From MCSAT to CDSAT and beyond

Stéphane Graham-Lengrand
SRI International

Dagstuhl, September 2021
From MCSAT to CDSAT and beyond

Introduction to MCSAT

CDSAT

Proofs in CDSAT

Quantified satisfiability
1. Introduction to MCSAT
MCSAT introduced in [dMJ13, JBdM13, Jov17], based on Conflict Resolution [KTV09] and other works on decision procedures such as

- LPSAT [WW99]
- Separation logic [WIGG05]
- Linear Rational Arithmetic [MKS09, KTV09, Cot10]
- Linear Integer Arithmetic [Jd11]
- Non-Linear Arithmetic [JdM12] (NLSAT)
The model-constructing approach to SMT-solving 1/2

MCSAT introduced in [dMJ13, JBdM13, Jov17], based on Conflict Resolution [KTV09] and other works on decision procedures such as

- LPSAT [WW99]
- Separation logic [WIGG05]
- Linear Rational Arithmetic [MKS09, KTV09, Cot10]
- Linear Integer Arithmetic [Jd11]
- Non-Linear Arithmetic [JdM12] (NLSAT)

MCSAT offers:

- a template for decision procedures
- an integration of such procedures with Boolean reasoning
The model-constructing approach to SMT-solving 1/2

MCSAT introduced in [dMJ13, JBdM13, Jov17], based on Conflict Resolution [KTV09] and other works on decision procedures such as

- LPSAT [WW99]
- Separation logic [WIGG05]
- Linear Rational Arithmetic [MKS09, KTV09, Cot10]
- Linear Integer Arithmetic [Jd11]
- Non-Linear Arithmetic [JdM12] (NLSAT)

MCSAT offers:

- a template for decision procedures
- an integration of such procedures with Boolean reasoning

The template is a generalisation of how CDCL works. It is an instance of conflict-driven reasoning.
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.

![Diagram showing SAT player on the left, UNSAT player on the right, and model building and proof building cycles]

- SAT player
- UNSAT player
- Model building
- Proof building
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.

Archetype of conflict-driven reasoning: CDCL

A conflict occurs when a clause is falsified:

- $a \Rightarrow b$
- $b \Rightarrow a$
- $a \Rightarrow b$
- $b \Rightarrow a$

SAT player

UNSAT player

model building

proof building
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.

Archetype of conflict-driven reasoning: **CDCL**
a conflict occurs when a clause is falsified
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.

Archetype of conflict-driven reasoning: CDCL

A conflict occurs when a clause is falsified

SAT player

\[ a \Rightarrow b \]
\[ b \Rightarrow \overline{a} \]
\[ \overline{a} \Rightarrow b \]
\[ \overline{b} \Rightarrow a \]

UNSAT player

model building

proof building
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy. Archetype of conflict-driven reasoning: CDCL

a conflict occurs when a clause is falsified

\[ a \Rightarrow b \]
\[ b \Rightarrow \overline{a} \]
\[ \overline{a} \Rightarrow b \]
\[ b \Rightarrow a \]
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.

Archetype of conflict-driven reasoning: **CDCL**

A conflict occurs when a clause is falsified.
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy. Archetype of conflict-driven reasoning: CDCL. A conflict occurs when a clause is falsified.
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.

Archetype of conflict-driven reasoning: **CDCL**
a conflict occurs when a clause is falsified

\[
\begin{align*}
    a & \Rightarrow b \\
    b & \Rightarrow \overline{a} \\
    \overline{a} & \Rightarrow b \\
    b & \Rightarrow a \\
    \overline{a} &
\end{align*}
\]
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.

Archetype of conflict-driven reasoning: CDCL

A conflict occurs when a clause is falsified

\[ a \Rightarrow b \]
\[ b \Rightarrow \overline{a} \]
\[ \overline{a} \Rightarrow \overline{b} \]
\[ \overline{b} \Rightarrow a \]
\[ \overline{a} \]

SAT player

UNSAT player

model building

proof building
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.

Archetype of conflict-driven reasoning: **CDCL**
a conflict occurs when a clause is falsified

\[
\begin{align*}
    a & \Rightarrow b \\
    b & \Rightarrow \overline{a} \\
    \overline{a} & \Rightarrow b \\
    \overline{b} & \Rightarrow a \\
    \overline{a}
\end{align*}
\]
Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable. It involves a trail where a putative model is being specified. It relies on a notion of conflict between the putative model and the formula it should satisfy.

Archetype of conflict-driven reasoning: **CDCL**

A conflict occurs when a clause is falsified.

\[ a \Rightarrow b \]
\[ b \Rightarrow \overline{a} \]
\[ \overline{a} \Rightarrow b \]
\[ \overline{b} \Rightarrow a \]
\[ \overline{a} \]
\[ \bot \]
MCSAT vs CDSAT

MCSAT tailored to theories with a standard model used for evaluating constraints (example: arithmetic)

Evaluation is a key aspect of MCSAT
MCSAT vs CDSAT

MCSAT tailored to theories with a standard model used for evaluating constraints (example: arithmetic)

**Evaluation** is a key aspect of MCSAT

Solving satisfiability problem

\[ \text{Satisfaction problem} \]

(set of constraints on variables \( x_1, \ldots, x_n \))

= finding values for variables \( x_1, \ldots, x_n \) (so that constraints evaluate to true)

CDSAT [BGLS19] (for Conflict-Driven Satisfiability) is a more abstract framework where:

- evaluation is not a mandatory ingredient of search
- theory reasoning is abstracted using inference systems
- theory reasoning can be performed in a union of theories
- Boolean theory can be given the same status as other theories.

As an abstract framework, it counts among its instances:

- Equality sharing scheme (Nelson-Oppen combinations)
- CDCL (with restarts, learning, etc)
- MCSAT (original [dMJ13] version)
MCSAT vs CDSAT

MCSAT tailored to theories with a standard model used for evaluating constraints (example: arithmetic)

**Evaluation** is a key aspect of MCSAT

Solving satisfiability problem

(set of constraints on variables $x_1, \ldots, x_n$)

$=$ finding values for variables $x_1, \ldots, x_n$

(so that constraints evaluate to true)

CDSAT [BGLS19] (for Conflict-Driven Satisfiability) is a more abstract framework where

- evaluation is not a mandatory ingredient of search
- theory reasoning is abstracted using inference systems
- theory reasoning can be performed in a union of theories
- Boolean theory can be given the same status as other theories.
MCSAT vs CDSAT

MCSAT tailored to theories with a standard model used for evaluating constraints (example: arithmetic)

**Evaluation** is a key aspect of MCSAT

Solving satisfiability problem

(set of constraints on variables $x_1, \ldots, x_n$)

= finding values for variables $x_1, \ldots, x_n$

(so that constraints evaluate to true)

CDSAT [BGLS19] (for Conflict-Driven Satisfiability) is a more abstract framework where

- evaluation is not a mandatory ingredient of search
- theory reasoning is abstracted using inference systems
- theory reasoning can be performed in a union of theories
- Boolean theory can be given the same status as other theories.

As an abstract framework, it counts among its instances:

- Equality sharing scheme (Nelson-Oppen combinations)
- CDCL (with restarts, learning, etc)
- MCSAT (original [dMJ13] version)
Run = alternation of search phases and conflict analysis phases

- It uses assignments to first-order variables (e.g., $x \leftarrow 3/4$) like CDCL uses Boolean assignments to Boolean variables;
- It may explain conflicts by introducing atoms that are not in the input.
The model-constructing approach to SMT-solving 2/2

Run = alternation of search phases and conflict analysis phases

▶ It uses assignments to first-order variables (e.g., $x \leftarrow \frac{3}{4}$) like CDCL uses Boolean assignments to Boolean variables;

▶ It may explain conflicts by introducing atoms that are not in the input.

▶ As in CDCL, it successively guesses values to assign to variables...
  ...while maintaining the invariant: *given the assignments made so far, none of the constraints evaluates to false*
The model-constructing approach to SMT-solving 2/2

Run = alternation of search phases and conflict analysis phases

- It uses assignments to first-order variables (e.g., $x \leftarrow 3/4$) like CDCL uses Boolean assignments to Boolean variables;
- It may explain conflicts by introducing atoms that are not in the input.
- As in CDCL, it successively guesses values to assign to variables... ...while maintaining the invariant: given the assignments made so far, none of the constraints evaluates to false
- To pick a value for variable $y$ after $x_1, \ldots, x_n$ were assigned values $v_1, \ldots, v_n$, simply worry about constraints over variables $x_1, \ldots, x_n, y$ (i.e. constraints that have become unit in $y$)
The model-constructing approach to SMT-solving 2/2

Run = alternation of search phases and conflict analysis phases

- It uses assignments to first-order variables (e.g., $x \leftarrow 3/4$) like CDCL uses Boolean assignments to Boolean variables;
- It may explain conflicts by introducing atoms that are not in the input.
- As in CDCL, it successively guesses values to assign to variables...
  ...while maintaining the invariant: *given the assignments made so far, none of the constraints evaluates to false*
- To pick a value for variable $y$ after $x_1, \ldots, x_n$ were assigned values $v_1, \ldots, v_n$, simply worry about constraints over variables $x_1, \ldots, x_n, y$ (i.e. constraints that have become unit in $y$)
- If all variables get values while maintaining invariant $\Rightarrow$ SAT
The model-constructing approach to SMT-solving 2/2

Run = alternation of search phases and conflict analysis phases

▶ It uses assignments to first-order variables (e.g., \( x \leftarrow \frac{3}{4} \)) like CDCL uses Boolean assignments to Boolean variables;

▶ It may explain conflicts by introducing atoms that are not in the input.

▶ As in CDCL, it successively guesses values to assign to variables . . .
  . . . while maintaining the invariant: *given the assignments made so far, none of the constraints evaluates to false*

▶ To pick a value for variable \( y \) after \( x_1, \ldots, x_n \) were assigned values \( v_1, \ldots, v_n \), simply worry about constraints over variables \( x_1, \ldots, x_n, y \) (i.e. constraints that have become unit in \( y \))

▶ If all variables get values while maintaining invariant \( \Rightarrow \) SAT

▶ If at any point the invariant cannot be maintained:
  There is a conflict.
  MCSAT performs a conflict analysis,
  backtracks over some of the assignments \( x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n \)
  and tries new ones
An example in Linear Rational Arithmetic

\[
\begin{align*}
\ell_0 : & (-2 \cdot y - x < 0), \\
\ell_1 : & (y + x < 0), \\
\ell_2 : & (y < -1)
\end{align*}
\]

unsatisfiable in Linear Rational Arithmetic (LRA).
An example in Linear Rational Arithmetic

\[
\begin{align*}
    l_0 &: (-2 \cdot y - x < 0), \\
    l_1 &: (y + x < 0), \\
    l_2 &: (y < -1)
\end{align*}
\]

unsatisfiable in Linear Rational Arithmetic (LRA).

▶ **Guess** a value, e.g., \( x \leftarrow 0 \)
An example in Linear Rational Arithmetic

\[ l_0: (-2 \cdot y - x < 0), \quad l_1: (y + x < 0), \quad l_2: (y < -1) \]

unsatisfiable in Linear Rational Arithmetic (LRA).

- **Guess** a value, e.g., \( x \leftarrow 0 \)

  Then \( l_0 \) yields lower bound \( y > 0 \)
An example in Linear Rational Arithmetic

\[
\begin{align*}
\text{l}_0 & : (-2 \cdot y - x < 0), \\
\text{l}_1 & : (y + x < 0), \\
\text{l}_2 & : (y < -1)
\end{align*}
\]

unsatisfiable in Linear Rational Arithmetic (LRA).

- **Guess** a value, e.g., \( x \leftarrow 0 \)

  Then \( l_0 \) yields lower bound \( y > 0 \)

Together with \( l_2 \), range of possible values for \( y \) is empty. What to do? Just undo \( x \leftarrow 0 \) & remember \( x \neq 0 \)?
An example in Linear Rational Arithmetic

\[ l_0 (\ -2 \cdot y - x < 0), \quad l_1 (y + x < 0), \quad l_2 (y < -1) \]

unsatisfiable in Linear Rational Arithmetic (LRA).

- **Guess** a value, e.g., \( x \leftarrow 0 \)
  
  Then \( l_0 \) yields lower bound \( y > 0 \)
  
  Together with \( l_2 \), range of possible values for \( y \) is empty. What to do? Just undo \( x \leftarrow 0 \) & remember \( x \neq 0 \)?
  
  **No!** Clash of bounds suggests a better conflict explanation, by inferring \( l_0 + 2l_2 \),
  
i.e., \( l_3 (\ -x < -2) \)

It rules out \( x \leftarrow 0 \), but also many values that would fail for the same reasons.
An example in Linear Rational Arithmetic

\[\begin{align*}
&\substack{l_0 \\ (-2y - x < 0)}, \\
&\substack{l_1 \\ (y + x < 0)}, \\
&\substack{l_2 \\ (y < -1)}
\end{align*}\]

unsatisfiable in Linear Rational Arithmetic (LRA).

\[\begin{align*}
&\substack{l_3 \\ (-x < -2)}
\end{align*}\]

- **Guess** a value, e.g., \(x \leftarrow 0\)
  
  Then \(l_0\) yields lower bound \(y > 0\)

Together with \(l_2\), range of possible values for \(y\) is empty. What to do? Just undo \(x \leftarrow 0\) & remember \(x \neq 0\)?

- **No!** Clash of bounds suggests a better conflict explanation, by inferring \(l_0 + 2l_2\),
  
  i.e., \((-x < -2)\)

  It rules out \(x \leftarrow 0\), but also many values that would fail for the same reasons.

- Now undo the guess but keep \(l_3\).
An example in Linear Rational Arithmetic

\( l_0 \) \((\neg 2 \cdot y - x < 0)\), \( l_1 \) \((y + x < 0)\), \( l_2 \) \((y < -1)\)

unsatisfiable in Linear Rational Arithmetic (LRA).

\( l_3 \) \((\neg x < -2)\)

- **Guess** a value, e.g., \( x \leftarrow 0 \)
  
  Then \( l_0 \) yields lower bound \( y > 0 \)
  
  Together with \( l_2 \), range of possible values for \( y \) is empty. What to do? Just undo \( x \leftarrow 0 \) & remember \( x \neq 0 \)?

- **No!** Clash of bounds suggests a better conflict explanation, by inferring \( l_0 + 2l_2 \), i.e., \((\neg x < -2)\)
  
  It rules out \( x \leftarrow 0 \), but also many values that would fail for the same reasons.

- **Now undo the guess but keep \( l_3 \).**
An example in Linear Rational Arithmetic

\[ (\neg 2 \cdot y - x < 0), \quad (y + x < 0), \quad (y < -1) \]

unsatisfiable in Linear Rational Arithmetic (LRA).

▶ Guess a value, e.g., \( x \leftarrow 3 \)

\[ l_0 \]
\[ l_1 \]
\[ l_2 \]
\[ l_3 \]

\[ (-x < -2) \]

▶ Now undo the guess but keep \( l_4 \).

▶ Spot that \( l_3 \) and \( l_4 \) leave no value for \( x \).

Clash of bounds suggests inferring \( l_3 + l_4 \), i.e.,

\[ l_5 \]
\[ (0 < -2) \]

▶ No guess to undo. UNSAT.
An example in Linear Rational Arithmetic

\[ \begin{align*}
&l_0 : (-2 \cdot y - x < 0), \\
&l_1 : (y + x < 0), \\
&l_2 : (y < -1)
\end{align*} \]

unsatisfiable in Linear Rational Arithmetic (LRA).

▶ **Guess** a value, e.g., $x \leftarrow 3$

Then $l_0$ yields lower bound $y > -\frac{3}{2}$ and $l_1$ yields upper bound $y < -3$
An example in Linear Rational Arithmetic

\[\begin{align*}
(l_0 & : (-2 \cdot y - x < 0)),
(l_1 & : (y + x < 0)),
(l_2 & : (y < -1))
\end{align*}\]

unsatisfiable in Linear Rational Arithmetic (LRA).

▷ **Guess** a value, e.g., \(x \leftarrow 3\)

Then \(l_0\) yields lower bound \(y > -\frac{3}{2}\) and \(l_1\) yields upper bound \(y < -3\)

▷ **Clash of bounds** suggests inferring \(l_0 + 2l_1\), i.e., \((x < 0)\).
An example in Linear Rational Arithmetic

\[ l_0 \quad (-2 \cdot y - x < 0), \quad l_1 \quad (y + x < 0), \quad l_2 \quad (y < -1) \]

unsatisfiable in Linear Rational Arithmetic (LRA).

\[ l_3 \quad (-x < -2) \quad l_4 \quad (x < 0) \]

▶ **Guess** a value, e.g., \( x \leftarrow 3 \)

Then \( l_0 \) yields lower bound \( y > -\frac{3}{2} \) and \( l_1 \) yields upper bound \( y < -3 \)

▶ Clash of bounds suggests inferring \( l_0 + 2l_1 \), i.e., \( (x < 0) \).

▶ Now undo the guess but keep \( l_4 \).
An example in Linear Rational Arithmetic

\[
\begin{align*}
    l_0 &: (-2 \cdot y - x < 0), \\
    l_1 &: (y + x < 0), \\
    l_2 &: (y < -1) \\
\end{align*}
\]

unsatisfiable in Linear Rational Arithmetic (LRA).

\[
\begin{align*}
    l_3 &: (-x < -2) \\
    l_4 &: (x < 0) \\
\end{align*}
\]

- **Guess** a value, e.g., \(x \leftarrow 3\)
  
  Then \(l_0\) yields lower bound \(y > -\frac{3}{2}\) and \(l_1\) yields upper bound \(y < -3\)

- Clash of bounds suggests inferring \(l_0 + 2l_1\),
  
  i.e., \((x < 0)\).

- Now undo the guess but keep \(l_4\).

- Spot that \(l_3\) and \(l_4\) leave no value for \(x\).
  
  Clash of bounds suggests inferring \(l_3 + l_4\),
  
  i.e., \((0 < -2)\).
An example in Linear Rational Arithmetic

\[
\begin{align*}
\ell_0 &: (-2 \cdot y - x < 0), \\
\ell_1 &: (y + x < 0), \\
\ell_2 &: (y < -1)
\end{align*}
\]

unsatisfiable in Linear Rational Arithmetic (LRA).

\[
\begin{align*}
\ell_3 &: (-x < -2) \\
\ell_4 &: (x < 0)
\end{align*}
\]

- **Guess** a value, e.g., $x \leftarrow 3$
  Then $\ell_0$ yields lower bound $y > -\frac{3}{2}$ and $\ell_1$ yields upper bound $y < -3$
- **Clash of bounds suggests inferring** $\ell_0 + 2\ell_1$, i.e., $\ell_5 = (x < 0)$.
- **Now undo the guess but keep** $\ell_4$.
- **Spot that** $\ell_3$ and $\ell_4$ leave no value for $x$. Clash of bounds suggests **inferring** $\ell_3 + \ell_4$, i.e., $\ell_5 = (0 < -2)$.
- **No guess to undo.** UNSAT.
**Search phase (satisfiable case)**

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_1}</td>
<td>(C_1^1, \ldots, C_j^1, \ldots)</td>
<td></td>
<td>(x_1)</td>
</tr>
<tr>
<td>{x_1, x_2}</td>
<td>(C_1^2, C_2^2, \ldots, C_j^2, \ldots)</td>
<td></td>
<td>(x_2)</td>
</tr>
<tr>
<td>{x_1, x_2, x_3}</td>
<td>(C_1^3, C_2^3, \ldots, C_j^3, \ldots)</td>
<td></td>
<td>(x_3)</td>
</tr>
<tr>
<td>\ldots</td>
<td>(C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{x_1, \ldots, x_i}</td>
<td>(C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots)</td>
<td></td>
<td>(x_i)</td>
</tr>
</tbody>
</table>
Search phase (satisfiable case)

Free var within Constraints (unit ones in red) Feasible set Var
\{x_1\} \quad C_1^1, \ldots, C_j^1, \ldots \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad x_1
\{x_1, x_2\} \quad C_1^2, \ldots, C_j^2, \ldots \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad x_2
\{x_1, x_2, x_3\} \quad C_1^3, \ldots, C_j^3, \ldots \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad x_3
\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\
\{x_1, \ldots, x_i\} \quad C_1^i, \ldots, C_j^i, \ldots \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad x_i
## Search phase (satisfiable case)

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_1}</td>
<td>(C_1^1, \ldots, C_j^1, \ldots)</td>
<td></td>
<td>(x_1)</td>
</tr>
<tr>
<td>{x_1, x_2}</td>
<td>(C_1^2, C_2^2, \ldots, C_j^2, \ldots)</td>
<td></td>
<td>(x_2)</td>
</tr>
<tr>
<td>{x_1, x_2, x_3}</td>
<td>(C_1^3, C_2^3, \ldots, C_j^3, \ldots)</td>
<td></td>
<td>(x_3)</td>
</tr>
<tr>
<td>\ldots</td>
<td>(C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots)</td>
<td></td>
<td>(x_i)</td>
</tr>
<tr>
<td>{x_1, \ldots, x_i}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Search phase (satisfiable case)

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_1}</td>
<td>(C_1^1, \ldots, C_j^1, \ldots)</td>
<td>(x_1)</td>
<td></td>
</tr>
<tr>
<td>{x_1, x_2}</td>
<td>(C_1^2, C_2, \ldots, C_j^2, \ldots)</td>
<td>(x_2)</td>
<td></td>
</tr>
<tr>
<td>{x_1, x_2, x_3}</td>
<td>(C_1^3, C_2^3, \ldots, C_j^3, \ldots)</td>
<td>(x_3)</td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td>(C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{x_1, \ldots, x_i}</td>
<td></td>
<td>(x_i)</td>
<td></td>
</tr>
</tbody>
</table>

What to do now?

Backtrack and try new values \(v_1', \ldots, v_n'\) for \(x_1, \ldots, x_n\) (i.e. try another \(\Gamma'\)).

How do we avoid picking the same values (i.e. the same \(\Gamma\))?

How do we avoid picking a \(\Gamma'\) that fails for the same reason \(\Gamma\) fails?
Search phase (satisfiable case)

Free var within Constraints (unit ones in red) Feasible set Var
\{x_1\} \quad C_1^1, \ldots, C_j^1, \ldots \quad \boxed{\text{green}} \quad x_1
\{x_1, x_2\} \quad C_1^2, C_2^2, \ldots, C_j^2, \ldots \quad \boxed{\text{green}} \quad x_2
\{x_1, x_2, x_3\} \quad C_1^3, C_2^3, \ldots, C_j^3, \ldots \quad \boxed{\text{green}} \quad x_3
\ldots
\{x_1, \ldots, x_i\} \quad C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots \quad \boxed{\text{green}} \quad x_i

What to do now?
Backtrack and try new values \(v_1', \ldots, v_n'\) for \(x_1, \ldots, x_n\) (i.e. try another \(\Gamma'\))
How do we avoid picking the same values (i.e. the same \(\Gamma\))?
How do we avoid picking a \(\Gamma'\) that fails for the same reason \(\Gamma\) fails?
Search phase (satisfiable case)

Free var within Constraints (unit ones in red) Feasible set Var

\{x_1\} \quad C_1^1, \ldots, C_j^1, \ldots \quad \quad \color{green} x_1
\{x_1, x_2\} \quad C_1^2, C_2^2, \ldots, C_j^2, \ldots \quad \quad \color{green} x_2
\{x_1, x_2, x_3\} \quad C_1^3, C_2^3, \ldots, C_j^3, \ldots \quad \quad \color{green} x_3
\ldots
\{x_1, \ldots, x_i\} \quad C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots \quad \quad \color{green} x_i

SAT
## Search phase (conflict case)

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_1}</td>
<td>$C_1^1, \ldots, C_j^1, \ldots$</td>
<td>\textcolor{green}{\text{\ }}</td>
<td>\textcolor{red}{x_1}\textcolor{green}{\ }</td>
</tr>
<tr>
<td>{x_1, x_2}</td>
<td>$C_1^2, C_2^2, \ldots, C_j^2, \ldots$</td>
<td>\textcolor{green}{\text{\ }}</td>
<td>\textcolor{red}{x_2}\textcolor{green}{\ }</td>
</tr>
<tr>
<td>{x_1, x_2, x_3}</td>
<td>$C_1^3, C_2^3, \ldots, C_j^3, \ldots$</td>
<td>\textcolor{green}{\text{\ }}</td>
<td>\textcolor{red}{x_3}\textcolor{green}{\ }</td>
</tr>
<tr>
<td>\ldots</td>
<td>$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$</td>
<td>\textcolor{green}{\text{\ }}</td>
<td>\textcolor{red}{x_i}\textcolor{green}{\ }</td>
</tr>
<tr>
<td>{x_1, \ldots, x_i}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Search phase (conflict case)

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{x_1}</code></td>
<td>$C_1^1, \ldots, C_j^1, \ldots$</td>
<td></td>
<td>$x_1$</td>
</tr>
<tr>
<td><code>{x_1, x_2}</code></td>
<td>$C_1^2, C_2^2, \ldots, C_j^2, \ldots$</td>
<td></td>
<td>$x_2$</td>
</tr>
<tr>
<td><code>{x_1, x_2, x_3}</code></td>
<td>$C_1^3, C_2^3, \ldots, C_j^3, \ldots$</td>
<td></td>
<td>$x_3$</td>
</tr>
<tr>
<td>\ldots</td>
<td>$C_1^i, C_2^i, \ldots, C_j^i, \ldots$</td>
<td></td>
<td>$x_i$</td>
</tr>
<tr>
<td><code>{x_1, \ldots, x_i}</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Search phase (conflict case)

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_1}</td>
<td>(C_1^1, \ldots, C_j^1, \ldots)</td>
<td>![Green box] 1</td>
<td>(x_1)</td>
</tr>
<tr>
<td>{x_1, x_2}</td>
<td>(C_1^2, C_2^2, \ldots, C_j^2, \ldots)</td>
<td>![Green box] 0, 1</td>
<td>(x_2)</td>
</tr>
<tr>
<td>{x_1, x_2, x_3}</td>
<td>(C_1^3, C_2^3, \ldots, C_j^3, \ldots)</td>
<td>![Green box] 0, 1</td>
<td>(x_3)</td>
</tr>
<tr>
<td>\ldots</td>
<td>(C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots)</td>
<td>![Green box] 0, 1</td>
<td>(x_i)</td>
</tr>
<tr>
<td>{x_1, \ldots, x_i}</td>
<td>(C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots)</td>
<td>![Green box] 0, 1</td>
<td>(x_i)</td>
</tr>
</tbody>
</table>

**What to do now?**
- Backtrack and try new values \(v_1', \ldots, v_n'\) for \(x_1, \ldots, x_n\) (i.e. try another \(\Gamma'\)).

**How do we avoid picking the same values (i.e. the same \(\Gamma\))?**

**How do we avoid picking a \(\Gamma'\) that fails for the same reason \(\Gamma\) fails?**
**Search phase (conflict case)**

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_1}</td>
<td>(C_1^1, \ldots, C_j^1, \ldots)</td>
<td>[\text{Red} ]</td>
<td>(x_1)</td>
</tr>
<tr>
<td>{x_1, x_2}</td>
<td>(C_1^2, C_2^2, \ldots, C_j^2, \ldots)</td>
<td>[\text{Red} ]</td>
<td>(x_2)</td>
</tr>
<tr>
<td>{x_1, x_2, x_3}</td>
<td>(C_1^3, C_2^3, \ldots, C_j^3, \ldots)</td>
<td>[\text{Red} ]</td>
<td>(x_3)</td>
</tr>
<tr>
<td>\ldots }</td>
<td>(C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots)</td>
<td>[\text{Green} ]</td>
<td>(x_i)</td>
</tr>
</tbody>
</table>

Conflict

What to do now?

Backtrack and try new values \(v'_1, \ldots, v'_n\) for \(x_1, \ldots, x_n\) (i.e. try another \(\Gamma'^\prime\))

How do we avoid picking the same values (i.e. the same \(\Gamma'^\prime\))?

How do we avoid picking a \(\Gamma'^\prime\) that fails for the same reason \(\Gamma\) fails?
**Search phase (conflict case)**

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>({x_1})</td>
<td>(C_1^1, \ldots, C_j^1, \ldots)</td>
<td>(x_1)</td>
<td></td>
</tr>
<tr>
<td>({x_1, x_2})</td>
<td>(C_1^2, C_2^2, \ldots, C_j^2, \ldots)</td>
<td>(x_2)</td>
<td></td>
</tr>
<tr>
<td>({x_1, x_2, x_3})</td>
<td>(C_1^3, C_2^3, \ldots, C_j^3, \ldots)</td>
<td>(x_3)</td>
<td></td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td></td>
</tr>
<tr>
<td>({x_1, \ldots, x_i})</td>
<td>(C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots)</td>
<td>(x_i)</td>
<td></td>
</tr>
</tbody>
</table>

**Conflict**
## Search phase (conflict case)

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_1}</td>
<td>(C_1^1, \ldots, C_j^1, \ldots)</td>
<td>(x_1)</td>
<td></td>
</tr>
<tr>
<td>{x_1, x_2}</td>
<td>(C_1^2, C_2^2, \ldots, C_j^2, \ldots)</td>
<td>(x_2)</td>
<td></td>
</tr>
<tr>
<td>{x_1, x_2, x_3}</td>
<td>(C_1^3, C_2^3, \ldots, C_j^3, \ldots)</td>
<td>(x_3)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>(C_1^i, C_2^i, \ldots, C_j^i, \ldots)</td>
<td>(x_i)</td>
<td></td>
</tr>
<tr>
<td>{x_1, \ldots, x_i}</td>
<td>(C_1^i, C_2^i, \ldots, C_4^i, \ldots, C_j^i, \ldots)</td>
<td>(x_i)</td>
<td></td>
</tr>
</tbody>
</table>

Conflict

What to do now?
Search phase (conflict case)

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_1}</td>
<td>{C_1^1, \ldots, C_j^1}</td>
<td>x_1</td>
<td></td>
</tr>
<tr>
<td>{x_1, x_2}</td>
<td>{C_1^2, C_2^2, \ldots, C_j^2}</td>
<td>x_2</td>
<td></td>
</tr>
<tr>
<td>{x_1, x_2, x_3}</td>
<td>{C_1^3, C_2^3, \ldots, C_j^3}</td>
<td>x_3</td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td></td>
</tr>
<tr>
<td>{x_1, \ldots, x_i}</td>
<td>{C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i}</td>
<td>\ldots</td>
<td></td>
</tr>
</tbody>
</table>

Conflict

What to do now?
Backtrack and try new values \(v'_1, \ldots, v'_n\) for \(x_1, \ldots, x_n\)
(i.e. try another \(\Gamma'\))
### Search phase (conflict case)

<table>
<thead>
<tr>
<th>Free var within</th>
<th>Constraints (unit ones in red)</th>
<th>Feasible set</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_1}</td>
<td>(C_1^1, \ldots, C_j^1, \ldots)</td>
<td>\cellcolor{green!20} (\bullet)</td>
<td>(x_1)</td>
</tr>
<tr>
<td>{x_1, x_2}</td>
<td>(C_1^2, C_2^2, \ldots, C_j^2, \ldots)</td>
<td>\cellcolor{green!20} (\bullet) \cellcolor{green!20} (\bullet)</td>
<td>(x_2)</td>
</tr>
<tr>
<td>{x_1, x_2, x_3}</td>
<td>(C_1^3, C_2^3, \ldots, C_j^3, \ldots)</td>
<td>\cellcolor{green!20} (\bullet) \cellcolor{green!20} (\bullet) \cellcolor{green!20} (\bullet)</td>
<td>(x_3)</td>
</tr>
<tr>
<td>\ldots</td>
<td>(C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots)</td>
<td>\cellcolor{green!20} (\bullet) \cellcolor{green!20} (\bullet) \cellcolor{green!20} (\bullet) \cellcolor{green!20} (\bullet) \cellcolor{green!20} (\bullet)</td>
<td>(x_i)</td>
</tr>
</tbody>
</table>

Conflicts

---

**What to do now?**

Backtrack and try new values \(v_1', \ldots, v_n'\) for \(x_1, \ldots, x_n\)  
(i.e. try another \(\Gamma'\))

---

**How do we avoid picking the same values (i.e. the same \(\Gamma\))?**

How do we avoid picking a \(\Gamma'\) that fails for the same reason \(\Gamma\) fails?
Conflict analysis

In case of conflict we have

- assigned values $x_1 \mapsto v_1, \ldots, x_1 \mapsto v_n$, i.e., a model $\mathcal{M}$
Conflict analysis
In case of conflict we have

- assigned values $x_1 \mapsto v_1, \ldots, x_1 \mapsto v_n$, i.e., a model $\mathcal{M}$
- a collection of unit constraints in $y$:
  \[ l_1(\overrightarrow{x}, y) \land \cdots \land l_m(\overrightarrow{x}, y) \]
Conflict analysis

In case of conflict we have

- assigned values \( x_1 \mapsto v_1, \ldots, x_1 \mapsto v_n \), i.e., a model \( \mathcal{M} \)
- a collection of unit constraints in \( y \): \( l_1(x, y) \land \cdots \land l_m(x, y) \)
- detected that no value can be assigned to \( y \) to extend \( \mathcal{M} \) into a model of those unit constraints: \( \mathcal{M} \not\models \exists y A \)

where \( \exists y A \) is \( \exists y(l_1(x, y) \land \cdots \land l_m(x, y)) \)
Conflict analysis

In case of conflict we have

- assigned values $x_1 \mapsto v_1, \ldots, x_1 \mapsto v_n$, i.e., a model $\mathcal{M}$
- a collection of **unit constraints** in $y$: $l_1(\overrightarrow{x}, y) \land \cdots \land l_m(\overrightarrow{x}, y)$
- detected that no value can be assigned to $y$ to extend $\mathcal{M}$ into a model of those unit constraints: $\mathcal{M} \not\models \exists y A$
  
  where $\exists y A$ is $\exists y (l_1(\overrightarrow{x}, y) \land \cdots \land l_m(\overrightarrow{x}, y))$

Models satisfying $\exists y A$

$\mathcal{M}$
Conflict analysis

In case of conflict we have

- assigned values \( x_1 \mapsto v_1, \ldots, x_1 \mapsto v_n \), i.e., a model \( \mathcal{M} \)
- a collection of unit constraints in \( y \): \( l_1(\vec{x}, y) \land \cdots \land l_m(\vec{x}, y) \)
- detected that no value can be assigned to \( y \) to extend \( \mathcal{M} \) into a model of those unit constraints: \( \mathcal{M} \not\models \exists y A \)

where \( \exists y A \) is \( \exists y (l_1(\vec{x}, y) \land \cdots \land l_m(\vec{x}, y)) \)

Models satisfying \( \exists y A \)

\( \mathcal{M} \)
Conflict analysis

In case of conflict we have

- assigned values $x_1 \mapsto v_1, \ldots, x_1 \mapsto v_n$, i.e., a model $\mathcal{M}$
- a collection of unit constraints in $y$: $l_1(\overrightarrow{x}, y) \land \cdots \land l_m(\overrightarrow{x}, y)$
- detected that no value can be assigned to $y$ to extend $\mathcal{M}$ into a model of those unit constraints: $\mathcal{M} \models \exists y A$

where $\exists y A$ is $\exists y (l_1(\overrightarrow{x}, y) \land \cdots \land l_m(\overrightarrow{x}, y))$

We seek to generalise $\mathcal{M}$ into a class of models that do not satisfy $\exists y A$ “for the same reason” $\mathcal{M}$ does not.
Conflict analysis

In case of conflict we have

- assigned values $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$, i.e., a model $\mathcal{M}$
- a collection of unit constraints in $y$: $l_1(\overrightarrow{x},y) \land \cdots \land l_m(\overrightarrow{x},y)$
- detected that no value can be assigned to $y$ to extend $\mathcal{M}$ into a model of those unit constraints: $\mathcal{M} \models \exists y A$

where $\exists y A$ is $\exists y (l_1(\overrightarrow{x},y) \land \cdots \land l_m(\overrightarrow{x},y))$

We seek to generalise $\mathcal{M}$ into a class of models that do not satisfy $\exists y A$ “for the same reason” $\mathcal{M}$ does not.
The theory lemmas

\( \exists y A \) is \( \exists y (l_1(\overrightarrow{x}, y) \land \cdots \land l_m(\overrightarrow{x}, y)) \)

Models satisfying \( \exists y A \)

\( \bullet M \)

Models satisfying \( \neg B \)

We characterise this class as those models not satisfying \( \neg B \), for some quantifier-free \( B \) (with \( \text{fv}(B) \subseteq \{\neg \rightarrow x\} \)) such that

\[ \overrightarrow{T} \models (\exists y A) \Rightarrow B \]

\( B \) is an interpolant of \( \exists y A \) at \( M \).

MCSAT considers the theory lemma \( A \Rightarrow B \) that rules out not only \( M \) but a set of similar models (we impose that \( B \) be a clause, so \( A \Rightarrow B \) is a clause).

If \( A \) results from Boolean reasoning, it performs Boolean conflict analysis on \( A \) (Boolean resolutions).

It backtracks to a point where \( A \Rightarrow B \) is no longer violated, e.g., \( B \) no longer evaluates (to false).
The theory lemmas

$\exists yA$ is $\exists y(l_1(\overrightarrow{x}, y) \land \cdots \land l_m(\overrightarrow{x}, y))$

- Models satisfying $\exists yA$
- $\models M$ Models satisfying $\neg B$

We characterise this class as those models not satisfying $B$, for some quantifier-free $B$ (with $\text{fv}(B) \subseteq \{\overrightarrow{x}\}$) such that

- $\mathcal{T} \models (\exists yA) \Rightarrow B$
- $\not\models M \not\models B$

$B$ is an interpolant of $\exists yA$ at $M$. 

MCSAT considers the theory lemma $A \Rightarrow B$ that rules out not only $M$ but a set of similar models (we impose that $B$ be a clause, so $A \Rightarrow B$ is a clause).

If $A$ results from Boolean reasoning, it performs Boolean conflict analysis on $A$ (Boolean resolutions). It backtracks to a point where $A \Rightarrow B$ is no longer violated, e.g., $B$ no longer evaluates (to false).
The theory lemmas

$\exists y A$ is $\exists y (l_1(x, y) \land \cdots \land l_m(x, y))$

Models satisfying $\exists y A$

$\bullet M$

Models satisfying $\neg B$

We characterise this class as those models not satisfying $B$, for some quantifier-free $B$ (with $\text{fv}(B) \subseteq \{x\}$) such that

1. $\mathcal{T} \models (\exists y A) \Rightarrow B$
2. $M \not\models B$

$B$ is an interpolant of $\exists y A$ at $M$.

MCSAT considers the theory lemma $A \Rightarrow B$

that rules out not only $M$ but a set of similar models

(we impose that $B$ be a clause, so $A \Rightarrow B$ is a clause).
The theory lemmas

$\exists y A$ is $\exists y (l_1(\vec{x}, y) \land \cdots \land l_m(\vec{x}, y))$

We characterise this class as those models not satisfying $B$, for some quantifier-free $B$ (with $\text{fv}(B) \subseteq \{\vec{x}\}$) such that

- $\mathcal{T} \models (\exists y A) \Rightarrow B$
- $\mathcal{M} \not\models B$

$B$ is an interpolant of $\exists y A$ at $\mathcal{M}$.

MCSAT considers the theory lemma $A \Rightarrow B$
that rules out not only $\mathcal{M}$ but a set of similar models
(we impose that $B$ be a clause, so $A \Rightarrow B$ is a clause).
If $A$ results from Boolean reasoning,
it performs Boolean conflict analysis on $A$ (Boolean resolutions).
The theory lemmas

\[ \exists yA \text{ is } \exists y(l_1(\overrightarrow{x}, y) \land \cdots \land l_m(\overrightarrow{x}, y)) \]

We characterise this class as those models not satisfying \( B \), for some quantifier-free \( B \) (with \( \text{fv}(B) \subseteq \{\overrightarrow{x}\} \)) such that

1. \( \mathcal{T} \models (\exists yA) \Rightarrow B \)
2. \( \mathcal{M} \not\models B \)

\( B \) is an interpolant of \( \exists yA \) at \( \mathcal{M} \).

MCSAT considers the theory lemma \( A \Rightarrow B \) that rules out not only \( \mathcal{M} \) but a set of similar models (we impose that \( B \) be a clause, so \( A \Rightarrow B \) is a clause).

If \( A \) results from Boolean reasoning, it performs Boolean conflict analysis on \( A \) (Boolean resolutions).

It backtracks to a point where \( A \Rightarrow B \) is no longer violated, e.g., \( B \) no longer evaluates (to false).
MCSAT theories

Give me a theory $\mathcal{T}$ with

- a nice way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;
- such an explanation mechanism

... and I’ll give you an MCSAT calculus for $\mathcal{T}$, using some adaptation of the 2-watched literals technique for tracking unit constraints
MCSAT theories

Give me a theory $\mathcal{T}$ with

- a nice way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;

- such an explanation mechanism, producing $B$ as a clause (or $\neg B$ as a cube)

... and I’ll give you an MCSAT calculus for $\mathcal{T}$, using some adaptation of the 2-watched literals technique for tracking unit constraints
MCSAT theories

Give me a theory $\mathcal{T}$ with

- a nice way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;

- such an explanation mechanism, producing $B$ as a clause (or $\neg B$ as a cube), satisfying some suitable conditions for termination;

... and I’ll give you an MCSAT calculus for $\mathcal{T}$, using some adaptation of the 2-watched literals technique for tracking unit constraints
MCSAT theories

Give me a theory $\mathcal{T}$ with

- a nice way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;

- such an explanation mechanism, producing $B$ as a clause (or $\neg B$ as a cube), satisfying some suitable conditions for termination;

- optionally, a nice way to propagate a value for a variable whose domain has become a singleton set;

...and I’ll give you an MCSAT calculus for $\mathcal{T}$, using some adaptation of the 2-watched literals technique for tracking unit constraints.
MCSAT theories

Give me a theory $\mathcal{T}$ with

- a nice way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;

- such an explanation mechanism, producing $B$ as a clause (or $\neg B$ as a cube), satisfying some suitable conditions for termination;

- optionally, a nice way to propagate a value for a variable whose domain has become a singleton set;

...and I’ll give you an MCSAT calculus for $\mathcal{T}$, using some adaptation of the 2-watched literals technique for tracking unit constraints

In Yices: Boolean, non-linear arithmetic, EUF, bitvectors (can be mixed)
In arithmetic

In linear arithmetic, Fourier-Motzkin resolution can be used to eliminate a variable:

\[
\begin{align*}
& e_1 - y \preceq_1 0 \quad e_2 + y \preceq_2 0 \\
\hline
& e_1 + e_2 \preceq_3 0
\end{align*}
\]

with \(\preceq_1, \preceq_2, \preceq_3 \in \{\leq, <\}\) such that...
In arithmetic

In linear arithmetic, Fourier-Motzkin resolution can be used to eliminate a variable:

\[ \begin{align*}
  e_1 - y & \leq_1 0 \\
  e_2 + y & \leq_2 0 \\
  e_1 + e_2 & \leq_3 0
\end{align*} \]

with \( \leq_1, \leq_2, \leq_3 \in \{\leq, <\} \) such that…

In non-linear arithmetic,
Yices uses Cylindrical Algebraic Decomposition (CAD).
In arithmetic

In linear arithmetic, Fourier-Motzkin resolution can be used to eliminate a variable:

\[
\begin{align*}
e_1 - y & \leq_1 0 \\
e_2 + y & \leq_2 0 \\
e_1 + e_2 & \leq_3 0
\end{align*}
\]

with \(\leq_1, \leq_2, \leq_3 \in \{\leq, <\}\) such that...

In non-linear arithmetic, Yices uses Cylindrical Algebraic Decomposition (CAD).

At the SMT-comp, Yices has won QF_NRA (single query track) up until 2021 when cvc5 used a new technique based on cylindrical algebraic coverings (Abraham et al). On the other hand in 2021, Yices won NRA (single query track), ahead of z3. See Section 4 on quantifiers.
2. CDSAT
Context

CDCL (Conflict-Driven Clause Learning)

- procedure for deciding the satisfiability of Boolean formulae
- uses assignments of Boolean values to variables, e.g., \( l \leftarrow \text{true} \)

MCSAT (Model-Constructing Satisfiability) [dMJ13, Jov17]

- generalises CDCL to theory reasoning
- uses first-order assignments, e.g., \( x \leftarrow \sqrt{2} \)
Context

CDCL (Conflict-Driven Clause Learning)

- procedure for deciding the satisfiability of Boolean formulae
- uses assignments of Boolean values to variables, e.g., \( l \leftarrow \text{true} \)

MCSAT (Model-Constructing Satisfiability) [dMJ13, Jov17]

- generalises CDCL to theory reasoning
- uses first-order assignments, e.g., \( x \leftarrow \sqrt{2} \)

CDSAT (Conflict-Driven Satisfiability) [BGLS19, BGLS20]

- generalises MCSAT: generic combinations of abstract theories
- can also use first-order assignments
- models theory reasoning with modules made of inference rules
Context

CDCL (Conflict-Driven Clause Learning)
- procedure for deciding the satisfiability of Boolean formulae
- uses assignments of Boolean values to variables, e.g., \( l \leftarrow \text{true} \)

MCSAT (Model-Constructing Satisfiability) [dMJ13, Jov17]
- generalises CDCL to theory reasoning
- uses first-order assignments, e.g., \( x \leftarrow \sqrt{2} \)

CDSAT (Conflict-Driven Satisfiability) [BGLS19, BGLS20]
- generalises MCSAT: generic combinations of abstract theories
- can also use first-order assignments
- models theory reasoning with modules made of inference rules

MCSAT and CDSAT can explicitly provide, for satisfiable formulae, the model’s assignments of values to variables
Context

CDCL (Conflict-Driven Clause Learning)
- procedure for deciding the satisfiability of Boolean formulae
- uses assignments of Boolean values to variables, e.g., $l \leftarrow \text{true}$

MCSAT (Model-Constructing Satisfiability) [dMJ13, Jov17]
- generalises CDCL to theory reasoning
- uses first-order assignments, e.g., $x \leftarrow \sqrt{2}$

CDSAT (Conflict-Driven Satisfiability) [BGLS19, BGLS20]
- generalises MCSAT: generic combinations of abstract theories
- can also use first-order assignments
- models theory reasoning with modules made of inference rules

MCSAT and CDSAT can explicitly provide, for satisfiable formulae, the model’s assignments of values to variables
CDSAT can also provide proofs of unsat
Traditional architecture of SMT-solving

* e.g. equality sharing / Nelson-Oppen [NO79]
In CDSAT

...the theory combination is organised directly in the main conflict-driven loop:

As in MCSAT, trail contains

- Boolean assignments
  \[ a \leftarrow \text{true} \]
- First-order assignments
  \[ y \leftarrow \frac{3}{4} \]
In CDSAT

...the theory combination is organised directly in the main conflict-driven loop:

As in MCSAT, trail contains

- Boolean assignments
  \[ a \leftarrow \text{true} \]
- First-order assignments
  \[ y \leftarrow \frac{3}{4} \]

Features of conflict-driven satisfiability:

- Boolean theory can have the same status as other theories.
- Theory-specific reasoning often consists of fine-grained reasoning inferences, e.g., Fourier-Motzkin resolution for LRA:
  \[ (t_1 < x), (x < t_2) \vdash t_1 < t_2 \]
What is a theory module?

A set of inferences of the form

\[(t_1 \leftarrow c_1), \ldots, (t_k \leftarrow c_k) \vdash_T (l \leftarrow b)\]

where

- each \(t_i \leftarrow c_i\) is a single \(T\)-assignment
  - (a term \(t_i\) and a \(T\)-value \(c_i\) of matching sorts)

- \(l \leftarrow b\) is a single Boolean assignment
  - (a term \(l\) of sort \(\text{Bool}\) and a truth value \(b\))
What is a theory module?

A set of inferences of the form

\[(t_1 \leftarrow c_1), \ldots, (t_k \leftarrow c_k) \vdash_T (l \leftarrow b)\]

where

- each \( t_i \leftarrow c_i \) is a single \( T \)-assignment
  (a term \( t_i \) and a \( T \)-value \( c_i \) of matching sorts)

- \( l \leftarrow b \) is a single Boolean assignment
  (a term \( l \) of sort \( \text{Bool} \) and a truth value \( b \))

**Abbreviations:** \((l \leftarrow \text{true})\) as \( l \) and \((l \leftarrow \text{false})\) as \( \overline{l} \)
What is a theory module?

A set of inferences of the form

\[(t_1 \leftarrow c_1), \ldots, (t_k \leftarrow c_k) \vdash_{\mathcal{T}} (l \leftarrow b)\]

where

- each \( t_i \leftarrow c_i \) is a single \( \mathcal{T} \)-assignment
  (a term \( t_i \) and a \( \mathcal{T} \)-value \( c_i \) of matching sorts)

- \( l \leftarrow b \) is a single Boolean assignment
  (a term \( l \) of sort Bool and a truth value \( b \))

**Abbreviations:** (\( l \leftarrow \text{true} \)) as \( l \) and (\( l \leftarrow \text{false} \)) as \( \overline{l} \)

- **Soundness requirement:**
  Every model of the premisses is a model of the conclusion:
  \( (t_1 \leftarrow c_1), \ldots, (t_k \leftarrow c_k) \models (l \leftarrow b) \)
What is a theory module?

A set of inferences of the form

\[(t_1 \leftarrow c_1), \ldots, (t_k \leftarrow c_k) \vdash_T (l \leftarrow b)\]

where

- each \( t_i \leftarrow c_i \) is a single \( T \)-assignment
  (a term \( t_i \) and a \( T \)-value \( c_i \) of matching sorts)

- \( l \leftarrow b \) is a single Boolean assignment
  (a term \( l \) of sort \( \text{Bool} \) and a truth value \( b \))

Abbreviations: \((l \leftarrow \text{true})\) as \( l \) and \((l \leftarrow \text{false})\) as \( \overline{l} \)

Soundness requirement:
Every model of the premisses is a model of the conclusion:
\((t_1 \leftarrow c_1), \ldots, (t_k \leftarrow c_k) \models (l \leftarrow b)\)

Examples:
\[(x \leftarrow \sqrt{2}), (y \leftarrow \sqrt{2}) \vdash_{\text{NRA}} (x \cdot y \simeq 2)\] (evaluation inference)

\[(\overline{l_1} \lor \cdots \lor \overline{l_n}), \overline{l_1} \cdots, \overline{l_{n-1}} \vdash_{\text{Bool}} l_n\] (unit propagation)
What is a theory module? (Equality inferences)

All theory modules have the equality inferences:

\[
\begin{align*}
\text{t}_1 & \leftarrow \text{c}_1, \text{t}_2 \leftarrow \text{c}_2 \vdash \Psi \quad \text{t}_1 \simeq \text{t}_2 \text{ if } \text{c}_1 \text{ and } \text{c}_2 \text{ are the same value} \\
\text{t}_1 & \leftarrow \text{c}_1, \text{t}_2 \leftarrow \text{c}_2 \vdash \Psi \quad \text{t}_1 \not\simeq \text{t}_2 \text{ if } \text{c}_1 \text{ and } \text{c}_2 \text{ are distinct values} \\
\end{align*}
\]

\[
\begin{align*}
\vdash \Psi \quad \text{t}_1 \simeq \text{t}_1 \quad & \text{reflexivity} \\
\text{t}_1 \simeq \text{t}_2 \vdash \Psi \quad \text{t}_2 \simeq \text{t}_1 \quad & \text{symmetry} \\
\text{t}_1 \simeq \text{t}_2, \text{t}_2 \simeq \text{t}_3 \vdash \Psi \quad \text{t}_1 \simeq \text{t}_3 \quad & \text{transitivity}
\end{align*}
\]
CDSAT states

Search states: simply trails.
A trail is a stack of justified assignments $H \vdash (t \leftarrow c)$ and decisions $?(t \leftarrow c)$ coming from different theories
Justification $H$: a set of assignments that appear earlier on the trail
CDSAT states

Search states: simply trails.
A trail is a stack of justified assignments $\vdash (t \leftarrow c)$ and decisions $\vdash(t \leftarrow c)$ coming from different theories
Justification $H$: a set of assignments that appear earlier on the trail

Each assignment on the trail has a level
(index of highest decision in transitive justification of the assignment)
CDSAT states

Search states: simply trails. A trail is a stack of justified assignments \( H \vdash (t \leftarrow c) \) and decisions \( ?(t \leftarrow c) \) coming from different theories.

Justification \( H \): a set of assignments that appear earlier on the trail.

Each assignment on the trail has a **level** (index of highest decision in transitive justification of the assignment).

Example (trail grows from left to right):

\[
\emptyset \vdash (x \equiv z), \emptyset \vdash (y \equiv z), ?(x \leftarrow \sqrt{2}), ?(y \leftarrow \text{blue}), ?(x \leftarrow \text{red}), H \vdash (x \not\equiv y)
\]

where \( H \) is \{\( (y \leftarrow \text{blue}), (x \leftarrow \text{red}) \)\}.

Everything is on the trail, including assertions from the input problem, with empty justifications

\( \text{(e.g., } \emptyset \vdash (C \leftarrow \text{true}) \text{ for an input clause } C) \),

Conflict states: \( \langle \Gamma; H \rangle \), trail \( \Gamma + \) set \( H \) of trail assignments that are in conflict.
CDSAT: Search rules

Let $\mathcal{T}$ be a theory with a specific $\mathcal{T}$-module.

**Decide**

$\Gamma \rightarrow \Gamma, ?(t \leftarrow c)$

**Deduce**

$\Gamma \rightarrow \Gamma, J \vdash (t \leftarrow b)$ if $J \vdash_{\mathcal{T}} (t \leftarrow b)$ and $J \subseteq \Gamma$, and $t \leftarrow \overline{b}$ is not in $\Gamma$,

All terms that are ever mentioned in a derivation are taken from a finite set $\mathcal{B}$ called *global basis*
CDSAT: Search rules

Let $\mathcal{T}$ be a theory with a specific $\mathcal{T}$-module.

**Decide**

\[
\Gamma \quad \rightarrow \quad \Gamma, \, ?(t \leftarrow c)
\]

**Deduce**

\[
\Gamma \quad \rightarrow \quad \Gamma, \, J \vdash (t \leftarrow b) \quad \text{if } J \vdash_{\mathcal{T}} (t \leftarrow b) \text{ and } J \subseteq \Gamma, \text{ and } t \leftarrow b \text{ is not in } \Gamma,
\]

**Conflict**

\[
\Gamma \quad \rightarrow \quad \langle \Gamma; \, J, \, (t \leftarrow b) \rangle \quad \text{if } J \vdash_{\mathcal{T}} (t \leftarrow b) \text{ and } J \subseteq \Gamma, \text{ and } t \leftarrow b \text{ is in } \Gamma \text{ and conflict level is } > 0
\]

**Fail**

\[
\Gamma \quad \rightarrow \quad \text{unsat} \quad \text{if } J \vdash_{\mathcal{T}} (t \leftarrow b) \text{ and } J \subseteq \Gamma, \text{ and } t \leftarrow b \text{ is in } \Gamma \text{ and conflict level is } 0
\]

All terms that are ever mentioned in a derivation are taken from a finite set $\mathcal{B}$ called *global basis*.
CDSAT: Search rules

Let $\mathcal{T}$ be a theory with a specific $\mathcal{T}$-module.

** Decide 
$\Gamma \rightarrow \Gamma, \bar{?}(t \leftarrow c)$ if . . .

** Deduce 
$\Gamma \rightarrow \Gamma, \exists J \vdash (t \leftarrow b)$ if $J \vdash \mathcal{T}(t \leftarrow b)$ and $J \subseteq \Gamma$, and $t \leftarrow \bar{b}$ is not in $\Gamma$, and . . .

** Conflict 
$\Gamma \rightarrow \langle \Gamma; J, (t \leftarrow \bar{b}) \rangle$ if $J \vdash \mathcal{T}(t \leftarrow b)$ and $J \subseteq \Gamma$, and $t \leftarrow \bar{b}$ is in $\Gamma$ and conflict level is $> 0$

** Fail 
$\Gamma \rightarrow \text{unsat}$ if $J \vdash \mathcal{T}(t \leftarrow b)$ and $J \subseteq \Gamma$, and $t \leftarrow \bar{b}$ is in $\Gamma$ and conflict level is 0

Extra side-conditions “. . .” to ensure termination (no impact on soundness)

All terms that are ever mentioned in a derivation are taken from a finite set $\mathcal{B}$ called global basis
CDSAT: Conflict analysis rules

Resolve
\[ \langle \Gamma; E \uplus \{A\} \rangle \rightarrow \langle \Gamma; E \uplus H \rangle \quad \text{if } H \vdash A \text{ is in } \Gamma \text{ and } \ldots \]

Learn
\[ \langle \Gamma; E \uplus H \rangle \rightarrow \Gamma \leq m, E \vdash L \quad \text{if } L \text{ is a “clausal form” of } H \text{ and } \ldots \]
\[ L \notin \Gamma, \, \bar{L} \notin \Gamma, \text{ and } E \subseteq \Gamma \leq m \]
CDSAT: Conflict analysis rules

Resolve
\[ \langle \Gamma; E \cup \{A\} \rangle \rightarrow \langle \Gamma; E \cup H \rangle \quad \text{if } H \vdash A \text{ is in } \Gamma \text{ and } \ldots \]

Learn
\[ \langle \Gamma; E \cup H \rangle \rightarrow \; \Gamma \leq m, \; E \vdash L \quad \text{if } L \text{ is a “clausal form” of } H \text{ and } \ldots \]
\[ L \not\in \Gamma, \; \overline{L} \not\in \Gamma, \text{ and } E \subseteq \Gamma \leq m \]
CDSAT: Conflict analysis rules

Resolve
\[ \langle \Gamma; E \uplus \{A\} \rangle \rightarrow \langle \Gamma; E \uplus H \rangle \text{ if } H \vdash A \text{ is in } \Gamma \text{ and } \ldots \]

Learn
\[ \langle \Gamma; E \uplus H \rangle \rightarrow \Gamma \leq m, E \vdash L \text{ if } L \text{ is a “clausal form” of } H \text{ and } \ldots \]
\[ L \notin \Gamma, \overline{L} \notin \Gamma, \text{ and } E \subseteq \Gamma \leq m \]

\( \Gamma \leq m \): the pruning of trail \( \Gamma \), removing all assignments of level \( > m \)
CDSAT: Conflict analysis rules

Resolve
\[ \langle \Gamma; E \cup \{ A \} \rangle \rightarrow \langle \Gamma; E \cup H \rangle \] if \( H \vdash A \) is in \( \Gamma \) and ... 

Learn
\[ \langle \Gamma; E \cup H \rangle \rightarrow \Gamma \leq^m, E \vdash L \] if \( L \) is a “clausal form” of \( H \) and ... 
\( L \notin \Gamma, \bar{L} \notin \Gamma \), and \( E \subseteq \Gamma \leq^m \)

\( \Gamma \leq^m \): the pruning of trail \( \Gamma \), removing all assignments of level \( > m \)

Clausal forms of \( H \) reify \( H \) in Boolean logic:
\[
((\bigwedge_{(l \leftarrow \text{true}) \in H} l) \land (\bigwedge_{(l \leftarrow \text{false}) \in H} \neg l)) \leftarrow \text{false}
\]
CDSAT: Conflict analysis rules

Resolve
$$\langle \Gamma; E \cup \{A\} \rangle \rightarrow \langle \Gamma; E \cup H \rangle \text{ if } H \vdash A \text{ is in } \Gamma \text{ and } \ldots$$

Learn
$$\langle \Gamma; E \cup H \rangle \rightarrow \Gamma \leq m, E \vdash L \text{ if } L \text{ is a "clausal form" of } H \text{ and } \ldots$$

$$L \notin \Gamma, \overline{L} \notin \Gamma, \text{ and } E \subseteq \Gamma \leq m$$

$$\Gamma \leq m$$: the pruning of trail $$\Gamma$$, removing all assignments of level $$> m$$

Clausal forms of $$H$$ reify $$H$$ in Boolean logic:

$$(((\bigwedge (l \leftarrow \text{true}) \in H \ l) \wedge (\bigwedge (l \leftarrow \text{false}) \in H \neg l)) \leftarrow \text{false}$$

$$(((\bigvee (l \leftarrow \text{true}) \in H \neg l) \vee (\bigvee (l \leftarrow \text{false}) \in H l)) \leftarrow \text{true}$$
CDSAT: Conflict analysis rules

Resolve
\[ \langle \Gamma; E \cup \{A\} \rangle \rightarrow \langle \Gamma; E \cup H \rangle \quad \text{if } H \vdash A \text{ is in } \Gamma \text{ and ...} \]

Learn
\[ \langle \Gamma; E \cup H \rangle \rightarrow \Gamma \leq m, \overline{E} \vdash \overline{L} \quad \text{if } L \text{ is a "clausal form" of } H \text{ and ...} \]
\[ L \notin \Gamma, \overline{L} \notin \Gamma, \text{ and } E \subseteq \Gamma \leq m \]

Undo
\[ \langle \Gamma; E \cup \{A\} \rangle \rightarrow \Gamma \leq m - 1 \quad \text{if } A \text{ is a first-order decision and ...} \]

UndoDecide
\[ \langle \Gamma; E \cup \{L\} \rangle \rightarrow \Gamma \leq m - 1, \overline{L} \quad \text{if } H \vdash L \text{ is in } \Gamma, \text{ and ...} \]

\[ \Gamma \leq m: \text{ the pruning of trail } \Gamma, \text{ removing all assignments of level } > m \]

Clausal forms of \( H \) reify \( H \) in Boolean logic:
\[
\begin{align*}
\left( \bigwedge_{(l \leftarrow \text{true}) \in H} l \right) \land \left( \bigwedge_{(l \leftarrow \text{false}) \in H} \neg l \right) & \leftarrow \text{false} \\
\left( \bigvee_{(l \leftarrow \text{true}) \in H} \neg l \right) \lor \left( \bigvee_{(l \leftarrow \text{false}) \in H} l \right) & \leftarrow \text{true}
\end{align*}
\]
An example with arithmetic, arrays, congruence

\[ f(a[i:= v][j]) \equiv w, \ w - 2 \equiv f(u), \ i \equiv j, \ u \equiv v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i:= v][j]) \equiv w )</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>( w - 2 \equiv f(u) )</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>( i \equiv j )</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>( u \equiv v )</td>
<td>{}</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i := v][j]) \simeq w, \ w - 2 \simeq f(u), \ i \simeq j, \ u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i := v][j]) \simeq w )</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>( w - 2 \simeq f(u) )</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>( i \simeq j )</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>( u \simeq v )</td>
<td>{}</td>
</tr>
<tr>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>?</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i:= v][j]) \simeq w, \ w - 2 \simeq f(u), \ i \simeq j, \ u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i:= v][j]) \simeq w )</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>( w - 2 \simeq f(u) )</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>( i \simeq j )</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>( u \simeq v )</td>
<td>{}</td>
</tr>
<tr>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>5</td>
<td>( v \leftarrow c )</td>
<td>?</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i:= v][j]) \simeq w, \ w - 2 \simeq f(u), \ i \simeq j, \ u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[ f(a[i:= v][j]) \simeq w ]</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>[ w - 2 \simeq f(u) ]</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>[ i \simeq j ]</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>[ u \simeq v ]</td>
<td>{}</td>
</tr>
<tr>
<td>4</td>
<td>[ u \leftarrow c ]</td>
<td>?</td>
</tr>
<tr>
<td>5</td>
<td>[ v \leftarrow c ]</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>[ a[i:= v][j] \leftarrow c ]</td>
<td>?</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i:= v][j]) \cong w, \ w - 2 \cong f(u), \ i \cong j, \ u \cong v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i:= v][j]) \cong w )</td>
<td>{ } 0</td>
</tr>
<tr>
<td>1</td>
<td>( w - 2 \cong f(u) )</td>
<td>{ } 0</td>
</tr>
<tr>
<td>2</td>
<td>( i \cong j )</td>
<td>{ } 0</td>
</tr>
<tr>
<td>3</td>
<td>( u \cong v )</td>
<td>{ } 0</td>
</tr>
<tr>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>? 1</td>
</tr>
<tr>
<td>5</td>
<td>( v \leftarrow c )</td>
<td>? 2</td>
</tr>
<tr>
<td>6</td>
<td>( a[i:= v][j] \leftarrow c )</td>
<td>? 3</td>
</tr>
<tr>
<td>7</td>
<td>( w \leftarrow 0 )</td>
<td>? 4</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i:= v][j]) \simeq w, \ w - 2 \simeq f(u), \ i \simeq j, \ u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i:= v][j]) \simeq w )</td>
<td>( {} ) 0</td>
</tr>
<tr>
<td>1</td>
<td>( w - 2 \simeq f(u) )</td>
<td>( {} ) 0</td>
</tr>
<tr>
<td>2</td>
<td>( i \simeq j )</td>
<td>( {} ) 0</td>
</tr>
<tr>
<td>3</td>
<td>( u \simeq v )</td>
<td>( {} ) 0</td>
</tr>
<tr>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>? 1</td>
</tr>
<tr>
<td>5</td>
<td>( v \leftarrow c )</td>
<td>? 2</td>
</tr>
<tr>
<td>6</td>
<td>( a[i:= v][j] \leftarrow c )</td>
<td>? 3</td>
</tr>
<tr>
<td>7</td>
<td>( w \leftarrow 0 )</td>
<td>? 4</td>
</tr>
<tr>
<td>8</td>
<td>( f(a[i:= v][j]) \leftarrow 0 )</td>
<td>? 5</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i:= v][j]) \simeq w, \ w - 2 \simeq f(u), \ i \simeq j, \ u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i:= v][j]) \simeq w )</td>
<td>{} 0</td>
</tr>
<tr>
<td>1</td>
<td>( w - 2 \simeq f(u) )</td>
<td>{} 0</td>
</tr>
<tr>
<td>2</td>
<td>( i \simeq j )</td>
<td>{} 0</td>
</tr>
<tr>
<td>3</td>
<td>( u \simeq v )</td>
<td>{} 0</td>
</tr>
<tr>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>? 1</td>
</tr>
<tr>
<td>5</td>
<td>( v \leftarrow c )</td>
<td>? 2</td>
</tr>
<tr>
<td>6</td>
<td>( a[i:= v][j] \leftarrow c )</td>
<td>? 3</td>
</tr>
<tr>
<td>7</td>
<td>( w \leftarrow 0 )</td>
<td>? 4</td>
</tr>
<tr>
<td>8</td>
<td>( f(a[i:= v][j]) \leftarrow 0 )</td>
<td>? 5</td>
</tr>
<tr>
<td>9</td>
<td>( f(u) \leftarrow -2 )</td>
<td>? 6</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i:= v][j]) \simeq w \, , \, w - 2 \simeq f(u) \, , \, i \simeq j \, , \, u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i:= v][j]) \simeq w )</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>( w-2 \simeq f(u) )</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>( i \simeq j )</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>( u \simeq v )</td>
<td>{}</td>
</tr>
<tr>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>5</td>
<td>( v \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>( a[i:= v][j] \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>7</td>
<td>( w \leftarrow 0 )</td>
<td>?</td>
</tr>
<tr>
<td>8</td>
<td>( f(a[i:= v][j]) \leftarrow 0 )</td>
<td>?</td>
</tr>
<tr>
<td>9</td>
<td>( f(u) \leftarrow -2 )</td>
<td>?</td>
</tr>
<tr>
<td>10</td>
<td>( u \simeq a[i:= v][j] )</td>
<td>{4, 6}</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i := v][j]) \simeq w, \ w - 2 \simeq f(u), \ i \simeq j, \ u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i := v][j]) \simeq w )</td>
<td>{} \ 0</td>
</tr>
<tr>
<td>1</td>
<td>( w - 2 \simeq f(u) )</td>
<td>{} \ 0</td>
</tr>
<tr>
<td>2</td>
<td>( i \simeq j )</td>
<td>{} \ 0</td>
</tr>
<tr>
<td>3</td>
<td>( u \simeq v )</td>
<td>{} \ 0</td>
</tr>
<tr>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>? \ 1</td>
</tr>
<tr>
<td>5</td>
<td>( v \leftarrow c )</td>
<td>? \ 2</td>
</tr>
<tr>
<td>6</td>
<td>( a[i := v][j] \leftarrow c )</td>
<td>? \ 3</td>
</tr>
<tr>
<td>7</td>
<td>( w \leftarrow 0 )</td>
<td>? \ 4</td>
</tr>
<tr>
<td>8</td>
<td>( f(a[i := v][j]) \leftarrow 0 )</td>
<td>? \ 5</td>
</tr>
<tr>
<td>9</td>
<td>( f(u) \leftarrow -2 )</td>
<td>? \ 6</td>
</tr>
<tr>
<td>10</td>
<td>( u \simeq a[i := v][j] )</td>
<td>{4, 6} \ 3</td>
</tr>
<tr>
<td>11</td>
<td>( f(u) \not\simeq f(a[i := v][j]) )</td>
<td>{8, 9} \ 6</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i:=v][j]) \simeq w, \ w - 2 \simeq f(u), \ i \simeq j, \ u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i:=v][j]) \simeq w )</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>( w-2 \simeq f(u) )</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>( i \simeq j )</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>( u \simeq v )</td>
<td>{}</td>
</tr>
<tr>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>5</td>
<td>( v \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>( a[i:=v][j] \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>7</td>
<td>( w \leftarrow 0 )</td>
<td>?</td>
</tr>
<tr>
<td>8</td>
<td>( f(a[i:=v][j]) \leftarrow 0 )</td>
<td>?</td>
</tr>
<tr>
<td>9</td>
<td>( f(u) \leftarrow -2 )</td>
<td>?</td>
</tr>
<tr>
<td>10</td>
<td>( u \simeq a[i:=v][j] )</td>
<td>{4, 6}</td>
</tr>
<tr>
<td>11</td>
<td>( f(u) \not\simeq f(a[i:=v][j]) )</td>
<td>{8, 9}</td>
</tr>
</tbody>
</table>

conflict \( E^1: \{10, 11\} \) | 6 |
An example with arithmetic, arrays, congruence

\[ f(a[i:= v][j]) \simeq w, \ w - 2 \simeq f(u), \ i \simeq j, \ u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f(a[i:= v][j]) \simeq w )</td>
<td>{}</td>
<td>0</td>
<td>( f(a[i:= v][j]) \simeq w )</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>( w - 2 \simeq f(u) )</td>
<td>{}</td>
<td>1</td>
<td>( w - 2 \simeq f(u) )</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>( i \simeq j )</td>
<td>{}</td>
<td>2</td>
<td>( i \simeq j )</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>( u \simeq v )</td>
<td>{}</td>
<td>3</td>
<td>( u \simeq v )</td>
<td>{}</td>
</tr>
<tr>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>?</td>
<td>4</td>
<td>( u \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>5</td>
<td>( v \leftarrow c )</td>
<td>?</td>
<td>5</td>
<td>( v \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>( a[i:= v][j] \leftarrow c )</td>
<td>?</td>
<td>6</td>
<td>( a[i:= v][j] \leftarrow c )</td>
<td>?</td>
</tr>
<tr>
<td>7</td>
<td>( w \leftarrow 0 )</td>
<td>?</td>
<td>7</td>
<td>( u \simeq a[i:= v][j] )</td>
<td>{4, 6}</td>
</tr>
<tr>
<td>8</td>
<td>( f(a[i:= v][j]) \leftarrow 0 )</td>
<td>?</td>
<td>8</td>
<td>( f(u) \simeq f(a[i:= v][j]) )</td>
<td>{7}</td>
</tr>
<tr>
<td>9</td>
<td>( f(u) \leftarrow -2 )</td>
<td>?</td>
<td>9</td>
<td>( f(u) \neq f(a[i:= v][j]) )</td>
<td>{8, 9}</td>
</tr>
<tr>
<td>10</td>
<td>( u \simeq a[i:= v][j] )</td>
<td>{4, 6}</td>
<td>11</td>
<td>( f(u) \neq f(a[i:= v][j]) )</td>
<td>{8, 9}</td>
</tr>
<tr>
<td></td>
<td>( \text{conflict } E^1: {10, 11} )</td>
<td>6</td>
<td></td>
<td>( \text{conflict } E^1: {10, 11} )</td>
<td>6</td>
</tr>
</tbody>
</table>
An example with arithmetic, arrays, congruence

\[ f(a[i:= v][j]) \simeq w, \ w - 2 \simeq f(u), \ i \simeq j, \ u \simeq v \]

<table>
<thead>
<tr>
<th>id</th>
<th>trail items</th>
<th>just. lev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[ f(a[i:= v][j]) \simeq w ] { } \ 0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[ w - 2 \simeq f(u) ] { } \ 0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[ i \simeq j ] { } \ 0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>[ u \simeq v ] { } \ 0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[ u \leftarrow c ] ? \ 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[ v \leftarrow c ] ? \ 2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[ a[i:= v][j] \leftarrow c ] ? \ 3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>[ w \leftarrow 0 ] ? \ 4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>[ f(a[i:= v][j]) \leftarrow 0 ] ? \ 5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>[ f(u) \leftarrow -2 ] ? \ 6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>[ u \simeq a[i:= v][j] ] {4, 6} \ 3</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>[ f(u) \not\simeq f(a[i:= v][j]) ] {8, 9} \ 6</td>
<td></td>
</tr>
</tbody>
</table>

conflict \[ E^1: \{10, 11\}\] \ 6

\[ f(u) \simeq f(a[i:= v][j]) \] \{7\} \ 3

...
Termination and Soundness

Termination:

Theorem: If the global basis $B$ is finite, CDSAT terminates.

Soundness:

Theorem: Since each theory module $T$ is made of sound inferences, if the calculus ends with a conflict of level 0, then the input was unsat.
Termination and Soundness

Termination:

Theorem: If the global basis $B$ is finite, CDSAT terminates.

How to determine $B$? It should be sufficiently large to allow each theory module to explain its conflicts via deductions.
Termination and Soundness

**Termination:**

**Theorem:** If the global basis $B$ is finite, CDSAT terminates.

How to determine $B$? It should be sufficiently large to allow each theory module to explain its conflicts via deductions.

For each theory module $T$ involved, and all finite sets $X$ of terms (think of it as the terms of the input), we must have a finite set of terms basis$_T(X)$, called [local basis](#)

(those terms possibly introduced by $T$ during the run)
Termination and Soundness

**Termination:**

**Theorem:** If the global basis $B$ is finite, CDSAT terminates.

How to determine $B$? It should be sufficiently large to allow each theory module to explain its conflicts via deductions.

For each theory module $T$ involved, and all finite sets $X$ of terms (think of it as the terms of the input), we must have a finite set of terms basis$_T(X)$, called local basis (those terms possibly introduced by $T$ during the run).

If the local bases of $T_1, \ldots, T_n$ satisfy some (collective) properties, then it is possible to define a finite global basis $B$ for $\bigcup_{k=1}^n T_k$.

**Soundness:**

Since each theory module $T$ is made of sound inferences, if the calculus ends with a conflict of level 0, then the input was unsat. (you can even get a proof)
Termination and Soundness

**Termination:**

**Theorem:** If the global basis $B$ is finite, CDSAT terminates.

How to determine $B$? It should be sufficiently large to allow each theory module to explain its conflicts via deductions.

For each theory module $T$ involved, and all finite sets $X$ of terms (think of it as the terms of the input), we must have a finite set of terms basis $T(X)$, called **local basis** (those terms possibly introduced by $T$ during the run).

If the local bases of $T_1, \ldots, T_n$ satisfy some (collective) properties, then it is possible to define a finite global basis $B$ for $\bigcup_{k=1}^n T_k$.

**Soundness:**

**Theorem:** Since each theory module $T$ is made of sound inferences, if the calculus ends with a conflict of level 0, then the input was unsat.

(you can even get a proof)
What happens if we never get unsat?

Do we have a model?
What happens if we never get unsat?

Do we have a model?

This relies on a completeness condition for theory modules:

A $\mathcal{T}$-module is complete if for any $\Gamma$,

- Either There exists a $\mathcal{T}$-model of the *theory view* of $\Gamma$
- Or $\mathcal{T}$ can make a (relevant & acceptable) decision
- Or a $\mathcal{T}$-inference can deduce a new assignment (for a term in the local basis)
What happens if we never get unsat?

Do we have a model?

This relies on a completeness condition for theory modules:

A $\mathcal{T}$-module is complete if for any $\Gamma$,

- Either There exists a $\mathcal{T}$-model of the theory view of $\Gamma$
- Or $\mathcal{T}$ can make a (relevant & acceptable) decision
- Or a $\mathcal{T}$-inference can deduce a new assignment (for a term in the local basis)

In a combination though, the $\mathcal{T}_k$-models have to agree on the sorts' cardinalities and equalities between shared variables/terms.
What happens if we never get unsat?

Do we have a model?

This relies on a **completeness condition** for theory modules:
A $\mathcal{T}$-module is **complete** if for any $\Gamma$,

- **Either** There exists a $\mathcal{T}$-model of the *theory view* of $\Gamma$
- **Or** $\mathcal{T}$ can make a (relevant & acceptable) decision
- **Or** a $\mathcal{T}$-inference can deduce a new assignment (for a term in the local basis)

In a combination though, the $\mathcal{T}_k$-models have to agree on the sorts’ cardinalities and equalities between shared variables/terms.

We present a version of completeness that takes care of this: $\mathcal{T}_0$-**completeness**, where $\mathcal{T}_0$ is a reference theory that can be used to synchronise cardinalities (for a combination of stably infinite theories, take $\mathcal{T}_0$ to force the interpretation of all sorts to be $\mathbb{N}$).
What happens if we never get unsat?

Do we have a model?

This relies on a **completeness condition** for theory modules:

A $\mathcal{T}$-module is **complete** if for any $\Gamma$,

- **Either** There exists a $\mathcal{T}$-model of the *theory view* of $\Gamma$
- **Or** $\mathcal{T}$ can make a (relevant & acceptable) decision
- **Or** a $\mathcal{T}$-inference can deduce a new assignment (for a term in the local basis)

In a combination though, the $\mathcal{T}_k$-models have to agree on the sorts’ cardinalities and equalities between shared variables/terms.

We present a version of completeness that takes care of this: **$\mathcal{T}_0$-completeness**, where $\mathcal{T}_0$ is a reference theory that can be used to synchronise cardinalities (for a combination of stably infinite theories, take $\mathcal{T}_0$ to force the interpretation of all sorts to be $\mathbb{N}$).

**Theorem:** Assume $\mathcal{T}_0$ has a complete module, and all other theories have $\mathcal{T}_0$-complete modules.

If CDSAT cannot make any further transitions, then the trail describes a model for the union of the (extended) theories.
Theory modules given as examples in our papers

- **EUF**

\[(t_i \sim u_i)_{i=1 \ldots n}, (f(t_1, \ldots, t_n) \not\approx f(u_1, \ldots, u_n)) \vdash_{\text{EUF}} \bot\]

- **Arrays**: similar, except for extensionality

- **LRA**: evaluation inference, Fourier-Motzkin resolution inference as in MCSAT, etc
Theory modules given as examples in our papers

- **EUF**

  \[(t_i \simeq u_i)_{i=1\ldots n}, (f(t_1, \ldots, t_n) \not\simeq f(u_1, \ldots, u_n)) \vdash_{\text{EUF}} \bot\]

- **Arrays**: similar, except for extensionality

- **LRA**: evaluation inference, Fourier-Motzkin resolution inference as in MCSAT, etc

- **Black box procedure for equality-sharing**: coarse-grain inferences

  \[l_1 \leftarrow b_1, \ldots, l_n \leftarrow b_n \vdash_{\mathcal{T}} \bot\]

  where \(l_1, \ldots, l_n\) are formulæ, and the conjunction of the literals corresponding to the Boolean assignments \(l_1 \leftarrow b_1, \ldots, l_n \leftarrow b_n\) is \(\mathcal{T}\)-unsatisfiable (as detected by the black box)
Theory modules given as examples in our papers

- EUF
  
  \[(t_i \simeq u_i)_{i=1...n}, (f(t_1, \ldots, t_n) \not\equiv f(u_1, \ldots, u_n)) \vdash_{\text{EUF}} \bot\]

- Arrays: similar, except for extensionality
  
  \[(\mathcal{T}_0\text{-}\text{complete for all } \mathcal{T}_0\text{ such that...})\]

- LRA: evaluation inference, Fourier-Motzkin resolution inference as in MCSAT, etc
  
  \[(\mathcal{T}_0\text{-}\text{complete for all } \mathcal{T}_0\text{ imposing } |Q| \text{ infinite})\]

- Black box procedure for equality-sharing: coarse-grain inferences
  
  \[l_1 \leftarrow b_1, \ldots, l_n \leftarrow b_n \vdash_{\mathcal{T}} \bot\]
  
  where \(l_1, \ldots, l_n\) are formulæ, and the conjunction of the literals corresponding to the Boolean assignments \(l_1 \leftarrow b_1, \ldots, l_n \leftarrow b_n\) is \(\mathcal{T}\)-unsatisfiable (as detected by the black box)

  \[(\mathcal{T}_0\text{-}\text{complete for all } \mathcal{T}_0\text{ imposing the cardinality of all known sorts but } \text{Bool to be countably infinite})\]
3. Proofs in CDSAT
Theory proofs

To keep track of the soundness invariants, we need to refer to theory inferences.
Theory proofs

To keep track of the soundness invariants, we need to refer to theory inferences. Each theory module comes with a “proof annotation system”

$$(t_1 \leftarrow c_1), \ldots, (t_k \leftarrow c_k) \vdash_T (l \leftarrow b)$$

is annotated as

$$a_1(t_1 \leftarrow c_1), \ldots, a_k(t_k \leftarrow c_k) \vdash_T j_T : (l \leftarrow b)$$
Theory proofs

To keep track of the soundness invariants, we need to refer to theory inferences. Each theory module comes with a "proof annotation system"

\[(t_1 \leftarrow c_1), \ldots , (t_k \leftarrow c_k) \vdash_T (l \leftarrow b)\]
is annotated as

\[a_1(t_1 \leftarrow c_1), \ldots , a_k(t_k \leftarrow c_k) \vdash_T j_T : (l \leftarrow b)\]

**Examples:**
\[a_1(x \leftarrow \sqrt{2}), a_2(y \leftarrow \sqrt{2}) \vdash_{NRA} \text{eval}(\{a_1, a_2\}) : (x \cdot y \simeq 2)\]
(evaluation inference)

\[a_0(l_1 \lor \cdots \lor l_n), a_1(l_1), \ldots , a_{k-1}(l_{n-1}) \vdash_{Bool} \text{UP}(a_0, \{a_1, \ldots , a_n\}) : l_n\]
(unit propagation)
Proof-terms and proof-carrying CDSAT

▶ A proof-carrying trail is a stack
  ▶ of justified assignments $H \vdash j : (t \leftarrow c)$
  ▶ and decisions (?)$(t \leftarrow c)$

▶ A proof-carrying conflict state is of the form $\langle \Gamma; H; c \rangle$

...where $j$ and $c$ respectively range over

Deduction proof terms $j ::= \text{in} \mid j_T \mid \text{lem}(H.c)$
Conflict proof term $c ::= \text{cfl}(j_T, a) \mid \text{res}(j, a^{A.c})$

in annotates an input assignment,
$j_T$ ranges over theory proofs for $T$, used for Deduce
$\text{lem}(H.c)$ annotates justified assignments that Learn places on trail
(clausal forms of $H$), binding the identifiers of $H$ in $c$
$\text{cfl}(j_T, a)$ annotates a conflict when it is created by Conflict
$\text{res}(j, a^{A.c})$ annotates a conflict resulting from the Resolve rule,
binding $a$ in $c$
Provability invariants that proof-terms keep track of

\[
\begin{align*}
A \text{ is an input} & \quad \quad J \vdash_T j_T : L \\
\emptyset \vdash \text{in} : A & \quad \quad J \vdash j_T : L \\
E \cup H \vdash c : \bot & \quad \quad L \text{ clausal form of } H \\
E \vdash \text{lem}(H.c) : L \\
J \vdash_T j_T : L & \quad \quad H \vdash j : A \\
J \cup \{a\} \vdash \text{cfl}(j_T, a) : \bot & \quad \quad E, aA \vdash c : \bot \\
E \cup H \vdash \text{res}(j, aA.c) : \bot
\end{align*}
\]
Provability invariants that proof-terms keep track of

\[
\begin{array}{c}
\text{\(A\) is an input} & \text{\(J \vdash \_ : L\)} & \text{\(E \cup H \vdash \_ : \bot\)} & \text{\(L\) clausal form of \(H\)} \\
\emptyset \vdash \text{in : } A & J \vdash \_ : L & E \vdash \text{lem(}H.c) : L \\
\end{array}
\]

\[
\begin{array}{c}
J \vdash \_ : L \\
\text{\(J \cup \{A\} \vdash \text{cfl(}j, A\) : \bot} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(H \vdash j : A\)} & \text{\(E, aA \vdash \_ : \bot\)} \\
E \cup H \vdash \text{res(}j, aA.c\) : \bot} \\
\end{array}
\]

Rules of CDSAT are adapted so as to use those proof-terms, and the soundness invariants are materialised as:

**Theorem**

- For every assignment \(H \vdash j : A\) on the trail, \(H \vdash j : A\)
- For every conflict state \(\langle \Gamma; E; c \rangle\), \(E \vdash c : \bot.\)
Provability invariants that proof-terms keep track of

\[
\begin{align*}
A \text{ is an input} & \quad J \vdash \mathcal{T} j_T : L \\
\emptyset \vdash \text{in} : A & \quad J \vdash j_T : L \\
E \cup H \vdash c : \bot & \quad L \text{ clausal form of } H
\end{align*}
\]

\[
\begin{align*}
J \vdash \mathcal{T} j_T : L & \quad H \vdash j : A \\
J \cup \{a\} \vdash \text{cfl}(j_T, a) : \bot & \quad E, aA \vdash c : \bot \\
E \cup H \vdash \text{res}(j, aA.c) : \bot
\end{align*}
\]

Rules of CDSAT are adapted so as to use those proof-terms, and the soundness invariants are materialised as:

**Theorem**

- For every assignment \( H \vdash j : A \) on the trail, \( H \vdash j : A \)
- For every conflict state \( \langle \Gamma; E; c \rangle \), \( E \vdash c : \bot \).

The proof system above can be seen as glueing a collection of inference systems \((\vdash_T)_T\).
Provability invariants that proof-terms keep track of

\[
\begin{align*}
A \text{ is an input} & \quad J \vdash_T j_T : L & \quad E \cup H \vdash c : \perp \\
\emptyset \vdash \text{in} : A & \quad J \vdash j_T : L & \quad L \text{ clausal form of } H \\
J \vdash_T j_T : L & \quad H \vdash j : A & \quad E, {}^a A \vdash c : \perp \\
J \cup \left\{{}^a L\right\} \vdash \text{cfl}(j_T, a) : \perp & \quad E \cup H \vdash \text{res}(j, {}^a A.c) : \perp 
\end{align*}
\]

Rules of CDSAT are adapted so as to use those proof-terms, and the soundness invariants are materialised as:

**Theorem**

- For every assignment \( H \vdash j : A \) on the trail, \( H \vdash j : A \)
- For every conflict state \( \langle \Gamma; E; c \rangle \), \( E \vdash c : \perp \).

The proof system above can be seen as glueing a collection of inference systems \((\vdash_T)_T\)
CDSAT is a search procedure for the resulting system
Different views about proof objects

Proof-carrying CDSAT can be considered exactly as defined above, where $\text{in}, j_\mathcal{T}, \text{lem}(H.c), \text{cfl}(j_\mathcal{T}, a), \text{res}(j, ^aA.c)$ are terms.
Different views about proof objects

Proof-carrying CDSAT can be considered exactly as defined above, where \( \text{in}, j_T, \text{lem}(H.c), \text{cfl}(j_T, a), \text{res}(j, ^aA.c) \) are terms.

Another proof format is desired for output?
Just interpret the terms in that format after the run
(proof reconstruction)
Different views about proof objects

Proof-carrying CDSAT can be considered exactly as defined above, where $\text{in}, j_\text{T}, \text{lem}(H,c), \text{cfl}(j_\text{T}, a), \text{res}(j, \text{\textasciitilde}A,c)$ are terms.

Another proof format is desired for output? Just interpret the terms in that format after the run (proof reconstruction)

Alternatively, proof-carrying CDSAT can directly manipulate proofs in the format, if equipped with the operations corresponding to the term constructs. The proof-terms denote the manipulated proofs, but are never constructed.
Example: resolution proofs

If input contains no first-order assignments, resolution trees (or DAGs) form a proof format equipped with the right operations.
Example: resolution proofs

If input contains no first-order assignments, resolution trees (or DAGs) form a proof format equipped with the right operations
Leaves of resolution proofs are labeled by
- either literals corresponding to input assignments $\emptyset \vdash \text{in}: A$
- or theory lemmas corresponding to theory proofs $J \vdash_T j_T : L$
Internal nodes are obtained by applying resolution rule, corresponding to $H \vdash \text{res}(j, ^aA.c) : \bot$ constructs.
Example: resolution proofs

If input contains no first-order assignments, resolution trees (or DAGs) form a proof format equipped with the right operations

Leaves of resolution proofs are labeled by

- either literals corresponding to input assignments $\emptyset \vdash \text{in}: A$
- or theory lemmas corresponding to theory proofs $J \vdash_T j_T : L$

Internal nodes are obtained by applying resolution rule, corresponding to $H \vdash \text{res}(j,aA,c): \bot$ constructs.

If input does contain first-order assignments the resolution format has to be slightly extended, so that it manipulates guarded clauses of the form

$$\{(t_1 \leftarrow c_1), \ldots, (t_n \leftarrow c_n)\} \Rightarrow C$$

where $(t_1 \leftarrow c_1), \ldots, (t_n \leftarrow c_n)$ are first-order assign. guarding clause $C$

Details in the paper.
LCF: answers that are correct-by-construction

Other “proof format”:

- A deduction proof $j$ with $H \vdash j : L$ is the pair $\langle H, L \rangle$, and
- A conflict proof $c$ with $H \vdash c : \bot$ is $H$. 
LCF: answers that are correct-by-construction

Other “proof format”:

- A deduction proof $j$ with $H \vdash j : L$ is the pair $\langle H, L \rangle$, and
- A conflict proof $c$ with $H \vdash c : \bot$ is $H$.

No proof object to check.
LCF: answers that are correct-by-construction

Other “proof format”:

▶ A deduction proof $j$ with $H \vdash j : L$ is the pair $\langle H, L \rangle$, and
▶ A conflict proof $c$ with $H \vdash c : \bot$ is $H$.

No proof object to check. But the LCF architecture [Mil79, GMW79] can be used to ensure the correctness of answers.
LCF: answers that are correct-by-construction

Other “proof format”:

- A deduction proof \( j \) with \( H \vdash j : L \) is the pair \( \langle H, L \rangle \), and
- A conflict proof \( c \) with \( H \vdash c : \bot \) is \( H \).

No proof object to check. But the LCF architecture [Mil79, GMW79] can be used to ensure the correctness of answers. LCF in a nutshell:

- A type \textit{theorem} is defined for provable formulae in a module of the prover called \textit{kernel}.

33/49
LCF: answers that are correct-by-construction

Other “proof format”:

- A deduction proof \( j \) with \( H \vdash j : L \) is the pair \( \langle H, L \rangle \), and
- A conflict proof \( c \) with \( H \vdash c : \bot \) is \( H \).

No proof object to check. But the LCF architecture [Mil79, GMW79] can be used to ensure the correctness of answers. LCF in a nutshell:

- A type \texttt{theorem} is defined for provable formulae in a module of the prover called \texttt{kernel}
- The definition of \texttt{theorem} is hidden outside the kernel
LCF: answers that are correct-by-construction

Other “proof format”:

- A deduction proof \( j \) with \( H \vdash j : L \) is the pair \( \langle H, L \rangle \), and
- A conflict proof \( c \) with \( H \vdash c : \bot \) is \( H \).

No proof object to check. But the LCF architecture [Mil79, GMW79] can be used to ensure the correctness of answers. LCF in a nutshell:

- A type theorem is defined for provable formulae in a module of the prover called kernel
- The definition of theorem is hidden outside the kernel
- The kernel exports primitives to construct its inhabitants, e.g. modus_ponens : theorem \( \rightarrow \) theorem \( \rightarrow \) theorem takes as arguments \( F \) and \( G \), checks that \( F \) is of the form \( G \Rightarrow R \), and returns \( R \) as an inhabitant of theorem.
LCF: answers that are correct-by-construction

Other “proof format”:

- A deduction proof \( j \) with \( H \vdash j : L \) is the pair \( \langle H, L \rangle \), and
- A conflict proof \( c \) with \( H \vdash c : \bot \) is \( H \).

No proof object to check. But the LCF architecture [Mil79, GMW79] can be used to ensure the correctness of answers. LCF in a nutshell:

- A type `theorem` is defined for provable formulae in a module of the prover called `kernel`
- The definition of `theorem` is hidden outside the kernel
- The kernel exports primitives to construct its inhabitants, e.g. `modus_ponens : theorem \rightarrow theorem \rightarrow theorem` takes as arguments \( F \) and \( G \), checks that \( F \) is of the form \( G \Rightarrow R \), and returns \( R \) as an inhabitant of `theorem`.
- Search procedures can be programmed using the primitives.
LCF: answers that are correct-by-construction

Other “proof format”:

- A deduction proof $j$ with $H \vdash j : L$ is the pair $\langle H, L \rangle$, and
- A conflict proof $c$ with $H \vdash c : \bot$ is $H$.

No proof object to check. But the LCF architecture [Mil79, GMW79] can be used to ensure the correctness of answers. LCF in a nutshell:

- A type `theorem` is defined for provable formulae in a module of the prover called `kernel`
- The definition of `theorem` is hidden outside the kernel
- The kernel exports primitives to construct its inhabitants, e.g. `modus_ponens : theorem -> theorem -> theorem` takes as arguments $F$ and $G$, checks that $F$ is of the form $G \Rightarrow R$, and returns $R$ as an inhabitant of `theorem`.
- Search procedures can be programmed using the primitives.
- Bugs in these procedures cannot jeopardise the property that any inhabitant of `theorem` is provable, if kernel is trusted.
LCF: answers that are correct-by-construction

Other “proof format”:

- A deduction proof \( j \) with \( H \vdash j : L \) is the pair \( \langle H, L \rangle \), and
- A conflict proof \( c \) with \( H \vdash c : \bot \) is \( H \).

No proof object to check. But the LCF architecture [Mil79, GMW79] can be used to ensure the correctness of answers. LCF in a nutshell:

- A type `theorem` is defined for provable formulae in a module of the prover called `kernel`
- The definition of `theorem` is hidden outside the kernel
- The kernel exports primitives to construct its inhabitants, e.g. `modus_ponens : theorem -> theorem -> theorem` takes as arguments \( F \) and \( G \), checks that \( F \) is of the form \( G \Rightarrow R \), and returns \( R \) as an inhabitant of `theorem`.
- Search procedures can be programmed using the primitives.
- Bugs in these procedures cannot jeopardise the property that any inhabitant of `theorem` is provable, if kernel is trusted

No proof object needs to be built in memory
CDSAT is well-suited to the LCF approach 1/2

Given a type `assign` for multiple assignments and `single_assign` for singleton assignments, a trusted kernel defines

\[
\text{type deduction} = \text{assign} \ast \text{single_assign} \\
\text{type conflict} = \text{assign}
\]

and exports

\[
\text{type deduction} \\
\text{type conflict} \\
in : \text{single_assign} \rightarrow \text{deduction} \\
\text{coerc} : \text{'k theory_handler} \\
\hspace{1cm} \rightarrow \text{'k theory_proof} \rightarrow \text{deduction} \\
\text{lem} : \text{conflict} \rightarrow \text{assign} \rightarrow \text{deduction} \\
\text{cfl} : \text{'k theory_handler} \\
\hspace{1cm} \rightarrow \text{'k theory_proof} \rightarrow \text{conflict} \\
\text{res} : \text{deduction} \rightarrow \text{conflict} \rightarrow \text{conflict}
\]
CDSAT is well-suited to the LCF approach 2/2

If the empty assignment is constructed in type `conflict`, input problem is guaranteed to be unsat, provided the kernel primitives and the implementation of theory proofs are trusted (code for the search plan does not have to be certified)
CDSAT is well-suited to the LCF approach 2/2

If the empty assignment is constructed in type conflict, input problem is guaranteed to be unsat, provided the kernel primitives and the implementation of theory proofs are trusted (code for the search plan does not have to be certified)

Answer is correct-by-construction, no proof object in memory.
4. Quantified satisfiability
Quantifiers

Due to the connection between MCSAT and quantifier elimination, we recently explored how MCSAT features could be used to extend Yices to support quantifiers.

*quantifier elimination*: For any formula $A$, there exists a quantifier-free formula $B$ such that $\lceil A \rceil = \lceil B \rceil$.
Quantifiers

Due to the connection between MCSAT and quantifier elimination, we recently explored how MCSAT features could be used to extend Yices to support quantifiers.

quantifier elimination: For any formula $A$, there exists a quantifier-free formula $B$ such that $\llbracket A \rrbracket = \llbracket B \rrbracket$

In practice though, if the size of $B$ is way bigger than the size of $A$, it may be unfeasible to compute $B$ or decide whether it is satisfiable.
Due to the connection between MCSAT and quantifier elimination, we recently explored how MCSAT features could be used to extend Yices to support quantifiers.

**Quantifier elimination**: For any formula $A$, there exists a quantifier-free formula $B$ such that $\lbrack A \rbrack = \lbrack B \rbrack$

In practice though, if the size of $B$ is way bigger than the size of $A$, it may be unfeasible to compute $B$ or decide whether it is satisfiable.

We have a better approach that we applied to NRA (non-linear arithmetic) and BV (bitvectors), on top of Yices/MCSAT.
Approximations

Idea: if the only reason to produce $B$ from $A$ is to decide whether $A$ is satisfiable, it may not be necessary to compute $B$ exactly. Approximations may suffice.

Def:

- An over-approximation of $A$ is a quantifier-free formula $O$ with $[A] \subseteq [O]$. If $O$ is unsat., then $A$ is unsat.

- An under-approximation of $A$ is a quantifier-free formula $U$ with $[U] \subseteq [A]$. If $U$ is sat., then $A$ is sat.
Basic idea of lazy quantifier elimination

Start with $U = \text{false}$ and $O = \text{true}$, and iteratively refine $U$ and $O$ until either $U$ is sat or $O$ is unsat.

Worst case: you may end up computing a quantifier-free formula $B$ such that $[A] = [B]$.

In practice, you hope the algorithm will stop earlier than that.

Question: how do we refine the approximations iteratively?
One quantifier at a time

Quantifier elimination:
Given $\exists y F(x, y)$ with quantifier-free $F(x, y)$, produce quantifier-free $B(x)$ with $(\exists y F(x, y)) \iff B(x)$ provable.

Model generalization:
If additionally given $\mathcal{M}$ satisfying $\exists y F(x, y)$, produce quantifier-free $U(x)$ satisfied by $\mathcal{M}$, with $U(x) \Rightarrow (\exists y F(x, y))$ provable.

Model interpolation:
If additionally given $\mathcal{M}$ not satisfying $\exists y F(x, y)$, produce quantifier-free $O(x)$ not satisfied by $\mathcal{M}$, with $(\exists y F(x, y)) \Rightarrow O(x)$ provable.

In blue: $F(x_1, x_2, y)$; its grey shadow: $\exists y F(x, y)$; in red: the under-approximation $U(x_1, x_2)$ / the over-approximation $O(x_1, x_2)$. 
A satisfiability algorithm for a slightly more general question

“Given a formula $A(\overrightarrow{z}, \overrightarrow{x})$ and a model $\mathcal{M}_{\overrightarrow{z}}$ on $\overrightarrow{z}$, produce either

- SAT($U(\overrightarrow{z})$), with $U(\overrightarrow{z})$ under-approx. of $\exists \overrightarrow{x} \ A(\overrightarrow{z}, \overrightarrow{x})$ satisfied by $\mathcal{M}_{\overrightarrow{z}}$; or
- UNSAT($O(\overrightarrow{z})$), with $O(\overrightarrow{z})$ over-approx. of $\exists \overrightarrow{x} \ A(\overrightarrow{z}, \overrightarrow{x})$ not satisfied by $\mathcal{M}_{\overrightarrow{z}}$."

(i.e. $\overrightarrow{z}$’s values are imposed, $\overrightarrow{x}$ are existentially quantified: values are up to us).
A satisfiability algorithm for a slightly more general question

“Given a formula $A(\vec{z}, \vec{x})$ and a model $\mathcal{M}_{\vec{z}}$ on $\vec{z}$, produce either

- $\text{SAT}(U(\vec{z}))$, with $U(\vec{z})$ under-approx. of $\exists \vec{x} A(\vec{z}, \vec{x})$ satisfied by $\mathcal{M}_{\vec{z}}$; or
- $\text{UNSAT}(O(\vec{z}))$, with $O(\vec{z})$ over-approx. of $\exists \vec{x} A(\vec{z}, \vec{x})$ not satisfied by $\mathcal{M}_{\vec{z}}$.”

(i.e. $\vec{z}$’s values are imposed, $\vec{x}$ are existentially quantified: values are up to us).

This generalizes the standard satisfiability question:

“Given a formula $A(\vec{x})$, produce either

- $\text{SAT}$, if $\exists \vec{x} A(\vec{x})$ is satisfied by the empty model (does not assign any value to any variable); or
- $\text{UNSAT}$, if not.”

If you have an algorithm to solve the more general problem, apply it on the empty model $\mathcal{M}$ and $A(\vec{x})$ ($\vec{z}$ is empty) and inspect the result:

- $\text{UNSAT}(O)$: return UNSAT
- $\text{SAT}(U)$: return SAT
The (recursive) satisfiability algorithm is a 2-player game

“Given a formula $A(\vec{z}, \vec{x})$ and a model $\mathcal{M}_{\vec{z}}$ on $\vec{z}$, produce either

- $\text{SAT}(U(\vec{z}))$, with $U(\vec{z})$ under-approx. of $\exists \vec{x} \ A(\vec{z}, \vec{x})$ satisfied by $\mathcal{M}_{\vec{z}}$; or
- $\text{UNSAT}(O(\vec{z}))$, with $O(\vec{z})$ over-approx. of $\exists \vec{x} \ A(\vec{z}, \vec{x})$ not satisfied by $\mathcal{M}_{\vec{z}}$.”
The (recursive) satisfiability algorithm is a 2-player game

"Given a formula $A(\vec{z}, \vec{x})$ and a model $\mathcal{M}_{\vec{z}}$ on $\vec{z}$, produce either

- $\text{SAT}(U(\vec{z}))$, with $U(\vec{z})$ under-approx. of $\exists \vec{x} A(\vec{z}, \vec{x})$ satisfied by $\mathcal{M}_{\vec{z}}$; or
- $\text{UNSAT}(O(\vec{z}))$, with $O(\vec{z})$ over-approx. of $\exists \vec{x} A(\vec{z}, \vec{x})$ not satisfied by $\mathcal{M}_{\vec{z}}$.

**Algorithm solve:**

(0) If $A(\vec{z}, \vec{x})$ is q-f, ask **whether $\mathcal{M}_{\vec{z}}$ extends to a model $\mathcal{M}$ of $A(\vec{z}, \vec{x})$**.

- If not, apply **model interpolation** on $\mathcal{M}_{\vec{z}}$ and $A(\vec{z}, \vec{x})$ to get $O(\vec{z})$; return $\text{UNSAT}(O(\vec{z}))$.

- Otherwise, apply **model generalization** on $\mathcal{M}$ and $A(\vec{z}, \vec{x})$ to get $U(\vec{z})$; return $\text{SAT}(U(\vec{z}))$.\"
The (recursive) satisfiability algorithm is a 2-player game “Given a formula \(A(\overrightarrow{z}, \overrightarrow{x})\) and a model \(M_{\overrightarrow{z}}\) on \(\overrightarrow{z}\), produce either

- \(\text{SAT}(U(\overrightarrow{z})),\) with \(U(\overrightarrow{z})\) under-approx. of \(\exists \overrightarrow{x} \ A(\overrightarrow{z}, \overrightarrow{x})\) satisfied by \(M_{\overrightarrow{z}}\); or
- \(\text{UNSAT}(O(\overrightarrow{z})),\) with \(O(\overrightarrow{z})\) over-approx. of \(\exists \overrightarrow{x} \ A(\overrightarrow{z}, \overrightarrow{x})\) not satisfied by \(M_{\overrightarrow{z}}\).”

Algorithm solve:

0) If \(A(\overrightarrow{z}, \overrightarrow{x})\) is q-f, ask whether \(M_{\overrightarrow{z}}\) extends to a model \(M\) of \(A(\overrightarrow{z}, \overrightarrow{x})\).
   - If not, apply model interpolation on \(M_{\overrightarrow{z}}\) and \(A(\overrightarrow{z}, \overrightarrow{x})\) to get \(O(\overrightarrow{z})\);
     return UNSAT\((O(\overrightarrow{z}))\).
   - Otherwise, apply model generalization on \(M\) and \(A(\overrightarrow{z}, \overrightarrow{x})\) to get \(U(\overrightarrow{z})\);
     return SAT\((U(\overrightarrow{z}))\).

1) If \(A(\overrightarrow{z}, \overrightarrow{x})\) is not q-f, rewrite it as \(F(\overrightarrow{z}, \overrightarrow{x}) \land \neg \exists \overrightarrow{y} \ A_{\text{rec}}(\overrightarrow{z}, \overrightarrow{x}, \overrightarrow{y}),\) where \(F(\overrightarrow{z}, \overrightarrow{x})\) is q-f
The (recursive) satisfiability algorithm is a 2-player game

“Given a formula $A(\overrightarrow{z}, \overrightarrow{x})$ and a model $M_{\overrightarrow{z}}$ on $\overrightarrow{z}$, produce either

- $\text{SAT}(U(\overrightarrow{z}))$, with $U(\overrightarrow{z})$ under-approx. of $\exists \overrightarrow{x} A(\overrightarrow{z}, \overrightarrow{x})$ satisfied by $M_{\overrightarrow{z}}$; or
- $\text{UNSAT}(O(\overrightarrow{z}))$, with $O(\overrightarrow{z})$ over-approx. of $\exists \overrightarrow{x} A(\overrightarrow{z}, \overrightarrow{x})$ not satisfied by $M_{\overrightarrow{z}}$.”

Algorithm $\text{solve}$:

(0) If $A(\overrightarrow{z}, \overrightarrow{x})$ is q-f, ask whether $M_{\overrightarrow{z}}$ extends to a model $M$ of $A(\overrightarrow{z}, \overrightarrow{x})$.

- If not, apply model interpolation on $M_{\overrightarrow{z}}$ and $A(\overrightarrow{z}, \overrightarrow{x})$ to get $O(\overrightarrow{z})$; return $\text{UNSAT}(O(\overrightarrow{z}))$.
- Otherwise, apply model generalization on $M$ and $A(\overrightarrow{z}, \overrightarrow{x})$ to get $U(\overrightarrow{z})$; return $\text{SAT}(U(\overrightarrow{z}))$.

(1) If $A(\overrightarrow{z}, \overrightarrow{x})$ is not q-f, rewrite it as $F(\overrightarrow{z}, \overrightarrow{x}) \land \neg \exists \overrightarrow{y} A_{\text{rec}}(\overrightarrow{z}, \overrightarrow{x}, \overrightarrow{y})$, where $F(\overrightarrow{z}, \overrightarrow{x})$ is q-f

(2) Set $L(\overrightarrow{z}, \overrightarrow{x}) := F(\overrightarrow{z}, \overrightarrow{x})$ as an over-approx. of $A(\overrightarrow{z}, \overrightarrow{x})$ to be refined.
The (recursive) satisfiability algorithm is a 2-player game

“Given a formula $A(\overrightarrow{z}, \overrightarrow{x})$ and a model $\mathcal{M}_{\overrightarrow{z}}$ on $\overrightarrow{z}$, produce either

- $\text{SAT}(U(\overrightarrow{z}))$, with $U(\overrightarrow{z})$ under-approx. of $\exists \overrightarrow{x} A(\overrightarrow{z}, \overrightarrow{x})$ satisfied by $\mathcal{M}_{\overrightarrow{z}}$; or
- $\text{UNSAT}(O(\overrightarrow{z}))$, with $O(\overrightarrow{z})$ over-approx. of $\exists \overrightarrow{x} A(\overrightarrow{z}, \overrightarrow{x})$ not satisfied by $\mathcal{M}_{\overrightarrow{z}}$.”

**Algorithm solve:**

1. If $A(\overrightarrow{z}, \overrightarrow{x})$ is q-f, ask whether $\mathcal{M}_{\overrightarrow{z}}$ extends to a model $\mathcal{M}$ of $A(\overrightarrow{z}, \overrightarrow{x})$.
   - If not, apply model interpolation on $\mathcal{M}_{\overrightarrow{z}}$ and $A(\overrightarrow{z}, \overrightarrow{x})$ to get $O(\overrightarrow{z})$; return $\text{UNSAT}(O(\overrightarrow{z}))$.
   - Otherwise, apply model generalization on $\mathcal{M}$ and $A(\overrightarrow{z}, \overrightarrow{x})$ to get $U(\overrightarrow{z})$; return $\text{SAT}(U(\overrightarrow{z}))$.

2. If $A(\overrightarrow{z}, \overrightarrow{x})$ is not q-f, rewrite it as $F(\overrightarrow{z}, \overrightarrow{x}) \land \neg \exists \overrightarrow{y} A_{\text{rec}}(\overrightarrow{z}, \overrightarrow{x}, \overrightarrow{y})$, where $F(\overrightarrow{z}, \overrightarrow{x})$ is q-f.

3. Set $L(\overrightarrow{z}, \overrightarrow{x}) := F(\overrightarrow{z}, \overrightarrow{x})$ as an over-approx. of $A(\overrightarrow{z}, \overrightarrow{x})$ to be refined.

4. Ask whether $\mathcal{M}_{\overrightarrow{z}}$ extends to a model $\mathcal{M}$ of $L(\overrightarrow{z}, \overrightarrow{x})$. 
The (recursive) satisfiability algorithm is a 2-player game:

"Given a formula $A(\vec{z}, \vec{x})$ and a model $M_{\vec{z}}$ on $\vec{z}$, produce either

- $\text{SAT}(U(\vec{z}))$, with $U(\vec{z})$ under-approx. of $\exists \vec{x} A(\vec{z}, \vec{x})$ satisfied by $M_{\vec{z}}$; or
- $\text{UNSAT}(O(\vec{z}))$, with $O(\vec{z})$ over-approx. of $\exists \vec{x} A(\vec{z}, \vec{x})$ not satisfied by $M_{\vec{z}}$.

Algorithm $\text{solve}$:

1. If $A(\vec{z}, \vec{x})$ is q-f, ask whether $M_{\vec{z}}$ extends to a model $M$ of $A(\vec{z}, \vec{x})$.
   - If not, apply model interpolation on $M_{\vec{z}}$ and $A(\vec{z}, \vec{x})$ to get $O(\vec{z})$; return $\text{UNSAT}(O(\vec{z}))$.
   - Otherwise, apply model generalization on $M$ and $A(\vec{z}, \vec{x})$ to get $U(\vec{z})$; return $\text{SAT}(U(\vec{z}))$.

2. If $A(\vec{z}, \vec{x})$ is not q-f, rewrite it as $F(\vec{z}, \vec{x}) \land \neg \exists \vec{y} A_{\text{rec}}(\vec{z}, \vec{x}, \vec{y})$, where $F(\vec{z}, \vec{x})$ is q-f.

3. Set $L(\vec{z}, \vec{x}) := F(\vec{z}, \vec{x})$ as an over-approx. of $A(\vec{z}, \vec{x})$ to be refined.

4. Ask whether $M_{\vec{z}}$ extends to a model $M$ of $L(\vec{z}, \vec{x})$.
   - If not, apply model interpolation on $M_{\vec{z}}$ and $L(\vec{z}, \vec{x})$ to get $O(\vec{z})$; return $\text{UNSAT}(O(\vec{z}))$.
The (recursive) satisfiability algorithm is a 2-player game

“Given a formula $A(\overrightarrow{z}, \overrightarrow{x})$ and a model $M_{\overrightarrow{z}}$ on $\overrightarrow{z}$, produce either

- SAT$(U(\overrightarrow{z}))$, with $U(\overrightarrow{z})$ under-approx. of $\exists \overrightarrow{x} A(\overrightarrow{z}, \overrightarrow{x})$ satisfied by $M_{\overrightarrow{z}}$; or
- UNSAT$(O(\overrightarrow{z}))$, with $O(\overrightarrow{z})$ over-approx. of $\exists \overrightarrow{x} A(\overrightarrow{z}, \overrightarrow{x})$ not satisfied by $M_{\overrightarrow{z}}$.”

Algorithm solve:

0. If $A(\overrightarrow{z}, \overrightarrow{x})$ is q-f, ask whether $M_{\overrightarrow{z}}$ extends to a model $M$ of $A(\overrightarrow{z}, \overrightarrow{x})$.

  ▶ If not, apply model interpolation on $M_{\overrightarrow{z}}$ and $A(\overrightarrow{z}, \overrightarrow{x})$ to get $O(\overrightarrow{z})$; return UNSAT$(O(\overrightarrow{z}))$.
  ▶ Otherwise, apply model generalization on $M$ and $A(\overrightarrow{z}, \overrightarrow{x})$ to get $U(\overrightarrow{z})$; return SAT$(U(\overrightarrow{z}))$.

1. If $A(\overrightarrow{z}, \overrightarrow{x})$ is not q-f, rewrite it as $F(\overrightarrow{z}, \overrightarrow{x}) \land \neg \exists \overrightarrow{y} A_{rec}(\overrightarrow{z}, \overrightarrow{x}, \overrightarrow{y})$, where $F(\overrightarrow{z}, \overrightarrow{x})$ is q-f.

2. Set $L(\overrightarrow{z}, \overrightarrow{x}) := F(\overrightarrow{z}, \overrightarrow{x})$ as an over-approx. of $A(\overrightarrow{z}, \overrightarrow{x})$ to be refined.

3. Ask whether $M_{\overrightarrow{z}}$ extends to a model $M$ of $L(\overrightarrow{z}, \overrightarrow{x})$.

  ▶ If not, apply model interpolation on $M_{\overrightarrow{z}}$ and $L(\overrightarrow{z}, \overrightarrow{x})$ to get $O(\overrightarrow{z})$; return UNSAT$(O(\overrightarrow{z}))$.
  ▶ Otherwise, recursively call solve on $M$ and $A_{rec}(\overrightarrow{z}, \overrightarrow{x}, \overrightarrow{y})$, and inspect the result:
    ▶ UNSAT$(O_{rec}(\overrightarrow{z}, \overrightarrow{x}))$ apply model generalization on $M$ and $F(\overrightarrow{z}, \overrightarrow{x}) \land \neg O_{rec}(\overrightarrow{z}, \overrightarrow{x})$ to get $U(\overrightarrow{z})$; return SAT$(U(\overrightarrow{z}))$.\[\]
The (recursive) satisfiability algorithm is a 2-player game

"Given a formula \(A(\overrightarrow{z}, \overrightarrow{x})\) and a model \(M_{\overrightarrow{z}}\) on \(\overrightarrow{z}\), produce either

- \(\text{SAT}(U(\overrightarrow{z}))\), with \(U(\overrightarrow{z})\) under-approx. of \(\exists \overrightarrow{x} A(\overrightarrow{z}, \overrightarrow{x})\) satisfied by \(M_{\overrightarrow{z}}\); or
- \(\text{UNSAT}(O(\overrightarrow{z}))\), with \(O(\overrightarrow{z})\) over-approx. of \(\exists \overrightarrow{x} A(\overrightarrow{z}, \overrightarrow{x})\) not satisfied by \(M_{\overrightarrow{z}}\)."

**Algorithm solve:**

1. If \(A(\overrightarrow{z}, \overrightarrow{x})\) is q-f, ask whether \(M_{\overrightarrow{z}}\) extends to a model \(M\) of \(A(\overrightarrow{z}, \overrightarrow{x})\).
   - If not, apply model interpolation on \(M_{\overrightarrow{z}}\) and \(A(\overrightarrow{z}, \overrightarrow{x})\) to get \(O(\overrightarrow{z})\); return \(\text{UNSAT}(O(\overrightarrow{z}))\).
   - Otherwise, apply model generalization on \(M\) and \(A(\overrightarrow{z}, \overrightarrow{x})\) to get \(U(\overrightarrow{z})\); return \(\text{SAT}(U(\overrightarrow{z}))\).

2. If \(A(\overrightarrow{z}, \overrightarrow{x})\) is not q-f, rewrite it as \(F(\overrightarrow{z}, \overrightarrow{x}) \land \neg \exists \overrightarrow{y} A_{\text{rec}}(\overrightarrow{z}, \overrightarrow{x}, \overrightarrow{y})\), where \(F(\overrightarrow{z}, \overrightarrow{x})\) is q-f

3. Set \(L(\overrightarrow{z}, \overrightarrow{x}) := F(\overrightarrow{z}, \overrightarrow{x})\) as an over-approx. of \(A(\overrightarrow{z}, \overrightarrow{x})\) to be refined

4. Ask whether \(M_{\overrightarrow{z}}\) extends to a model \(M\) of \(L(\overrightarrow{z}, \overrightarrow{x})\).
   - If not, apply model interpolation on \(M_{\overrightarrow{z}}\) and \(L(\overrightarrow{z}, \overrightarrow{x})\) to get \(O(\overrightarrow{z})\); return \(\text{UNSAT}(O(\overrightarrow{z}))\).
   - Otherwise, recursively call solve on \(M\) and \(A_{\text{rec}}(\overrightarrow{z}, \overrightarrow{x}, \overrightarrow{y})\), and inspect the result:
     - \(\text{UNSAT}(O_{\text{rec}}(\overrightarrow{z}, \overrightarrow{x}))\) apply model generalization on \(M\) and \(F(\overrightarrow{z}, \overrightarrow{x}) \land \neg O_{\text{rec}}(\overrightarrow{z}, \overrightarrow{x})\) to get \(U(\overrightarrow{z})\); return \(\text{SAT}(U(\overrightarrow{z}))\).
     - \(\text{SAT}(U_{\text{rec}}(\overrightarrow{z}, \overrightarrow{x}))\)
     Set \(L(\overrightarrow{z}, \overrightarrow{x}) := L(\overrightarrow{z}, \overrightarrow{x}) \land \neg U_{\text{rec}}(\overrightarrow{z}, \overrightarrow{x})\) and go back to (3).
How to answer the 3 kinds of queries

**Model extension:** Does model $\mathcal{M}$ on $\vec{x}$ extend to a model of a q-f formula $L(\vec{x}, \vec{y})$?

**Model generalization**

**Model interpolation**

It depends on the theory $T$. At SRI, we have implemented those procedures for:

- the *Booleans*, the theory of *bitvectors, real arithmetic* (linear and non-linear).

In those theories, we can apply procedure `solve` to lazily eliminate quantifiers in the view of determining satisfiability of any formula.

- Model generalization techniques already widely used in the field.
- Model extension not too difficult to achieve using regular SMT constraints.
- Model interpolation based on **MCSAT**.
Implementation and related works

SRI’s Yices SMT-solver [https://yices.csl.sri.com/] for quantifier-free formulas offers an API that includes `check-with-model`, `model-interpolant`, and `generalize-model`.

See also related works:

▶ Bjørner and Janota’s algorithm for “playing with quantified satisfaction”, inspired by QBF [BJ15] and used in z3. A two-player game (one wanting to satisfy $A$, the other one $\neg A$). Based on model projection and unsat cores, but no model interpolation used.

▶ Monniaux’s work on quantifier elimination [Mon08, Mon10]. It uses a ground SMT-solver as a black box (for purely existential problems), and also performs some QE-elimination steps (e.g., FM resolutions) independently from the SMT-solver.

▶ The ANR Decert work on Linear Integer arithmetic, which extends Fourier-Motzkin with simplex-based techniques [BCC12].
Implementation and related works


The solving algorithm is implemented in an OCaml solver called YicesQS (for Quantified Satisfaction): [https://github.com/disteph/yicesQS](https://github.com/disteph/yicesQS)
using the new `yices2_ocaml_bindings`
[https://github.com/SRI-CSL/yices2_ocaml_bindings](https://github.com/SRI-CSL/yices2_ocaml_bindings)
that can be used to query Yices via its C API from OCaml programs.
Implementation and related works

SRI’s Yices SMT-solver [https://yices.csl.sri.com/] for quantifier-free formulas offers an API that includes check-with-model, model-interpolant, and generalize-model.

The solving algorithm is implemented in an OCaml solver called YicesQS (for Quantified Satisfaction): [https://github.com/disteph/yicesQS](https://github.com/disteph/yicesQS) using the new [yices2_ocaml_bindings](https://github.com/SRI-CSL/yices2_ocaml_bindings) that can be used to query Yices via its C API from OCaml programs.

See also related works:

- Bjørner and Janota’s algorithm for “playing with quantified satisfaction”, inspired by QBF [BJ15] and used in z3. A two-player game (one wanting to satisfy $A$, the other one $\neg A$). Based on model projection and unsat cores, but no model interpolation used.

- Monniaux’s work on quantifier elimination [Mon08, Mon10]. It uses a ground SMT-solver as a black box (for purely existential problems), and also performs some QE-elimination steps (e.g., FM resolutions) independently from the SMT-solver.

- The ANR Decert work on Linear Integer arithmetic, which extends Fourier-Motzkin with simplex-based techniques [BCC+12]
```ocaml
let sat_answer x game reason =
let open Game in
let model = match x with
| `over -> Context.get_model game.context_over ~keep_subst:true
| `under -> Context.get_model game.context_under ~keep_subst:true
in
let true_of_model = Term.(reason &&& game.ground) in
let gen_model = Model.generalize_model model true_of_model game.newvars `YICES_GEN_DEFAULT
in
Term.(andN gen_model)

let rec solve game model =
match Context.check_with_model game.context_over model.model model.model.model.support with
| `STATUS_UNSAT ->
  let interpolant = Context.get_model_interpolant game.context_over in
  Sat Term.(not1 interpolant)
| `STATUS_SAT ->
  let newmodel = Context.get_model game.context_over ~keep_subst:true in
  let rec under_solve = function
  | [] -> None
  | under_i::tail ->
    Context.push game.context_under;
    Context.assert_formula game.context_under_under_i;
    match Context.check_with_model game.context_under_and model.model model.model.support with
    | `STATUS_UNSAT -> Context.pop game.context_under; under_solve tail
    | `STATUS_SAT ->
      let term = sat_answer `under game under_i in
      Context.pop game.context_under;
      Some term
  in
  begin
    match under_solve !(game.under) with
    | Some term -> Sat term
    | None ->
      let rec aux reasons = function
      | [] ->
        let reason = Term.(andN reasons in
        if not(List.is_empty reasons) then game.under := reason::!(game.under);
        Sat(sat_answer `over game reason)
        | (u,_)::opponents when not (Model.get_bool_value newmodel u)
          -> aux (Term.(not1 u::reasons) opponents
        | (u,opponent)::opponents ->
          let recur = solve opponent { support = opponent.rigid; model = newmodel } in
          match recur with
          | Unsat reason -> aux (reason::reasons) opponents
          | Sat reason ->
            let learnt = Term.(u == not1 reason) in
            Context.assert_formula game.context_over learnt;
            Context.assert_formula game.context_under learnt;
            (* Not necessary; useful? *)
            game.over := learnt::!(game.over);
            solve game model
          in
          aux [] game.foralls
  end
```
Termination of algorithm solve

Even if you can perform model extension/interpolation/generalization for theory $\mathcal{T}$, it is not always the case that this makes algorithm solve terminate: the incremental refinement of the over- and under-approximations may not converge in finite time.
Termination of algorithm solve

Even if you can perform model extension/interpolation/generalization for theory $\mathcal{T}$, it is not always the case that this makes algorithm solve terminate: the incremental refinement of the over- and under-approximations may not converge in finite time.

Fortunately, this is the case for the Booleans and the bitvectors (number of models is finite; incrementally refining approximations will converge).
Termination of algorithm `solve`

Even if you can perform model extension/interpolation/generalization for theory $\mathcal{T}$, it is not always the case that this makes algorithm `solve` terminate: the incremental refinement of the over- and under-approximations may not converge in finite time.

Fortunately, this is the case for the Booleans and the bitvectors (number of models is finite; incrementally refining approximations will converge).

Much less obviously, this is also the case for (linear and non-linear) real arithmetic: approximations will converge, and quantifiers can be eliminated.

**Linear arithmetic:** Fourier-Motzkin,

**Non-linear arithmetic:** cylindrical algebraic decomposition (CAD)
Termination of algorithm \texttt{solve}

Even if you can perform model extension/interpolation/generalization for theory $\mathcal{T}$, it is not always the case that this makes algorithm \texttt{solve} terminate: the incremental refinement of the over- and under-approximations may not converge in finite time.

Fortunately, this is the case for the Booleans and the bitvectors (number of models is finite; incrementally refining approximations \textit{will converge}).

Much less obviously, this is also the case for (linear and non-linear) real arithmetic: approximations will converge, and quantifiers can be eliminated.  
\textit{Linear arithmetic}: Fourier-Motzkin,  
\textit{Non-linear arithmetic}: cylindrical algebraic decomposition (CAD)

All of these theories are decidable (-ish).
Related work and future work

Investigate related approaches:

▶ The ANR Decert work on Linear Integer arithmetic, which extends Fourier-Motzkin with simplex-based techniques [BCC+12]

▶ Monniaux’s work on quantifier elimination [Mon08, Mon10]. It uses a ground SMT-solver as a black box (for purely existential problems), and also performs some QE-elimination steps (e.g., FM resolutions) independently from the SMT-solver.

▶ Dutertre’s work on solving “EF problems” (∃∀) in Yices, also relying on a ground SMT-solver considered as a black box.

How would the Bjørner-Janota approach work in a combination of theories?

Just as our CDSAT system generalises MCSAT to a combination of theories, what would be the equivalent for the Bjørner-Janota approach?
Questions?
A simplex-based extension of fourier-motzkin for solving linear integer arithmetic.

Conflict-driven satisfiability for theory combination: Transition system and completeness.

Submitted.

N. Bjorner and M. Janota.
Playing with quantified satisfaction.
In M. Davis, A. Fehnker, A. McIver, and A. Voronkov, editors, *Proc.*