# Cylindrical Algebraic Decomposition with Equational Constraints

**Matthew England**
Coventry University

**CAD Journal Club**
Coventry University
30th April 2021

# Outline

# Outline

# Outline

# Quantifier Elimination

Quantifier Elimination (QE): Given a logical formula with quantifiers $\forall$ (for all) or $\exists$ (there exists) QE means to identify an equivalent quantifier free formula.

Real QE: Quantifier Elimination in the case where the atoms of the input formula are sign constraints on polynomials with rational coefficients. For example:

$$(\exists x)(x^2 + ax + b = 0) \iff a^2 - 4b \geq 0$$

Real QE has numerous applications throughout engineering and the sciences.

Tarski shows that Real QE is always possible in 1940s. Collins provided the first a tractable method in the 1970s with CAD.

# History

QE is formerly a problem of computational logic. Whether it is possible depends on the domain of the formula atoms. Algorithms to achieve it (when its possible) must be tailored to those domains.

The algorithms for RealQE have been developed in the field of Computer Algebra. Ignored by those working in computational logic (until recently). Similarly, computational algebra tended to ignore the logical structure. Common approach:

- Take the set of polynomials present in formula.
- Build a CAD sign-invariant for them.
- Examine the constraints at a sample point of each cell.

The main algorithm (CAD) is blind to the logical structure!

**Exception:** Equational Constraints – a CAD optimisation made with respect to the logical structure.

# The Recent Past and DEWCAD

Things have changed in the last decade:

- In 2012 Jovanovic and De Moura (Microsoft) developed the NLSAT algorithm which applied a version of Collin's CAD theory within an SMT solver.
- SMT-RAT since 2011(?) developed at Aachen with variety of modules based on computer algebra algorithms.
- The EU SC$^2$ project ran from 2016-2018 to bring together researchers from Symbolic Computation and Satisfiability Checking. Numerous new collaborations started.
- DEWCAD project funded by UK's EPSRC for 2021-2024 to develop (amongst many other things) an SMT solver within a computer algebra system (Maple).

So Logic and Algebra live in harmony now! But: Equational Constraint savings still make sense even in the SMT framework.

# Outline

# Key Message on Complexity

Conventional wisdom is that working with polynomial systems is doubly exponential.

Key message of this talk is that while that is true, what they are doubly exponential in can vary a lot.

Particular result of the accompanying paper is that CAD is doubly exponential in $n - \ell$ where

- $n$ is number of variables;
- $\ell$ is number of (primitive) equational constraints (in different main variables).

# What do we mean by polynomial systems?

E.g. Algorithms for solving problems to do with polynomial systems. In particular:

Gröbner Bases (GB): for working with systems of polynomial equations; and ideals they define.
Building a GB allows us to easily find the dimension, the number of zeros, etc.

Cylindrical Algebraic Decomposition (CAD): for working with systems of polynomial constraints (so also $<, <=, >, >=, <>$).
Building a CAD allows us to describe simply the true regions, perform quantifier elimination, etc.

# What do we mean by doubly exponential?

These algorithms are often described simply as doubly exponential.

This really means:
**doubly exponential in the number of variables *n*.**

All other dependencies, such as the number of polynomials $m$ or their degree $d$, are polynomial in their quantities (albeit with an exponent possibly exponential in $n$).

# Key Message on Complexity

Conventional wisdom is that working with polynomial systems is doubly exponential.

**Key message** of this talk is that while that is true, what they are doubly exponential in can vary, and need not be the number of variables.

Particular result of the accompanying paper is that CAD is doubly exponential in $n - \ell$ where

- $n$ is number of variables;
- $\ell$ is number of (primitive) equational constraints (in different main variables).

# Inspiration from Gröbner Bases

- A particularly useful generating set $G$ of an ideal $I$.
- The ideal generated by the leading terms of $I$ is generated by the leading terms of the GB
- Introduced by Prof. Buchberger in 1965.

E.W. Mayr and A.R. Meyer.
*The complexity of the word problems for commutative semigroups and polynomial ideals*.
Advances in Mathematics, 46(3):305–329, 1982.

Showed that the computation of Gröbner bases is doubly exponential in the number of variables.

# GB Complexity

D. Lazard.
*Gröbner Bases, Gaussian elimination and resolution of systems of algebraic equations.*
In Proc. EUROCAL 83, pages 146–157, 1983.

But, complexity of GB for a zero-dimensional ideal is *only* singly-exponential in $n$.

E.W. Mayr and S. Ritscher.
*Dimension-dependent bounds for Gröbner bases of polynomial ideals.*
Journal of Symbolic Computation, 49:78–94, 2013.

In fact, there are upper and lover bounds for GB which are singly exponential in $n$, but *doubly* exponential in $r$ where $r$ is the actual dimension of the ideal. Only in the worst case is $r = n$.

# A Recent Example

Tereso spoke to use recently about the preprint paper:

H. Li, B. Xia, H. Zhang and T. Zheng.
*Choosing the Variable Ordering for Cylindrical Algebraic Decomposition via Exploiting Chordal Structure.*
Preprint: ArXiv 2102.00823, 2021.

They showed that with a good choice of ordering that preserves chordality the double exponent was in elimination tree height ($\leq$ number of variables).

# Equational Constraints

An Equational Constraint (EC) is a polynomial equation that is logically implied by a formula. I.e. if the formula is true then the equation must be satisfied.

Not all equations in a QE input are equational constraints: we need to look at the logical structure to determine.

Informally: the dimension of the solution space should reduce by one for every EC we have.

Can we somehow reduce the computation also? Yes, but it is not trivial to extract the full savings!

# Key Message on Complexity

Conventional wisdom is that working with polynomial systems is doubly exponential.

Key message of this talk is that while that is true, what they are doubly exponential in can vary a lot.

The particular result the author has worked on shows that CAD is doubly exponential in $n - \ell$ where

- $n$ is number of variables;
- $\ell$ is number of (primitive) equational constraints (in different main variables).

# Key Message on Complexity

Conventional wisdom is that working with polynomial systems is doubly exponential.

Key message of this talk is that while that is true, what they are doubly exponential in can vary a lot.

The particular result the author has worked on shows that CAD is doubly exponential in $n - \ell$ where

- $n$ is number of variables;
- $\ell$ is number of (primitive) equational constraints (in different main variables).

# Outline

# Outline

# What is a CAD?

A Cylindrical Algebraic Decomposition (CAD) is a mathematical object (but commonly refers also to an algorithm which creates one). Defined in the 1970s by George Collins who also gave the first CAD algorithm. A CAD is:

- a decomposition meaning a partition of $\mathbb{R}^n$ into connected subsets called cells;
- (semi)-algebraic meaning that each cell can be defined by a sequence of polynomial equations and inequations.
- cylindrical meaning the cells are arranged in a useful manner - their projections (relative to a given variable ordering) are either equal or disjoint.

Those projections map onto an induced CAD of the lower dimensional space. The cells with equal projections form a cylinder over one cell from the CAD in the lower space.

# What is a CAD?

A Cylindrical Algebraic Decomposition (CAD) is a mathematical object (but commonly refers also to an algorithm which creates one). Defined in the 1970s by George Collins who also gave the first CAD algorithm. A CAD is:

- a decomposition meaning a partition of $\mathbb{R}^n$ into connected subsets called cells;
- (semi)-algebraic meaning that each cell can be defined by a sequence of polynomial equations and inequations.
- cylindrical meaning the cells are arranged in a useful manner - their projections (relative to a given variable ordering) are either equal or disjoint.

Those projections map onto an induced CAD of the lower dimensional space. The cells with equal projections form a cylinder over one cell from the CAD in the lower space.

# What is a CAD?

A Cylindrical Algebraic Decomposition (CAD) is a mathematical object (but commonly refers also to an algorithm which creates one). Defined in the 1970s by George Collins who also gave the first CAD algorithm. A CAD is:

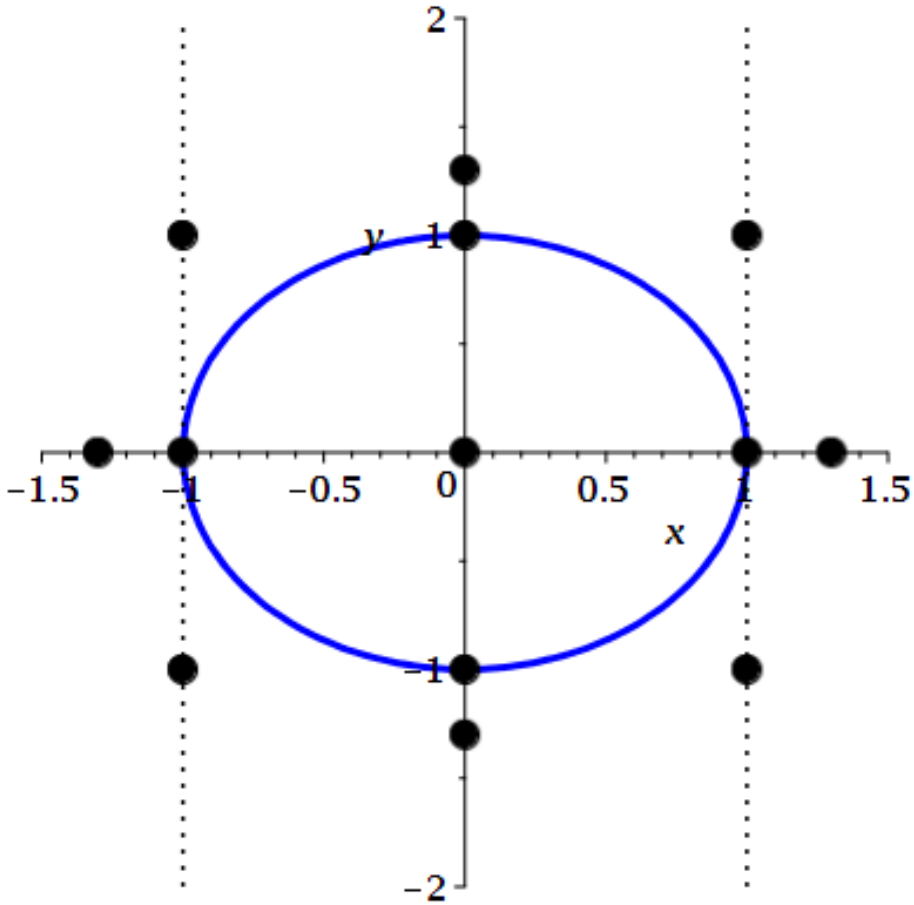
- a decomposition meaning a partition of $\mathbb{R}^n$ into connected subsets called cells;
- (semi)-algebraic meaning that each cell can be defined by a sequence of polynomial equations and inequations.
- cylindrical meaning the cells are arranged in a useful manner - their projections (relative to a given variable ordering) are either equal or disjoint.

Those projections map onto an induced CAD of the lower dimensional space. The cells with equal projections form a cylinder over one cell from the CAD in the lower space.

# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

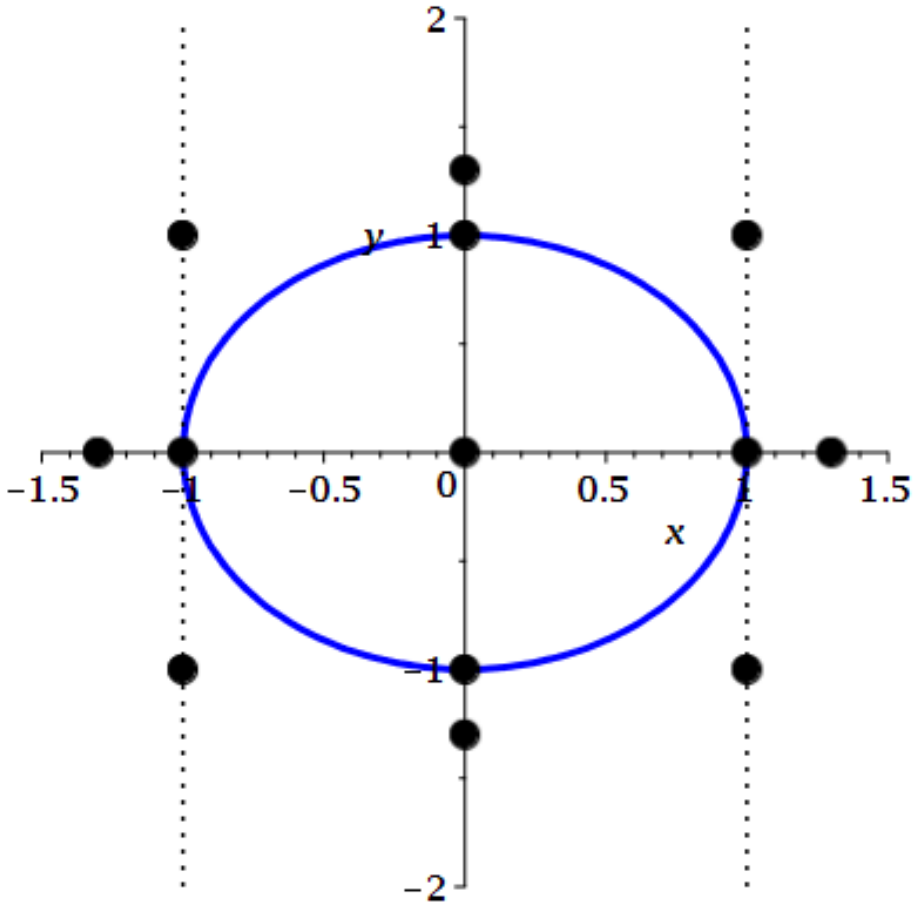
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

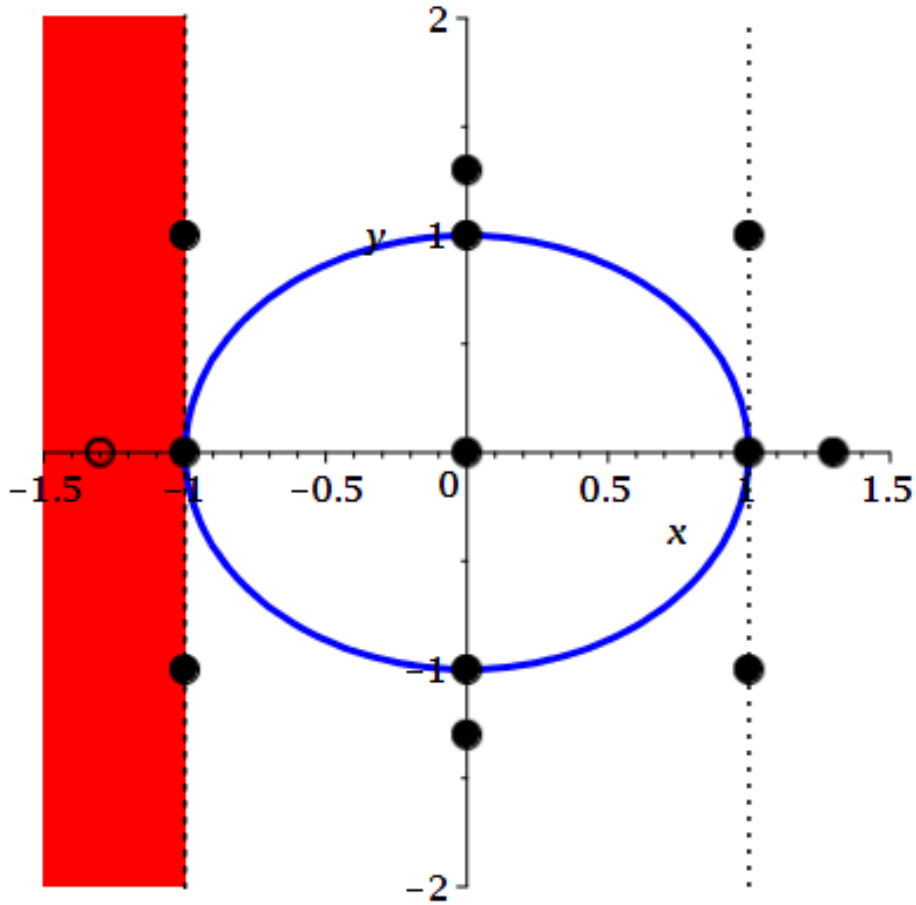
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

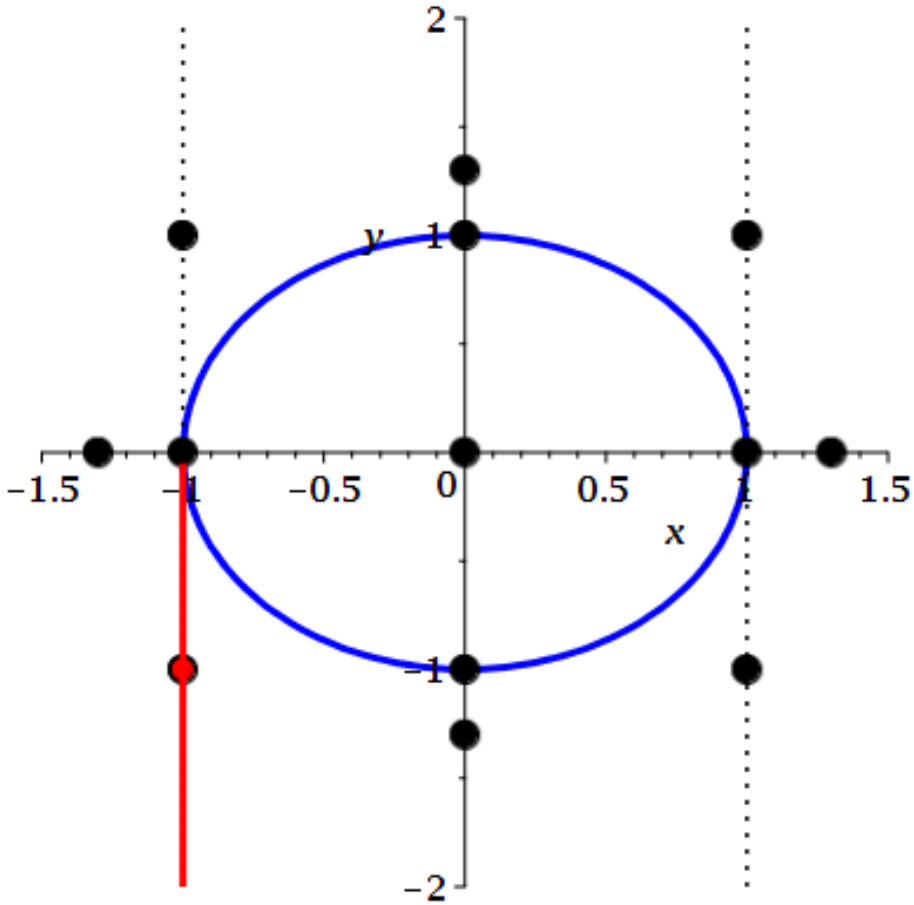
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

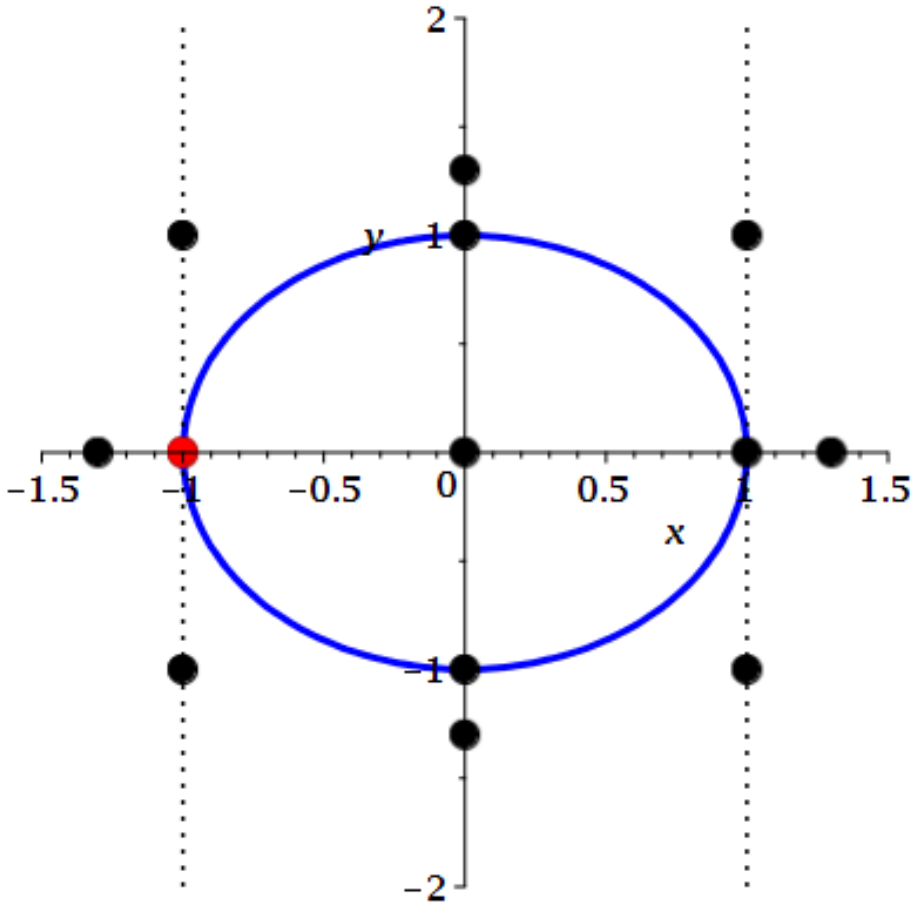
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

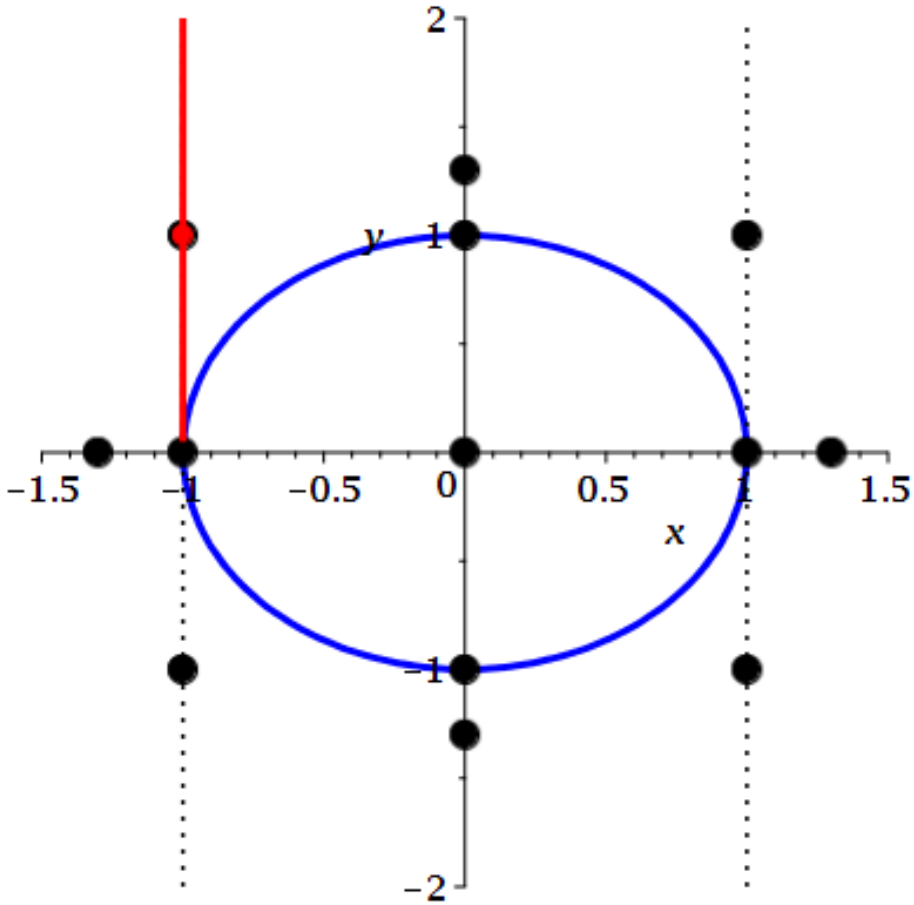
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

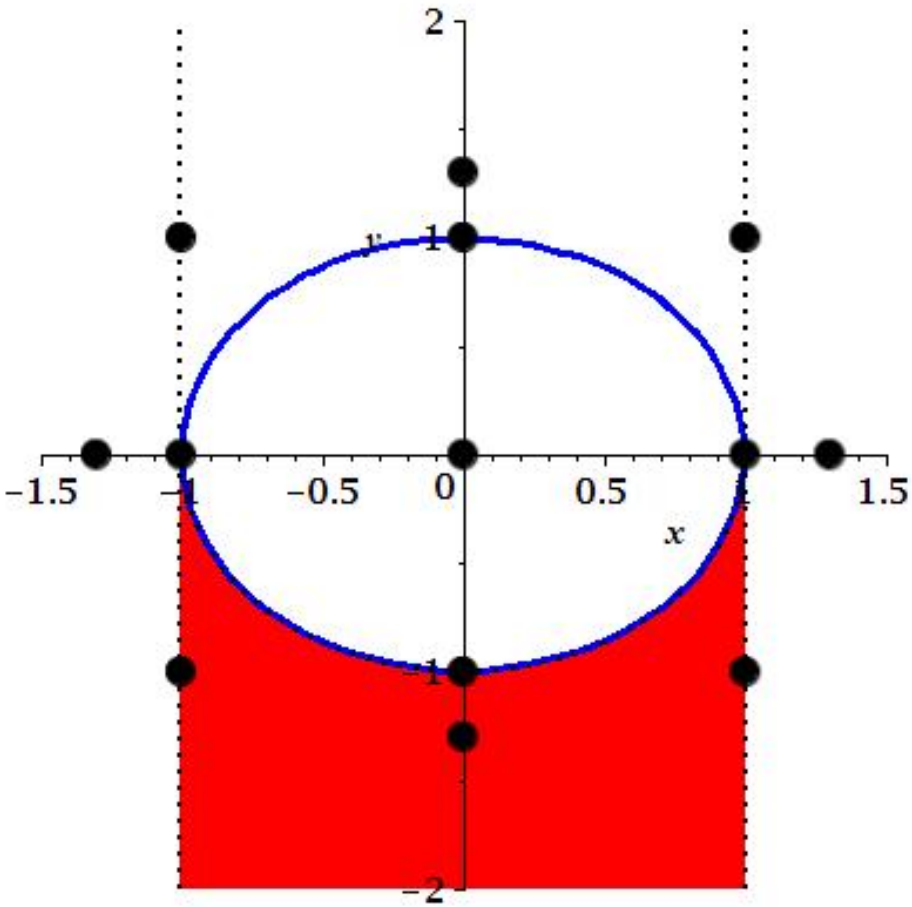
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

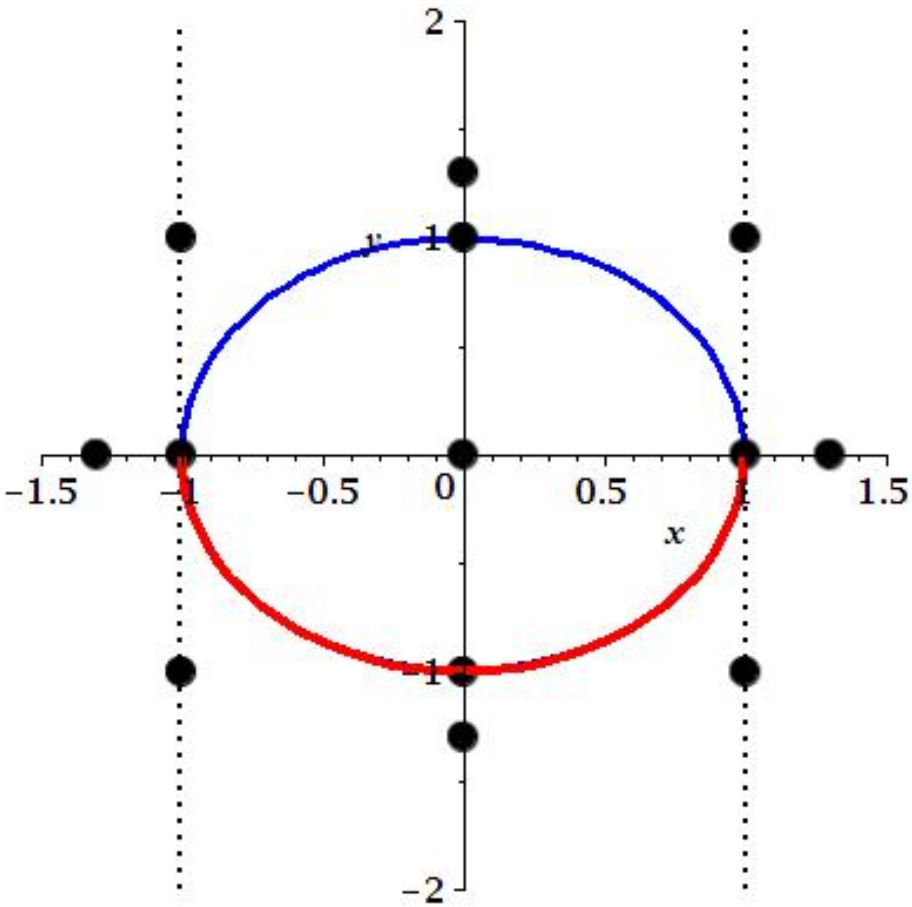
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

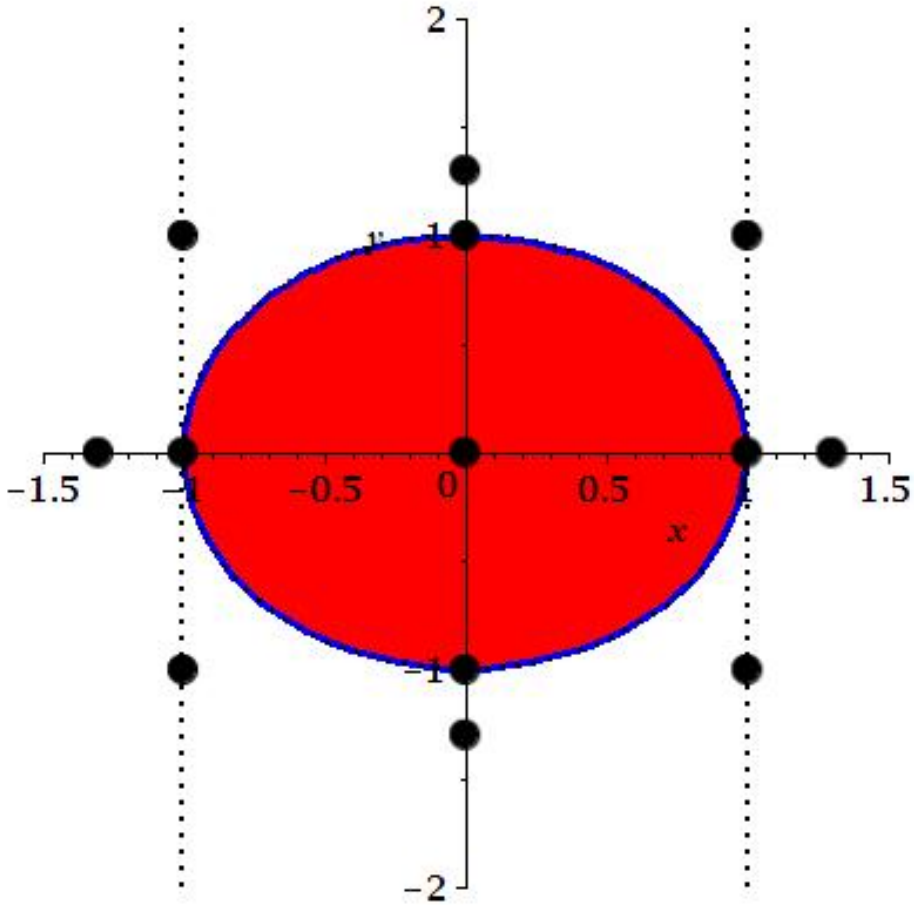
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

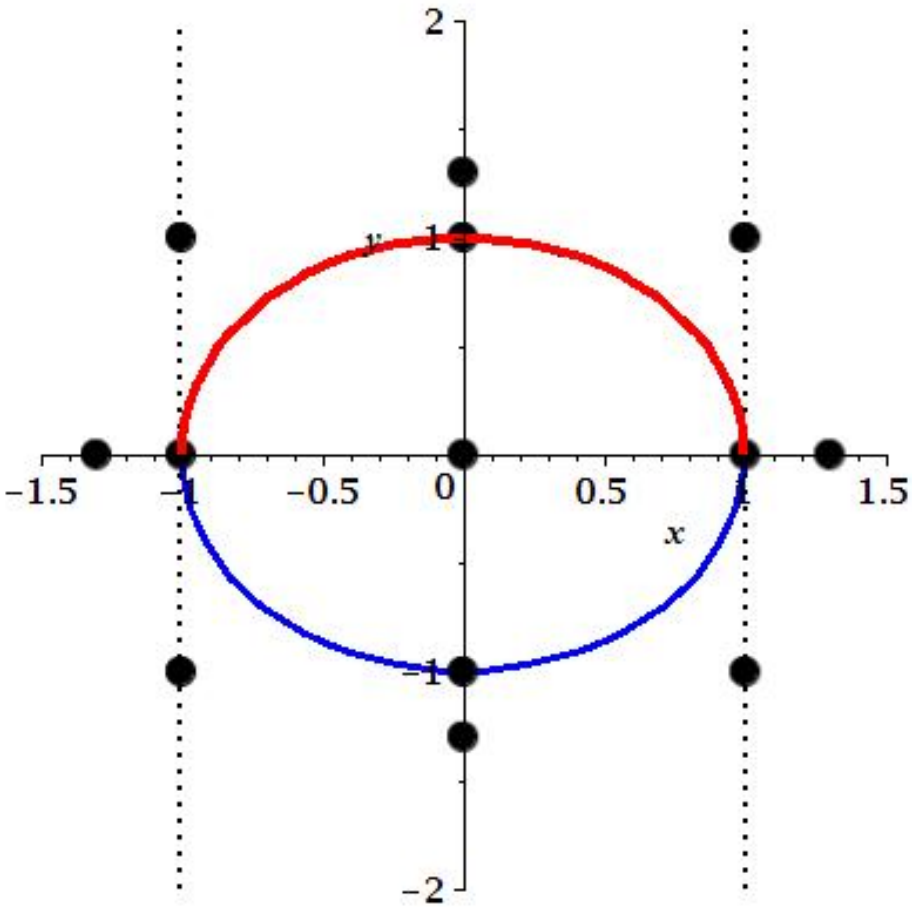
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

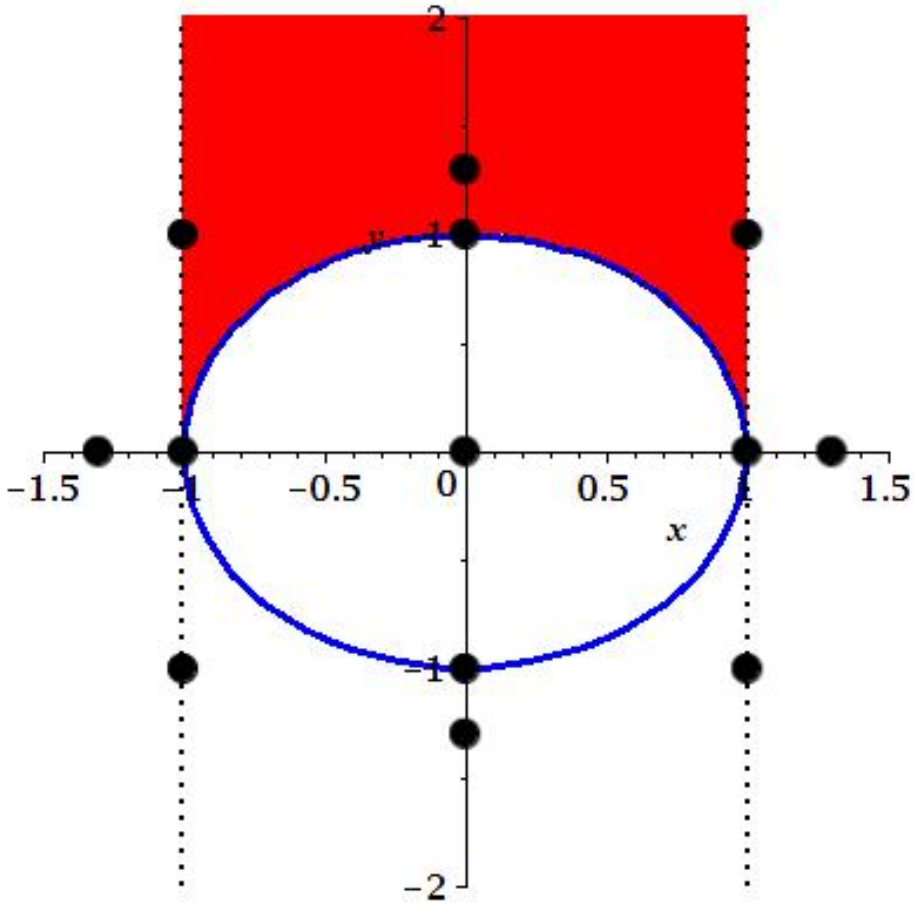
E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# Sign-invariance

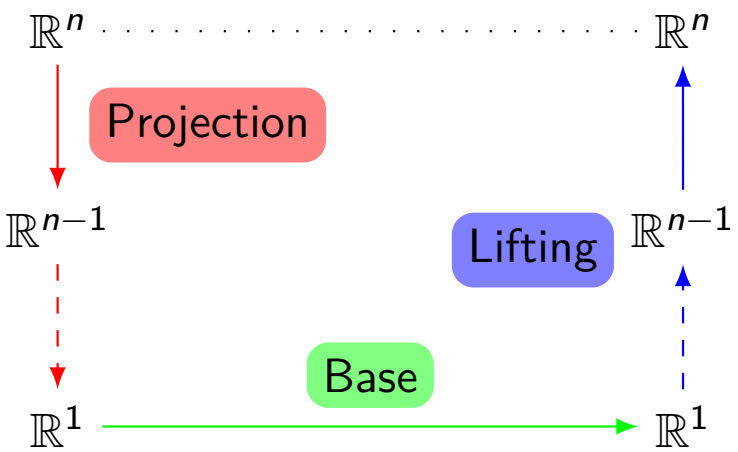
Traditionally a CAD is produced from a set of polynomials such that each polynomial has constant sign (positive, zero or negative) in each cell. Such a CAD is said to be sign-invariant.

E.g. A CAD produced to be sign-invariant for the polynomial $f := x^2 + y^2 - 1$ will commonly have 13 cells.



# How to build a CAD?

Projection: Apply an operator to the input polynomials in $\mathbb{R}^n$ to receive a set in $\mathbb{R}^{n-1}$. Repeat.
Base: Build CAD of $\mathbb{R}^1$ sign-invariant for univariate polynomials by isolating their real roots.
Lifting: Identify cells in $\mathbb{R}^2$ by: taking each cell in $\mathbb{R}^1$; evaluating bivariate polynomials at sample point; isolating real roots. Repeat.




The key mathematics is proving that working at a sample point guarantees sign-invariance over whole cell. Requires appropriate projection operator (usual ingredients: resultants, discriminants and coefficients).

# Outline

# Why build a CAD?

The original motivation for CAD was Quantifier Elimination (QE).

Given a polynomial formula with quantifiers $\forall$ (for all) or $\exists$ (there exists) QE means identify an equivalent quantifier free formula.

A CAD for the polynomials involved can perform QE over real numbers; thus solve problems throughout engineering & science.

- derivation of optimal numerical schemes (Erascu-Hong, 2014)
- artificial intelligence (Todai Robot Project)
- automated theorem proving (Paulson, 2012)
- bio-chemical network analysis (Bradford et al., 2017)
- automated loop parellisation (Grösslinger et al. 2006)
- analysis of economic hypotheses (Mulligan et al., 2018)

$\vdots$

# Why not build a CAD?

J.H. Davenport and J. Heintz.
*Real quantifier elimination is doubly exponential.*
Journal of Symbolic Computation, 5(1-2):29–35, 1988

CAD is worst case complexity doubly exponential.

C. Brown and J.H. Davenport.
*The complexity of quantifier elimination and cylindrical algebraic decomposition.*
In Proc. ISSAC '07, pages 54–60. ACM, 2007.

By the end of projection we have $M$ polynomials in $\mathbb{R}^1$, each of degree $D$, where $D = d^{2^{O(n)}}$ and $M = m^{2^{O(n)}}$. There are also lower bounds with $D = d^{2^{\Omega(n)}}$ and $M = m^{2^{\Omega(n)}}$ in which the underlying polynomials are all simple. The difficulty of CAD resides in the complicated number of ways simple polynomials can interact.

# Complexity Bound

R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson.
*Truth table invariant cylindrical algebraic decomposition.*
Journal of Symbolic Computation, 76:1–35, 2016.

A bound on the total number of cells produced for a sign-invariant CAD has dominant term

$$(2d)^{2^n-1} m^{2^n-1} 2^{2^{n-1}-1}.$$

- This was based on using McCallum's operator for sign-invariant CAD (although Brown and Lazard operators would have same double exponents).
- Studies show the cell count to be closely correlated to the actual timings of implementations.

# Proof Sketch

Table shows number and degree of polynomials produced by projection.

| Variables | Number | Degree |
|---|---|---|
| $n$ | $m$ | $d$ |
| $n-1$ | $2m^2$ | $2d^2$ |
| $n-2$ | $4m^4$ | $8d^4$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $n-r$ | $2^{2^{r-1}}m^{2^r}$ | $2^{2^r-1}d^{2^r}$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $1$ | $2^{2^{n-2}}m^{2^{n-1}}$ | $2^{2^{n-1}-1}d^{2^{n-1}}$ |

Let $m_r$ be number of polynomials in $r$ variables, and $d_r$ their degree.

Decompose real line according to real roots of univariate polynomials: at most $m_1 d_1$ roots so at most $(2m_1 d_1 + 1)$ cells.

Over each one we decompose the cylinder in the plane into at most $(2m_2 d_2 + 1)$ cells.

Hence in $\mathbb{R}^n$ at most: $\prod_{i=1}^{n}(2m_i d_i + 1)$ cells.

Omit $+1$s for dominant term, with has the closed form on prior slide.

# Proof Sketch

Table shows number and degree of polynomials produced by projection.

| Variables | Number | Degree |
|---|---|---|
| $n$ | $m$ | $d$ |
| $n-1$ | $2m^2$ | $2d^2$ |
| $n-2$ | $4m^4$ | $8d^4$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $n-r$ | $2^{2^{r-1}}m^{2^r}$ | $2^{2^r-1}d^{2^r}$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $1$ | $2^{2^{n-2}}m^{2^{n-1}}$ | $2^{2^{n-1}-1}d^{2^{n-1}}$ |

Let $m_r$ be number of polynomials in $r$ variables, and $d_r$ their degree.

Decompose real line according to real roots of univariate polynomials: at most $m_1 d_1$ roots so at most $(2m_1 d_1 + 1)$ cells.

Over each one we decompose the cylinder in the plane into at most $(2m_2 d_2 + 1)$ cells.

Hence in $\mathbb{R}^n$ at most: $\prod_{i=1}^{n}(2m_i d_i + 1)$ cells.

Omit $+1$s for dominant term, with has the closed form on prior slide.

# Proof Sketch

Table shows number and degree of polynomials produced by projection.

| Variables | Number | Degree |
|---|---|---|
| $n$ | $m$ | $d$ |
| $n-1$ | $2m^2$ | $2d^2$ |
| $n-2$ | $4m^4$ | $8d^4$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $n-r$ | $2^{2^{r-1}}m^{2^r}$ | $2^{2^r-1}d^{2^r}$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $1$ | $2^{2^{n-2}}m^{2^{n-1}}$ | $2^{2^{n-1}-1}d^{2^{n-1}}$ |

Let $m_r$ be number of polynomials in $r$ variables, and $d_r$ their degree.

Decompose real line according to real roots of univariate polynomials: at most $m_1 d_1$ roots so at most $(2m_1 d_1 + 1)$ cells.

Over each one we decompose the cylinder in the plane into at most $(2m_2 d_2 + 1)$ cells.

Hence in $\mathbb{R}^n$ at most: $\prod_{i=1}^{n}(2m_i d_i + 1)$ cells.

Omit $+1$s for dominant term, with has the closed form on prior slide.

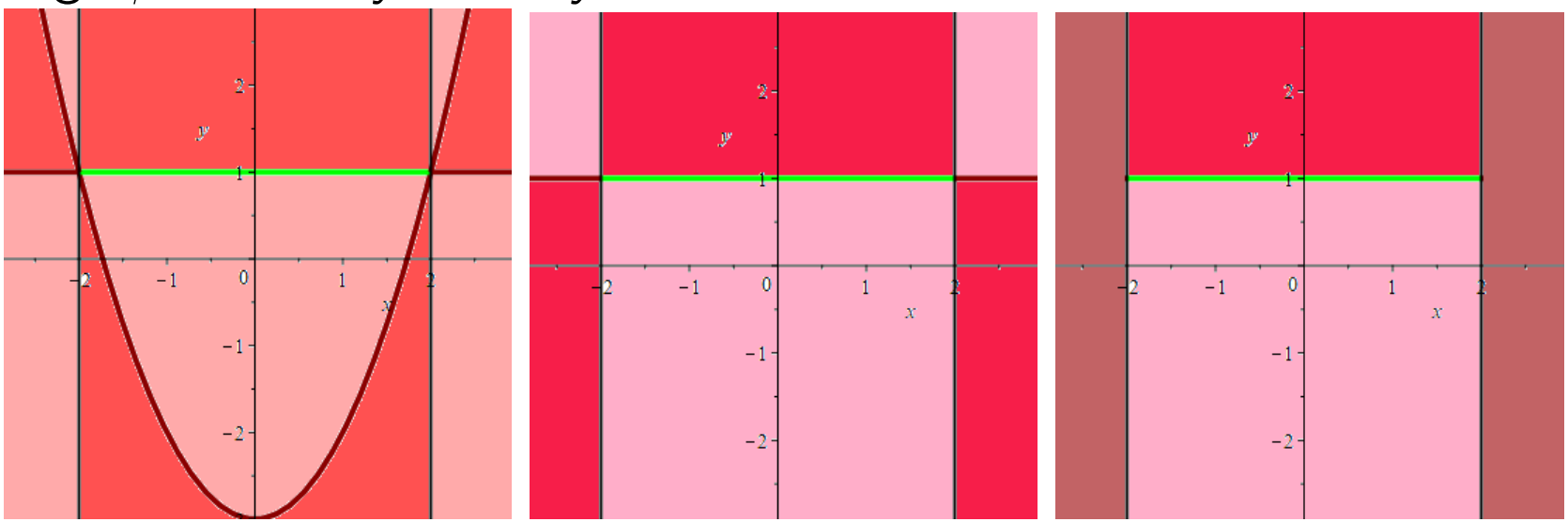
# Outline

# Truth Invariance

Sign-invariant CAD is more than we need for most applications. More appropriate is a CAD truth-invariant for formulae.

- Implied by sign-invariance of polynomials in formulae;
- But can often be achieved with far less cells.

E.g. $\phi := x^2 - y < 3 \wedge y = 1$. All three CADs are truth invariant:




Left is a traditional sign-invariant CAD, right is the coarsest truth invariant CAD. The middle can be obtained algorithmically.

# How to achieve truth invariant algorithmically?

There are various approaches to achieve truth-invariance:

- Refine sign-invariant CAD once produced. But then no savings in computation of CAD itself, only future work.
- Adapt projection according to formula structure. I.e. essentially ignore some of the constraints in parts of space where they are not relevant (produced middle image).
- Terminate lifting if truth can be determined early (potentially produce right image).

# Equational Constraints

An Equational Constraint (EC) is a polynomial equation logically implied by a Quantifier Free Tarski Formula (QFF).

An EC is explicit when it is an atom of the input formula and implicit otherwise.

E.g. $f = 0$ is an explicit EC of the formula $\phi = (f = 0) \wedge \varphi$.

E.g. $f_1 f_2 = 0$ is an implicit EC for

$$\psi = (f_1 = 0 \wedge \varphi_1) \vee (f_2 = 0 \wedge \varphi_2).$$

E.g. $f^2 + g^2 \leq 0$ has two implicit ECs: $f = 0$ and $g = 0$.

# Equational Constraints

An Equational Constraint (EC) is a polynomial equation logically implied by a Quantifier Free Tarski Formula (QFF).

An EC is explicit when it is an atom of the input formula and implicit otherwise.

E.g. $f = 0$ is an explicit EC of the formula $\phi = (f = 0) \wedge \varphi$.

E.g. $f_1 f_2 = 0$ is an implicit EC for

$$\psi = (f_1 = 0 \wedge \varphi_1) \vee (f_2 = 0 \wedge \varphi_2).$$

E.g. $f^2 + g^2 \leq 0$ has two implicit ECs: $f = 0$ and $g = 0$.

# Key observation on ECs

In 1998 Collins observed that truth invariance of a formula with equational constraint $f = 0$ is implied by:

- Sign-invariance for $f$; and
- Sign-invariance for all other polynomials $g_i$ when $f = 0$.

From this there is a natural adaption of the projection stage of the CAD algorithm.

For example, the projection usually takes resultants of every pair of polynomials (to capture their intersections) but in this case let us take only resultants which involve the EC polynomial.

When the EC is explicit this reduced projection is a subset of the original sign-invariant projection.

# Outline

# Outline

# Improved CAD with ECs

1998: Basic idea proposed by Collins.

1999: McCallum produces operator for first projection only.

2001: McCallum describes how to propagate and use ECs in subsequent projections.

2015: England-Bradford-Davenport:

- corresponding improvements to lifting stage;
- start to consider effect on complexity.

**ISSAC'15 Ex**
1,118,205 S.I. cells reduced to:

- 11,961, 30,233, 158,475 or 158,451 cells;
- 21,079 cells by default, (as low as 5633);
- 113, 103 or 93 cells;

depending on EC designation.

# Improved CAD with ECs

1998: Basic idea proposed by Collins.

1999: McCallum produces operator for first projection only.

2001: McCallum describes how to propagate and use ECs in subsequent projections.

2015: England-Bradford-Davenport:

- corresponding improvements to lifting stage;
- start to consider effect on complexity.

**ISSAC'15 Ex**
1,118,205 S.I. cells reduced to:

- 11,961, 30,233, 158,475 or 158,451 cells;
- 21,079 cells by default, (as low as 5633);
- 113, 103 or 93 cells;

depending on EC designation.

# Improved CAD with ECs

1998: Basic idea proposed by Collins.

1999: McCallum produces operator for first projection only.

2001: McCallum describes how to propagate and use ECs in subsequent projections.

2015: England-Bradford-Davenport:

- corresponding improvements to lifting stage;
- start to consider effect on complexity.

**ISSAC'15 Ex**
1,118,205 S.I. cells reduced to:

- 11,961, 30,233, 158,475 or 158,451 cells;
- 21,079 cells by default, (as low as 5633);
- 113, 103 or 93 cells;

depending on EC designation.

# EC Propagation

EC theory allows for a single declared EC for each projection.

If you have more than one EC with the same main variable you must make a choice: affects the coarseness of decomposition.

If you have two in the same main variable then you can still get twice the savings, via EC Propagation. Essentially, observe that if $f$ and $g$ are ECs for a formula then $\text{res}(f, g)$ is also an implicit EC with one fewer variable. It is also one guaranteed to be identified in the projection.

EC Propagation works as long as the ECs are independent, i.e. not linear combinations of each other.

Given three ECs there are three different propagations; given x ECs there are $\frac{1}{2}x(x-1)$. So EC designation becomes an important choice. Currently made based on crude heuristics.

# EC Propagation

EC theory allows for a single declared EC for each projection.

If you have more than one EC with the same main variable you must make a choice: affects the coarseness of decomposition.

If you have two in the same main variable then you can still get twice the savings, via EC Propagation. Essentially, observe that if $f$ and $g$ are ECs for a formula then $\text{res}(f, g)$ is also an implicit EC with one fewer variable. It is also one guaranteed to be identified in the projection.

EC Propagation works as long as the ECs are independent, i.e. not linear combinations of each other.

Given three ECs there are three different propagations; given x ECs there are $\frac{1}{2}x(x-1)$. So EC designation becomes an important choice. Currently made based on crude heuristics.

# EC Propagation

EC theory allows for a single declared EC for each projection.

If you have more than one EC with the same main variable you must make a choice: affects the coarseness of decomposition.

If you have two in the same main variable then you can still get twice the savings, via EC Propagation. Essentially, observe that if $f$ and $g$ are ECs for a formula then $\text{res}(f, g)$ is also an implicit EC with one fewer variable. It is also one guaranteed to be identified in the projection.

EC Propagation works as long as the ECs are independent, i.e. not linear combinations of each other.

Given three ECs there are three different propagations; given x ECs there are $\frac{1}{2}x(x-1)$. So EC designation becomes an important choice. Currently made based on crude heuristics.

# Side Note: Moving away from traditional CAD

One of our contributions was to describe how EC projection allows for savings when lifting also. This was theoretically trivial but required discarding long embedded CAD principles:

1. That the projection polynomials are a fixed set.
2. That the invariance structure of the final CAD can be expressed as sign-invariance of certain polynomials.

The second is just a mental leap but the first required substantial engineering changes to implementations. Instead of a set of polynomials we need to keep a database on them so we only use the appropriate ones at appropriate points.
Similar infrastructure needed for incremental CAD.

# Complexity

Suppose we have $\ell$ primitive ECs with successive main variables (starting from the main variable of the system).

M. England, R. Bradford and J.H. Davenport.
*Improving the use of equational constraints in cylindrical algebraic decomposition*
In Proc. ISSAC '15, pages 165–172. ACM, 2015.

The dominant term in the bound on the number of CAD cells is:

$$(2d)^{2^n-1} m^{2^{n-\ell}-2} 2^{\ell 2^{n-\ell}-3\ell}$$

The double exponent on the number of polynomials, $m$, reduces for each EC identified. **But** the double exponent of the degree, $d$, does not drop similarly.

# Sketch of Proof

We assume ECs for the first $\ell$ projections. Less projection polynomials, so Number column changes.

| Variables | Number | Degree |
|---|---|---|
| $n$ | $m$ | $d$ |
| $n-1$ | $2m$ | $2d^2$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $n-\ell$ | $2^\ell m$ | $2^{2^\ell-1}d^{2^\ell}$ |
| $n-(\ell+1)$ | $2^{2\ell}m^2$ | $2^{2^{\ell+1}-1}d^{2^{\ell+1}}$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $n-(\ell+r)$ | $2^{2^r\ell}m^{2^r}$ | $2^{2^{\ell+r}-1}d^{2^{\ell+r}}$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $1$ | $2^{2^{(n-1-\ell)}\ell}m^{2^{n-1-\ell}}$ | $2^{2^{n-1}-1}d^{2^{n-1}}$ |

But the degree of them is unaffected.

# Outline

# Why is the degree so high?

Projection and EC propagation uses repeated iterated resultants. Having less of them doesn't change their degree.

Iterated univariate resultants produced like this may be more complicated that the information they need to encode. For example, if we care only of the intersection between multiple polynomials then the true multivariate resultant has lower degree.

L. Busé and B. Mourrain.
*Explicit factors of some iterated resultants and discriminants.*
Mathematics of Computation, 78:345–386, 2009.

Busé and Mourrain explain how it is contained as a factor of iterated univariate resultants.

# Can we get away with using only multivariate resultants?

**No**. For normal CAD projection the pairwise resultants of all polynomials are a necessity for proving the delineability conditions which allow for working at a sample point.

**BUT** If we have multiple ECs then we only care about what happens when all are satisfied and thus the multivariate resultant.

**AND** the reduced lifting of the ISSAC 2015 paper showed that in this case it is only the degrees of the ECs that contribute to the complexity (from polynomials of that main variable).

So together, this allows us to bound the degree complexity with results for multivariate resultants applied to EC projections.

# Can we get away with using only multivariate resultants?

**No**. For normal CAD projection the pairwise resultants of all polynomials are a necessity for proving the delineability conditions which allow for working at a sample point.

**BUT** If we have multiple ECs then we only care about what happens when all are satisfied and thus the multivariate resultant.

**AND** the reduced lifting of the ISSAC 2015 paper showed that in this case it is only the degrees of the ECs that contribute to the complexity (from polynomials of that main variable).

So together, this allows us to bound the degree complexity with results for multivariate resultants applied to EC projections.

# So new improved bound

Our CASC 2016 paper reformulated the Busé-Mourrain results so instead of concerning total degree in all variables, we have the case of degree in at most one variable (relevant for CAD analysis).

This allowed us to present an improve complexity bound for CAD with $\ell$ primitive ECs in different main variables. This time the exponent of $d$ was

$$2^{(n-\ell)}\tfrac{1}{2}(\ell^2+\ell+2)-\tfrac{1}{2}(\ell^2+\ell)-2.$$

So **CAD with ECs should be doubly exponential in $n-\ell$** (all double exponents).

# But how to implement?

The analysis suggested that a CAD which tracked only the multivariate resultants we care about at EC levels offered the better bound. But how to implement and compose with regular CAD?

Our CASC 2016 paper suggested replacing the original ECs (and their propagation) by their Gröbner Basis and using the lowest degree polynomials at each level as declared ECs. The GB is known to encode the multivariate resultant. We can then proceed with the regular CAD implementation.

But GB is itself doubly exponential? Yes, but in practice GB implementations can solve problems many orders of magnitude greater than CAD. Any problem on which CAD is tractable, GB for ECs is trivial. GB is already a common pre-processing step for most CAD implementations.

# But how to implement?

The analysis suggested that a CAD which tracked only the multivariate resultants we care about at EC levels offered the better bound. But how to implement and compose with regular CAD?

Our CASC 2016 paper suggested replacing the original ECs (and their propagation) by their Gröbner Basis and using the lowest degree polynomials at each level as declared ECs. The GB is known to encode the multivariate resultant. We can then proceed with the regular CAD implementation.

But GB is itself doubly exponential? Yes, but in practice GB implementations can solve problems many orders of magnitude greater than CAD. Any problem on which CAD is tractable, GB for ECs is trivial. GB is already a common pre-processing step for most CAD implementations.

# But how to implement?

The analysis suggested that a CAD which tracked only the multivariate resultants we care about at EC levels offered the better bound. But how to implement and compose with regular CAD?

Our CASC 2016 paper suggested replacing the original ECs (and their propagation) by their Gröbner Basis and using the lowest degree polynomials at each level as declared ECs. The GB is known to encode the multivariate resultant. We can then proceed with the regular CAD implementation.

But GB is itself doubly exponential? Yes, but in practice GB implementations can solve problems many orders of magnitude greater than CAD. Any problem on which CAD is tractable, GB for ECs is trivial. GB is already a common pre-processing step for most CAD implementations.

# Outline

# Outline

# Primitivity Restrictions

There are restrictions of the preceding discussion:

- That the ECs be in successive main variable, starting from that of the system.

This restriction was for ease of the analysis. We do not expect it to change the main conclusion.

- That the ECs be primitive.

This restriction was a necessity of the theory however. The best way to remove it is not clear. Further, the final section of our JSC journal paper shows that the key family of examples in the classic papers [Davenport and Heintz, 1988] and [Brown and Davenport, 2007] proving CAD complexity bounds contain only non-primitive ECs.

# Primitivity Restrictions

There are restrictions of the preceding discussion:

- That the ECs be in successive main variable, starting from that of the system.

This restriction was for ease of the analysis. We do not expect it to change the main conclusion.

- That the ECs be primitive.

This restriction was a necessity of the theory however. The best way to remove it is not clear. Further, the final section of our JSC journal paper shows that the key family of examples in the classic papers [Davenport and Heintz, 1988] and [Brown and Davenport, 2007] proving CAD complexity bounds contain only non-primitive ECs.

# Well-orientedness restriction

Everything we did was formulated using McCallum's family of CAD Projection operators. All of these carry with them a well-orientedness conditions which is easy to check and usually always satisfied.

The new Lazard projection operator for sign-invariance has no such condition (at the expense of a little extra work during lifting on cases that would have previously failed). The extension of this operator to the various EC variants is key DEWCAD aim.

All complexity results are expected to remain, but it removes the case of non well-orientedness.

# Preconditioning CAD with GB literature


B. Buchberger and H. Hong.
*Speeding up quantifier elimination by Gröbner bases*
Tech. Report, 91-06. RISC, Johannes Kepler Uni, 1991


D.J. Wilson, R.J. Bradford, and J.H. Davenport.
Speeding up cylindrical algebraic decomposition by Gröbner bases
In Intelligent Computer Mathematics (LNCS 7362), pages 280-294. Springer, 2012.


Z. Huang, M. England, J.H. Davenport and L.C. Paulson
*Using Machine Learning to decide when to Precondition Cylindrical Algebraic Decomposition with Gröbner Bases.*
To appear in Proc. SYNASC 2016 (IEEE).

Each of these found that GB pre-conditioning for CAD was usually, but not universally beneficial.

# Why not universally beneficial?

**Pessimistic Answer:**

- This paper was concerned with theoretical worst case complexity. Bounding that does not guarantee good performance every time.

**Optimistic Answer:**

- The CAD technology used in those papers did not take full advantage of multiple ECs and lifting savings as outlined here. Retesting should show improved results.

# Publications

Talk is based on work in the following papers:

M. England, R. Bradford and J.H. Davenport.

*Improving the use of equational constraints in cylindrical algebraic decompositions.* Proc. ISSAC '15, pp. 165–172. ACM, 2015. DOI: 10.1145/2755996.2756678

M. England and J.H. Davenport.

*The complexity of cylindrical algebraic decomposition with respect to polynomial degree.* Proc. CASC '16, LNCS 9890, pp. 172-192. Springer, 2016. DOI: 10.1007/978-3-319-45641-6_12

M. England, R. Bradford and J.H. Davenport.

Cylindrical Algebraic Decomposition with Equational Constraints. Journal of Symbolic Computation, vol. 100, pp. 38-71 Elsevier, 2020. DOI: 10.1016/j.jsc.2019.07.019