree for the general monic quadratic equation, $p := x^2 + bx + c$, under variable ordering $c \prec b \prec x$.

Experiments

From: www.cs.bath.ac.uk/~djw42/RCTTICADexamples.txt

		RC-TTICAD [BCD ⁺ 14]		RC-Inc-CAD [CMM12]		MMA	QEPCAD	
Problem	n	Cells	Time	Cells	Time	Time	Cells	Time
MontesS10	7	3643	19.1	3643	28.3	T/O	T/O	—
Wang 93	5	507	44.4	507	49.1	897.1	FAIL	—
Rose	3	3069	200.9	7075	498.8	T/O	FAIL	—
gLS-3-2	11	222821	3087.5	—	T/O	T/O	FAIL	—
BC-P-4	4	543	1.6	2007	13.6	11.9	51763	8.6

In all these cases Redlog gave an error, and an old version of SyNRAC gave an error or timed out. Full details, including many cases where Mathematica did well, are in [BCD⁺14]

Conclusions

- TTICAD theory offers great advantages over both sign-invariant CAD and CAD with equational constraint.
- Allows for an unoptimised Maple implementation to compete with the state of the art.
- The timings for our implementation could certainly be improved using established techniques.
- Preferable would probably be the incorporation of TTICAD into the well-established software.

Future Algorithmic Work

- Can we use improved projection at more than the first level / make use of more than one equational constraint from a QFF?
- Can we avoid unnecessary lifting if the truth of a clause is already known?
- What can be done (for Projection/Lifting) when the input is not well-oriented? Note that Regular Chains doesn't have this issue.

So how do I use these tools?

That's actually a very good question: there's a lot of choice in how to phrase the question

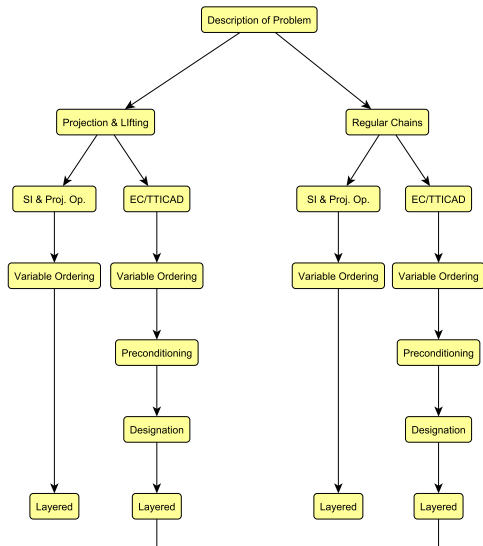
- 1 Choice of variable ordering (where permitted)
- 2 Choice of equalities
- 3 Choice of overall technology (Projection/Regular Chains/...)
- 4 Choice of how the problem is posed
- 5 (including Gröbner pre-conditioning)



Choice of software: no software has (even close to) all the techniques, and each has extra “features”

These are **not** independent questions

How might this look? Wilson's thesis





R.J. Bradford, C. Chen, J.H. Davenport, M. England,
M. Moreno Maza, and D.J. Wilson.

Truth Table Invariant Cylindrical Algebraic Decomposition by
Regular Chains.




In *Proceedings CASC 2014*, pages 44–58, 2014.



C.W. Brown and J.H. Davenport.

The Complexity of Quantifier Elimination and Cylindrical
Algebraic Decomposition.

In C.W. Brown, editor, *Proceedings ISSAC 2007*, pages 54–60,
2007.

-  R.J. Bradford, J.H. Davenport, M. England, S. McCallum, and D.J. Wilson.
Cylindrical Algebraic Decompositions for Boolean Combinations.
In Proceedings ISSAC 2013, pages 125–132, 2013.
-  C.W. Brown.
Improved Projection for Cylindrical Algebraic Decomposition.
J. Symbolic Comp., 32:447–465, 2001.
-  C. Chen and M. Moreno Maza.
An Incremental Algorithm for Computing Cylindrical Algebraic Decompositions.
<http://arxiv.org/abs/1210.5543>, 2012.

-  C. Chen, M. Moreno Maza, B. Xia, and L. Yang.
Computing Cylindrical Algebraic Decomposition via Triangular Decomposition.
In J. May, editor, *Proceedings ISSAC 2009*, pages 95–102, 2009.
-  G.E. Collins.
Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition.
In *Proceedings 2nd. GI Conference Automata Theory & Formal Languages*, pages 134–183, 1975.
-  J.H. Davenport and J. Heintz.
Real Quantifier Elimination is Doubly Exponential.
J. Symbolic Comp., 5:29–35, 1988.



H. Hong.

Improvements in CAD-Based Quantifier Elimination.

PhD thesis, OSU-CISRC-10/90-TR29 Ohio State University,
1990.



S. McCallum.

An Improved Projection Operation for Cylindrical Algebraic
Decomposition of Three-dimensional Space.

J. Symbolic Comp., 5:141–161, 1988.



O. Zariski.

On equimultiple subvarieties of algebraic hypersurfaces.

Proc. Nat. Acad. Sci. USA, 72:1425–1426, 1975.