Incremental CAD
Making CAD work for SMT solving
based on [KA18]
Satisfiability Checking and Symbolic Computation

EU project to stimulate cooperations
More than 50 partners and associates

Industry: Altran, BTC, ClearSy, Imandra, L4B, Maplesoft, Microsoft, MJC2, NAG, SRI, Systerel, Wolfram

Also at ICMS: Erika Ábrahám, James Davenport, Matthew England, Stephen Forrest, Xiao-Shan Gao, Jürgen Gerhard, Jan Horacek, Martin Kreuzer, Alexei Lisitsa, Thomas Sturm
Satisfiability Modulo Theories (SMT)

Is an existentially quantified \( \text{first-order} \) formula \( \varphi \) \text{satisfiable}?

\[ \exists x. \varphi(x) \equiv true \]
Satisfiability Modulo Theories (SMT)

Is an existentially quantified first-order formula $\phi$ satisfiable?

$$\exists x. \phi(x) \equiv true$$

Applications:

- Software verification, test-case generation
- Termination proving
- Controller synthesis
- Scheduling and planning
- Product design automation
- And growing ...
SMT solving

\[ \varphi \]

SAT solver

SAT or UNSAT

SAT + witness

or

UNSAT + reason

theory constraints

CAD
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]

\[ SAT \text{ or } UNSAT \]

\[ SAT + \text{ witness} \]

\[ UNSAT + \text{ reason} \]
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]

\{x > 0, x^2 > 0\}

SAT solver → SAT or UNSAT

CAD → SAT + witness

or

UNSAT + reason
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]

\{x > 0, x^2 > 0\} \rightarrow \text{SAT} + x \leftrightarrow 1

\text{SAT solver} \rightarrow \text{SAT or UNSAT}

\text{CAD}
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]

SAT solver \rightarrow SAT or UNSAT

\{ x > 0, x^2 > 0, x^3 < 0 \} \rightarrow SAT + x \leftrightarrow 1
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \]

\{x > 0, x^2 > 0, x^3 < 0\} \quad UNSAT + \{x > 0, x^3 < 0\}
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0) \]

\{x > 0, x^2 > 0, x^3 < 0\}  \quad \text{UNSAT} + \{x > 0, x^3 < 0\}

SAT solver  \quad \text{SAT or UNSAT}

CAD
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0) \]

\{ x > 0, \neg x^3 < 0, x = 3 \} \quad \text{UNSAT} + \{ x > 0, x^3 < 0 \}
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0) \]

\[ \{ x > 0, \neg x^3 < 0, x = 3 \} \]

SAT + \( x \mapsto 3 \)

SAT solver → SAT or UNSAT

CAD → SAT solver

Incremental CAD: Making CAD work for SMT solving
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0) \]

\[ \{ x > 0, \neg x^3 < 0, x = 3, x^2 > 0 \} \quad \text{SAT} + x \leftrightarrow 3 \]

SAT solver \rightarrow \text{SAT or UNSAT}
SMT solving

\[ x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0) \]

\{x > 0, \neg x^3 < 0, x = 3, x^2 > 0\} \quad \text{SAT} + x \leftrightarrow 3

SAT solver \quad \text{SAT, } x \leftrightarrow 3

CAD
$x > 0 \land (x^2 > 0 \lor x < 0) \land (x^3 < 0 \lor x = 3) \land (\neg x > 0 \lor \neg x^3 < 0)$

SAT solver

SAT, $x \mapsto 3$

$\{x > 0, \neg x^3 < 0, x = 3, x^2 > 0\}$

SAT + $x \mapsto 3$
Cylindrical Algebraic Decomposition

Constraints

\[ P_n \subset \mathbb{Z}[x_1 \ldots x_n] \]

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

\[ \vdots \]

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

Knowledge

\[ Z_n \subset Z_{n-1} \times \mathbb{R} \]

\[ \vdots \]

\[ Z_2 \subset Z_1 \times \mathbb{R} \]

\[ Z_1 \subset \mathbb{R} \]

Note: We only deal with the purely existential case!
Cylindrical Algebraic Decomposition

Constraints

\[ P_n \subset \mathbb{Z}[x_1..x_n] \]
\[ P_{n-1} \subset \mathbb{Z}[x_1..x_{n-1}] \]
\[ \vdots \]
\[ P_1 \subset \mathbb{Z}[x_1] \]

Solutions

\[ Z_n \subset Z_{n-1} \times \mathbb{R} \]
\[ \vdots \]
\[ Z_2 \subset Z_1 \times \mathbb{R} \]
\[ Z_1 \subset \mathbb{R} \]

Note: We only deal with the purely existential case!
Cylindrical Algebraic Decomposition

Constraints

\[ P_n \subseteq \mathbb{Z}[x_1 \ldots x_n] \]

\[ P_{n-1} \subseteq \mathbb{Z}[x_1 \ldots x_{n-1}] \]

\[ \vdots \]

\[ P_1 \subseteq \mathbb{Z}[x_1] \]

Solutions

\[ Z_n \subseteq Z_{n-1} \times \mathbb{R} \]

\[ Z_2 \subseteq Z_1 \times \mathbb{R} \]

\[ \vdots \]

\[ \text{Lift}(P_k, Z_{k-1}) \]

\[ Z_1 \subseteq \mathbb{R} \]

Note: We only deal with the purely existential case!
SMT compliancy

- *Stop early* when a satisfying witness is found
- *Add constraints* and check again
- *Remove constraints*
- *Provide reason for unsatisfiability*
SMT compliancy

- Stop early when a satisfying witness is found
  - How to get there fast?
  - How to keep a consistent state?
- Add constraints and check again

- Remove constraints

- Provide reason for unsatisfiability
SMT compliancy

- **Stop early** when a satisfying witness is found
  - How to get there **fast**?
  - How to keep a **consistent state**?
- **Add constraints** and check again
  - How to **extend projection and lifting** dynamically?
  - How to **retain as much information as possible**?
- **Remove constraints**

- Provide **reason for unsatisfiability**
SMT compliancy

- **Stop early** when a satisfying witness is found
  How to get there fast?
  How to keep a consistent state?

- **Add constraints** and check again
  How to extend projection and lifting dynamically?
  How to retain as much information as possible?

- **Remove constraints**
  How to remove some of the polynomials and samples?
  How to throw away as little information as possible?

- Provide reason for unsatisfiability
Stop early when a satisfying witness is found
How to get there fast?
How to keep a consistent state?

Add constraints and check again
How to extend projection and lifting dynamically?
How to retain as much information as possible?

Remove constraints
How to remove some of the polynomials and samples?
How to throw away as little information as possible?

Provide reason for unsatisfiability
Which constraints reject all the samples?
Solved in [JDF15], though implementation differs.
Taking a step back

What is the purpose of CAD for us?
Extract $P_n$ from constraints

For $k = n \ldots 2$: $P_{n-1} = \text{Proj}(P_n)$

Sample $Z_1$ from $P_1$

For $k = 2 \ldots n$: $Z_k = \text{Lift}(P_n, Z_{n-1})$

Extract solutions from $Z_n$
CAD – Traditional approach [Col75]

- Extract $P_n$ from constraints
- For $k = n \ldots 2$: $P_{n-1} = \text{Proj}(P_n)$
- Sample $Z_1$ from $P_1$
- For $k = 2 \ldots n$: $Z_k = \text{Lift}(P_n, Z_{n-1})$
- Extract solutions from $Z_n$

- We compute all polynomials
- We compute all sample points
- We have no idea how to add or remove constraints
Observations:

- Every $s \in Z_{k-1}$ can be lifted separately.
- Every $s \in Z_{k-1}$ induces a separate set $Z^s_k \subseteq Z_k$. 

Partial CAD [CH91]
Partial CAD [CH91]

Observations:

- Every \( s \in \mathbb{Z}_{k-1} \) can be lifted separately.
- Every \( s \in \mathbb{Z}_{k-1} \) induces a separate set \( \mathbb{Z}^s_k \subseteq \mathbb{Z}_k \).

Essential idea

Consider the lifting to be a tree of sample points.
Explore the tree recursively.
Evaluate partial sample points during exploration.
Eagerly propagate evaluation results and skip redundant sample points.
Observations:

- Every $s \in Z_{k-1}$ can be lifted separately.
- Every $s \in Z_{k-1}$ induces a separate set $Z^s_k \subseteq Z_k$.

Essential idea

Consider the lifting to be a tree of sample points.
 Explore the tree recursively.
Evaluate partial sample points during exploration.
Eagerly propagate evaluation results and skip redundant sample points.

Additionally:

- Lift $s \in Z_{k-1}$ with every $p \in P_k$ separately.
- Keep a queue of remaining lifting steps $(s, p)$ for continuation.
Partial projection

Rough template for recent projection operators [McC98, Bro01]

\[
Proj(P) = \{\text{disc}(p), \text{coeffs}^*(p) \mid p \in P\} \cup \\
\{\text{res}(p, q) \mid p, q \in P\}
\]
Partial projection

Rough template for recent projection operators [McC98, Bro01]

\[ Proj(P) = \{ \text{disc}(p), \text{coeffs}^*(p) \mid p \in P\} \cup \{ \text{res}(p, q) \mid p, q \in P\} \]

Observations:

- Every step is local to only one or two polynomials.
- We can interrupt the computation frequently.
Partial projection

Rough template for recent projection operators [McC98, Bro01]

\[ \text{Proj}(P) = \{ \text{disc}(p), \text{coeffs}^*(p) \mid p \in P \} \cup \{ \text{res}(p, q) \mid p, q \in P \} \]

Observations:
- Every step is local to only one or two polynomials.
- We can interrupt the computation frequently.

Key ideas:
- Split \( \text{Proj}(P) \) into a sequence of projection steps.
- Keep a queue of projection steps for continuation.
Observations:

- We lift every \((s, p)\) individually.
- We can also lift \((s, \cdot)\) and guess.
- We can stop as soon as a satisfying sample point is found.
Lazy projection

Observations:

- We lift every \((s, p)\) individually.
- We can also lift \((s, \cdot)\) and guess.
- We can stop as soon as a satisfying sample point is found.

Why should we even start with the projection?
Lazy projection

Observations:

- We lift every \((s, p)\) individually.
- We can also lift \((s, \cdot)\) and guess.
- We can stop as soon as a satisfying sample point is found.

Why should we even start with the projection?

Key ideas:

- Start with lifting.
- Perform lifting with respect to an incomplete projection.
- Only when lifting is complete*, spend time on the projection.
Lazy partial CAD

1. Perform lifting.
2. Return \textit{SAT} if satisfying sample point was found.
3. Return \textit{UNSAT} if the projection is complete.
4. Perform a projection step, go back to 1.
Lazy partial CAD

1. Perform lifting.
2. Return **SAT** if satisfying sample point was found.
3. Return **UNSAT** if the projection is complete.
4. Perform a projection step, go back to 1.

Observations:

- Completely driven by the search for a solution.
- Eventually **converges** to a complete CAD (if UNSAT).
- **New choices** to be made:
  - Order of lifting steps? (DFS? BFS? Something else?)
  - Order of projection steps? (Level? Degree?)
- Can be **continued** easily.
What about SMT compliancy now?

Reminder:

✓ early abort
  ▶ add constraints
  ▶ remove constraints
✓ reasons for unsatisfiability
Adding constraints

Observations:

- Partial projection maintains a queue of remaining projection steps.
- Partial lifting maintains a queue of remaining lifting steps.
- We can extend these queues for a new polynomial.
- We can continue from there.
Observations:

- Partial projection maintains a queue of remaining projection steps.
- Partial lifting maintains a queue of remaining lifting steps.
- We can extend these queues for a new polynomial.
- We can continue from there.

If partial projection and partial lifting is in place...

- ... adding new polynomials is easy.
- ... extending the partial CAD is natural.
- ... all previous computations can be reused.
Our lazy partial CAD is **monotically growing**.

⇒ adding constraints is easy, but removing constraints is hard.
Our lazy partial CAD is **monotically growing**.  
⇒ adding constraints is easy, but removing constraints is hard.

Different options:

- Keep everything and ignore removals.
- **Reset** whenever we remove something.
- Save a **snapshot** once in a while and restore it.
- Figure out **which polynomials to remove** and update properly.
Removing constraints

Our lazy partial CAD is **monotically growing**.

⇒ adding constraints is easy, but removing constraints is hard.

Different options:

- Keep everything and **ignore removals**.
  Constraints accumulate and CAD eventually blows up.

- **Reset** whenever we remove something.

- Save a **snapshot** once in a while and restore it.

- Figure out **which polynomials to remove** and update properly.
Our lazy partial CAD is monotically growing.  
⇒ adding constraints is easy, but removing constraints is hard.

Different options:

- Keep everything and ignore removals.  
  Constraints accumulate and CAD eventually blows up.
- **Reset** whenever we remove something.  
  Mostly destroys the idea of retaining state.
- Save a snapshot once in a while and restore it.
- Figure out which polynomials to remove and update properly.
Our lazy partial CAD is \textit{monotically growing}.

$\Rightarrow$ \textit{adding constraints is easy, but removing constraints is hard.}

Different options:

- Keep everything and \textit{ignore removals}.
  Constraints accumulate and CAD eventually blows up.

- \textbf{Reset} whenever we remove something.
  Mostly destroys the idea of retaining state.

- Save a \textit{snapshot} once in a while and restore it.
  When to snapshot? How many to keep? Memory usage?

- Figure out \textit{which polynomials to remove} and update properly.
Our lazy partial CAD is monotically growing.
⇒ adding constraints is easy, but removing constraints is hard.

Different options:

- Keep everything and ignore removals.
  Constraints accumulate and CAD eventually blows up.
- **Reset** whenever we remove something.
  Mostly destroys the idea of retaining state.
- Save a snapshot once in a while and restore it.
  When to snapshot? How many to keep? Memory usage?
- Figure out which polynomials to remove and update properly.
  How to do that?
Add \( z^2 + y^2 + x^2 < 4 \)

\[ z^2 + y^2 + x^2 < 4 \]
Add $z^2 + y^2 + x^2 - 4$

$z^2 + y^2 + x^2 < 4$

$z$

$z^2 + y^2 + x^2 - 4$

$y$

$x$
Backtracking in Projection

Project $z^2 + y^2 + x^2 - 4$

$z^2 + y^2 + x^2 < 4$

$z$

$z^2 + y^2 + x^2 - 4$

$y$

$y^2 + x^2 - 4$

$x$
Backtracking in Projection

Project $y^2 + x^2 - 4$

$z^2 + y^2 + x^2 < 4$

$z^2 + y^2 + x^2 - 4$

$y^2 + x^2 - 4$

$x^2 - 4$
Add $z^2 < 1$

$z^2 < 1$  $z^2 + y^2 + x^2 < 4$

$z$

$z^2 + y^2 + x^2 - 4$

$y$

$y^2 + x^2 - 4$

$x$

$x^2 - 4$
Add $z^2 - 1$

\[ z^2 < 1 \quad z^2 + y^2 + x^2 < 4 \]

\[
\begin{array}{c|c|c}
  z & z^2 - 1 & z^2 + y^2 + x^2 - 4 \\
\end{array}
\]

\[
\begin{array}{c|c}
  y & y^2 + x^2 - 4 \\
\end{array}
\]

\[
\begin{array}{c|c}
  x & x^2 - 4 \\
\end{array}
\]
Backtracking in Projection

Project $z^2 - 1$ and $z^2 + y^2 + x^2 - 4$

$z^2 < 1$  $z^2 + y^2 + x^2 < 4$

<table>
<thead>
<tr>
<th>$z$</th>
<th>$z^2 - 1$</th>
<th>$z^2 + y^2 + x^2 - 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>$y^2 + x^2 - 3$</td>
<td>$y^2 + x^2 - 4$</td>
</tr>
<tr>
<td>$x$</td>
<td>$x^2 - 4$</td>
<td></td>
</tr>
</tbody>
</table>
Backtracking in Projection

Project $y^2 + x^2 - 3$

$z^2 < 1 \quad z^2 + y^2 + x^2 < 4$

<table>
<thead>
<tr>
<th>$z$</th>
<th>$z^2 - 1$</th>
<th>$z^2 + y^2 + x^2 - 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>$y^2 + x^2 - 3$</td>
<td>$y^2 + x^2 - 4$</td>
</tr>
<tr>
<td>$x$</td>
<td>$x^2 - 3$</td>
<td>$x^2 - 4$</td>
</tr>
</tbody>
</table>
Backtracking in Projection

Add \(x^2 y - 3y > 0\)

\[
\begin{align*}
z^2 &< 1 \\
z^2 + y^2 + x^2 &< 4 \\
x^2 y - 3y &> 0
\end{align*}
\]

\[
\begin{array}{ccc}
z & z^2 - 1 & z^2 + y^2 + x^2 - 4 \\ y & y^2 + x^2 - 3 & y^2 + x^2 - 4 \\ x & x^2 - 3 & x^2 - 4
\end{array}
\]
Add \((x^2 - 3)y\)

\[\begin{align*}
  z^2 &< 1 \\
  z^2 + y^2 + x^2 &< 4 \\
  x^2y - 3y &> 0
\end{align*}\]

Given:

- \(z\):
  - \(z^2 - 1\)
  - \(z^2 + y^2 + x^2 - 4\)

- \(y\):
  - \(y^2 + x^2 - 3\)
  - \(y^2 + x^2 - 4\)
  - \((x^2 - 3)y\)

- \(x\):
  - \(x^2 - 3\)
  - \(x^2 - 4\)
Backtracking in Projection

Project $(x^2 - 3)y$

$$z^2 < 1 \quad z^2 + y^2 + x^2 < 4 \quad x^2y - 3y > 0$$

$$z^2 - 1 \quad z^2 + y^2 + x^2 - 4$$

$$y^2 + x^2 - 3 \quad y^2 + x^2 - 4 \quad (x^2 - 3)y$$

$$x^2 - 3 \quad x^2 - 4$$
Backtracking in Projection

Project $y^2 + x^2 - 4$ and $(x^2 - 3)y$

\[
\begin{align*}
  z^2 &< 1 \\
  z^2 + y^2 + x^2 &< 4 \\
  x^2y - 3y &> 0
\end{align*}
\]

\[
\begin{array}{llll}
  z & & & \\
  & z^2 - 1 & z^2 + y^2 + x^2 - 4 & \\
  y & & & \\
  & y^2 + x^2 - 3 & y^2 + x^2 - 4 & (x^2 - 3)y \\
  x & & & \\
  & x^2 - 3 & x^2 - 4 & x^4 - 7x^2 + 12
\end{array}
\]
Backtracking in Projection

Project $y^2 + x^2 - 3$ and $(x^2 - 3)y$

$\begin{align*}
  z^2 < 1 & \quad z^2 + y^2 + x^2 < 4 & \quad x^2y - 3y > 0 \\
z & \quad z^2 - 1 & \quad z^2 + y^2 + x^2 - 4 \\
y & \quad y^2 + x^2 - 3 & \quad y^2 + x^2 - 4 & \quad (x^2 - 3)y \\
x & \quad x^2 - 3 & \quad x^2 - 4 & \quad x^4 - 7x^2 + 12
\end{align*}$
Backtracking in Projection

Remove $z^2 < 1$

$z^2 + y^2 + x^2 < 4 \quad x^2y - 3y > 0$

\[
\begin{array}{c}
  z \\
  z^2 - 1 \\
  \rightarrow z^2 + y^2 + x^2 - 4
\end{array}
\]

\[
\begin{array}{c}
  y \\
  y^2 + x^2 - 3 \\
  \rightarrow y^2 + x^2 - 4 \\
  \rightarrow (x^2 - 3)y
\end{array}
\]

\[
\begin{array}{c}
  x \\
  x^2 - 3 \\
  \rightarrow x^2 - 4 \\
  \rightarrow x^4 - 7x^2 + 12
\end{array}
\]
Remove $z^2 - 1$

$z^2 < 1$

$z^2 + y^2 + x^2 < 4$

$x^2 y - 3y > 0$

$y^2 + x^2 - 3$

$y^2 + x^2 - 4$

$(x^2 - 3)y$

$x^2 - 3$

$x^2 - 4$

$x^4 - 7x^2 + 12$
Backtracking in Projection

Remove $z^2 - 1$

$z^2 - 1$ $z^2 + y^2 + x^2 < 4$ $x^2 y - 3y > 0$

$z$

$z^2 - 1$ $z^2 + y^2 + x^2 - 4$

$y$

$y^2 + x^2 - 3$ $y^2 + x^2 - 4$ $(x^2 - 3)y$

$x$

$x^2 - 3$ $x^2 - 4$ $x^4 - 7x^2 + 12$
Backtracking in Projection

Remove $y^2 + x^2 - 3$

$z^2 < 1$

$z^2 + y^2 + x^2 < 4$

$x^2 y - 3y > 0$

$z^2 < 1$

$z^2 + y^2 + x^2 - 4$

$y^2 + x^2 - 3$

$y^2 + x^2 - 4$

$(x^2 - 3)y$

$x^2 - 3$

$x^2 - 4$

$x^4 - 7x^2 + 12$
Backtracking in Projection

Remove $y^2 + x^2 - 3$

$z^2 < 1$  
$z^2 + y^2 + x^2 < 4$  
$x^2 y - 3y > 0$

$z$

$z^2 = 1$  
$z^2 + y^2 + x^2 - 4$

$y$

$y^2 + x^2 - 3$  
$y^2 + x^2 - 4$  
$(x^2 - 3)y$

$x$

$x^2 - 3$  
$x^2 - 4$  
$x^4 - 7x^2 + 12$
Backtracking in Projection

\[ z^2 + y^2 + x^2 < 4 \quad x^2y - 3y > 0 \]

\[
\begin{align*}
z &\quad z^2 + y^2 + x^2 - 4 \\
y &\quad y^2 + x^2 - 4 \quad (x^2 - 3)y \\
x &\quad x^2 - 3 \quad x^2 - 4 \quad x^4 - 7x^2 + 12
\end{align*}
\]
Pruning the lifting tree

Prune lifting after polynomials are removed.

Observations:
- A sample is either a root of one or more polynomial(s) or a value in-between two roots.
- We store the reasons for a root as a set of polynomials.
- To prune: remove a root if the reasons are gone.
- Remove one of the neighboring samples with every root.
Prune lifting after polynomials are removed.

Observations:

- A sample is either
  - a root of one or more polynomial(s) or
  - a value in-between two roots.
- We store the reasons for a root as a set of polynomials.
Prune lifting after polynomials are removed.

Observations:

- A sample is either
  - a root of one or more polynomial(s) or
  - a value in-between two roots.
- We store the reasons for a root as a set of polynomials.

To prune:

- Remove a root if the reasons are gone.
- Remove one of the neighboring samples with every root.
Evaluation

Is it worth it?
Experiments

- Benchmarks from SMT-LIB QF_NRA (11354 from 10 sources)
- Only SAT + CAD with different options:
Experiments

- Benchmarks from SMT-LIB QF_NRA (11354 from 10 sources)
- Only SAT + CAD with different options:
  - $\text{CAD}_{\text{Naive}}$: Fresh CAD on every theory call
  - $\text{CAD}_{\text{Eager}}$: Eagerly compute full projection
  - $\text{CAD}_{\text{Simple}}$: Eagerly add one constraint at a time
  - $\text{CAD}_{\text{Full}}$: Incremental projection
Experiments

- Benchmarks from SMT-LIB QF_NRA (11354 from 10 sources)
- Only SAT + CAD with different options:
  - $CAD_{Naive}$: Fresh CAD on every theory call
  - $CAD_{Eager}$: Eagerly compute full projection
  - $CAD_{Simple}$: Eagerly add one constraint at a time
  - $CAD_{Full}$: Incremental projection

<table>
<thead>
<tr>
<th>Solver</th>
<th>solved</th>
<th>runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CAD_{Naive}$</td>
<td>5571</td>
<td>0.69</td>
</tr>
<tr>
<td>$CAD_{Eager}$</td>
<td>7559</td>
<td>0.60</td>
</tr>
<tr>
<td>$CAD_{Simple}$</td>
<td>7924</td>
<td>1.11</td>
</tr>
<tr>
<td>$CAD_{Full}$</td>
<td>8158</td>
<td>1.22</td>
</tr>
</tbody>
</table>
Consider CAD as search method for a satisfying solution.
Perform projection and lifting incrementally.

More details in [KA18]
Conclusion

- Consider CAD as search method for a satisfying solution.
- Perform projection and lifting incrementally.

- Queues allow for easy continuation.
- Track reasons for polynomials and samples for removal.
Consider CAD as search method for a satisfying solution.
Perform projection and lifting incrementally.

Queues allow for easy continuation.
Track reasons for polynomials and samples for removal.

Very beneficial for practical solving.
More details in [KA18]
Further topics

- Factorization of polynomials?
  Integrates easily, only slight improvement

- Equational constraints?
  Somewhat tricky, only slight improvements [Hae17, HKÁ18]

- Impact of different heuristics?
  Surprisingly small, as long as we exploit incrementality

- Delineating polynomials?
  Integrates easily, somewhat obsolete (?)

- Generic quantifier elimination?
  Disable early abort and obtain a full CAD.

- Implementation?
  Bookkeeping is somewhat involved, but not to bad. See SMT-RAT!
References


