Techniques for NRA in SMT

How to solve Nonlinear Real Arithmetic

... and a lot of references

Contains mostly other people’s work!

Contains joint work with: Erika Ábrahám, Florian Corzilius, James Davenport, Matthew England, Rebecca Haehn, Jasper Nalbach
Satisfiability modulo theories

Let’s skip that...
Here: Theory of the Reals
Here: Theory of the Reals

Nonlinear Real Arithmetic:

- real variables $v := x_i \in \mathbb{R}$
- constants $c := q \in \mathbb{Z}$
- terms $t := v \mid c \mid t + t \mid t \cdot t$
- atoms $a := t \sim 0$, $\sim \in \{<, >, \leq, \geq, =, \neq\}$

Intuition: polynomials over real variables compared to zero.

Does cover: $t > t$, rational constants, division (encoding with auxiliary variables)

Does not cover: transcendental constants, non-polynomial functions

Linear arithmetic: essentially a solved problem.

Use Simplex (or sometimes Fourier-Motzkin)

Techniques for NRA in SMT

SMT for NRA

SMT for Nonlinear Real Arithmetic
Here: Theory of the Reals

Nonlinear Real Arithmetic:

- real variables \( v := x_i \in \mathbb{R} \)
- constants \( c := q \in \mathbb{Z} \)
- terms \( t := v \mid c \mid t + t \mid t \cdot t \)
- atoms \( a := t \sim 0, \sim \in \{<, >, \leq, \geq, =, \neq\} \)

Intuition: polynomials over real variables compared to zero.
Here: Theory of the Reals

Nonlinear Real Arithmetic:

- real variables $v := x_i \in \mathbb{R}$
- constants $c := q \in \mathbb{Z}$
- terms $t := v | c | t + t | t \cdot t$
- atoms $a := t \sim 0$, $\sim \in \{<, >, \leq, \geq, =, \neq\}$

Intuition: polynomials over real variables compared to zero.

Does cover: $t > t$, rational constants, division (encoding with auxiliary variables)
Does not cover: transcendental constants, non-polynomial functions
Here: Theory of the Reals

Nonlinear Real Arithmetic:

- **real variables** $v := x_i \in \mathbb{R}$
- **constants** $c := q \in \mathbb{Z}$
- **terms** $t := v \mid c \mid t + t \mid t \cdot t$
- **atoms** $a := t \sim 0$, $\sim \in \{<,>,\leq,\geq,=,\neq\}$

Intuition: polynomials over real variables compared to zero.

**Does cover:** $t > t$, rational constants, division (encoding with auxiliary variables)

**Does not cover:** transcendental constants, non-polynomial functions

**Linear arithmetic:** essentially a solved problem.

Use Simplex (or sometimes Fourier-Motzkin)
complete (we have decision procedures that are sound and complete)

admits quantifier elimination (quantifiers are conceptually easy)
complete (we have decision procedures that are sound and complete)
admits quantifier elimination (quantifiers are conceptually easy)

Some methods:

- [Tarski 1951] Tarski: first complete method, non-elementary complexity
- [Buchberger 1965] Gröbner bases: limited applicability, standard tool in CA
- [Collins 1974] CAD: complete, doubly exponential complexity
- [Weispfenning 1988] VS: up to bounded degree, singly exponential complexity
- [Fontaine et al. 2017] Subtropical satisfiability: incomplete reduction to LRA
- [Irfan 2018] Linearization: incomplete, axiom instantiation
- [Ábrahám et al. 2021] CDCAC: conflict-driven CAD
- and some more...
Overview

1. SMT for NRA
2. Linearization
3. Interval Constraint Propagation
4. Subtropical Satisfiability
5. Gröbner Bases
6. Virtual Substitution
7. Cylindrical Algebraic Decomposition
8. Conflict-Driven Cylindrical Algebraic Coverings
9. Related topics
Linearization by example

- Linearize atoms
- Solve
- Identify conflicts
- Instantiate axioms
- Add as lemmas
- Repeat

Techniques for NRA in SMT • Linearization

[Irfan 2018] [Cimatti et al. 2018]
Linearization by example

- Linearize atoms
  - \( x \cdot y \leq 0 \land x < 0 \land x + y = 0 \)
  - linearize: \( z \leq 0 \land x < 0 \land x + y = 0 \)  \( z := x \cdot y \)
  - atoms: \( z \leq 0 \land x < 0 \land x + y = 0 \)
  - solve: \( x \mapsto -1, y \mapsto 1, z \mapsto 0 \)
  - conflict: \( 0 \neq -1 \cdot 1 \)
  - axiom: \( z = 0 \Rightarrow (x = 0 \lor y = 0) \)
  - add axiom as lemma, proceed to next theory call

- Solve
  - \( x \cdot y \leq 0 \land x < 0 \land x + y = 0 \)

- Identify conflicts
  - \( 0 \neq -1 \cdot 1 \)

- Instantiate axioms
  - \( z \leq 0 \land x < 0 \land x + y = 0 \land z \neq 0 \)

- Add as lemmas
  - \( x \mapsto -1, y \mapsto 1, z \mapsto -1 \)
  - SAT!
Intuition: **iteratively teach the linear solver** about the nonlinear parts, add lemmas that cut away unsatisfiable regions.
Intuition: *iteratively teach the linear solver* about the nonlinear parts, add lemmas that cut away unsatisfiable regions.

**More axioms:** zeroes, monotonicity, commutativity, symmetry w.r.t. signs, tangent planes, ...
Intuition: *iteratively teach the linear solver* about the nonlinear parts, add lemmas that cut away unsatisfiable regions.

More axioms: zeroes, monotonicity, commutativity, symmetry w.r.t. signs, tangent planes, ... 

Problems: difficult to identify models (linear solver only finds corners), linear solver only finds rational assignments \( x^2 = 2 \)
Linearization

Intuition: iteratively teach the linear solver about the nonlinear parts, add lemmas that cut away unsatisfiable regions.

More axioms: zeroes, monotonicity, commutativity, symmetry w.r.t. signs, tangent planes, ...

Problems: difficult to identify models (linear solver only finds corners), linear solver only finds rational assignments ($x^2 = 2$)

Extensions:
- Repair model (if easily possible)
- Transcendental functions ($\sin, \cos, ...$)
- extended operators in general
Interval Constraint Propagation by example

\[ y > x^2 \land y < -x^2 + 2x \land y \leq 1 - x \]
Interval Constraint Propagation by example

\[ y > x^2 \land y < -x^2 + 2x \land y \leq 1 - x \]

\[ y > x^2 \Rightarrow y \in (0, \infty) \]

\[ x \times y \in (-\infty, \infty) \times (0, \infty) \]
Interval Constraint Propagation by example

\[ y > x^2 \land y < -x^2 + 2x \land y \leq 1 - x \]

\[ y > x^2 \Rightarrow y \in (0, \infty) \]

\[ x > 0.5x^2 + y \Rightarrow x \in (0, \infty) \]

\[ x \times y \]

\[ (-\infty, \infty) \times (0, \infty) \]

\[ (0, \infty) \times (0, \infty) \]
Interval Constraint Propagation by example

\[ y > x^2 \land y < -x^2 + 2x \land y \leq 1 - x \]

\[ y > x^2 \Rightarrow y \in (0, \infty) \]

\[ x > 0.5x^2 + y \Rightarrow x \in (0, \infty) \]

\[ x \leq -y + 1 \Rightarrow x \in (0, 1) \]

\[ x \times y \]

\[ (-\infty, \infty) \times (0, \infty) \]

\[ (0, \infty) \times (0, \infty) \]

\[ (0, 1) \times (0, \infty) \]
$y > x^2 \land y < -x^2 + 2x \land y \leq 1 - x$

$x \times y$

$x \times y$

$x > 0.5x^2 + y \Rightarrow x \in (0, \infty)$

$(0, \infty) \times (0, \infty)$

$x \leq -y + 1 \Rightarrow x \in (0, 1)$

$(0, 1) \times (0, \infty)$

$y \leq -x + 1 \Rightarrow y \in (0, 1)$

$(0, 1) \times (0, 1)$
\[ y > x^2 \land y < -x^2 + 2x \land y \leq 1 - x \]
\[ y > x^2 \implies y \in (0, \infty) \]
\[ x > 0.5x^2 + y \implies x \in (0, \infty) \]
\[ x \leq -y + 1 \implies x \in (0, 1) \]
\[ y \leq -x + 1 \implies y \in (0, 1) \]
\[ \text{guess midpoint} \ (0.5, 0.5) \in (0, 1) \times (0, 1) \]
\[ x \times y \]
\[ (-\infty, \infty) \times (0, \infty) \]
\[ (0, \infty) \times (0, \infty) \]
\[ (0, 1) \times (0, \infty) \]
\[ (0, 1) \times (0, 1) \]
Core idea:

- Maintain interval assignment (that represents the current box)
- Perform over-approximating contractions until
  - the current box is empty (UNSAT),
  - we can guess a model (SAT), or
  - we reach a threshold.
- When reaching a threshold
  - we terminate with unknown or
  - split: $x \in [0, 5] \leadsto (x < 3 \lor x \geq 3)$
Core idea:

- Maintain interval assignment (that represents the current box)
- Perform over-approximating contractions until
  - the current box is empty (UNSAT),
  - we can guess a model (SAT), or
  - we reach a threshold.
- When reaching a threshold
  - we terminate with unknown or
  - split: \( x \in [0, 5] \leadsto (x < 3 \lor x \geq 3) \)

- Incomplete solving procedure
- Used as preprocessor for other techniques [Loup et al. 2013]
- Delicate tuning of heuristics (splitting, thresholds, model guessing)
Core idea: reduce $p = 0$ to a linear problem in the exponents of $p$

- Assume $p(1, \ldots, 1) < 0$ (otherwise consider $-p$)
- Find $x \in \mathbb{R}_+^n$ such that $p(x) > 0$
- Solve $p(y) = 0$ with $y$ on the line $(1, \ldots, 1) - x$
Core idea: reduce $p = 0$ to a linear problem in the exponents of $p$

- Assume $p(1, \ldots, 1) < 0$ (otherwise consider $-p$)
- Find $x \in \mathbb{R}^n_+$ such that $p(x) > 0$
- Solve $p(y) = 0$ with $y$ on the line $(1, \ldots, 1) - x$
- Incomplete (no such $y$ exists, though $p = 0$ has a solution)
Core idea: reduce $p = 0$ to a \textit{linear} problem in the \textit{exponents} of $p$

- Assume $p(1, \ldots, 1) < 0$ (otherwise consider $-p$)
- Find $x \in \mathbb{R}^n_+$ such that $p(x) > 0$
- Solve $p(y) = 0$ with $y$ on the line $(1, \ldots, 1) - x$
- Incomplete (no such $y$ exists, though $p = 0$ has a solution)

Core problem: \textbf{How to find $x \in \mathbb{R}^n_+$ such that $p(x) > 0$?}
Core idea: reduce \( p = 0 \) to a linear problem in the exponents of \( p 

- Assume \( p(1, \ldots, 1) < 0 \) (otherwise consider \( -p \))
- Find \( x \in \mathbb{R}^n_+ \) such that \( p(x) > 0 \)
- Solve \( p(y) = 0 \) with \( y \) on the line \((1, \ldots, 1) \) — \( x \)
- Incomplete (no such \( y \) exists, though \( p = 0 \) has a solution)

Core problem: How to find \( x \in \mathbb{R}^n_+ \) such that \( p(x) > 0 \)?

For \( n = 1 \): \( \lim_{x \to \infty} p(x) = \infty \) if \( \text{lcoeff}(p) > 0 \). Increase \( x \) as necessary.
Subtropical satisfiability

Core idea: reduce \( p = 0 \) to a linear problem in the exponents of \( p \)

- Assume \( p(1, \ldots, 1) < 0 \) (otherwise consider \(-p\))
- Find \( x \in \mathbb{R}^n_+ \) such that \( p(x) > 0 \)
- Solve \( p(y) = 0 \) with \( y \) on the line \((1, \ldots, 1) - x\)
- Incomplete (no such \( y \) exists, though \( p = 0 \) has a solution)

Core problem: How to find \( x \in \mathbb{R}^n_+ \) such that \( p(x) > 0 \)?

For \( n = 1 \): \( \lim_{x \to \infty} p(x) = \infty \) if \( \text{lcoeff}(p) > 0 \). Increase \( x \) as necessary.
For \( n \geq 2 \): search direction in exponent space such that the largest exponent in this direction is positive. Increase \( x \) in this direction as necessary.
Core idea: reduce $p = 0$ to a linear problem in the exponents of $p$

- Assume $p(1, \ldots, 1) < 0$ (otherwise consider $-p$)
- Find $x \in \mathbb{R}^n_+$ such that $p(x) > 0$
- Solve $p(y) = 0$ with $y$ on the line $(1, \ldots, 1) - x$
- Incomplete (no such $y$ exists, though $p = 0$ has a solution)

Core problem: How to find $x \in \mathbb{R}^n_+$ such that $p(x) > 0$?

For $n = 1$: $\lim_{x \to \infty} p(x) = \infty$ if $l\text{coeff}(p) > 0$. Increase $x$ as necessary.
For $n \geq 2$: search direction in exponent space such that the largest exponent in this direction is positive. Increase $x$ in this direction as necessary.

$$p = -y + 2xy^3 - 3x^2y^2 - x^3 - 4x^3y^3$$
Core idea: reduce $p = 0$ to a linear problem in the exponents of $p$

- Assume $p(1, \ldots, 1) < 0$ (otherwise consider $-p$)
- Find $x \in \mathbb{R}_+^n$ such that $p(x) > 0$
- Solve $p(y) = 0$ with $y$ on the line $(1, \ldots, 1) - x$
- Incomplete (no such $y$ exists, though $p = 0$ has a solution)

Core problem: How to find $x \in \mathbb{R}_+^n$ such that $p(x) > 0$?

For $n = 1$: $\lim_{x \to \infty} p(x) = \infty$ if $\text{lcoeff}(p) > 0$. Increase $x$ as necessary.

For $n \geq 2$: search direction in exponent space such that the largest exponent in this direction is positive. Increase $x$ in this direction as necessary.

$$p = -y + 2xy^3 - 3x^2y^2 - x^3 - 4x^3y^3$$

Find hyperplane that separates a positive node
Core idea: reduce $p = 0$ to a linear problem in the exponents of $p$

- Assume $p(1, \ldots, 1) < 0$ (otherwise consider $-p$)
- Find $x \in \mathbb{R}_+^n$ such that $p(x) > 0$
- Solve $p(y) = 0$ with $y$ on the line $(1, \ldots, 1) - x$
- Incomplete (no such $y$ exists, though $p = 0$ has a solution)

Core problem: How to find $x \in \mathbb{R}_+^n$ such that $p(x) > 0$?

For $n = 1$: $\lim_{x \to \infty} p(x) = \infty$ if $\text{lcoeff}(p) > 0$. Increase $x$ as necessary.

For $n \geq 2$: search direction in exponent space such that the largest exponent in this direction is positive. Increase $x$ in this direction as necessary.

$$p = -y + 2xy^3 - 3x^2y^2 - x^3 - 4x^3y^3$$

Find hyperplane that separates a positive node

Encoding in QF_LRA
Growing degree only impacts coefficient size
Canonical generators for a polynomial ideal

For us: Normal form for sets of polynomials

Maintains set of common complex roots

The workhorse of computer algebra for polynomial equalities

Mature implementations (every CAS)

Doubly exponential in worst case, but usually much faster.
Gröbner basis

- Canonical generators for a polynomial ideal
- For us: Normal form for sets of polynomials
- Maintains set of common complex roots
- The workhorse of computer algebra for polynomial equalities
- Mature implementations (every CAS)
- Doubly exponential in worst case, but usually much faster.

Relevant for SMT: $\exists x \in \mathbb{C}^n. p(x) = 0$
Gröbner basis

- Canonical generators for a polynomial ideal
- For us: Normal form for sets of polynomials
- Maintains set of common complex roots
- The workhorse of computer algebra for polynomial equalities
- Mature implementations (every CAS)
- Doubly exponential in worst case, but usually much faster.

Relevant for SMT: $\exists x \in \mathbb{C}^n. p(x) = 0$

But: What about inequalities? How to go from $\mathbb{C}$ to $\mathbb{R}$?
see [Junges 2012] for some approaches.
Core idea:
- Use **solution formula** to solve polynomial equation for $x$
- **Substitute value** for $x$ into remaining equations
- Repeat for remaining variables
Core idea:

- Use **solution formula** to solve polynomial equation for $x$
- **Substitute value** for $x$ into remaining equations
- Repeat for remaining variables

What about **inequalities**?

- Construct test candidates for all **sign-invariant** regions in $x$
- Always try the roots and the **smallest values of the intermediate intervals**

- Introduces special terms $t + \varepsilon$ and $-\infty$
Algorithmic core: a collection of substitution rules

**Example:** Substitute $e + \varepsilon$ for $x$ into $a \cdot x^2 + b \cdot x + c > 0$:

\[
( (ax^2 + bx + c > 0)[e//x] )
\]

\[
\lor ( (ax^2 + bx + c = 0)[e//x] \land (2ax + b > 0)[e//x] )
\]

\[
\lor ( (ax^2 + bx + c = 0)[e//x] \land (2ax + b = 0)[e//x] \land (2a > 0)[e//x] )
\]
Algorithmic core: a collection of substitution rules

Example: Substitute $e + \varepsilon$ for $x$ into $a \cdot x^2 + b \cdot x + c > 0$:

\[
\begin{align*}
\lor & \quad \left( (ax^2 + bx + c > 0)[e//x] \right) \\
\lor & \quad \left( (ax^2 + bx + c = 0)[e//x] \land (2ax + b > 0)[e//x] \right) \\
\lor & \quad \left( (ax^2 + bx + c = 0)[e//x] \land (2ax + b = 0)[e//x] \land (2a > 0)[e//x] \right)
\end{align*}
\]

Not always applicable:

- Solution formulas only exist up to degree four
- The above rule may introduce a degree growth
- Efficient if applicable
- [Košta et al. 2015] uses FO formulas, allows arbitrary but fixed degrees (needs precomputed substitution rules obtained by quantifier elimination)
The core idea: sign-invariance (or rather truth-table equivalence)

\[ \text{sgn}(p(a)) = \text{sgn}(p(b)) \quad \forall p \in \varphi \quad \Rightarrow \quad \varphi(a) = \varphi(b) \]

For our purpose, \(a\) and \(b\) are equivalent!
The core idea: sign-invariance (or rather truth-table equivalence)

$$\text{sgn}(p(a)) = \text{sgn}(p(b)) \forall p \in \varphi \implies \varphi(a) = \varphi(b)$$

For our purpose, \(a\) and \(b\) are equivalent!

Construct a sign-invariant decomposition of \(\mathbb{R}^n\):

$$\text{cell } C \subset \mathbb{R}^n : \forall a, b \in C : \varphi(a) = \varphi(b)$$

Abstraction: \(\mathbb{R}^n\) to finite set of cells, consider a single \(a \in C\) per cell.
The core idea: sign-invariance (or rather truth-table equivalence)

\[ \text{sgn}(p(a)) = \text{sgn}(p(b)) \quad \forall p \in \varphi \quad \Rightarrow \quad \varphi(a) = \varphi(b) \]

For our purpose, \( a \) and \( b \) are equivalent!

Construct a sign-invariant decomposition of \( \mathbb{R}^n \):

\[ \text{cell } C \subset \mathbb{R}^n : \forall a, b \in C : \varphi(a) = \varphi(b) \]

Abstraction: \( \mathbb{R}^n \) to finite set of cells, consider a single \( a \in C \) per cell.

\[ \varphi = (p > 0) \land (q < 0) \]

Solution space
The core idea: sign-invariance (or rather truth-table equivalence)

$$\text{sgn}(p(a)) = \text{sgn}(p(b)) \ \forall p \in \varphi \ \Rightarrow \ \varphi(a) = \varphi(b)$$

For our purpose, $a$ and $b$ are equivalent!

Construct a sign-invariant decomposition of $\mathbb{R}^n$:

$$\text{cell } C \subset \mathbb{R}^n : \forall a, b \in C : \varphi(a) = \varphi(b)$$

Abstraction: $\mathbb{R}^n$ to finite set of cells, consider a single $a \in C$ per cell.
The core idea: **sign-invariance** (or rather truth-table equivalence)

\[ \text{sgn}(p(a)) = \text{sgn}(p(b)) \quad \forall p \in \varphi \quad \Rightarrow \quad \varphi(a) = \varphi(b) \]

For our purpose, \( a \) and \( b \) are equivalent!

Construct a **sign-invariant decomposition** of \( \mathbb{R}^n \):

\[ \text{cell } C \subset \mathbb{R}^n : \forall a, b \in C : \varphi(a) = \varphi(b) \]

**Abstraction**: \( \mathbb{R}^n \) to finite set of cells, consider a single \( a \in C \) per cell.

\[ \varphi = (p > 0) \land (q < 0) \]

Sample points
The core idea: **sign-invariance** (or rather truth-table equivalence)

\[ \text{sgn}(p(a)) = \text{sgn}(p(b)) \quad \forall p \in \varphi \quad \Rightarrow \quad \varphi(a) = \varphi(b) \]

For our purpose, \( a \) and \( b \) are equivalent!

Construct a **sign-invariant decomposition** of \( \mathbb{R}^n \):

\[ \text{cell } C \subset \mathbb{R}^n : \forall a, b \in C : \varphi(a) = \varphi(b) \]

**Abstraction**: \( \mathbb{R}^n \) to finite set of cells, consider a single \( a \in C \) per cell.

\[ \varphi = (p > 0) \land (q < 0) \]

Actual sample points

Arranged in **cylinders**
Proceed dimension-wise: project to lower-dimensional problem, lift results.
Proceed dimension-wise: project to lower-dimensional problem, lift results.

Intuition

Critical points
Proceed dimension-wise: project to lower-dimensional problem, lift results.

Intuition

Critical points

Project sample
Proceed *dimension-wise*: project to lower-dimensional problem, lift results.

**Intuition**

- Critical points
- Project sample
- Solve 1-dim
Proceed \textit{dimension-wise}: project to lower-dimensional problem, lift results.

\begin{itemize}
  \item Intuition
  \item Critical points
  \item Project sample
  \item Solve 1-dim
  \item Lift to 2-dim
\end{itemize}
Cylindrical Algebraic Decomposition in $\mathbb{R}^2$

Proceed **dimension-wise**: project to lower-dimensional problem, lift results.

**Intuition**
- Critical points
- Project sample
- Solve 1-dim
- Lift to 2-dim

**Implementation**
Cylindrical Algebraic Decomposition in $\mathbb{R}^2$

Proceed dimension-wise: project to lower-dimensional problem, lift results.

**Intuition**
- Critical points
- Project sample
- Solve 1-dim
- Lift to 2-dim

**Implementation**
- Project polynomials

resultant
Proceed dimension-wise: project to lower-dimensional problem, lift results.

**Intuition**
- Critical points
- Project sample
- Solve 1-dim
- Lift to 2-dim

**Implementation**
- Project polynomials
- Solve 1-dim

*Cylindrical Algebraic Decomposition in $\mathbb{R}^2$*
Proceed dimension-wise: project to lower-dimensional problem, lift results.

Intuition

Critical points
Project sample
Solve 1-dim
Lift to 2-dim

Implementation

Project polynomials
Solve 1-dim
Lift to 2-dim

resultant
Cylindrical Algebraic Decomposition in $\mathbb{R}^n$

Theory atoms $\rightarrow P_n \subset \mathbb{Z}[x_1..x_n]$

$P_{n-1} \subset \mathbb{Z}[x_1..x_{n-1}]$

$\vdots$

$P_1 \subset \mathbb{Z}[x_1]$

$\rightarrow S_n \subset S_{n-1} \times \mathbb{R}$

$\vdots$

$S_2 \subset S_1 \times \mathbb{R}$

$\rightarrow S_1 \subset \mathbb{R}$

Solutions
Cylindrical Algebraic Decomposition in $\mathbb{R}^n$

Theory atoms $\rightarrow P_n \subset \mathbb{Z}[x_1..x_n] \rightarrow P_{n-1} \subset \mathbb{Z}[x_1..x_{n-1}] \rightarrow \ldots \rightarrow P_1 \subset \mathbb{Z}[x_1]$

$Proj(P_k) \rightarrow \ldots \rightarrow S_1 \subset \mathbb{R}$

$S_n \subset S_{n-1} \times \mathbb{R} \rightarrow \ldots \rightarrow S_2 \subset S_1 \times \mathbb{R} \rightarrow \text{Solutions}$

$Lift(P_k, S_{k-1})$
Cylindrical Algebraic Decomposition in $\mathbb{R}^n$

Theory atoms $\rightarrow$ $P_n \subseteq \mathbb{Z}[x_1..x_n]$

$P_n \subseteq \mathbb{Z}[x_1..x_n] \rightarrow S_n \subseteq S_{n-1} \times \mathbb{R}$

$P_{n-1} \subseteq \mathbb{Z}[x_1..x_{n-1}] \rightarrow L(\text{Proj}(P_k), S_{k-1})$

$P_1 \subseteq \mathbb{Z}[x_1] \rightarrow S_1 \subseteq \mathbb{R}$

Projection:

- Intersections (resultants)
- Flipping points (discriminants)
- Singularities (coefficients)
Cylindrical Algebraic Decomposition in $\mathbb{R}^n$

Theory atoms $P_n \subset \mathbb{Z}[x_1..x_n]$ → Solutions $S_n \subset S_{n-1} \times \mathbb{R}$

$P_{n-1} \subset \mathbb{Z}[x_1..x_{n-1}]$ → $S_{n-1} \subset S_{n-2} \times \mathbb{R}$

$\vdots$

$P_1 \subset \mathbb{Z}[x_1]$ → $S_1 \subset \mathbb{R}$

Projection:
- Intersections (resultants)
- Flipping points (discriminants)
- Singularities (coefficients)

Lifting:
- Substitution $s \in S_k$, $p \in P_{k+1}$
  $p(s) \rightarrow p' \in \mathbb{Z}[x_{k+1}]^{**}$
- Isolate real roots of $p'$
Final notes on CAD

- Asymptotic complexity: \((n \cdot m)^{2^r}\) (\(r\) variables, \(m\) polynomials of degree \(n\))
- Oftentimes way faster, but worst-case occurs in practice!
- Best complete method that is known and implemented. [Hong 1991]
- Active research:
  - Lifting [Collins 1974] [Lazard 1994] [McCallum et al. 2016] [McCallum et al. 2019]
  - Equational constraints [Collins 1998] [McCallum 1999] [McCallum 2001] [England et al. 2015] [Haehn et al. 2018] [Nair et al. 2019]
  - Variable ordering [England et al. 2014] [Huang et al. 2014] [Nalbach et al. 2019] [Florescu et al. 2019]
  - Adaptions [Jovanović et al. 2012] [Brown 2013] [Brown 2015] [Ábrahám et al. 2021]
- Implementation needs groundwork: polynomial computation (resultants, multivariate gcd, optionally multivariate factorization), real algebraic numbers (representation, multivariate root isolation)
Core idea: use CAD techniques in a conflict-driven way.

My intuition: MCSAT turned into a theory solver.
Core idea: use **CAD techniques** in a **conflict-driven** way.
My intuition: MCSAT turned into a theory solver.

- Fix a **variable ordering**
- For the $k$th variable
  - Use constraints to **exclude unsatisfiable intervals**
  - **Guess** a value for the $k$th variable
  - Recurse to $k + 1$st variable and obtain
    - a **full variable assignment** ($\rightarrow$ return SAT)
    - or a **covering** for the $k + 1$st variable
  - Use **CAD machinery** to infer an interval for the $k$th variable
- Until the collected intervals form a **covering** for the $k$th variable
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]

No constraint for \( x \)
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]

No constraint for \( x \)

Guess \( x \mapsto 0 \)
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]

No constraint for \( x \)

Guess \( x \mapsto 0 \)

\( c_1 \rightarrow y \notin (-1, \infty) \)
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]

No constraint for \( x \)

Guess \( x \mapsto 0 \)

- \( c_1 \rightarrow y \notin (-1, \infty) \)
- \( c_2 \rightarrow y \notin (-\infty, 0.75) \)
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]

No constraint for \( x \)

Guess \( x \mapsto 0 \)

\[ c_1 \rightarrow y \notin (-1, \infty) \]
\[ c_2 \rightarrow y \notin (-\infty, 0.75) \]
\[ c_3 \rightarrow y \notin (-\infty, 0.5) \]
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]

No constraint for \( x \)

Guess \( x \mapsto 0 \)

- \( c_1 \rightarrow y \notin (-1, \infty) \)
- \( c_2 \rightarrow y \notin (-\infty, 0.75) \)
- \( c_3 \rightarrow y \notin (-\infty, 0.5) \)

Construct covering \((-\infty, 0.5), (-1, \infty)\)
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]

No constraint for \( x \)
Guess \( x \mapsto 0 \)
\[ c_1 \rightarrow y \notin (-1, \infty) \]
\[ c_2 \rightarrow y \notin (-\infty, 0.75) \]
\[ c_3 \rightarrow y \notin (-\infty, 0.5) \]
Construct covering \((-\infty, 0.5), (-1, \infty)\)
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]

No constraint for \( x \)

Guess \( x \mapsto 0 \)

- \( c_1 \rightarrow y \notin (-1, \infty) \)
- \( c_2 \rightarrow y \notin (-\infty, 0.75) \)
- \( c_3 \rightarrow y \notin (-\infty, 0.5) \)

Construct covering \((-\infty, 0.5), (-1, \infty)\)

Construct interval for \( x \)

\( x \notin (-2, 3) \)
An example

\[ c_1 : 4 \cdot y < x^2 - 4 \quad c_2 : 4 \cdot y > 4 - (x - 1)^2 \quad c_3 : 4 \cdot y > x + 2 \]

No constraint for \(x\)

Guess \(x \mapsto 0\)

- \(c_1 \rightarrow y \notin (-1, \infty)\)
- \(c_2 \rightarrow y \notin (-\infty, 0.75)\)
- \(c_3 \rightarrow y \notin (-\infty, 0.5)\)

Construct covering \((-\infty, 0.5), (-1, \infty)\)

Construct interval for \(x\)

- \(x \notin (-2, 3)\)

New guess for \(x\)
The main algorithm

```python
function get_unsat_cover((s_1, ..., s_{i-1}))

\[ I := \text{get_unsat_intervals}(s) \]

while \( \bigcup_{I \in I} I \neq \mathbb{R} \) do

\[ s_i := \text{sample_outside}(I) \]

if \( i = n \) then return (SAT, (s_1, ..., s_{i-1}, s_i))

\( (f, O) := \text{get_unsat_cover}((s_1, ..., s_{i-1}, s_i)) \)

if \( f = \text{SAT} \) then return (SAT, O)

else if \( f = \text{UNSAT} \) then

\[ R := \text{construct_characterization}((s_1, ..., s_{i-1}, s_i), O) \]

\[ J := \text{interval_from_characterization}((s_1, ..., s_{i-1}), s_i, R) \]

\[ I := I \cup \{J\} \]

end

end

return (UNSAT, I)
```
The main algorithm

function get_unsat_cover((s₁,...,sᵢ₋₁))

\[ \mathbb{I} := \text{get_unsat_intervals}(s) \]

while \( \bigcup_{I \in \mathbb{I}} I \neq \mathbb{R} \) do

\[ sᵢ := \text{sample_outside}(\mathbb{I}) \]

if \( i = n \) then return \((\text{SAT}, (s₁,...,sᵢ₋₁,sᵢ))\)

\[(f, O) := \text{get_unsat_cover}((s₁,...,sᵢ₋₁,sᵢ))\]

if \( f = \text{SAT} \) then return \((\text{SAT}, O)\)

else if \( f = \text{UNSAT} \) then

\[ R := \text{construct_characterization}((s₁,...,sᵢ₋₁,sᵢ), O) \]

\[ J := \text{interval_from_characterization}((s₁,...,sᵢ₋₁), sᵢ, R) \]

\[ \mathbb{I} := \mathbb{I} \cup \{ J \} \]

end

end

return \((\text{UNSAT}, \mathbb{I})\)
function get_unsat_cover((s_1, \ldots, s_{i-1}))

\[ I := \text{get_unsat_intervals}(s) \]

while \( \bigcup_{I \in I} I \neq \mathbb{R} \) do

\[ s_i := \text{sample_outside}(I) \]

if \( i = n \) then return (SAT, (s_1, \ldots, s_{i-1}, s_i))

\( (f, O) := \text{get_unsat_cover}((s_1, \ldots, s_{i-1}, s_i)) \)

if \( f = \text{SAT} \) then return (SAT, O)
else if \( f = \text{UNSAT} \) then

\[ R := \text{construct_characterization}((s_1, \ldots, s_{i-1}, s_i), O) \]

\[ J := \text{interval_from_characterization}((s_1, \ldots, s_{i-1}), s_i, R) \]

\[ I := I \cup \{J\} \]

end

end

return (UNSAT, I)
The main algorithm

function get_unsat_cover((s_1, \ldots, s_{i-1}))

\[ \Pi := \text{get_unsat_intervals}(s) \]

while \( \bigcup_{I \in \Pi} I \neq \mathbb{R} \) do

\[ s_i := \text{sample_outside}(\Pi) \]

if \( i = n \) then return \( (\text{SAT}, (s_1, \ldots, s_{i-1}, s_i)) \)

\((f, O) := \text{get_unsat_cover}((s_1, \ldots, s_{i-1}, s_i))\)

if \( f = \text{SAT} \) then return \( (\text{SAT}, O) \)

else if \( f = \text{UNSAT} \) then

\[ R := \text{construct_characterization}((s_1, \ldots, s_{i-1}, s_i), O) \]

\[ J := \text{interval_from_characterization}((s_1, \ldots, s_{i-1}, s_i), R) \]

\[ \Pi := \Pi \cup \{J\} \]

end

end

return \( (\text{UNSAT}, \Pi) \)
The main algorithm

function get_unsat_cover((s₁,...,sᵢ−₁))

II := get_unsat_intervals(s)
while \( \bigcup_{I \in II} I \neq \mathbb{R} \) do
  \( s_i := \text{sample_outside}(II) \)
  if \( i = n \) then return (SAT, (s₁,...,sᵢ−₁, sᵢ))
  \((f, O) := \text{get_unsat_cover}((s₁,...,sᵢ−₁, sᵢ))\)
  if \( f = \text{SAT} \) then return (SAT, O)
  else if \( f = \text{UNSAT} \) then
    \( R := \text{construct_characterization}((s₁,...,sᵢ−₁, sᵢ)) \)
    \( J := \text{interval_from_characterization}((s₁,...,sᵢ−₁), sᵢ, R) \)
    II := II ∪ \{J\}
  end
end
return (UNSAT, II)
function get_unsat_cover((s₁, ..., sᵢ−₁))

\[ \mathbb{I} := \text{get_unsat_intervals}(s) \]

while \( \bigcup_{I \in \mathbb{I}} I \neq \mathbb{R} \) do

\[ sᵢ := \text{sample_outside}(\mathbb{I}) \]

if \( i = n \) then return \((\text{SAT}, (s₁, ..., sᵢ−₁, sᵢ))\)

\((f, O) := \text{get_unsat_cover}((s₁, ..., sᵢ−₁, sᵢ))\)

if \( f = \text{SAT} \) then return \((\text{SAT}, O)\)

else if \( f = \text{UNSAT} \) then

\[ R := \text{construct_characterization}((s₁, ..., sᵢ−₁, sᵢ)) \]

\[ J := \text{interval_from_characterization}((s₁, ..., sᵢ−₁, sᵢ)) \]

\[ \mathbb{I} := \mathbb{I} \cup \{J\} \]

end

end

return \((\text{UNSAT}, \mathbb{I})\)
The main algorithm

function get_unsat_cover((s₁,…,sᵢ−₁))

\[ \mathbb{I} := \text{get_unsat_intervals}(s) \]

while \( \bigcup_{I \in \mathbb{I}} I \neq \mathbb{R} \) do

\[ s_i := \text{sample_outside}(\mathbb{I}) \]

if \( i = n \) then return \((\text{SAT}, (s_1, \ldots, s_{i−1}, s_i))\)

\( (f, O) := \text{get_unsat_cover}((s_1, \ldots, s_{i−1}, s_i)) \)

if \( f = \text{SAT} \) then return \((\text{SAT}, O)\)

else if \( f = \text{UNSAT} \) then

\( R := \text{construct_characterization}((s_1, \ldots, s_{i−1}, s_i)) \)

\( J := \text{interval_from_characterization}((s_1, \ldots, s_{i−1}, s_i), R) \)

\( \mathbb{I} := \mathbb{I} \cup \{J\} \)

end

end

return \((\text{UNSAT}, \mathbb{I})\)
construct_characterization

Identify region around sample
construct_characterization

Identify region around sample
construct_characterization

Identify region around sample CAD projection:

Discriminants (and coefficients)
Resultants
Identify region around sample CAD projection:

- Discriminants (and coefficients)
- Resultants
Identify region around sample CAD projection:

Discriminants (and coefficients)

Resultants
Identify region around sample CAD projection:

Discriminants (and coefficients)

Resultants

Improvement over CAD:

Resultants between neighbouring intervals only!
Numerical methods [Kremer 2013]:
focus on good approximation, but no formal guarantees

Tarski’s method [Tarski 1951]:
theoretical breakthrough only, non-elementary complexity

Grigor’ev and Vorobjov [Grigor’ev et al. 1988], Renegar [Renegar 1988]:
singly exponential, but impractical (see [Hong 1991])

Basu, Pollack and Roy [Basu et al. 1996]:
“realizable sign conditions”, has not been implemented (yet)

Other CAD-based methods:
Regular Chains [Chen et al. 2009], NuCAD [Brown 2015]
Quantifiers:
- Theory of the Reals admits quantifier elimination
- CAD constructs $\varphi'$ for $Q_x \varphi(x, y) \iff \varphi'(y)$

Theory combination with Array, BV, FP, String, ... [Nelson et al. 1979]

Transcendentals: extend linearization [Cimatti et al. 2018] [Irfan 2018]

Optimization: CAD can optimize for an objective [Kremer 2020]

Integers: Branch&Bound complements BitBlasting [Kremer et al. 2016]
Other approaches for (QF_)NRA:

- **MCSAT / NLSAT:**
  - Theory model construction integrated in the core solver
  - SMT-RAT, yices, z3 [Jovanović et al. 2012] [Jovanović et al. 2013] [Moura et al. 2013]
    [Nalbach et al. 2019] [Kremer 2020]

- **CAD is a stand-alone tool:**
  - Maple / RegularChains [Chen et al. 2009]
  - Mathematica [Strzeboński 2014]
  - QEPCAD B [Brown 2003]
  - Redlog / Reduce [Dolzmann et al. 1997]

These can be **integrated as theory solvers** [Fontaine et al. 2018] [Kremer 2018]
cvc5

- SMT solver developed at Stanford University & University of Iowa
- Supports a wide variety of theories (and their combinations)
  Arithmetic (linear, non-linear, transcendentals), Arrays, Bags & Sets,
  Bit-vectors, Datatypes, Floating-point, Separation logic, Strings,
  Uninterpreted functions
- Also Quantifiers, Syntax-Guided Synthesis [Reynolds et al. 2019], UNSAT Cores,
  verifiable Proofs
cvc5

- SMT solver developed at Stanford University & University of Iowa
- Supports a wide variety of theories (and their combinations)
  Arithmetic (linear, non-linear, transcendentals), Arrays, Bags & Sets,
  Bit-vectors, Datatypes, Floating-point, Separation logic, Strings,
  Uninterpreted functions
- Also Quantifiers, Syntax-Guided Synthesis [Reynolds et al. 2019], UNSAT Cores,
  verifiable Proofs

obtain cvc5 from
  https://cvc4.github.io/downloads.html or
  https://github.com/cvc5/cvc5
cvc5 for QF_NRA

- Linearization (--nl-ext)
- CDCAC (--nl-cad)
- Also: ICP-style propagations (--nl-icp)
cvc5 for QF_NRA

- Linearization (--nl-ext)
- CDCAC (--nl-cad)
- Also: ICP-style propagations (--nl-icp)

Default strategy: incremental linearization with a small subset of the axioms and CDCAC
cvc5 for QF_NRA

- Linearization (--nl-ext)
- CDCAC (--nl-cad)
- Also: ICP-style propagations (--nl-icp)

Default strategy: incremental linearization with a small subset of the axioms and CDCAC

<table>
<thead>
<tr>
<th>Experiments on QF_NRA (11489 in total)</th>
<th>solved</th>
<th>sat</th>
<th>unsat</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvc5</td>
<td>10634</td>
<td>5001</td>
<td>5633</td>
</tr>
<tr>
<td>yices 2.6.2</td>
<td>10341</td>
<td>4904</td>
<td>5437</td>
</tr>
<tr>
<td>z3 4.8.10</td>
<td>10288</td>
<td>5093</td>
<td>5195</td>
</tr>
</tbody>
</table>
in progress / future work:
- Better integration of Linearization, CDCAC and ICP
- Preprocessing for nonlinear arithmetic
- Improve proofs
- Improve incrementality (in particular CDCAC)
- Improvements within CDCAC (heuristics, factorization, ... )
References I


References II


References IV


References V

References VI


References VII

References VIII


