Beyond Equational Constraints in Cylindrical Algebraic Decomposition

James Davenport
University of Bath

Joint work with: Russell Bradford, Matthew England, Scott McCallum and David Wilson

2013 SIAM Conference on Applied Algebraic Geometry
Colorado State Universty.
1–4 August 2013

Supported by EPSRC Grant EP/J003247/1.

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
   - Conclusions

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
   - Conclusions

Cylindrical algebraic decomposition

A Cylindrical Algebraic Decomposition (CAD) is a partition of $\mathbb{R}^n$ into cells arranged cylindrically (meaning their projections by dropping trailing coordinates 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.

Originally motivated for use in quantifier elimination. Has also been applied directly on problems as diverse as algebraic simplification and robot motion planning, essentially because the output is very explicit.

Cylindrical algebraic decomposition

A Cylindrical Algebraic Decomposition (CAD) is a partition of $\mathbb{R}^n$ into cells arranged cylindrically (meaning their projections by dropping trailing coordinates 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.

Originally motivated for use in quantifier elimination. Has also been applied directly on problems as diverse as algebraic simplification and robot motion planning, essentially because the output is very explicit.

Cylindrical algebraic decomposition

A Cylindrical Algebraic Decomposition (CAD) is a partition of $\mathbb{R}^n$ into cells arranged cylindrically (meaning their projections by dropping trailing coordinates 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.

Originally motivated for use in quantifier elimination. Has also been applied directly on problems as diverse as algebraic simplification and robot motion planning, essentially because the output is very explicit.

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

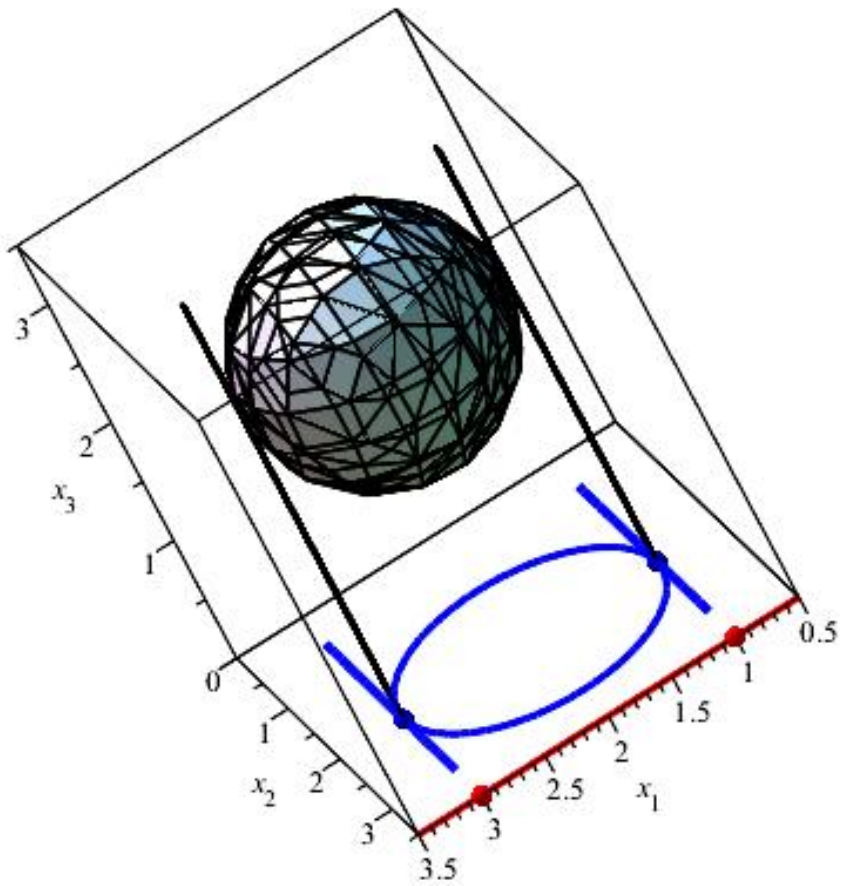
- A CAD of $\mathbb{R}$ is produced using the roots of the univariate polynomials and intervals between.
- Over each cell: the bivariate polynomials are evaluated at a sample point, a stack is built consisting of sections (the roots) and sectors (the intervals). Together these are a CAD of $\mathbb{R}^2$.

⋮

- 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

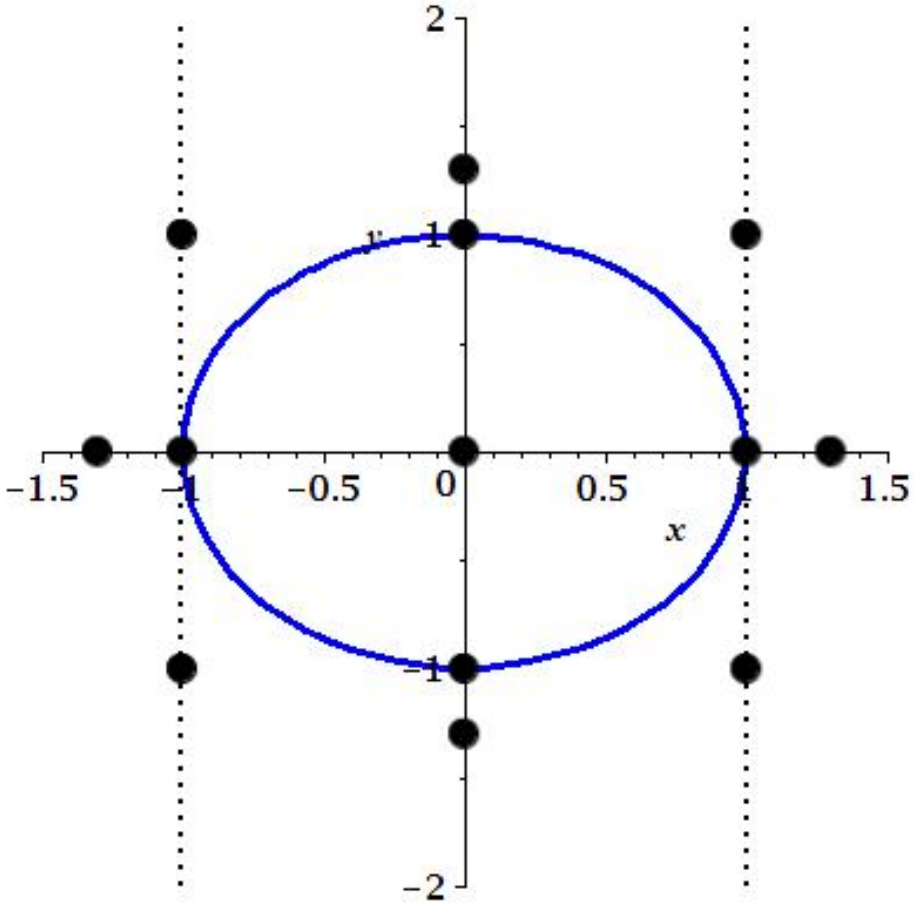
- A CAD of $\mathbb{R}$ is produced using the roots of the univariate polynomials and intervals between.
- Over each cell: the bivariate polynomials are evaluated at a sample point, a stack is built consisting of sections (the roots) and sectors (the intervals). Together these are a CAD of $\mathbb{R}^2$.

$\vdots$

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

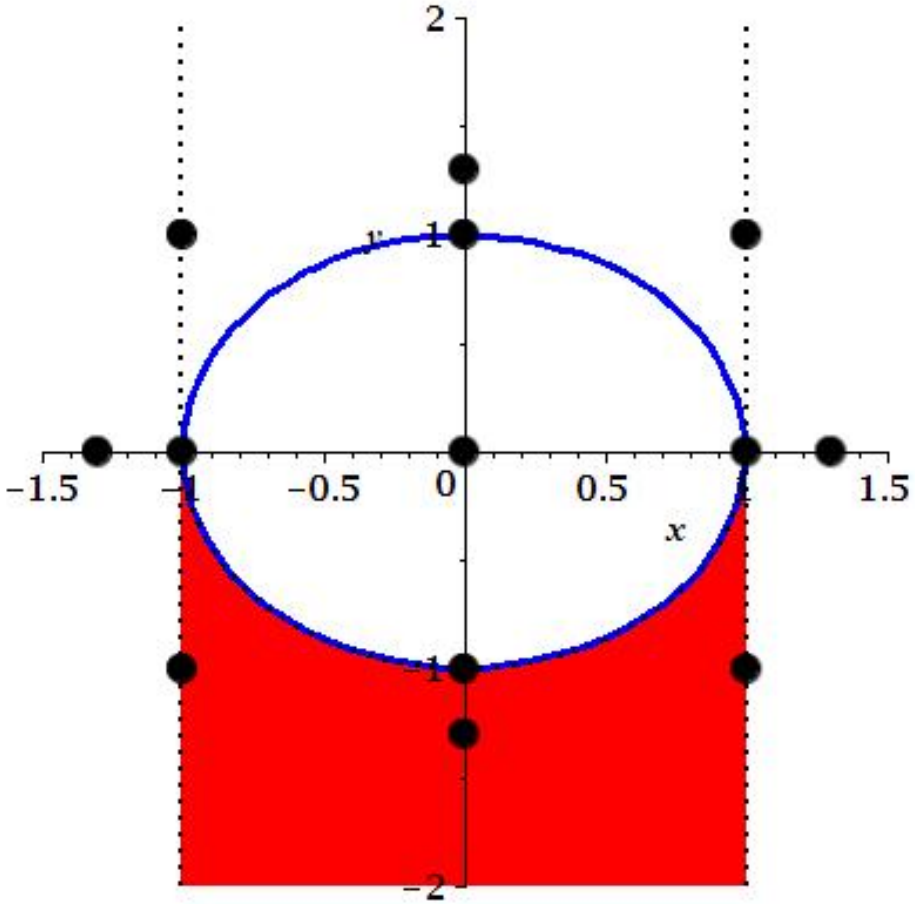
The projection operator is defined so the CAD is sign-invariant.

Lifting example



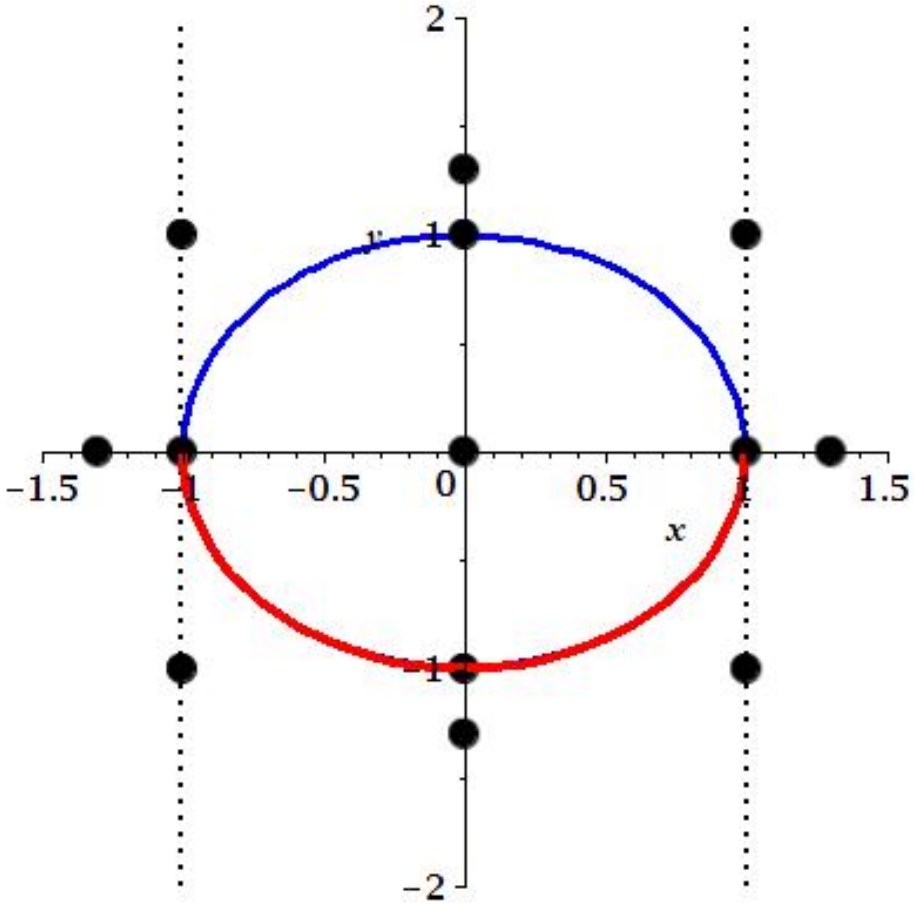
A CAD of $\mathbb{R}^2$ which is sign-invariant with respect to the circle. Each black dot represents a cell.

Lifting example



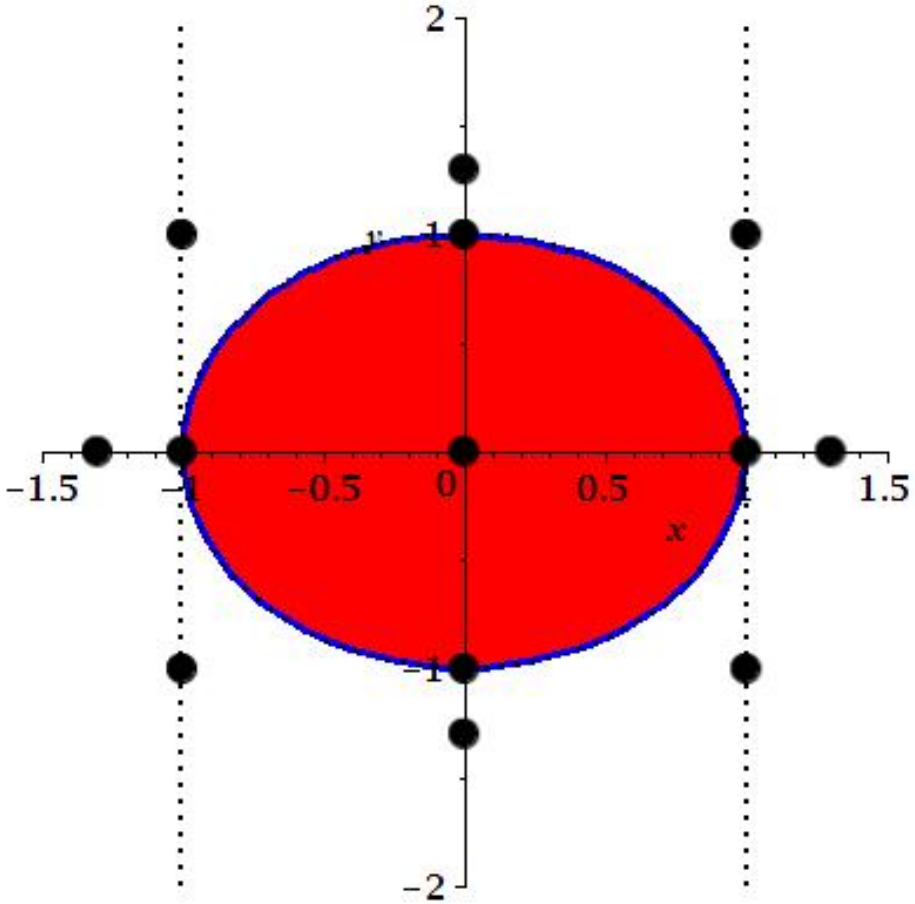
A CAD of $\mathbb{R}^2$ which is sign-invariant with respect to the circle. Each black dot represents a cell.

Lifting example



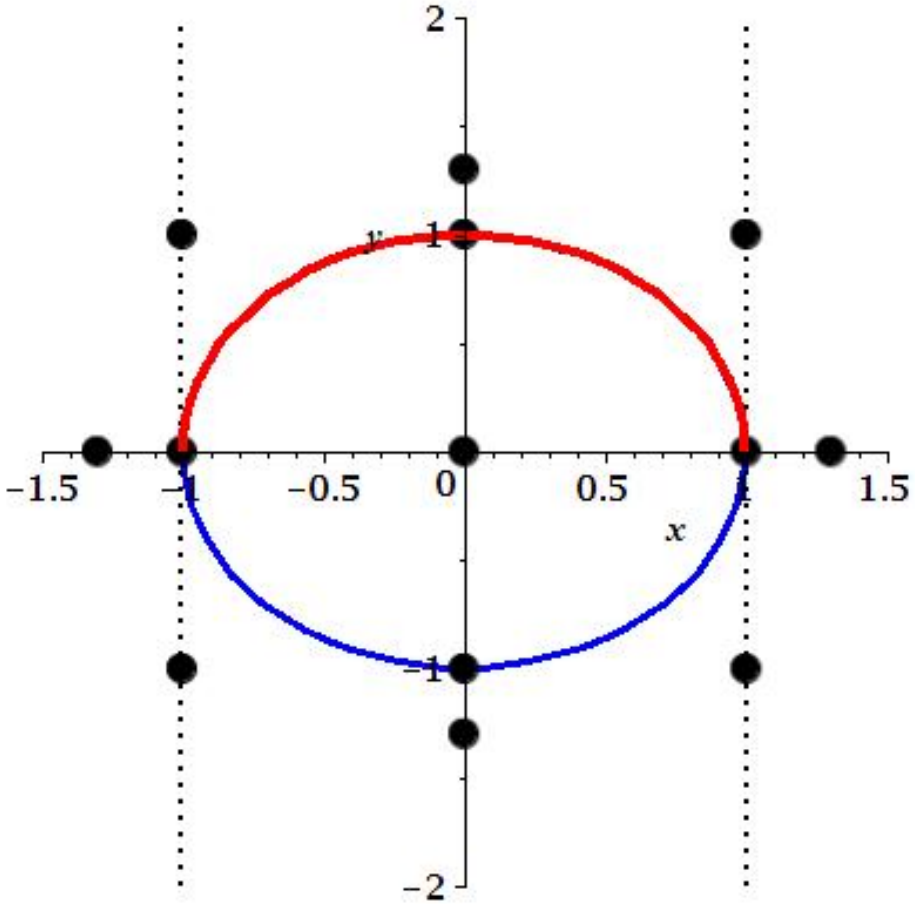
A CAD of $\mathbb{R}^2$ which is sign-invariant with respect to the circle. Each black dot represents a cell.

Lifting example



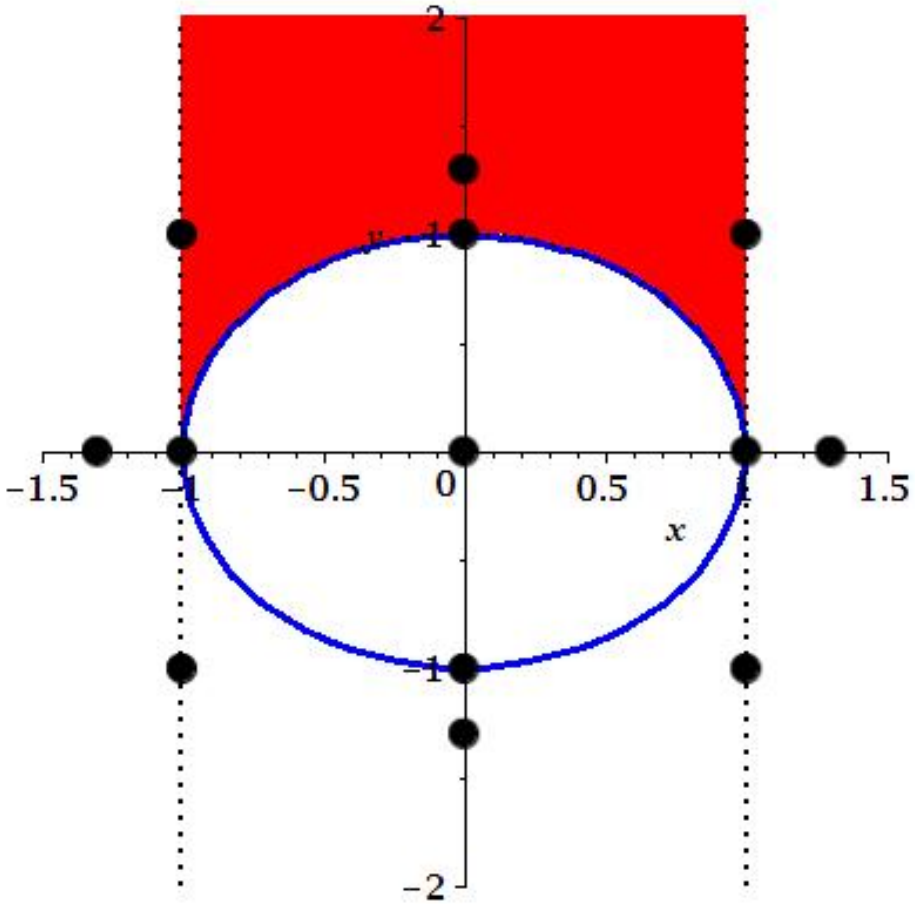
A CAD of $\mathbb{R}^2$ which is sign-invariant with respect to the circle. Each black dot represents a cell.

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.

CAD is powerful — too powerful

A CAD to resolve $\exists y \forall z \quad f(x,y,z) = 0 \wedge g(x,y,z) > 0$ will also resolve

- $\forall y \exists z \quad f(x,y,z) = 0 \wedge g(x,y,z) > 0$
- $\forall y \forall z \quad f(x,y,z) < 0 \vee g(x,y,z) = 0$
- $\exists z \quad f(x,y,z) = 0 \Rightarrow g(x,y,z) > 0$

and any other formula where the quantified variables occur in the same order.

As a consequence, CAD is doubly exponential in the number of variables [DH88, BD07]

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 is aborted if the value of the quantified formula on the whole stack is already apparent.

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.

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 give 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.

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
   - Conclusions

Simple motivating example

Consider the polynomials:

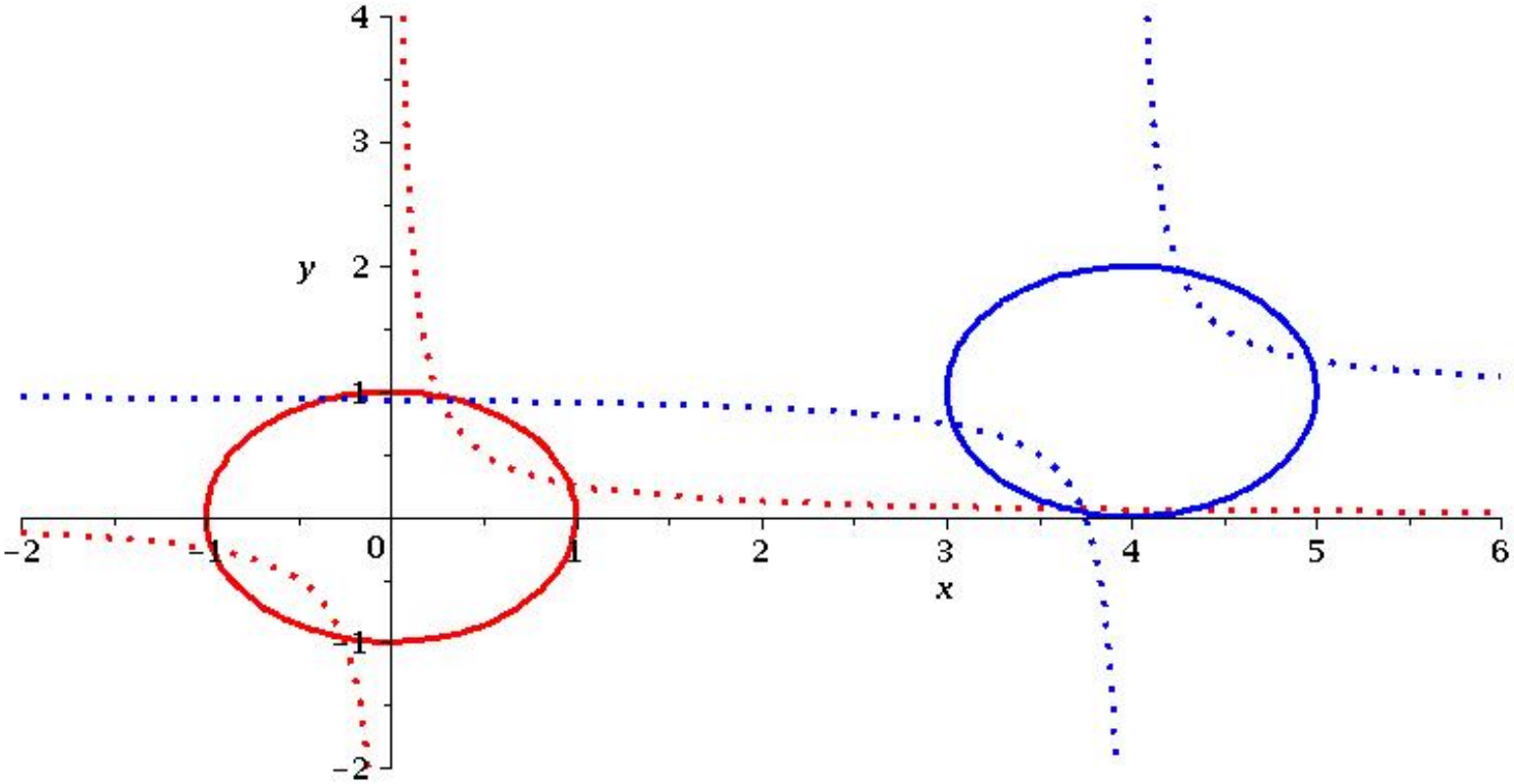
$$f_1 := x^2 + y^2 - 1 \qquad g_1 := xy - \tfrac{1}{4}$$
$$f_2 := (x-4)^2 + (y-1)^2 - 1 \qquad g_2 := (x-4)(y-1) - \tfrac{1}{4}$$

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



Simple motivating example

Consider the polynomials:

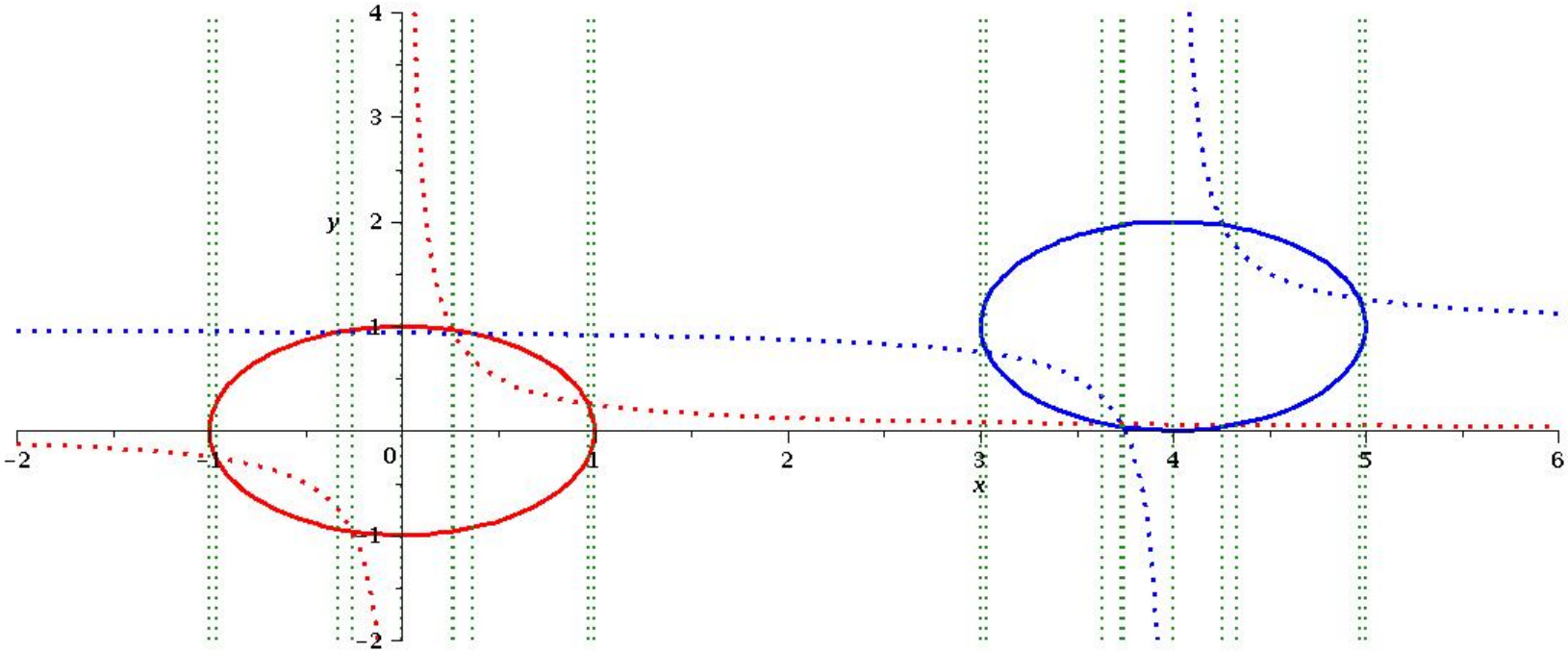
$$f_1 := x^2 + y^2 - 1 \qquad g_1 := xy - \tfrac{1}{4}$$
$$f_2 := (x-4)^2 + (y-1)^2 - 1 \qquad g_2 := (x-4)(y-1) - \tfrac{1}{4}$$

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: sign-invariant CAD



All curve intersections identified.

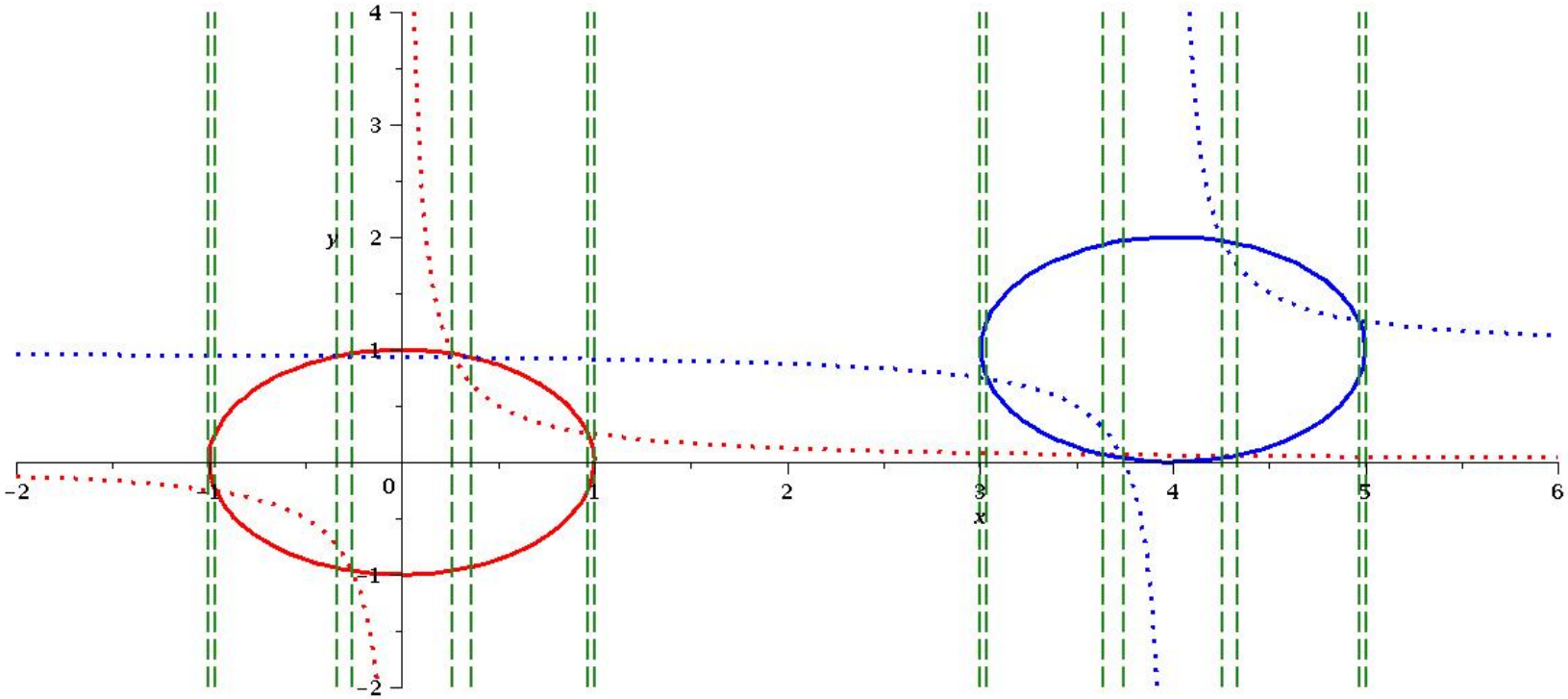
Simple motivating example continued

We could instead employ the theory of equational constraints.

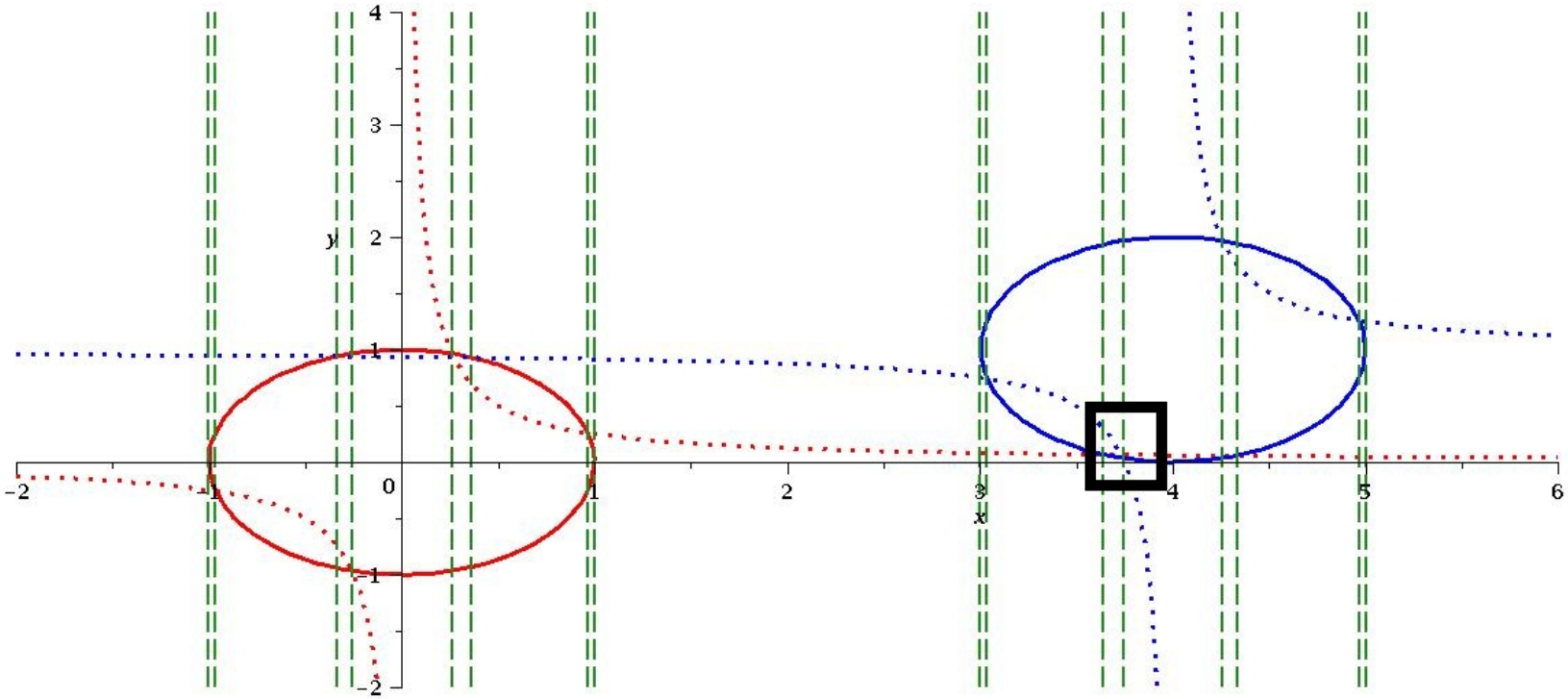
Although Φ has no explicit equational constraint the equation $f_1 f_2 = 0$ is implied implicitly.

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

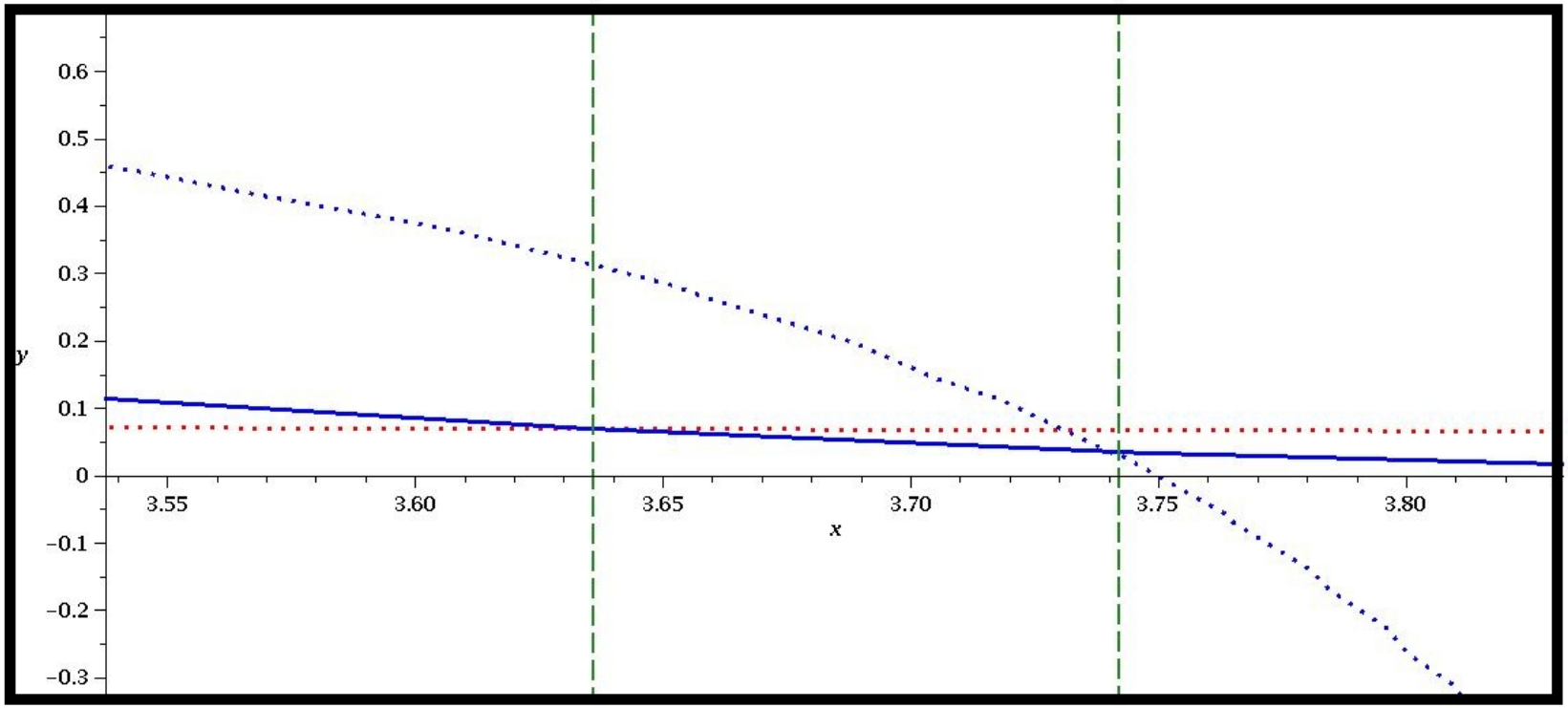
Example: CAD with equational constraint



Example: CAD with equational constraint



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:

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

where

$$\begin{aligned} P_{E_i}(A_i) &= P(E_i) \cup \{\mathrm{res}_{x_n}(e, a)\}_{e \in E_i,\, a \in A_i \setminus E_i} \\ P(A) &= \{\mathrm{disc}(a),\, \mathrm{coeffs}_{x_n}(a),\, \mathrm{res}_{x_n}(a, b)\}_{a,b \in A} \\ \mathrm{Res}^{\times}(\mathcal{E}) &= \{\mathrm{res}_{x_n}(e, \hat{e}) \mid \exists i, j :\ e \in E_i, \hat{e} \in E_j, i < j, e \neq \hat{e}\} \end{aligned}$$

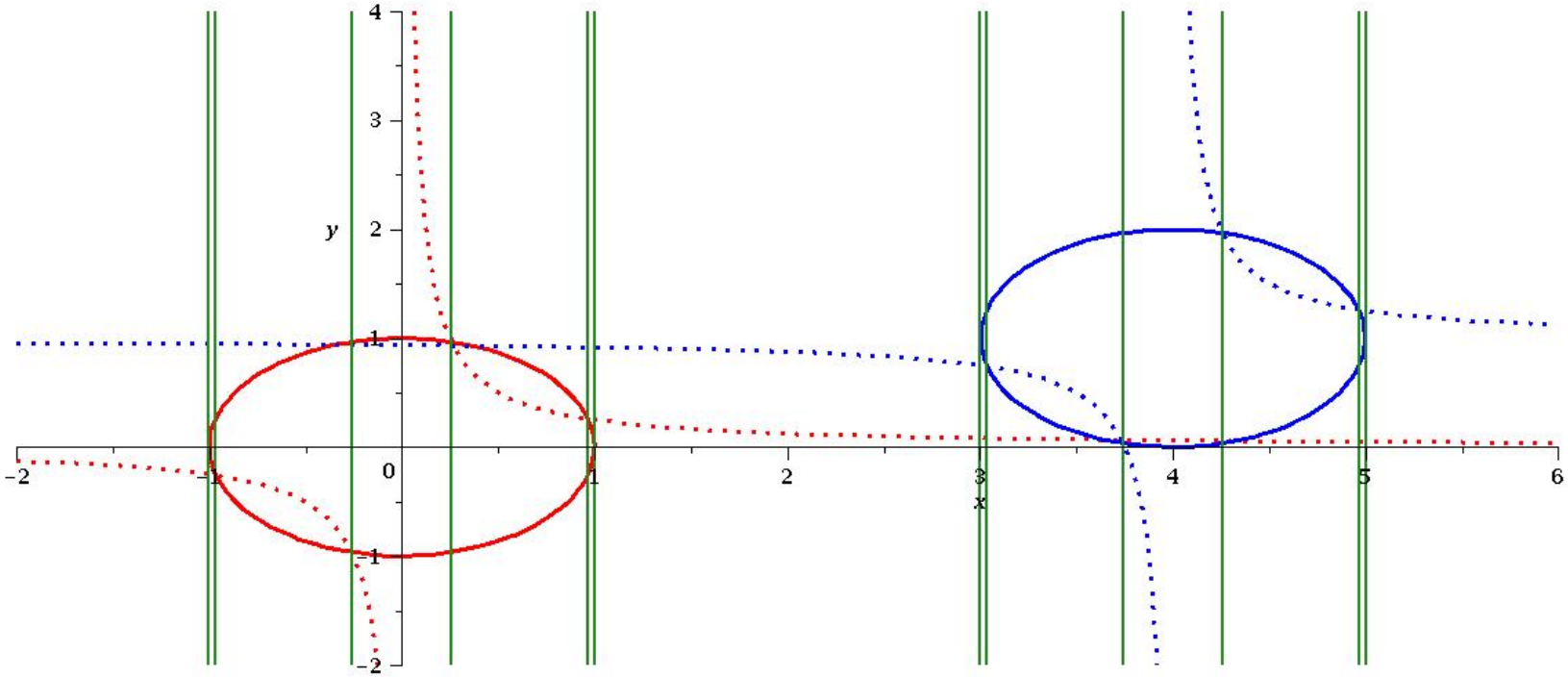
Using the operator to build a TTICAD

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.

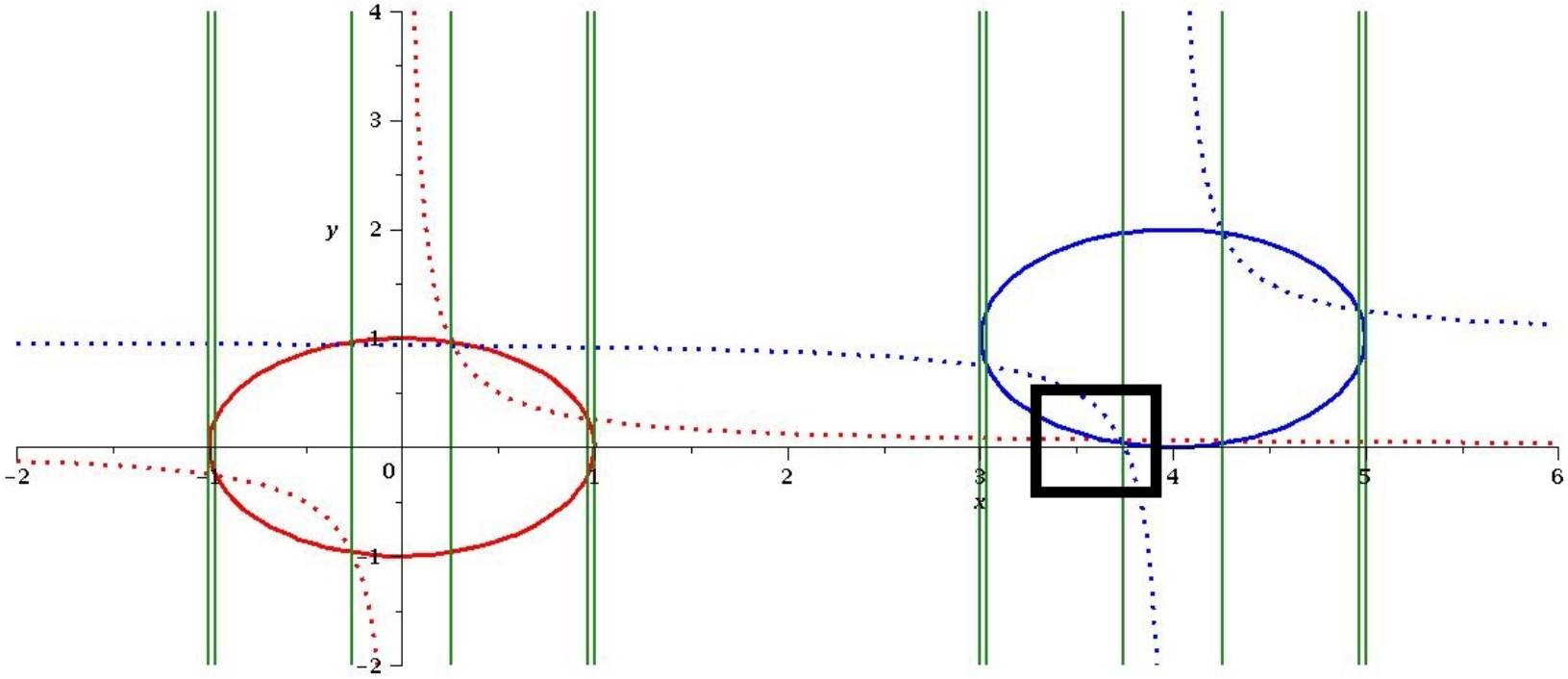
Example: TTICAD

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).



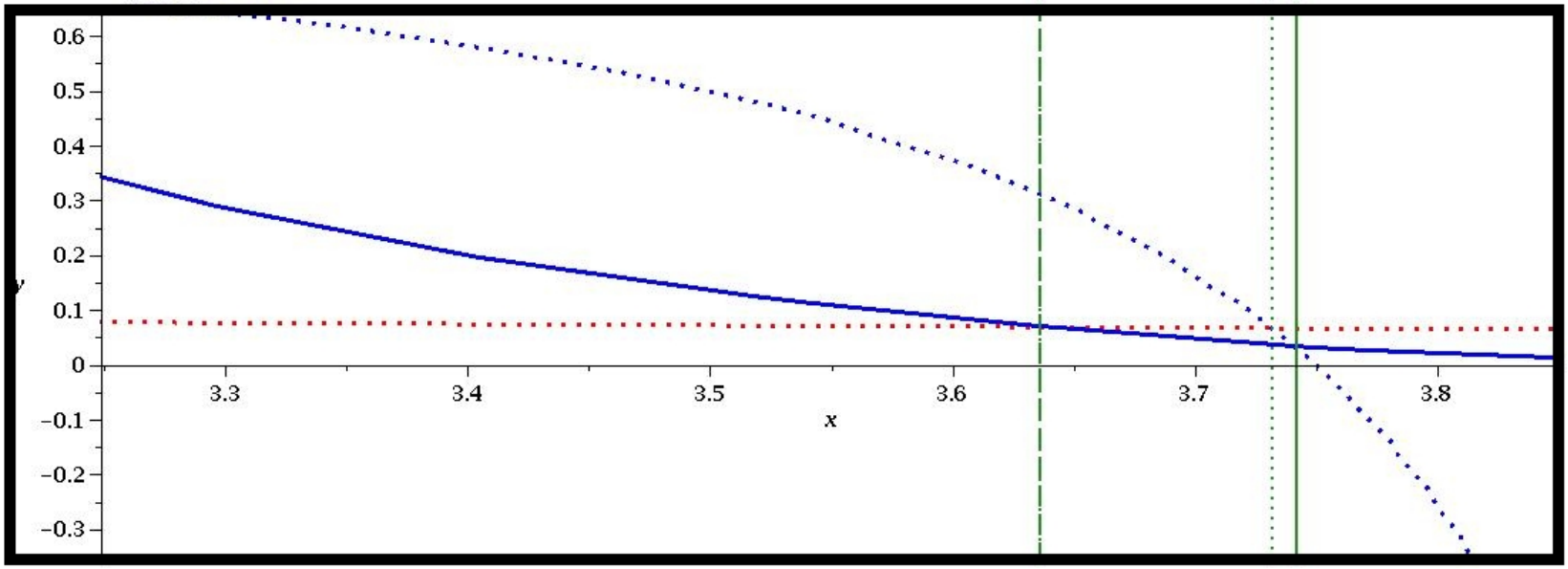
Example: TTICAD

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).



Example: TTICAD

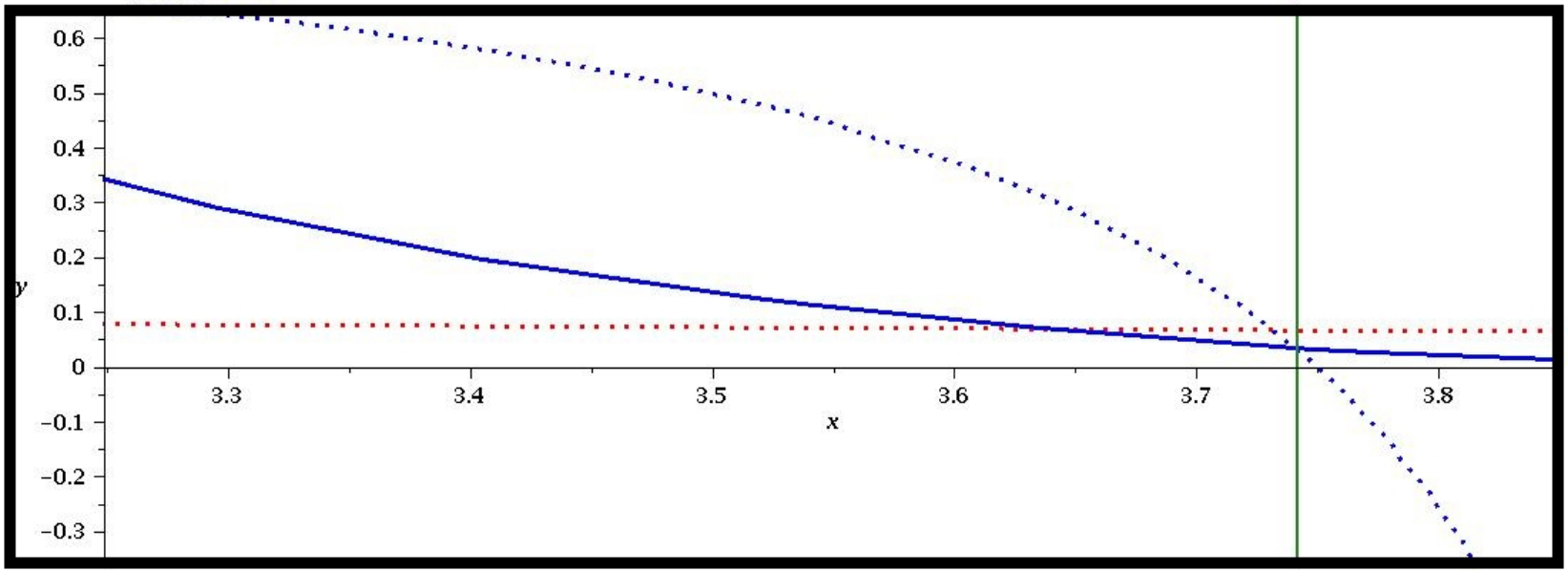
A TTICAD for the motivating example is built with 105 cells (compared to 317 and 249). This identified 12 points on the real line (compared to 20 and 16).



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 require 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.

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 require 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.

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
- Conclusions

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 and 17. 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 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

Next 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.

Conclusions

- TTICAD theory offers great advantages over both sign-invariant CAD and CAD with equational constraint.
- Allows for an unoptimised 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.

Conclusions

- TTICAD theory offers great advantages over both sign-invariant CAD and CAD with equational constraint.
- Allows for an unoptimised 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 Work

- 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.

- 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 when the input is not well-oriented?

Future Work

- 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.

- 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 when the input is not well-oriented?

Future Work

- 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.

- 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 when the input is not well-oriented?

Further Information I

S. McCallum
An improved projection operator for cylindrical algebraic decomposition.
Quantifier Elimination and Cylindrical Algebraic Decomposition, pages 242–268, Springer, 1998.

S. McCallum
On Projection in CAD-based Quantifier Elimination with Equational Constraints.
Proc. ISSAC '98, pages 145–149, ACM, 1998.

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.

Further Information II

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.

J.H. Davenport and J. Heintz.
Real Quantifier Elimination is Doubly Exponential.
J. Symbolic Comp., 5:29–35, 1988.

Contact Details

J.H.Davenport@bath.ac.uk
http://staff.bath.ac.uk/masjhd/