From MCSAT to CDSAT and beyond

Stéphane Graham-Lengrand
SRI International

Dagstuhl, September 2021

From MCSAT to CDSAT and beyond

1. Introduction to MCSAT

The model-constructing approach to SMT-solving 1/2

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

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

The model-constructing approach to SMT-solving 1/2

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

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

MCSAT offers:

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

The model-constructing approach to SMT-solving 1/2

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

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

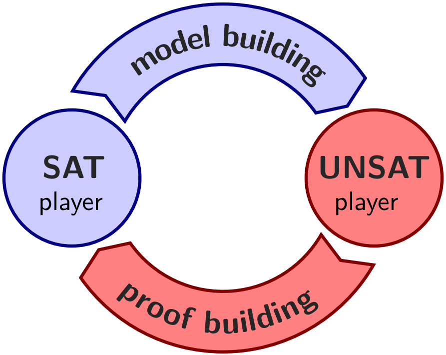
MCSAT offers:

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

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

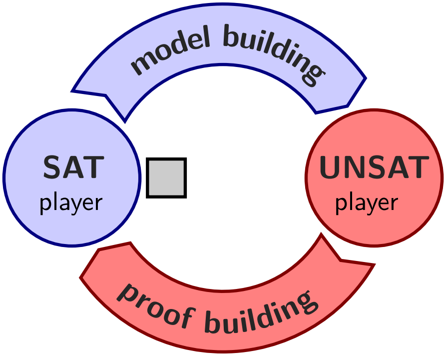
Conflict-driven reasoning

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



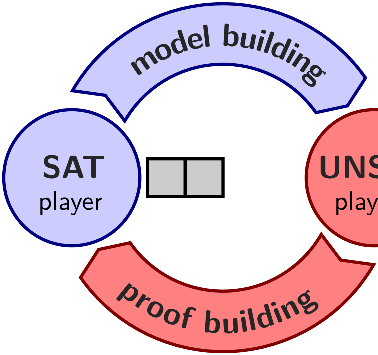
Conflict-driven reasoning

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.



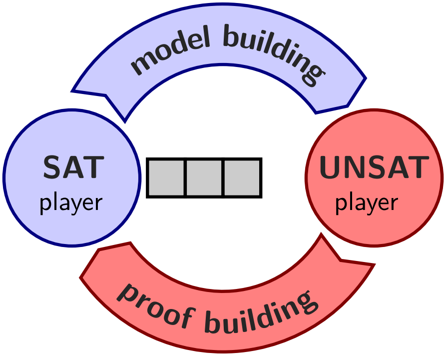
Conflict-driven reasoning

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.



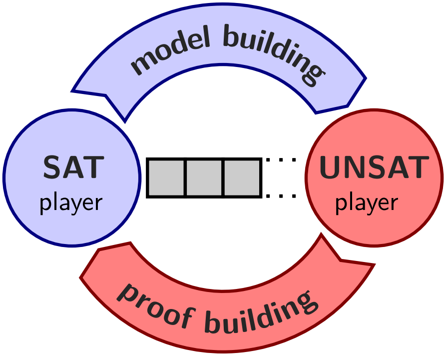
Conflict-driven reasoning

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.



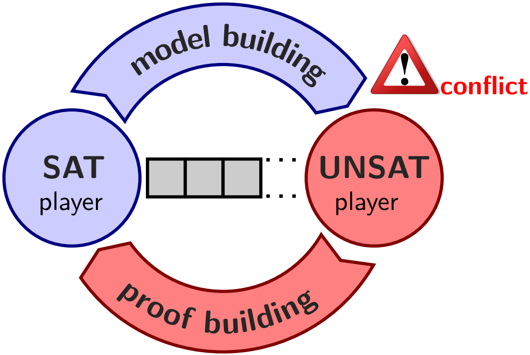
Conflict-driven reasoning

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.



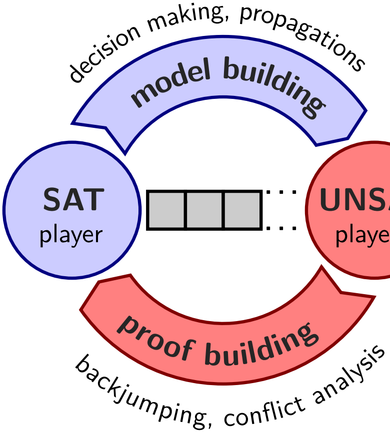
Conflict-driven reasoning

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

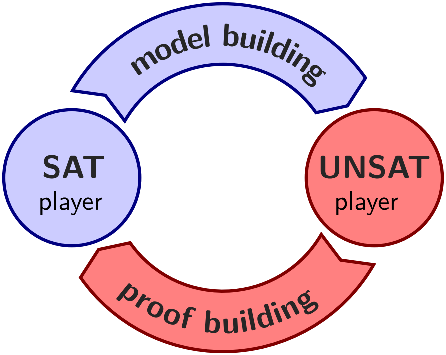
- 2-player game to determine whether a formula is satisfiable.
- It involves a trail where a putative model is being specified.
- It relies on a notion of conflict between the putative model and the formula it should satisfy.
- Archetype of conflict-driven reasoning: **CDCL**
- a conflict occurs when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

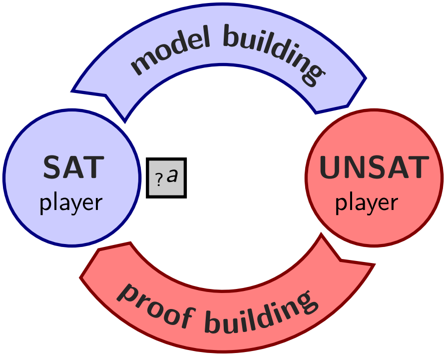
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



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

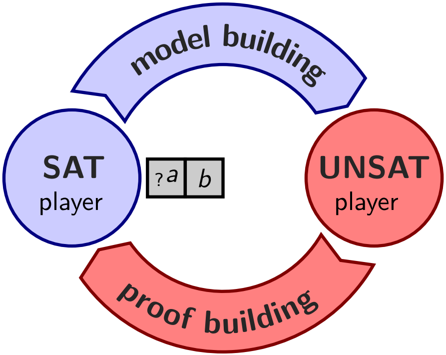
$$a \Rightarrow b$$
$$b \Rightarrow \overline{a}$$
$$\overline{a} \Rightarrow \overline{b}$$
$$\overline{b} \Rightarrow a$$



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

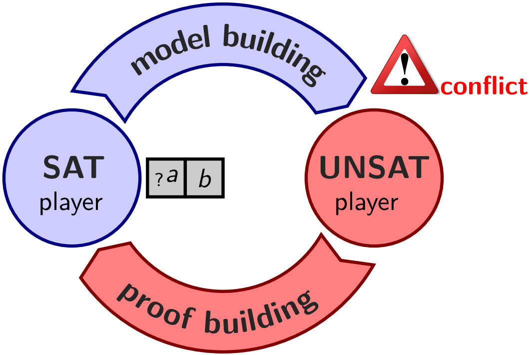
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



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

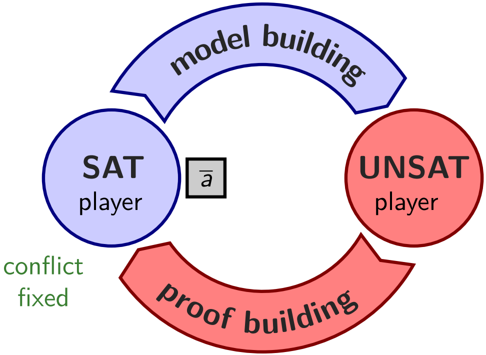
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



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

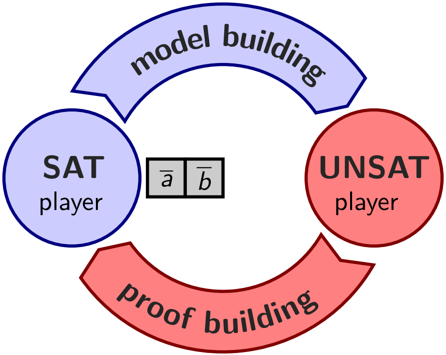
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$
$\overline{a}$



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

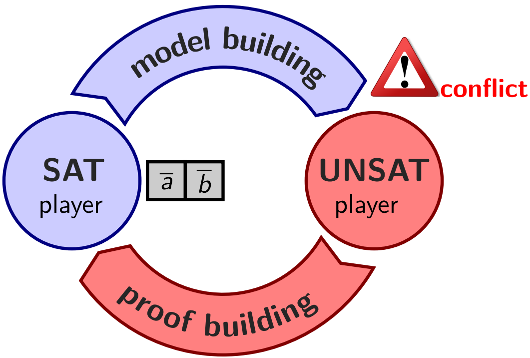
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$
$\overline{a}$



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

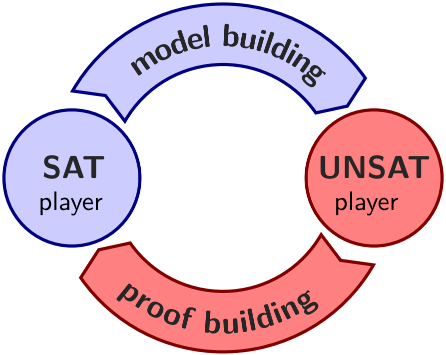
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$
$\overline{a}$



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

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



MCSAT vs CDSAT

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

Evaluation is a key aspect of MCSAT

MCSAT vs CDSAT

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

Evaluation is a key aspect of MCSAT

Solving satisfiability problem

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

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

(so that constraints evaluate to true)

MCSAT vs CDSAT

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

Evaluation is a key aspect of MCSAT

Solving satisfiability problem

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

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

(so that constraints evaluate to true)

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

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

MCSAT vs CDSAT

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

Evaluation is a key aspect of MCSAT

Solving satisfiability problem

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

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

(so that constraints evaluate to true)

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

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

As an abstract framework, it counts among its instances:

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

The model-constructing approach to SMT-solving 2/2

Run = alternation of search phases and conflict analysis phases

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

The model-constructing approach to SMT-solving 2/2

Run = alternation of search phases and conflict analysis phases

- It uses assignments to first-order variables (e.g., $x \leftarrow 3/4$) like CDCL uses Boolean assignments to Boolean variables;
- It may explain conflicts by introducing atoms that are not in the input.
- As in CDCL, it successively guesses values to assign to variables…
  …while maintaining the invariant: *given the assignments made so far, none of the constraints evaluates to false*

The model-constructing approach to SMT-solving 2/2

Run = alternation of search phases and conflict analysis phases

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

The model-constructing approach to SMT-solving 2/2

Run = alternation of search phases and conflict analysis phases

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

The model-constructing approach to SMT-solving 2/2

Run = alternation of search phases and conflict analysis phases

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

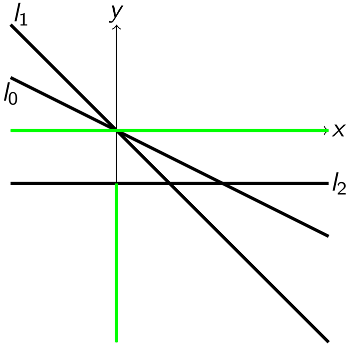
  There is a conflict.

  MCSAT performs a conflict analysis,
  backtracks over some of the assignments $x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n$
  and tries new ones

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

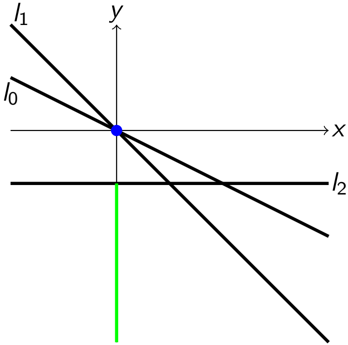


An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

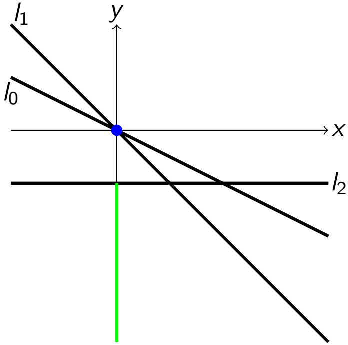
- Guess a value, e.g., $x \leftarrow 0$



An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).



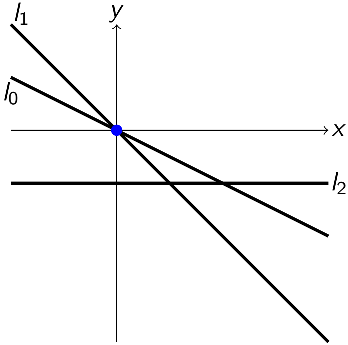
- Guess a value, e.g., $x \leftarrow 0$

  Then l_0 yields lower bound $y > 0$

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).



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

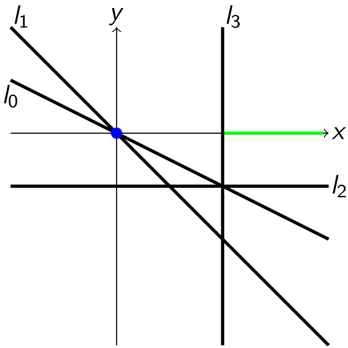
  Then l_0 yields lower bound $y > 0$

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

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).



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

  Then l_0 yields lower bound $y > 0$

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

- **No!** Clash of bounds suggests a better conflict explanation, by inferring $l_0 + 2l_2$,

  i.e., $\overbrace{(-x < -2)}^{l_3}$

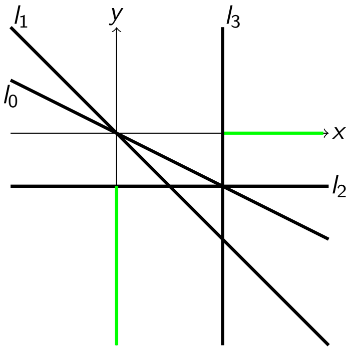
  It rules out $x \leftarrow 0$, but also many values that would fail for the same reasons.

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

$$\overbrace{(-x < -2)}^{l_3}$$



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

  Then l_0 yields lower bound $y > 0$

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

- **No!** Clash of bounds suggests a better conflict explanation, by inferring $l_0 + 2l_2$,

  i.e., $\overbrace{(-x < -2)}^{l_3}$

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

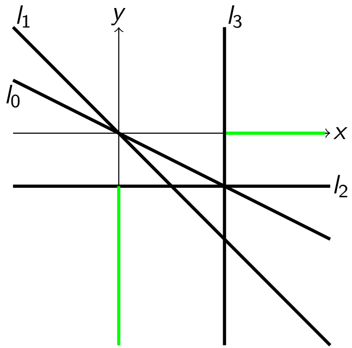
- Now undo the guess but keep l_3.

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

$$\overbrace{(-x < -2)}^{l_3}$$



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

  Then l_0 yields lower bound $y > 0$

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

- **No!** Clash of bounds suggests a better conflict explanation, by inferring $l_0 + 2l_2$,

  i.e., $\overbrace{(-x < -2)}^{l_3}$

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

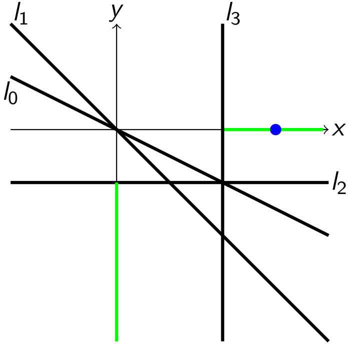
- Now undo the guess but keep l_3.

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

$$\overbrace{(-x < -2)}^{l_3}$$



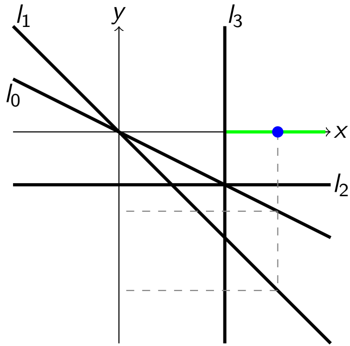
- Guess a value, e.g., $x{\leftarrow}3$

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

$$\overbrace{(-x < -2)}^{l_3}$$



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

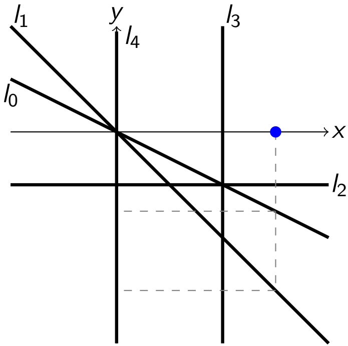
  Then l_0 yields lower bound $y > -\frac{3}{2}$ and l_1 yields upper bound $y < -3$

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

$$\overbrace{(-x < -2)}^{l_3}$$



- Guess a value, e.g., $x{\leftarrow}3$

  Then l_0 yields lower bound $y > -\frac{3}{2}$ and l_1 yields upper bound $y < -3$
- Clash of bounds suggests inferring $l_0 + 2l_1$,

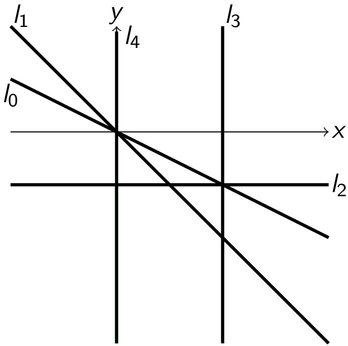
  i.e., $\overbrace{(x < 0)}^{l_4}$.

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

$$\overbrace{(-x < -2)}^{l_3} \quad \overbrace{(x < 0)}^{l_4}$$



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

  Then l_0 yields lower bound $y > -\frac{3}{2}$ and l_1 yields upper bound $y < -3$
- Clash of bounds suggests inferring $l_0 + 2l_1$,

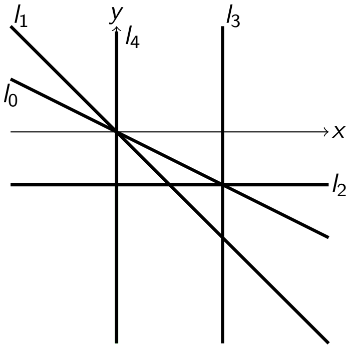
  i.e., $\overbrace{(x < 0)}^{l_4}$.
- Now undo the guess but keep l_4.

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

$$\overbrace{(-x < -2)}^{l_3} \qquad \overbrace{(x < 0)}^{l_4}$$



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

  Then l_0 yields lower bound $y > -\frac{3}{2}$ and l_1 yields upper bound $y < -3$
- Clash of bounds suggests inferring $l_0 + 2l_1$,

  i.e., $\overbrace{(x < 0)}^{l_4}$.
- Now undo the guess but keep l_4.
- Spot that l_3 and l_4 leave no value for x. Clash of bounds suggests inferring $l_3 + l_4$,

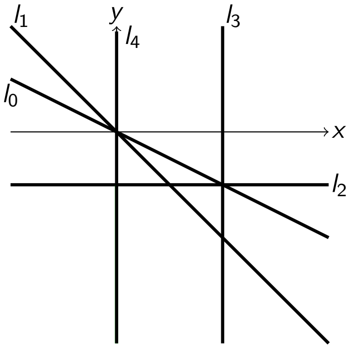
  i.e., $\overbrace{(0 < -2)}^{l_5}$.

An example in Linear Rational Arithmetic

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

unsatisfiable in Linear Rational Arithmetic (LRA).

$$\overbrace{(-x < -2)}^{l_3} \qquad \overbrace{(x < 0)}^{l_4}$$



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

  Then l_0 yields lower bound $y > -\frac{3}{2}$ and l_1 yields upper bound $y < -3$
- Clash of bounds suggests inferring $l_0 + 2l_1$,

  i.e., $\overbrace{(x < 0)}^{l_4}$.
- Now undo the guess but keep l_4.
- Spot that l_3 and l_4 leave no value for x. Clash of bounds suggests inferring $l_3 + l_4$,

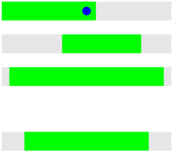
  i.e., $\overbrace{(0 < -2)}^{l_5}$.
- No guess to undo. UNSAT.

Search phase (satisfiable case)

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

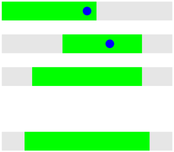
Search phase (satisfiable case)

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



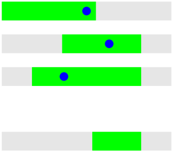
Search phase (satisfiable case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C^1_1, \ldots, C^1_j, \ldots$		x_1
$\{x_1, x_2\}$	$C^2_1, C^2_2, \ldots, C^2_j, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C^3_1, C^3_2, \ldots, C^3_j, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C^i_1, C^i_2, \ldots, C^i_{42}, \ldots, C^i_j, \ldots$		x_i

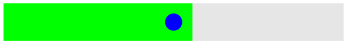


Search phase (satisfiable case)

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

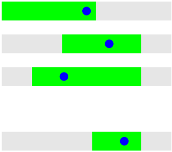


Search phase (satisfiable case)

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

Search phase (satisfiable case)

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



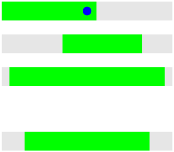
SAT

Search phase (conflict case)

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

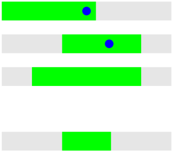
Search phase (conflict case)

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

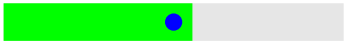


Search phase (conflict case)

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

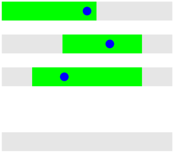


Search phase (conflict case)

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

Search phase (conflict case)

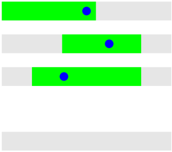
Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots,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



Conflict

Search phase (conflict case)

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

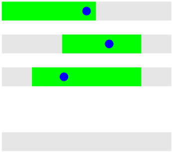


Conflict

What to do now?

Search phase (conflict case)

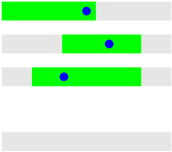
Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots,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



Conflict

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

Search phase (conflict case)

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

Conflict

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

How do we avoid picking the same values (i.e. the same Γ)?
How do we avoid picking a Γ' that fails for the same reason Γ fails?

Conflict analysis

In case of conflict we have

- assigned values $x_1 \mapsto v_1, \ldots, x_1 \mapsto v_n$, i.e., a model $\mathfrak{M}$

Conflict analysis

In case of conflict we have

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

Conflict analysis

In case of conflict we have

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

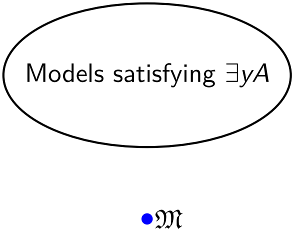
where $\exists y A$ is $\exists y(l_1(\overrightarrow{x}, y) \wedge \cdots \wedge l_m(\overrightarrow{x}, y))$

Conflict analysis

In case of conflict we have

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

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

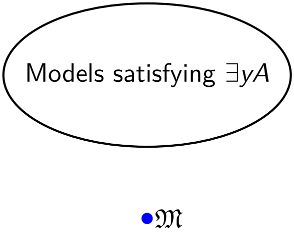


Conflict analysis

In case of conflict we have

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

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

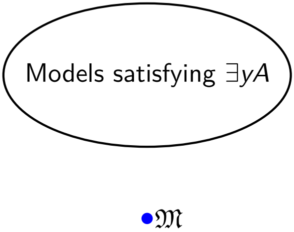


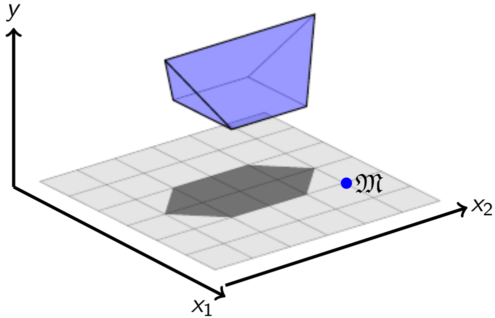
Conflict analysis

In case of conflict we have

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

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





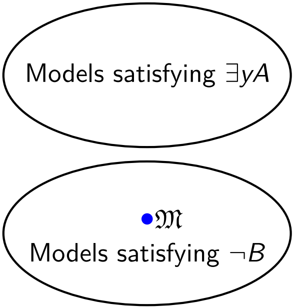
We seek to generalise $\mathfrak{M}$ into a class of models that do not satisfy $\exists y A$ "for the same reason" $\mathfrak{M}$ does not.

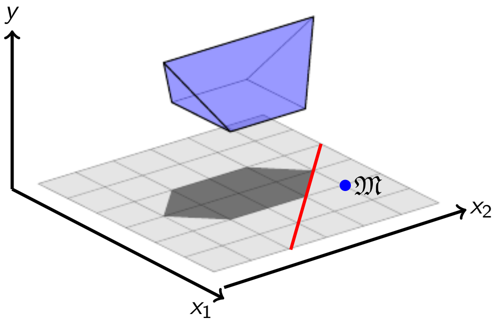
Conflict analysis

In case of conflict we have

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

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

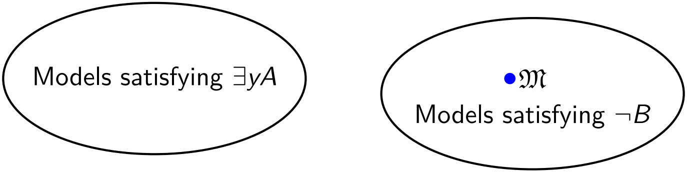




We seek to generalise $\mathfrak{M}$ into a class of models that do not satisfy $\exists y A$ "for the same reason" $\mathfrak{M}$ does not.

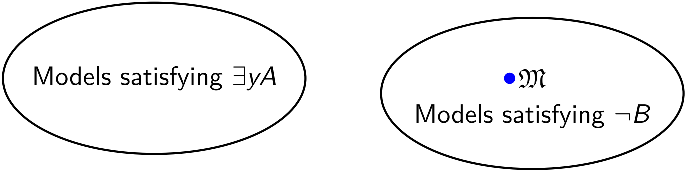
The theory lemmas

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



The theory lemmas

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



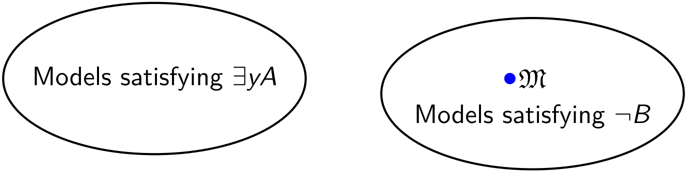
We characterise this class as those models not satisfying B, for some quantifier-free B (with $\mathrm{fv}(B) \subseteq \{\overrightarrow{x}\}$) such that

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

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

The theory lemmas

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



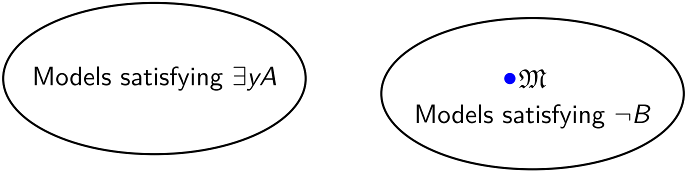
We characterise this class as those models not satisfying B, for some quantifier-free B (with $\mathrm{fv}(B) \subseteq \{\overrightarrow{x}\}$) such that

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

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

The theory lemmas

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



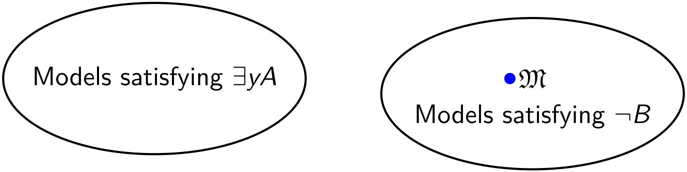
We characterise this class as those models not satisfying B, for some quantifier-free B (with $\mathrm{fv}(B) \subseteq \{\overrightarrow{x}\}$) such that

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

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

The theory lemmas

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



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

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

B is an interpolant of $\exists y A$ at $\mathfrak{M}$.
MCSAT considers the theory lemma $A \Rightarrow B$
that rules out not only $\mathfrak{M}$ but a set of similar models
(we impose that B be a clause, so $A \Rightarrow B$ is a clause).
If A results from Boolean reasoning,
it performs Boolean conflict analysis on A (Boolean resolutions).
It backtracks to a point where $A \Rightarrow B$ is no longer violated,
e.g., B no longer evaluates (to false).

MCSAT theories

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

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

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

MCSAT theories

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

- a nice way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;
- such an explanation mechanism, producing B as a clause (or $\neg B$ as a cube)

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

MCSAT theories

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

- a nice way of representing domains of feasible values,
  and how they are affected (i.e. reduced) by unit constraints;
- such an explanation mechanism,
  producing B as a clause (or $\neg B$ as a cube) ,
  satisfying some suitable conditions for termination;

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

MCSAT theories

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

- a nice way of representing domains of feasible values,
  and how they are affected (i.e. reduced) by unit constraints;
- such an explanation mechanism,
  producing B as a clause (or $\neg B$ as a cube) ,
  satisfying some suitable conditions for termination;
- optionally, a nice way to propagate a value for a variable whose domain has become a singleton set;

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

MCSAT theories

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

- a nice way of representing domains of feasible values,
  and how they are affected (i.e. reduced) by unit constraints;
- such an explanation mechanism,
  producing B as a clause (or $\neg B$ as a cube) ,
  satisfying some suitable conditions for termination;
- optionally, a nice way to propagate a value for a variable whose domain has become a singleton set;

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

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

In arithmetic

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

$$\frac{e_1 - y \lessdot_1 0 \quad e_2 + y \lessdot_2 0}{e_1 + e_2 \lessdot_3 0}$$

with $\lessdot_1, \lessdot_2, \lessdot_3 \in \{\leq, <\}$ such that...

In arithmetic

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

$$\frac{e_1 - y \lessdot_1 0 \quad e_2 + y \lessdot_2 0}{e_1 + e_2 \lessdot_3 0}$$

with $\lessdot_1, \lessdot_2, \lessdot_3 \in \{\leq, <\}$ such that...

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

In arithmetic

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

$$\frac{e_1 - y \lessdot_1 0 \quad e_2 + y \lessdot_2 0}{e_1 + e_2 \lessdot_3 0}$$

with $\lessdot_1, \lessdot_2, \lessdot_3 \in \{\leq, <\}$ such that...

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

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

2. CDSAT

Context

CDCL (Conflict-Driven Clause Learning)

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

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

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

Context

CDCL (Conflict-Driven Clause Learning)

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

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) [BGLS19, BGLS20]

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

Context

CDCL (Conflict-Driven Clause Learning)

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

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) [BGLS19, BGLS20]

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

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

Context

CDCL (Conflict-Driven Clause Learning)

- procedure for deciding the satisfiability of Boolean formulae
- uses assignments of Boolean values to variables, e.g., $l \leftarrow$true

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

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

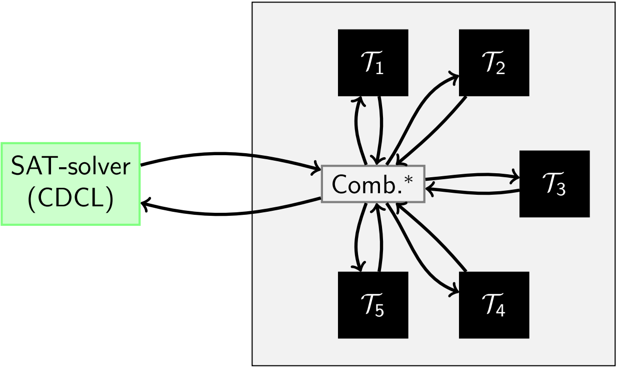
CDSAT (Conflict-Driven Satisfiability) [BGLS19, BGLS20]

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

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

CDSAT can also provide proofs of unsat

Traditional architecture of SMT-solving



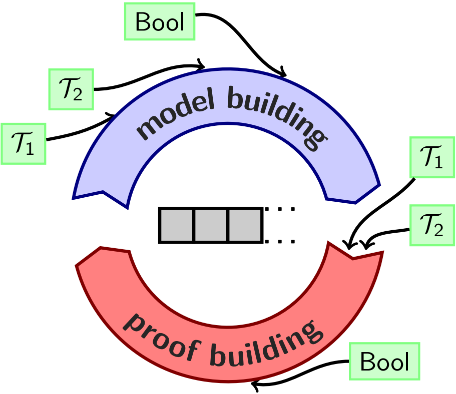
* e.g. equality sharing / Nelson-Oppen [NO79]

In CDSAT

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

As in MCSAT, trail contains

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

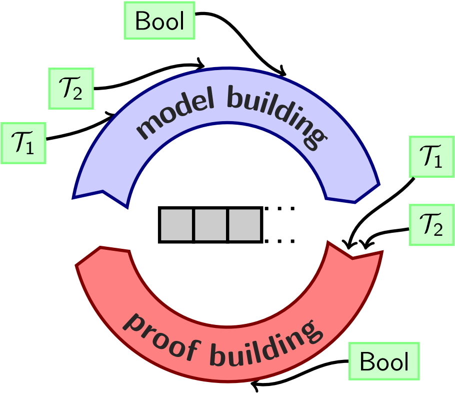


In CDSAT

...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:

$$(t_1 < x), (x < t_2) \vdash t_1 < t_2$$

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

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

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 $\bar{l}$
- Soundness requirement:
  Every model of the premisses is a model of the conclusion:
  $(t_1 \leftarrow \mathfrak{c}_1), \ldots, (t_k \leftarrow \mathfrak{c}_k) \models (l \leftarrow \mathfrak{b})$

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: $(l \leftarrow \text{true})$ as l and $(l \leftarrow \text{false})$ as $\overline{l}$
- Soundness requirement:
  Every model of the premisses is a model of the conclusion:
  $(t_1 \leftarrow \mathfrak{c}_1), \dots, (t_k \leftarrow \mathfrak{c}_k) \models (l \leftarrow \mathfrak{b})$

Examples:

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

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

What is a theory module? (Equality inferences)

All theory modules have the equality inferences:

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

$$
\begin{array}{rlr}
\vdash_{\mathcal{T}} & t_1 \simeq t_1 & \text{reflexivity} \\
t_1 \simeq t_2 \vdash_{\mathcal{T}} & t_2 \simeq t_1 & \text{symmetry} \\
t_1 \simeq t_2, t_2 \simeq t_3 \vdash_{\mathcal{T}} & t_1 \simeq t_3 & \text{transitivity}
\end{array}
$$

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

Each assignment on the trail has a *level*

(index of highest decision in transitive justification of the assignement)

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

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

Example (trail grows from left to right):

$${}_{\emptyset\vdash}(x \simeq z), {}_{\emptyset\vdash}(y \simeq z), {}_{?}(x{\leftarrow}\sqrt{2}), {}_{?}(y{\leftarrow}\mathrm{blue}), {}_{?}(x{\leftarrow}\mathrm{red}), {}_{H\vdash}(x \not\simeq y)$$

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: Search rules

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

Decide

$$\Gamma \longrightarrow \Gamma, {}_{?}(t{\leftarrow}\mathfrak{c})$$

Deduce

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

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

CDSAT: Search rules

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

Decide

$$\Gamma \longrightarrow \Gamma, {}_{?}(t{\leftarrow}\mathfrak{c})$$

Deduce

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

Conflict

$$\Gamma \longrightarrow \langle \Gamma; J, (t{\leftarrow}\overline{\mathfrak{b}}) \rangle \qquad \text{if } J \vdash_{\mathcal{T}} (t{\leftarrow}\mathfrak{b}) \text{ and } J \subseteq \Gamma, \text{ and } t{\leftarrow}\overline{\mathfrak{b}} \text{ is in } \Gamma \text{ and conflict level is } > 0$$

Fail

$$\Gamma \longrightarrow \text{unsat} \qquad \text{if } J \vdash_{\mathcal{T}} (t{\leftarrow}\mathfrak{b}) \text{ and } J \subseteq \Gamma, \text{ and } t{\leftarrow}\overline{\mathfrak{b}} \text{ is in } \Gamma \text{ and conflict level is } 0$$

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

CDSAT: Search rules

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

Decide

$$\Gamma \longrightarrow \Gamma, {}_{?}(t \leftarrow \mathfrak{c}) \qquad \text{if } \ldots$$

Deduce

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

Conflict

$$\Gamma \longrightarrow \langle \Gamma; J, (t \leftarrow \overline{\mathfrak{b}}) \rangle \qquad \text{if } J \vdash_{\mathcal{T}} (t \leftarrow \mathfrak{b}) \text{ and } J \subseteq \Gamma, \text{ and } t \leftarrow \overline{\mathfrak{b}} \text{ is in } \Gamma \text{ and conflict level is } > 0$$

Fail

$$\Gamma \longrightarrow \text{unsat} \qquad \text{if } J \vdash_{\mathcal{T}} (t \leftarrow \mathfrak{b}) \text{ and } J \subseteq \Gamma, \text{ and } t \leftarrow \overline{\mathfrak{b}} \text{ is in } \Gamma \text{ and conflict level is } 0$$

Extra side-conditions "..." to ensure termination

(no impact on soundness)

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

CDSAT: Conflict analysis rules

Resolve

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

Learn

$\langle \Gamma; E \uplus H \rangle \quad \longrightarrow \quad \Gamma^{\leq m}, {}_{E\vdash} L \quad$ if L is a "clausal form" of H and ...

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

CDSAT: Conflict analysis rules

Resolve

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

Learn

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

CDSAT: Conflict analysis rules

Resolve

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

Learn

$$\langle \Gamma; E \uplus H \rangle \quad \longrightarrow \quad \Gamma^{\leq m}, {}_{E}\!\vdash L \quad \text{if } L \text{ is a ``clausal form'' of } H \text{ and } \ldots$$

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

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

CDSAT: Conflict analysis rules

Resolve

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

Learn

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

$\Gamma^{\leq m}$: the pruning of trail Γ, removing all assignments of level $> m$
Clausal forms of H **reify** H in Boolean logic:

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

CDSAT: Conflict analysis rules

Resolve

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

Learn

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

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

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

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

CDSAT: Conflict analysis rules

Resolve

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

Learn

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

Undo

$\langle\Gamma; E \uplus \{A\}\rangle \longrightarrow \Gamma^{\leq m-1}$ if A is a first-order decision and ...

UndoDecide

$\langle\Gamma; E \uplus \{L\}\rangle \longrightarrow \Gamma^{\leq m-1}, {}_{?}\overline{L}$ if ${}_{H}\vdash L$ is in Γ, and ...

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

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

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

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w{-}2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w{-}2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u{\leftarrow}\mathfrak{c}$	?	1

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w{-}2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u{\leftarrow}\mathfrak{c}$	?	1
5	$v{\leftarrow}\mathfrak{c}$	?	2

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w-2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u \leftarrow \mathfrak{c}$	?	1
5	$v \leftarrow \mathfrak{c}$	?	2
6	$a[i{:=}\,v][j] \leftarrow \mathfrak{c}$	?	3

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w{-}2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u{\leftarrow}\mathfrak{c}$	?	1
5	$v{\leftarrow}\mathfrak{c}$	?	2
6	$a[i{:=}\,v][j]{\leftarrow}\mathfrak{c}$	?	3
7	$w{\leftarrow}0$	?	4

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w{-}2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u{\leftarrow}\mathfrak{c}$	?	1
5	$v{\leftarrow}\mathfrak{c}$	?	2
6	$a[i{:=}\,v][j]{\leftarrow}\mathfrak{c}$	?	3
7	$w{\leftarrow}0$	?	4
8	$f(a[i{:=}\,v][j]){\leftarrow}0$	?	5

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w-2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u \leftarrow \mathfrak{c}$	?	1
5	$v \leftarrow \mathfrak{c}$	?	2
6	$a[i{:=}\,v][j] \leftarrow \mathfrak{c}$	?	3
7	$w \leftarrow 0$	?	4
8	$f(a[i{:=}\,v][j]) \leftarrow 0$	?	5
9	$f(u) \leftarrow -2$	?	6

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w{-}2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u{\leftarrow}\mathfrak{c}$	?	1
5	$v{\leftarrow}\mathfrak{c}$	?	2
6	$a[i{:=}\,v][j]{\leftarrow}\mathfrak{c}$	?	3
7	$w{\leftarrow}0$	?	4
8	$f(a[i{:=}\,v][j]){\leftarrow}0$	?	5
9	$f(u){\leftarrow}{-2}$	?	6
10	$u \simeq a[i{:=}\,v][j]$	$\{4,6\}$	3

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\ v][j]) \simeq w$	$\{\}$	0
1	$w-2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u \leftarrow \mathfrak{c}$	?	1
5	$v \leftarrow \mathfrak{c}$	?	2
6	$a[i{:=}\ v][j] \leftarrow \mathfrak{c}$	?	3
7	$w \leftarrow 0$	?	4
8	$f(a[i{:=}\ v][j]) \leftarrow 0$	?	5
9	$f(u) \leftarrow -2$	?	6
10	$u \simeq a[i{:=}\ v][j]$	$\{4,6\}$	3
11	$f(u) \not\simeq f(a[i{:=}\ v][j])$	$\{8,9\}$	6

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w-2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u \leftarrow \mathfrak{c}$	?	1
5	$v \leftarrow \mathfrak{c}$	?	2
6	$a[i{:=}\,v][j] \leftarrow \mathfrak{c}$	?	3
7	$w \leftarrow 0$	?	4
8	$f(a[i{:=}\,v][j]) \leftarrow 0$	?	5
9	$f(u) \leftarrow -2$	?	6
10	$\mathbf{u} \simeq \mathbf{a}[\mathbf{i}{:=}\,\mathbf{v}][\mathbf{j}]$	$\{4,6\}$	3
11	$\mathbf{f}(\mathbf{u}) \not\simeq \mathbf{f}(\mathbf{a}[\mathbf{i}{:=}\,\mathbf{v}][\mathbf{j}])$	$\{8,9\}$	6
	conflict E^1: $\{10, 11\}$		6

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w{-}2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u{\leftarrow}\mathfrak{c}$	?	1
5	$v{\leftarrow}\mathfrak{c}$	?	2
6	$a[i{:=}\,v][j]{\leftarrow}\mathfrak{c}$	?	3
7	$w{\leftarrow}0$	?	4
8	$f(a[i{:=}\,v][j]){\leftarrow}0$	?	5
9	$f(u){\leftarrow}{-}2$	?	6
10	$\mathbf{u} \simeq \mathbf{a}[\mathbf{i}{:=}\,\mathbf{v}][\mathbf{j}]$	$\{4,6\}$	3
11	$\mathbf{f}(\mathbf{u}) \not\simeq \mathbf{f}(\mathbf{a}[\mathbf{i}{:=}\,\mathbf{v}][\mathbf{j}])$	$\{8,9\}$	6
	conflict E^1: $\{10,11\}$		6

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w{-}2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u{\leftarrow}\mathfrak{c}$	?	1
5	$v{\leftarrow}\mathfrak{c}$	?	2
6	$a[i{:=}\,v][j]{\leftarrow}\mathfrak{c}$	?	3
7	$u \simeq a[i{:=}\,v][j]$	$\{4,6\}$	3
8	$f(u) \simeq f(a[i{:=}\,v][j])$	$\{7\}$	3

An example with arithmetic, arrays, congruence

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

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w-2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u \leftarrow \mathfrak{c}$	?	1
5	$v \leftarrow \mathfrak{c}$	?	2
6	$a[i{:=}\,v][j] \leftarrow \mathfrak{c}$	?	3
7	$w \leftarrow 0$	?	4
8	$f(a[i{:=}\,v][j]) \leftarrow 0$	?	5
9	$f(u) \leftarrow -2$	?	6
10	$\mathbf{u \simeq a[i{:=}\,v][j]}$	$\{4,6\}$	3
11	$\mathbf{f(u) \not\simeq f(a[i{:=}\,v][j])}$	$\{8,9\}$	6
	conflict E^1: $\{10,11\}$		6

id	trail items	just.	lev.
0	$f(a[i{:=}\,v][j]) \simeq w$	$\{\}$	0
1	$w-2 \simeq f(u)$	$\{\}$	0
2	$i \simeq j$	$\{\}$	0
3	$u \simeq v$	$\{\}$	0
4	$u \leftarrow \mathfrak{c}$	?	1
5	$v \leftarrow \mathfrak{c}$	?	2
6	$a[i{:=}\,v][j] \leftarrow \mathfrak{c}$	?	3
7	$u \simeq a[i{:=}\,v][j]$	$\{4,6\}$	3
8	$f(u) \simeq f(a[i{:=}\,v][j])$	$\{7\}$	3

...

Termination and Soundness

Termination:

Theorem: If the global basis $\mathcal{B}$ is finite, CDSAT terminates.

Termination and Soundness

Termination:

Theorem: If the global basis $\mathcal{B}$ is finite, CDSAT terminates.

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

Termination and Soundness

Termination:

Theorem: If the global basis $\mathcal{B}$ is finite, CDSAT terminates.

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

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

Termination and Soundness

Termination:

Theorem: If the global basis $\mathcal{B}$ is finite, CDSAT terminates.

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

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

If the local bases of $\mathcal{T}_1, \ldots, \mathcal{T}_n$ satisfy some (collective) properties,
then it is possible to define a finite global basis $\mathcal{B}$ for $\bigcup_{k=1}^{n} \mathcal{T}_k$.

Termination and Soundness

Termination:

Theorem: If the global basis $\mathcal{B}$ is finite, CDSAT terminates.

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

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

If the local bases of $\mathcal{T}_1, \ldots, \mathcal{T}_n$ satisfy some (collective) properties,
then it is possible to define a finite global basis $\mathcal{B}$ for $\bigcup_{k=1}^{n} \mathcal{T}_k$.

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

What happens if we never get unsat?

Do we have a model?

What happens if we never get unsat?

Do we have a model?

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

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

What happens if we never get unsat?

Do we have a model?

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

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

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

What happens if we never get unsat?

Do we have a model?

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

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

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

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

What happens if we never get unsat?

Do we have a model?

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

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

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

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

Theorem: Assume $\mathcal{T}_0$ has a complete module, and all other theories have $\mathcal{T}_0$-complete modules.
If CDSAT cannot make any further transitions, then the trail describes a model for the union of the (extended) theories.

Theory modules given as examples in our papers

- EUF

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

- Arrays: similar, except for extensionality

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

Theory modules given as examples in our papers

- EUF
$$(t_i \simeq u_i)_{i=1\ldots n}, (f(t_1,\ldots,t_n) \not\simeq f(u_1,\ldots,u_n)) \vdash_{\mathsf{EUF}} \quad \bot$$
- Arrays: similar, except for extensionality
- LRA: evaluation inference, Fourier-Motzkin resolution inference as in MCSAT, etc
- Black box procedure for equality-sharing: coarse-grain inferences
$$l_1{\leftarrow}\mathfrak{b}_1,\ldots,l_n{\leftarrow}\mathfrak{b}_n \vdash_{\mathcal{T}} \bot$$
where $l_1,\ldots,l_n$ are formulæ, and the conjunction of the literals corresponding to the Boolean assignments $l_1{\leftarrow}b_1,\ldots,l_n{\leftarrow}b_n$ is $\mathcal{T}$-unsatisfiable (as detected by the black box)

Theory modules given as examples in our papers

- EUF ($\mathcal{T}_0$-complete for all $\mathcal{T}_0$)

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

- Arrays: similar, except for extensionality

  ($\mathcal{T}_0$-complete for all $\mathcal{T}_0$ such that...)

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

  ($\mathcal{T}_0$-complete for all $\mathcal{T}_0$ imposing $|Q|$ infinite)

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

$$l_1 \leftarrow \mathfrak{b}_1, \ldots, l_n \leftarrow \mathfrak{b}_n \vdash_{\mathcal{T}} \bot$$

  where $l_1, \ldots, l_n$ are formulæ, and the conjunction of the literals corresponding to the Boolean assignments $l_1 \leftarrow b_1, \ldots, l_n \leftarrow b_n$ is $\mathcal{T}$-unsatisfiable (as detected by the black box)
  ($\mathcal{T}_0$-complete for all $\mathcal{T}_0$ imposing the cardinality of all known sorts but Bool to be countably infinite)

3. Proofs in CDSAT

Theory proofs

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

Theory proofs

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

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

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:

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

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

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

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

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
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 object to check.

LCF: answers that are correct-by-construction

Other "proof format":

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

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

LCF: answers that are correct-by-construction

Other "proof format":

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

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

- A type `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 object to check. But the LCF architecture [Mil79, GMW79] can be used to ensure the correctness of answers. LCF in a nutshell:

- A type `theorem` is defined for provable formulae in a module of the prover called kernel
- The definition of `theorem` is hidden outside the kernel

LCF: answers that are correct-by-construction

Other "proof format":

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

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

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

LCF: answers that are correct-by-construction

Other "proof format":

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

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

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

LCF: answers that are correct-by-construction

Other "proof format":

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

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

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

LCF: answers that are correct-by-construction

Other "proof format":

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

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

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

No proof object needs to be built in memory

CDSAT is well-suited to the LCF approach 1/2

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

```
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.

4. Quantified satisfiability

Quantifiers

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

quantifier elimination: For any formula A, there exists a quantifier-free formula B such that $[\![A]\!] = [\![B]\!]$

Quantifiers

Due to the connection between MCSAT and quantifier elimination, we recently explored how MCSAT features could be used to extend Yices to support quantifiers.
quantifier elimination: For any formula A, there exists a quantifier-free formula B such that $[\![A]\!] = [\![B]\!]$

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

Quantifiers

Due to the connection between MCSAT and quantifier elimination, we recently explored how MCSAT features could be used to extend Yices to support quantifiers.
quantifier elimination: For any formula A, there exists a quantifier-free formula B such that $[\![A]\!] = [\![B]\!]$

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

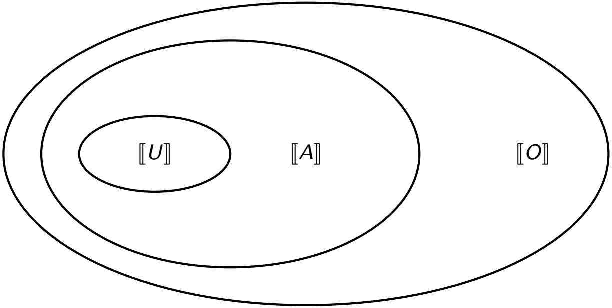
We have a better approach that we applied to NRA (non-linear arithmetic) and BV (bitvectors), on top of Yices/MCSAT.

Approximations

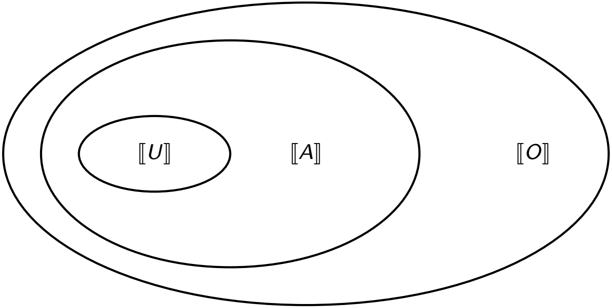
Idea: if the only reason to produce B from A is to decide whether A is satisfiable, it may not be necessary to compute B exactly.
Approximations may suffice.

Def:

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



Basic idea of lazy quantifier elimination

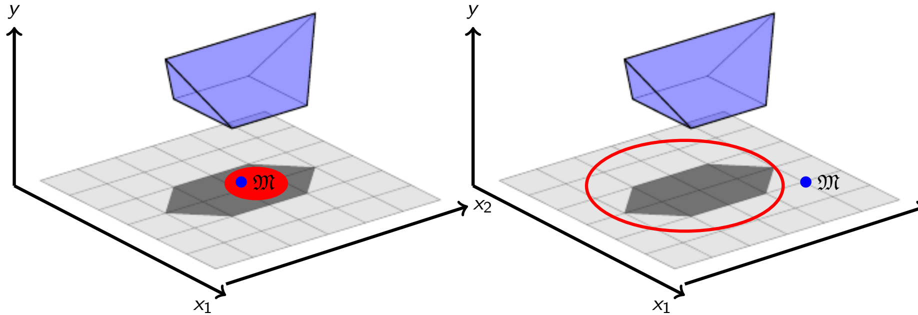


Start with $U =$ false and $O =$ true, and iteratively refine U and O until either U is sat or O is unsat.
Worst case: you may end up computing a quantifier-free formula B such that $[\![A]\!] = [\![B]\!]$.
In practice, you hope the algorithm will stop earlier than that.

Question: how do we refine the approximations iteratively?

One quantifier at a time

Quantifier elimination:
Given $\exists y\ F(\vec{x}, y)$ with quantifier-free $F(\vec{x}, y)$, produce quantifier-free $B(\vec{x})$ with $(\exists y\ F(\vec{x}, y)) \Leftrightarrow B(\vec{x})$ provable.
Model generalization:
If additionally given $\mathfrak{M}$ satisfying $\exists y\ F(\vec{x}, y)$, produce quantifier-free $U(\vec{x})$ satisfied by $\mathfrak{M}$, with $U(\vec{x}) \Rightarrow (\exists y\ F(\vec{x}, y))$ provable.
Model interpolation:
If additionally given $\mathfrak{M}$ *not* satisfying $\exists y\ F(\vec{x}, y)$, produce quantifier-free $O(\vec{x})$ *not* satisfied by $\mathfrak{M}$, with $(\exists y\ F(\vec{x}, y)) \Rightarrow O(\vec{x})$ provable.



In **blue**: $F(x_1, x_2, y)$; its **grey shadow**: $\exists y\ F(\vec{x}, y)$;
in **red**: the under-approximation $U(x_1, x_2)$ / the over-approximation $O(x_1, x_2)$.

A satisfiability algorithm for a slightly more general question

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

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

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

A satisfiability algorithm for a slightly more general question

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

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

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

This generalizes the standard satisfiability question:
"Given a formula $A(\vec{x})$, produce either

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

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

- UNSAT(O): return UNSAT
- SAT(U): return SAT

The (recursive) satisfiability algorithm is a 2-player game

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

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

The (recursive) satisfiability algorithm is a 2-player game

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

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

Algorithm `solve`:

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

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

The (recursive) satisfiability algorithm is a 2-player game

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

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

Algorithm `solve`:

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

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

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

The (recursive) satisfiability algorithm is a 2-player game

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

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

Algorithm `solve`:

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

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

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

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

The (recursive) satisfiability algorithm is a 2-player game

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

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

Algorithm `solve`:

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

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

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

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

(3) Ask **whether $\mathfrak{M}_{\vec{z}}$ extends to a model $\mathfrak{M}$ of $L(\vec{z}, \vec{x})$**.

The (recursive) satisfiability algorithm is a 2-player game

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

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

Algorithm `solve`:

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

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

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

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

(3) Ask **whether $\mathfrak{M}_{\vec{z}}$ extends to a model $\mathfrak{M}$ of $L(\vec{z}, \vec{x})$**.

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

The (recursive) satisfiability algorithm is a 2-player game

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

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

Algorithm `solve`:

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

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

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

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

(3) Ask **whether $\mathfrak{M}_{\vec{z}}$ extends to a model $\mathfrak{M}$ of $L(\vec{z},\vec{x})$**.

- If not, apply **model interpolation** on $\mathfrak{M}_{\vec{z}}$ and $L(\vec{z},\vec{x})$ to get $O(\vec{z})$;
  return UNSAT($O(\vec{z})$).
- Otherwise, **recursively call `solve` on $\mathfrak{M}$ and $A_{rec}(\vec{z},\vec{x},\vec{y})$**,

  and inspect the result:

  - UNSAT($O_{\text{rec}}(\vec{z},\vec{x})$)
    apply **model generalization** on $\mathfrak{M}$ and $F(\vec{z},\vec{x}) \wedge \neg O_{\text{rec}}(\vec{z},\vec{x})$ to get $U(\vec{z})$; return SAT($U(\vec{z})$).

The (recursive) satisfiability algorithm is a 2-player game

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

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

Algorithm `solve`:

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

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

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

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

(3) Ask **whether $\mathfrak{M}_{\vec{z}}$ extends to a model $\mathfrak{M}$ of $L(\vec{z}, \vec{x})$**.

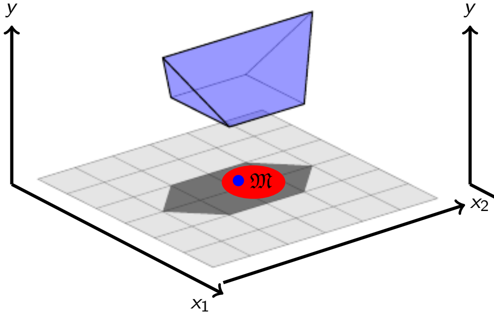
- If not, apply **model interpolation** on $\mathfrak{M}_{\vec{z}}$ and $L(\vec{z}, \vec{x})$ to get $O(\vec{z})$; return UNSAT($O(\vec{z})$).
- Otherwise, **recursively call** `solve` **on $\mathfrak{M}$ and $A_{rec}(\vec{z}, \vec{x}, \vec{y})$**,

  and inspect the result:
  - UNSAT($O_{\text{rec}}(\vec{z}, \vec{x})$)
    apply **model generalization** on $\mathfrak{M}$ and $F(\vec{z}, \vec{x}) \wedge \neg O_{\text{rec}}(\vec{z}, \vec{x})$ to get $U(\vec{z})$; return SAT($U(\vec{z})$).
  - SAT($U_{\text{rec}}(\vec{z}, \vec{x})$)
    Set $L(\vec{z}, \vec{x}) := L(\vec{z}, \vec{x}) \wedge \neg U_{\text{rec}}(\vec{z}, \vec{x})$ and go back to (3).

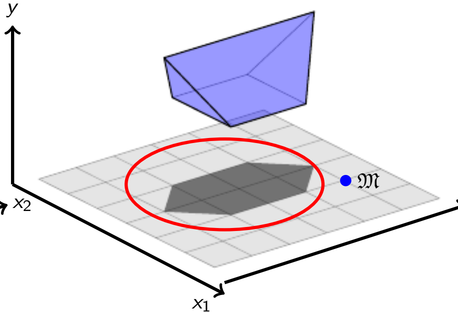
How to answer the 3 kinds of queries

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

Model generalization



Model interpolation



It depends on the theory $\mathcal{T}$. At SRI, we have implemented those procedures for: the *Booleans*, the theory of *bitvectors*, *real arithmetic* (linear and non-linear).
In those theories, we can apply procedure `solve` to lazily eliminate quantifiers in the view of determining satisfiability of any formula.

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

Implementation and related works

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

Implementation and related works

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

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

Implementation and related works

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

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

See also related works:

- Bjørner and Janota's algorithm for "playing with quantified satisfaction", inspired by QBF [BJ15] and used in z3. A two-player game (one wanting to satisfy A, the other one $\neg A$). Based on model projection and unsat cores, but no model interpolation used.
- Monniaux's work on quantifier elimination [Mon08, Mon10]. It uses a ground SMT-solver as a black box (for purely existential problems), and also performs some QE-elimination steps (e.g., FM resolutions) independently from the SMT-solver.
- The ANR Decert work on Linear Integer arithmetic, which extends Fourier-Motzkin with simplex-based techniques [BCC+12]

```ocaml
type answer =
  | Unsat of Term.t
  | Sat of Term.t

let sat_answer x game reason =
  let open Game in
  let model = match x with
    | `over  -> Context.get_model game.context_over ~keep_subst:true
    | `under -> Context.get_model game.context_under ~keep_subst:true
  in
  let true_of_model = Term.(reason &&& game.ground) in
  let gen_model =
    Model.generalize_model model true_of_model game.newvars `YICES_GEN_DEFAULT
  in
  Term.(andN gen_model)

let rec solve game model =
  match Context.check_with_model game.context_over model.model model.support with
  | `STATUS_UNSAT ->
    let interpolant = Context.get_model_interpolant game.context_over in
    Unsat Term.(not1 interpolant)

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

Termination of algorithm `solve`

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

Termination of algorithm `solve`

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

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

Termination of algorithm `solve`

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

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

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

Termination of algorithm `solve`

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

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

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

All of these theories are decidable (-ish).

Related work and future work

Investigate related approaches:

- The ANR Decert work on Linear Integer arithmetic, which extends Fourier-Motzkin with simplex-based techniques [BCC+12]
- Monniaux's work on quantifier elimination [Mon08, Mon10]. It uses a ground SMT-solver as a black box (for purely existential problems), and also performs some QE-elimination steps (e.g., FM resolutions) independently from the SMT-solver.
- Dutertre's work on solving "EF problems" ($\exists\forall$) in Yices, also relying on a ground SMT-solver considered as a black box.

How would the Bjørner-Janota approach work in a combination of theories?
Just as our CDSAT system generalises MCSAT to a combination of theories, what would be the equivalent for the Bjørner-Janota approach?

Questions?

F. Bobot, S. Conchon, E. Contejean, M. Iguernelala, A. Mahboubi, A. Mebsout, and G. Melquiond.
A simplex-based extension of fourier-motzkin for solving linear integer arithmetic.
In B. Gramlich, D. Miller, and U. Sattler, editors, *Automated Reasoning*, pages 67–81. Springer Berlin Heidelberg, 2012.

M. P. Bonacina, S. Graham-Lengrand, and N. Shankar.
Conflict-driven satisfiability for theory combination: Transition system and completeness.
J. of Automated Reasoning, 64(3):579–609, 2019.

M. P. Bonacina, S. Graham-Lengrand, and N. Shankar.
Conflict-driven satisfiability for theory combination: Lemmas, modules, and proofs, 2020.
Submitted.

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