Proofs in Conflict-Driven Theory Combination

Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar

CPP’2018, 9th January 2018
Context: Satisfiability Modulo Theories (SMT)

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: Satisfiability Modulo Theories (SMT)

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) [BGLS17]
- generalises MCSAT: generic combinations of abstract theories
- can also use first-order assignments
- models theory reasoning with modules made of inference rules
Context: Satisfiability Modulo Theories (SMT)

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) [BGLS17]
- 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: Satisfiability Modulo Theories (SMT)

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) [BGLS17]
- 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

This paper concerns the dual situation of unsatisfiable formulae: there exists a proof (of the formula’s negation)
Questions addressed

- Which information does CDSAT need to record, during a run, in order to justify an answer “unsat” by a proof?
Questions addressed

- Which information does CDSAT need to record, during a run, in order to justify an answer “unsat” by a proof?
- Is the production of a proof by CDSAT tied to a particular proof format?
Questions addressed

- Which information does CDSAT need to record, during a run, in order to justify an answer “unsat” by a proof?
- Is the production of a proof by CDSAT tied to a particular proof format?
- Can we trust a CDSAT implementation to produce correct answers “unsat” without building proofs in memory? If so which parts of the implementation are critical (i.e., can affect the correctness of an answer “unsat”)?

CADE’2017 version of CDSAT: no clause learning mechanism
By design: simpler to present
+ emphasis that learning is not needed for completeness
Questions addressed

- Which information does CDSAT need to record, during a run, in order to justify an answer “unsat” by a proof?
- Is the production of a proof by CDSAT tied to a particular proof format?
- Can we trust a CDSAT implementation to produce correct answers “unsat” without building proofs in memory? If so which parts of the implementation are critical (i.e., can affect the correctness of an answer “unsat”)?
- Is the issue of producing proofs, or correct answers “unsat”, related to learning mechanisms, as in pure SAT-solving?
Questions addressed

- Which information does CDSAT need to record, during a run, in order to justify an answer “unsat” by a proof?
- Is the production of a proof by CDSAT tied to a particular proof format?
- Can we trust a CDSAT implementation to produce correct answers “unsat” without building proofs in memory? If so which parts of the implementation are critical (i.e., can affect the correctness of an answer “unsat”)?
- Is the issue of producing proofs, or correct answers “unsat”, related to learning mechanisms, as in pure SAT-solving?
- Actually, is there a learning mechanism on CDSAT?
Questions addressed

- Which information does CDSAT need to record, during a run, in order to justify an answer “unsat” by a proof?
- Is the production of a proof by CDSAT tied to a particular proof format?
- Can we trust a CDSAT implementation to produce correct answers “unsat” without building proofs in memory? If so which parts of the implementation are critical (i.e., can affect the correctness of an answer “unsat”)?
- Is the issue of producing proofs, or correct answers “unsat”, related to learning mechanisms, as in pure SAT-solving?
- Actually, is there a learning mechanism on CDSAT?

CADE’2017 version of CDSAT: no clause learning mechanism
Questions addressed

▶ Which information does CDSAT need to record, during a run, in order to justify an answer “unsat” by a proof?
▶ Is the production of a proof by CDSAT tied to a particular proof format?
▶ Can we trust a CDSAT implementation to produce correct answers “unsat” without building proofs in memory? If so which parts of the implementation are critical (i.e., can affect the correctness of an answer “unsat”)?
▶ Is the issue of producing proofs, or correct answers “unsat”, related to learning mechanisms, as in pure SAT-solving?
▶ Actually, is there a learning mechanism on CDSAT?

CADE’2017 version of CDSAT: no clause learning mechanism
By design: simpler to present
+ emphasis that learning is not needed for completeness
Questions addressed

- Which information does CDSAT need to record, during a run, in order to justify an answer “unsat” by a proof?
- Is the production of a proof by CDSAT tied to a particular proof format?
- Can we trust a CDSAT implementation to produce correct answers “unsat” without building proofs in memory? If so which parts of the implementation are critical (i.e., can affect the correctness of an answer “unsat”)?
- Is the issue of producing proofs, or correct answers “unsat”, related to learning mechanisms, as in pure SAT-solving?
- Actually, is there a learning mechanism on CDSAT?

CADE’2017 version of CDSAT: no clause learning mechanism
By design: simpler to present
  + emphasis that learning is not needed for completeness

Here, we start by adding learning mechanisms to CDSAT.
Conflict-driven theory combination

The CDSAT system - with learning

Proof production
1. Conflict-driven theory combination
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
- \( a \Rightarrow b \)
- \( b \Rightarrow a \)
- \( a \Rightarrow b \)
- \( b \Rightarrow a \)
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:

- \( a \Rightarrow b \)
- \( b \Rightarrow a \)
- \( a \Rightarrow b \)
- \( b \Rightarrow a \)

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.

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

model building

proof building

UNSAT player
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 ⇒ b
b ⇒ a
```

SAT player

UNSAT player

Model building

Proof building

Conflict
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.

---

**Diagram:**

- **SAT player**
  - Decision making, propagations
  - Model building

- **UNSAT player**
  - Proof building
  - Backjumping, conflict analysis
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.

\[
\begin{align*}
& a \Rightarrow b \\
& b \Rightarrow \overline{a} \\
& \overline{a} \Rightarrow \overline{b} \\
& \overline{b} \Rightarrow a
\end{align*}
\]
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

\[
\begin{align*}
    a & \Rightarrow b \\
    b & \Rightarrow \overline{a} \\
    \overline{a} & \Rightarrow \overline{b} \\
    \overline{b} & \Rightarrow a \\
\end{align*}
\]
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

\[
\begin{align*}
a &\Rightarrow b \\
b &\Rightarrow \overline{a} \\
\overline{a} &\Rightarrow \overline{b} \\
\overline{b} &\Rightarrow a
\end{align*}
\]
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.

\[
\begin{align*}
a & \Rightarrow b \\
b & \Rightarrow \overline{a} \\
\overline{a} & \Rightarrow \overline{b} \\
\overline{b} & \Rightarrow a \\
\overline{a} &
\end{align*}
\]
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.

\[
\begin{align*}
    a & \Rightarrow b \\
    b & \Rightarrow \overline{a} \\
    \overline{a} & \Rightarrow \overline{b} \\
    \overline{b} & \Rightarrow a \\
    \overline{a} & \quad \text{(unsat player)}
\end{align*}
\]
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

\[
\begin{align*}
  a &\Rightarrow b \\
  b &\Rightarrow \overline{a} \\
  \overline{a} &\Rightarrow \overline{b} \\
  \overline{b} &\Rightarrow a \\
  \overline{a}
\end{align*}
\]
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

ea conflict occurs when a clause is falsified

\[ a \Rightarrow b \]
\[ b \Rightarrow \overline{a} \]
\[ \overline{a} \Rightarrow \overline{b} \]
\[ \overline{b} \Rightarrow a \]
\[ \overline{a} \]
\[ \bot \]
Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).
Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).

- **Guess** a value, e.g., \( y \leftarrow 0 \)
Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).

- **Guess** a value, e.g., \( y \leftarrow 0 \)
  - Then \( \mathcal{A}_0 \) yields lower bound \( x > 0 \)
Conflict-driven reasoning can be used for (other) theories

\[ l_0 \quad \begin{array}{l} \quad (-2 \cdot x - y < 0), \\ \quad l_1 \quad (x + y < 0), \\ \quad l_2 \quad (x < -1) \end{array} \]

unsatisfiable in Linear Rational Arithmetic (LRA).

- **Guess** a value, e.g., \( y \leftarrow 0 \)
  - Then \( l_0 \) yields lower bound \( x > 0 \)
  - Together with \( l_2 \), range of possible values for \( x \) is empty
  - What to do? just undo \( y \leftarrow 0 \) and remember that \( y \neq 0 \)?
  - \( \text{No!} \) Clash of bounds suggests a better conflict explanation, by inferring \( l_0 + l_2 \), i.e., \( l_3 \)
  - \( l_3 \quad (\neg y < -2) \)
  - It rules out \( y \leftarrow 0 \), but also many values that would fail for the same reasons.
  - \( \text{Now undo the guess but keep } l_3. \)
  - and so on... (when there is no guess to undo, problem is UNSAT)
Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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

- **No!** Clash of bounds suggests a better conflict explanation,
  by inferring \( l_0 + 2l_2 \), i.e., \( (−y < −2) \)
  It rules out \( y \leftarrow 0 \),
  but also many values that would fail for the same reasons.
Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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

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

- Now undo the guess but keep \( l_3 \).
Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).

- **Guess** a value, e.g., \( y \leftarrow 0 \)
  - Then \( l_0 \) yields lower bound \( x > 0 \)
  - Together with \( l_2 \), range of possible values for \( x \) is empty
  - What to do? just undo \( y \leftarrow 0 \) and remember that \( y \neq 0 \)?
- **No!** Clash of bounds suggests a better conflict explanation,
  \[ l_3 \]

by inferring \( l_0 + 2l_2 \), i.e., \( (-y < -2) \)

- It rules out \( y \leftarrow 0 \),
  - but also many values that would fail for the same reasons.
- **Now undo the guess but keep** \( l_3 \).
- **and so on...**
  (when there is no guess to undo, problem is 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
  \( y \leftarrow 3/4 \)
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
  - $y \leftarrow 3/4$

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:
  \[(t_1 < x), (x < t_2) \vdash t_1 < t_2\]
2. The CDSAT system - with learning
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

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

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 \( \text{Bool} \) and a truth value \( b \))

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

A set of inferences of the form

\[(t_1 \leftarrow c_1), \ldots, (t_k \leftarrow c_k) \vdash^T (l \leftarrow b)\]

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 \(\text{Bool}\) 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:

\[(t_1 \leftarrow c_1), \ldots, (t_k \leftarrow c_k) \models (l \leftarrow b)\]
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:**
\[(x \leftarrow \sqrt{2}), (y \leftarrow \sqrt{2}) \vdash_{\text{NLRA}} (x \cdot y \simeq 2)\] (evaluation inference)
\[(l_1 \lor \cdots \lor l_n), \overline{l_1} \cdots, \overline{l_{n-1}} \vdash_{\text{Bool}} l_n\] (unit propagation)
What is a theory module? (Equality inferences)

All theory modules have the equality inferences:

\[
\begin{align*}
& t_1 \leftarrow c_1, t_2 \leftarrow c_2 \vdash t_1 \simeq t_2 & \text{if } c_1 \text{ and } c_2 \text{ are the same value} \\
& t_1 \leftarrow c_1, t_2 \leftarrow c_2 \vdash t_1 \not\simeq t_2 & \text{if } c_1 \text{ and } c_2 \text{ are distinct values} \\
& \vdash t_1 \simeq t_1 & \text{reflexivity} \\
& t_1 \simeq t_2 \vdash t_2 \simeq t_1 & \text{symmetry} \\
& t_1 \simeq t_2, t_2 \simeq t_3 \vdash t_1 \simeq t_3 & \text{transitivity}
\end{align*}
\]
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 $H \vdash (t \leftarrow c)$ and decisions $? (t \leftarrow c)$ coming from different theories.

Justification $H$: a set of assignments that appear earlier on the trail.

Example (trail grows from left to right):

$\emptyset \vdash (x \simeq z)$, $\emptyset \vdash (y \simeq z)$, $? (x \leftarrow \sqrt{2})$, $? (y \leftarrow \text{blue})$, $? (x \leftarrow \text{red})$, $H \vdash (x \neq y)$

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

Example (trail grows from left to right):

\[
\emptyset \vdash (x \simeq z), \emptyset \vdash (y \simeq z), ?(x \leftarrow \sqrt{2}), ?(y \leftarrow \text{blue}), ?(x \leftarrow \text{red}), H \vdash (x \neq y)
\]

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

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

**Example** (trail grows from left to right):

\[
\emptyset \vdash (x \equiv z), \emptyset \vdash (y \equiv z), ?(x \leftarrow \sqrt{2}), ?(y \leftarrow \text{blue}), ?(x \leftarrow \text{red}), H \vdash (x \neq y)
\]

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

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

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

**In this paper**, new rule for solving/exiting conflicts: Learn
Example: exiting a conflict without learning a clause

Input problem $H_0$ including: $(\neg l_2 \lor \neg l_4 \lor \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

$$\neg l_2 \lor \neg l_4 \lor \neg l_5$$

$$\emptyset \vdash (\neg l_2 \lor \neg l_4 \lor \neg l_5)$$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:

$\neg l_2 \lor \neg l_4 \lor \neg l_5$

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic.

Initial trail $\Gamma_0$ including:

$\emptyset \vdash \neg l_2 \lor \neg l_4 \lor \neg l_5$

Search rules extend $\Gamma_0$ into

$\Gamma_0 \cap \emptyset \vdash \neg l_2 \lor \neg l_4 \lor \neg l_5$

In first conflict, both $l_4$ and $l_5$ depend on the latest decision $l_4$.

After applying Resolve, only $l_4$ does. Time to stop conflict analysis.

Rule Learn can exit the conflict with trail $\Gamma_0$, $l_2$, $l_4$ where

$H \vdash l_4$

where $H$ is

$\{ \neg l_2 \lor \neg l_4 \lor \neg l_5 \}, l_2 \}$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including: $(\neg l_2 \lor \neg l_4 \lor \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, ?A_1$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:

$$\neg l_2 \lor \neg l_4 \lor \neg l_5$$

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic.

Initial trail $\Gamma_0$ including:

$$\emptyset \vdash (\neg l_2 \lor \neg l_4 \lor \neg l_5)$$

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, ?A_1, ?l_2$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including: $(\neg l_2 \lor \neg l_4 \lor \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, \; ?A_1, \; ?l_2, \; ?A_3$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:

$$(-l_2 \lor -l_4 \lor -l_5)$$

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic.

Initial trail $\Gamma_0$ including:

$$\emptyset \vdash (-l_2 \lor -l_4 \lor -l_5)$$

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, ?A_1, ?l_2, ?A_3, ?l_4$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:
$$(-l_2 \vee -l_4 \vee -l_5)$$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic.

Initial trail $\Gamma_0$ including:
$$\emptyset \vdash (-l_2 \vee -l_4 \vee -l_5)$$

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, \ ?A_1, \ ?l_2, \ ?A_3, \ ?l_4, \ l_4 \vdash l_5$

In first conflict, both $l_4$ and $l_5$ depend on the latest decision $?l_4$.

After applying Resolve, only $l_4$ does. Time to stop conflict analysis.

Rule Learn can exit the conflict with trail $\Gamma_0, \ ?A_1, \ ?l_2, \ ?A_3, \ ?l_4, \ l_4 \vdash l_5$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, ?A_1, ?l_2, ?A_3, ?l_4, l_4 \rightarrow l_5$

(involving unrelated decisions $A_1$ and $A_3$)
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:

$$(-l_2 \lor -l_4 \lor -l_5)$$

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

$$\emptyset \vdash (-l_2 \lor -l_4 \lor -l_5)$$

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, ?A_1, ?l_2, ?A_3, ?l_4, l_4 \vdash l_5$

(involving unrelated decisions $A_1$ and $A_3$)

First conflict:

$$\langle \Gamma; (-l_2 \lor -l_4 \lor -l_5), l_2, l_4, l_5 \rangle$$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including: $\neg l_2 \lor \neg l_4 \lor \neg l_5$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic
Initial trail $\Gamma_0$ including:

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, \ ?A_1, \ ?l_2, \ ?A_3, \ ?l_4, \ l_4 \vdash l_5$
(involving unrelated decisions $A_1$ and $A_3$)

First conflict: $\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4, l_5 \rangle$
Resolving $l_5$: $\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4 \rangle$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:

$$(-l_2 \lor -l_4 \lor -l_5)$$

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

$$\emptyset \vdash (-l_2 \lor -l_4 \lor -l_5)$$

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, \ ?A_1, \ ?l_2, \ ?A_3, \ ?l_4, \ l_4 \vdash l_5$

(involving unrelated decisions $A_1$ and $A_3$)

First conflict:

$$\langle \Gamma; (-l_2 \lor -l_4 \lor -l_5), l_2, l_4, l_5 \rangle$$

Resolving $l_5$:

$$\langle \Gamma; (-l_2 \lor -l_4 \lor -l_5), l_2, l_4 \rangle$$

In first conflict, both $l_4$ and $l_5$ depend on the latest decision $?l_4$. After applying Resolve, only $l_4$ does. Time to stop conflict analysis.
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:

$$\neg l_2 \lor \neg l_4 \lor \neg l_5$$

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

$$\emptyset \vdash \neg l_2 \lor \neg l_4 \lor \neg l_5$$

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, ?A_1, ?l_2, ?A_3, ?l_4, l_4 \vdash l_5$

(involving unrelated decisions $A_1$ and $A_3$)

First conflict:

$$\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4, l_5 \rangle$$

Resolving $l_5$:

$$\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4 \rangle$$

In first conflict, both $l_4$ and $l_5$ depend on the latest decision $?l_4$. After applying Resolve, only $l_4$ does. Time to stop conflict analysis.

Rule Learn can exit the conflict with trail

$\Gamma_0, ?A_1, ?l_2, \overline{H \vdash l_4}$

where $H$ is $\{(\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2\}$
Example: exiting a conflict without learning a clause

Input problem $H_0$ including:

$\neg l_2 \lor \neg l_4 \lor \neg l_5$

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

$\emptyset \vdash (\neg l_2 \lor \neg l_4 \lor \neg l_5)$

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, ?A_1, ?l_2, ?A_3, ?l_4, l_4 \vdash l_5$

(involving unrelated decisions $A_1$ and $A_3$)

First conflict:

$\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4, l_5 \rangle$

Resolving $l_5$:

$\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4 \rangle$

In first conflict, both $l_4$ and $l_5$ depend on the latest decision $?l_4$.

After applying Resolve, only $l_4$ does. Time to stop conflict analysis.

Rule Learn can exit the conflict with trail

$\Gamma_0, ?A_1, ?l_2, H \vdash l_4$

where $H$ is $\{(\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2 \}$
Example: exiting a conflict learning a clause

Input problem $H_0$ including:

$$\neg l_2 \lor \neg l_4 \lor \neg l_5$$

with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

$$\emptyset \vdash (\neg l_2 \lor \neg l_4 \lor \neg l_5)$$

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, \uparrow A_1, \uparrow l_2, \uparrow A_3, \uparrow l_4, l_4 \vdash l_5$

(involving unrelated decisions $A_1$ and $A_3$)

First conflict:

$$\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4, l_5 \rangle$$

Resolving $l_5$:

$$\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4 \rangle$$

In first conflict, both $l_4$ and $l_5$ depend on the latest decision $\uparrow l_4$.

After applying Resolve, only $l_4$ does. Time to stop conflict analysis.

Rule Learn can exit the conflict and learn a clause:

$$\Gamma_0, \uparrow A_1, \uparrow l_2, \quad H' \vdash (\neg l_2 \lor \neg l_4)$$

where $H'$ is $\{ (\neg l_2 \lor \neg l_4 \lor \neg l_5) \}$
Example: exiting a conflict learning a clause

Input problem \( H_0 \) including:
with \( l_4 = (x \leq y) \) and \( l_5 = (f(x) \leq f(y)) \) in a theory where \( f \) is monotonic

Initial trail \( \Gamma_0 \) including:

\[
\emptyset \vdash (\neg l_2 \lor \neg l_4 \lor \neg l_5)
\]

Search rules extend \( \Gamma_0 \) into \( \Gamma = \Gamma_0, \ ?A_1, \ ?l_2, \ ?A_3, \ ?l_4, \ l_4 \vdash l_5 \)
(involving unrelated decisions \( A_1 \) and \( A_3 \))

First conflict:

\[
\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4, l_5 \rangle
\]

Resolving \( l_5 \):

\[
\langle \Gamma; (\neg l_2 \lor \neg l_4 \lor \neg l_5), l_2, l_4 \rangle
\]

In first conflict, both \( l_4 \) and \( l_5 \) depend on the latest decision \( ?l_4 \).
After applying Resolve, only \( l_4 \) does. Time to stop conflict analysis.

Rule Learn can exit the conflict and learn a clause:

\[
\Gamma_0, \ ?A_1, \ ?l_2, \ H' \vdash (\neg l_2 \lor \neg l_4) \quad \text{where } H' \text{ is } \{ (\neg l_2 \lor \neg l_4 \lor \neg l_5) \}
\]

Then Deduce can derive \( l_4 \) as before:

\[
\Gamma_0, \ ?A_1, \ ?l_2, \ H' \vdash (\neg l_2 \lor \neg l_4), \ \{ (\neg l_2 \lor \neg l_4), l_2 \} \vdash l_4
\]
Example: exiting a conflict learning a clause & restarting

Input problem $H_0$ including:

$\neg{\text{l}_2} \lor \neg{\text{l}_4} \lor \neg{\text{l}_5}$

with $\text{l}_4 = (x \leq y)$ and $\text{l}_5 = (f(x) \leq f(y))$ in a theory where $f$ is monotonic

Initial trail $\Gamma_0$ including:

$\emptyset \vdash \neg{\text{l}_2} \lor \neg{\text{l}_4} \lor \neg{\text{l}_5}$

Search rules extend $\Gamma_0$ into $\Gamma = \Gamma_0, ?A_1, ?l_2, ?A_3, ?l_4, \text{l}_4 \vdash \text{l}_5$

(involving unrelated decisions $A_1$ and $A_3$)

First conflict:

$\langle \Gamma; (\neg{\text{l}_2} \lor \neg{\text{l}_4} \lor \neg{\text{l}_5}), l_2, l_4, l_5 \rangle$

Resolving $l_5$:

$\langle \Gamma; (\neg{\text{l}_2} \lor \neg{\text{l}_4} \lor \neg{\text{l}_5}), l_2, l_4 \rangle$

In first conflict, both $l_4$ and $l_5$ depend on the latest decision $?l_4$.

After applying Resolve, only $l_4$ does. Time to stop conflict analysis.

Rule Learn can exit the conflict and learn a clause, and restart:

$\Gamma_0, H' \vdash (\neg{\text{l}_2} \lor \neg{\text{l}_4})$

where $H'$ is $\set{(\neg{\text{l}_2} \lor \neg{\text{l}_4} \lor \neg{\text{l}_5})}$
The Learn rule introduced in this paper

\[ \langle \Gamma; E \cup H \rangle \rightarrow \Gamma', \quad E \vdash \lnot L \quad \text{if } L \text{ is a “clausal form of } H \text{”, } L \notin \Gamma, \neg L \notin \Gamma \]

\( \Gamma' \): a pruning of \( \Gamma \) undoing at least the latest decision involved,

\( E \subseteq \Gamma' \)
The Learn rule introduced in this paper

\[ \langle \Gamma; E \cup H \rangle \rightarrow \Gamma', \overline{E} \vdash L \] if \( L \) is a “clausal form of \( H \)”, \( L \notin \Gamma \), \( \overline{L} \notin \Gamma \)

\( \Gamma' \): a pruning of \( \Gamma \) undoing at least the latest decision involved, \( E \subseteq \Gamma' \)
The Learn rule introduced in this paper

\[ \langle \Gamma; E \cup H \rangle \rightarrow \Gamma', \models E \vdash L \] if \( L \) is a “clausal form of \( H \)”, \( L \notin \Gamma, \overline{L} \notin \Gamma \)

\( \Gamma' \): a pruning of \( \Gamma \) undoing at least the latest decision involved,
\( E \subseteq \Gamma' \)

“Clausal forms of \( H \)” reify \( H \) in Boolean logic:

\[ (((\bigwedge_{(l \leftarrow \text{true}) \in H} l) \land (\bigwedge_{(l \leftarrow \text{false}) \in H} \lnot l)) \leftarrow \text{false} \]
The Learn rule introduced in this paper

\[ \langle \Gamma; E \cup H \rangle \rightarrow \Gamma', \text{ } \overline{E} \vdash L \text{ if } L \text{ is a “clausal form of } H \text{”, } L \not\in \Gamma, \text{ } \overline{L} \not\in \Gamma \]

\( \Gamma' \): a pruning of \( \Gamma \) undoing at least the latest decision involved,
\( E \subseteq \Gamma' \)

“Clausal forms of \( H \)” reify \( H \) in Boolean logic:

\[
\begin{align*}
(((\bigwedge (l \leftarrow true) \in H \ l) \land (\bigwedge (l \leftarrow false) \in H \ \neg l)) \leftarrow false \quad & \\
(((\bigvee (l \leftarrow true) \in H \ \neg l) \lor (\bigvee (l \leftarrow false) \in H \ l)) \leftarrow true
\end{align*}
\]

This rule generalises the CADE’2017 one (sufficient for completeness)

models clause learning by reifying (Boolean parts of) conflicts

models clause learning + restarts, a common practice in SAT/SMT-solving

Which version to apply depends on your search strategy (particularly for restarts)

All version are OK with respect to termination of CDSAT
The Learn rule introduced in this paper

\[ \langle \Gamma; E \cup H \rangle \rightarrow \Gamma', \bar{E} \vdash L \] if \( L \) is a “clausal form of \( H \)”, \( L \notin \Gamma, \bar{L} \notin \Gamma \)

\( \Gamma' \): a pruning of \( \Gamma \) undoing at least the latest decision involved, \( E \subseteq \Gamma' \)

“Clausal forms of \( H \)” reify \( H \) in Boolean logic:

\[
\begin{align*}
((\bigwedge (l \leftarrow \text{true}) \in H \ l) \land (\bigwedge (l \leftarrow \text{false}) \in H \ \neg l)) & \leftarrow \text{false} \\
((\bigvee (l \leftarrow \text{true}) \in H \ \neg l) \lor (\bigvee (l \leftarrow \text{false}) \in H \ l)) & \leftarrow \text{true}
\end{align*}
\]

This rule

- generalises the CADE’2017 one (sufficient for completeness)
- models clause learning by reifying (Boolean parts of) conflicts
- models clause learning + restarts,
  a common practice in SAT/SMT-solving
The Learn rule introduced in this paper

\[ \langle \Gamma; E \cup H \rangle \rightarrow \Gamma', \overline{E \vdash L} \text{ if } L \text{ is a “clausal form of } H \text{”, } L \notin \Gamma, \overline{L} \notin \Gamma \]

\(\Gamma'\): a pruning of \(\Gamma\) undoing at least the latest decision involved, \(E \subseteq \Gamma'\)

“Clausal forms of \(H\)” reify \(H\) in Boolean logic:

\[
((\forall (l \leftarrow true) \in H \ l) \land (\forall (l \leftarrow false) \in H \ \overline{l})) \leftarrow false \\
((\lor (l \leftarrow true) \in H \ \overline{l}) \lor (\lor (l \leftarrow false) \in H \ l)) \leftarrow true
\]

This rule

- generalises the CADE’2017 one (sufficient for completeness)
- models clause learning by reifying (Boolean parts of) conflicts
- models clause learning + restarts,
  a common practice in SAT/SMT-solving

Which version to apply depends on your search strategy
(particularly for restarts)
The Learn rule introduced in this paper

\[ \langle \Gamma; E \cup H \rangle \longrightarrow \Gamma', \overline{E} L \text{ if } L \text{ is a “clausal form of } H, \text{ } L \notin \Gamma, \overline{L} \notin \Gamma \]

\[ \Gamma': \text{ a pruning of } \Gamma \text{ undoing at least the latest decision involved, } \]
\[ E \subseteq \Gamma' \]

“Clausal forms of } H” reify } H \text{ in Boolean logic:

\[ (((\bigwedge ( l \leftarrow true) \in H l) \land (\bigwedge ( l \leftarrow false) \in H \neg l)) \leftarrow false \]
\[ (((\bigvee ( l \leftarrow true) \in H \neg l) \lor (\bigvee ( l \leftarrow false) \in H l)) \leftarrow true \]

This rule

- generalises the CADE’2017 one (sufficient for completeness)
- models clause learning by reifying (Boolean parts of) conflicts
- models clause learning + restarts,
  a common practice in SAT/SMT-solving

Which version to apply depends on your search strategy
(particularly for restarts)

All version are OK with respect to termination of CDSAT
3. Proof production
Soundness invariants, and rules that may affect them

- For every assignment $H |- A$ on the trail, $H \models A$;
- For every conflict state $\langle \Gamma; E \rangle$, $E \models \bot$. 

Next step: keep track of invariant via proof-theoretical information

Let $T$ be a theory with a specific $T$-module.

Deduce $\Gamma \rightarrow \Gamma$, $J |- (t \leftarrow b)$ if $J \models \models T(t \leftarrow b)$ and $J \subseteq \Gamma$, and $t \leftarrow b$ is not in $\Gamma$.

Conflict $\Gamma \rightarrow \langle \Gamma; J, (t \leftarrow b) \rangle$ if $J \models \models T(t \leftarrow b)$ and $J \subseteq \Gamma$, and $t \leftarrow b$ is in $\Gamma$.

Resolve $\langle \Gamma; E \cup \{A\} \rangle \rightarrow \langle \Gamma; E \cup H \rangle$ if $H \models A$ is in $\Gamma$.

Learn $\langle \Gamma; E \cup H \rangle \rightarrow \Gamma'$, $E |- L$ if $L$ is a "clausal form" of $H$ 

$L/e \not\in \Gamma$, $L/e \not\in \Gamma$, and $E \subseteq \Gamma'$.
Soundness invariants, and rules that may affect them

- For every assignment $H \vdash A$ on the trail, $H \models A$;
- For every conflict state $\langle \Gamma; E \rangle$, $E \models \bot$.

Next step: keep track of invariant via proof-theoretical information.
Soundness invariants, and rules that may affect them

- For every assignment $H \vdash A$ on the trail, $H \models A$;
- For every conflict state $\langle \Gamma; E \rangle$, $E \models \bot$.

Next step: keep track of invariant via proof-theoretical information

Let $T$ be a theory with a specific $T$-module.

Deduce

$$\Gamma \rightarrow \Gamma, J \vdash (t \leftarrow b) \quad \text{if} \quad J \vdash_T (t \leftarrow b) \quad \text{and} \quad J \subseteq \Gamma,$$

and $t \leftarrow \overline{b}$ is not in $\Gamma$. 

"clausal form" of $L$
Soundness invariants, and rules that may affect them

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

Next step: keep track of invariant via proof-theoretical information

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

**Deduce**

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

**Conflict**

\[
\Gamma \quad \rightarrow \quad \langle \Gamma; J, (t \leftarrow \overline{b}) \rangle \quad \text{if} \quad J \vdash_{\mathcal{T}} (t \leftarrow b) \quad \text{and} \quad J \subseteq \Gamma, \quad \text{and} \quad t \leftarrow b \text{ is in } \Gamma
\]
Soundness invariants, and rules that may affect them

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

Next step: keep track of invariant via proof-theoretical information

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

**Deduce**

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

**Conflict**

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

**Resolve**

\[
\langle \Gamma; E \cup \{A\} \rangle \quad \rightarrow \quad \langle \Gamma; E \cup H \rangle \quad \text{if } H \vdash A \text{ is in } \Gamma
\]
Soundness invariants, and rules that may affect them

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

Next step: keep track of invariant via proof-theoretical information

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

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

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

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

\[ \text{Learn} \]
\[ \langle \Gamma; E \cup H \rangle \rightarrow \Gamma', E \vdash L \quad \text{if} \quad L \quad \text{is a “clausal form” of} \quad H \quad L \notin \Gamma, \quad \bar{L} \notin \Gamma, \quad \text{and} \quad E \subseteq \Gamma' \]
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”

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

is annotated as

\[a_1(t_1 \leftarrow c_1), \ldots, a_k(t_k \leftarrow c_k) \vdash \mathcal{T} j\mathcal{T} : (l \leftarrow b)\]
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”

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

is annotated as

\[^{a_1}(t_1 \leftarrow c_1), \ldots, ^{a_k}(t_k \leftarrow c_k) \vdash _\mathcal{T} j : (l \leftarrow b)\]

Examples:

\[^{a_1}(x \leftarrow \sqrt{2}), \ ^{a_2}(y \leftarrow \sqrt{2}) \vdash _{\text{NLRA}} \text{eval}\left(\{a_1, a_2\}\right): (x \cdot y \simeq 2)\]

(evaluation inference)

\[^{a_0}(l_1 \lor \cdots \lor l_n), \ ^{a_1}(\overline{l_1}), \ldots, ^{a_{k-1}}(\overline{l_{n-1}}) \vdash _{\text{Bool}} \text{UP}\left(a_0, \{a_1, \ldots, a_n\}\right): l_n\]

(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

<table>
<thead>
<tr>
<th>Deduction proof terms</th>
<th>$j ::= \text{in} \mid j_\mathcal{T} \mid \text{lem}(H.c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conflict proof term</td>
<td>$c ::= \text{cfl}(j_\mathcal{T}, a) \mid \text{res}(j, ^aA.c)$</td>
</tr>
</tbody>
</table>

- $\text{in}$ annotates an input assignment,
- $j_\mathcal{T}$ ranges over theory proofs for $\mathcal{T}$, 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 $c$
- $\text{cfl}(j_\mathcal{T}, a)$ annotates a conflict when it is created by Conflict
- $\text{res}(j, ^aA.c)$ annotates a conflict resulting from the Resolve rule, binding $a$ in $c$
Provability invariants that proof-terms keep track of

\[ A \text{ is an input} \]
\[ \emptyset \vdash \text{in} : A \]
\[ J \vdash j_T : L \]
\[ J \cup \{^aL\} \vdash \text{cfl}(j_T, a) : \bot \]

\[ J \cup \{^aL\} \vdash \text{cfl}(j_T, a) : \bot \]
\[ J \vdash j_T : L \]
\[ J \vdash j_T : L \]
\[ E \cup H \vdash c : \bot \]

\[ E \cup H \vdash \text{res}(j, ^aA.c) : \bot \]

L clausal form of \( H \)

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

\[ \text{Theorem} \]
\[ \text{For every assignment } H \vdash j : A \text{ on the trail, } H \vdash \square \square \square j : A \]
\[ \text{For every conflict state } \langle \Gamma; E; c \rangle, E \vdash \square \square \square c : \bot \]

The proof system above can be seen as glueing a collection of
inference systems (\( \square \square \square T \))

CDSAT is a search procedure for the resulting system
Provability invariants that proof-terms keep track of

\[
\begin{align*}
A \text{ is an input} & \quad J \vdash \pi : L & \quad E \cup H \vdash c : \bot \\
\emptyset \vdash \text{in} : A & \quad J \vdash \pi : L & \quad E \vdash \text{lem}(H.c) : L \\
J \vdash \pi : L & \quad J \cup \{\pi\} \vdash \text{cfl}(\pi, a) : \bot & \quad H \vdash \pi : A \quad E, ^aA \vdash c : \bot \\
& \quad E \cup H \vdash \text{res}(\pi, ^aA.c) : \bot & \\
\end{align*}
\]

L clausal form of H

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

**Theorem**

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

\[ A \text{ is an input} \]
\[ \emptyset \vdash \text{in} : A \]
\[ J \vdash \text{in} : A \]
\[ J \vdash j_T : L \]
\[ E \cup H \vdash c : \bot \]
\[ E \vdash \text{lem} (H.c) : L \]
\[ J \vdash j_T : L \]
\[ J \cup \{^a \overline{L} \} \vdash \text{cfl}(j_T, a) : \bot \]
\[ H \vdash j : A \]
\[ E, ^a A \vdash c : \bot \]
\[ E \cup H \vdash \text{res}(j, ^a A.c) : \bot \]

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

\[ \begin{align*}
A \text{ is an input} & \quad J \vdash_T j_T : L & \quad E \cup H \vdash c : \bot \\
\emptyset \vdash \text{in} : A & \quad J \vdash j_T : L & \quad E \vdash \text{lem}(H.c) : L \\
J \vdash_T j_T : L & \quad H \vdash j : A & \quad E, ^aA \vdash c : \bot \\
J \cup \{ ^aL \} \vdash \text{cfl}(j_T, a) : \bot & \quad E \cup H \vdash \text{res}(j, ^aA.c) : \bot
\end{align*} \]

L clausal form of \(H\)

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 \((\vdash_T)_T\)

CDSAT is a search procedure for the resulting system
Satisfiability Modulo Assignments (SMA)

An SMT-problem with input clauses $C_1, \ldots, C_n$ is treated by running CDSAT on the initial trail $\emptyset \vdash \text{in}: C_1, \ldots, \emptyset \vdash \text{in}: C_n$. But the CDSAT system can accept inputs with first-order assignments, e.g. $\emptyset \vdash \text{in}: (x \leftarrow \frac{3}{4})$, $\emptyset \vdash \text{in}: (x \leq y)$, $\emptyset \vdash \text{in}: (y \leq 0)$. Such problems are called SMA problems.

If there are no first-order inputs and the problem is unsat, then the final proof-term will not mention any deduction proof-term $H \vdash \vdash \vdash j: L$ nor any conflict proof $H \vdash \vdash \vdash c: \bot$ such that $H$ contains a first-order assignment. Easy optimisation in that case: the construction of any such proof-term during the run can be omitted. Theory modules do not have to provide theory proofs $H \vdash \vdash \vdash T j: T : L$ if $H$ contains a first-order assign. (typically: evaluation inferences).
Satisfiability Modulo Assignments (SMA)

An SMT-problem with input clauses $C_1, \ldots, C_n$ is treated by running CDSAT on the initial trail $\emptyset \vdash_{\text{in}}: C_1, \ldots, \emptyset \vdash_{\text{in}}: C_n$

But the CDSAT system can accept inputs with first-order assignments, e.g: $\emptyset \vdash_{\text{in}}: (x \leftarrow \frac{3}{4}), \emptyset \vdash_{\text{in}}: (x \leq y), \emptyset \vdash_{\text{in}}: (y \leq 0)$

Such problems are called SMA problems.
Satisfiability Modulo Assignments (SMA)

An SMT-problem with input clauses $C_1, \ldots, C_n$ is treated by running CDSAT on the initial trail $\emptyset \vdash_{\text{in}}: C_1, \ldots, \emptyset \vdash_{\text{in}}: C_n$

But the CDSAT system can accept inputs with first-order assignments, e.g:

- $\emptyset \vdash_{\text{in}}: (x \leftarrow 3/4)$
- $\emptyset \vdash_{\text{in}}: (x \leq y)$
- $\emptyset \vdash_{\text{in}}: (y \leq 0)$

Such problems are called SMA problems.

If there are no first-order inputs and the problem is unsat, then the final proof-term will not mention any deduction proof-term $H \vdash j : L$ nor any conflict proof $H \vdash c : \bot$ such that $H$ contains a first-order assignment
Satisfiability Modulo Assignments (SMA)

An SMT-problem with input clauses $C_1, \ldots, C_n$ is treated by running CDSAT on the initial trail $\emptyset \vdash_{\text{in}}: C_1, \ldots, \emptyset \vdash_{\text{in}}: C_n$

But the CDSAT system can accept inputs with first-order assignments, e.g: $\emptyset \vdash_{\text{in}}: (x \leftarrow \frac{3}{4}), \emptyset \vdash_{\text{in}}: (x \leq y), \emptyset \vdash_{\text{in}}: (y \leq 0)$

Such problems are called SMA problems.

If there are no first-order inputs and the problem is unsat, then the final proof-term will not mention any deduction proof-term $H \vdash j: L$ nor any conflict proof $H \vdash c: \bot$ such that $H$ contains a first-order assignment.

Easy optimisation in that case: the construction of any such proof-term during the run can be omitted.
Satisfiability Modulo Assignments (SMA)

An SMT-problem with input clauses $C_1, \ldots, C_n$ is treated by running CDSAT on the initial trail $\emptyset \vdash_{\text{in}}: C_1, \ldots, \emptyset \vdash_{\text{in}}: C_n$

But the CDSAT system can accept inputs with first-order assignments, e.g: $\emptyset \vdash_{\text{in}}: (x \leftarrow 3/4)$, $\emptyset \vdash_{\text{in}}: (x \leq y)$, $\emptyset \vdash_{\text{in}}: (y \leq 0)$

Such problems are called SMA problems.

If there are no first-order inputs and the problem is unsat, then the final proof-term will not mention any deduction proof-term $H \vdash j : L$ nor any conflict proof $H \vdash c : \bot$ such that $H$ contains a first-order assignment

Easy optimisation in that case: the construction of any such proof-term during the run can be omitted.

Theory modules do not have to provide theory proofs $H \vdash_T j_T : L$ if $H$ contains a first-order assign. (typically: evaluation inferences)
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.
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 \( \text{in}, j_T, \text{lem}(H.c), \text{cfl}(j_T, a), \text{res}(j, ^aA.c) \) are terms.

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

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 $H \vdash \text{res}(j, ^a A.c) : \bot$ 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 (SMA problems) 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 $H$. 

No proof-checking. But the LCF architecture \cite{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
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-checking.
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-checking. 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-checking. 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}
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-checking. 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-checking. 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`.
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-checking. 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.
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-checking. 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-checking. 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

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

```plaintext
type deduction = assign*single_assign
type conflict = assign
```

and exports

```plaintext
type deduction
type conflict
in : single_assign -> deduction
coerc : 'k theory_handler
   -> 'k theory_proof -> deduction
lem : conflict -> assign -> deduction
clf : 'k theory_handler
       -> 'k theory_proof -> conflict
res : deduction -> conflict -> 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.
Conclusion

- Proof-producing CDSAT clarifies at what point CDSAT needs to record proof information to justify answers “unsat”, and how.
Conclusion

- **Proof-producing CDSAT** clarifies at what point CDSAT needs to record proof information to justify answers “unsat”, and how.
- **Proof-producing CDSAT** only requires a small proof system, which glues together a collection of inference systems in a modular way.

Proof-terms map to resolution proofs + theory lemmas if this is preferred format.

If inputs contain first-order assignments, this format has to be generalised with guarded clauses.

Proof-terms can be convenient for translations to proof assistants (c.f. SMTCoq [AFG11]).

CDSAT is suited to the LCF principles, which are standard.
Conclusion

- **Proof-producing CDSAT** clarifies at what point CDSAT needs to record proof information to justify answers “unsat”, and how.

- **Proof-producing CDSAT** only requires a small proof system, which glues together a collection of inferences systems in a modular way.

- **Clause learning** is still not needed for completeness of CDSAT / its proof system ... but is critical for efficiency of search, and compresses proofs by sharing subproofs.
Conclusion

- **Proof-producing CDSAT** clarifies **at what point** CDSAT needs to record proof information to justify answers “unsat”, and **how**.
- **Proof-producing CDSAT** only requires a small proof system, which **glues together a collection of inferences systems** in a modular way.
- **Clause learning** is still not needed for completeness of CDSAT / its proof system ... but is critical for efficiency of search, and compresses proofs by sharing subproofs.
- **Nothing exotic:**
  - Proof terms map to resolution proofs + theory lemmas if this is preferred format.
    - If inputs contain first-order assignments, this format has to be generalised with guarded clauses
  - Proof-terms can be convenient for translations to proof assistants (c.f. SMTCoq [AFG+11])
  - CDSAT is suited to the LCF principles, which are standard
Conclusion

- Proof-producing CDSAT clarifies at what point CDSAT needs to record proof information to justify answers “unsat”, and how.
- Proof-producing CDSAT only requires a small proof system, which glues together a collection of inferences systems in a modular way.
- Clause learning is still not needed for completeness of CDSAT / its proof system ...but is critical for efficiency of search, and compresses proofs by sharing subproofs.
- Nothing exotic:
  - Proof terms map to resolution proofs + theory lemmas if this is preferred format.
    
    If inputs contain first-order assignments, this format has to be generalised with guarded clauses
  - Proof-terms can be convenient for translations to proof assistants (c.f. SMTCoq [AFG+11])
  - CDSAT is suited to the LCF principles, which are standard
Ongoing and future work

- Proof-of-concept implementation is available at https://github.com/disteph/cdsat
  Currently working on more performance-driven implementation.
Ongoing and future work

- Proof-of-concept implementation is available at [https://github.com/disteph/cdsat](https://github.com/disteph/cdsat)
  Currently working on more performance-driven implementation.

- Issue of cost: Penalty of building proof-terms?
  Penalty of having a code developed in the correct-by-construction approach?
Ongoing and future work

- Proof-of-concept implementation is available at [https://github.com/disteph/cdsat](https://github.com/disteph/cdsat)
  Currently working on more performance-driven implementation.

- Issue of cost: Penalty of building proof-terms?
  Penalty of having a code developed in the correct-by-construction approach?

- Use proof-terms for interpolation?
  (See Tanja Schindler’s talk on interpolation in a related context! 16:30 at VMCAI)
Verifying SAT and SMT in Coq for a fully automated decision procedure.
In G. Faure, S. Lengrand, and A. Mahboubi, editors, Proc. of the 2011 Work. on Proof-Search in Axiomatic Theories and Type Theories (PSATTT’11), 2011.
Available at http://hal.inria.fr/PSATTT11

Satisfiability modulo theories and assignments.
In L. de Moura, editor, Proc. of the 26th Int. Conf. on Automated Deduction (CADE’17), volume 10395 of LNAI.
Springer-Verlag, 2017.

L. M. de Moura and D. Jovanovic.
A model-constructing satisfiability calculus.
In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, Proc. of the 14th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI’13), volume 7737 of LNCS, pages 1–12. Springer-Verlag, 2013.
Adapting the rules

Deduce
\[ \Gamma \rightarrow \Gamma, \ j \vdash_{\mathcal{T}} (t \leftarrow b) \]
if \( J \vdash_{\mathcal{T}} j_{\mathcal{T}} : (t \leftarrow b), \ J \subseteq \Gamma \),
and \( t \leftarrow b \) is not in \( \Gamma \)

Conflict
\[ \Gamma \rightarrow \langle \Gamma; \ J, (t \leftarrow \overline{b}) ; \ \text{cfl}(j_k, a) \rangle \]
if \( J \vdash_{\mathcal{T}} j_{\mathcal{T}} : (t \leftarrow b), \ J \subseteq \Gamma \),
and \( t \leftarrow b \) is in \( \Gamma \) with id \( a \)

Resolve
\[ \langle \Gamma; \ E \cup \{ A \} ; \ c \rangle \rightarrow \langle \Gamma; \ E \cup H ; \ \text{res}(j, aA.c) \rangle \]
if \( H \vdash j : A \) is in \( \Gamma \) with id \( a \)

Learn
\[ \langle \Gamma; \ E \cup H ; \ c \rangle \rightarrow \Gamma', \ E \vdash_{\text{lem}} (H,c) : L \]
if \( L \) is a “clausal form” of \( H \)
\( L \notin \Gamma, \ \overline{L} \notin \Gamma, \) and \( E \subseteq \Gamma' \)