Proofs in Satisfiability Modulo Theories

Clark Barrett (NYU)
Leonardo de Moura (Microsoft Research)
Pascal Fontaine (Inria, Loria, U. Lorraine)

APPA: All about Proofs, Proofs for All
$\forall X \,.\, X\Pi$

July 18, 2014

Outline

Motivation

Automatic analysis of computer hardware and software requires *engines* capable of reasoning efficiently about large and complex systems.

Boolean engines such as *Binary Decision Diagrams* and *SAT solvers* are typical engines of choice for today's industrial verification applications.

However, systems are usually designed and modeled at a higher level than the Boolean level and the translation to Boolean logic can be expensive.

A primary goal of research in *Satisfiability Modulo Theories* (SMT) is to create verification engines that can reason natively at a higher level of abstraction, while still retaining the speed and automation of today's Boolean engines.

Satisfiability Modulo Theories

Is the following formula satisfiable?

$$\textit{read}(\textit{write}(a, i, v), i) \neq v$$

Satisfiability Modulo Theories

Is the following formula satisfiable?

$\mathit{read}(\mathit{write}(a, i, v), i) \neq v$

- If the set of allowable models is unrestricted, then the answer is yes.

Satisfiability Modulo Theories

Is the following formula satisfiable?

$\textit{read}(\textit{write}(a, i, v), i) \neq v$

- If the set of allowable models is unrestricted, then the answer is yes.
- However, if we only consider models that obey the axioms for *read* and *write* then the answer is no.

Satisfiability Modulo Theories

T-satisfiability

For a theory T, the T*-satisfiability problem* consists of deciding whether there exists a model $\mathcal{A}$ and variable assignment α such that $(\mathcal{A}, \alpha) \models T \cup \varphi$ for a given formula φ.

SAT and Theories

- An SMT solver uses a fast SAT solver for Boolean reasoning
- Coupled with specialized theory solvers for theory reasoning

What is SMT good for?

Generic Reasoning

- Given some conditions X, is it possible for Y to happen, and if so how?
- X and Y must be expressible in logic
- SMT offers a lot of expressive power
- Possibility to define a new theory if all else fails

What SMT is NOT good for

- Reasoning in the presense of uncertainty (e.g. probabilities)
- Heavy use of quantifiers
- Difficult constraints with no Boolean structure (e.g. Linear Programs)

Proofs and SMT: a history

First Attempts

- Cooperating Validity Checker (CVC), 2002[a]
  - First SMT solver to attempt proof-production
  - Wanted to be able to independently certify results
  - Aid in finding and correcting correctness bugs
  - Surprisingly - most important contribution was use in producing explanations of inconsistency

[a] Stump, Barrett, Dill. **CVC: A Cooperating Validity Checker**, CAV '02.

Proofs and SMT: a history

Communication with skeptical proof assistants

- CVC Lite, 2005[a]
  - Successor to CVC, ad hoc proof format
  - Translator from proof format to HOL Light
  - Provide access to efficient decision procedures within HOL Light
  - And enable use of HOL Light as a proof-checker for CVC Lite
- haRVey, 2006[b]
  - Integration with Isabelle/HOL
- CVC3, 2008[c]
  - Effort to certify SMT-LIB benchmark library
  - Found benchmarks with incorrect status
  - Found bug in CVC3

[a] McLaughlin, Barrett, Ge. **Cooperating Theorem Provers: A Case Study Combining HOL-Light and CVC Lite**, PDPAR '05.

[b] Fontaine, Marion, Merz, Nieto, Tiu. **Expressiveness + Automation + Soundness: Towards Combining SMT Solvers and Interactive Proof Assistants**, TACAS '06.

[c] Ge, Barrett. **Proof Translation and SMT-LIB Benchmark Certification: A Preliminary Report**, SMT '08.

Proofs and SMT: a history

Additinal solvers support proofs

- Fx7, 2008[a]
  - Quantified reasoning, custom proof-checker
- MathSAT4, 2008[b]
  - Internal proof engine for unsat cores and interpolants
- Z3, 2008[c]
  - Proof traces - single rule for theory lemmas
- veriT, 2009[d]
  - Proof production a primary goal in veriT

[a] Moskal. **Rocket-Fast Proof Checking for SMT Solvers**, TACAS '08.

[b] Bruttomesso, Cimatti, Franzén, Griggio, Sebastiani. **The MathSAT 4 SMT Solver**, CAV '08.

[c] de Moura, Bjørner. **Proofs and Refutations, and Z3**, LPAR '08.

[d] Bouton, de Oliveira, Déharbe, Fontaine. **veriT: An Open, Trustable and Efficient SMT-Solver**, CADE '09.

Proofs and SMT: a history

Current Status

- No agreed-upon format for proofs in SMT
- Solvers targeting self-contained, independently-checkable proofs
  - CVC4, veriT
- Proof traces
  - Z3
- Solvers using proof technology to drive other features (e.g. interpolants)
  - MathSAT, SMTInterpol

Satisfiability Modulo Theories ≈ SAT + expressiveness

Satisfiability of first-order formulas
with interpreted and non-interpreted predicates and functions

Interpreted: Axioms (e.g. arrays) or Structure (e.g. linear arithmetic)

- SAT solvers

$$\neg\big[\,(p \Rightarrow q) \Rightarrow \big[\,(\neg p \Rightarrow q) \Rightarrow q\big]\big]$$

- congruence closure (uninterpreted symbols + equality)

$$a = b \wedge \big[f(a) \neq f(b) \vee (p(a) \wedge \neg p(b))\big]$$

- in combination with arithmetic

$$a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \big[f(a) \neq f(b) \vee (p(a) \wedge \neg p(b + x))\big]$$

- quantifiers
- ...

Alt-Ergo, Barcelogic, CVC4, MathSAT, OpenSMT, SMTInterpol, veriT, Yices, z3 ...

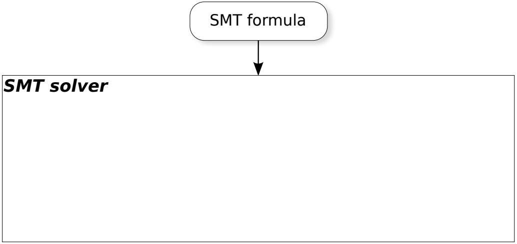
Standard input language: SMT-LIB 2.0

$$a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x)) \right]$$

In SMT-LIB 2.0 format:

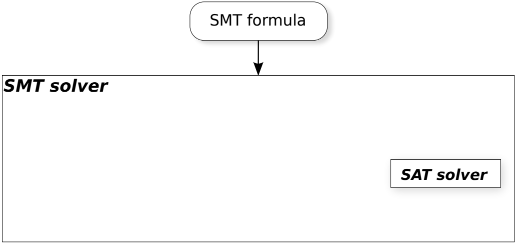
```
(set-logic QF_UFLRA)
(set-info :source | Example formula in SMT-LIB 2.0 |)
(set-info :smt-lib-version 2.0)
(declare-fun f (Real) Real)
(declare-fun q (Real) Bool)
(declare-fun a () Real)
(declare-fun b () Real)
(declare-fun x () Real)
(assert (and (<= a b) (<= b (+ a x)) (= x 0)
             (or (not (= (f a) (f b)))
                 (and (q a) (not (q (+ b x)))))))
(check-sat)
(exit)
```

From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x)) \right]$

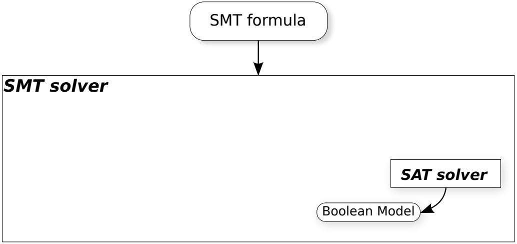
From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x)) \right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)}) \right]$

From propositional SAT to SMT

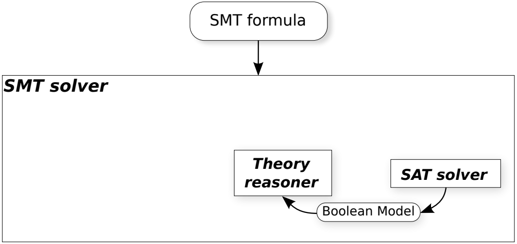


Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

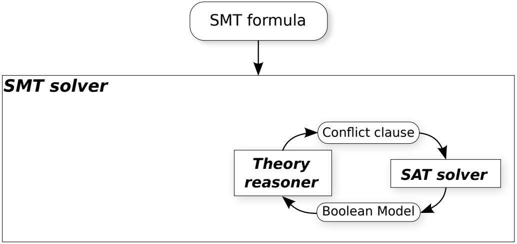
Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

From propositional SAT to SMT



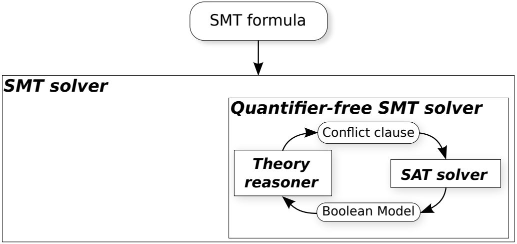
Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$
To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$
Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$
Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$
To SAT solver: $p_{a\leq b} \wedge p_{b\leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$
Boolean model: $p_{a\leq b}, p_{b\leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$
Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable
New clause: $\neg p_{a\leq b} \vee \neg p_{b\leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x)) \right]$

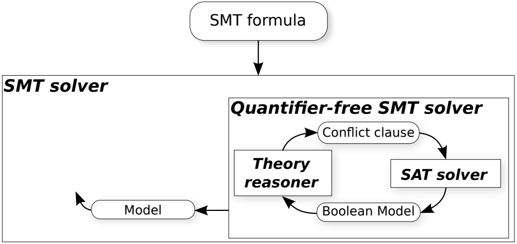
To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)}) \right]$

Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b+x))\right]$

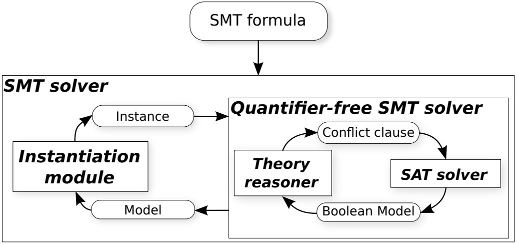
To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

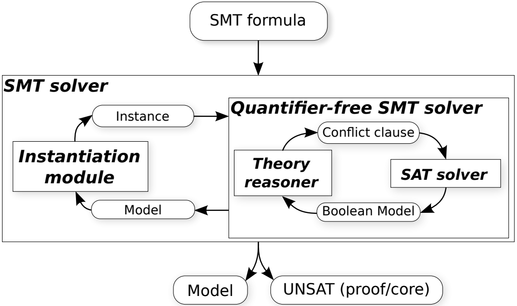
New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$
To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$
Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$
Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable
New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

From propositional SAT to SMT: in practice

- online decision procedures
  theory checks propositional assignment on the fly
- small explanations
  unsat core of propositional assignment
  discard classes of propositional assignments (not one by one)
- theory propagation
  instead of guessing propositional variable assignments, SAT solver assigns theory-entailed literals
- ackermannization, simplifications, and other magic

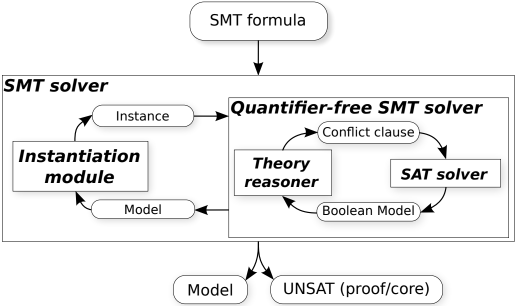
Theory and quantifier reasoning

- theory reasoning techniques specific to theories...
- ...but (mostly) interact similarly with the SAT solver
- uninterpreted symbols and equality: congruence closure
- linear arithmetic: mostly simplex
- quantifiers: mostly instantiation

More details to come later (with proof production)

Outline

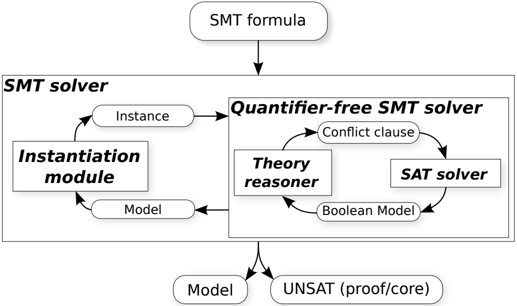
From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b+x))\right]$

SMT proof: interleaving of SAT proof and theory reasoning proof

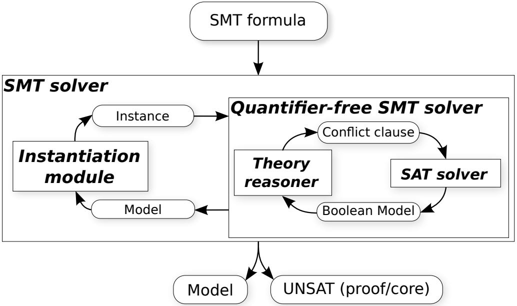
From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x)) \right]$
To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)}) \right]$

SMT proof: interleaving of SAT proof and theory reasoning proof

From propositional SAT to SMT



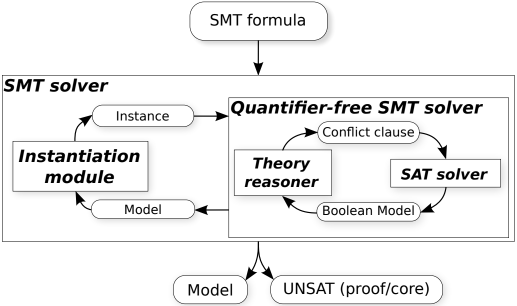
Input: $a \le b \wedge b \le a + x \wedge x = 0 \wedge \left[f(a) \ne f(b) \vee (q(a) \wedge \neg q(b + x))\right]$

To SAT solver: $p_{a \le b} \wedge p_{b \le a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

New theory clause: $\neg p_{a \le b} \vee \neg p_{b \le a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

SMT proof: interleaving of SAT proof and theory reasoning proof

From propositional SAT to SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x)) \right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)}) \right]$

New theory clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

New theory clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{q(a)} \vee p_{q(b+x)}$

SMT proof: interleaving of SAT proof and theory reasoning proof

SMT in practice

- online decision procedures
  theory checks propositional assignment on the fly
  No influence on proof
- small explanations
  unsat core of propositional assignment
  discard classes of propositional assignments (not one by one)
  No influence on proof (small theory clauses)
- theory propagation
  instead of guessing propositional variable assignments, SAT solver assigns theory-entailed literals
  May need explanation (theory clause)
- ackermannization, simplifications, and other magic
  Sometimes cumbersome to prove

Challenge: collect enough information

Theory reasoning proofs
Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$

Theory reasoning proofs
Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$

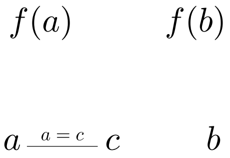
$f(a)$ $f(b)$

a c b

- each term in its equivalence class

Theory reasoning proofs
Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$
And literals: $a = c$



- each term in its equivalence class
- equality $\longrightarrow$ class merge

Theory reasoning proofs
Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$
And literals: $a = c, c = b$

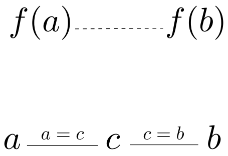
$$f(a) \qquad f(b)$$

$$a \xrightarrow{\;a=c\;} c \xrightarrow{\;c=b\;} b$$

- each term in its equivalence class
- equality $\longrightarrow$ class merge

Theory reasoning proofs
Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$
And literals: $a = c, c = b$



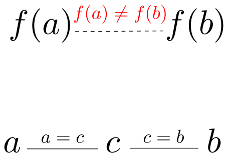
- each term in its equivalence class
- equality $\longrightarrow$ class merge
- congruence $\longrightarrow$ class merge

Theory reasoning proofs

Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$

And literals: $a = c, c = b, f(a) \neq f(b)$



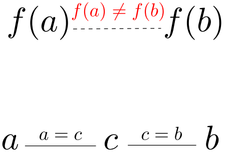
- each term in its equivalence class
- equality $\longrightarrow$ class merge
- congruence $\longrightarrow$ class merge
- detect conflicts

Theory reasoning proofs

Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$

And literals: $a = c, c = b, f(a) \neq f(b)$

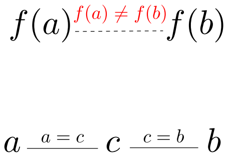


- each term in its equivalence class
- equality $\longrightarrow$ class merge
- congruence $\longrightarrow$ class merge
- detect conflicts

In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proofs
Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$
And literals: $a = c, c = b, f(a) \neq f(b)$



- each term in its equivalence class
- equality $\longrightarrow$ class merge
- congruence $\longrightarrow$ class merge
- detect conflicts

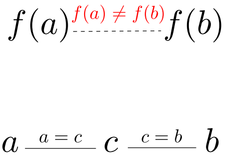
In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:

Theory reasoning proofs

Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$
And literals: $a = c, c = b, f(a) \neq f(b)$



- each term in its equivalence class
- equality $\longrightarrow$ class merge
- congruence $\longrightarrow$ class merge
- detect conflicts

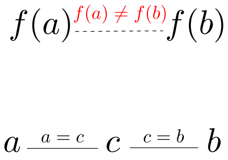
In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:

- conflict $f(a) \neq f(b)$ with an implied literal

Theory reasoning proofs
Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$
And literals: $a = c, c = b, f(a) \neq f(b)$



- each term in its equivalence class
- equality $\longrightarrow$ class merge
- congruence $\longrightarrow$ class merge
- detect conflicts

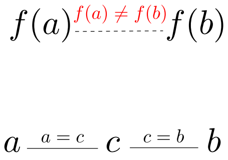
In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:

- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \vee f(a) = f(b)$

Theory reasoning proofs
Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$
And literals: $a = c, c = b$, $f(a) \neq f(b)$



- each term in its equivalence class
- equality $\longrightarrow$ class merge
- congruence $\longrightarrow$ class merge
- detect conflicts

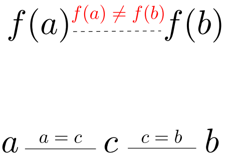
In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:

- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \vee f(a) = f(b)$
- and $a = b$ comes from transitivity: $a \neq c \vee c \neq b \vee a = b$

Theory reasoning proofs
Congruence closure

Consider the terms: a, b, c, $f(a)$, $f(b)$
And literals: $a = c, c = b$, $f(a) \neq f(b)$



- each term in its equivalence class
- equality $\longrightarrow$ class merge
- congruence $\longrightarrow$ class merge
- detect conflicts

In practice: efficient (merge, congruence and conflict detection)

Theory reasoning proof, from graph:

- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \vee f(a) = f(b)$
- and $a = b$ comes from transitivity: $a \neq c \vee c \neq b \vee a = b$
- *resolution* compute the theory clause: $a \neq c \vee c \neq b \vee f(a) = f(b)$

Theory reasoning proofs

Combination of theories

Theory reasoning proof, with combination of theories:

- conflict $f(a) \neq f(b)$ with an implied literal

Theory reasoning proofs
Combination of theories

Theory reasoning proof, with combination of theories:

- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \vee f(a) = f(b)$

Theory reasoning proofs
Combination of theories

Theory reasoning proof, with combination of theories:

- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \lor f(a) = f(b)$
- and $a = b$ comes from another theory clause:
  $\neg a \leq b \lor \neg b \leq a + x \lor x \neq 0 \lor a = b$

Theory reasoning proofs
Combination of theories

Theory reasoning proof, with combination of theories:

- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \vee f(a) = f(b)$
- and $a = b$ comes from another theory clause:
  $\neg a \leq b \vee \neg b \leq a + x \vee x \neq 0 \vee a = b$
- *resolution* compute the theory clause:
  $\neg a \leq b \vee \neg b \leq a + x \vee x \neq 0 \vee f(a) = f(b)$

Theory reasoning proofs
Combination of theories

Theory reasoning proof, with combination of theories:

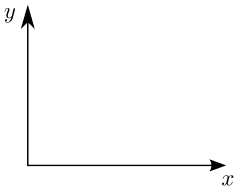
- conflict $f(a) \neq f(b)$ with an implied literal
- entailed by congruence: $a \neq b \vee f(a) = f(b)$
- and $a = b$ comes from another theory clause:
  $\neg a \leq b \vee \neg b \leq a + x \vee x \neq 0 \vee a = b$
- *resolution* compute the theory clause:
  $\neg a \leq b \vee \neg b \leq a + x \vee x \neq 0 \vee f(a) = f(b)$

Over-simplification :

- delayed theory combination
- model-based combination

Theory reasoning proofs
Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

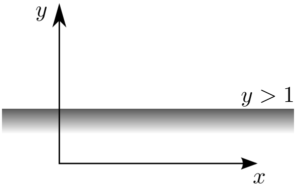


Theory reasoning proofs

Linear arithmetic

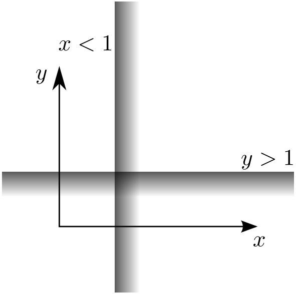
- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

- $y > 1$



Theory reasoning proofs
Linear arithmetic

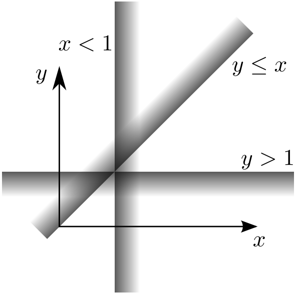
- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate



- $y > 1, x < 1$

Theory reasoning proofs
Linear arithmetic

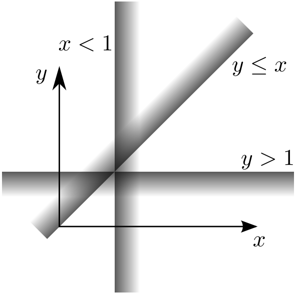
- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate



- $y > 1$, $x < 1$, $y \leq x$

Theory reasoning proofs
Linear arithmetic

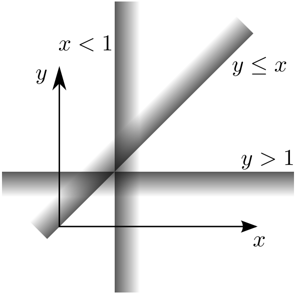
- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate



- $y > 1$, $x < 1$, $y \leq x$
- inconsistency

Theory reasoning proofs
Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate

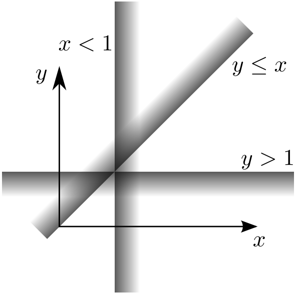


- $y > 1, x < 1, y \leq x$
- inconsistency

$$\begin{array}{rr} & x < 1 \\ + & y \leq x \\ - & y > 1 \\ \hline & 0 < 0 \end{array}$$

Theory reasoning proofs
Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate



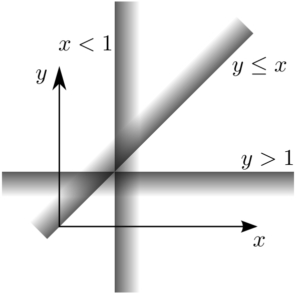
- $y > 1$, $x < 1$, $y \leq x$
- inconsistency

$$\begin{array}{rr} & x < 1 \\ + & y \leq x \\ - & y > 1 \\ \hline & 0 < 0 \end{array}$$

- Clause: $\neg y > 1 \vee \neg x < 1 \vee \neg y \leq x$

Theory reasoning proofs
Linear arithmetic

- Many linear arithmetic decision procedures based on simplex
- Simplex detects inconsistency
- Farkas lemma can be used to provide certificate



- $y > 1$, $x < 1$, $y \leq x$
- inconsistency

$$\begin{array}{rr} & x < 1 \\ + & y \leq x \\ - & y > 1 \\ \hline & 0 < 0 \end{array}$$

- Clause: $\neg y > 1 \vee \neg x < 1 \vee \neg y \leq x$

And also

- integers: branches, cuts
- simplifications, bound propagations...

Quantifiers and proofs

- Quantifiers mainly come from instantiation
- Proof is simply

$$\neg\forall x\, \varphi(x) \vee \varphi(t)$$

- $\forall x \varphi(x)$ is an abstract Boolean variable for the SAT solver
- Resolution, again
- Skolemization is a problem though

Other theories

Other theories

- arrays
- inductive data types
- bit-vectors
- strings
- non-linear arithmetic

Outline

CVC4 proof (1/3)

```
(check
(% a var_real
(% b var_real
(% x var_real
(% f (term (arrow Real Real))
(% q (term (arrow Real Bool))
(% @F1 (th_holds (<=_Real (a_var_real a) (a_var_real b)))
(% @F2 (th_holds (<=_Real (a_var_real b) (+_Real (a_var_real a) (a_var_real x))))
(% @F3 (th_holds (= Real (a_var_real x) (a_real 0/1)))
(% @F4 (th_holds (or (not (= Real (apply _ _ f (a_var_real a)) (apply _ _ f (a_var_real b))))
                     (and (= Bool (apply _ _ q (a_var_real a)) btrue)
                          (= Bool (apply _ _ q (+_Real (a_var_real b) (a_var_real x))) bfalse))))
(: (holds cln)

(decl_atom (<=_Real (a_var_real a) (a_var_real b)) (\ v1 (\ a1
(decl_atom (<=_Real (a_var_real b) (+_Real (a_var_real a) (a_var_real x))) (\ v2 (\ a2
(decl_atom (= Real (a_var_real x) (a_real 0/1)) (\ v3 (\ a3
(decl_atom (= Real (a_var_real a) (a_var_real b)) (\ v4 (\ a4
(decl_atom (= Real (apply _ _ f (a_var_real a)) (apply _ _ f (a_var_real b))) (\ v5 (\ a5
(decl_atom (= Bool (apply _ _ q (a_var_real a)) btrue) (\ v6 (\ a6
(decl_atom (= Bool (apply _ _ q (+_Real (a_var_real b) (a_var_real x))) bfalse) (\ v7 (\ a7
(decl_atom (<=_Real (a_var_real b) (a_var_real a)) (\ v8 (\ a8
(decl_atom (= Real (a_var_real a) (+_Real (a_var_real b) (a_var_real x))) (\ v9 (\ a9
(decl_atom (and (= Bool (apply _ _ q (a_var_real a)) btrue)
                (= Bool (apply _ _ q (+_Real (a_var_real b) (a_var_real x))) bfalse))
  (\ v10 (\ a10
```

CVC4 proof (2/3)

```
; CNFication
(satlem _ _ (asf _ _ _ a1 (\ l1 (clausify_false (contra _ @F1 l1)))) (\ C1
(satlem _ _ (asf _ _ _ a2 (\ l2 (clausify_false (contra _ @F2 l2)))) (\ C2
(satlem _ _ (asf _ _ _ a3 (\ l3 (clausify_false (contra _ @F3 l3)))) (\ C3
(satlem _ _ (ast _ _ _ a5 (\ l5 (asf _ _ _ a6 (\ l6 (clausify_false (contra _
  (and_elim_1 _ _ (or_elim_1 _ _ (not_not_intro _ l5) @F4)) l6)))))) (\ C4
(satlem _ _ (ast _ _ _ a5 (\ l5 (asf _ _ _ a7 (\ l7 (clausify_false (contra _
  (and_elim_2 _ _ (or_elim_1 _ _ (not_not_intro _ l5) @F4)) l7)))))) (\ C5

; Theory lemmas
; ~a4 ^ a1 ^ a8 => false
(satlem _ _ (asf _ _ _ a4 (\ l4 (ast _ _ _ a1 (\ l1 (ast _ _ _ a8 (\ l8
 (clausify_false (contra _ l1
 (or_elim_1 _ _ (not_not_intro _ (<=_to_>=_Real _ _ l8)) (not_=_to_>=_=<_Real _ _ l4))))))))))
 (\ C6
; a2 ^ a3 ^ ~a8 => false
(satlem _ _ (ast _ _ _ a2 (\ l2 (ast _ _ _ a3 (\ l3 (asf _ _ _ a8 (\ l8 (clausify_false
 (poly_norm_>= _ _ _ (<=_to_>=_Real _ _ l2) (pn_- _ _ _ _ _ (pn_+ _ _ _ _ _
  (pn_var a) (pn_var x)) (pn_var b)) (\ pn2
 (poly_norm_= _ _ _ (symm _ _ _ l3) (pn_- _ _ _ _ _ (pn_const 0/1) (pn_var x)) (\ pn3
 (poly_norm_> _ _ _ (not_<=_to_>_Real _ _ l8) (pn_- _ _ _ _ _ (pn_var b) (pn_var a)) (\ pn8
 (lra_contra_> _ (lra_add_>_>= _ _ _ pn8 (lra_add_=_>= _ _ _ pn3 pn2)))))))))))))))) (\ C7
; a4 ^ ~a5 => false
(satlem _ _ (ast _ _ _ a4 (\ l4 (asf _ _ _ a5 (\ l5 (clausify_false
 (contra _ (cong _ _ _ _ _ _ (refl _ f) l4) l5)))))) (\ C8
```

CVC4 proof (3/3)

```
; a3 ^ a4 ^ ~a9 => false
(satlem _ _ (ast _ _ _ a3 (\ l3 (ast _ _ _ a4 (\ l4 (asf _ _ _ a9 (\ l9 (clausify_false
 (poly_norm_= _ _ _ (symm _ _ _ l3) (pn_- _ _ _ _ _ _ (pn_const 0/1) (pn_var x)) (\ pn3
 (poly_norm_= _ _ _ l4 (pn_- _ _ _ _ _ _ (pn_var a) (pn_var b)) (\ pn4
 (poly_norm_distinct _ _ _ l9 (pn_- _ _ _ _ _ _ (pn_+ _ _ _ _ _
  (pn_var b) (pn_var x)) (pn_var a)) (\ pn9
 (lra_contra_distinct _ (lra_add_=_distinct _ _ _
  (lra_add_=_= _ _ _ pn3 pn4) pn9)))))))))))))) (\ C9
; a9 ^ a6 ^ a7 => false
(satlem _ _ (ast _ _ _ a9 (\ l9 (ast _ _ _ a6 (\ l6 (ast _ _ _ a7 (\ l7 (clausify_false
 (contra _ (trans _ _ _ _ (trans _ _ _ _ (symm _ _ _ l6) (cong _ _ _ _ _ _
  (refl _ q) l9)) l7) b_true_not_false)))))))) (\ C10

; Resolution proof
(satlem_simplify _ _ _ (R _ _ (Q _ _ (Q _ _ C6 C1 v1) (Q _ _ (Q _ _ C7 C2 v2) C3 v3) v8)
(Q _ _ (Q _ _ (Q _ _ (Q _ _ (R _ _ C9 C10 v9) C3 v3) C4 v6) C5 v7) C8 v5) v4)
(\ x x))))))))))))))))))))))))))))))))))))))))))))))))))))))))
```

veriT proof (1/2)

```
(set .c1 (input :conclusion ((and (<= a b) (<= b (+ a x)) (= x 0)
                             (or (not (= (f b) (f a))) (and (q a) (not (q (+ b x)))))))))
(set .c2 (and :clauses (.c1) :conclusion ((<= a b))))
(set .c3 (and :clauses (.c1) :conclusion ((<= b (+ a x)))))
(set .c4 (and :clauses (.c1) :conclusion ((= x 0))))
(set .c5 (and :clauses (.c1) :conclusion
           ((or (not (= (f b) (f a))) (and (q a) (not (q (+ b x))))))))
(set .c6 (and_pos :conclusion ((not (and (q a) (not (q (+ b x))))) (q a))))
(set .c7 (and_pos :conclusion ((not (and (q a) (not (q (+ b x))))) (not (q (+ b x))))))
(set .c8 (or :clauses (.c5) :conclusion
           ((not (= (f b) (f a))) (and (q a) (not (q (+ b x)))))))
(set .c9 (eq_congruent :conclusion ((not (= a b)) (= (f b) (f a)))))
(set .c10 (la_disequality :conclusion ((or (= a b) (not (<= a b)) (not (<= b a))))))
(set .c11 (or :clauses (.c10) :conclusion ((= a b) (not (<= a b)) (not (<= b a)))))
(set .c12 (resolution :clauses (.c11 .c2) :conclusion ((= a b) (not (<= b a)))))
(set .c13 (la_generic :conclusion ((not (<= b (+ a x))) (<= b a) (not (= x 0)))))
(set .c14 (resolution :clauses (.c13 .c3 .c4) :conclusion ((<= b a))))
(set .c15 (resolution :clauses (.c12 .c14) :conclusion ((= a b))))
(set .c16 (resolution :clauses (.c9 .c15) :conclusion ((= (f b) (f a)))))
(set .c17 (resolution :clauses (.c8 .c16) :conclusion ((and (q a) (not (q (+ b x)))))))
(set .c18 (resolution :clauses (.c6 .c17) :conclusion ((q a))))
(set .c19 (resolution :clauses (.c7 .c17) :conclusion ((not (q (+ b x))))))
```

veriT proof (2/2)

```
(set .c20 (eq_congruent_pred :conclusion ((not (= a (+ b x))) (not (q a)) (q (+ b x)))))
(set .c21 (resolution :clauses (.c20 .c18 .c19) :conclusion ((not (= a (+ b x))))))
(set .c22 (la_disequality :conclusion
          ((or (= a (+ b x)) (not (<= a (+ b x))) (not (<= (+ b x) a))))))
(set .c23 (or :clauses (.c22) :conclusion
          ((= a (+ b x)) (not (<= a (+ b x))) (not (<= (+ b x) a)))))
(set .c24 (resolution :clauses (.c23 .c21) :conclusion
          ((not (<= a (+ b x))) (not (<= (+ b x) a)))))
(set .c25 (eq_congruent_pred :conclusion
          ((not (= a b)) (not (= (+ a x) (+ b x))) (<= a (+ b x)) (not (<= b (+ a x))))))
(set .c26 (eq_congruent :conclusion ((not (= a b)) (not (= x x)) (= (+ a x) (+ b x)))))
(set .c27 (eq_reflexive :conclusion ((= x x))))
(set .c28 (resolution :clauses (.c26 .c27) :conclusion ((not (= a b)) (= (+ a x) (+ b x)))))
(set .c29 (resolution :clauses (.c25 .c28) :conclusion
          ((not (= a b)) (<= a (+ b x)) (not (<= b (+ a x))))))
(set .c30 (resolution :clauses (.c29 .c3 .c15) :conclusion ((<= a (+ b x)))))
(set .c31 (resolution :clauses (.c24 .c30) :conclusion ((not (<= (+ b x) a)))))
(set .c32 (la_generic :conclusion ((<= (+ b x) a) (not (= a b)) (not (= x 0)))))
(set .c33 (resolution :clauses (.c32 .c4 .c15 .c31) :conclusion ()))
```

z3 proof (1/2)

```
(let (($x82 (q b)) (?x49 (* (- 1.0) b)) (?x50 (+ a ?x49))
      ($x51 (<= ?x50 0.0)) (?x35 (f b)) (?x34 (f a))
      ($x36 (= ?x34 ?x35)) ($x37 (not $x36))
      ($x43 (or $x37 (and (q a) (not (q (+ b x))))))
      ($x33 (= x 0.0)) (?x57 (+ a ?x49 x)) ($x56 (>= ?x57 0.0))
      ($x44 (and (<= a b) (<= b (+ a x)) $x33 $x43))
      (@x60 (monotonicity (rewrite (= (<= a b) $x51))
                          (rewrite (= (<= b (+ a x)) $x56))
                          (= $x44 (and $x51 $x56 $x33 $x43))))
      (@x61 (mp (asserted $x44) @x60 (and $x51 $x56 $x33 $x43)))
      (@x62 (and-elim @x61 $x51)) ($x71 (>= ?x50 0.0)))
(let ((@x70 (trans (monotonicity (and-elim @x61 $x33) (= ?x57 (+ a ?x49 0.0)))
             (rewrite (= (+ a ?x49 0.0) ?x50)) (= ?x57 ?x50))))
(let ((@x74 (mp (and-elim @x61 $x56) (monotonicity @x70 (= $x56 $x71)) $x71)))
(let ((@x121 (monotonicity (symm ((_ th-lemma arith eq-propagate 1 1) @x74 @x62 (= a b)) (= b a))
                           (= $x82 (q a)))))
(let (($x38 (q a)) ($x96 (or (not $x38) $x82)) ($x97 (not $x96)))
(let ((@x115 (monotonicity (symm ((_ th-lemma arith eq-propagate 1 1) @x74 @x62 (= a b)) (= b a))
                           (= ?x35 ?x34))))
(let (($x100 (or $x37 $x97)))
(let ((@x102 (monotonicity (rewrite (= (and $x38 (not $x82)) $x97))
                           (= (or $x37 (and $x38 (not $x82))) $x100))))
(let (($x85 (not $x82)))
(let (($x88 (and $x38 $x85)))
(let (($x91 (or $x37 $x88)))
(let ((@x81 (trans (monotonicity (and-elim @x61 $x33) (= (+ b x) (+ b 0.0)))
             (rewrite (= (+ b 0.0) b)) (= (+ b x) b))))
(let ((@x87 (monotonicity (monotonicity @x81 (= (q (+ b x)) $x82)) (= (not (q (+ b x))) $x85))))
```

z3 proof (2/2)

```
(let ((@x93 (monotonicity (monotonicity @x87 (= (and $x38 (not (q (+ b x)))) $x88))
                          (= $x43 $x91))))
(let ((@x103 (mp (mp (and-elim @x61 $x43) @x93 $x91) @x102 $x100)))
(let ((@x119 (unit-resolution (def-axiom (or $x96 $x38))
                              (unit-resolution @x103 (symm @x115 $x36) $x97) $x38)))
(let ((@x118 (unit-resolution (def-axiom (or $x96 $x85))
                              (unit-resolution @x103 (symm @x115 $x36) $x97) $x85)))
(unit-resolution @x118 (mp @x119 (symm @x121 (= $x38 $x82)) $x82) false)))))))))))))))
```

Outline

Applications

Current Applications

- Proof reconstruction within skeptical proof assistants [a], [b], [c]
- Interpolant generation [d], [e], [f]
- Unsat core computation [g]

---

[a] Keller. **A Matter of Trust: Skeptical Communication Between Coq and External Provers**, PhD Thesis, Ecole Polytechnique, 2013.

[b] Armand, Faure, Grégoire, Keller, Thery, Werner. **A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses**, CPP '11.

[c] Böhme. **Proof Reconstruction for Z3 in Isabelle/HOL**, SMT'09.

[d] Reynolds, Tinelli, Hadarean. **Certified Interpolant Generation for EUF**, SMT '11.

[e] Hofferek, Gupta, Könighofer, Jiang, Bloem. **Synthesizing Multiple Boolean Functions using Interpolation on a Single Proof**, FMCAD '13.

[f] McMillan. **Interpolants from Z3 Proofs**, FMCAD '11.

[g] Déharbe, Fontaine, Guyot, Voisin. **SMT Solvers for Rodin**, Abstract State Machines '12.

Challenges

Challenges

- Challenge to collect and store proof information efficiently
- Producing proofs for sophisticated preprocessing techniques
- Producing proofs for modules that use external tools
- Standardizing a proof format

Lean Theorem Prover

- New theorem prover started by L. de Moura and Soonho Kong.
- Contributors: Jeremy Avigad, Cody Roux, Floris van Doorn, Parikshit Khanna
- Many thanks to: Georges Gonthier, Nikhil Swamy, Vladimir Voevodsky

- Open source (Apache 2.0), `https://github.com/leanprover/lean`
- can be used as an automatic prover (SMT), and as a proof assistant
- Based on Type Theory, and incorporates ideas of many other systems:
  Agda, Coq, HOL-Light, Isabelle, PVS, ...

Lean: Two Layers Architecture

- First layer: type checker, APIs for creating terms, environment, ...
- Configuration options: e.g., impredicative Prop, proof irrelevance, ...
- Universe polymorphism.
- 5k lines of C++ code.

- Second layer: additional (trusted) components.
- Example: inductive datatypes (extra 500 lines of code).
- We currently support two flavors/instances: Standard and HoTT.

Lean: As a Library

- Meant to be used as a standalone system and as a software library.
- Extensive API and can be easily embedded in other systems.
- SMT solvers can use the Lean API to create proof terms that can be independently checked.
- APIs in C++, Lua (and Python coming soon).

Lean: Proofs

- More expressive language for encoding proofs provides several advantages.
- We can easily add new “proof rules” without modifying the proof checker (i.e., type checker).
- Proof rules such as mp and monotonicity used in Z3 are just theorems in Lean.

Lean: Automation

- First, define theory, then prove theorems/properties, then implement automation.
- Example: suppose we are implementing a procedure for Presburger Arithmetic.

```
theorem add_comm (n m:nat) : n + m = m + n
:= induction_on m
  (trans (add_zero_right _) (symm (add_zero_left _)))
  (take k IH,
     calc n + succ k = succ (n+k) : add_succ_right _ _
                 ... = succ (k + n) : {IH}
                 ... = succ k + n : symm (add_succ_left _ _))
```

Lean: Automation

- Pre-processing steps such as Skolemization can be supported in a similar way.

```
theorem skolem_th {A : Type} {B : A -> Type} {P : forall x : A, B x -> Bool} :
              (forall x, exists y, P x y) = (exists f, (forall x, P x (f x)))
 := iff_intro
     (assume H : (forall x, exists y, P x y), axiom_of_choice H)
     (assume H : (exists f, (forall x, P x (f x))),
        take x, obtain (fw : forall x, B x) (Hw : forall x, P x (fw x)), from H,
          exists_intro (fw x) (Hw x))
```

Lean: Pre-processing

- The pre-processing "issue" is addressed by providing a generic rewriting engine that can use any previously proved theorems.
- The engine accepts two kinds of theorems: congruence theorems and (conditional) equations.
- It also supports a λ-Prolog like engine.

```
theorem forall_or_distributel {A : Type} (p : Bool) (q : A -> Bool)
        : (forall x, q x \/ p) = ((forall x, q x) \/ p)
theorem forall_or_distributer {A : Type} (p : Bool) (q : A -> Bool)
        : (forall x, p \/ q x) = (p \/ forall x, q x)
```