# Proving theorems with quantifier elimination techniques

Stéphane Graham-Lengrand

Upscale, 9th October 2019

# Quantifier elimination

A property that is very simple to express:

A theory $\mathcal{T}$ admits quantifier elimination if for every formula $A$ there is a quantifier-free formula $B$ such that $\mathcal{T} \models A \Leftrightarrow B$ (with $\mathrm{fv}(B) \subseteq \mathrm{fv}(A)$)

# Quantifier elimination

A property that is very simple to express:

A theory $\mathcal{T}$ admits quantifier elimination if for every formula $A$ there is a quantifier-free formula $B$ such that $\mathcal{T} \models A \Leftrightarrow B$ (with $\mathrm{fv}(B) \subseteq \mathrm{fv}(A)$)

The property is useful for showing the decidability of provability/satisfiability in $\mathcal{T}$:
If

1. it can be shown constructively that $\mathcal{T}$ admits quantifier elimination (there is an algorithm that produces $B$ from $A$),
2. and provability/satisfiability in $\mathcal{T}$ of closed quantifier-free formulae is decidable,

then provability/satisfiability in $\mathcal{T}$ of all formulae is decidable.

# Examples

- Linear arithmetic over natural numbers / integers
  Presburger, 1929
- Linear arithmetic over rational / real numbers
  Fourier, 1827 - Motzkin, 1936
- Non-linear arithmetic over real / complex numbers
  Tarski, 1948
- Abelian groups — Szmielew, 1955
- Absolutely free term algebras — Malcev, 1971

# The interesting quantified form

$\mathcal{T}$ admits quantifier elimination as soon as formulae of the form

$$\exists y(l_1 \wedge \cdots \wedge l_m)$$

(where $l_1, \ldots, l_n$ are literals with free variables in $\{x_1, \ldots, x_n, y\}$) can be transformed into an equivalent quantifier-free form $B$ (with free variables in $\{x_1, \ldots, x_n\}$).

# The interesting quantified form

$\mathcal{T}$ admits quantifier elimination as soon as formulae of the form

$$\exists y(l_1 \wedge \cdots \wedge l_m)$$

(where $l_1, \ldots, l_n$ are literals with free variables in $\{x_1, \ldots, x_n, y\}$) can be transformed into an equivalent quantifier-free form $B$ (with free variables in $\{x_1, \ldots, x_n\}$).

Indeed, quantifiers can then be eliminated from any formula $A$, by induction on $A$:

# The interesting quantified form

$\mathcal{T}$ admits quantifier elimination as soon as formulae of the form

$$\exists y(l_1 \wedge \cdots \wedge l_m)$$

(where $l_1, \ldots, l_n$ are literals with free variables in $\{x_1, \ldots, x_n, y\}$) can be transformed into an equivalent quantifier-free form $B$ (with free variables in $\{x_1, \ldots, x_n\}$).

Indeed, quantifiers can then be eliminated from any formula $A$, by induction on $A$:

- $\exists yA$:
  - eliminate quantifiers from $A$ to get $\exists yB$ with quantifier-free $B$,
  - turn $B$ into DNF,
  - distribute $\exists y$ over disjunctions
    to get a disjunction of formulae of the form $\exists y(l_1 \wedge \cdots \wedge l_m)$
  - apply the main hypothesis;
- $\forall yA$: express it first as $\neg\exists y\neg A$, then eliminate quantifiers from $\neg A$ and continue as above;
- propositional connectives: straightforward

# Semantical interpretation of the transformation

Writing $\overrightarrow{x}$ for $x_1, \ldots, x_n$, the transformation

$$\exists y(l_1(\overrightarrow{x}, y) \wedge \cdots \wedge l_m(\overrightarrow{x}, y)) \rightsquigarrow B(\overrightarrow{x})$$

can be interpreted semantically as follows:
In a $\mathcal{T}$-model $\mathfrak{M}$, the set of $n+1$-tuples satisfying $\mathfrak{M}(l_1 \wedge \cdots \wedge l_m)$ is projected as a set of $n$-tuples satisfying $\mathfrak{M}(B)$.

# Semantical interpretation of the transformation

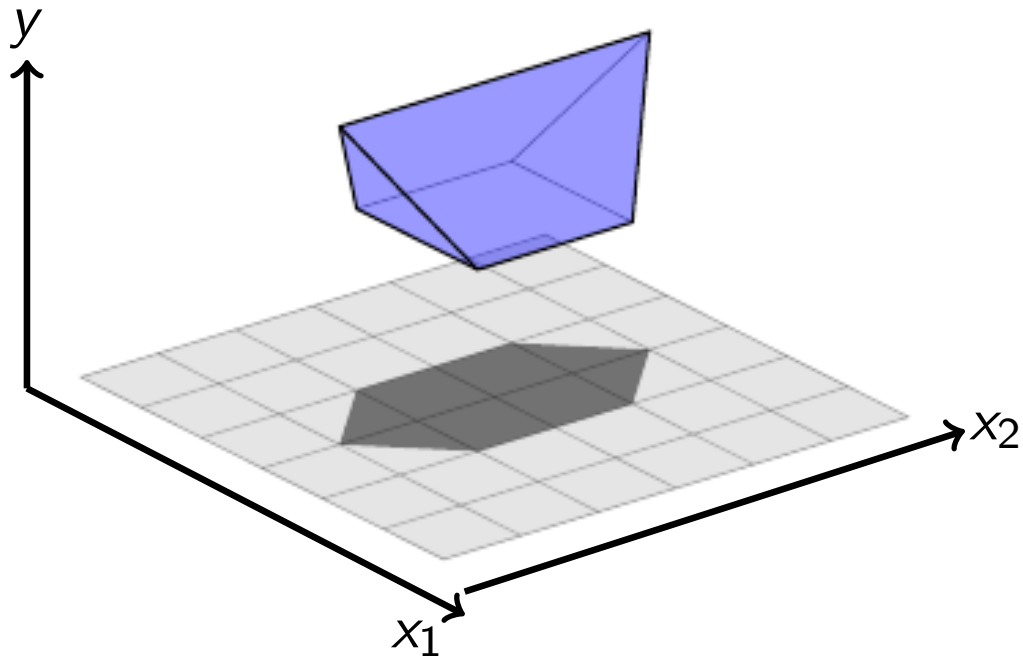
Writing $\overrightarrow{x}$ for $x_1, \ldots, x_n$, the transformation

$$\exists y(l_1(\overrightarrow{x}, y) \wedge \cdots \wedge l_m(\overrightarrow{x}, y)) \rightsquigarrow B(\overrightarrow{x})$$

can be interpreted semantically as follows:

In a $\mathcal{T}$-model $\mathfrak{M}$, the set of $n+1$-tuples satisfying $\mathfrak{M}(l_1 \wedge \cdots \wedge l_m)$ is projected as a set of $n$-tuples satisfying $\mathfrak{M}(B)$.

Example in linear arithmetic:

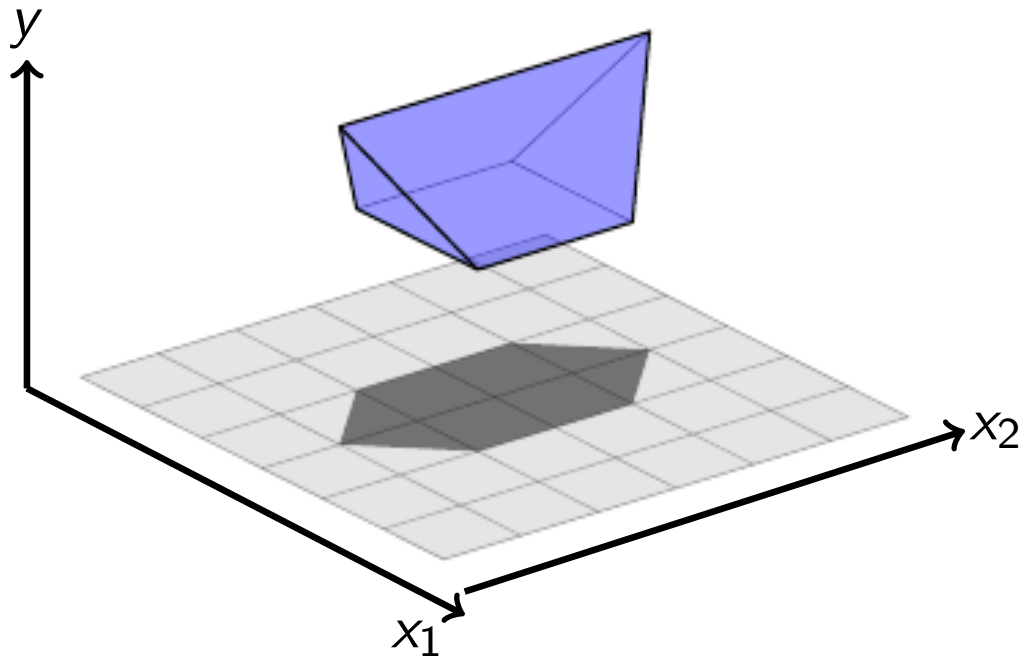


# Semantical interpretation of the transformation

Writing $\overrightarrow{x}$ for $x_1, \ldots, x_n$, the transformation

$$\exists y(l_1(\overrightarrow{x}, y) \wedge \cdots \wedge l_m(\overrightarrow{x}, y)) \rightsquigarrow B(\overrightarrow{x})$$

can be interpreted semantically as follows:
In a $\mathcal{T}$-model $\mathfrak{M}$, the set of $n+1$-tuples satisfying $\mathfrak{M}(l_1 \wedge \cdots \wedge l_m)$ is projected as a set of $n$-tuples satisfying $\mathfrak{M}(B)$.
Example in linear arithmetic:




Similar interpretation in non-linear arithmetic, just with more complex shapes

# Proofs

If quantifier elimination is used to decide that a closed formula $A$ is valid/provable, what would the corresponding proof of $A$ look like?

# Proofs

If quantifier elimination is used to decide that a closed formula $A$ is valid/provable, what would the corresponding proof of $A$ look like?

The basic algorithm exhibits a sequence

$$A = A_0 \rightsquigarrow A_1 \rightsquigarrow \cdots \rightsquigarrow A_p$$

where $A_p$ is a quantifier-free formula with $\mathcal{T} \models A_p$,
and for $0 \leq i < p$, we have $\mathcal{T} \models A_i \Leftrightarrow A_{i+1}$ with $\mathrm{fv}(A_{i+1}) \subseteq \mathrm{fv}(A_i)$

# Proofs

If quantifier elimination is used to decide that a closed formula $A$ is valid/provable, what would the corresponding proof of $A$ look like?

The basic algorithm exhibits a sequence

$$A = A_0 \rightsquigarrow A_1 \rightsquigarrow \cdots \rightsquigarrow A_p$$

where $A_p$ is a quantifier-free formula with $\mathcal{T} \models A_p$,
and for $0 \leq i < p$, we have $\mathcal{T} \models A_i \Leftrightarrow A_{i+1}$ with $\mathrm{fv}(A_{i+1}) \subseteq \mathrm{fv}(A_i)$

We need to aggregate a proof of $\mathcal{T} \vdash A_p$ with proofs of the $\mathcal{T} \vdash A_i \Leftrightarrow A_{i+1}$, which involve

- $\mathcal{T}$-agnostic proofs, e.g., distributing $\exists$ over $\vee$, transforming into DNF, etc
- $\mathcal{T}$-specific proofs for eliminating the "base cases" of QE: the projections.

# Proofs

If quantifier elimination is used to decide that a closed formula $A$ is valid/provable, what would the corresponding proof of $A$ look like?

The basic algorithm exhibits a sequence

$$A = A_0 \rightsquigarrow A_1 \rightsquigarrow \cdots \rightsquigarrow A_p$$

where $A_p$ is a quantifier-free formula with $\mathcal{T} \models A_p$,
and for $0 \leq i < p$, we have $\mathcal{T} \models A_i \Leftrightarrow A_{i+1}$ with $\mathrm{fv}(A_{i+1}) \subseteq \mathrm{fv}(A_i)$

We need to aggregate a proof of $\mathcal{T} \vdash A_p$ with proofs of the $\mathcal{T} \vdash A_i \Leftrightarrow A_{i+1}$, which involve

- $\mathcal{T}$-agnostic proofs, e.g., distributing $\exists$ over $\vee$, transforming into DNF, etc
- $\mathcal{T}$-specific proofs for eliminating the "base cases" of QE: the projections.

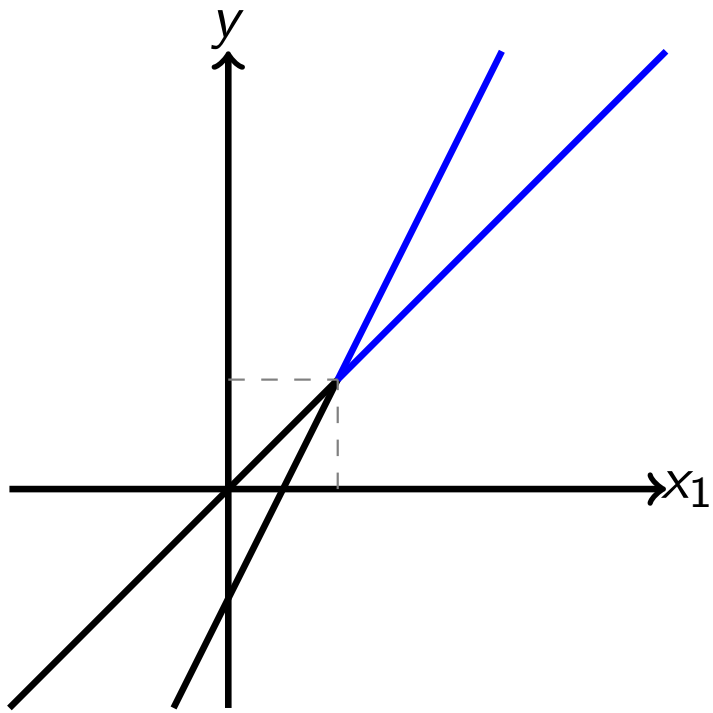
(of course in powerful logics we could use reflexivity and a correctness proof of the algorithm)

# Projection proofs - example in LRA

$$\exists y((x_1 - y \leq 0) \wedge (y - 2x_1 + 1 \leq 0))$$

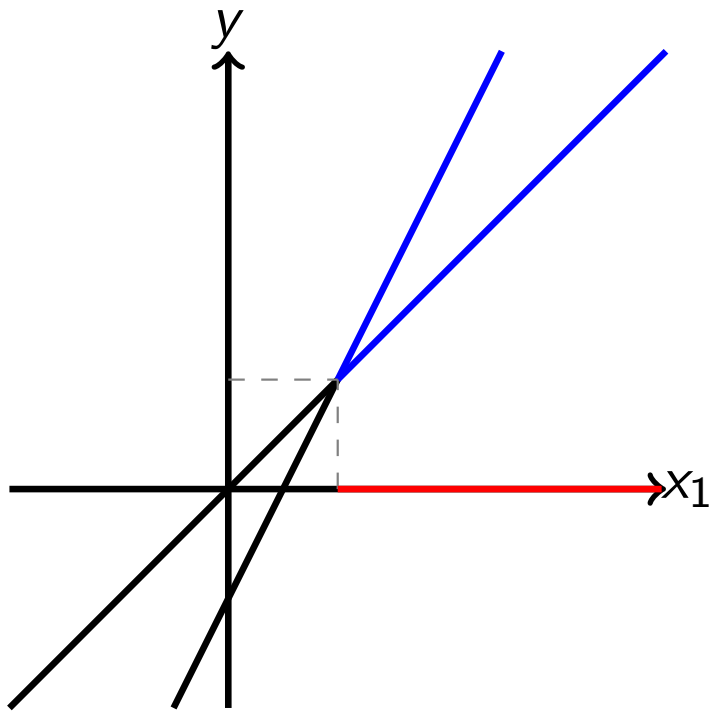
# Projection proofs - example in LRA

$$\exists y((x_1 - y \leq 0) \wedge (y - 2x_1 + 1 \leq 0))$$



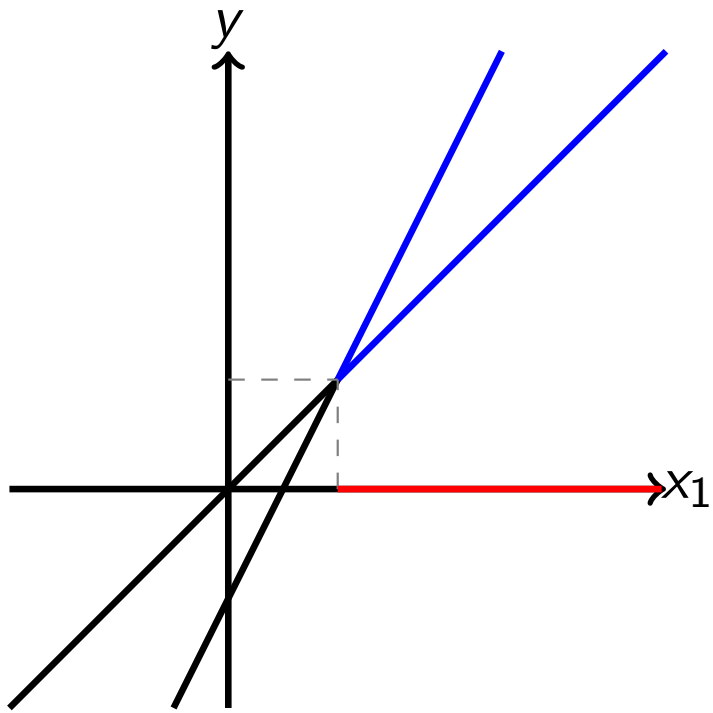
# Projection proofs - example in LRA

$$\exists y((x_1 - y \leq 0) \wedge (y - 2x_1 + 1 \leq 0))$$



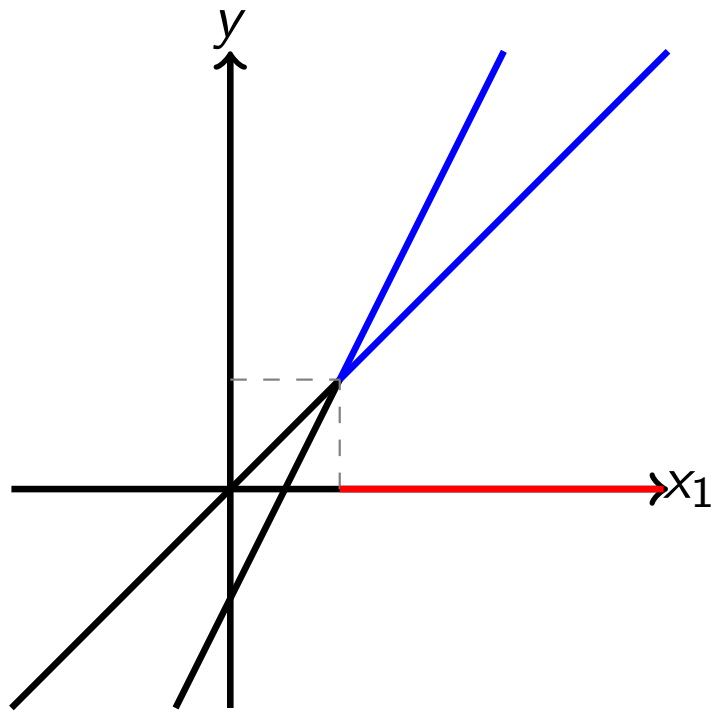
# Projection proofs - example in LRA

$$\exists y((x_1 - y \leq 0) \wedge (y - 2x_1 + 1 \leq 0)) \rightsquigarrow -x_1 + 1 \leq 0$$



# Projection proofs - example in LRA

$$\exists y((x_1 - y \leq 0) \wedge (y - 2x_1 + 1 \leq 0)) \rightsquigarrow -x_1 + 1 \leq 0$$



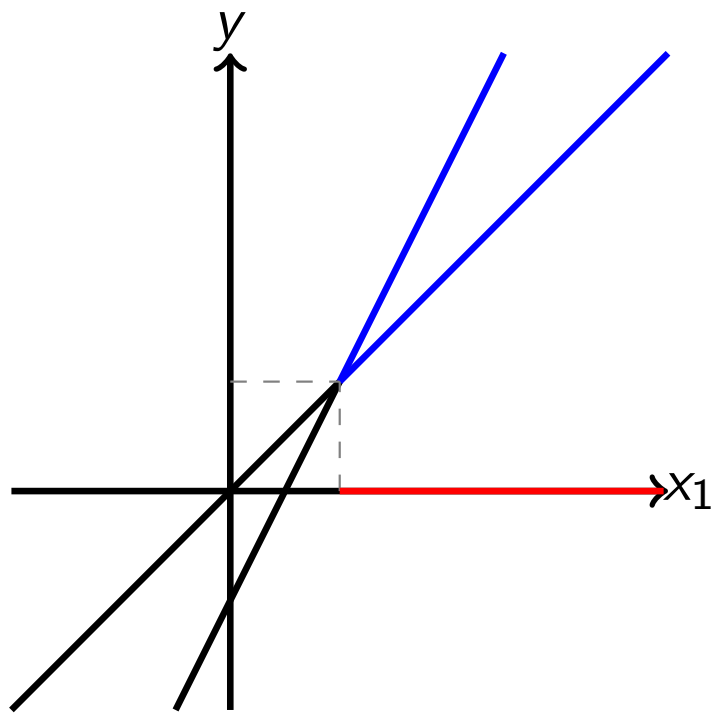

Proof involves the Fourier-Motzkin resolution rule

$$\frac{e_1 - y \leq 0 \quad e_2 + y \leq 0}{e_1 + e_2 \leq 0}$$

with the premisses being normalised in $y$,

# Projection proofs - example in LRA

$$\exists y((x_1 - y \leq 0) \wedge (y - 2x_1 + 1 \leq 0)) \rightsquigarrow -x_1 + 1 \leq 0$$




Proof involves the Fourier-Motzkin resolution rule

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

with the premisses being normalised in $y$,
and $\lessdot_i \in \{\leq, <\}$ such that...

# On “quantifier-free” problems

Special case when formulae are quantifier-free
but have free variables
= target of ground SMT-solving (Satisfiability Modulo Theories)

# On “quantifier-free” problems

Special case when formulae are quantifier-free
but have free variables
= target of ground SMT-solving (Satisfiability Modulo Theories)

Is $A$ provable? Free variables behave as if universally quantified.

Is $A$ satisfiable? Free variables behave as if existentially quantified.

# On "quantifier-free" problems

Special case when formulae are quantifier-free
but have free variables
= target of ground SMT-solving (Satisfiability Modulo Theories)

Is $A$ provable? Free variables behave as if universally quantified.

Is $A$ satisfiable? Free variables behave as if existentially quantified.

Is closed formula $\exists x_1 \ldots \exists x_n B$ (with quantifier-free $B$) satisfiable?

# On "quantifier-free" problems

Special case when formulae are quantifier-free
but have free variables
= target of ground SMT-solving (Satisfiability Modulo Theories)

Is $A$ provable? Free variables behave as if universally quantified.

Is $A$ satisfiable? Free variables behave as if existentially quantified.

Is closed formula $\exists x_1 \ldots \exists x_n B$ (with quantifier-free $B$) satisfiable?

Eliminate $x_n$, then $x_{n-1}$, etc, then $x_1$, then see if you have $\top$ or $\bot$

# On "quantifier-free" problems

Special case when formulae are quantifier-free
but have free variables
= target of ground SMT-solving (Satisfiability Modulo Theories)

Is $A$ provable? Free variables behave as if universally quantified.
Is $A$ satisfiable? Free variables behave as if existentially quantified.

Is closed formula $\exists x_1 \ldots \exists x_n B$ (with quantifier-free $B$) satisfiable?

Eliminate $x_n$, then $x_{n-1}$, etc, then $x_1$, then see if you have $\top$ or $\bot$

This is one of the ways to prove that Boolean resolution is refutationally complete in propositional logic:

$$\frac{C_1 \vee \neg y \quad C_2 \vee y}{C_1 \vee C_2}$$

If $B$ is a propositional CNF (a set of clauses), apply all possible resolutions on $x_n$, keep the new clauses, forget the clauses with $x_n$. Then do the same with $x_{n-1}$, etc. At the end: the set of clauses is $\emptyset$ ($B$ is sat) or $\{\bot\}$ ($B$ is unsat). In the latter case, this quantifier elimination process produces a proof of $B \Rightarrow \bot$ as a resolution proof.

# CDCL and Boolean resolution

Boolean resolution offers a proof format for unsatisfiable (quantifier-free) Boolean CNF like $B$.

# CDCL and Boolean resolution

Boolean resolution offers a proof format for unsatisfiable (quantifier-free) Boolean CNF like $B$.

Technically, saturating $B$ by repeated instances of resolution always terminates.

# CDCL and Boolean resolution

Boolean resolution offers a proof format for unsatisfiable (quantifier-free) Boolean CNF like $B$.

Technically, saturating $B$ by repeated instances of resolution always terminates.

Quantifier elimination is one strategy for applying resolutions until $B$ can be determined to be sat or unsat.

# CDCL and Boolean resolution

Boolean resolution offers a proof format for unsatisfiable (quantifier-free) Boolean CNF like $B$.

Technically, saturating $B$ by repeated instances of resolution always terminates.

Quantifier elimination is one strategy for applying resolutions until $B$ can be determined to be sat or unsat.

But we know there's a better one: Conflict-Driven Clause Learning (CDCL), the main algorithm used in SAT-solvers.

- resolutions are performed "lazily"
- in response to explicit model construction attempts for $B$ that "failed" (the truth values that have been attempted are in conflict with one of the clauses)

# Back to LRA

Boolean resolution

$$\frac{C_1 \vee \neg y \quad C_2 \vee y}{C_1 \vee C_2}$$

Fourier-Motzkin resolution

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

Just like Boolean resolution is refutationally complete for (quantifier-free) Boolean logic, Fourier-Motzkin resolution is refutationally complete for (quantifier-free) LRA.

Can be proved by the same quantifier elimination argument.

# Back to LRA

Boolean resolution

$$\frac{C_1 \vee \neg y \quad C_2 \vee y}{C_1 \vee C_2}$$

Fourier-Motzkin resolution

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

Just like Boolean resolution is refutationally complete for (quantifier-free) Boolean logic, Fourier-Motzkin resolution is refutationally complete for (quantifier-free) LRA.
Can be proved by the same quantifier elimination argument.

Just like quantifier elimination gives a strategy for constructing Boolean resolution proofs (though not a great one), quantifier elimination gives a strategy for constructing F-M resolution proofs, (though not a great one).

# Back to LRA

Boolean resolution

$$\frac{C_1 \vee \neg y \quad C_2 \vee y}{C_1 \vee C_2}$$

Fourier-Motzkin resolution

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

Just like Boolean resolution is refutationally complete for (quantifier-free) Boolean logic, Fourier-Motzkin resolution is refutationally complete for (quantifier-free) LRA.
Can be proved by the same quantifier elimination argument.

Just like quantifier elimination gives a strategy for constructing Boolean resolution proofs (though not a great one), quantifier elimination gives a strategy for constructing F-M resolution proofs, (though not a great one).

Is there a better strategy for constructing F-M resolution proofs?
(. . . just like CDCL was a better strategy for constructing Boolean resolution proofs)

# Back to LRA

Boolean resolution

$$\frac{C_1 \vee \neg y \quad C_2 \vee y}{C_1 \vee C_2}$$

Fourier-Motzkin resolution

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

Just like Boolean resolution is refutationally complete for (quantifier-free) Boolean logic, Fourier-Motzkin resolution is refutationally complete for (quantifier-free) LRA.
Can be proved by the same quantifier elimination argument.

Just like quantifier elimination gives a strategy for constructing Boolean resolution proofs (though not a great one), quantifier elimination gives a strategy for constructing F-M resolution proofs, (though not a great one).

Is there a better strategy for constructing F-M resolution proofs? (. . . just like CDCL was a better strategy for constructing Boolean resolution proofs)

Could it be generalised to other theories with similar QE mechanisms?

# A lazy approach to quantifier elimination

CDCL (Conflict-Driven Clause Learning)

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

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

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

# A lazy approach to quantifier elimination

CDCL (Conflict-Driven Clause Learning)

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

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

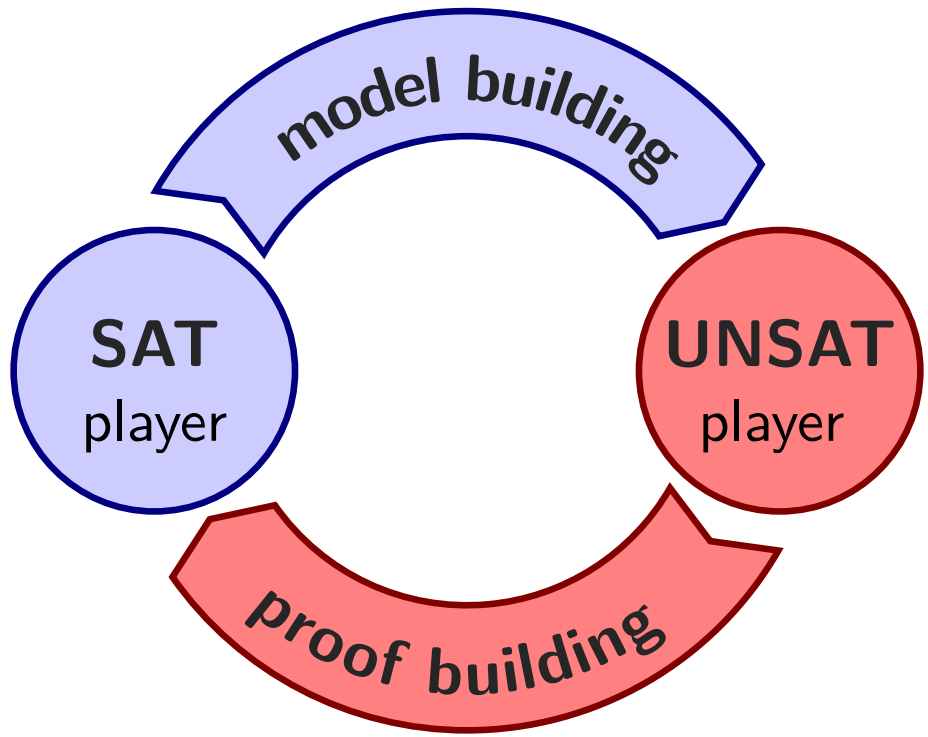
- generalises CDCL to theory reasoning
- uses first-order assignments, e.g., $x \leftarrow \sqrt{2}$

CDSAT (Conflict-Driven Satisfiability) [BGLS17, BGLS17]

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

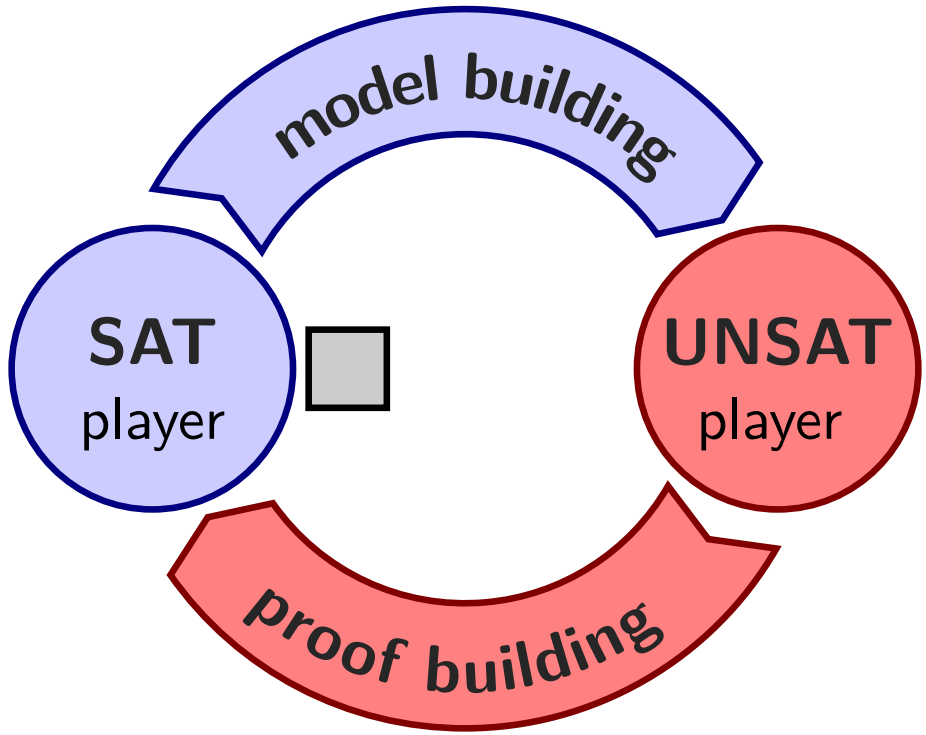
# Conflict-driven reasoning

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.



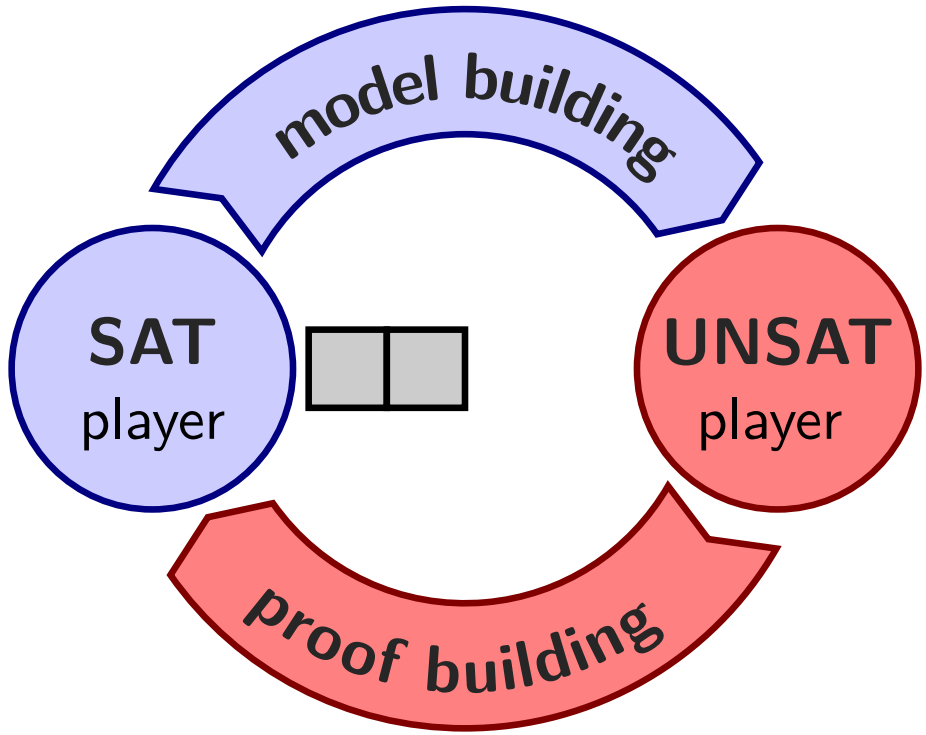
# Conflict-driven reasoning

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.



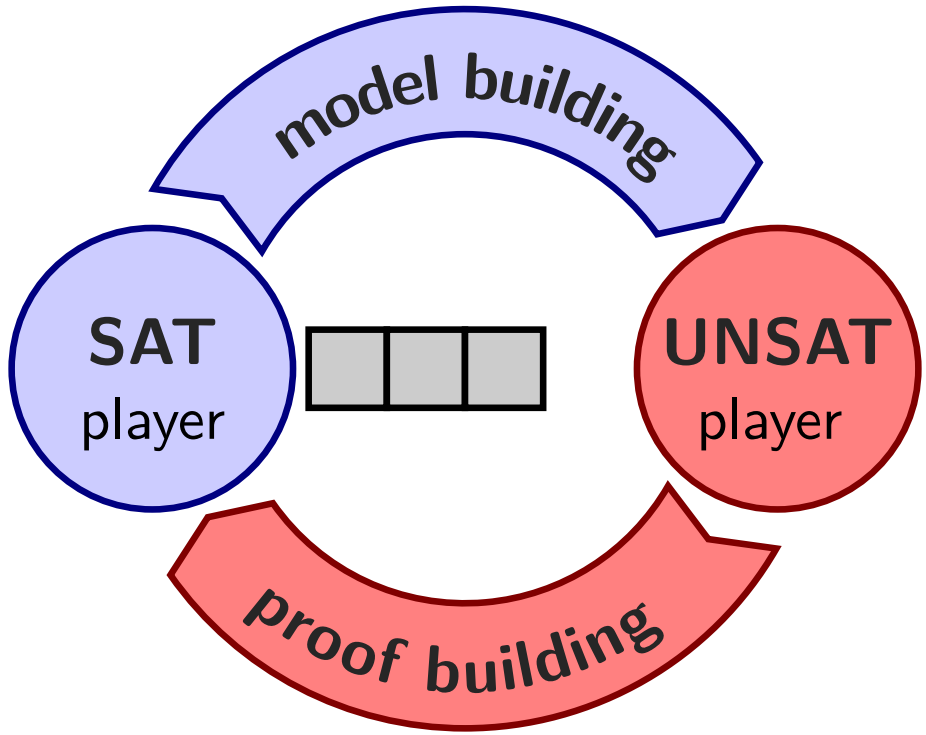
# Conflict-driven reasoning

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.



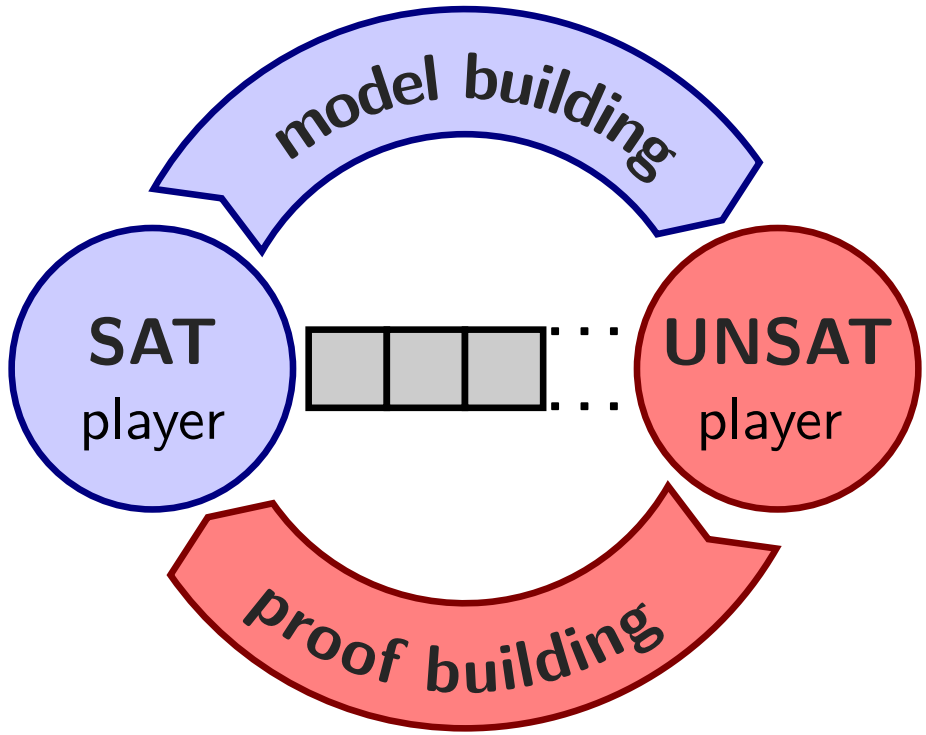
# Conflict-driven reasoning

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.



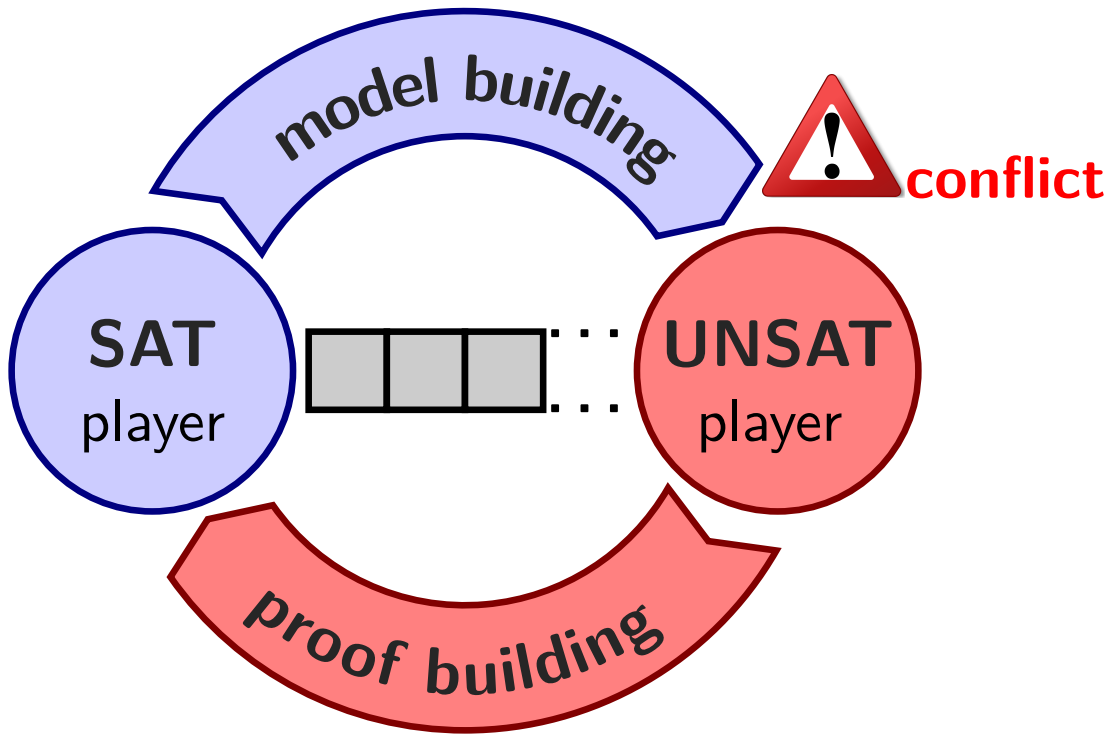
# Conflict-driven reasoning

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.



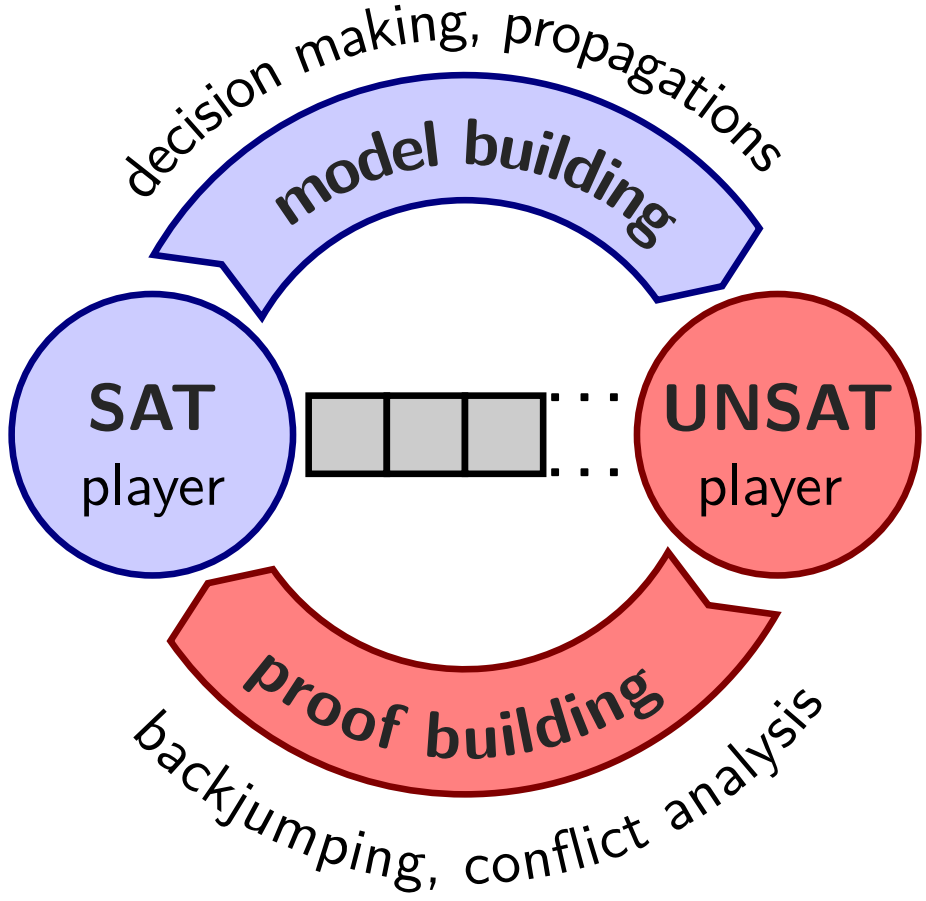
# Conflict-driven reasoning

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

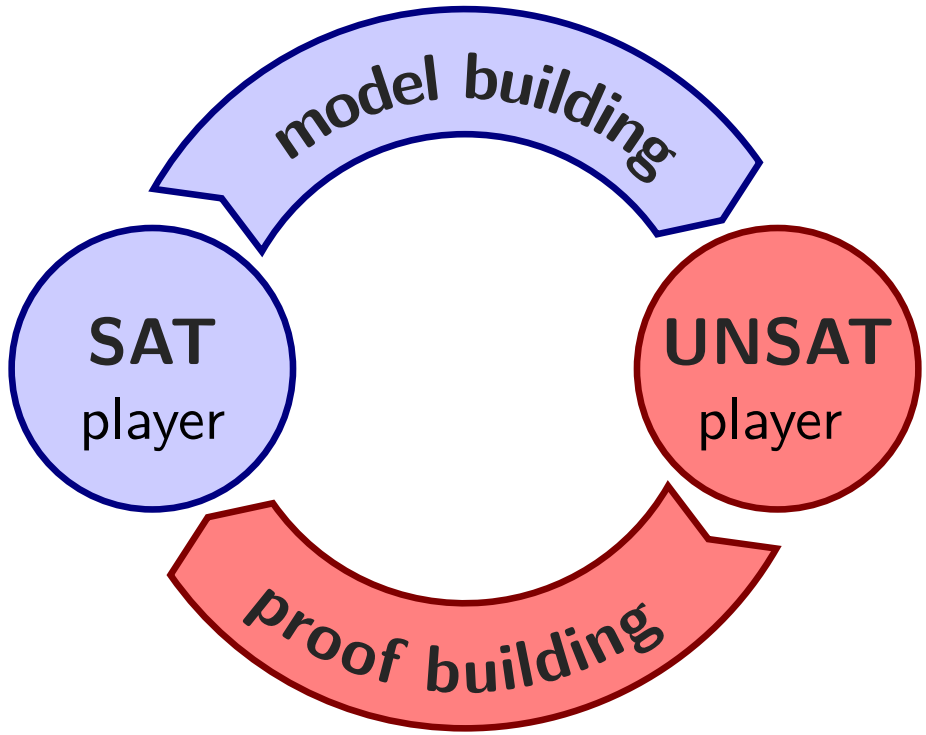
2-player game to determine whether a formula is satisfiable.
It involves a trail where a putative model is being specified.
It relies on a notion of conflict between the putative model and the formula it should satisfy.
Archetype of conflict-driven reasoning: **CDCL**
a conflict occurs when a clause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

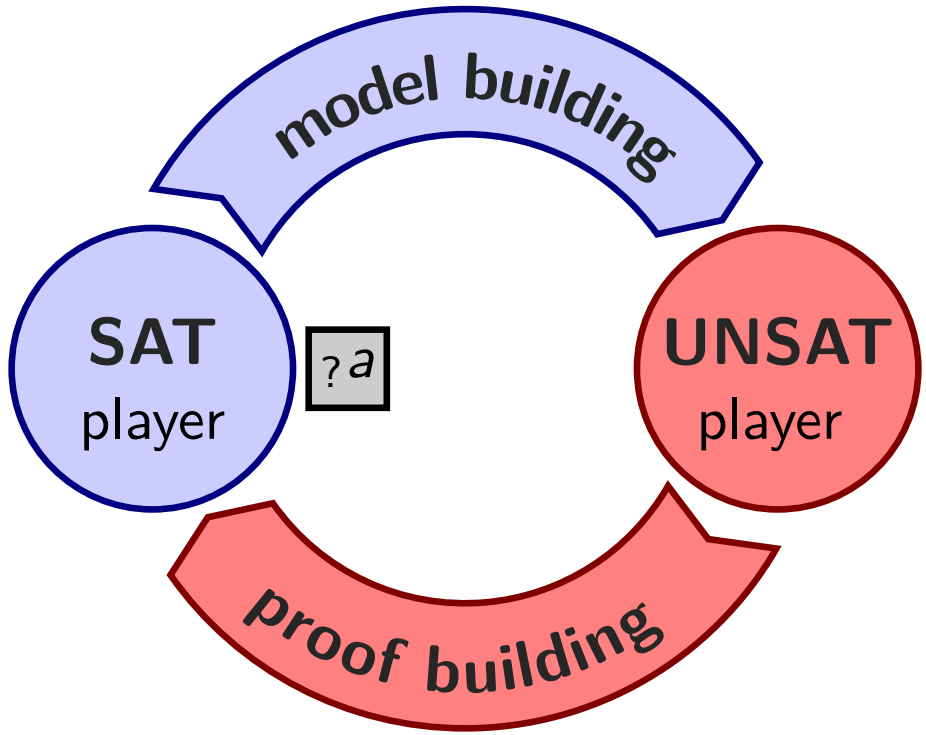
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



# 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

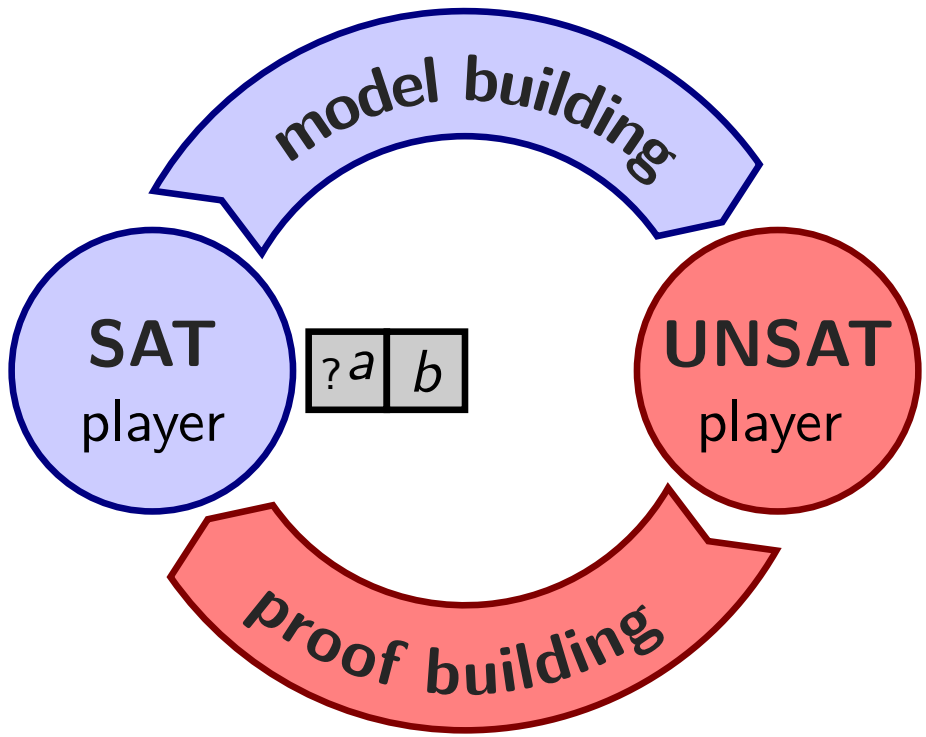
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



# 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

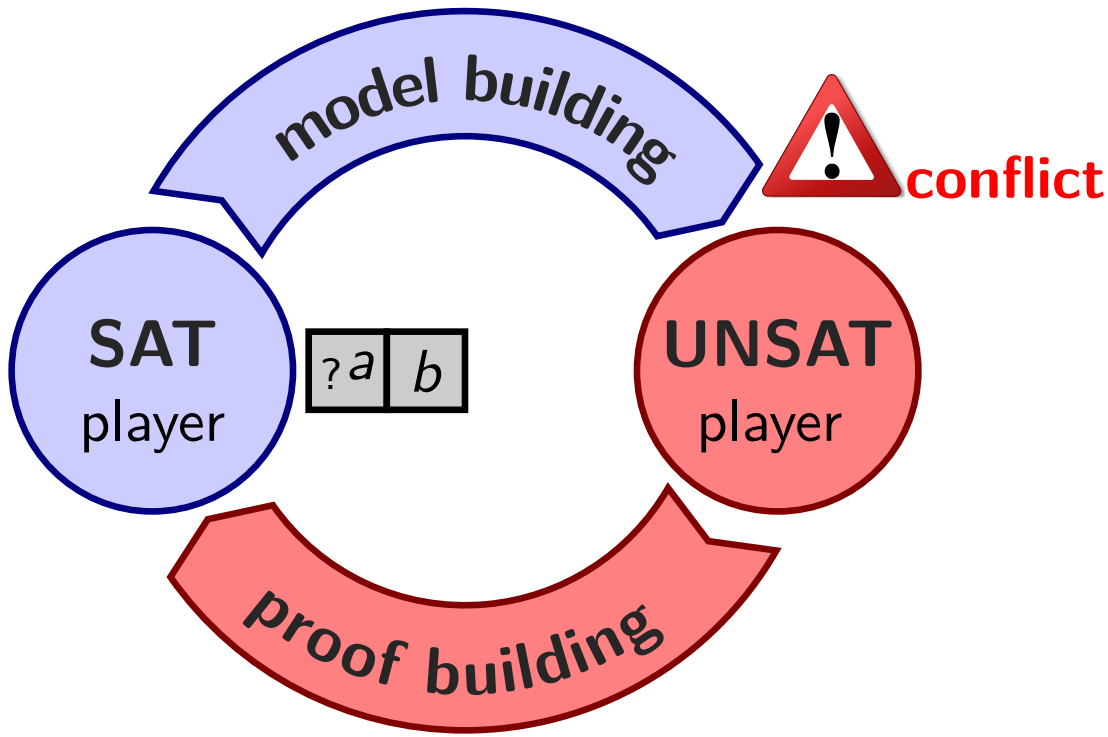
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



# 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

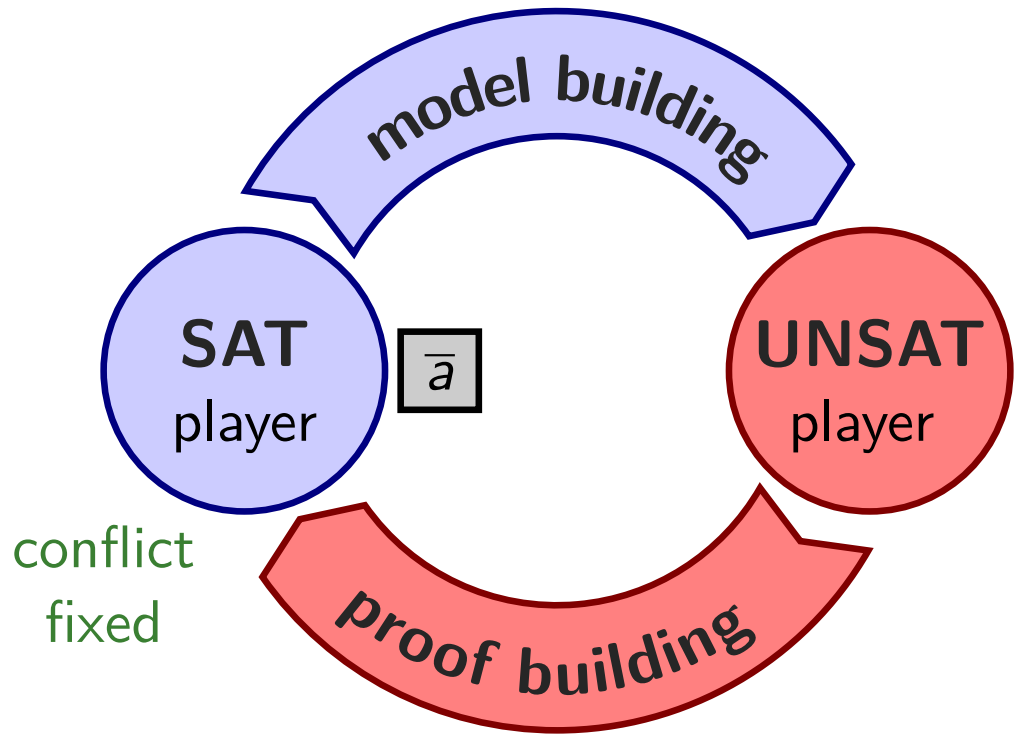
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$



# 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

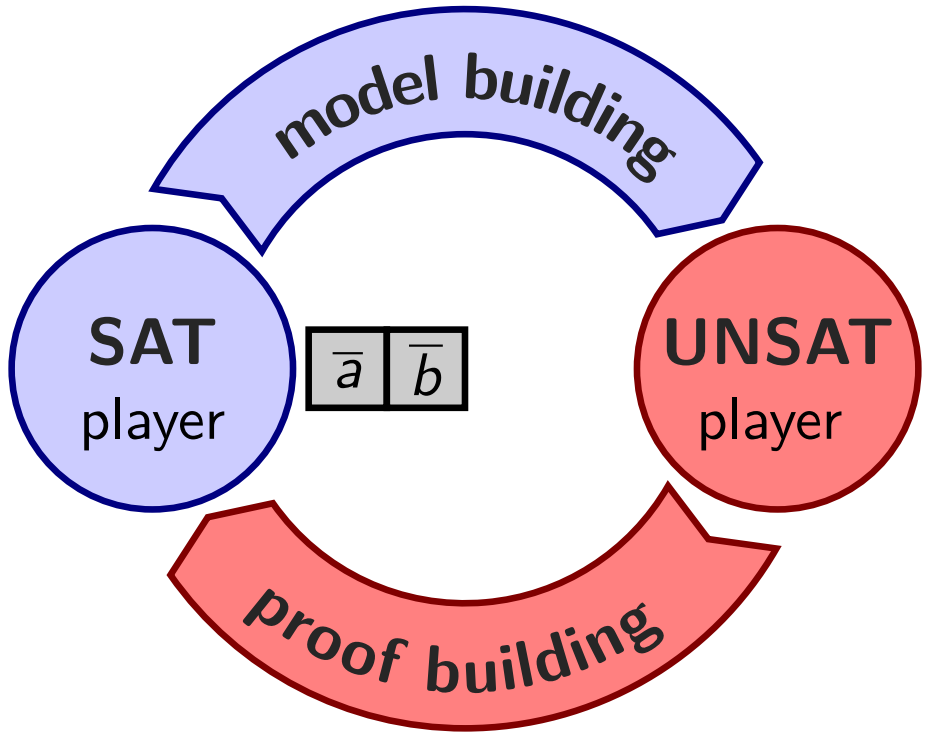
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$
$\overline{a}$



# 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

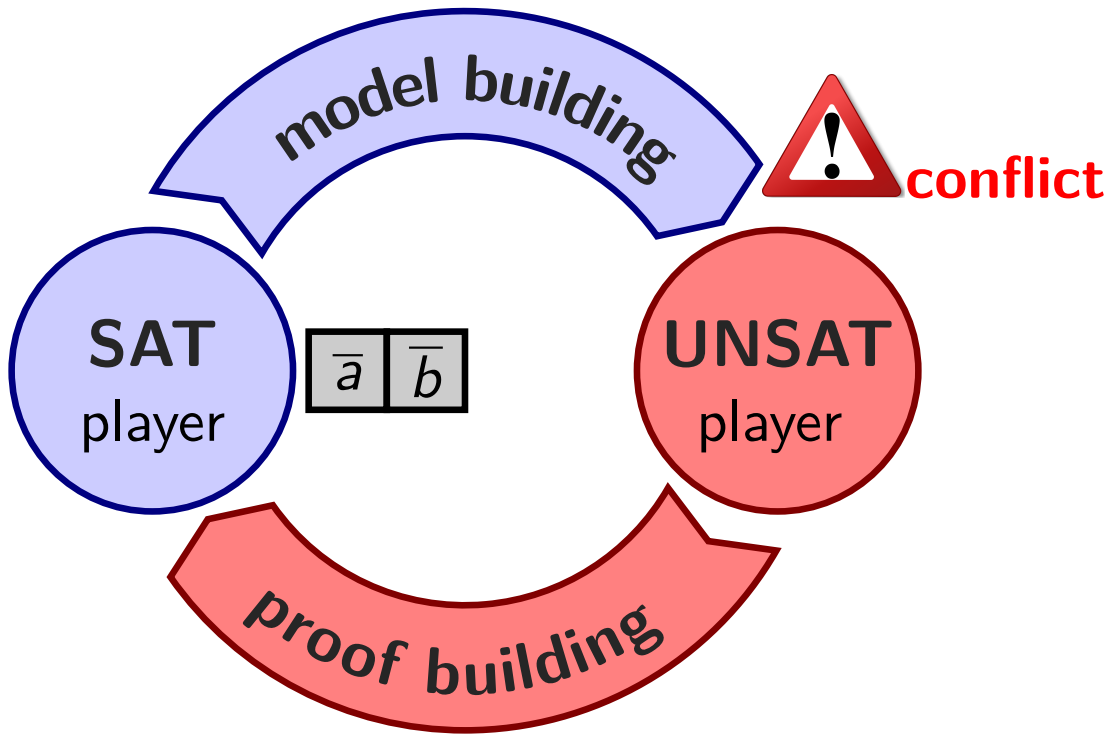
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$
$\overline{a}$



# 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

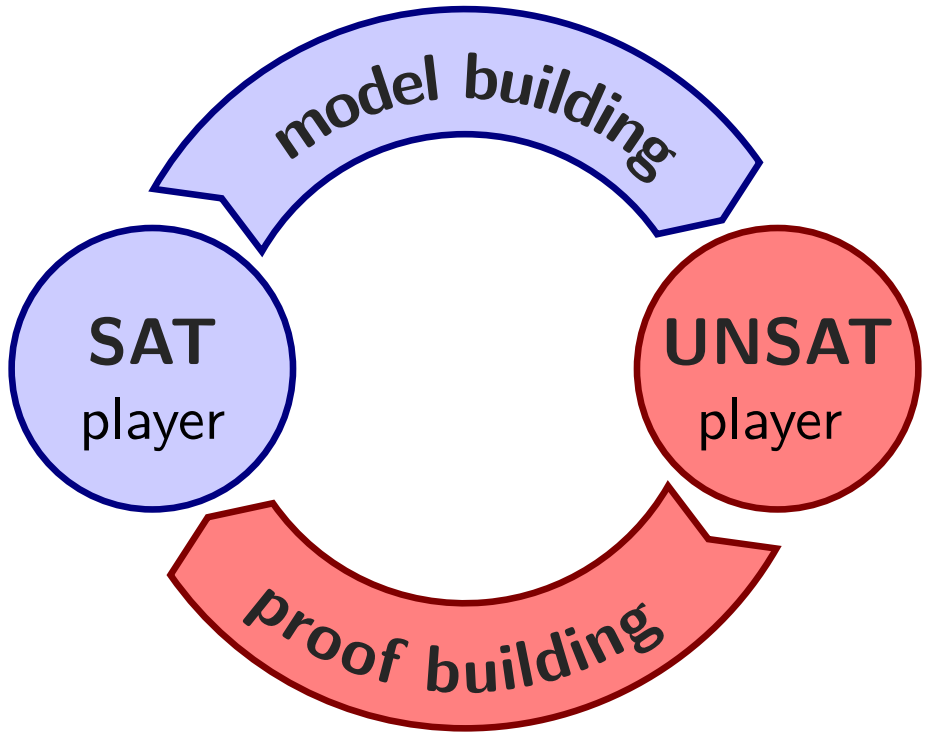
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$
$\overline{a}$



# 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

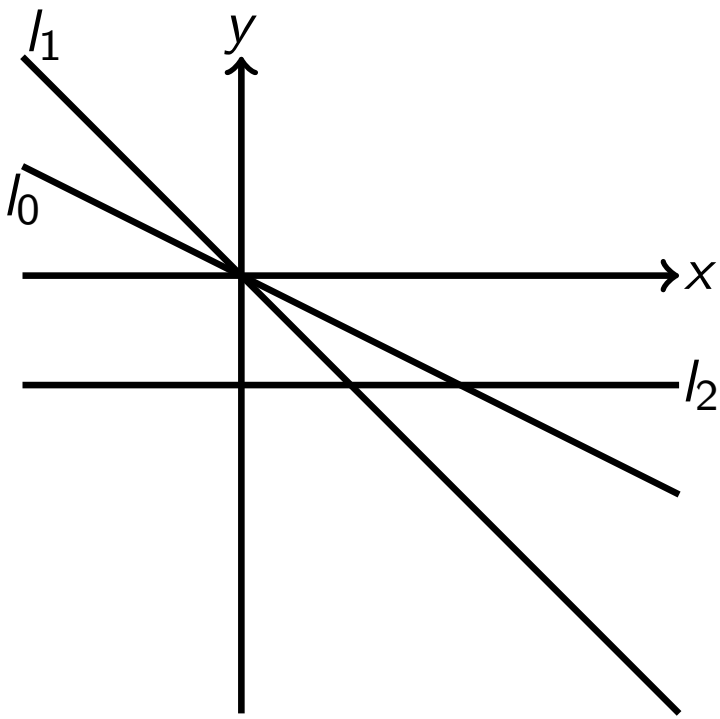
$a \Rightarrow b$
$b \Rightarrow \overline{a}$
$\overline{a} \Rightarrow \overline{b}$
$\overline{b} \Rightarrow a$
$\overline{a}$
$\bot$



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

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

unsatisfiable in Linear Rational Arithmetic (LRA).

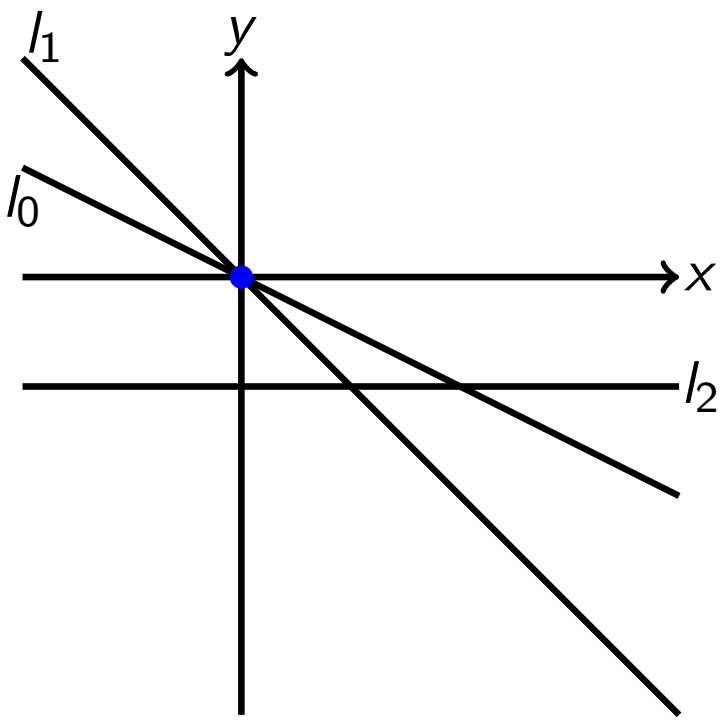


# Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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

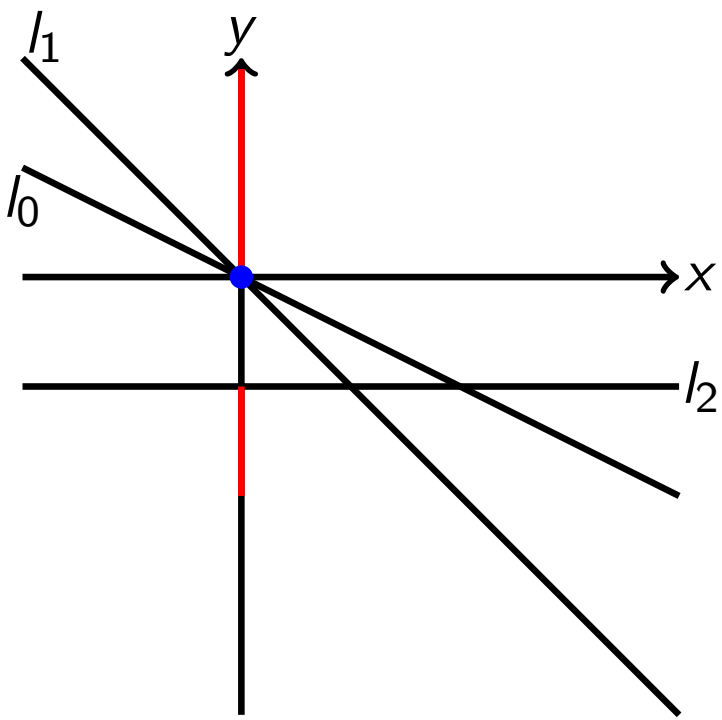


# Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).

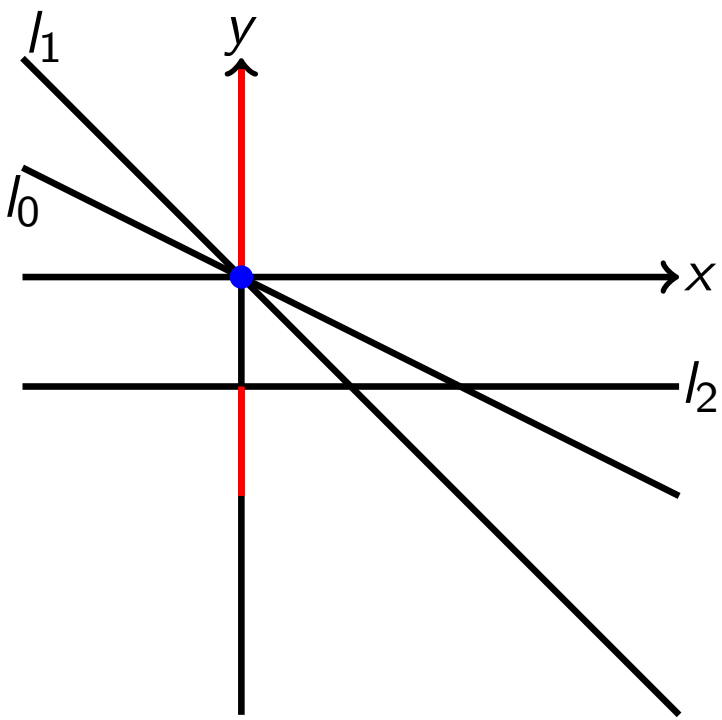
- Guess a value, e.g., $x \leftarrow 0$
  Then $l_0$ yields lower bound $y > 0$



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

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

unsatisfiable in Linear Rational Arithmetic (LRA).

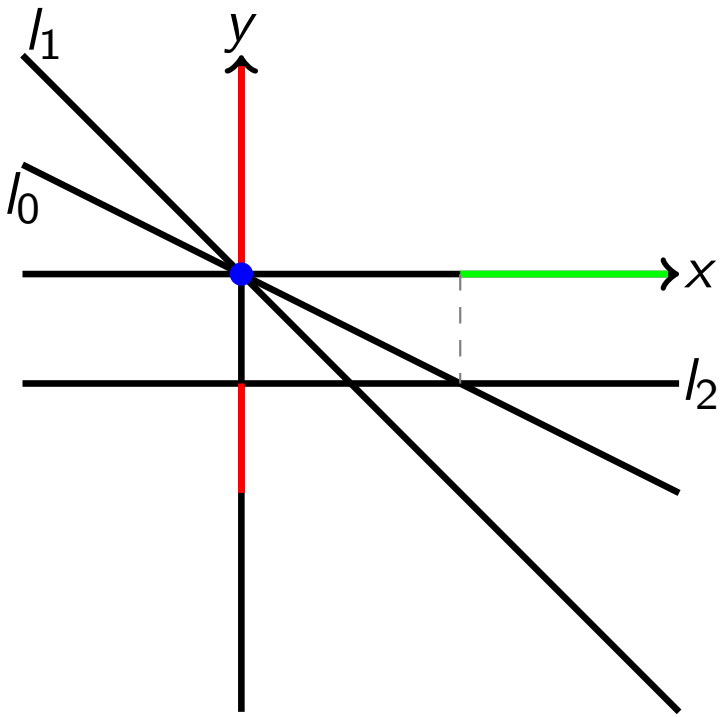



- Guess a value, e.g., $x \leftarrow 0$
  Then $l_0$ yields lower bound $y > 0$
  Together with $l_2$, range of possible values for $y$ is empty. What to do?
  Just undo $x \leftarrow 0$ & remember $x \neq 0$?

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

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

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

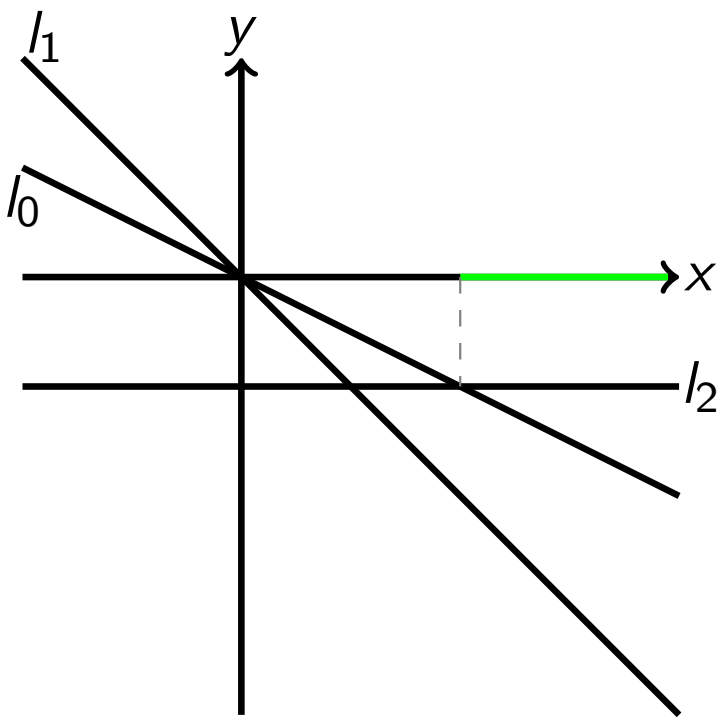
  $l_0 + 2l_2$, i.e., $\overbrace{(-x < -2)}^{l_3}$
  It rules out $x \leftarrow 0$, but also many values that would fail for the same reasons.

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

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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



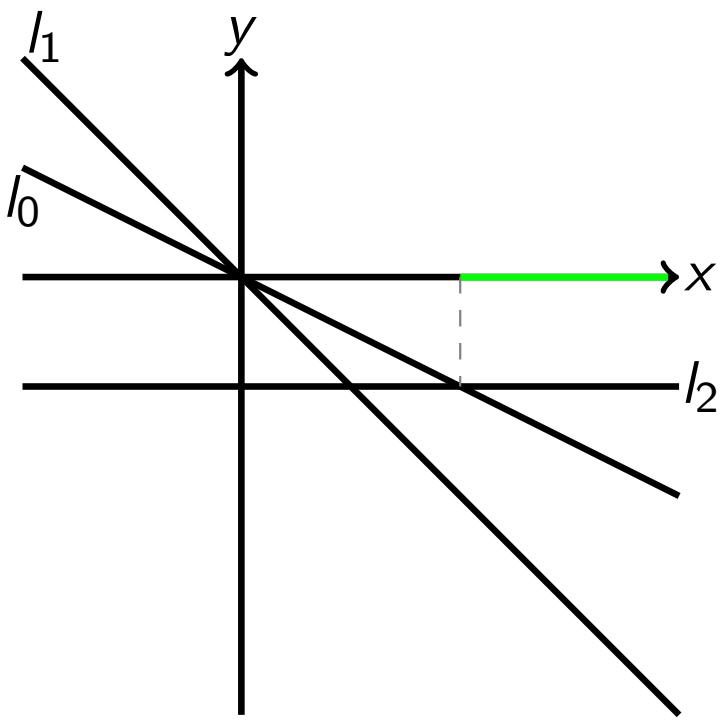

- Guess a value, e.g., $x \leftarrow 0$
  Then $l_0$ yields lower bound $y > 0$
  Together with $l_2$, range of possible values for $y$ is empty. What to do?
  Just undo $x \leftarrow 0$ & remember $x \neq 0$?
- **No!** Clash of bounds suggests a better conflict explanation, by inferring
  $l_0 + 2l_2$,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

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

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

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 $l_3$.

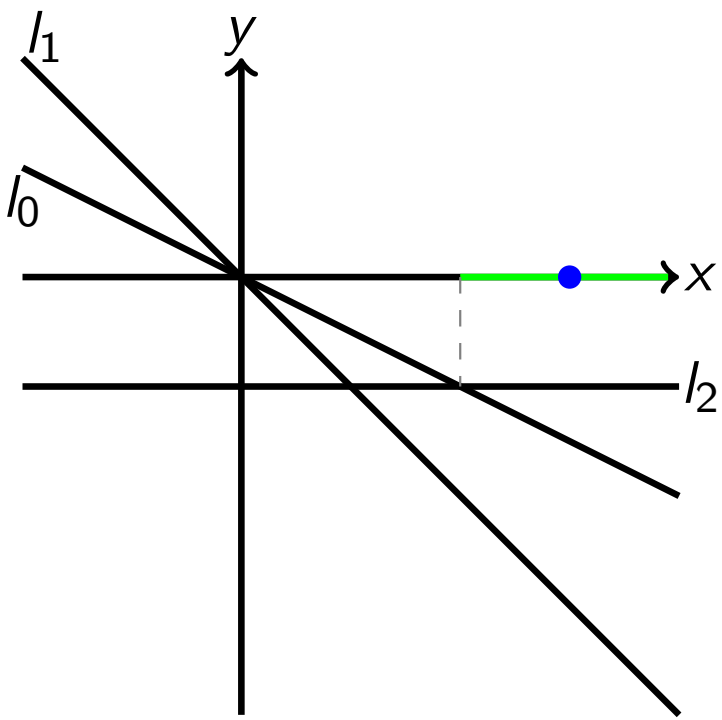
# Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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

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

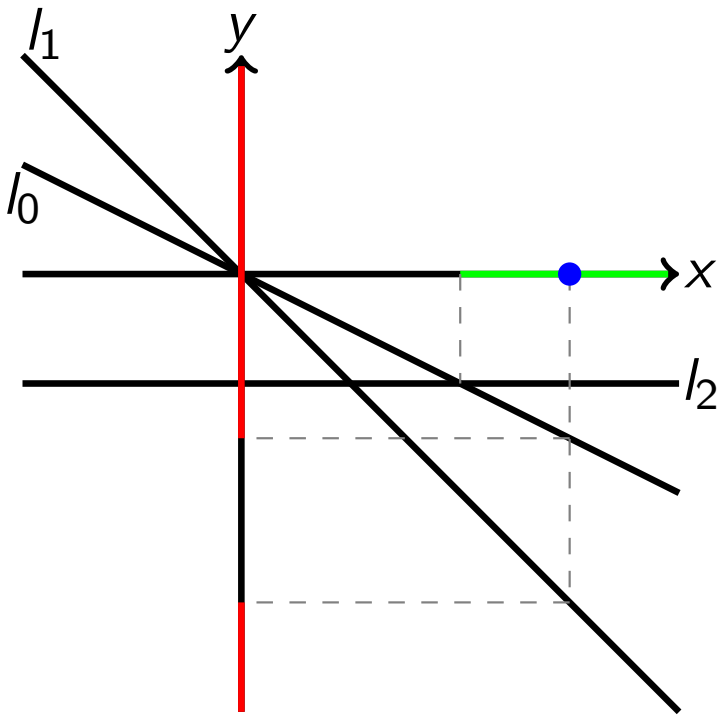


# Conflict-driven reasoning can be used for (other) theories

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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



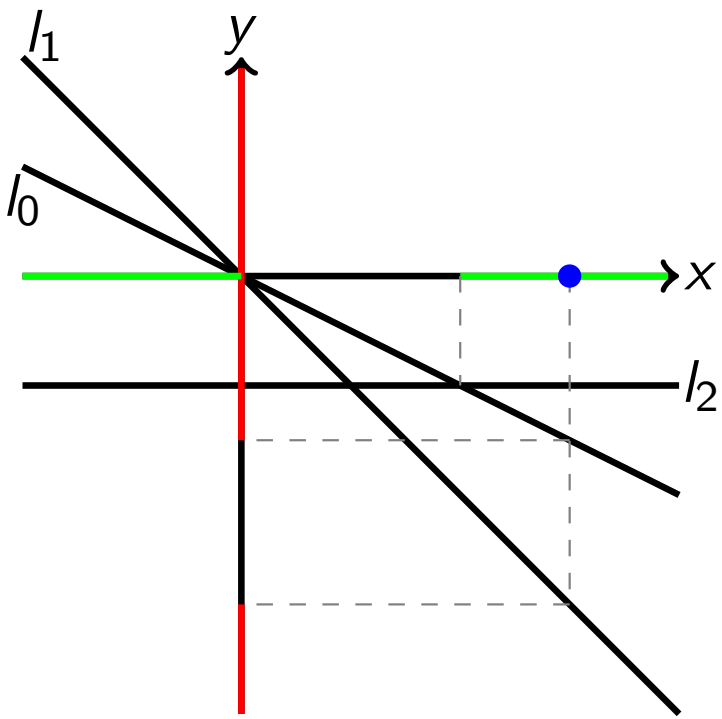

- Guess a value, e.g., $x \leftarrow 3$
  Then $l_0$ yields lower bound $y > -\frac{3}{2}$
  and $l_1$ yields upper bound $y < -3$

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

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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



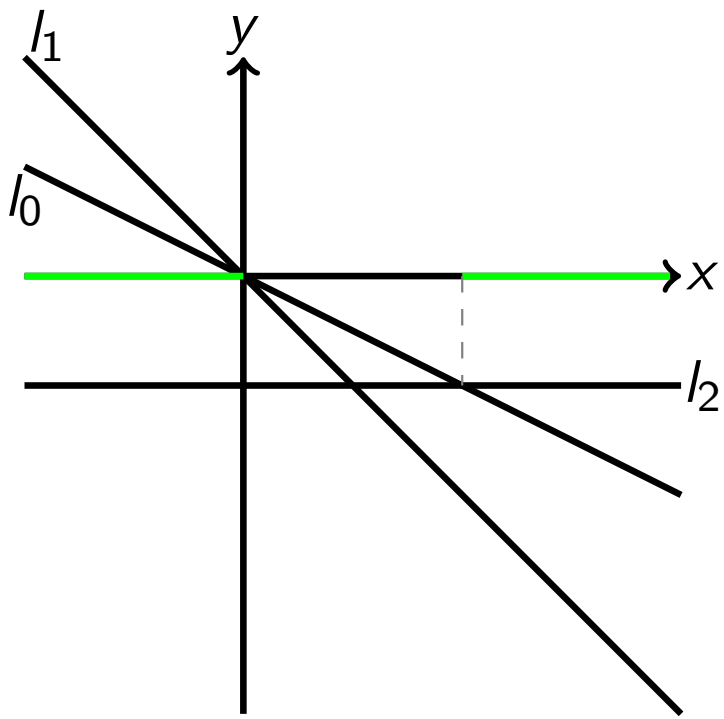

- Guess a value, e.g., $x \leftarrow 3$
  Then $l_0$ yields lower bound $y > -\frac{3}{2}$
  and $l_1$ yields upper bound $y < -3$
- Clash of bounds suggests inferring
  $l_0 + 2l_1$, i.e., $\overbrace{(x < 0)}^{l_4}$.

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

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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



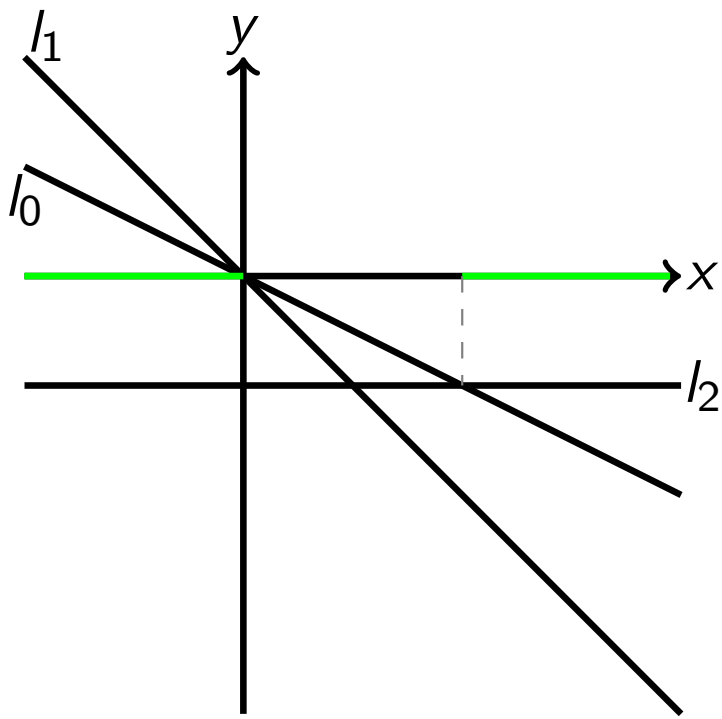

- Guess a value, e.g., $x \leftarrow 3$
  Then $l_0$ yields lower bound $y > -\frac{3}{2}$
  and $l_1$ yields upper bound $y < -3$
- Clash of bounds suggests inferring
  $l_0 + 2l_1$, i.e., $\overbrace{(x < 0)}^{l_4}$.
- Now undo the guess but keep $l_4$.

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

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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



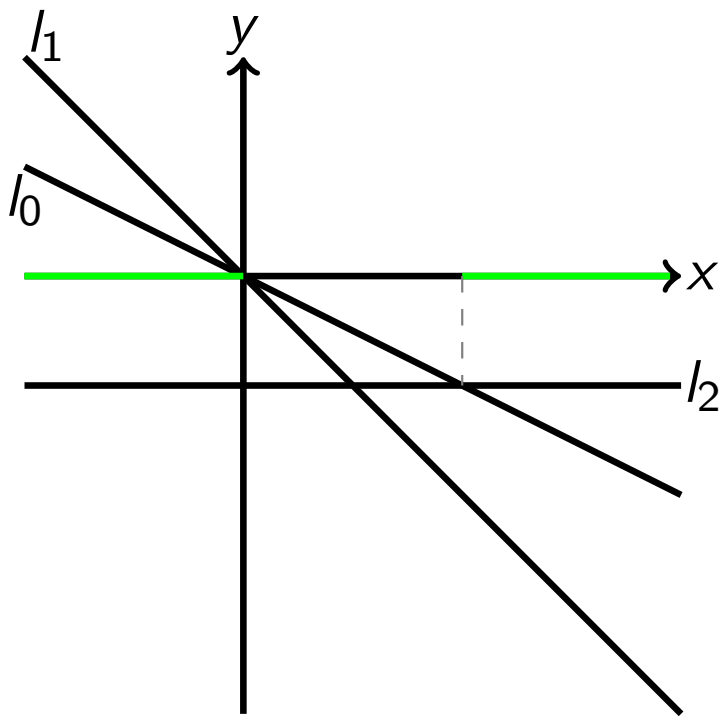

- Guess a value, e.g., $x \leftarrow 3$
  Then $l_0$ yields lower bound $y > -\frac{3}{2}$
  and $l_1$ yields upper bound $y < -3$
- Clash of bounds suggests inferring
  $l_0 + 2l_1$, i.e., $\overbrace{(x < 0)}^{l_4}$.
- Now undo the guess but keep $l_4$.
- Spot that $l_3$ and $l_4$ leave no value for $x$. Clash of bounds suggests inferring
  $l_3 + l_4$, i.e., $\overbrace{(0 < -2)}^{l_5}$.

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

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

unsatisfiable in Linear Rational Arithmetic (LRA).

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




- 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.

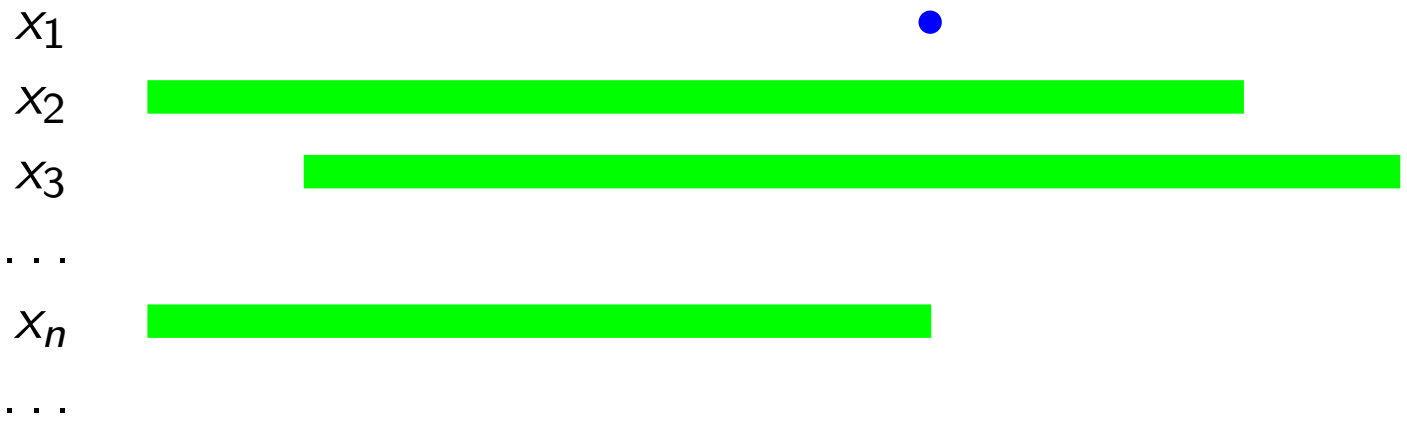
# More generally

Maintain for each variable of the problem a range of possible values


- Assign values to variables in sequence
- Detect when a constraint to satisfy, $l(\overrightarrow{x}, y)$ becomes unit in $y$: all other variables have been assigned values
- Compute how unit constraint $l(\overrightarrow{x}, y)$ restricts the range of possible values for $y$ (if it does).
- If the range of some variable becomes empty, raise a conflict, learn a lemma, and backtrack.
- Otherwise proceed to the next variable.
- If all variables ever get assigned: problem is SAT. If there is no more guess to backtrack over: problem is UNSAT.

# More generally

Maintain for each variable of the problem a range of possible values




- Assign values to variables in sequence
- Detect when a constraint to satisfy, $l(\overrightarrow{x}, y)$ becomes unit in $y$: all other variables have been assigned values
- Compute how unit constraint $l(\overrightarrow{x}, y)$ restricts the range of possible values for $y$ (if it does).
- If the range of some variable becomes empty, raise a conflict, learn a lemma, and backtrack.
- Otherwise proceed to the next variable.
- If all variables ever get assigned: problem is SAT. If there is no more guess to backtrack over: problem is UNSAT.

# More generally

Maintain for each variable of the problem a range of possible values


- Assign values to variables in sequence
- Detect when a constraint to satisfy, $l(\overrightarrow{x}, y)$ becomes unit in $y$: all other variables have been assigned values
- Compute how unit constraint $l(\overrightarrow{x}, y)$ restricts the range of possible values for $y$ (if it does).
- If the range of some variable becomes empty, raise a conflict, learn a lemma, and backtrack.
- Otherwise proceed to the next variable.
- If all variables ever get assigned: problem is SAT. If there is no more guess to backtrack over: problem is UNSAT.

# More generally

Maintain for each variable of the problem a range of possible values


- Assign values to variables in sequence
- Detect when a constraint to satisfy, $l(\overrightarrow{x}, y)$ becomes unit in $y$: all other variables have been assigned values
- Compute how unit constraint $l(\overrightarrow{x}, y)$ restricts the range of possible values for $y$ (if it does).
- If the range of some variable becomes empty, raise a conflict, learn a lemma, and backtrack.
- Otherwise proceed to the next variable.
- If all variables ever get assigned: problem is SAT. If there is no more guess to backtrack over: problem is UNSAT.

# The theory lemmas 1/2

These are the building block of proofs for UNSAT.

We look for them when raising a conflict.

# The theory lemmas 1/2

These are the building block of proofs for UNSAT.

We look for them when raising a conflict. In that case we have

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

# The theory lemmas 1/2

These are the building block of proofs for UNSAT.

We look for them when raising a conflict. In that case 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 $y$: $l_1(\overrightarrow{x}, y) \wedge \cdots \wedge l_m(\overrightarrow{x}, y)$

# The theory lemmas 1/2

These are the building block of proofs for UNSAT.

We look for them when raising a conflict. In that case 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 $\exists y(l_1(\overrightarrow{x}, y) \wedge \cdots \wedge l_m(\overrightarrow{x}, y))$

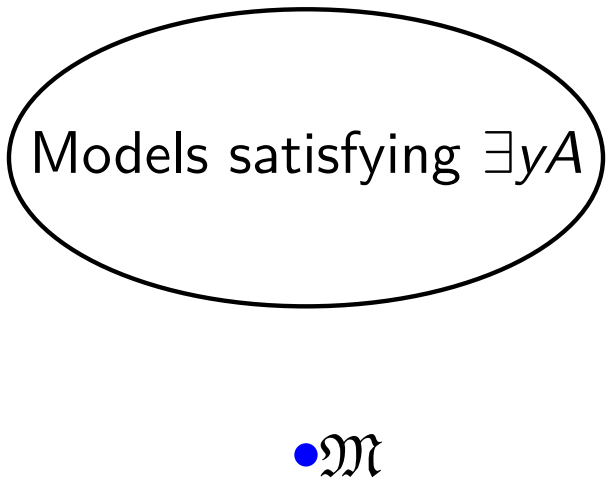
# The theory lemmas 1/2

These are the building block of proofs for UNSAT.

We look for them when raising a conflict. In that case 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 $\exists y(l_1(\overrightarrow{x}, y) \wedge \cdots \wedge l_m(\overrightarrow{x}, y))$



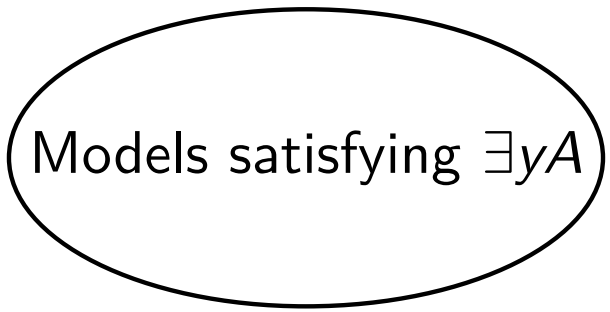
# The theory lemmas 1/2

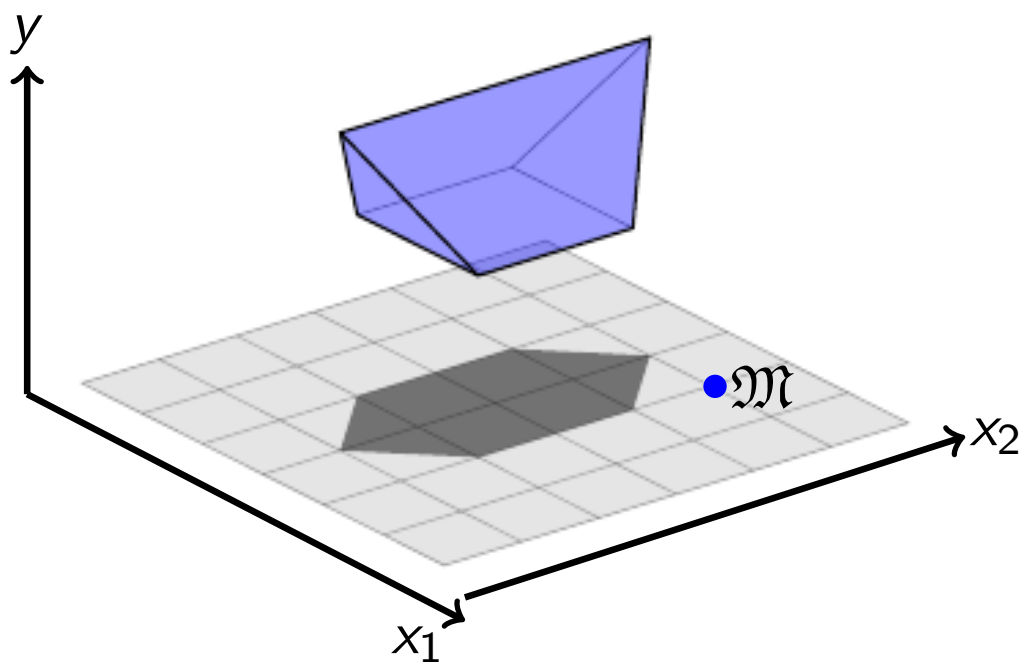
These are the building block of proofs for UNSAT.

We look for them when raising a conflict. In that case 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 $y$: $l_1(\vec{x}, y) \wedge \cdots \wedge l_m(\vec{x}, y)$
- 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 $\exists y(l_1(\vec{x}, y) \wedge \cdots \wedge l_m(\vec{x}, y))$



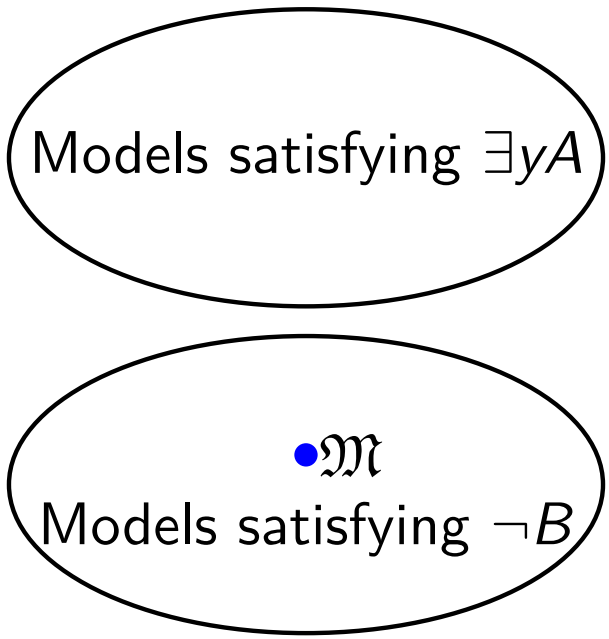
# The theory lemmas 1/2

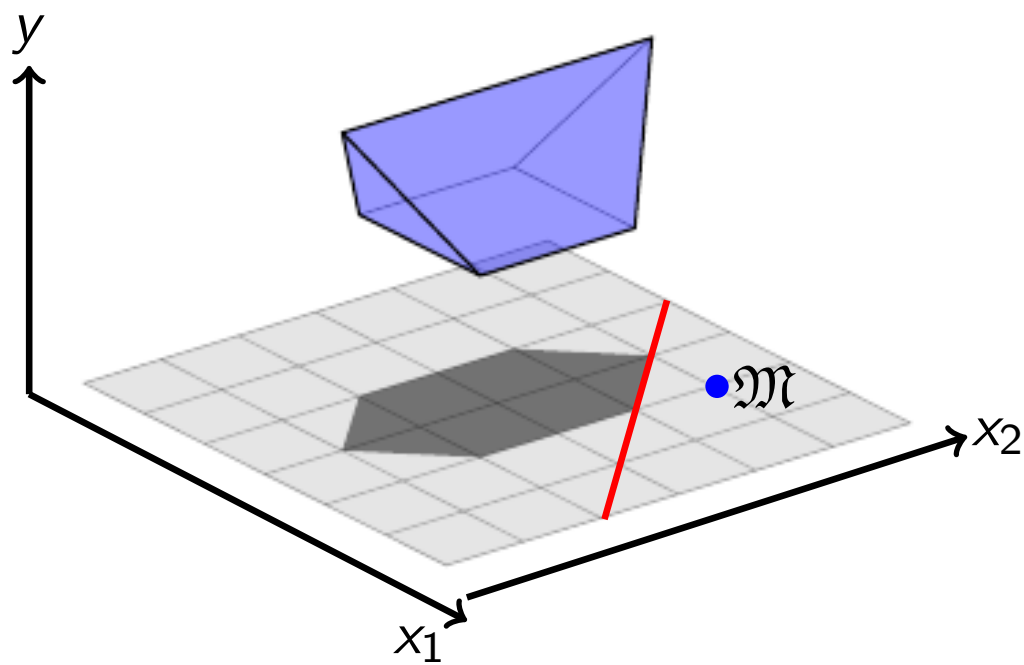
These are the building block of proofs for UNSAT.

We look for them when raising a conflict. In that case 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 $y$: $l_1(\vec{x}, y) \wedge \cdots \wedge l_m(\vec{x}, y)$
- 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 $\exists y(l_1(\vec{x}, y) \wedge \cdots \wedge l_m(\vec{x}, y))$

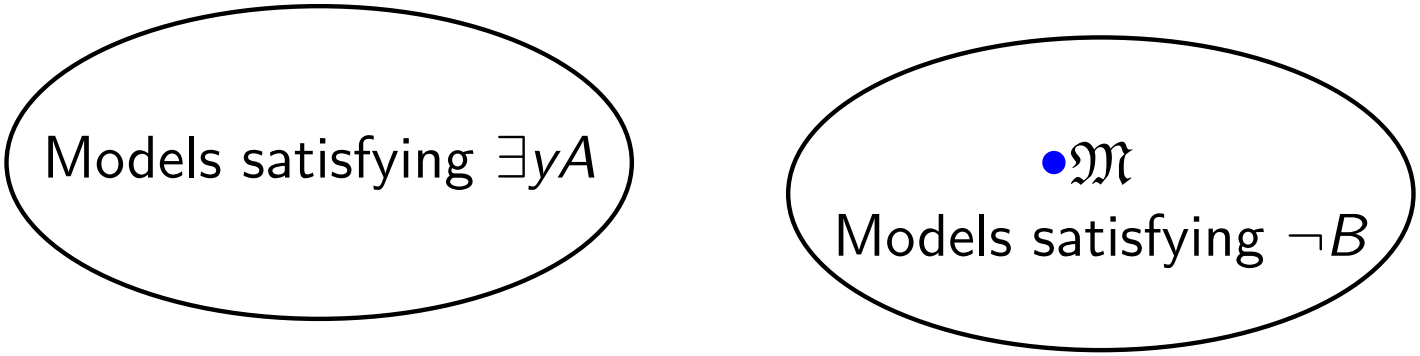



We seek to generalise $\mathfrak{M}$ into a class of models that do not satisfy $\exists y A$ "for the same reason" $\mathfrak{M}$ does not.

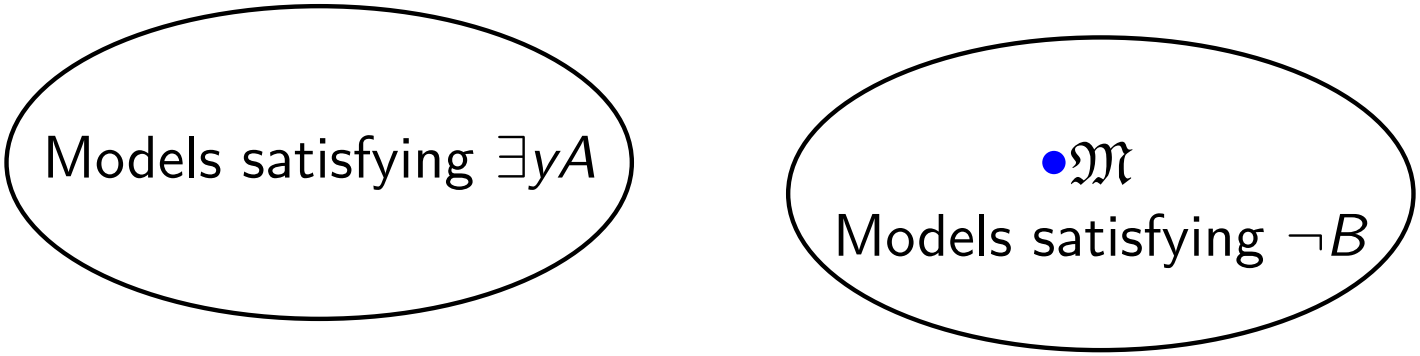
# The theory lemmas 2/2

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



# The theory lemmas 2/2

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

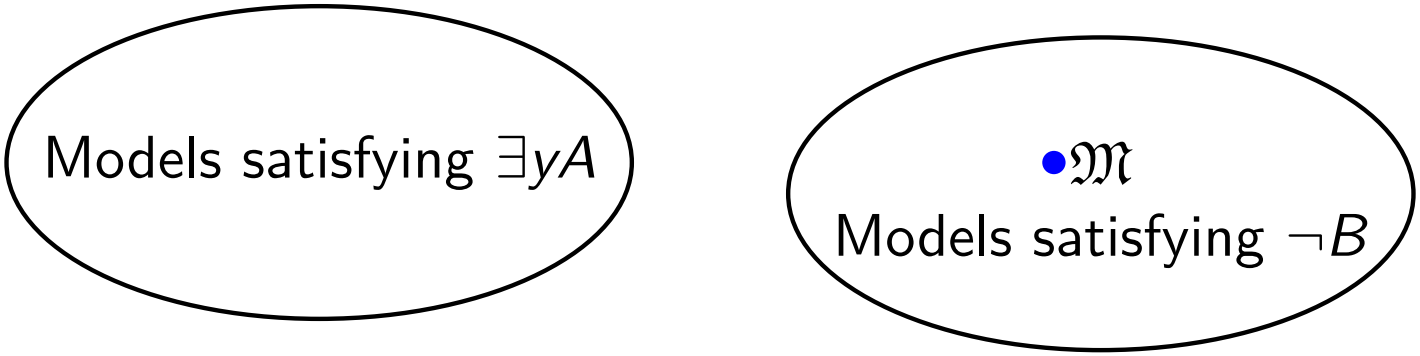



We characterise this class as those models not satisfying $B$, for some quantifier-free $B$ (with $\mathrm{fv}(B) \subseteq \{\overrightarrow{x}\}$) such that

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

# The theory lemmas 2/2

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



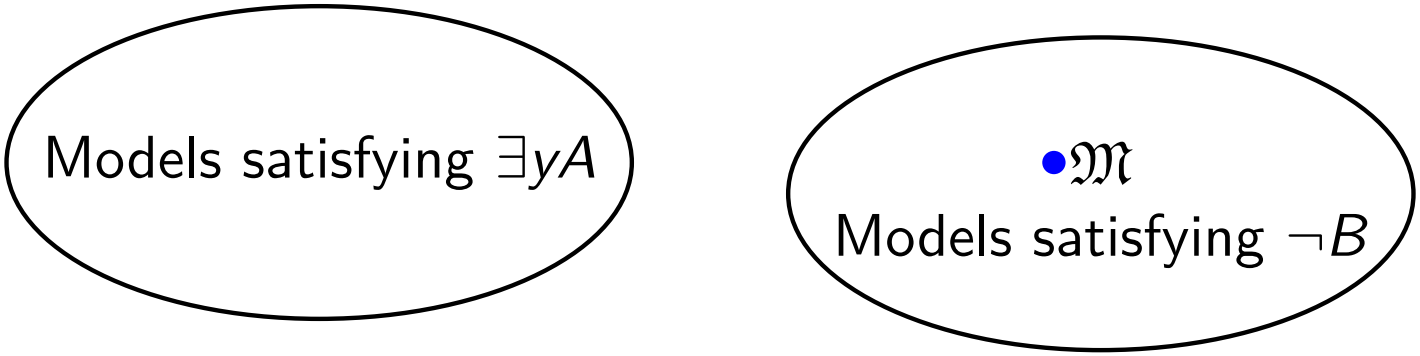

We characterise this class as those models not satisfying $B$, for some quantifier-free $B$ (with $\mathrm{fv}(B) \subseteq \{\overrightarrow{x}\}$) such that

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

This is called an interpolant.

# The theory lemmas 2/2

$\exists y A$ 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

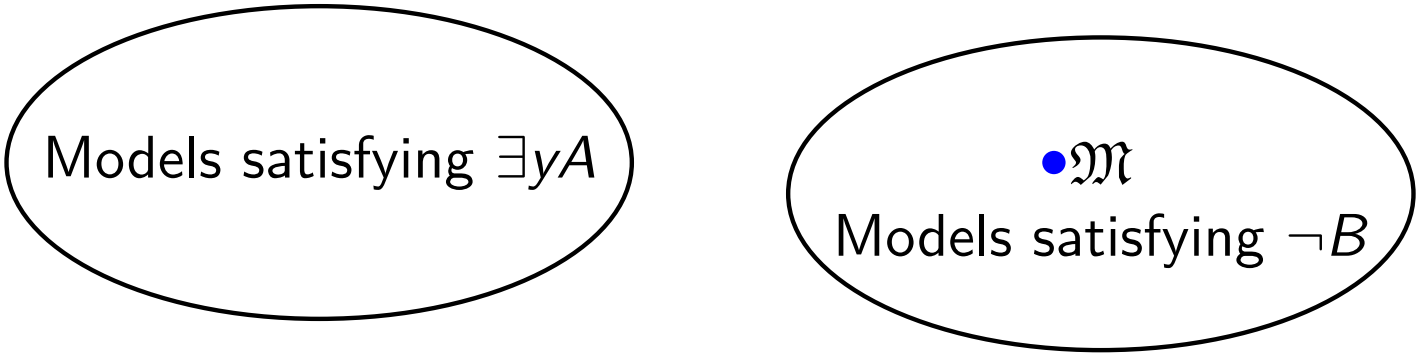
- $\mathcal{T} \models (\exists y A) \Rightarrow B$
- $\mathfrak{M} \not\models B$

This is called an interpolant.

This is almost a quantifier-elimination step. Main difference: We only require $\mathcal{T} \models (\exists y A) \Rightarrow B$, not $\mathcal{T} \models (\exists y A) \Leftrightarrow B$

# The theory lemmas 2/2

$\exists y A$ 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

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

This is called an interpolant.

This is almost a quantifier-elimination step. Main difference: We only require $\mathcal{T} \models (\exists y A) \Rightarrow B$, not $\mathcal{T} \models (\exists y A) \Leftrightarrow B$

For LRA, $B$ is provided by Fourier-Motzkin resolution from only two of the unit constraints

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

# Main difference with standard QE

We only require $\mathcal{T} \models (\exists y A) \Rightarrow B$, not $\mathcal{T} \models (\exists y A) \Leftrightarrow B$

# Main difference with standard QE

We only require $\mathcal{T} \models (\exists y A) \Rightarrow B$, not $\mathcal{T} \models (\exists y A) \Leftrightarrow B$
This is because we do not seek to remove $y$ from the problem, we simply add $B$ to the problem

# Main difference with standard QE

We only require $\mathcal{T} \models (\exists y A) \Rightarrow B$, not $\mathcal{T} \models (\exists y A) \Leftrightarrow B$
This is because we do not seek to remove $y$ from the problem, we simply add $B$ to the problem . . . in order to "defeat" model $\mathfrak{M}$ (so that our next model construction attempt avoids building models like $\mathfrak{M}$)

# Main difference with standard QE

We only require $\mathcal{T} \models (\exists y A) \Rightarrow B$, not $\mathcal{T} \models (\exists y A) \Leftrightarrow B$
This is because we do not seek to remove $y$ from the problem, we simply add $B$ to the problem . . . in order to "defeat" model $\mathfrak{M}$ (so that our next model construction attempt avoids building models like $\mathfrak{M}$)
The construction of the proofs (e.g. Boolean/FM resolution steps)

- is local to the current model $\mathfrak{M}$
- is guided by model construction attempts
- is lazy

# Main difference with standard QE

We only require $\mathcal{T} \models (\exists y A) \Rightarrow B$, not $\mathcal{T} \models (\exists y A) \Leftrightarrow B$
This is because we do not seek to remove $y$ from the problem, we simply add $B$ to the problem . . . in order to "defeat" model $\mathfrak{M}$ (so that our next model construction attempt avoids building models like $\mathfrak{M}$)
The construction of the proofs (e.g. Boolean/FM resolution steps)

- is local to the current model $\mathfrak{M}$
- is guided by model construction attempts
- is lazy

In fact

- $B$ may not take all of the constraints in $A$ into account (just enough to defeat $\mathfrak{M}$)
- $A$ itself does not contain all constraints involving $y$ in the full problem, only those that $\mathfrak{M}$ has made unit in $y$

An objective is to make the production of $B$ as cheap as possible

# In practice

In non-linear arithmetic, computing the projections for quantifier elimination is done by Cylindrical Algebraic Decomposition (CAD).
Rather expensive if full projections are computed
(i.e., completely eliminate a variable $y$ from the entire problem)

# In practice

In non-linear arithmetic, computing the projections for quantifier elimination is done by Cylindrical Algebraic Decomposition (CAD).
Rather expensive if full projections are computed
(i.e., completely eliminate a variable $y$ from the entire problem)

Using partial / model-guided projections (i.e., lazy quantifier elimination) turns out to be a good approach in practice.
Implemented in the Yices SMT-solver by Jovanović
(probably the best solver for non-linear arithmetic)

# Bit vectors

Similar situation with the theory of bitvectors: $(y||x) <_u (y<<3)$

# Bit vectors

Similar situation with the theory of bitvectors: $(y||x) <_u (y<<3)$

Just like Boolean logic, bitvectors talk about a finite number of models. As in Boolean logic, quantifiers can be eliminated.

# Bit vectors

Similar situation with the theory of bitvectors: $(y||x) <_u (y<<3)$

Just like Boolean logic, bitvectors talk about a finite number of models. As in Boolean logic, quantifiers can be eliminated.

Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

# Bit vectors

Similar situation with the theory of bitvectors: $(y||x) <_u (y<<3)$

Just like Boolean logic, bitvectors talk about a finite number of models. As in Boolean logic, quantifiers can be eliminated.

Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

Describing complex operations (typically $\times$) as a CNF can blow up the size of the formula.

# Bit vectors

Similar situation with the theory of bitvectors: $(y||x) <_u (y<<3)$

Just like Boolean logic, bitvectors talk about a finite number of models. As in Boolean logic, quantifiers can be eliminated.

Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

Describing complex operations (typically $\times$) as a CNF can blow up the size of the formula.

In [GLJ17] we described a conflict-driven approach to bitvectors

- A management of ranges using Binary Decision Diagrams

  $x_1$

  $x_2$

  $x_3$

  $\ldots$

  $x_n$

  $\ldots$

- Some proposals for producing theory lemmas.

# Bit vectors

Similar situation with the theory of bitvectors: $(y||x) <_u (y<<3)$

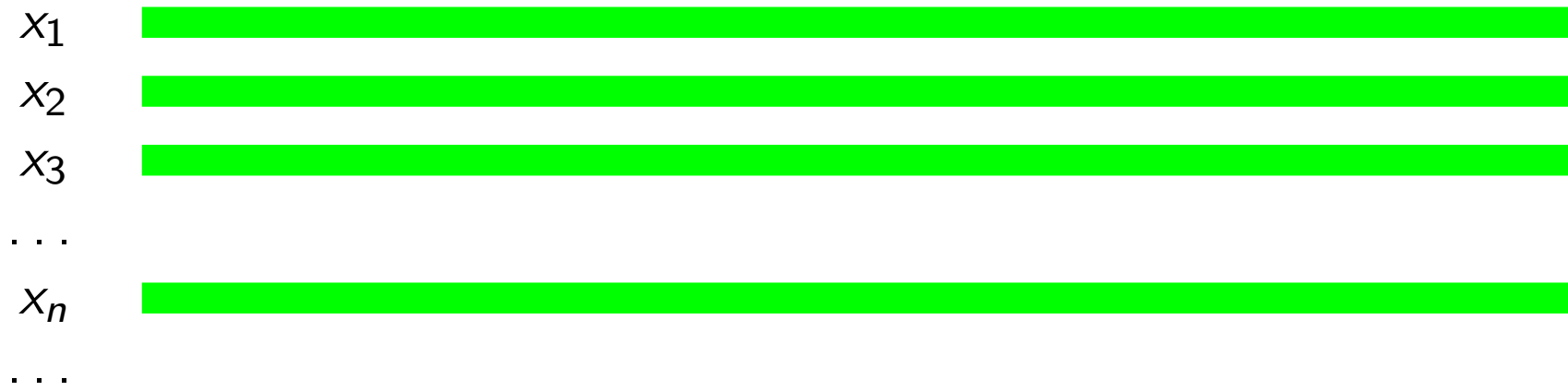
Just like Boolean logic, bitvectors talk about a finite number of models. As in Boolean logic, quantifiers can be eliminated.

Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

Describing complex operations (typically $\times$) as a CNF can blow up the size of the formula.

In [GLJ17] we described a conflict-driven approach to bitvectors

- A management of ranges using Binary Decision Diagrams




- Some proposals for producing theory lemmas.

# Bit vectors

Similar situation with the theory of bitvectors: $(y||x) <_u (y<<3)$

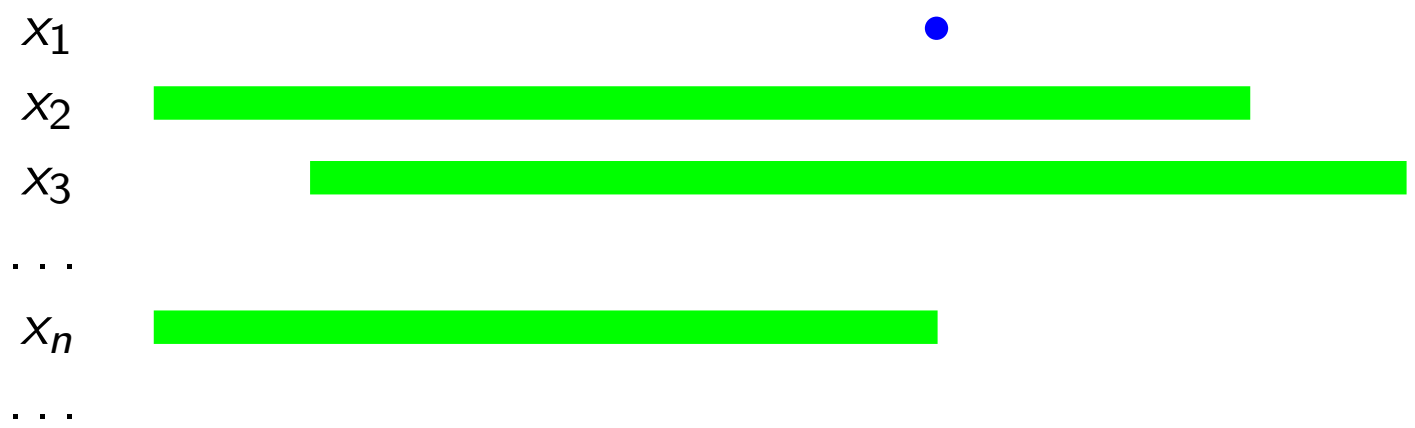
Just like Boolean logic, bitvectors talk about a finite number of models. As in Boolean logic, quantifiers can be eliminated.

Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

Describing complex operations (typically $\times$) as a CNF can blow up the size of the formula.

In [GLJ17] we described a conflict-driven approach to bitvectors

- A management of ranges using Binary Decision Diagrams




- Some proposals for producing theory lemmas.

# Bit vectors

Similar situation with the theory of bitvectors: $(y||x) <_u (y<<3)$

Just like Boolean logic, bitvectors talk about a finite number of models. As in Boolean logic, quantifiers can be eliminated.

Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

Describing complex operations (typically $\times$) as a CNF can blow up the size of the formula.

In [GLJ17] we described a conflict-driven approach to bitvectors

- A management of ranges using Binary Decision Diagrams


- Some proposals for producing theory lemmas.

# Bit vectors

Similar situation with the theory of bitvectors: $(y||x) <_u (y{<<}3)$

Just like Boolean logic, bitvectors talk about a finite number of models. As in Boolean logic, quantifiers can be eliminated.

Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

Describing complex operations (typically $\times$) as a CNF can blow up the size of the formula.

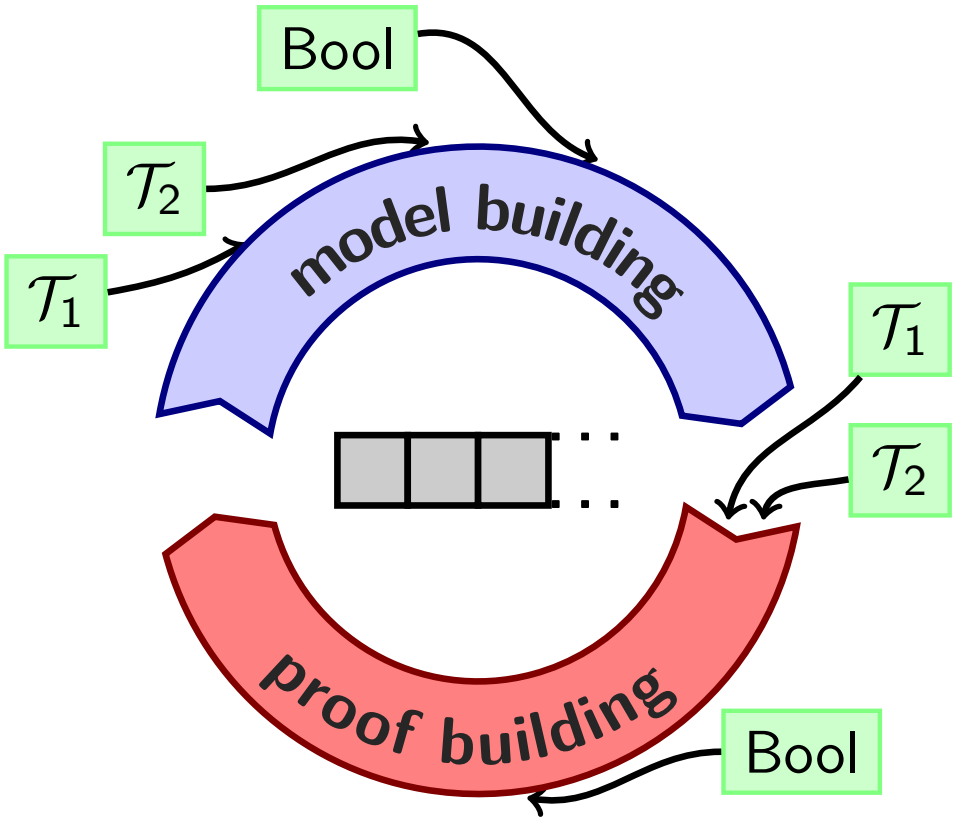
In [GLJ17] we described a conflict-driven approach to bitvectors

- A management of ranges using Binary Decision Diagrams


- Some proposals for producing theory lemmas.

# In CDSAT

In CDSAT [BGLS17, BGLS17] we propose a comprehensive framework for conflict-driven reasoning modulo several (but disjoint) theories.
The theory combination is organised directly in the main conflict-driven loop:



# What about arbitrarily quantified problems?

After all, being able to eliminate a quantifier $\mathcal{T} \models (\exists y A) \Leftrightarrow B$ allows the treatment of an arbitrarily quantified formula.

# What about arbitrarily quantified problems?

After all, being able to eliminate a quantifier $\mathcal{T} \models (\exists y A) \Leftrightarrow B$ allows the treatment of an arbitrarily quantified formula.

If we are only interested in determining satisfiability or provability, can we avoid the expensive full computation of equivalent quantifier-free formulae?
Can we have a lazier approach as in the existential case?

# What about arbitrarily quantified problems?

After all, being able to eliminate a quantifier $\mathcal{T} \models (\exists y A) \Leftrightarrow B$ allows the treatment of an arbitrarily quantified formula.

If we are only interested in determining satisfiability or provability, can we avoid the expensive full computation of equivalent quantifier-free formulae?
Can we have a lazier approach as in the existential case?

The basic quantifier elimination algorithm, which treats $\forall$ as $\neg\exists\neg$, heavily relies on the elimination of a quantified variable being an equivalence.
We can anticipate that the weaker condition $\mathcal{T} \models (\exists y A) \Rightarrow B$ required in MCSAT may be insufficient when treating alternations of $\forall$ and $\exists$

# The approach by Bjørner and Janota 1/3

In [BJ15], Bjørner and Janota propose an algorithm for "playing with quantified satisfaction", inspired by QBF.

# The approach by Bjørner and Janota 1/3

In [BJ15], Bjørner and Janota propose an algorithm for "playing with quantified satisfaction", inspired by QBF.

The SAT and UNSAT players from MCSAT exchange roles at every quantifier alternation:

- The universal player (UP) chooses values for universally quantified variables, trying to make the formula false;
- The existential player (EP) chooses values for existentially quantified variables, trying to make the formula true.

# The approach by Bjørner and Janota 1/3

In [BJ15], Bjørner and Janota propose an algorithm for "playing with quantified satisfaction", inspired by QBF.

The SAT and UNSAT players from MCSAT exchange roles at every quantifier alternation:

- The universal player (UP) chooses values for universally quantified variables, trying to make the formula false;
- The existential player (EP) chooses values for existentially quantified variables, trying to make the formula true.

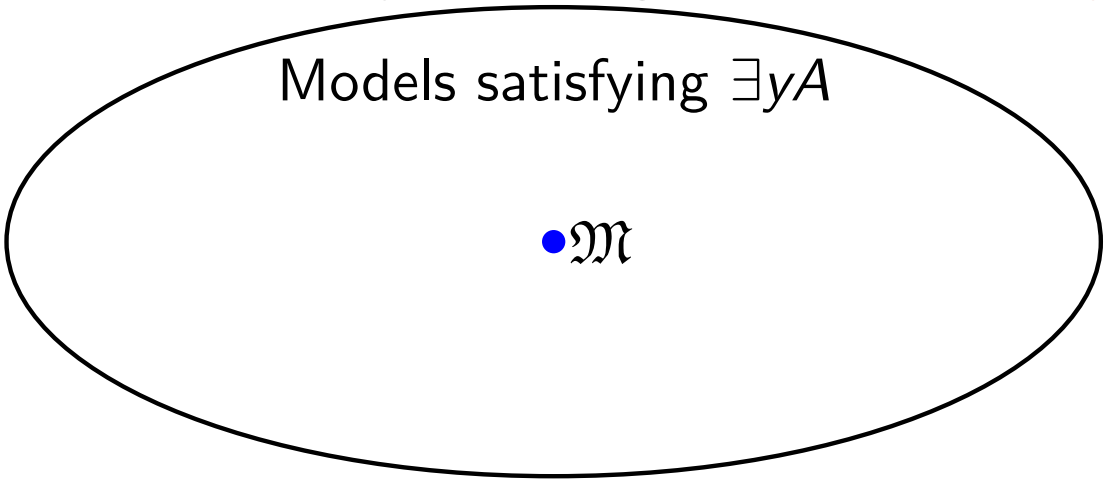
In order to determine whether $\forall x \exists y A$ is satisfiable:

- UP first chooses a value for $x$;
- If EP cannot find a value for $y$ to make $A$ true, UP has won.
- If on the contrary EP can find a value, UP needs to understand what was wrong about its choice for $x$.

Relies on theory $\mathcal{T}$ having model-based projection

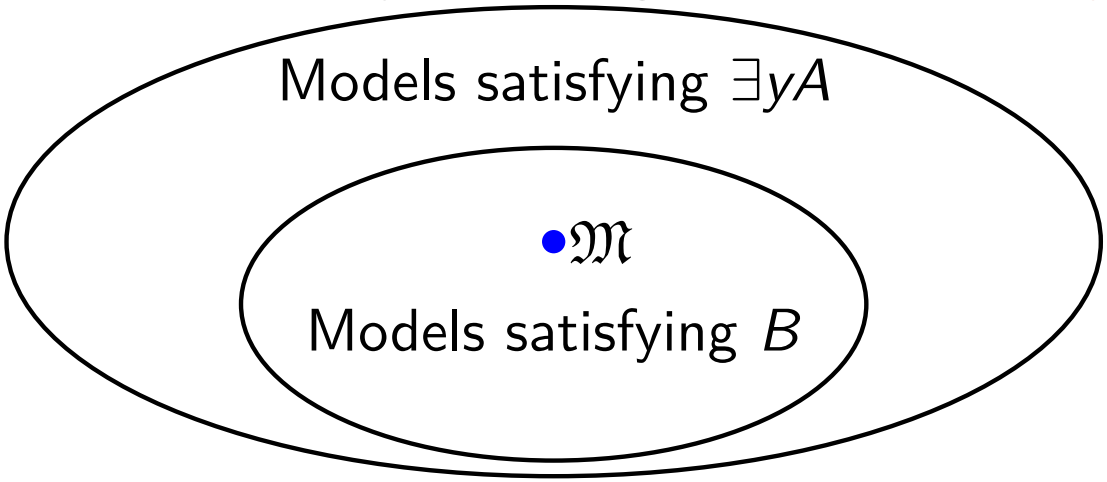
# The approach by Bjørner and Janota 2/3

Relies on theory $\mathcal{T}$ having model-based projection



# The approach by Bjørner and Janota 2/3

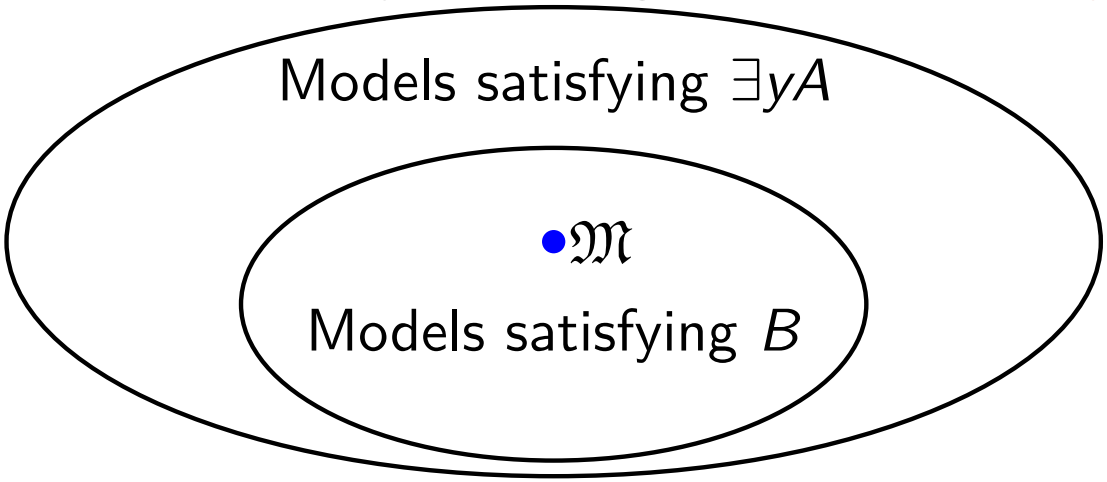
Relies on theory $\mathcal{T}$ having model-based projection



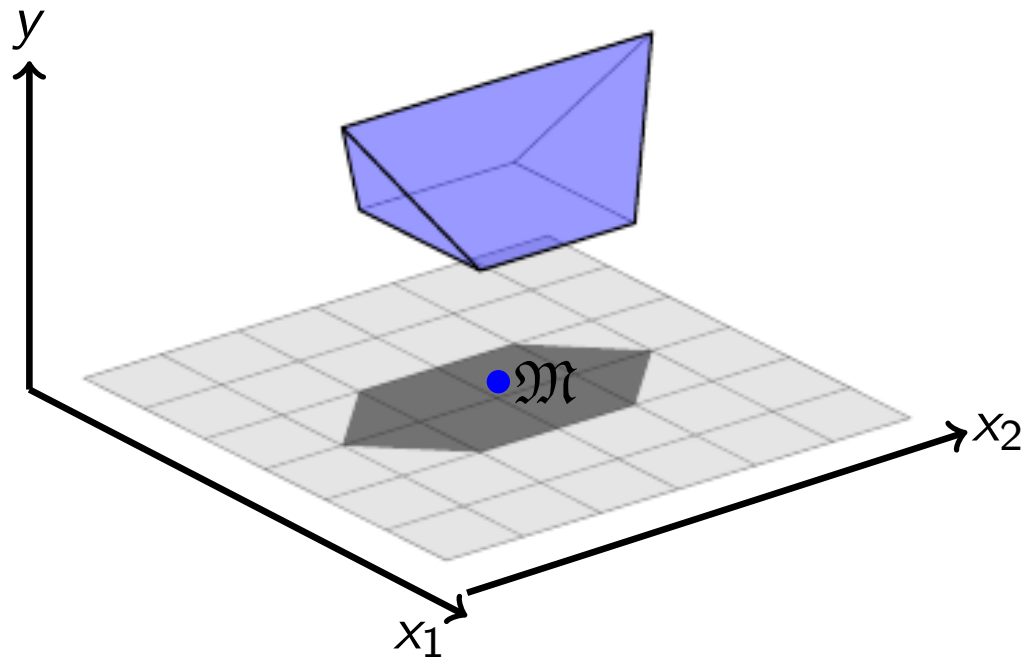

From $\mathfrak{M} \models \exists yA$,
produce quantifier-free
$B = mbp(\mathfrak{M}, \exists yA)$
such that
$\mathcal{T} \models B \Rightarrow \exists yA$.

# The approach by Bjørner and Janota 2/3

Relies on theory $\mathcal{T}$ having model-based projection

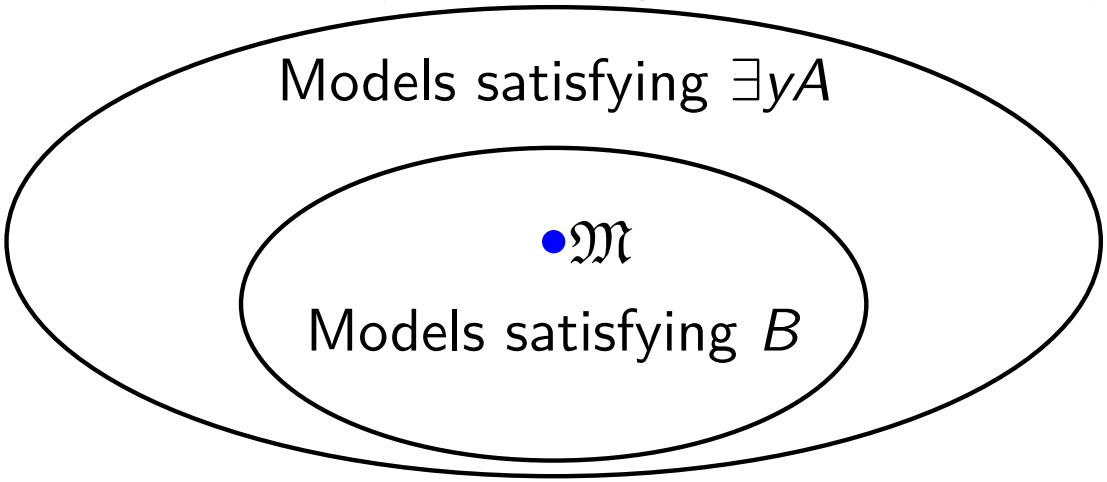



From $\mathfrak{M} \models \exists yA$,
produce quantifier-free
$B = mbp(\mathfrak{M}, \exists yA)$
such that
$\mathcal{T} \models B \Rightarrow \exists yA$.

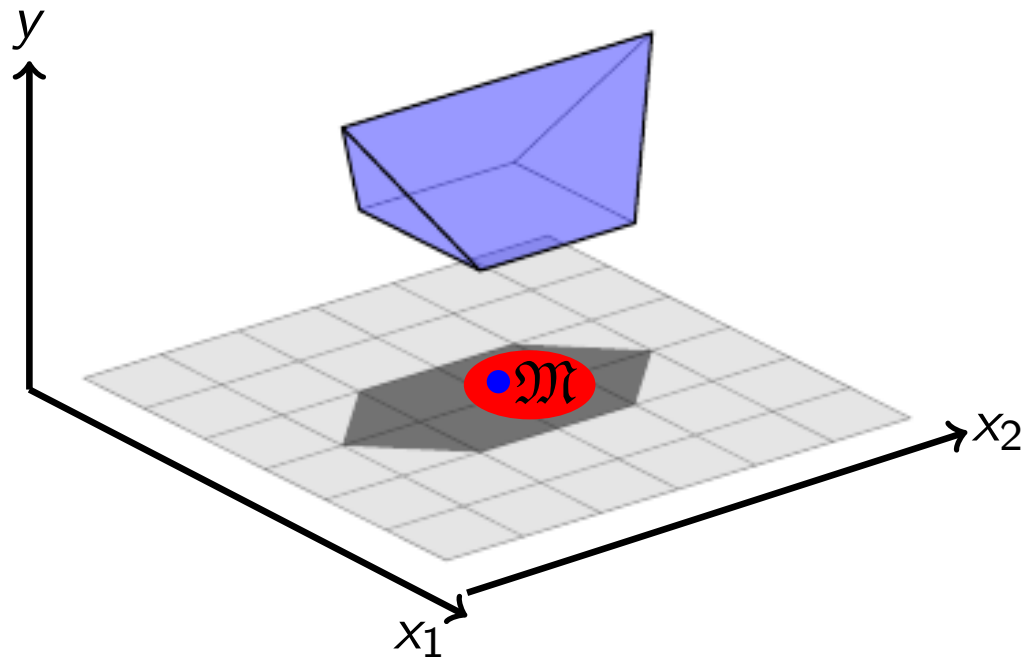


# The approach by Bjørner and Janota 2/3

Relies on theory $\mathcal{T}$ having model-based projection




From $\mathfrak{M} \models \exists yA$,
produce quantifier-free
$B = mbp(\mathfrak{M}, \exists yA)$
such that
$\mathcal{T} \models B \Rightarrow \exists yA$.



# The approach by Bjørner and Janota 3/3

# The approach by Bjørner and Janota 3/3

And $\exists y A$ should be covered by finitely many $B$'s:
$\mathcal{T} \models (\exists y A) \Leftrightarrow \bigvee_i mbp(\mathfrak{M}_i, \exists y A)$ for a finite number of models $\mathfrak{M}_i$.

specific form of quantifier elimination

# The approach by Bjørner and Janota 3/3

And $\exists y A$ should be covered by finitely many $B$'s:
$\mathcal{T} \models (\exists y A) \Leftrightarrow \bigvee_i mbp(\mathfrak{M}_i, \exists y A)$ for a finite number of models $\mathfrak{M}_i$.

specific form of quantifier elimination

From there, an algorithm is given that applies the mbp computations lazily.

# Questions:

On purely existential problems, does the Bjørner-Janota algorithm collapse to MCSAT?
I suspect not: the form of laziness seems distinct, but it also seems complementary.

# Questions:

On purely existential problems, does the Bjørner-Janota algorithm collapse to MCSAT?
I suspect not: the form of laziness seems distinct, but it also seems complementary.

Ongoing work:

- extend MCSAT with the Bjørner-Janota approach for handling arbitrary alternations of quantifiers.
- what is the notion of proof production in this approach?

# 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?


M. Armand, G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Werner.
Verifying SAT and SMT in Coq for a fully automated decision procedure.
In G. Faure, S. Lengrand, and A. Mahboubi, editors, *Proc. of the 2011 Work. on Proof-Search in Axiomatic Theories and Type Theories (PSATTT'11)*, 2011.
Available at `http://hal.inria.fr/PSATTT11`


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