Using the Regular Chains Library to Build Cylindrical Algebraic Decompositions by Projecting and Lifting

Matthew England

Joint work with: R. Bradford, J.H. Davenport & D. Wilson
The University of Bath

The 4th International Congress on Mathematical Software
Hanyang University, Seoul, Korea
August 5-9 2014

Supported by EPSRC Grant EP/J003247/1.

1. Background material
    - Cylindrical Algebraic Decomposition
    - How to build a CAD

2. The PROJECTIONCAD package
    - Motivation and implementation
    - Functionality

Outline

1. Background material
   - Cylindrical Algebraic Decomposition
   - How to build a CAD
2. The ProjectionCAD package
   - Motivation and implementation
   - Functionality

What is a CAD?

A Cylindrical Algebraic Decomposition (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 are either equal or disjoint.

Example - Cylindrical Algebraic Decomposition

A CAD of $\mathbb{R}^2$ is given by the following collections of 13 cells:

$$\begin{aligned}
&[x < -1, y = y],\\
&[x = -1, y < 0], [x = -1, y = 0], [x = -1, y > 0],\\
&[-1 < x < 1, y^2 + x^2 - 1 > 0, y > 0],\\
&[-1 < x < 1, y^2 + x^2 - 1 = 0, y > 0],\\
&[-1 < x < 1, y^2 + x^2 - 1 < 0],\\
&[-1 < x < 1, y^2 + x^2 - 1 = 0, y < 0],\\
&[-1 < x < 1, y^2 + x^2 - 1 < 0, y < 0],\\
&[x = 1, y < 0], [x = 1, y = 0], [x = 1, y > 0],\\
&[x > 1, y = y]
\end{aligned}$$

Example - Cylindrical Algebraic Decomposition

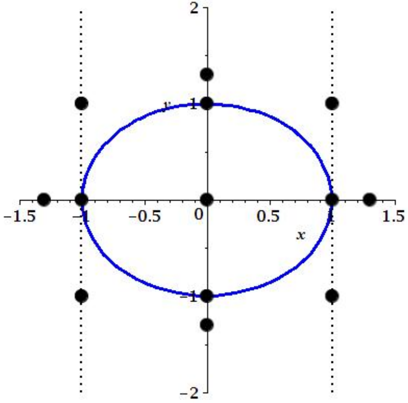
A CAD of $\mathbb{R}^2$ is given by the following collections of 13 cells:

$$
\begin{array}{rl}
x < -1 \{ & [x < -1, y = y], \\
x = -1 \{ & [x = -1, y < 0], [x = -1, y = 0], [x = -1, y > 0], \\
\{ & [-1 < x < 1, y^2 + x^2 - 1 > 0, y > 0], \\
\{ & [-1 < x < 1, y^2 + x^2 - 1 = 0, y > 0], \\
-1 < x < 1 \{ & [-1 < x < 1, y^2 + x^2 - 1 < 0], \\
\{ & [-1 < x < 1, y^2 + x^2 - 1 = 0, y < 0], \\
\{ & [-1 < x < 1, y^2 + x^2 - 1 < 0, y < 0], \\
x = 1 \{ & [x = 1, y < 0], [x = 1, y = 0], [x = 1, y > 0], \\
x > 1 \{ & [x > 1, y = y]
\end{array}
$$

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.

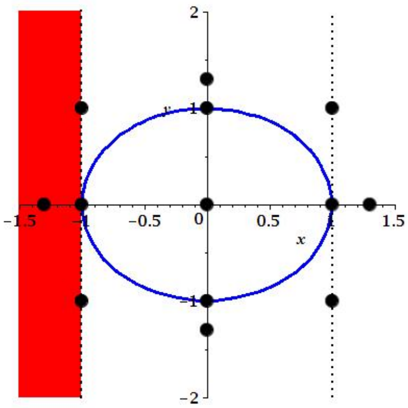
The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$.



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.

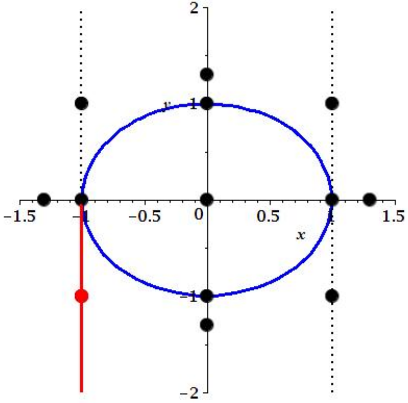
The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$.



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.

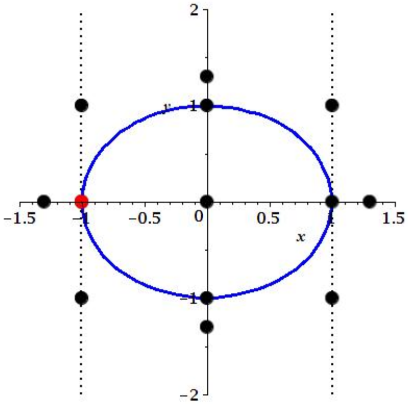
The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$.



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.

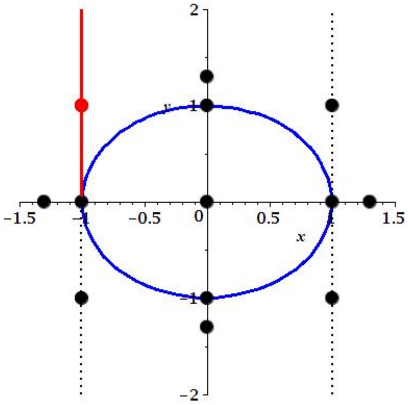
The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$.



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.

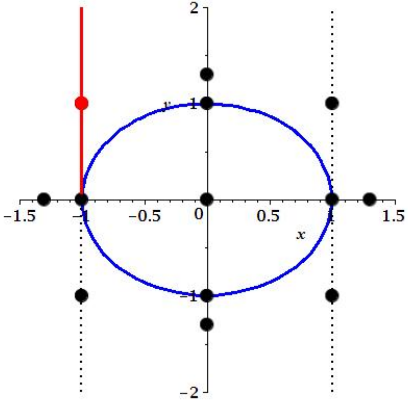
The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$.



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.

The example from the previous slide was a sign-invariant CAD for the polynomial $x^2 + y^2 - 1$.

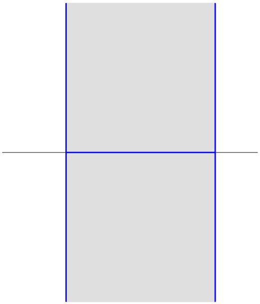


Sign-invariance means we need only test one sample point per cell to determine behaviour of the polynomials. Various applications: quantifier elimination, optimisation, theorem proving, ...

CAD Terminology

The cylindricity property means that all cells in a CAD of $\mathbb{R}^d$ lie in the cylinder above a cell, $c \in \mathbb{R}^{d-1}$.

I.e. in $c \times \mathbb{R}$.



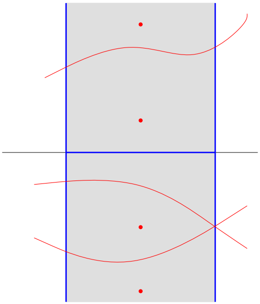
CAD Terminology

The cylindricity property means that all cells in a CAD of $\mathbb{R}^d$ lie in the cylinder above a cell, $c \in \mathbb{R}^{d-1}$.

I.e. in $c \times \mathbb{R}$.
We call the decomposition of the cylinder a stack. It consists of:

- sections of polynomials (cells where a polynomial vanishes);
- sectors cells in-between (or above / below) sections.

E.g. This stack has 3 sections and 4 sectors.

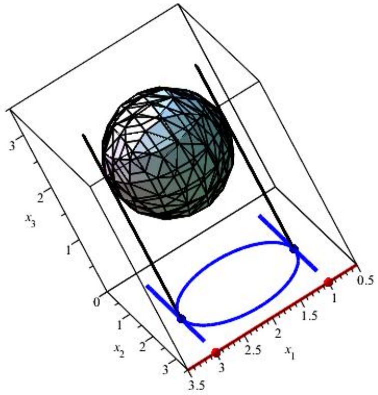


Outline

CAD via Projection and lifting

Traditionally CAD algorithms work by a process of:

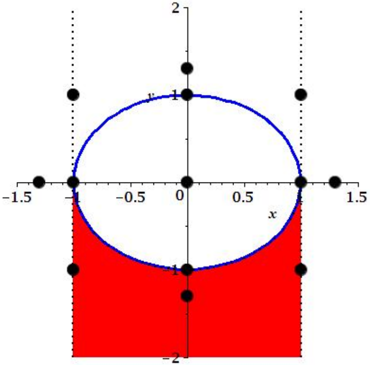
- Projection: to derive a set of polynomials from the input which can define the decomposition



CAD via Projection and lifting

Traditionally CAD algorithms work by a process of:

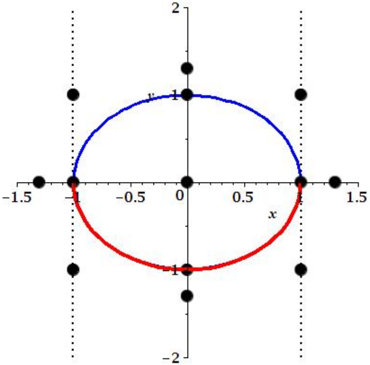
- Projection: to derive a set of polynomials from the input which can define the decomposition
- Lifting to incrementally build CADs by dimension.



CAD via Projection and lifting

Traditionally CAD algorithms work by a process of:

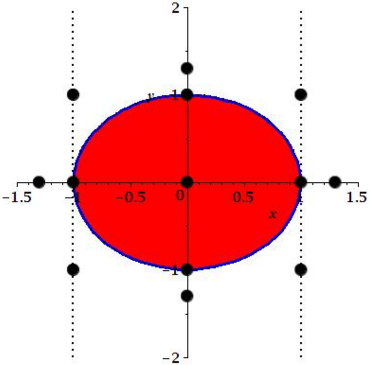
- Projection: to derive a set of polynomials from the input which can define the decomposition
- Lifting to incrementally build CADs by dimension.



CAD via Projection and lifting

Traditionally CAD algorithms work by a process of:

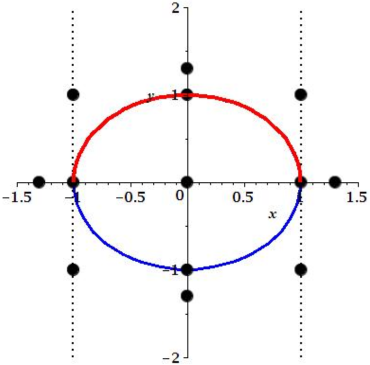
- Projection: to derive a set of polynomials from the input which can define the decomposition
- Lifting to incrementally build CADs by dimension.



CAD via Projection and lifting

Traditionally CAD algorithms work by a process of:

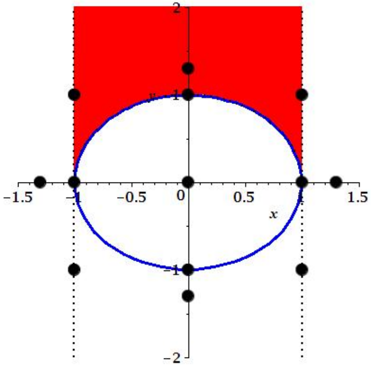
- Projection: to derive a set of polynomials from the input which can define the decomposition
- Lifting to incrementally build CADs by dimension.



CAD via Projection and lifting

Traditionally CAD algorithms work by a process of:

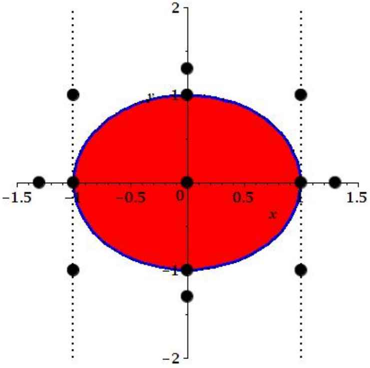
- Projection: to derive a set of polynomials from the input which can define the decomposition
- Lifting to incrementally build CADs by dimension.

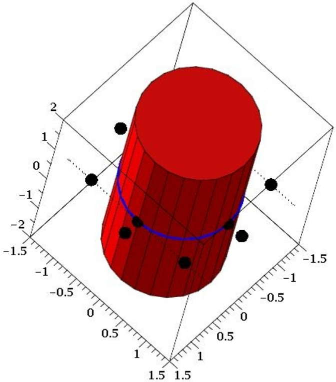


CAD via Projection and lifting

Traditionally CAD algorithms work by a process of:

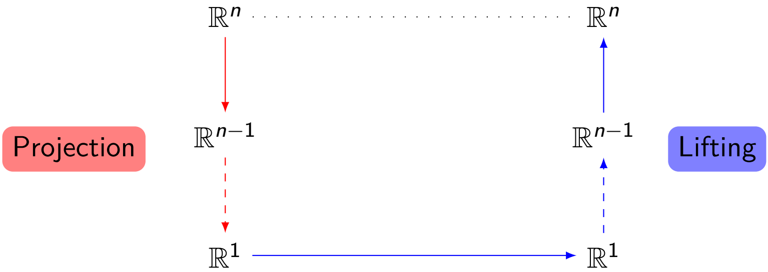
- Projection: to derive a set of polynomials from the input which can define the decomposition
- Lifting to incrementally build CADs by dimension.





CAD via Triangular Decomposition

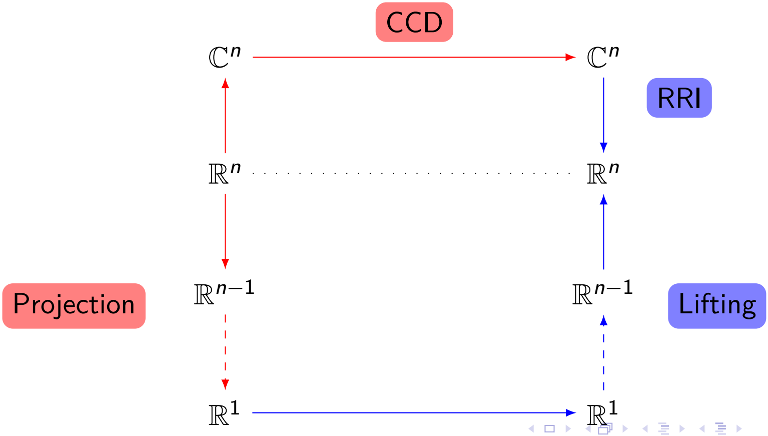
< 2009 All CAD research broadly within Collin's projection and lifting framework.



CAD via Triangular Decomposition

$<$ 2009: All CAD research broadly within Collin's projection and lifting framework.

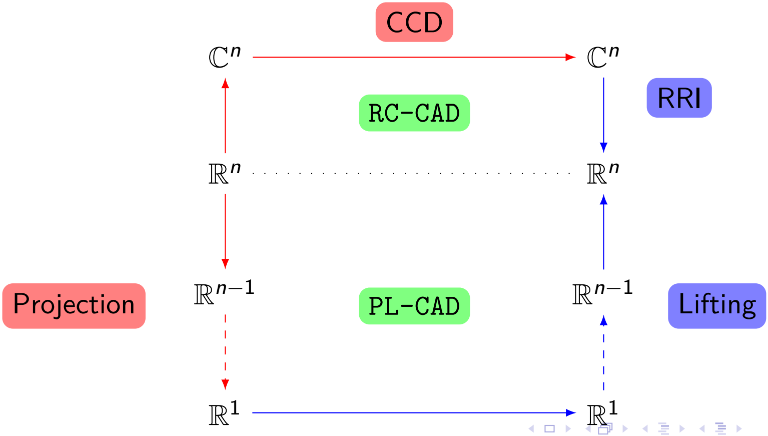
ISSAC 2009: Chen, Moreno Maza, Xia & Yang give new approach.



CAD via Triangular Decomposition

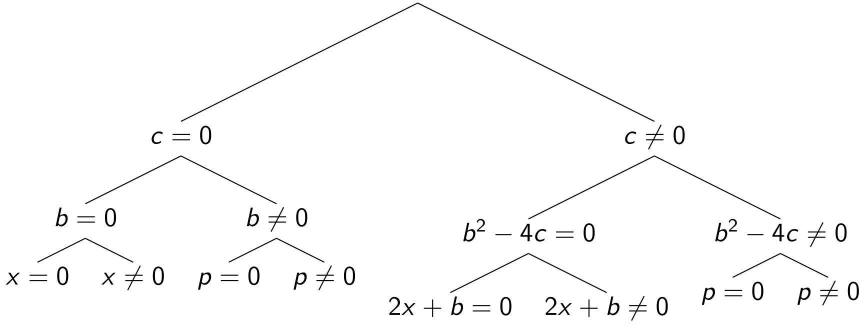
$<$ 2009: All CAD research broadly within Collin's projection and lifting framework.

ISSAC 2009: Chen, Moreno Maza, Xia & Yang give new approach.



Cylindrical complex decompositions

RC-CAD starts with a complex cylindrical decomposition (CCD).
The tree below represents a sign-invariant CCD for $p := x^2 + bx + c$ under variable ordering $c \prec b \prec x$.



Cylindrical complex decompositions

`RC-CAD` starts with a complex cylindrical decomposition (CCD).
The tree below represents a sign-invariant CCD for $p := x^2 + bx + c$ under variable ordering $c \prec b \prec x$.

- $c = 0$
  - $b = 0$
    - $x = 0$
    - $x \neq 0$
  - $b \neq 0$
    - $p = 0$
    - $p \neq 0$
- $c \neq 0$
  - $b^2 - 4c = 0$
    - $2x + b = 0$
    - $2x + b \neq 0$
  - $b^2 - 4c \neq 0$
    - $p = 0$
    - $p \neq 0$

The key advantage is case distinction: the polynomial b is not sign-invariant for the whole decomposition, only when required.

Outline

1. Background material
   - Cylindrical Algebraic Decomposition
   - How to build a CAD
2. The ProjectionCAD package
   - Motivation and implementation
   - Functionality

The ProjectionCAD package

A Maple package developed at the University of Bath which builds CADs via projection and lifting.

- Currently freely available from the authors;
- Plans to integrate it into the RegularChains Library at `http://www.regularchains.org/`

Originally developed to compare the theory of `PL-CAD` and `RC-CAD` in an *implementation independent* context.
Later used to implement new theory for `PL-CAD` (summarised later) which has in turn led to new theory for RC-CAD (see the talks of Davenport and Moreno Maza).

Motivation

ProjectionCAD uses the `RegularChains` Library to create stacks of cells in the lifting phase. The main motivation:

- Uses efficient algorithms for triangular decomposition to compute with algebraic numbers.
- Ensures ProjectionCAD always uses the best available sub-algorithms (such as new code for real root isolation).
- ProjectionCAD can match output formats with the RC-CAD implementations. Allows for easy comparison and use of the intuitive *tree-like* `piecewise` structure.

Implementation I

In the PL-CAD framework cells are constructed to decompose a cylinder into a stack according to the signs of given projection polynomials. We use the command in the RegularChains Library for this. It was originally developed as a sub-algorithm to `MakeSemiAlgebraic`: the tool for converting a CCD into a CAD.

Difficulties: The RegularChains command assumed that in addition to delineability, the polynomials separate above the cell, meaning they are coprime and square-free throughout.

Implementation II

To overcome the difficulties polynomials were pre-processed before the RegularChains algorithm was called:

- To ensure they were coprime a variant of the `Triangularize` algorithm was repeatedly called to find the zeros of a polynomial also zeros of a regular chain but not zeros of another set (polynomials already processed).
- To ensure they were squarefree an analogue of Musser's algorithm for square-free factorization adapted for regular chains was repeatedly used.

Outline

1. Background material
   - Cylindrical Algebraic Decomposition
   - How to build a CAD
2. The ProjectionCAD package
   - Motivation and implementation
   - Functionality

Basic functionality

ProjectionCAD can build sign-invariant CADs:

- using either the Collins or McCallum projection operators;
- in a variety of output formats (including one's useful for future computation and others designed for human readability);

Basic functionality

ProjectionCAD can build sign-invariant CADs:

- using either the Collins or McCallum projection operators;
- in a variety of output formats (including one's useful for future computation and others designed for human readability);
- with the stronger property of order-invariance if requested (so each polynomial vanishes to constant order in each cell);
- with minimal delineating polynomials built automatically (avoiding unnecessary failure declarations).

Equational constraints

Equational Constraint: an equation logically implied by a formula.

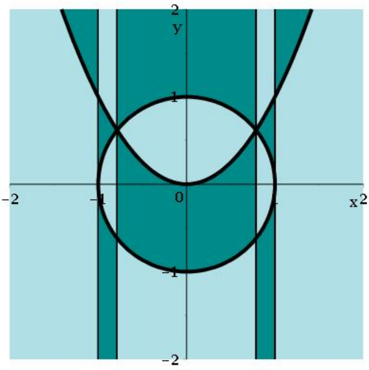
Given a formula we seek a CAD so that the Boolean value is constant on each cell. A CAD sign-invariant for the polynomials involved would achieve this. However, McCallum defined a projection operator which leads to a CAD on which:

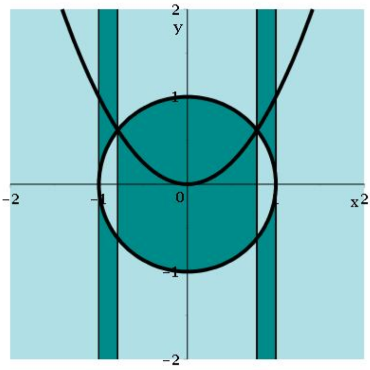
- the polynomial defining the EC is sign-invariant;
- the polynomials defining other constraints are sign-invariant if the EC is satisfied.

ProjectionCAD has an implementation of this more efficient (less cells, less computation time) projection. It also uses additional optimisation in the lifting stage to give further efficiencies.

Equational constraints example

Consider the formulae $\phi := (x^2 + y^2 - 1 = 0) \wedge (x^2 - y > 0)$.





Truth-table invariance

Given a sequence of formulae a truth-table invariant CAD (TTICAD) is a CAD such that each formula has constant Boolean value on each cell.

- Together with McCallum the Bath team validated a new projection operator to build TTICADs which makes savings from equational constraints.
- Only one formula need have an EC to generate savings over a sign-invariant CAD for the polynomials involved.
- Implemented in ProjectionCAD (including savings in the lifting stage).

Why build a TTICAD?

A TTICAD can be useful for:

- An application providing a sequence of separate formulae

For example, algebraic simplification of identities involving multi-valued functions requires checking validity on regions of complex space divided by the branch cuts of the functions involved: a TTICAD for the formulae describing the cuts is exactly the desired object.

Why build a TTICAD?

A TTICAD can be useful for:

- An application providing a sequence of separate formulae

For example, algebraic simplification of identities involving multi-valued functions requires checking validity on regions of complex space divided by the branch cuts of the functions involved: a TTICAD for the formulae describing the cuts is exactly the desired object.

- Finding a truth-invariant CAD for a parent formula

A TTICAD for the defining sub-formula is truth-invariant for the parent. TTICAD can be the most efficient known approach (especially if there is no EC for the parent formula).

TTICAD Example I

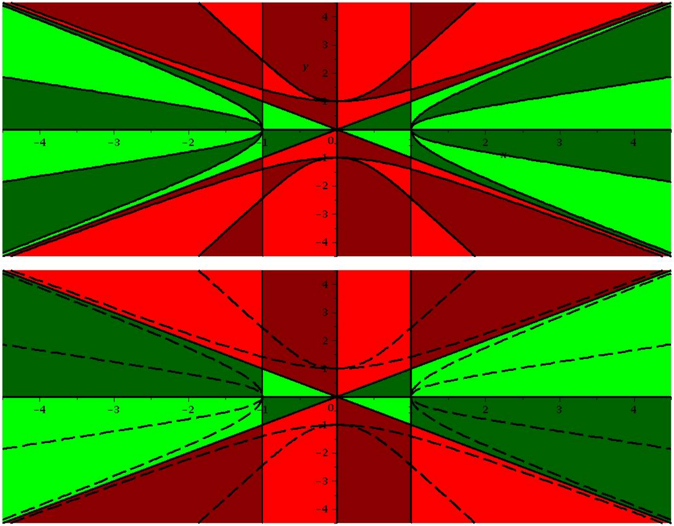
Consider $\sqrt{z^2-1}\sqrt{z^2+1}=\sqrt{z^4-1}$. Most software takes $\sqrt{\ }$ to be the positive root, in which case the identity is not always true.

For $z = x + \mathrm{i}y$, the functions involved have branch cuts:

$$\begin{aligned}
\varphi_1 &:= 2xy = 0 \wedge x^2 - y^2 < 1, \\
\varphi_2 &:= 2xy = 0 \wedge x^2 - y^2 < -1, \\
\varphi_3 &:= 4x^3y - 4xy^3 = 0 \wedge x^4 - 6x^2y^2 + y^4 < 1.
\end{aligned}$$

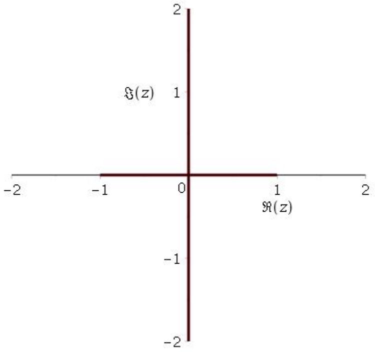
Either a TTICAD for $\{\varphi_1, \varphi_2, \varphi_3\}$ or a sign-invariant CAD for the polynomials involved would decompose $\mathbb{C} = \mathbb{R}^2$ according to these cuts. We then need to test the truth at a finite number of sample points.

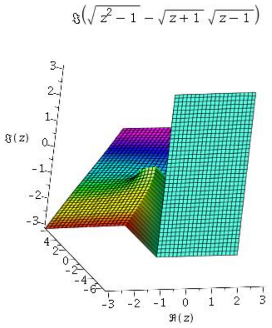
TTICAD Example II



Sidenote: Branch Cuts in ICMS Demo Session

We have another package, BranchCuts for calculating and visualising the branch cuts of multi-valued functions in Maple. It was integrated into the `FunctionAdvisor` in Maple 17.





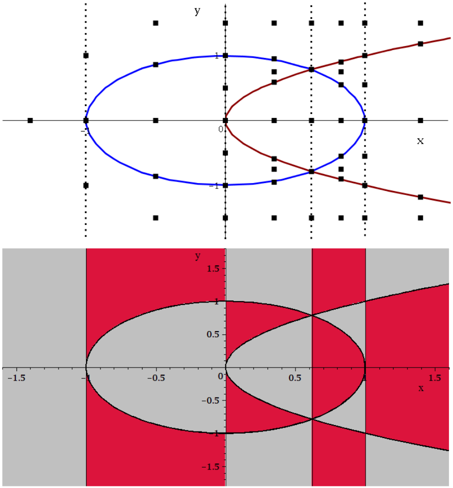
Layered Sub-CADs

A layered sub-CAD is a subset of cells from a CAD consisting of those with a prescribed dimension and higher.

- Can be produced more efficiently than a full CAD.
- Implemented in ProjectionCAD by truncating the lifting process appropriately.
- Useful if problems known to have solutions of a given dimensions, or if solutions are only needed to a specific accuracy.
- Can build layered CADs directly or incrementally (one layer at a time starting with cells of full-dimension and working down).

Layered Sub-CAD Example

Consider a sign invariant CAD for $\{x^2 + y^2 - 1, x - y^2\}$.



Variety CADs

A variety sub-CAD is a subset of cells from a CAD consisting of those which lie on a prescribed variety.

- Can be produced more efficiently than a full CAD.
- Implemented in ProjectionCAD when the variety is an equational constraint for the input.
- Much smaller output than a full-CAD, with time savings also possible depending on the dimension of the variety.
- Useful if all that is required is a description of solutions to formulae.

Other functionality

Other functionality in ProjectionCAD:

- Combinations of layered and variety sub-CADs with each other and different projection operators (for example, layered variety truth-table invariant sub-CADs are available).
- User commands for stack generation and induced CADs (the CADs of lower dimensional space produced as part of a computation) allowing for easy experimentation.
- Heuristics to help with choices such as variable ordering, EC designation, and how best to prepare formulae for TTICAD.

The End

References to the theory implemented in ProjectionCAD are summarised in the ICMS paper, along with the technical details of how the RegularChains routines are used.

Contact Details

M.England@bath.ac.uk
http://www.cs.bath.ac.uk/~me350/

Thanks for listening!