Advancing computer algebra through other algorithmic domains

Matthew England
Coventry University
Coventry, UK

Future Algorithms
Online Conference
July 2024

Al-Khwarizmi

Al-Khwarizmi (780−850) was a Persian scholar whose work the conference is named after. He made many contributions including:

- Introducing the Arabic numeral system to the west. The latinised form of his name was first applied to **algorism** for arithmetic with such numericals; and later to **algorithm** meaning a sequence of rigorous instructions to perform a task.
- He wrote a book on solve linear and quadratic equations, with abbreviated name **Al-Jabr**. From this we get the word **algebra** for mathematical statements using variables for unspecified values.

My talk today is on **algorithms** for **algebra**!

Al-Khwarizmi

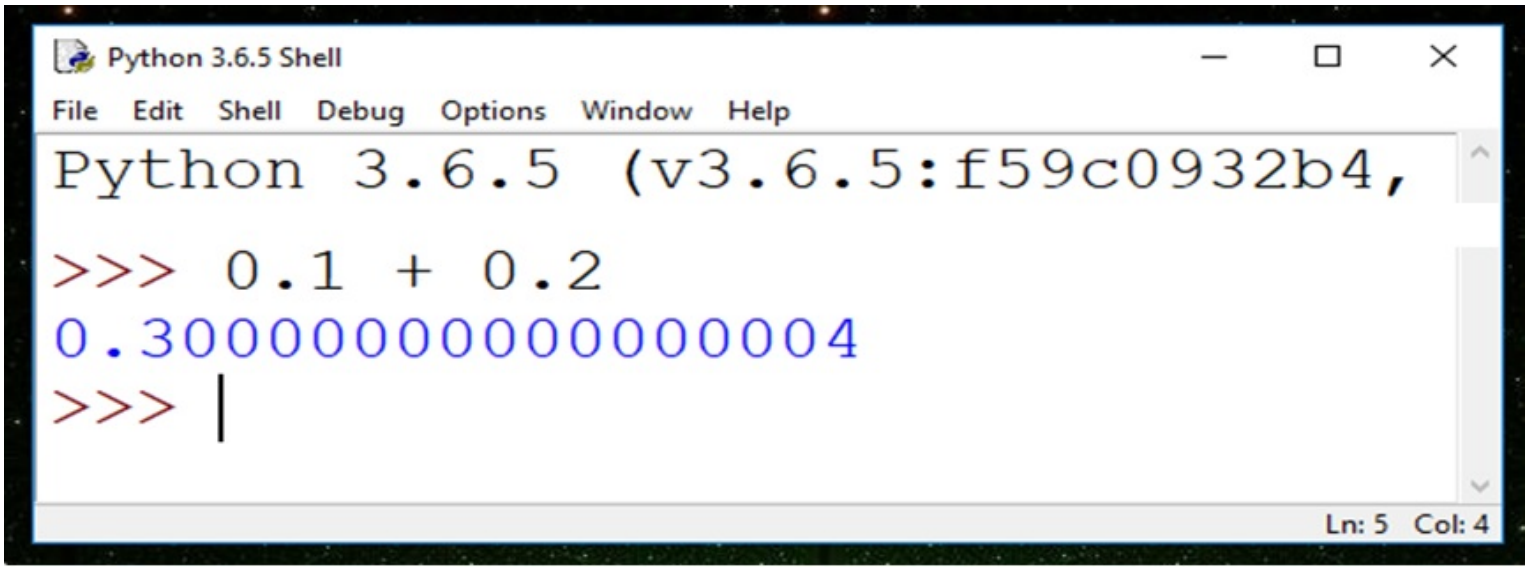
Al-Khwarizmi (780−850) was a Persian scholar whose work the conference is named after. He made many contributions including:

- Introducing the Arabic numeral system to the west. The latinised form of his name was first applied to **algorism** for arithmetic with such numericals; and later to **algorithm** meaning a sequence of rigorous instructions to perform a task.
- He wrote a book on solve linear and quadratic equations, with abbreviated name **Al-Jabr**. From this we get the word **algebra** for mathematical statements using variables for unspecified values.

My talk today is on **algorithms** for **algebra**!

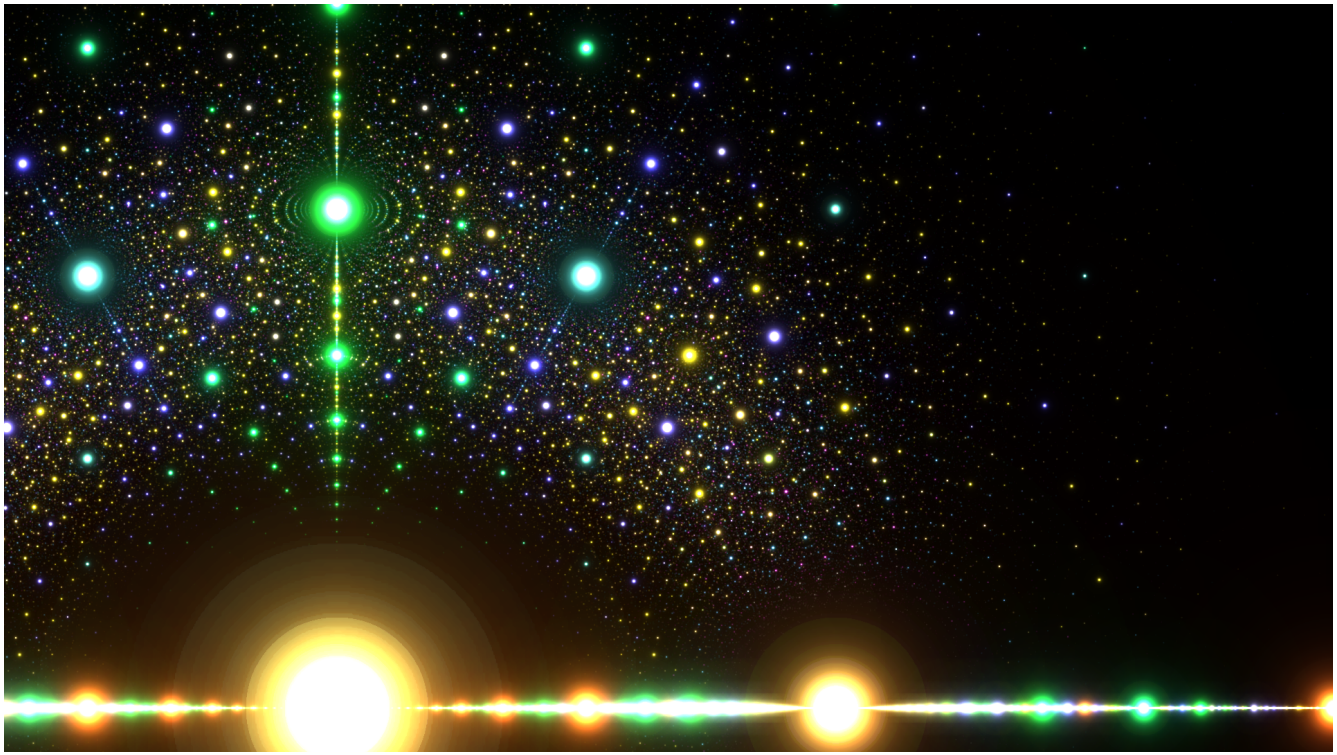
What is Computer Algebra? (Slide 2/26)

A **Computation Algebra System (CAS)** is computer software for manipulating mathematical expressions and objects. A CAS gives *exact answers*, in contrast to the more common approximate numbers given by floating point arithmetic.
For example, a CAS can compute with the symbols $\frac{1}{3}$, π and $\sqrt{2}$ rather than their approximations 0.33333, 3.14159 and 1.41421.
Note: floating point arithmetic inaccurate even for finite decimals!



Exact Arithmetic

You learnt rules (algorithms) to do arithmetic with fractions at school. There are also algorithms to do arithmetic with **algebraic numbers**: those defined as the root of a polynomial equation, like $\sqrt{2}$ which we define as the second root of $x^2 - 2$.

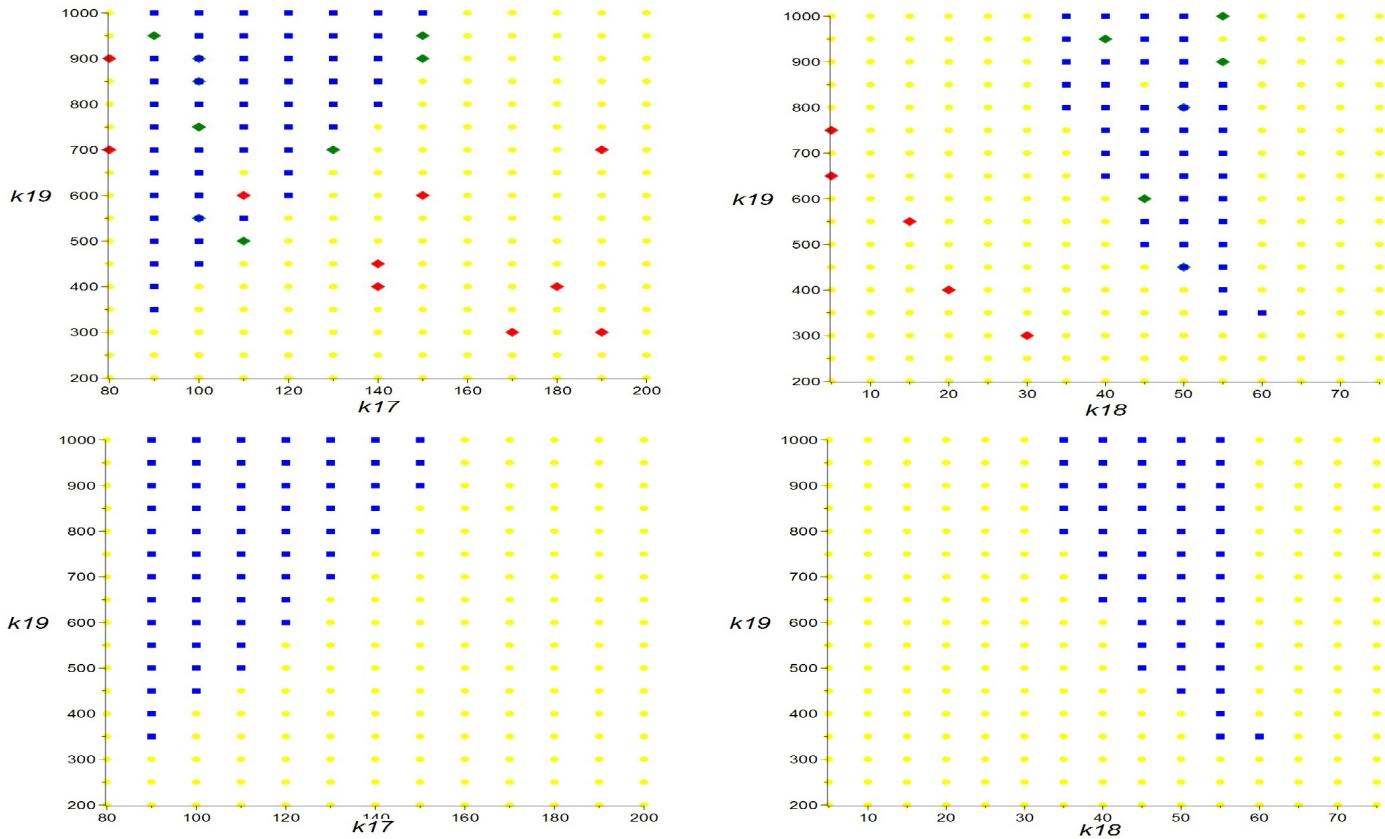


Source: Stephen J. Brooks (Wikipedia)

Why Use Computer Algebra? 1/3

Symbolic Computation usually take lots more computer resources than numerical computation. So why use it?

- The problem may be ill suited for a numerical solution (e.g. chaotic behaviour; small inaccuracies give different answers).

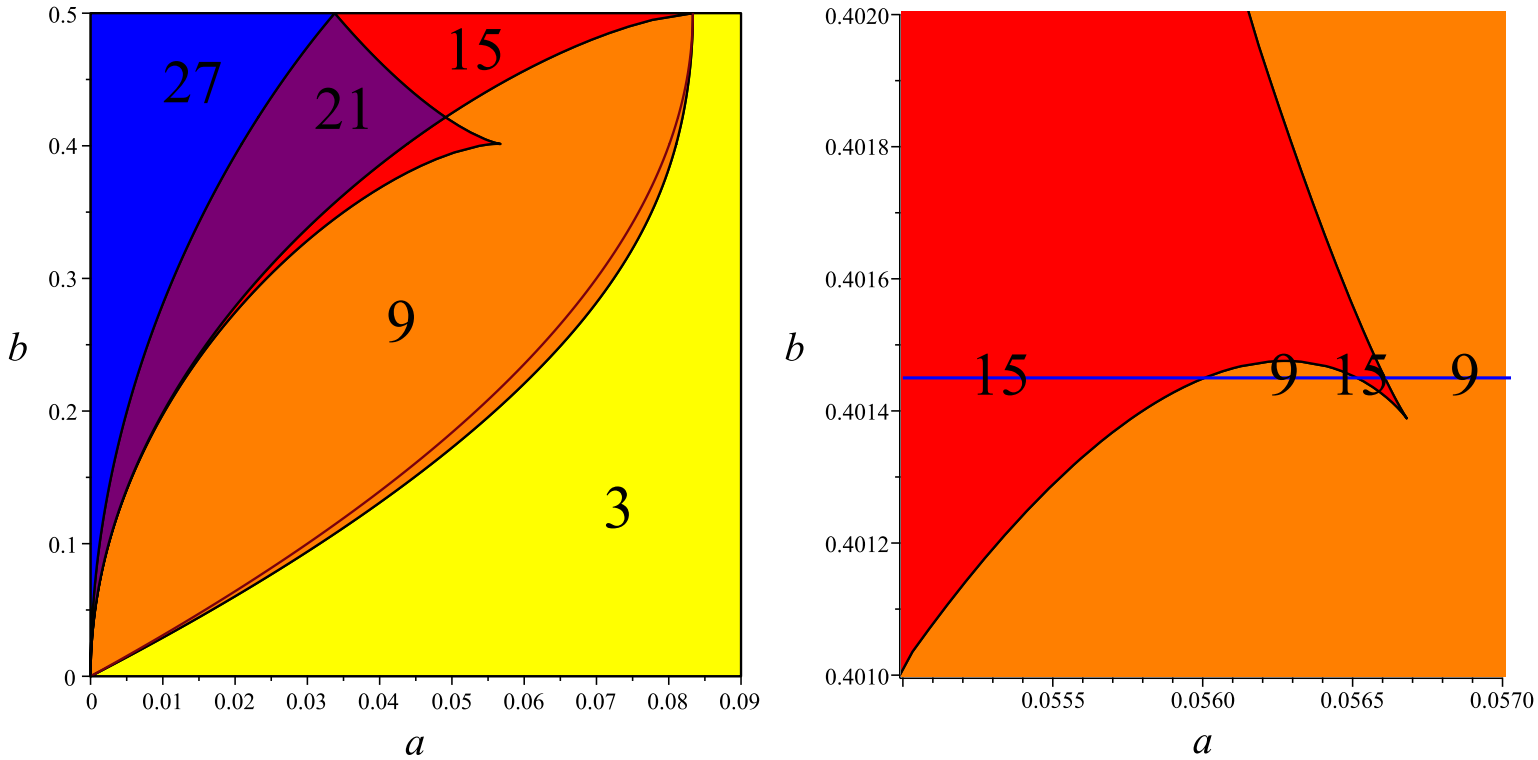


Why Use Computer Algebra? 2/3

- Accuracy may be particularly important (e.g. safety critical systems; formal verification; large numerics).

Why Use Computer Algebra? 3/3

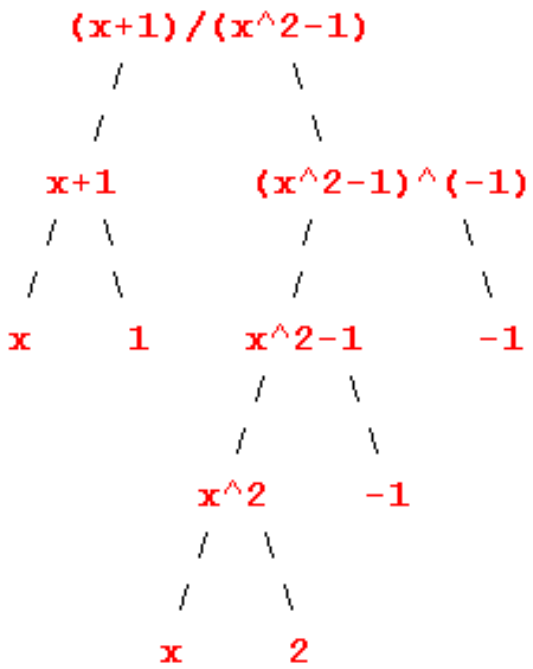
- Because exact solutions offer fundamental insight that can be missing from numerical ones. Understand not just what the solution is but what it means; what properties it has.



Computer Algebra Research Involves... (Slide 7/26)

- **Computer Science:** Inventing algorithms and data structures; implementing them; optimising them; evaluating them and designing the accompanying user-interfaces.

E.g. A CAS will use unassigned variables as objects: so rather than causing errors as they would in normal programming languages, these are stored in expression trees like the below



We need rules for manipulating, simplifying and evaluating such trees.

Computer Algebra Research Involves... (Slide 8/26)

- **Mathematics:** Any algorithms we write should be proven to provide the exact answer. It is not enough to show some tests of them working: we need mathematical guarantees they will always work!

Computer Algebra was one of the earliest forms of computing (the first CASs were released in the 1960s). But it is still an active area of research today.

I am currently employed on the UKRI EPSRC funded **DEWCAD Project**: Pushing Back the Doubly-Exponential Wall of Cylindrical Algebraic Decomposition. Project # EP/T015748/1.

Engineering and Physical Sciences Research Council

Quantifiers

Algebra uses variables to express and model the real world. Our models can be far more expressive if we also allow **quantifiers**: operators to specify how many individuals in the domain of discourse for a variable are concerned by the statement.

- We use the **Existential Quantifier** ($\exists$ pronounced *there exists*) to specify that at least one individual is concerned.
- We use the **Universal Quantifier** ($\forall$ pronounced *for all*) to specify that all individuals are concerned.

When quantifiers are applied to real polynomial constraint systems, it is a fact that there always exists an *equivalent* logical statement that does not involve quantifiers.

- Statements logically equivalent if true for the same values.
- The quantifier free version is often longer but also can be easier to understand / analyse / visualise.

Quantifiers

Algebra uses variables to express and model the real world. Our models can be far more expressive if we also allow **quantifiers**: operators to specify how many individuals in the domain of discourse for a variable are concerned by the statement.

- We use the **Existential Quantifier** ($\exists$ pronounced *there exists*) to specify that at least one individual is concerned.
- We use the **Universal Quantifier** ($\forall$ pronounced *for all*) to specify that all individuals are concerned.

When quantifiers are applied to real polynomial constraint systems, it is a fact that there always exists an *equivalent* logical statement that does not involve quantifiers.

- Statements logically equivalent if true for the same values.
- The quantifier free version is often longer but also can be easier to understand / analyse / visualise.

Real QE

Real Quantifier Elimination (Real QE)

Given: A quantified formulae of real polynomial constraints;
Produce: a logically equivalent quantifier free formula.

Fully quantified examples:

Input: $\exists x, x^2 + 3x + 1 > 0$

Output: `True` e.g. when $x = 0$

Input: $\forall x, x^2 + 3x + 1 > 0$

Output: `False` e.g. when $x = -1$

Input: $\forall x, x^2 + 1 > 0$

Output: `True`

Partially quantified example:

Input: $\forall x, x^2 + bx + 1 > 0$

Output: ???

The answer depends on the free (unquantified) variable, b.

Real QE

Real Quantifier Elimination (Real QE)

Given: A quantified formulae of real polynomial constraints;
Produce: a logically equivalent quantifier free formula.

Fully quantified examples:

Input: $\exists x, x^2 + 3x + 1 > 0$
Output: `True` e.g. when $x = 0$

Input: $\forall x, x^2 + 3x + 1 > 0$
Output: `False` e.g. when $x = -1$

Input: $\forall x, x^2 + 1 > 0$
Output: `True`

Partially quantified example:

Input: $\forall x, x^2 + bx + 1 > 0$
Output: ???

The answer depends on the free (unquantified) variable, b.

Partially quantified Tarski formulae example (Slide 11/26)

Consider $\forall x, x^2 + bx + 1 > 0$.
When $b = 0$ and $b = 3$:

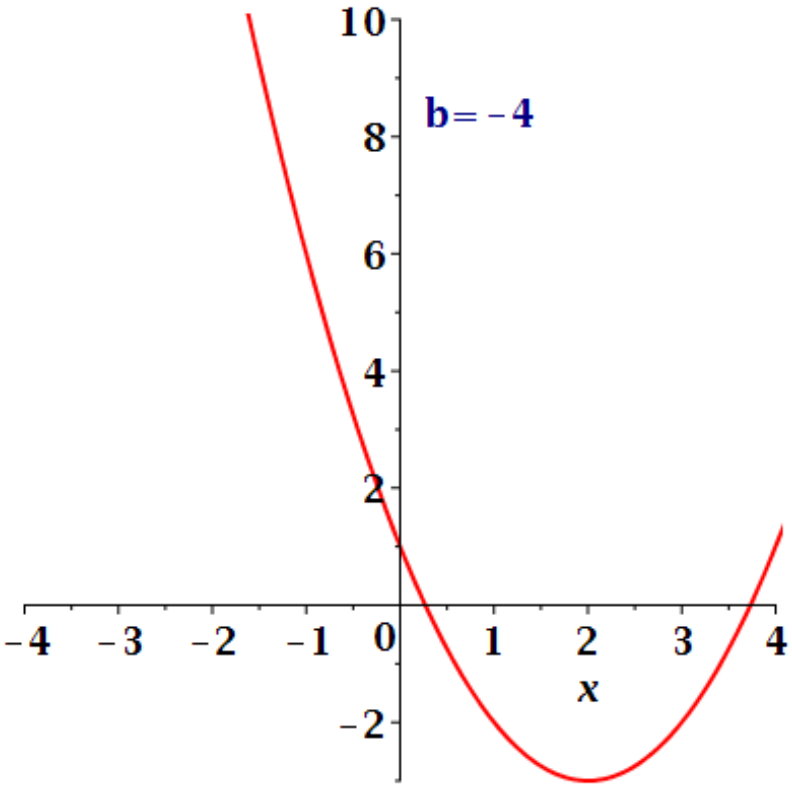
$$\forall x, x^2 + 1 > 0 \text{ is } \texttt{True},$$
$$\forall x, x^2 + 3x + 1 > 0 \text{ is } \texttt{False}.$$

So in general the answer depends on b.

Input: $\forall x, x^2 + bx + 1 > 0$

Output: $(-2 < b) \wedge (b < 2)$

The technology we look at today can answer such questions automatically.



Partially quantified Tarski formulae example (Slide 11/26)

Consider $\forall x, x^2 + bx + 1 > 0$.
When $b = 0$ and $b = 3$:

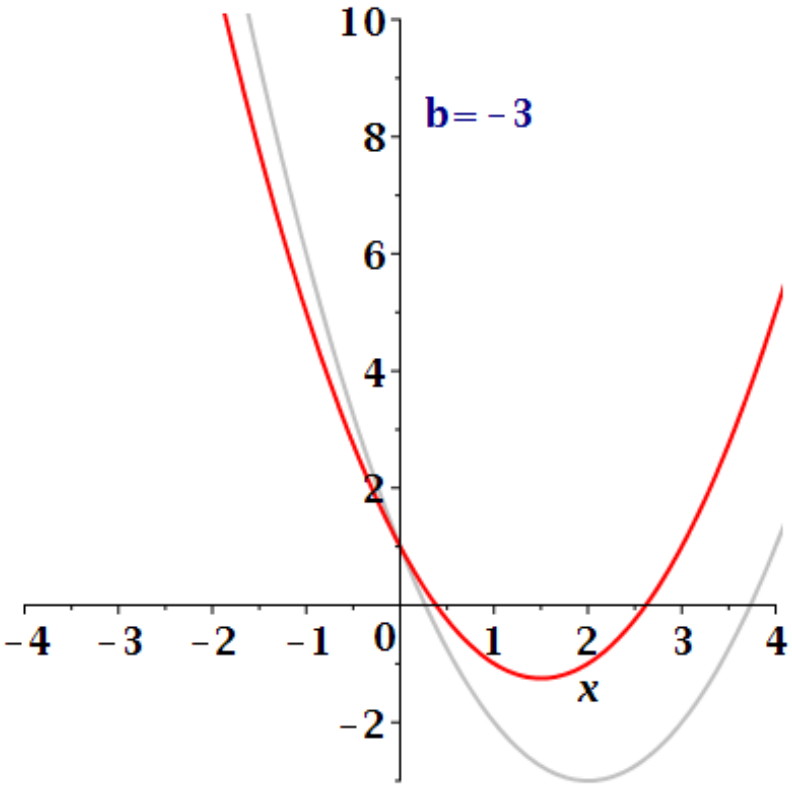
$$\forall x, x^2 + 1 > 0 \text{ is } \texttt{True} \ ,$$
$$\forall x, x^2 + 3x + 1 > 0 \text{ is } \texttt{False} \ .$$

So in general the answer depends on b.

Input: $\forall x, x^2 + bx + 1 > 0$

Output: $(-2 < b) \wedge (b < 2)$

The technology we look at today can answer such questions automatically.



Partially quantified Tarski formulae example (Slide 11/26)

Consider $\forall x, x^2 + bx + 1 > 0$.
When $b = 0$ and $b = 3$:

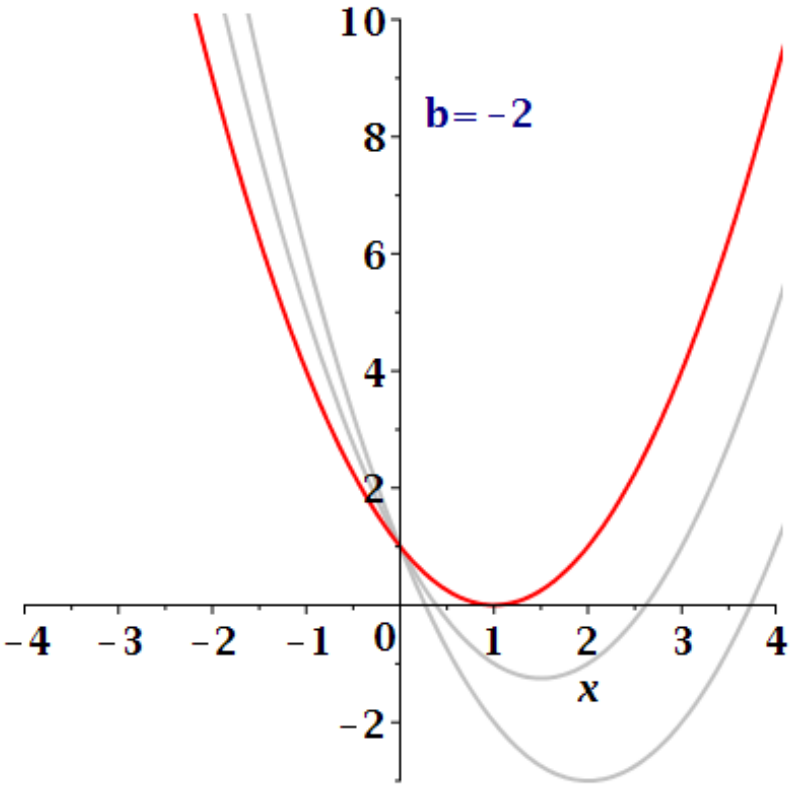
$$\forall x, x^2 + 1 > 0 \text{ is } \texttt{True},$$
$$\forall x, x^2 + 3x + 1 > 0 \text{ is } \texttt{False}.$$

So in general the answer depends on b.

Input: $\forall x, x^2 + bx + 1 > 0$

Output: $(-2 < b) \wedge (b < 2)$

The technology we look at today can answer such questions automatically.



Partially quantified Tarski formulae example (Slide 11/26)

Consider $\forall x, x^2 + bx + 1 > 0$.
When $b = 0$ and $b = 3$:

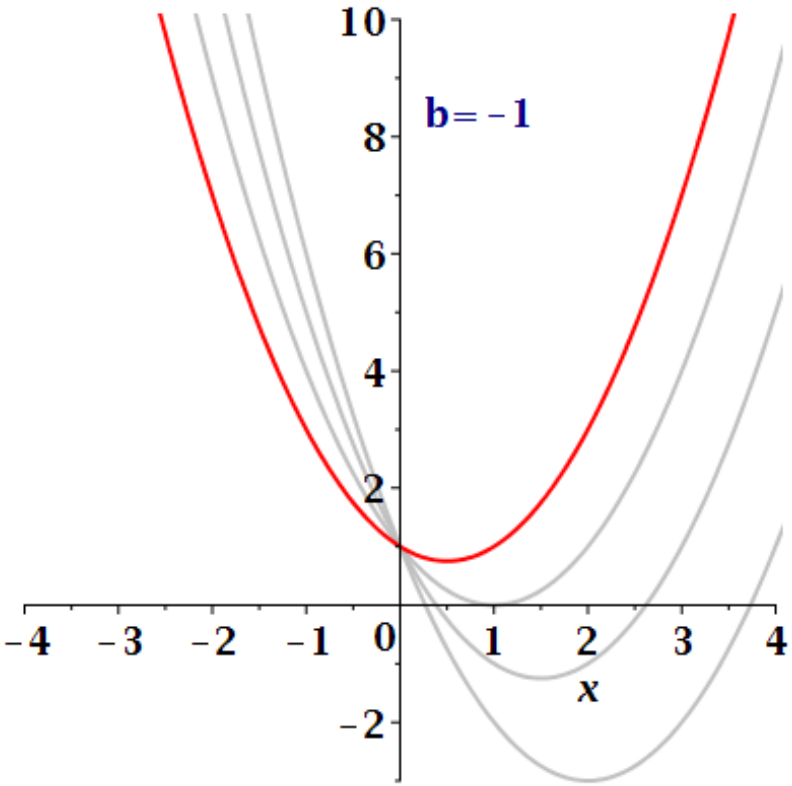
$$\forall x, x^2 + 1 > 0 \text{ is } \texttt{True},$$
$$\forall x, x^2 + 3x + 1 > 0 \text{ is } \texttt{False}.$$

So in general the answer depends on b.

Input: $\forall x, x^2 + bx + 1 > 0$

Output: $(-2 < b) \wedge (b < 2)$

The technology we look at today can answer such questions automatically.



Partially quantified Tarski formulae example (Slide 11/26)

Consider $\forall x, x^2 + bx + 1 > 0$.
When $b = 0$ and $b = 3$:

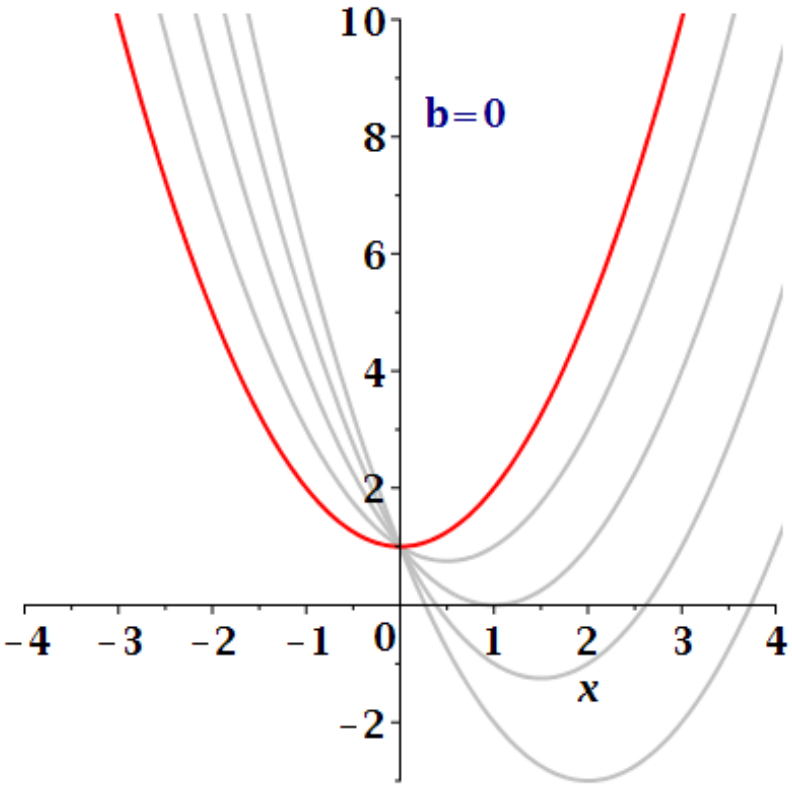
$$\forall x, x^2 + 1 > 0 \text{ is } \texttt{True},$$
$$\forall x, x^2 + 3x + 1 > 0 \text{ is } \texttt{False}.$$

So in general the answer depends on b.

Input: $\forall x, x^2 + bx + 1 > 0$

Output: $(-2 < b) \wedge (b < 2)$

The technology we look at today can answer such questions automatically.



Partially quantified Tarski formulae example (Slide 11/26)

Consider $\forall x, x^2 + bx + 1 > 0$.
When $b = 0$ and $b = 3$:

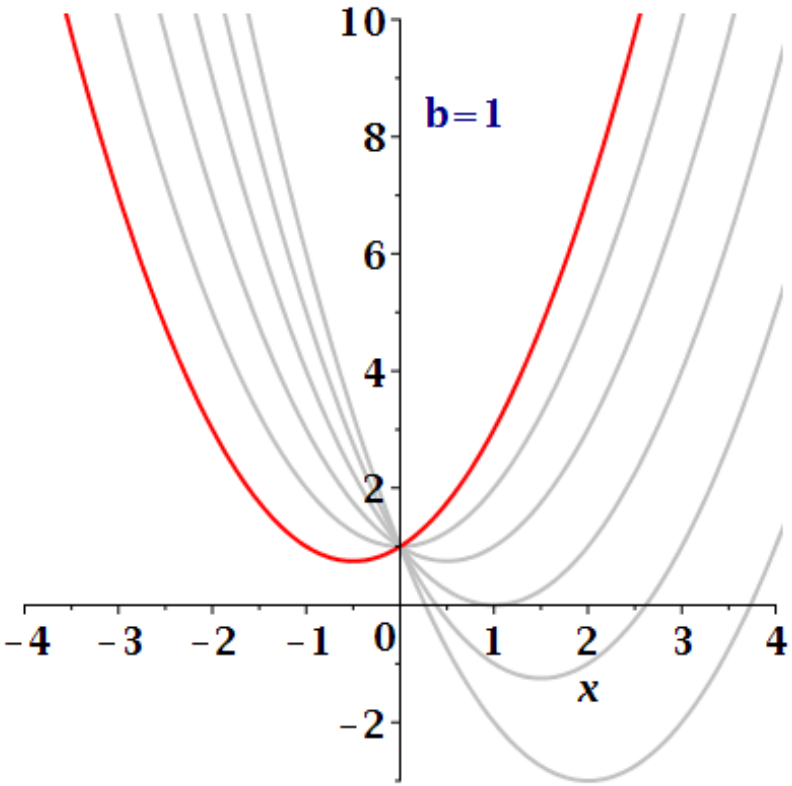
$$\forall x, x^2 + 1 > 0 \text{ is } \texttt{True},$$
$$\forall x, x^2 + 3x + 1 > 0 \text{ is } \texttt{False}.$$

So in general the answer depends on b.

Input: $\forall x, x^2 + bx + 1 > 0$

Output: $(-2 < b) \wedge (b < 2)$

The technology we look at today can answer such questions automatically.



Partially quantified Tarski formulae example

Consider $\forall x, x^2 + bx + 1 > 0$.
When $b = 0$ and $b = 3$:

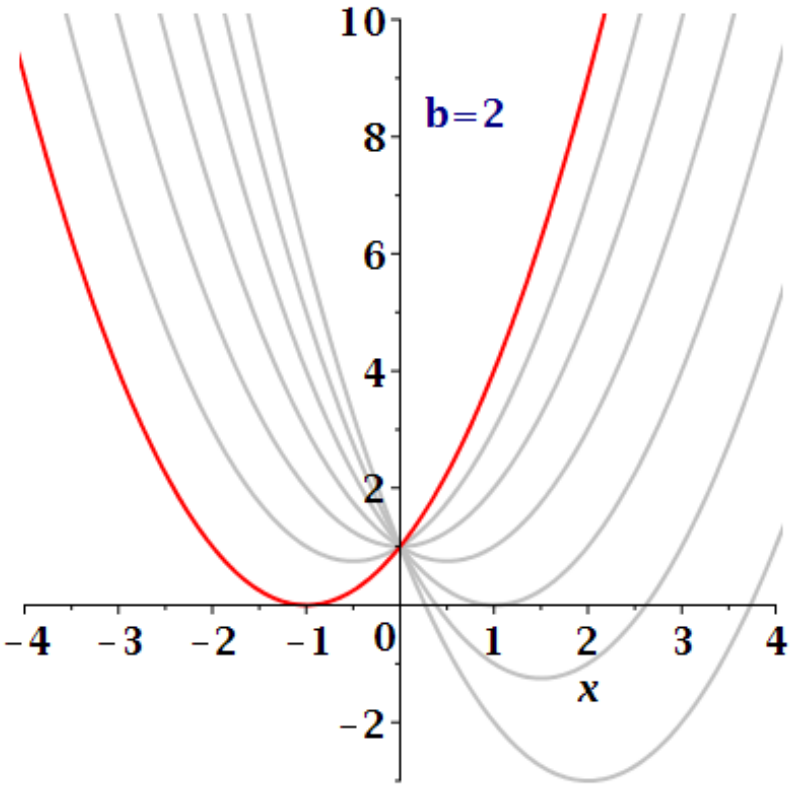
$$\forall x, x^2 + 1 > 0 \text{ is } \texttt{True},$$
$$\forall x, x^2 + 3x + 1 > 0 \text{ is } \texttt{False}.$$

So in general the answer depends on b.

Input: $\forall x, x^2 + bx + 1 > 0$

Output: $(-2 < b) \wedge (b < 2)$

The technology we look at today can answer such questions automatically.



Partially quantified Tarski formulae example (Slide 11/26)

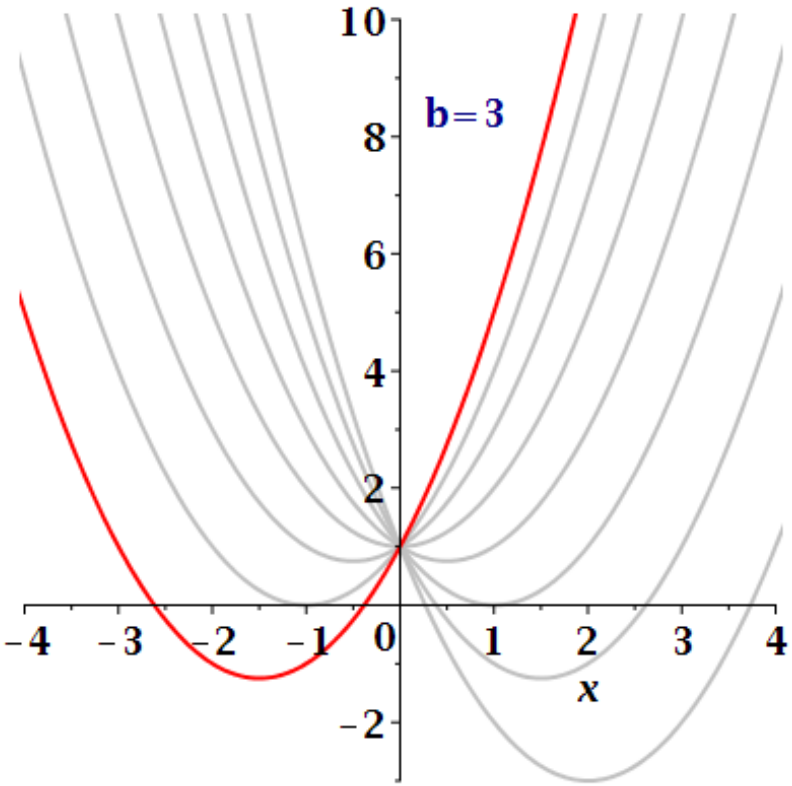
Consider $\forall x, x^2 + bx + 1 > 0$.
When $b = 0$ and $b = 3$:

$$\forall x, x^2 + 1 > 0 \text{ is } \texttt{True},$$
$$\forall x, x^2 + 3x + 1 > 0 \text{ is } \texttt{False}.$$

So in general the answer depends on b.

Input: $\forall x, x^2 + bx + 1 > 0$
Output: $(-2 < b) \wedge (b < 2)$

The technology we look at today can answer such questions automatically.



Partially quantified Tarski formulae example (Slide 11/26)

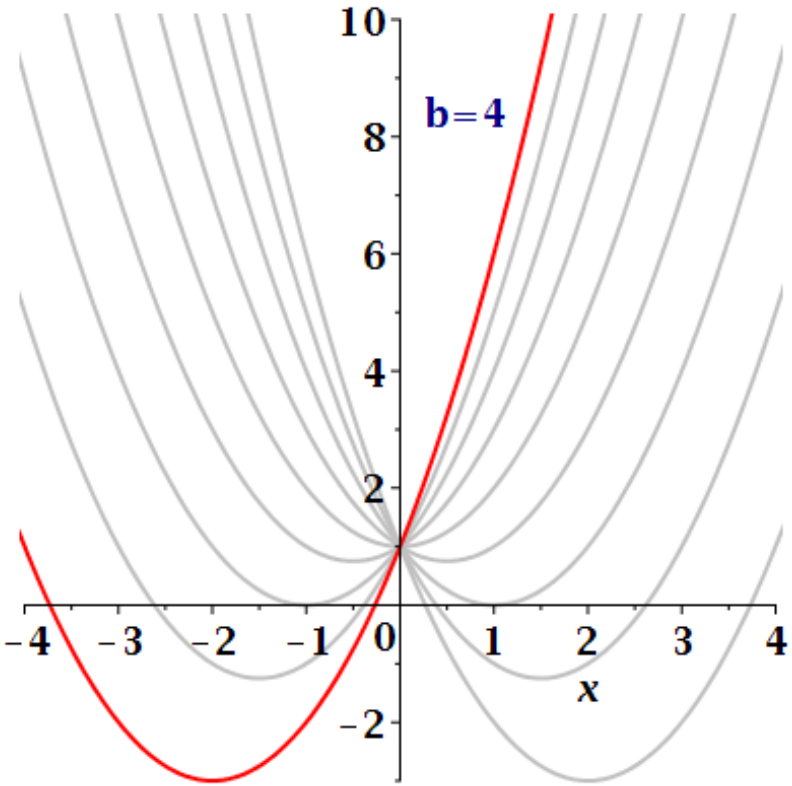
Consider $\forall x, x^2 + bx + 1 > 0$.
When $b = 0$ and $b = 3$:

$$\forall x, x^2 + 1 > 0 \text{ is } \texttt{True},$$
$$\forall x, x^2 + 3x + 1 > 0 \text{ is } \texttt{False}.$$

So in general the answer depends on b.

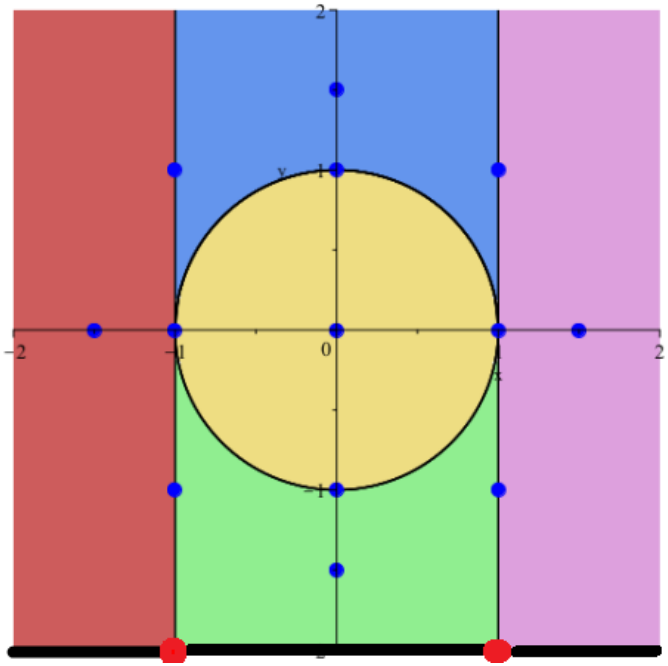
$$\text{Input: } \forall x, x^2 + bx + 1 > 0$$
$$\text{Output: } (-2 < b) \wedge (b < 2)$$

The technology we look at today can answer such questions automatically.



Cylindrical Algebraic Decomposition (Slide 12/26)

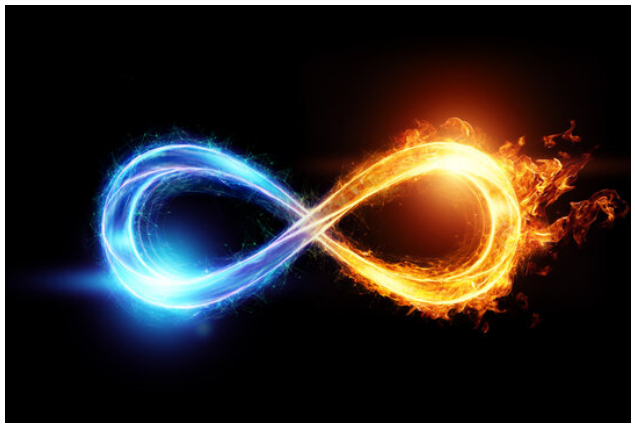
Real QE may be achieved by building a **Cylindrical Algebraic Decomposition (CAD)**. This is a mathematical object which **decomposes** the space of our variables into separate cells. Each cell is **algebraic** meaning it is defined by an (unquantified) algebraic formula. The cells are arranged **cylindrically** meaning they are stacked on top of each other.



Finally, the whole CAD is produced **sign-invariant** for a given set of polynomials: meaning it has the guarantee that each polynomial has a constant sign (positive, negative, or zero) in each cell.

Why is a CAD good?

A sign-invariant decomposition means we can test a **finite** set of points and understand what happens for the polynomial system in an **infinite** real space. We test one point per cell!

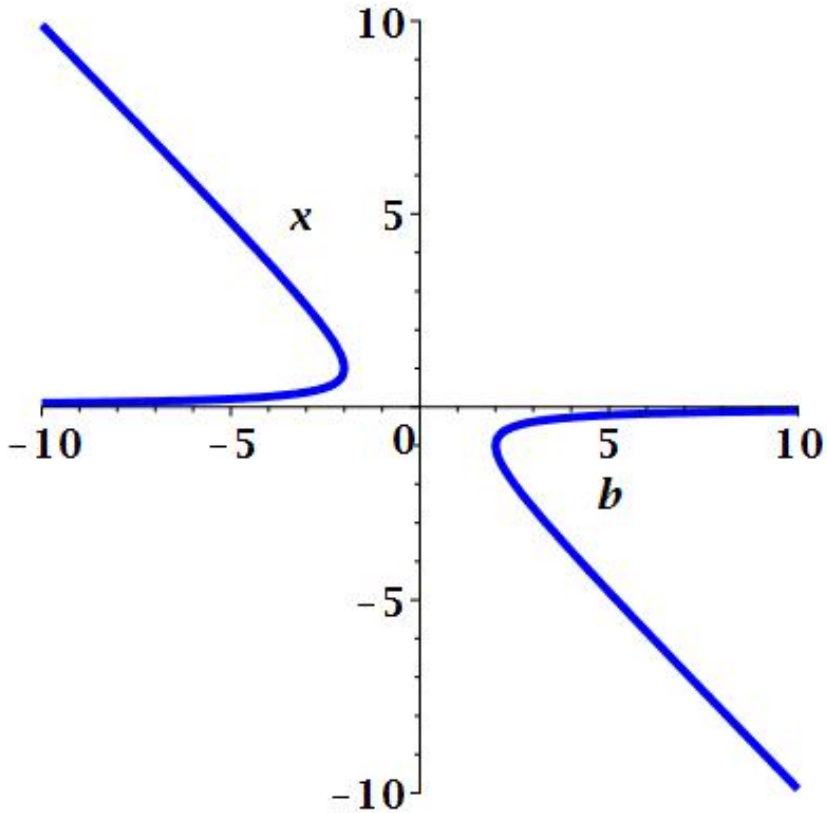


- The algebraic property means we can identify the cells of interest and then reconstruct solution formula to describe them easily.
- The cylindrical property means we can project and take inverse of cells easily. We will see why that is useful in a moment.

QE via CAD Example

How to determine with CAD?

$$\exists x, x^2 + bx + 1 \leq 0$$



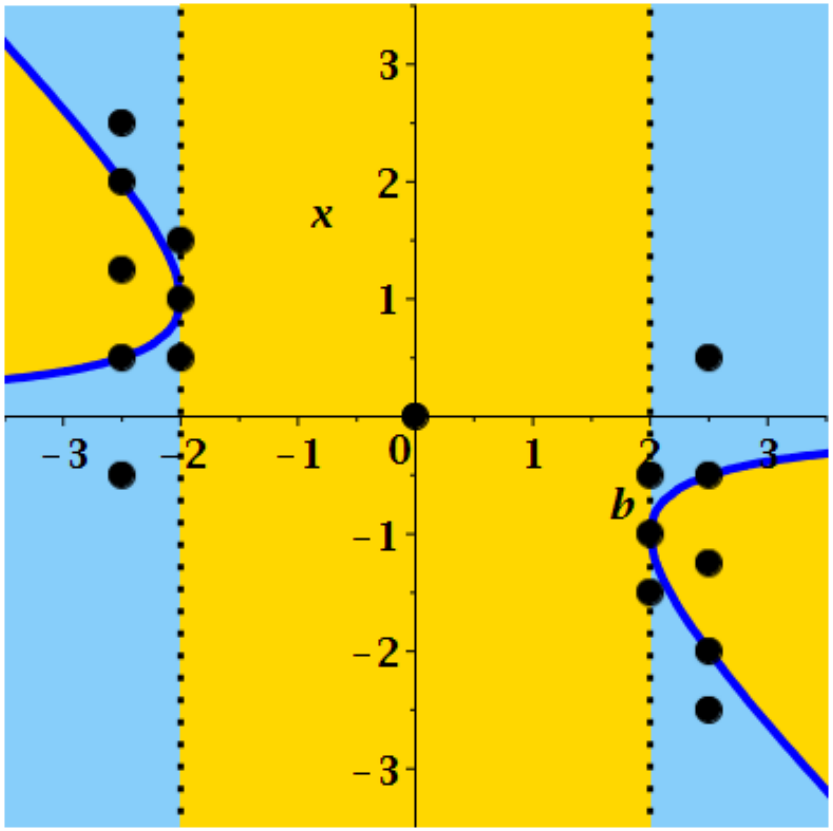
QE via CAD Example

How to determine with CAD?

$$\exists x, x^2 + bx + 1 \leq 0$$

To solve we:

Build a sign-invariant CAD for $f = x^2 + bx + 1$.



QE via CAD Example

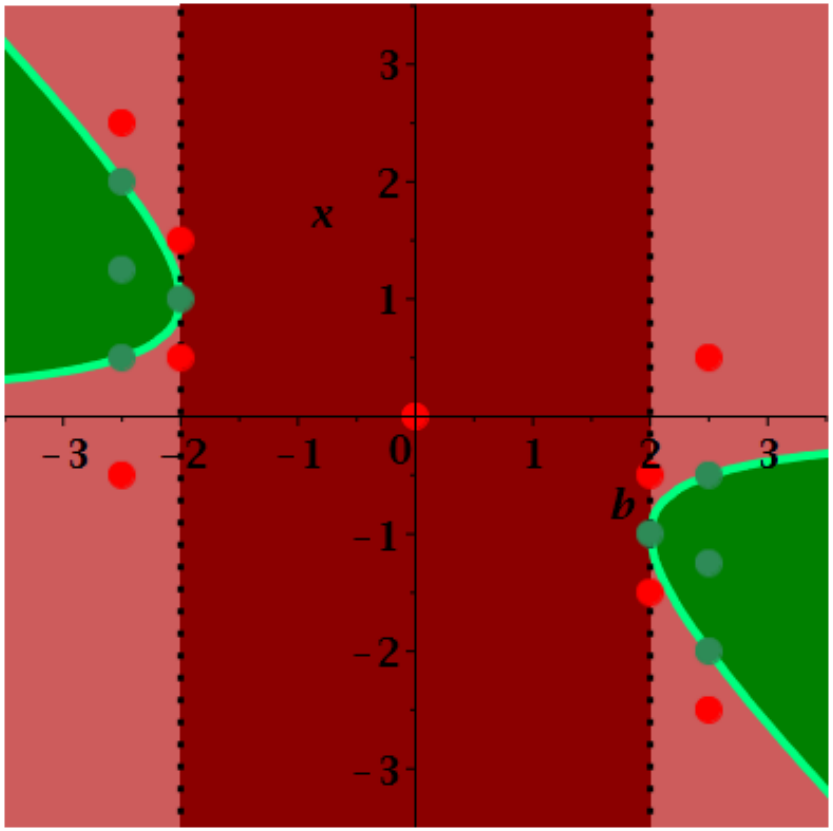
How to determine with CAD?

$$\exists x, x^2 + bx + 1 \leq 0$$

To solve we:

Build a sign-invariant CAD for $f = x^2 + bx + 1$.

Tag each cell true or false according to $f \leq 0$.



QE via CAD Example

How to determine with CAD?

$$\exists x, x^2 + bx + 1 \leq 0$$

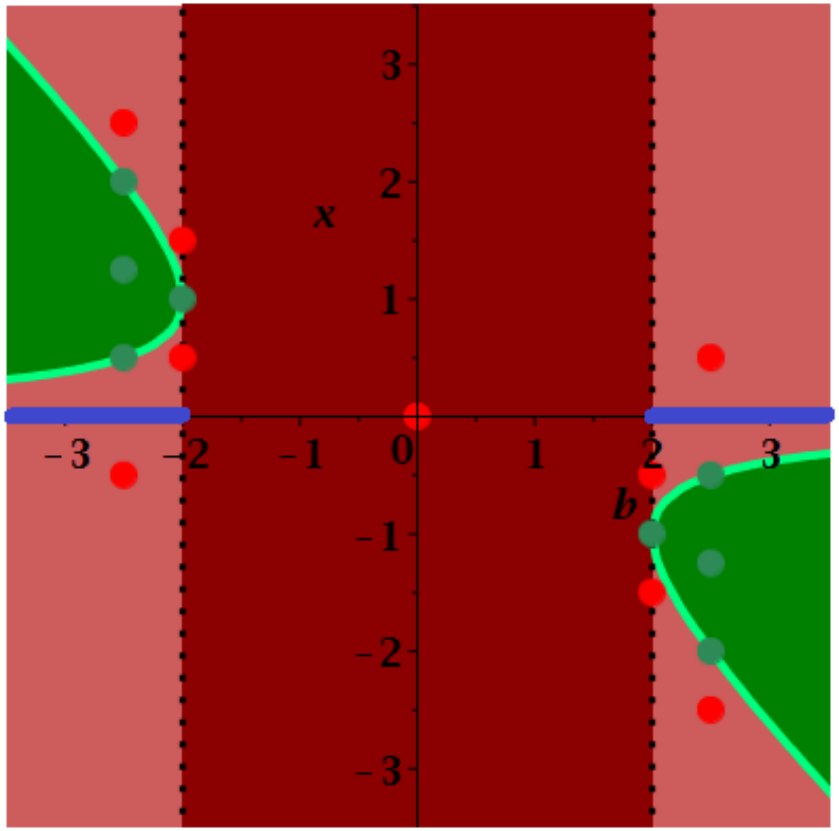
To solve we:

Build a sign-invariant CAD for $f = x^2 + bx + 1$.

Tag each cell true or false according to $f \leq 0$.

Take disjunction of projections of true cells:

$$b < -2 \vee b = -2 \vee b = 2 \vee b > 2$$



QE via CAD Example

How to determine with CAD?

$$\exists x, x^2 + bx + 1 \leq 0$$

To solve we:

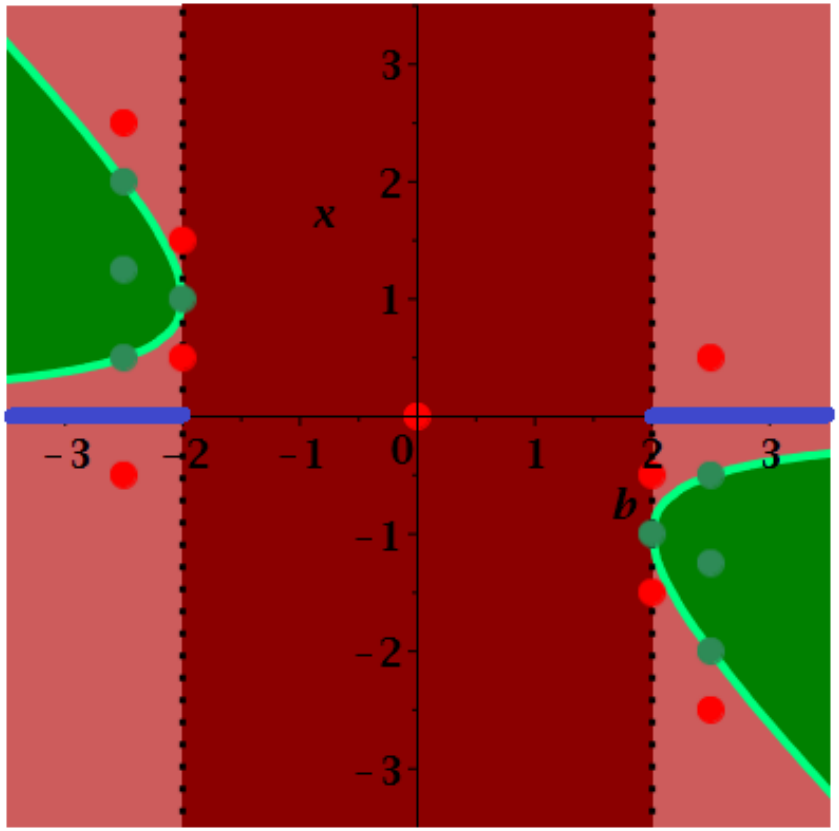
Build a sign-invariant CAD for $f = x^2 + bx + 1$.

Tag each cell true or false according to $f \leq 0$.

Take disjunction of projections of true cells:

$$\Longrightarrow$$

$$b \leq -2 \vee b \geq 2$$



Eliminating a universal quantifier

So we can eliminate existential quantifiers by **projecting** the true cells in a CAD. What about universal quantifiers?

- Eliminate universal quantifiers by using the relation

$$\forall x P(x) = \neg \exists x \neg P(x)$$

and then proceeding with existential QE.

Recall our original example was $\forall x, x^2 + bx + 1 > 0$. The above leads us to study $\exists x, x^2 + bx + 1 \leq 0$ which from the previous slide we know has solution $b \leq -2 \vee b \geq +2$. The solution to our universally quantified problem is then the negation of this: $-2 < b \wedge b < +2$ (as we found numerically earlier).

Note: cylindricity makes negating and projecting cells easy.

How to build a CAD?

The algorithm involves a sequence of algebraic computations, performed in a CAS.

- First a projection stage identifies lots more more polynomials whose roots indicate changes in the behaviour of the original input polynomials.
- Then we iteratively decompose space with respect to the roots of these, working at a sample point to render polynomials univariate and extrapolating conclusions to the cell.

It involves lots of **computer science** (programming the algorithm and data-structures) and **mathematics** (proving that our conclusions at a sample point are valid across the cell).

Trying out a Real QE tool yourself

Both of the big proprietary Computer Algebra Systems have QE implementations: MATHEMATICA (the `Resolve` command) and MAPLE (`RegularChains:-SemiAlgebraicSetTools`; `QuantifierElimination:-CylindricalAlgebraicDecompose`; and `RootFinding:-Parametric:-CellDecomposition`).

The specialist computer algebra system QEPCAD-B is dedicated to QE via CAD and is available for free. It can be used via an the interface TARSKI, is available as a sub-package of SAGE (for Python) and as part of GeoGebra Discovery which you can access for free online: https://autgeo.online/

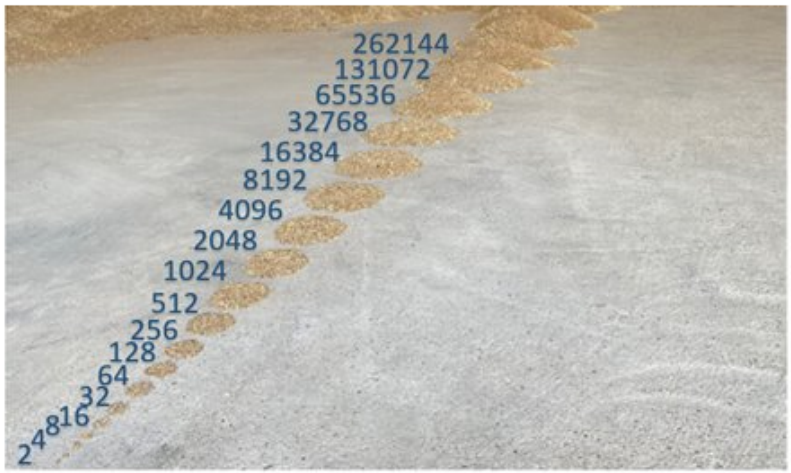
CAD implementations also exist in the SMT-solvers, SMT-RAT, YICES2, Z3 and CVC5.

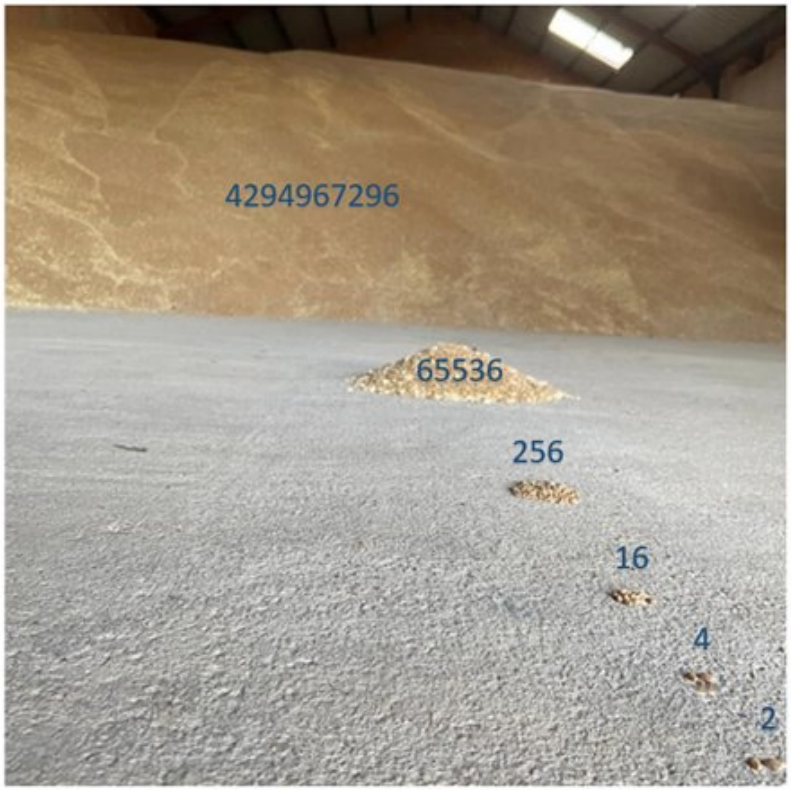
The doubly exponential wall of CAD

Real QE via CAD works in theory. But in practice it is too expensive. The cost of building a CAD can grow doubly exponentially with the number of variables.

$$2^{2^3} = 256 \qquad 2^{2^4} = 65,536 \qquad 2^{2^5} = 4,294,967,296$$

Images by Tereso del Rio





Pushing back the doubly exponential wall (Slide 19/26)

We have experimented with using ideas and tools from other areas of computer science:

- Machine Learning (AI)
- SAT-solvers

to improve CAD: pushing back the doubly exponential wall!

Machine Learning and Computer Algebra

Machine Learning (ML) tools use statistics and large quantities of data to learn how to perform tasks not explicitly programmed. ML is behind the modern Artificial Intelligence (AI) technology.

ML can only ever offer probabilistic guidance: it is never guaranteed to be correct always. 99% accuracy is great for chat generation and image recognition, but not to replace computer algebra where we need exact answers.

However, ML techniques can be used to make non-critical choices and guide searches. Used in this way, the ML performance has no affect on the mathematical correctness of the final result, but can make a substantial contribution to both computational efficiency and presentation of the end result.

Machine Learning and Computer Algebra (Slide 20/26)

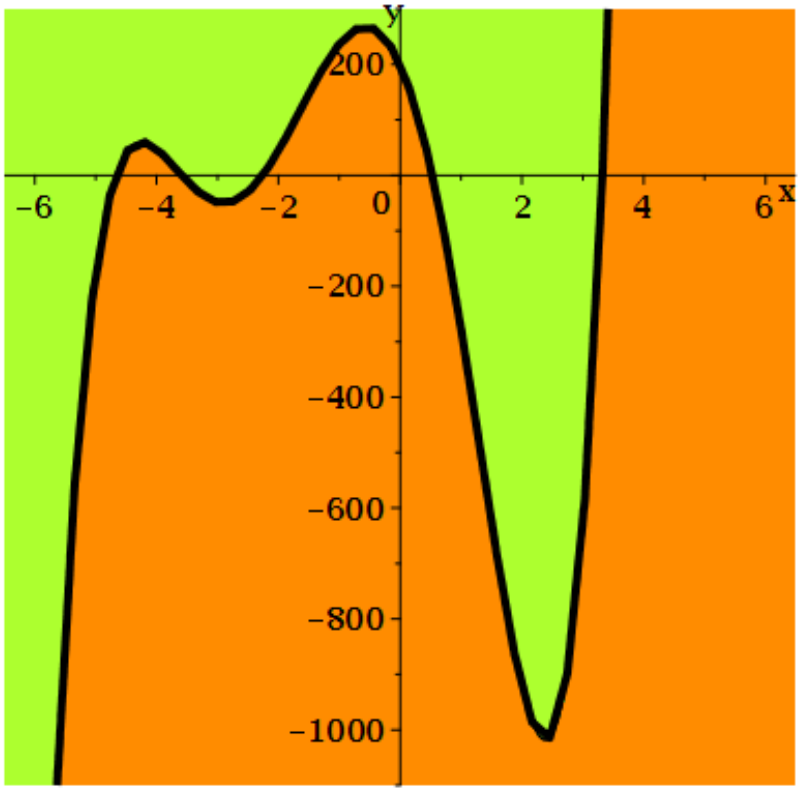
Machine Learning (ML) tools use statistics and large quantities of data to learn how to perform tasks not explicitly programmed. ML is behind the modern Artificial Intelligence (AI) technology.

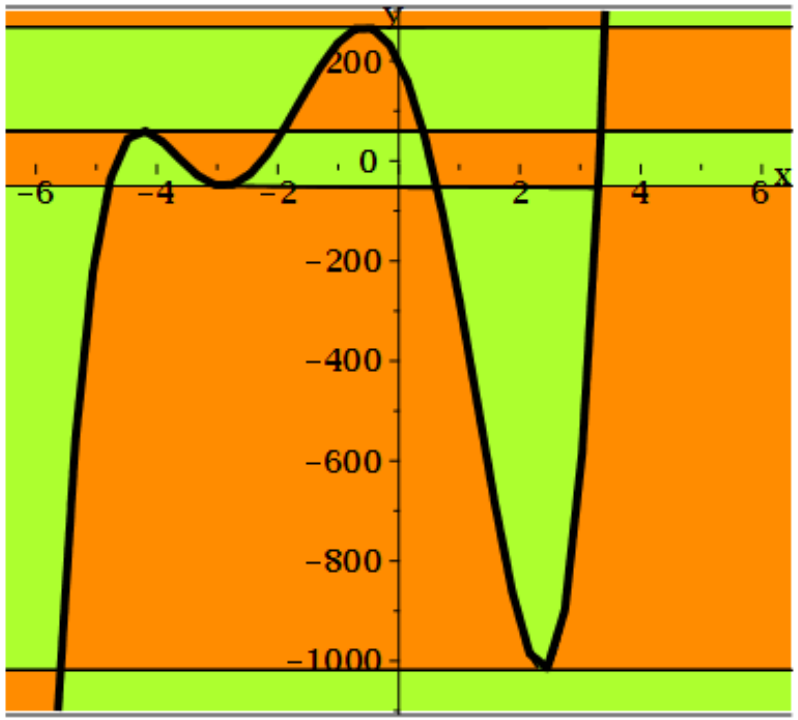
ML can only ever offer probabilistic guidance: it is never guaranteed to be correct always. 99% accuracy is great for chat generation and image recognition, but not to replace computer algebra where we need exact answers.

However, ML techniques can be used to make non-critical choices and guide searches. Used in this way, the ML performance has no affect on the mathematical correctness of the final result, but can make a substantial contribution to both computational efficiency and presentation of the end result.

Variable Ordering for CAD

CAD requires a variable ordering: there can be multiple valid orderings which lead to an acceptable decomposition, but some lead to smaller decompositions via less computation.





CAD Variable Ordering Choice

When using CAD for QE we must project variables in the order of quantification, but we are free to change order within quantifier blocks (and with the free variables).
E.g. to solve $\exists x \exists y \forall a \forall b F(x, y, a, b)$ we could use any of these orderings to build our CAD:

- $x \succ y \succ a \succ b$
- $x \succ y \succ b \succ a$
- $y \succ x \succ a \succ b$
- $y \succ x \succ b \succ a$

Any of these produce a CAD we could use for QE. But some may have **many** more cells which take **much** more time to produce.

Our work with ML for CAD

We have experimented with:

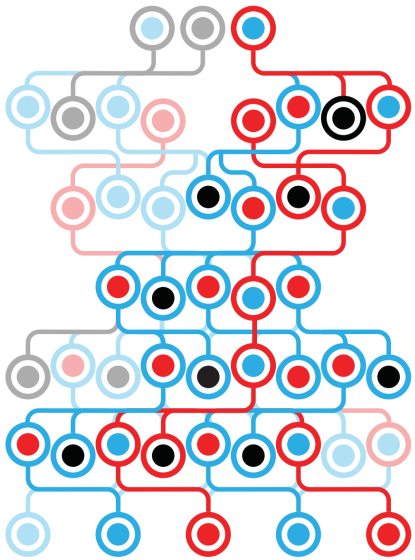
- different machine learning **models** (algorithms which train on data to make the predictions);
- different ways to **embed** the polynomials (how to present them to the ML models);
- different ways to **frame the problem** to machine learning (e.g. classify an ordering, predict how long an order will take, predict which variable to order next);
- **data augmentation** (to maximise our learning from the dataset of real world problems);
- **explainable AI** to produce ML independent code.

The first use of AI for computer algebra optimisation.

SAT-solvers

SAT solvers dedicated to solving Boolean satisfiability problems: formulae where the variables are just either `true` or `false`.

These problems *should* be hard: the space of possible solutions is huge and grows exponentially with number of variables. But SAT-solvers can solve practical problems with tens of thousands of variables!



They use sophisticated search algorithms which

1. often find solutions quickly; and
2. when there is no solution rule out huge chunks of the space in one go.

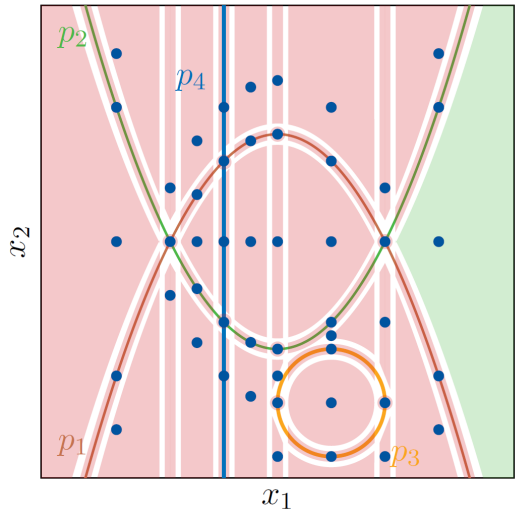
Re-designing CAD inspired by SAT

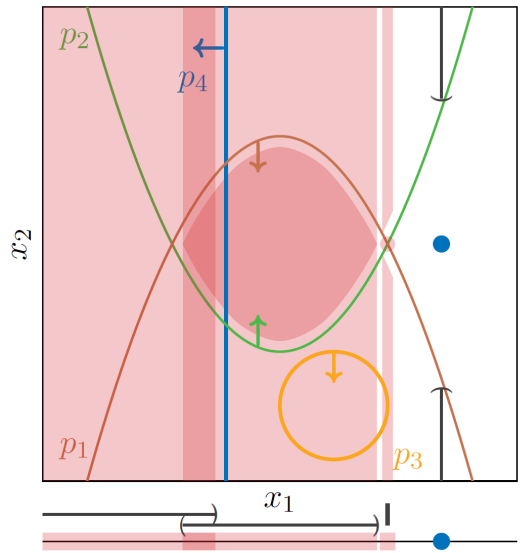
We have designed new algorithms:

- Use the same mathematical theory to prove their correctness as CAD;
- But perform different computation inspired by SAT-solvers.

The new algorithms:

- are search based, computing one CAD cell at a time instead of all at once;
- analyse to rule out similar cases (growing cells as large as possible);
- build coverings of space rather than decompositions (overlapping cells);
- use SAT-solvers themselves to analyse any logical structure.





Summary

- Computer algebra is an interesting and active field of research using both computer science (algorithms) and mathematics (algebra). Al-Khwarizmi would have been interested!
- Real Quantifier Elimination is a powerful tool in mathematics which can be useful in science and engineering.
- Although the mathematics of CAD was established almost 50 years ago, recently significance progress has been made by integrating this with other algorithmic domains: SAT-solvers and Machine Learning.

Message: When working on a problem look for inspiration from other areas that you can transfer to your area!

Thanks for Listening

These slides and links to formal publications are available here:

https://matthewengland.coventry.domains/