Cylindrical Algebraic Decompositions for Boolean Combinations

Matthew England
The University of Bath

Joint work with: Russell Bradford, James Davenport, Scott McCallum and David Wilson

Western University Canada
17th June 2013

Supported by EPSRC Grant EP/J003247/1.
Outline

1 Introduction
   - Cylindrical Algebraic Decomposition
   - Why build a CAD?
   - 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
   - Why build a CAD?
   - CAD for Boolean Combinations

2. Developing TTICAD
   - Motivation
   - New Projection Operator
   - Important Technicalities

3. TTICAD in Practice
   - Implementation in Maple
   - Experimental Results
   - Conclusions
A Cylindrical Algebraic Decomposition (CAD) is a mathematical object. Defined in the 1970s by Collins who also gave the first algorithm to compute one. A CAD is:

- a decomposition meaning a partition of $\mathbb{R}^n$ into connected subsets called cells;
- 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.
A Cylindrical Algebraic Decomposition (CAD) is a mathematical object. Defined in the 1970s by Collins who also gave the first algorithm to compute one. A CAD is:

- a decomposition meaning a partition of $\mathbb{R}^n$ into connected subsets called cells;
- 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.
A Cylindrical Algebraic Decomposition (CAD) is a mathematical object. Defined in the 1970s by Collins who also gave the first algorithm to compute one. A CAD is:

- a decomposition meaning a partition of $\mathbb{R}^n$ into connected subsets called cells;
- 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 I

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

$$
x < -1 \{ [x < -1, y = y], \\
x = -1 \{ [x = -1, y < 0], [x = -1, y = 0], [x = -1, y > 0], \\
\{ [-1 < x < 1, y^2 + x^2 - 1 > 0, y > 0], \\
\{ [-1 < x < 1, y^2 + x^2 - 1 = 0, y > 0], \\
-1 < x < 1 \{ [-1 < x < 1, y^2 + x^2 - 1 < 0], \\
\{ [-1 < x < 1, y^2 + x^2 - 1 = 0, y < 0], \\
\{ [-1 < x < 1, y^2 + x^2 - 1 < 0, y < 0], \\
x = 1 \{ [x = 1, y < 0], [x = 1, y = 0], [x = 1, y > 0], \\
x > 1 \{ [x > 1, y = y]
$$
A CAD of $\mathbb{R}^2$ is given by the following collections of 13 cells:

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

- $-1 < x < 1 \{ [-1 < x < 1, y^2 + x^2 - 1 < 0] \}$
  - $\{ [-1 < x < 1, y^2 + x^2 - 1 = 0, y < 0] \}$
  - $\{ [-1 < x < 1, y^2 + x^2 - 1 < 0, y < 0] \}$

- $x = 1 \{ [x = 1, y < 0], [x = 1, y = 0], [x = 1, y > 0] \}$

- $x > 1 \{ [x > 1, y = y] \}$
Example - Cylindrical Algebraic Decomposition II

The cylindricity allows for the CAD to be displayed as a tree:

\[
\begin{align*}
&c_1 \\
&\quad c_2 \quad y < 0 \\
&\quad c_3 \quad y = 0 \\
&\quad c_4 \quad 0 < y \\
&\quad c_5 \quad y < -\sqrt{-x^2 + 1} \\
&\quad c_6 \quad y = -\sqrt{-x^2 + 1} \\
&\quad c_7 \quad -\sqrt{-x^2 + 1} < y < \sqrt{-x^2 + 1} \quad \text{And } (-1 < x, x < 1) \\
&\quad c_8 \quad y = +\sqrt{-x^2 + 1} \\
&\quad c_9 \quad \sqrt{-x^2 + 1} < y \\
&\quad c_{10} \quad y < 0 \\
&\quad c_{11} \quad y = 0 \\
&\quad c_{12} \quad 0 < y \\
&\quad c_{13} \quad x < -1 \\
\end{align*}
\]
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 CAD from the previous slide was sign-invariant for the polynomial $x^2 + y^2 - 1$. 
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 CAD from the previous slide was sign-invariant for the polynomial \( x^2 + y^2 - 1 \).
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 CAD from the previous slide was sign-invariant for the polynomial $x^2 + y^2 - 1$. 
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 CAD from the previous slide was sign-invariant for the polynomial $x^2 + y^2 - 1$. 
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 CAD from the previous slide was sign-invariant for the polynomial $x^2 + y^2 - 1$. 
How to build a CAD

The main idea behind Collin’s algorithm is to build a CAD recursively, i.e. first build a CAD of $\mathbb{R}^1$, then a CAD of $\mathbb{R}^2$ and continue until a CAD of $\mathbb{R}^n$ is produced. The algorithm has two main phases: Projection and Lifting.

The projection phase identifies polynomials in the lower dimensional spaces necessary to build the CADs. The lifting phase then uses these to construct the cells.
A *projection operator* $P$ is applied repeatedly to the polynomials, each time producing a new set of polynomials in one less variable.

The variables are ordered: $x_1 > x_2 \cdots > x_n$. The *main variable* of a polynomial is the highest ordered variable occurring.

The projection operator is defined to output polynomials which describe important changes in behaviour in the input polynomials. For example, it includes the *discriminant* of polynomials (describing the behaviour of their roots) and the *resultant* of pairs of polynomials (describing their intersections).

Collins original operator can give far more output than required. There has been much subsequent research on its refinement.
The projection operator applied to the sphere identifies the circle. The projection operator applied to the circle identifies two points on the real line.
The CAD of $\mathbb{R}^n$ is constructed recursively as follows:

**Base case:** A CAD of $\mathbb{R}$ is produced using the roots of the univariate polynomials and the intervals in-between.

**Generic case:** Suppose we have a CAD of $\mathbb{R}^k$. To construct a CAD of $\mathbb{R}^{k+1}$ do the following for each cell in $\mathbb{R}^k$:

1. Consider the cylinder over the cell.

For example, suppose we have a CAD of $\mathbb{R}$ and a particular cell $D$ which is an interval of $\mathbb{R}$.

Then the **cylinder** over $D$ is $D \times \mathbb{R}$.
How to build a CAD - Lifting II

2. Identify the projection polynomials with main variable $x_{k+1}$ and evaluate each at a sample point of the cell.

3. Find the roots of these univariate polynomials.

4. Construct a stack over the cell: a collection of cells of $\mathbb{R}^{k+1}$. These consist of sections (where a polynomial has a root) and sectors (the intervals in-between).

Together the stacks are a CAD of $\mathbb{R}^{k+1}$.

E.g. This stack has 3 sections and 4 sectors.
Consider the circle defined by the graph of $f = y^2 + x^2 - 1$.

- $P(f)$ gives the univariate polynomials $\{x - 1, x + 1\}$.
- The roots are $\pm 1$ so a CAD of $\mathbb{R}^1$ is produced with 5 cells.

Consider the cell where $-1 < x < 1$. We can take the sample point $x = 0$ leading to $f_{x=0} = y^2 - 1$. This has two roots and thus the stack over the cell consists of two sections and three sectors.

Combining stacks for the cells in $\mathbb{R}^1$ gives the CAD of $\mathbb{R}^2$ with 13 cells.
How to build a CAD - Example

Consider the circle defined by the graph of \( f = y^2 + x^2 - 1 \).

- \( P(f) \) gives the univariate polynomials \( \{x - 1, x + 1\} \).
- The roots are \( \pm 1 \) so a CAD of \( \mathbb{R}^1 \) is produced with 5 cells.

- Consider the cell where \(-1 < x < 1\). We can take the sample point \( x = 0 \) leading to \( f_{x=0} = y^2 - 1 \). This has two roots and thus the stack over the cell consists of two sections and three sectors.

- Combining stacks for the cells in \( \mathbb{R}^1 \) gives the CAD of \( \mathbb{R}^2 \) with 13 cells.
How to build a CAD - Example

Consider the circle defined by the graph of $f = y^2 + x^2 - 1$.

- $P(f)$ gives the univariate polynomials $\{x - 1, x + 1\}$.
- The roots are $\pm 1$ so a CAD of $\mathbb{R}^1$ is produced with 5 cells.

- Consider the cell where $-1 < x < 1$. We can take the sample point $x = 0$ leading to $f_{x=0} = y^2 - 1$. This has two roots and thus the stack over the cell consists of two sections and three sectors.
- Combining stacks for the cells in $\mathbb{R}^1$ gives the CAD of $\mathbb{R}^2$ with 13 cells.
Consider the circle defined by the graph of \( f = y^2 + x^2 - 1 \).

- \( P(f) \) gives the univariate polynomials \( \{x - 1, x + 1\} \).
- The roots are \( \pm 1 \) so a CAD of \( \mathbb{R}^1 \) is produced with 5 cells.

- Consider the cell where \(-1 < x < 1\). We can take the sample point \( x = 0 \) leading to \( f_{x=0} = y^2 - 1 \). This has two roots and thus the stack over the cell consists of two sections and three sectors.
- Combining stacks for the cells in \( \mathbb{R}^1 \) gives the CAD of \( \mathbb{R}^2 \) with 13 cells.
How to build a CAD - Example

Consider the circle defined by the graph of $f = y^2 + x^2 - 1$.

- $P(f)$ gives the univariate polynomials $\{x - 1, x + 1\}$.
- The roots are $\pm 1$ so a CAD of $\mathbb{R}^1$ is produced with 5 cells.

Consider the cell where $-1 < x < 1$. We can take the sample point $x = 0$ leading to $f_{x=0} = y^2 - 1$. This has two roots and thus the stack over the cell consists of two sections and three sectors.

Combining stacks for the cells in $\mathbb{R}^1$ gives the CAD of $\mathbb{R}^2$ with 13 cells.
Consider the circle defined by the graph of \( f = y^2 + x^2 - 1 \).

- \( P(f) \) gives the univariate polynomials \( \{x - 1, x + 1\} \).
- The roots are \( \pm 1 \) so a CAD of \( \mathbb{R}^1 \) is produced with 5 cells.

- Consider the cell where \(-1 < x < 1\). We can take the sample point \( x = 0 \) leading to \( f_{x=0} = y^2 - 1 \). This has two roots and thus the stack over the cell consists of two sections and three sectors.

- Combining stacks for the cells in \( \mathbb{R}^1 \) gives the CAD of \( \mathbb{R}^2 \) with 13 cells.
A CAD of $\mathbb{R}^2$ for a circle is built along the way to a CAD of $\mathbb{R}^3$ for a sphere. The lifting is repeated for each cell in the CAD of $\mathbb{R}^2$. 
A CAD of $\mathbb{R}^2$ for a circle is built along the way to a CAD of $\mathbb{R}^3$ for a sphere. The lifting is repeated for each cell in the CAD of $\mathbb{R}^2$. 
CAD was originally invented for quantifier elimination (QE).

A formula is a boolean connection of polynomial equations and inequations. A formula may contain the quantifiers \( \forall \) (for all) or \( \exists \) (there exists). Quantifier elimination is the task of identifying an equivalent formula which is quantifier free.

A CAD for the polynomials in a formula can be used for QE. QE is used in turn to solve problems in engineering and science.
CAD can in theory be used for robot motion planning.

Define the robots and the environment using polynomials and calculate a CAD. The descriptions of individual cells can identify admissible positions for the robot and adjacency information would allow for path finding.
CAD can be used to improve computer algebra itself. At Bath we work on schemes to improve the simplification of formulae involving multi-valued functions. The basic idea is to decompose the domain according to the branch cuts of the functions involved and test proposed identities on each cell.

The branch cuts of $\arcsin\left(2z\sqrt{1-z^2}\right)$. 
Why not to build a CAD?

- Can be computationally expensive.
- Algorithms have, in the worst case, doubly exponential complexity in the number of variables.
- Output can contain far more information that required for application.

It is important to optimise the CAD algorithm for its application.
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.
- Results and algorithms on the adjacency of CAD cells.
- CAD via triangular decomposition (developed here at Western and behind Maple’s CAD command).
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.
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.
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 in 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
   - Why build a CAD?
   - CAD for Boolean Combinations

2. Developing TTICAD
   - Motivation
   - New Projection Operator
   - Important Technicalities

3. TTICAD in Practice
   - Implementation in Maple
   - Experimental Results
   - Conclusions
Consider the polynomials:

\[ f_1 := x^2 + y^2 - 1 \quad \quad \quad g_1 := xy - \frac{1}{4} \]
\[ f_2 := (x - 4)^2 + (y - 1)^2 - 1 \quad \quad \quad g_2 := (x - 4)(y - 1) - \frac{1}{4} \]

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

\[ \Phi := (f_1 = 0 \land g_1 < 0) \lor (f_2 = 0 \land g_2 < 0) \]

We could solve the problem using a full sign-invariant CAD for \( \{f_1, g_1, f_2, g_2\} \), (\texttt{QEPCAD} and \texttt{MAPLE} both use 317 cells). In particular, the induced CAD of \( \mathbb{R}^1 \) has 41 cells (identifying 20 points).
Example: graph of polynomials
Consider the polynomials:

\[ f_1 := x^2 + y^2 - 1 \quad \quad g_1 := xy - \frac{1}{4} \]
\[ f_2 := (x - 4)^2 + (y - 1)^2 - 1 \quad \quad g_2 := (x - 4)(y - 1) - \frac{1}{4} \]

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

\[ \Phi := (f_1 = 0 \land g_1 < 0) \lor (f_2 = 0 \land g_2 < 0) \]

We could solve the problem using a full sign-invariant CAD for \( \{f_1, g_1, f_2, g_2\} \). \texttt{Qepcad} and \texttt{Maple} would both use 317 cells. The induced CAD of \( \mathbb{R}^1 \) has 41 cells (identifying 21 points).
Example: sign-invariant CAD

All curve intersections identified.
We could instead employ the theory of equational constraints. Although $\Phi$ 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. The induced CAD of $\mathbb{R}^1$ has 33 cells (identifying 16 points).
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 $\mathcal{E} = \{E_i\}_{i=1}^t$ nonempty 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 our paper, along with a formal verification. Key points:

- Apply the reduced projection once to find projection polynomials $\mathcal{P}$ in $n-1$ variables.
- Use McCallum’s verified algorithm to build a sign-invariant CAD of $\mathbb{R}^{n-1}$ for $\mathcal{P}$.
- Perform a final lift with respect to the equational constraints.
A TTICAD for the motivating example is built with 105 cells, (compared to 317 and 249). The induced CAD of $\mathbb{R}^1$ has 25 cells identifying 12 points, (compared to 20 and 16).
A TTICAD for the motivating example is built with 105 cells, (compared to 317 and 249). The induced CAD of $\mathbb{R}^1$ has 25 cells identifying 12 points, (compared to 20 and 16).
Example: TTICAD

A TTICAD for the motivating example is built with 105 cells, (compared to 317 and 249). The induced CAD of $\mathbb{R}^1$ has 25 cells identifying 12 points, (compared to 20 and 16).
A TTICAD for the motivating example is built with 105 cells, (compared to 317 and 249). The induced CAD of $\mathbb{R}^1$ has 25 cells identifying 12 points, (compared to 20 and 16).
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:

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

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

1 Introduction
   - Cylindrical Algebraic Decomposition
   - Why build a CAD?
   - 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 and 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/

Utilises existing commands relating to CAD from Maple’s RegularChains package.

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

There are various existing implementations of CAD including Qepcad, Maple and 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/

Utilises existing commands relating to CAD from Maple’s RegularChains package.

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.
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


Cylindrical algebraic decomposition for boolean combinations.
Preprint at http://opus.bath.ac.uk/33926.

Contact Details

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

Thanks for listening!