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., $l \leftarrow \mathsf{true}$

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., $l \leftarrow \mathsf{true}$

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., $l \leftarrow \mathsf{true}$

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., $l \leftarrow \mathsf{true}$

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")?

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

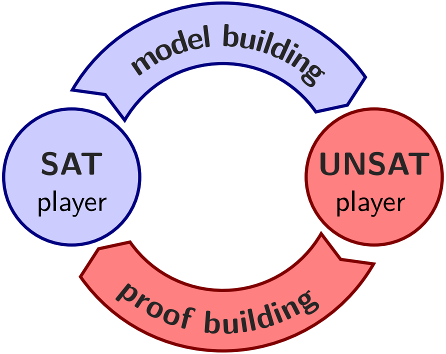
The CDSAT system - with learning

Proof production

1. Conflict-driven theory combination

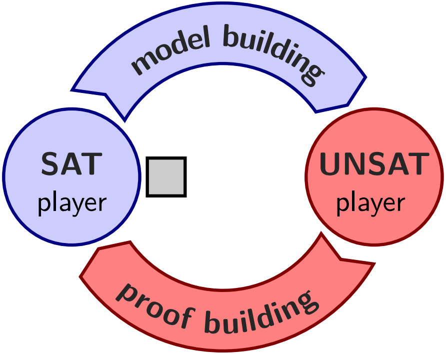
Conflict-driven reasoning

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.



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.

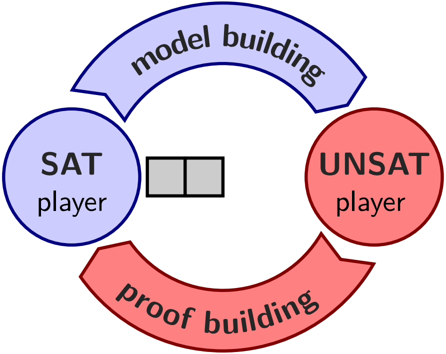


Conflict-driven reasoning

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.

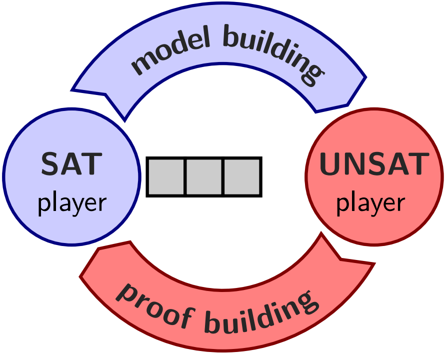


Conflict-driven reasoning

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.

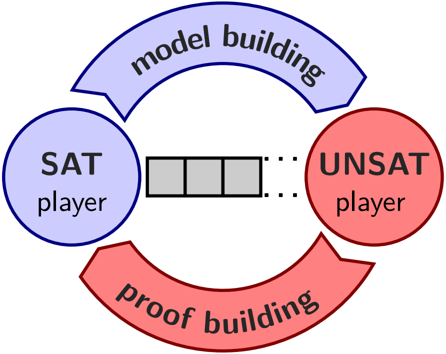


Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable.

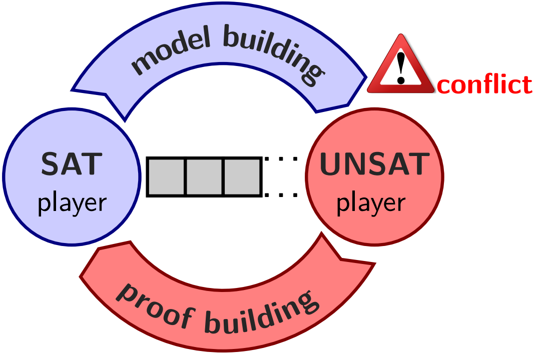
It involves a trail where a putative model is being specified.

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.



Conflict-driven reasoning

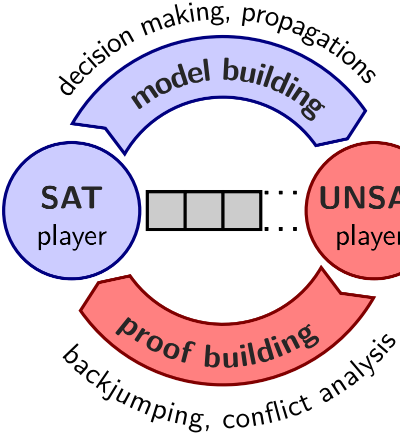
2-player game to determine whether a formula is satisfiable.

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**

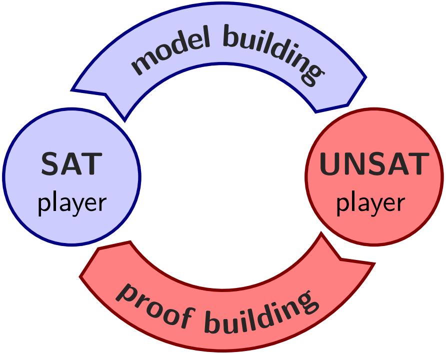
a conflict occurs when a clause is falsified



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

$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable.

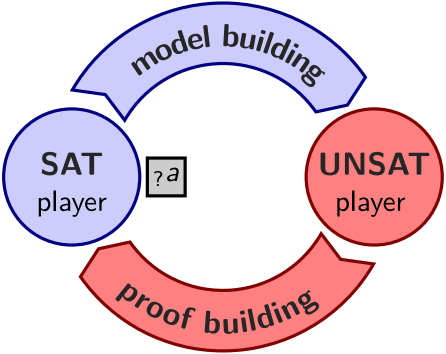
It involves a trail where a putative model is being specified.

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

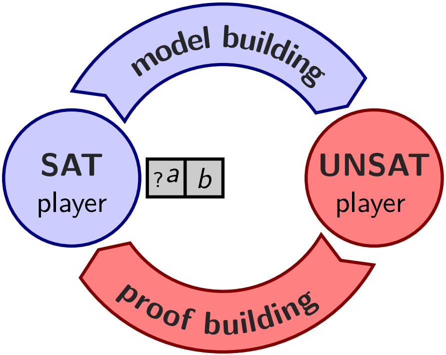
$$a \Rightarrow b$$
$$b \Rightarrow \overline{a}$$
$$\overline{a} \Rightarrow \overline{b}$$
$$\overline{b} \Rightarrow a$$



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

$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable.

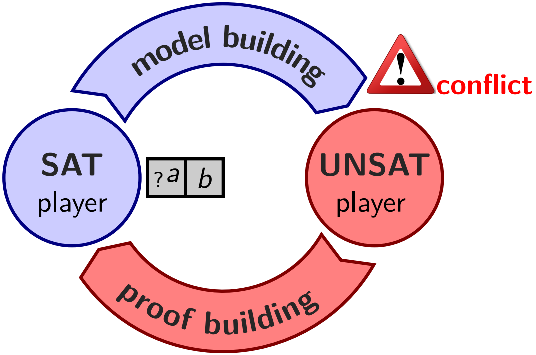
It involves a trail where a putative model is being specified.

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

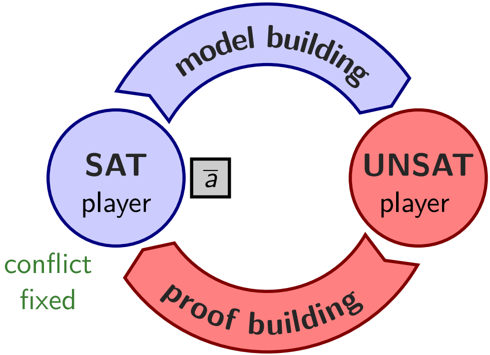
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



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

$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$
$\overline{a}$



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

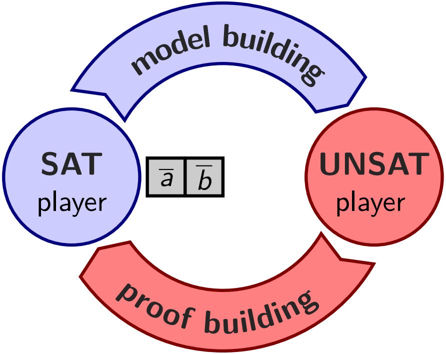
$a \Rightarrow b$

$b \Rightarrow \overline{a}$

$\overline{a} \Rightarrow \overline{b}$

$\overline{b} \Rightarrow a$

$\overline{a}$



Conflict-driven reasoning

2-player game to determine whether a formula is satisfiable.

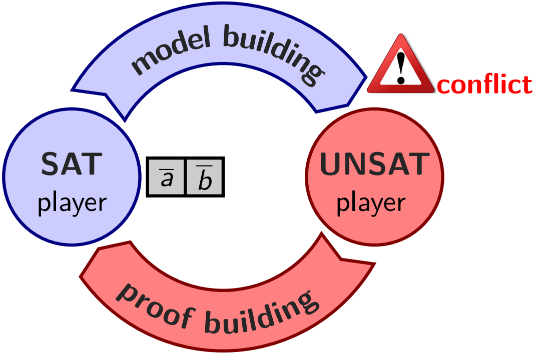
It involves a trail where a putative model is being specified.

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

$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$
$\overline{a}$



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

$a \Rightarrow b$

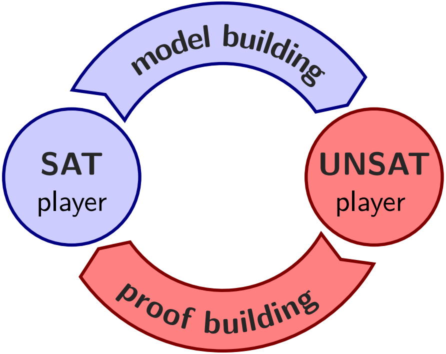
$b \Rightarrow \overline{a}$

$\overline{a} \Rightarrow \overline{b}$

$\overline{b} \Rightarrow a$

$\overline{a}$

$\perp$



Conflict-driven reasoning can be used for (other) theories

$$\overbrace{(-2\cdot x - y < 0)}^{l_0}, \qquad \overbrace{(x + y < 0)}^{l_1}, \qquad \overbrace{(x < -1)}^{l_2}$$

unsatisfiable in Linear Rational Arithmetic (LRA).

Conflict-driven reasoning can be used for (other) theories

$$\overbrace{(-2\cdot x - y < 0)}^{l_0}, \qquad \overbrace{(x + y < 0)}^{l_1}, \qquad \overbrace{(x < -1)}^{l_2}$$

unsatisfiable in Linear Rational Arithmetic (LRA).

- Guess a value, e.g., $y \leftarrow 0$

Conflict-driven reasoning can be used for (other) theories

$$\overbrace{(-2{\cdot}x - y < 0)}^{l_0}, \qquad \overbrace{(x + y < 0)}^{l_1}, \qquad \overbrace{(x < -1)}^{l_2}$$

unsatisfiable in Linear Rational Arithmetic (LRA).

- Guess a value, e.g., $y{\leftarrow}0$
  Then l_0 yields lower bound $x > 0$

Conflict-driven reasoning can be used for (other) theories

$$\overbrace{(-2{\cdot}x - y < 0)}^{l_0}, \qquad \overbrace{(x + y < 0)}^{l_1}, \qquad \overbrace{(x < -1)}^{l_2}$$

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 $y \neq 0$?

Conflict-driven reasoning can be used for (other) theories

$$\overbrace{(-2{\cdot}x - y < 0)}^{l_0}, \qquad \overbrace{(x + y < 0)}^{l_1}, \qquad \overbrace{(x < -1)}^{l_2}$$

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., $\overbrace{(-y < -2)}^{l_3}$
  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

$$\overbrace{(-2{\cdot}x - y < 0)}^{l_0}, \quad \overbrace{(x + y < 0)}^{l_1}, \quad \overbrace{(x < -1)}^{l_2}$$

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., $\overbrace{(-y < -2)}^{l_3}$
  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.

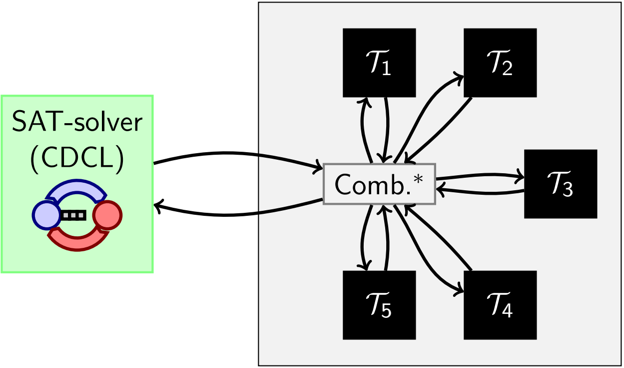
Conflict-driven reasoning can be used for (other) theories

$$\overbrace{(-2\cdot x - y < 0)}^{l_0}, \quad \overbrace{(x + y < 0)}^{l_1}, \quad \overbrace{(x < -1)}^{l_2}$$

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., $\overbrace{(-y < -2)}^{l_3}$
  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



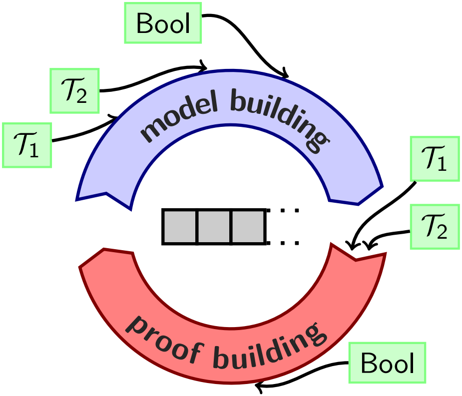
* e.g. equality sharing / Nelson-Oppen [NO79]

In CDSAT

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

As in MCSAT, trail contains

- Boolean assignments
  $a \leftarrow \mathrm{true}$
- First-order assignments
  $y \leftarrow 3/4$

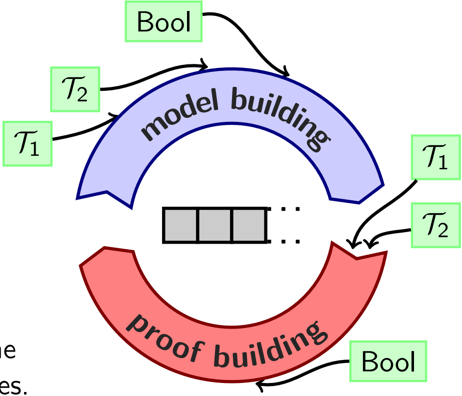


In CDSAT

. . . 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

$$(t_1 \leftarrow \mathfrak{c}_1), \ldots, (t_k \leftarrow \mathfrak{c}_k) \vdash_{\mathcal{T}} (l \leftarrow \mathfrak{b})$$

where

- each $t_i \leftarrow \mathfrak{c}_i$ is a single $\mathcal{T}$-assignment
  (a term t_i and a $\mathcal{T}$-value $\mathfrak{c}_i$ of matching sorts)
- $l \leftarrow \mathfrak{b}$ is a single Boolean assignment
  (a term l of sort Bool and a truth value $\mathfrak{b}$)

What is a theory module?

A set of inferences of the form

$$(t_1 \leftarrow \mathfrak{c}_1), \ldots, (t_k \leftarrow \mathfrak{c}_k) \vdash_{\mathcal{T}} (l \leftarrow \mathfrak{b})$$

where

- each $t_i \leftarrow \mathfrak{c}_i$ is a single $\mathcal{T}$-assignment
  (a term t_i and a $\mathcal{T}$-value $\mathfrak{c}_i$ of matching sorts)
- $l \leftarrow \mathfrak{b}$ is a single Boolean assignment
  (a term l of sort Bool and a truth value $\mathfrak{b}$)

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

$$(t_1 \leftarrow \mathfrak{c}_1), \dots, (t_k \leftarrow \mathfrak{c}_k) \vdash_{\mathcal{T}} (l \leftarrow \mathfrak{b})$$

where

- each $t_i \leftarrow \mathfrak{c}_i$ is a single $\mathcal{T}$-assignment
  (a term t_i and a $\mathcal{T}$-value $\mathfrak{c}_i$ of matching sorts)
- $l \leftarrow \mathfrak{b}$ is a single Boolean assignment
  (a term l of sort Bool and a truth value $\mathfrak{b}$)
  Abbreviations: ($l \leftarrow$true) as l and ($l \leftarrow$false) as $\bar{l}$
- Soundness requirement:
  Every model of the premisses is a model of the conclusion:
  $(t_1 \leftarrow \mathfrak{c}_1), \dots, (t_k \leftarrow \mathfrak{c}_k) \models (l \leftarrow \mathfrak{b})$

What is a theory module?

A set of inferences of the form

$$(t_1 \leftarrow \mathfrak{c}_1), \ldots, (t_k \leftarrow \mathfrak{c}_k) \vdash_{\mathcal{T}} (l \leftarrow \mathfrak{b})$$

where

- each $t_i \leftarrow \mathfrak{c}_i$ is a single $\mathcal{T}$-assignment
  (a term t_i and a $\mathcal{T}$-value $\mathfrak{c}_i$ of matching sorts)
- $l \leftarrow \mathfrak{b}$ is a single Boolean assignment
  (a term l of sort Bool and a truth value $\mathfrak{b}$)
  Abbreviations: $(l \leftarrow \mathrm{true})$ as l and $(l \leftarrow \mathrm{false})$ as $\overline{l}$
- Soundness requirement:
  Every model of the premisses is a model of the conclusion:
  $(t_1 \leftarrow \mathfrak{c}_1), \ldots, (t_k \leftarrow \mathfrak{c}_k) \models (l \leftarrow \mathfrak{b})$

Examples:

$(x \leftarrow \sqrt{2}), (y \leftarrow \sqrt{2}) \vdash_{\mathrm{NLRA}} (x \cdot y \simeq 2)$ (evaluation inference)

$(l_1 \vee \cdots \vee l_n), \overline{l_1} \ldots, \overline{l_{n-1}} \vdash_{\mathrm{Bool}} l_n$ (unit propagation)

What is a theory module? (Equality inferences)

All theory modules have the equality inferences:

$$
\begin{array}{rll}
t_1 \leftarrow \mathfrak{c}_1, t_2 \leftarrow \mathfrak{c}_2 \vdash_{\mathcal{T}} & t_1 \simeq t_2 & \text{if } \mathfrak{c}_1 \text{ and } \mathfrak{c}_2 \text{ are the same value} \\
t_1 \leftarrow \mathfrak{c}_1, t_2 \leftarrow \mathfrak{c}_2 \vdash_{\mathcal{T}} & t_1 \not\simeq t_2 & \text{if } \mathfrak{c}_1 \text{ and } \mathfrak{c}_2 \text{ are distinct values} \\
\\
\vdash_{\mathcal{T}} & t_1 \simeq t_1 & \text{reflexivity} \\
t_1 \simeq t_2 \vdash_{\mathcal{T}} & t_2 \simeq t_1 & \text{symmetry} \\
t_1 \simeq t_2, t_2 \simeq t_3 \vdash_{\mathcal{T}} & t_1 \simeq t_3 & \text{transitivity}
\end{array}
$$

CDSAT states

Search states: simply trails.

A trail is a stack of justified assignments ${}_{H\vdash}(t{\leftarrow}\mathfrak{c})$ and decisions ${}_{?}(t{\leftarrow}\mathfrak{c})$ 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 ${}_{H\vdash}(t{\leftarrow}\mathfrak{c})$ and decisions ${}_{?}(t{\leftarrow}\mathfrak{c})$ coming from different theories
Justification H: a set of assignments that appear earlier on the trail

Example (trail grows from left to right):

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

where H is $\{(y{\leftarrow}\mathrm{blue}),\ (x{\leftarrow}\mathrm{red})\}$

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

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

CDSAT states

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

Example (trail grows from left to right):

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

where H is $\{(y{\leftarrow}\mathrm{blue}),\ (x{\leftarrow}\mathrm{red})\}$

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

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

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

CDSAT states

Search states: simply trails.
A trail is a stack of justified assignments ${}_{H\vdash}(t\leftarrow\mathfrak{c})$ and decisions ${}_{?}(t\leftarrow\mathfrak{c})$ 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 Γ + 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: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

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: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

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

Initial trail Γ_0 including: ${}_{\emptyset \vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

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

Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_0 into $\Gamma \;=\; \Gamma_0$

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_0 into $\Gamma \;=\; \Gamma_0,\ {}_{?}A_1$

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_0 into $\Gamma \; = \quad \Gamma_0, \; {}_{?}A_1, \; {}_{?}l_2$

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

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

Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_0 into $\Gamma \; = \quad \Gamma_0,\ {}_{?}A_1,\ {}_{?}l_2,\ {}_{?}A_3$

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_0 into $\Gamma \;=\; \Gamma_0,\ {}_{?}A_1,\ {}_{?}l_2,\ {}_{?}A_3,\ {}_{?}l_4,\ {}_{l_4\vdash}l_5$

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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)

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: $_{\emptyset}\vdash (\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$
Resolving l_5: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4 \rangle$

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset \vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$
Resolving l_5: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4 \rangle$
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.

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$
Resolving l_5: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4 \rangle$
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
$\Gamma_0,\ {}_{?}A_1,\ {}_{?}l_2,\ {}_{H\vdash}\overline{l_4}$

where H is $\{(\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2\}$

Example: exiting a conflict without learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$
Resolving l_5: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4 \rangle$
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
$\Gamma_0,\ {}_{?}A_1,\ {}_{?}l_2,\ {}_{H\vdash}\overline{l_4}$

where H is $\{(\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2\}$

Example: exiting a conflict learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$
Resolving l_5: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4 \rangle$
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:
$\Gamma_0,\ {}_{?}A_1,\ {}_{?}l_2,\ {}_{H'\vdash}(\neg l_2 \vee \neg l_4)$
where H' is $\{(\neg l_2 \vee \neg l_4 \vee \neg l_5)\}$

Example: exiting a conflict learning a clause

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset}\vdash(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$
Resolving l_5: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4 \rangle$
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:
$\Gamma_0,\ {}_{?}A_1,\ {}_{?}l_2,\ {}_{H'}\vdash(\neg l_2 \vee \neg l_4)$
where H' is $\{(\neg l_2 \vee \neg l_4 \vee \neg l_5)\}$
Then Deduce can derive $\overline{l_4}$ as before:
$\Gamma_0,\ {}_{?}A_1,\ {}_{?}l_2,\ {}_{H'}\vdash(\neg l_2 \vee \neg l_4),\ {}_{\{(\neg l_2 \vee \neg l_4), l_2\}}\vdash \overline{l_4}$

Example: exiting a conflict learning a clause & restarting

Input problem H_0 including: $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
with $l_4 = (x \leq y)$ and $l_5 = (f(x) \leq f(y))$ in a theory where f is monotonic
Initial trail Γ_0 including: ${}_{\emptyset}\!\vdash\!(\neg l_2 \vee \neg l_4 \vee \neg l_5)$

Search rules extend Γ_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: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$
Resolving l_5: $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4 \rangle$
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, and restart:
$\Gamma_0,\ {}_{H'}\!\vdash\!(\neg l_2 \vee \neg l_4)$

where H' is $\{(\neg l_2 \vee \neg l_4 \vee \neg l_5)\}$

The Learn rule introduced in this paper

$$\langle \Gamma; E \uplus H \rangle \longrightarrow \Gamma', {}_{E}\vdash L \quad \text{if } L \text{ is a "clausal form of } H\text{", } L \notin \Gamma,\ \overline{L} \notin \Gamma$$

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

The Learn rule introduced in this paper

$$\langle \Gamma; E \uplus H \rangle \longrightarrow \Gamma', {}_{E}\vdash L \quad \text{if } L \text{ is a "clausal form of } H\text{"}, L \notin \Gamma, \overline{L} \notin \Gamma$$

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

The Learn rule introduced in this paper

$$\langle \Gamma; E \uplus H \rangle \longrightarrow \Gamma', {}_{E\vdash} L \quad \text{if } L \text{ is a "clausal form of } H\text{"}, L \notin \Gamma, \overline{L} \notin \Gamma$$

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

"Clausal forms of H" **reify** H in Boolean logic:

$$((\bigwedge_{(l \leftarrow \mathsf{true}) \in H} l) \wedge (\bigwedge_{(l \leftarrow \mathsf{false}) \in H} \neg l)) \leftarrow \mathsf{false}$$

The Learn rule introduced in this paper

$$\langle \Gamma; E \uplus H \rangle \longrightarrow \Gamma', {}_{E\vdash} L \quad \text{if } L \text{ is a "clausal form of } H\text{"}, L \notin \Gamma, \overline{L} \notin \Gamma$$

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

"Clausal forms of H" **reify** H in Boolean logic:

$$((\bigwedge_{(l \leftarrow \mathsf{true}) \in H} l) \wedge (\bigwedge_{(l \leftarrow \mathsf{false}) \in H} \neg l)) \leftarrow \mathsf{false}$$
$$((\bigvee_{(l \leftarrow \mathsf{true}) \in H} \neg l) \vee (\bigvee_{(l \leftarrow \mathsf{false}) \in H} l)) \leftarrow \mathsf{true}$$

The Learn rule introduced in this paper

$$\langle \Gamma; E \uplus H \rangle \longrightarrow \Gamma', {}_{E}{\vdash} L \quad \text{if } L \text{ is a "clausal form of } H\text{"}, L \notin \Gamma, \overline{L} \notin \Gamma$$

Γ': a pruning of Γ undoing at least the latest decision involved,
$E \subseteq \Gamma'$
"Clausal forms of H" **reify** H in Boolean logic:

$$((\textstyle\bigwedge_{(l \leftarrow \mathrm{true}) \in H} l) \wedge (\bigwedge_{(l \leftarrow \mathrm{false}) \in H} \neg l)) \leftarrow \mathrm{false}$$
$$((\textstyle\bigvee_{(l \leftarrow \mathrm{true}) \in H} \neg l) \vee (\bigvee_{(l \leftarrow \mathrm{false}) \in H} l)) \leftarrow \mathrm{true}$$

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

The Learn rule introduced in this paper

$$\langle \Gamma; E \uplus H \rangle \longrightarrow \Gamma', {}_{E}{\vdash} L \quad \text{if } L \text{ is a ``clausal form of } H\text{''}, L \notin \Gamma, \overline{L} \notin \Gamma$$

Γ': a pruning of Γ undoing at least the latest decision involved,
$E \subseteq \Gamma'$
"Clausal forms of H" **reify** H in Boolean logic:

$$\begin{array}{c}((\bigwedge_{(l \leftarrow \text{true}) \in H} l) \wedge (\bigwedge_{(l \leftarrow \text{false}) \in H} \neg l)) {\leftarrow} \text{false} \\ ((\bigvee_{(l \leftarrow \text{true}) \in H} \neg l) \vee (\bigvee_{(l \leftarrow \text{false}) \in H} l)) {\leftarrow} \text{true}\end{array}$$

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

$$\langle \Gamma; E \uplus H \rangle \longrightarrow \Gamma', {}_{E}\vdash L \quad \text{if } L \text{ is a "clausal form of } H\text{", } L \notin \Gamma,\ \overline{L} \notin \Gamma$$

Γ': a pruning of Γ undoing at least the latest decision involved,
$E \subseteq \Gamma'$
"Clausal forms of H" **reify** H in Boolean logic:

$$((\bigwedge_{(l \leftarrow \mathsf{true}) \in H} l) \wedge (\bigwedge_{(l \leftarrow \mathsf{false}) \in H} \neg l)) \leftarrow \mathsf{false}$$
$$((\bigvee_{(l \leftarrow \mathsf{true}) \in H} \neg l) \vee (\bigvee_{(l \leftarrow \mathsf{false}) \in H} l)) \leftarrow \mathsf{true}$$

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

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

Deduce

$$\Gamma \quad \longrightarrow \quad \Gamma, {}_{J\vdash}(t \leftarrow \mathfrak{b}) \qquad \text{if } J \vdash_{\mathcal{T}} (t \leftarrow \mathfrak{b}) \text{ and } J \subseteq \Gamma, \text{ and } t \leftarrow \overline{\mathfrak{b}} \text{ is not in } \Gamma$$

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.

Deduce

$$\Gamma \longrightarrow \Gamma, {}_{J\vdash}(t{\leftarrow}\mathfrak{b}) \quad \text{if } J \vdash_{\mathcal{T}} (t{\leftarrow}\mathfrak{b}) \text{ and } J \subseteq \Gamma, \text{ and } t{\leftarrow}\overline{\mathfrak{b}} \text{ is not in } \Gamma$$

Conflict

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

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.

Deduce

$$\Gamma \longrightarrow \Gamma, {}_{J\vdash}(t \leftarrow \mathfrak{b}) \quad \text{if } J \vdash_{\mathcal{T}} (t \leftarrow \mathfrak{b}) \text{ and } J \subseteq \Gamma, \text{ and } t \leftarrow \overline{\mathfrak{b}} \text{ is not in } \Gamma$$

Conflict

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

Resolve

$$\langle \Gamma; E \uplus \{A\} \rangle \longrightarrow \langle \Gamma; E \cup H \rangle \quad \text{if } {}_{H\vdash}A \text{ is in } \Gamma$$

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.

Deduce

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

Conflict

$\Gamma \longrightarrow \langle \Gamma; J, (t \leftarrow \overline{\mathfrak{b}}) \rangle$ if $J \vdash_{\mathcal{T}} (t \leftarrow \mathfrak{b})$ and $J \subseteq \Gamma$, and $t \leftarrow \overline{\mathfrak{b}}$ is in Γ

Resolve

$\langle \Gamma; E \uplus \{A\} \rangle \longrightarrow \langle \Gamma; E \cup H \rangle$ if ${}_{H\vdash}A$ is in Γ

Learn

$\langle \Gamma; E \uplus H \rangle \longrightarrow \Gamma', {}_{E\vdash}L$ if L is a "clausal form" of H $L \notin \Gamma$, $\overline{L} \notin \Gamma$, and $E \subseteq \Gamma'$

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"

$$(t_1 \leftarrow \mathfrak{c}_1), \ldots, (t_k \leftarrow \mathfrak{c}_k) \vdash_{\mathcal{T}} (l \leftarrow \mathfrak{b})$$

is annotated as

$${}^{a_1}(t_1 \leftarrow \mathfrak{c}_1), \ldots, {}^{a_k}(t_k \leftarrow \mathfrak{c}_k) \vdash_{\mathcal{T}} j_{\mathcal{T}} : (l \leftarrow \mathfrak{b})$$

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"

$$(t_1 \leftarrow \mathfrak{c}_1), \ldots, (t_k \leftarrow \mathfrak{c}_k) \vdash_{\mathcal{T}} (l \leftarrow \mathfrak{b})$$

is annotated as

$$^{a_1}(t_1 \leftarrow \mathfrak{c}_1), \ldots, {}^{a_k}(t_k \leftarrow \mathfrak{c}_k) \vdash_{\mathcal{T}} j_{\mathcal{T}} : (l \leftarrow \mathfrak{b})$$

Examples:

$^{a_1}(x \leftarrow \sqrt{2}), {}^{a_2}(y \leftarrow \sqrt{2}) \vdash_{\mathrm{NLRA}} \mathrm{eval}(\{a_1, a_2\}) : (x \cdot y \simeq 2)$
(evaluation inference)

$^{a_0}(l_1 \vee \cdots \vee l_n), {}^{a_1}(\overline{l_1}), \ldots, {}^{a_{k-1}}(\overline{l_{n-1}}) \vdash_{\mathrm{Bool}} \mathrm{UP}(a_0, \{a_1, \ldots, a_n\}) : l_n$
(unit propagation)

Proof-terms and proof-carrying CDSAT

- A proof-carrying trail is a stack
  - of justified assignments ${}_{H\vdash j:}(t\leftarrow\mathfrak{c})$
  - and decisions ${}_{?}(t\leftarrow\mathfrak{c})$
- A proof-carrying conflict state is of the form $\langle\Gamma;\ H\ ;\ c\rangle$

...where j and c respectively range over

$$\begin{array}{lrl} \text{Deduction proof terms} & j ::= & \mathsf{in} \ \mid\ j_{\mathcal{T}} \ \mid\ \mathsf{lem}(H.c) \\ \text{Conflict proof term} & c ::= & \mathsf{cfl}(j_{\mathcal{T}}, a) \ \mid\ \mathsf{res}(j, {}^{a}A.c) \end{array}$$

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

Provability invariants that proof-terms keep track of

$$\frac{A \text{ is an input}}{\emptyset \vdash \mathsf{in} : A} \qquad \frac{J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L}{J \vdash j_{\mathcal{T}} : L} \qquad \frac{E \uplus H \vdash c : \bot}{E \vdash \mathsf{lem}(H.c) : L}\ L \text{ clausal form of } H$$

$$\frac{J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L}{J \cup \{{}^{a}\overline{L}\} \vdash \mathsf{cfl}(j_{\mathcal{T}}, a) : \bot} \qquad \frac{H \vdash j : A \qquad E, {}^{a}A \vdash c : \bot}{E \cup H \vdash \mathsf{res}(j, {}^{a}A.c) : \bot}$$

Provability invariants that proof-terms keep track of

$$\frac{A \text{ is an input}}{\emptyset \vdash \mathsf{in} : A} \qquad \frac{J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L}{J \vdash j_{\mathcal{T}} : L} \qquad \frac{E \uplus H \vdash c : \bot}{E \vdash \mathsf{lem}(H.c) : L}\ L \text{ clausal form of } H$$

$$\frac{J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L}{J \cup \{{}^{a}\overline{L}\} \vdash \mathsf{cfl}(j_{\mathcal{T}}, a) : \bot} \qquad \frac{H \vdash j : A \qquad E, {}^{a}A \vdash c : \bot}{E \cup H \vdash \mathsf{res}(j, {}^{a}A.c) : \bot}$$

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,* $\quad H \vdash j : A$
- *For every conflict state* $\langle \Gamma;\ E\ ;\ c \rangle$*,* $\quad E \vdash c : \bot$.

Provability invariants that proof-terms keep track of

$$\frac{A \text{ is an input}}{\emptyset \vdash \mathsf{in} : A} \qquad \frac{J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L}{J \vdash j_{\mathcal{T}} : L} \qquad \frac{E \uplus H \vdash c : \bot}{E \vdash \mathsf{lem}(H.c) : L}\ L \text{ clausal form of } H$$

$$\frac{J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L}{J \cup \{{}^{a}\overline{L}\} \vdash \mathsf{cfl}(j_{\mathcal{T}}, a) : \bot} \qquad \frac{H \vdash j : A \qquad E, {}^{a}A \vdash c : \bot}{E \cup H \vdash \mathsf{res}(j, {}^{a}A.c) : \bot}$$

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,* $\quad H \vdash j : A$
- *For every conflict state* $\langle \Gamma;\ E\ ;\ c\rangle$, $\quad E \vdash c : \bot$.

The proof system above can be seen as glueing a collection of inference systems $(\vdash_{\mathcal{T}})_{\mathcal{T}}$

Provability invariants that proof-terms keep track of

$$\frac{A \text{ is an input}}{\emptyset \vdash \mathsf{in} : A} \qquad \frac{J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L}{J \vdash j_{\mathcal{T}} : L} \qquad \frac{E \uplus H \vdash c : \bot}{E \vdash \mathsf{lem}(H.c) : L}\ L \text{ clausal form of } H$$

$$\frac{J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L}{J \cup \{{}^{a}\overline{L}\} \vdash \mathsf{cfl}(j_{\mathcal{T}}, a) : \bot} \qquad \frac{H \vdash j : A \qquad E, {}^{a}A \vdash c : \bot}{E \cup H \vdash \mathsf{res}(j, {}^{a}A.c) : \bot}$$

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,* $\quad H \vdash j : A$
- *For every conflict state* $\langle \Gamma;\ E\ ;\ c \rangle$, $\quad E \vdash c : \bot$.

The proof system above can be seen as glueing a collection of inference systems $(\vdash_{\mathcal{T}})_{\mathcal{T}}$
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 ${}_{\emptyset\vdash\mathsf{in}:} C_1, \ldots, {}_{\emptyset\vdash\mathsf{in}:} C_n$

Satisfiability Modulo Assignments (SMA)

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

But the CDSAT system can accept inputs with first-order assignments, e.g: ${}_{\emptyset\vdash\mathsf{in}:}(x\leftarrow{}^3/_4)$, ${}_{\emptyset\vdash\mathsf{in}:}(x\leq y)$, ${}_{\emptyset\vdash\mathsf{in}:}(y\leq 0)$
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 ${}_{\emptyset\vdash\mathsf{in}:}\, C_1, \ldots, {}_{\emptyset\vdash\mathsf{in}:}\, C_n$

But the CDSAT system can accept inputs with first-order assignments, e.g: ${}_{\emptyset\vdash\mathsf{in}:}(x \leftarrow {}^3\!/\!_4),\ {}_{\emptyset\vdash\mathsf{in}:}(x \leq y),\ {}_{\emptyset\vdash\mathsf{in}:}(y \leq 0)$
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 ${}_{\emptyset \vdash \mathsf{in}:} C_1, \ldots, {}_{\emptyset \vdash \mathsf{in}:} C_n$

But the CDSAT system can accept inputs with first-order assignments, e.g: ${}_{\emptyset \vdash \mathsf{in}:}(x \leftarrow 3/4)$, ${}_{\emptyset \vdash \mathsf{in}:}(x \leq y)$, ${}_{\emptyset \vdash \mathsf{in}:}(y \leq 0)$
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 $_{\emptyset \vdash \mathsf{in}:} C_1, \ldots, _{\emptyset \vdash \mathsf{in}:} C_n$

But the CDSAT system can accept inputs with first-order assignments, e.g: $_{\emptyset \vdash \mathsf{in}:} (x \leftarrow 3/4)$, $_{\emptyset \vdash \mathsf{in}:} (x \leq y)$, $_{\emptyset \vdash \mathsf{in}:} (y \leq 0)$
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 $H \vdash_{\mathcal{T}} j_{\mathcal{T}} : L$
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 $\mathsf{in}, j_{\mathcal{T}}, \mathsf{lem}(H.c), \mathsf{cfl}(j_{\mathcal{T}}, a), \mathsf{res}(j, {}^{a}A.c)$ are terms.

Different views about proof objects

Proof-carrying CDSAT can be considered exactly as defined above, where $\mathsf{in}, j_{\mathcal{T}}, \mathsf{lem}(H.c), \mathsf{cfl}(j_{\mathcal{T}}, a), \mathsf{res}(j, {}^{a}A.c)$ 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 $\mathsf{in}, j_{\mathcal{T}}, \mathsf{lem}(H.c), \mathsf{cfl}(j_{\mathcal{T}}, a), \mathsf{res}(j, {}^{a}A.c)$ 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 $\emptyset \vdash \mathsf{in} : A$
- or theory lemmas corresponding to theory proofs $J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L$

Internal nodes are obtained by applying resolution rule,
corresponding to $H \vdash \mathsf{res}(j, {}^{a}A.c) : \bot$ 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 $\emptyset \vdash \mathsf{in} : A$
- or theory lemmas corresponding to theory proofs $J \vdash_{\mathcal{T}} j_{\mathcal{T}} : L$

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

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

$$\{(t_1 \leftarrow \mathfrak{c}_1), \ldots, (t_n \leftarrow \mathfrak{c}_n)\} \Rightarrow C$$

where $(t_1 \leftarrow \mathfrak{c}_1), \ldots, (t_n \leftarrow \mathfrak{c}_n)$ 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-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 `theorem` is defined for provable formulae in a module of the prover called 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

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 1/2

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

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

and exports

```
type deduction
type conflict
in    : single_assign -> deduction
coerc : 'k theory_handler
        -> 'k theory_proof -> deduction
lem   : conflict -> assign -> deduction
cfl   : '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 inferences systems in a modular way.

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`
  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`
  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)

M. Armand, G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Wener.
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`

M. P. Bonacina, S. Graham-Lengrand, and N. Shankar.
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

$\Gamma \longrightarrow \Gamma, {}_{J \vdash j_{\mathcal{T}}:}(t \leftarrow \mathfrak{b})$ if $J \vdash_{\mathcal{T}} j_{\mathcal{T}} : (t \leftarrow \mathfrak{b})$, $J \subseteq \Gamma$, and $t \leftarrow \overline{\mathfrak{b}}$ is not in Γ

Conflict

$\Gamma \longrightarrow \langle \Gamma;\ J, (t \leftarrow \overline{\mathfrak{b}})\ ;\ \mathrm{cfl}(j_k, a) \rangle$ if $J \vdash_{\mathcal{T}} j_{\mathcal{T}} : (t \leftarrow \mathfrak{b})$, $J \subseteq \Gamma$, and $t \leftarrow \overline{\mathfrak{b}}$ is in Γ with id a

Resolve

$\langle \Gamma;\ E \uplus \{A\}\ ;\ c \rangle \longrightarrow \langle \Gamma;\ E \cup H\ ;\ \mathrm{res}(j, {}^{a}A.c) \rangle$

if ${}_{H \vdash j:} A$ is in Γ with id a

Learn

$\langle \Gamma;\ E \uplus H\ ;\ c \rangle \quad \longrightarrow \quad \Gamma', {}_{E \vdash \mathrm{lem}(H.c):} L$ if L is a "clausal form" of H, $L \notin \Gamma$, $\overline{L} \notin \Gamma$, and $E \subseteq \Gamma'$