Proof Certificates in Satisfiability Modulo Theories

Clark Barrett, Stanford University

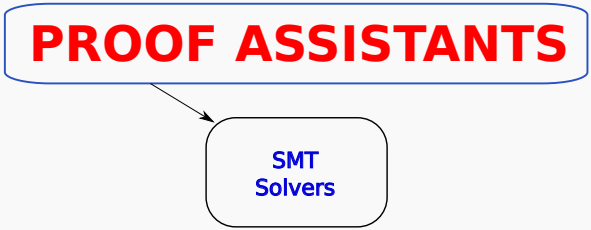
Agenda

- Can we trust SMT solvers
- Producing SMT proofs
  - Challenges
  - Our approach
- Leveraging SMT proofs to help proof assistants
- Current and future work

Joint work with:

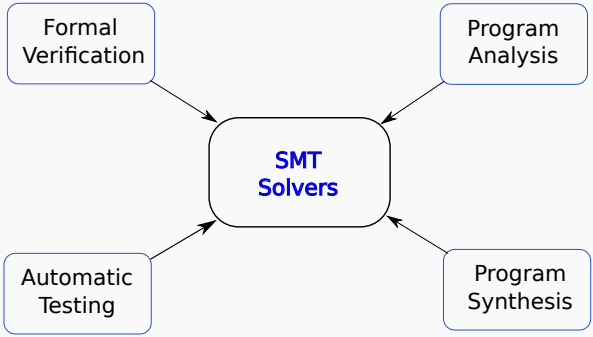
- cvc5 proofs: Haniel Barbosa, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar
- Proof visualization, Alethe proof checking: Haniel Barbosa, Vinícius Braga, Bruno Andreotti
- Isabelle/HOL integration: Leni Aniva, Haniel Barbosa, Mathias Fleury, Hanna Lachnitt
- Lean integration: Haniel Barbosa, Harun Khan, Tomaz Mascarenhas, Abdalrhman Mohamed, Wojciech Nawrocki

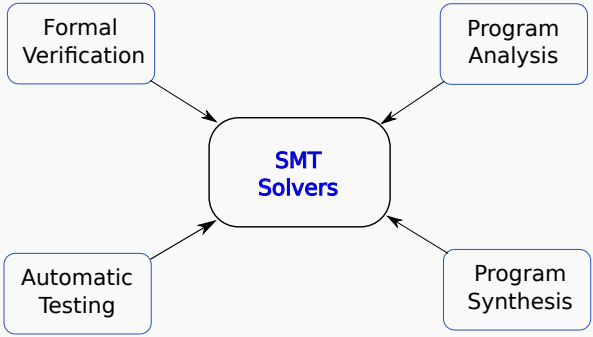
Many thanks to Haniel Barbosa for letting me use his slides.



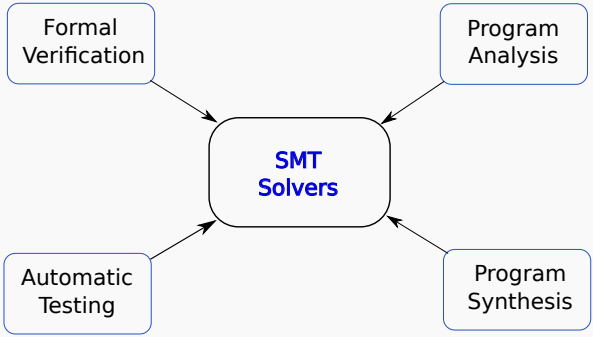
For example, SMT solvers are called from proof assistants

- Any fraction of wrong answers is bad


Formal Verification
Program Analysis
SMT Solvers
Automatic Testing
Program Synthesis



For example, SMT solvers called billions of times a day at AWS in a security critical setting...



For example, SMT solvers called billions of times a day at AWS in a security critical setting...

- Even a tiny fraction of wrong answers is catastrophic

Bugs in SMT Solvers

- State-of-the-art solvers are large projects:
  - cvc5: 300k LoC (C++)
  - z3: 500k LoC (C++)

Bugs in SMT Solvers

- State-of-the-art solvers are large projects:
  - CVC5: 300k LoC (C++)
  - Z3: 500k LoC (C++)

- How do developers try to avoid bugs?
  - Code reviews
  - Testing on benchmark sets
  - Random input testing

Bugs in SMT Solvers

- State-of-the-art solvers are large projects:
  - cvc5: 300k LoC (C++)
  - z3: 500k LoC (C++)
- How do developers try to avoid bugs?
  - Code reviews
  - Testing on benchmark sets
  - Random input testing
- But bugs remain:
  - Every year SMT-COMP has disagreements between solvers
  - Fuzzing tools often find bugs in solvers

Can We Just Certify the Solvers?

- Large, complex code bases are too costly to certify
- A (simpler) certified system can be too slow [**Fleury2018**; **Fleury2019-sat**]
- Certifying/qualifying a system freezes it, potentially blocking improvements
  - Working around adding new features slow and costly [**Burdy2018**]

For your consideration: proofs!

- Proofs are a justification of the logical reasoning the solver has performed to find a solution
- A proof can be checked *independently*
  - Checkers have smaller trusted base
    - LFSC: 5.5k (C++) LoC checker + 2k (LFSC) LoC signatures
    - ALETHE: the CARCARA checker (and elaborator): 12k (Rust) LoC
  - Proof checking is generally more efficiently than solving the problem
- Proofs can be reconstructed within skeptical proof assistants
  - Every logical inference verified by trusted kernel
  - Proof calculus can be embedded in the proof assistant (and proven correct)
  - Proof steps are replayed within the proof assistant
- Confidence in results is decoupled from the solver's implementation

Applications of SMT Proofs

- Strong correctness guarantees
  - High-quality proofs can be used to facilitate automated compliance
- Integrations with other systems
  - Automation in interactive theorem proving
  - External proof cecking can identify bugs in proof rules
- Valuable for debugging
- Formalization of proof rules improves code base
  - Uncovers existing issues
  - Forces modular and clean code design
  - Improves tool robustness
- A rich source of data that can be mined for various purposes (e.g., interpolation, profiling)

SMT solving

- A cooperation of propositional reasoning and theory-specific reasoning
- Employs a SAT solver to perform propositional reasoning
- Employs a combination of procedures for theory reasoning
  - E.g. equality and uninterpreted functions (EUF) (Congruence Closure)
  - E.g. linear integer/real arithmetic (LIA, LRA) (Simplex)
  - Combination of theories (Nelson-Oppen)
- Decidability depends on the theories being used
  - E.g. strings, non-linear integer arithmetic are incomplete
  - Problems involving axioms (user-defined theories) are at best semi-decidable

Boolean Satisfiability (SAT)

Propositional formulas in CNF:

$$\begin{array}{lcl} C & ::= & p \mid \neg p \mid C \vee C \\ \varphi & ::= & C \mid \varphi \wedge \varphi \end{array}$$

Given a formula φ in propositional logic, find an assignment $\mathcal{M}$ mapping every proposition φ to $\{\top, \bot\}$ such that $\mathcal{M}(\varphi) = \top$ (or $\mathcal{M} \models \varphi$).

Boolean Satisfiability (SAT)

Propositional formulas in CNF:

$$\begin{array}{rcl} C & ::= & p \mid \neg p \mid C \vee C \\ \varphi & ::= & C \mid \varphi \wedge \varphi \end{array}$$

Given a formula φ in propositional logic, find an assignment $\mathcal{M}$ mapping every proposition φ to $\{\top, \bot\}$ such that $\mathcal{M}(\varphi) = \top$ (or $\mathcal{M} \models \varphi$).

Example

Is $\varphi = (p \vee \neg q) \wedge (\neg r \vee \neg p) \wedge q$ satisfiable?

Boolean Satisfiability (SAT)

Propositional formulas in CNF:

$$\begin{array}{rcl} C & ::= & p \mid \neg p \mid C \vee C \\ \varphi & ::= & C \mid \varphi \wedge \varphi \end{array}$$

Given a formula φ in propositional logic, find an assignment $\mathcal{M}$ mapping every proposition φ to $\{\top, \bot\}$ such that $\mathcal{M}(\varphi) = \top$ (or $\mathcal{M} \models \varphi$).

Example

Is $\varphi = (p \vee \neg q) \wedge (\neg r \vee \neg p) \wedge q$ satisfiable? **Yes**

$$\mathcal{M}(p) = \top, \mathcal{M}(q) = \top, \mathcal{M}(r) = \bot \quad \Rightarrow \quad \mathcal{M}(\varphi) = \top$$

Boolean Satisfiability (SAT)

Propositional formulas in CNF:

$$\begin{array}{rcl} C & ::= & p \mid \neg p \mid C \vee C \\ \varphi & ::= & C \mid \varphi \wedge \varphi \end{array}$$

Given a formula φ in propositional logic, find an assignment $\mathcal{M}$ mapping every proposition φ to $\{\top, \bot\}$ such that $\mathcal{M}(\varphi) = \top$ (or $\mathcal{M} \models \varphi$).

Example

Is $\varphi = (p \vee \neg q) \wedge (\neg r \vee \neg p) \wedge q \wedge (r \vee \neg q)$ satisfiable?

Boolean Satisfiability (SAT)

Propositional formulas in CNF:

$$\begin{array}{rcl} C & ::= & p \mid \neg p \mid C \vee C \\ \varphi & ::= & C \mid \varphi \wedge \varphi \end{array}$$

Given a formula φ in propositional logic, find an assignment $\mathcal{M}$ mapping every proposition φ to $\{\top, \bot\}$ such that $\mathcal{M}(\varphi) = \top$ (or $\mathcal{M} \models \varphi$).

Example

Is $\varphi = (p \vee \neg q) \wedge (\neg r \vee \neg p) \wedge q \wedge (r \vee \neg q)$ satisfiable? **No**

No combination of valuations for these propositions such that φ is $\top$.

Boolean Satisfiability (SAT)

Unsatisfiability proof of $(p \vee \neg q) \wedge (\neg r \vee \neg p) \wedge q \wedge (r \vee \neg q)$:

$$\dfrac{\dfrac{\dfrac{r \vee \neg q \qquad q}{r}\,\textsc{Res} \qquad \neg r \vee \neg p}{\neg p}\,\textsc{Res} \qquad \dfrac{p \vee \neg q \qquad q}{p}\,\textsc{Res}}{\bot}\,\textsc{Res}$$

Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

$$
\begin{array}{lcl}
t & ::= & x \mid f(t, \ldots, t) \\
\varphi & ::= & p(t, \ldots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x_1 \ldots x_n.\ \varphi
\end{array}
$$

Given a formula φ in FOL and background theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, finding a model $\mathcal{M}$ giving an *interpretation* to all terms and predicates such that $\mathcal{M} \models_{\mathcal{T}_1,\ldots,\mathcal{T}_n} \varphi$

Example

Is φ satisfiable modulo *equality* and *arithmetic*?

Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

$$\begin{array}{rcl} t & ::= & x \mid f(t, \ldots, t) \\ \varphi & ::= & p(t, \ldots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x_1 \ldots x_n.\ \varphi \end{array}$$

Given a formula φ in FOL and background theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, finding a model $\mathcal{M}$ giving an *interpretation* to all terms and predicates such that $\mathcal{M} \models_{\mathcal{T}_1,\ldots,\mathcal{T}_n} \varphi$

Example

Is φ satisfiable modulo *equality* and *arithmetic*?

$$\varphi \quad = \quad (x_1 \geq 0) \wedge (x_1 < 1) \quad \wedge \quad (f(x_1) \not\approx f(0)) \quad \vee \quad x_2 + x_1 > x_2 + 1)$$

Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

$$\begin{array}{rcl} t & ::= & x \mid f(t, \ldots, t) \\ \varphi & ::= & p(t, \ldots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x_1 \ldots x_n.\ \varphi \end{array}$$

Given a formula φ in FOL and background theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, finding a model $\mathcal{M}$ giving an *interpretation* to all terms and predicates such that $\mathcal{M} \models_{\mathcal{T}_1,\ldots,\mathcal{T}_n} \varphi$

Example

Is φ satisfiable modulo *equality* and *arithmetic*?

$$\varphi \quad = \quad \underbrace{(x_1 \geq 0) \wedge (x_1 < 1)}_{\text{LIA}} \quad \wedge \quad \underbrace{(f(x_1) \not\approx f(0)}_{\text{EUF}} \quad \vee \quad \underbrace{x_2 + x_1 > x_2 + 1)}_{\text{LIA}}$$

Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

$$\begin{array}{lcl} t & ::= & x \mid f(t, \ldots, t) \\ \varphi & ::= & p(t, \ldots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x_1 \ldots x_n.\ \varphi \end{array}$$

Given a formula φ in FOL and background theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, finding a model $\mathcal{M}$ giving an *interpretation* to all terms and predicates such that $\mathcal{M} \models_{\mathcal{T}_1,\ldots,\mathcal{T}_n} \varphi$

Example

Is φ satisfiable modulo *equality* and *arithmetic*?

$$\varphi \quad = \quad \underbrace{(x_1 \geq 0) \wedge (x_1 < 1)}_{\text{LIA}} \quad \wedge \quad \underbrace{(f(x_1) \not\simeq f(0)}_{\text{EUF}} \quad \vee \quad \underbrace{x_2 + x_1 > x_2 + 1)}_{\text{LIA}}$$

$$\varphi \quad \models_{\text{LIA}} \quad x_1 \simeq 0$$

Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

$$\begin{array}{rcl} t & ::= & x \mid f(t, \ldots, t) \\ \varphi & ::= & p(t, \ldots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x_1 \ldots x_n.\ \varphi \end{array}$$

Given a formula φ in FOL and background theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, finding a model $\mathcal{M}$ giving an *interpretation* to all terms and predicates such that $\mathcal{M} \models_{\mathcal{T}_1,\ldots,\mathcal{T}_n} \varphi$

Example

Is φ satisfiable modulo *equality* and *arithmetic*?

$$\varphi \quad = \quad \underbrace{(x_1 \geq 0) \wedge (x_1 < 1)}_{\text{LIA}} \quad \wedge \quad (\underbrace{f(x_1) \not\simeq f(0)}_{\text{EUF}} \quad \vee \quad \underbrace{x_2 + x_1 > x_2 + 1)}_{\text{LIA}}$$

$$\begin{array}{rcl} \varphi & \models_{\text{LIA}} & x_1 \simeq 0 \\ x_1 \simeq 0 & \models_{\text{EUF}} & f(x_1) \simeq f(0) \end{array}$$

Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

$$\begin{array}{rcl} t & ::= & x \mid f(t, \ldots, t) \\ \varphi & ::= & p(t, \ldots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x_1 \ldots x_n.\ \varphi \end{array}$$

Given a formula φ in FOL and background theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, finding a model $\mathcal{M}$ giving an *interpretation* to all terms and predicates such that $\mathcal{M} \models_{\mathcal{T}_1,\ldots,\mathcal{T}_n} \varphi$

Example

Is φ satisfiable modulo *equality* and *arithmetic*?

$$\varphi \quad = \quad \underbrace{(x_1 \geq 0) \wedge (x_1 < 1)}_{\text{LIA}} \quad \wedge \quad (\underbrace{f(x_1) \not\simeq f(0)}_{\text{EUF}} \quad \vee \quad \underbrace{x_2 + x_1 > x_2 + 1}_{\text{LIA}})$$

$$\begin{array}{rcl} \varphi & \models_{\text{LIA}} & x_1 \simeq 0 \\ x_1 \simeq 0 & \models_{\text{EUF}} & f(x_1) \simeq f(0) \\ x_1 \simeq 0 & \models_{\text{LIA}} & x_2 + x_1 \not> x_2 + 1 \end{array}$$

Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

$$\begin{array}{rcl} t & ::= & x \mid f(t, \ldots, t) \\ \varphi & ::= & p(t, \ldots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x_1 \ldots x_n.\ \varphi \end{array}$$

Given a formula φ in FOL and background theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, finding a model $\mathcal{M}$ giving an *interpretation* to all terms and predicates such that $\mathcal{M} \models_{\mathcal{T}_1,\ldots,\mathcal{T}_n} \varphi$

Example

Is φ satisfiable modulo *equality* and *arithmetic*?

$$\varphi \quad = \quad \underbrace{(x_1 \geq 0) \wedge (x_1 < 1)}_{\text{LIA}} \quad \wedge \quad (\underbrace{f(x_1) \not\simeq f(0)}_{\text{EUF}} \quad \vee \quad \underbrace{x_2 + x_1 > x_2 + 1}_{\text{LIA}})$$

$$\begin{array}{rcl} \varphi & \models_{\text{LIA}} & x_1 \simeq 0 \\ x_1 \simeq 0 & \models_{\text{EUF}} & f(x_1) \simeq f(0) \\ x_1 \simeq 0 & \models_{\text{LIA}} & x_2 + x_1 \not> x_2 + 1 \end{array}$$

Therefore $\models_{\text{EUF} \cup \text{LIA}} \neg\varphi$

Satisfiability Modulo Theories (SMT)

Unsatisfiability proof of $(x_1 \geq 0) \wedge (x_1 < 1) \wedge (f(x_1) \not\simeq f(0) \vee x_2 + x_1 > x_2 + 1)$:

Let Π_1:

$$\frac{x_1 \geq 0 \qquad x_1 < 1}{x_1 \simeq 0}\ \text{LIA}$$

Then the final proof is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_1}{x_1 \simeq 0}}{f(x_1) \simeq f(0))}\ \text{EUF} \qquad f(x_1) \not\simeq f(0) \vee x_2 + x_1 > x_2 + 1}{x_2 + x_1 > x_2 + 1}\ \text{Res} \qquad \dfrac{\dfrac{\Pi_1}{x_1 \simeq 0}}{x_2 + x_1 \not> x_2 + 1}\ \text{LIA}}{\bot}\ \text{Res}$$

Challenges for SMT proofs

- Collecting and storing proofs efficiently

  Many attempts, but not there yet

  [**Sutcliffe2004**; **Katz2016**; **Hadarean2015**; **Moskal2008-rocket**; **deMoura2008-proofs**; **Schulz2013-e**; **Kovacs2013**; **Weidenbach2009**; **Bouton2009**]
- Proofs for sophisticated preprocessing and rewriting techniques

  Initial progress but many challenges remain

  [**Barbosa2020**]
- Proofs for complex procedures in theory solvers (e.g., CAD, strings)

  Open
- Standardizing a proof format

  Open
- Scalable, trustworthy checking

  Many attempts, but not there yet

  [**Blanchette2013**; **Stump2013**; **Ekici2017**; **Barbosa2020**; **Schurr2021**]

The Journey

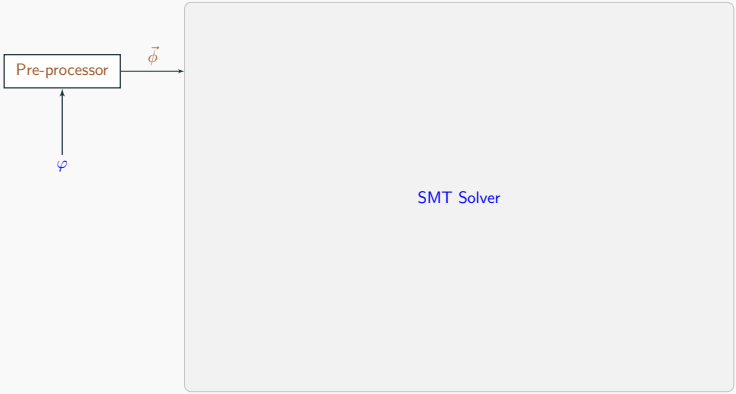
- CVC4's old proof module struggled with many of those challenges
- For two years, we reimplemented its proof module *from scratch*
  - Producing proofs should not significantly change the behavior of the solver
    - Incorporate (almost) all relevant optimizations
    - Coarse-grained steps for non-supported inferences
  - Modular infrastructure allowing fine-grained error localization
    - Independent proof components, combined in a trusted manner
    - Every rule associated with an internal proof checker
  - Custom eager/lazy generation of proofs
    - Proof reconstruction (elaboration) via internal post-processing
  - Support internal proof format and conversions to different proof formats
    - LFSC, Alethe, Lean

Proof module architecture for CDCL($\mathcal{T}$)

Proof module architecture for CDCL($\mathcal{T}$)

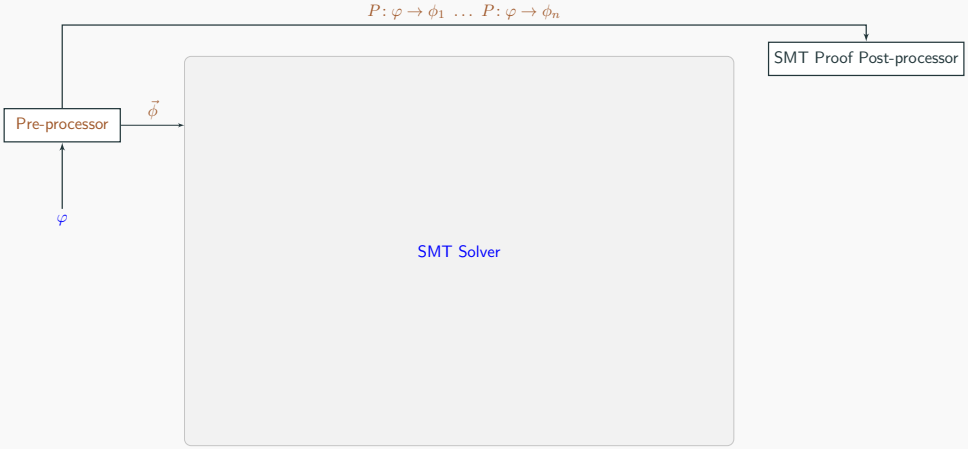
Proof module architecture for CDCL($\mathcal{T}$)

Proof module architecture for CDCL($\mathcal{T}$)



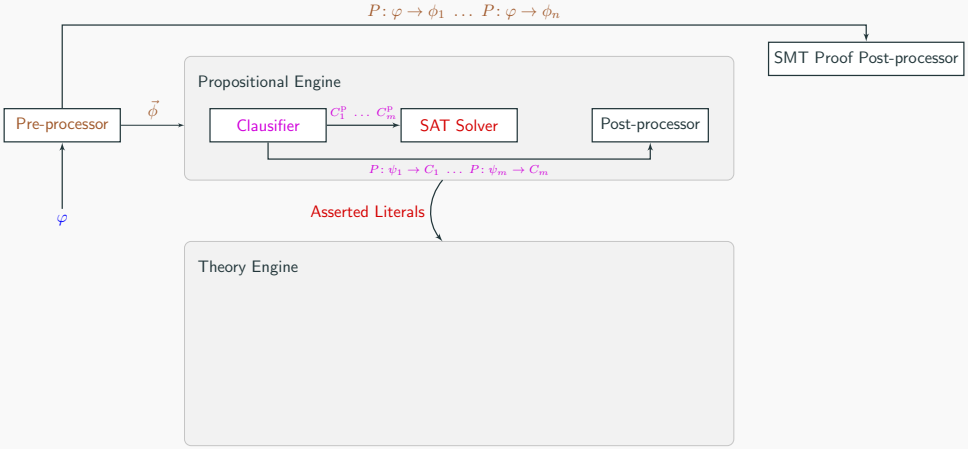
- Preprocessor simplifies formula globally:
  $x \simeq t \wedge F[x] \;\mapsto\; F[t] \qquad F[(ite\ P\ t_1\ t_2)] \;\mapsto\; F[t'] \wedge P \to t' \simeq t_1 \wedge \neg P \to t' \simeq t_2$

Proof module architecture for CDCL($\mathcal{T}$)



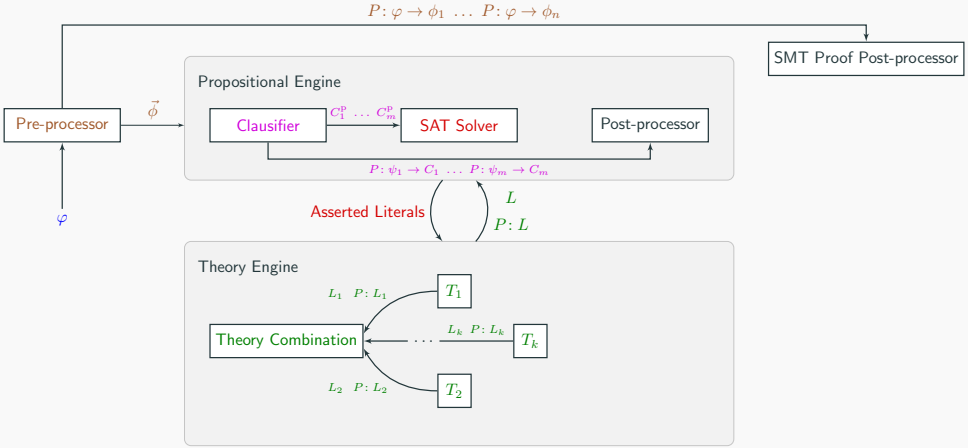
- Preprocessor simplifies formula globally:
  $x \simeq t \wedge F[x] \ \mapsto \ F[t] \qquad F[(ite\ P\ t_1\ t_2)] \ \mapsto \ F[t'] \wedge P \to t' \simeq t_1 \wedge \neg P \to t' \simeq t_2$

Proof module architecture for CDCL($\mathcal{T}$)



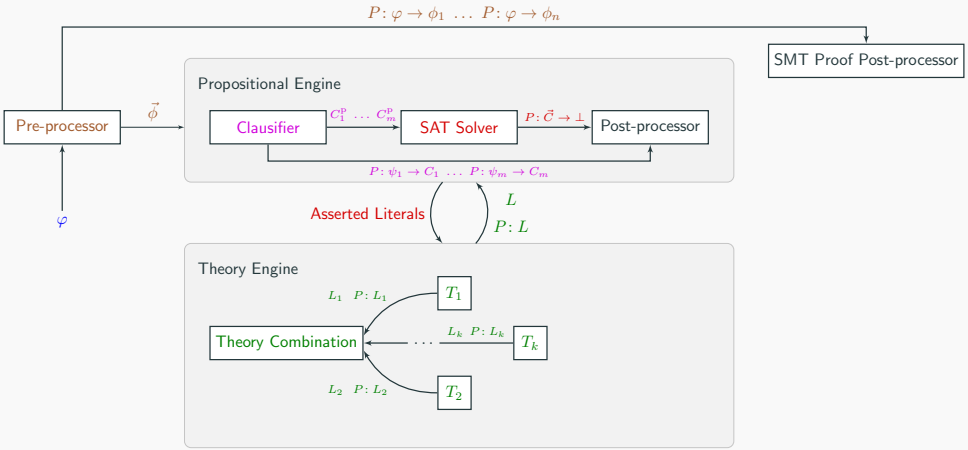
- Clausifier converts to Conjunctive Normal Form (CNF)
  SAT solver asserts literals that must hold based on Boolean abstraction

Proof module architecture for CDCL($\mathcal{T}$)



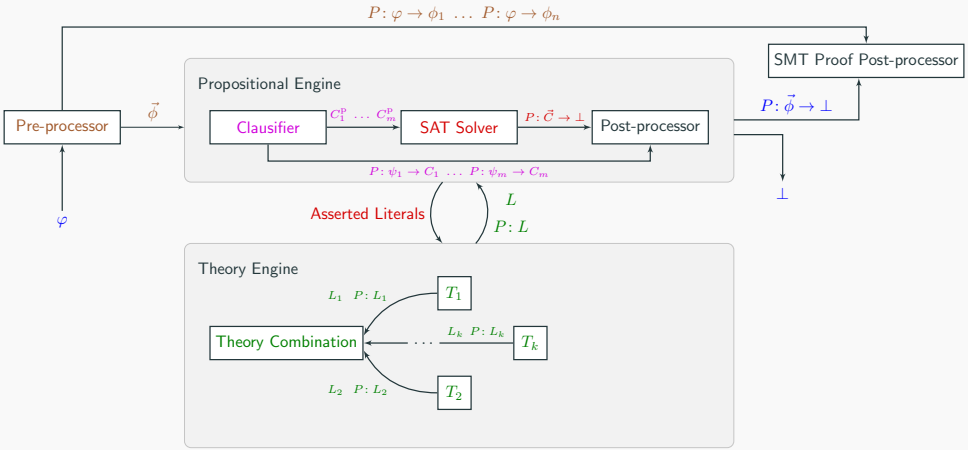
- Theory solvers check consistency in the theory

Proof module architecture for CDCL($\mathcal{T}$)



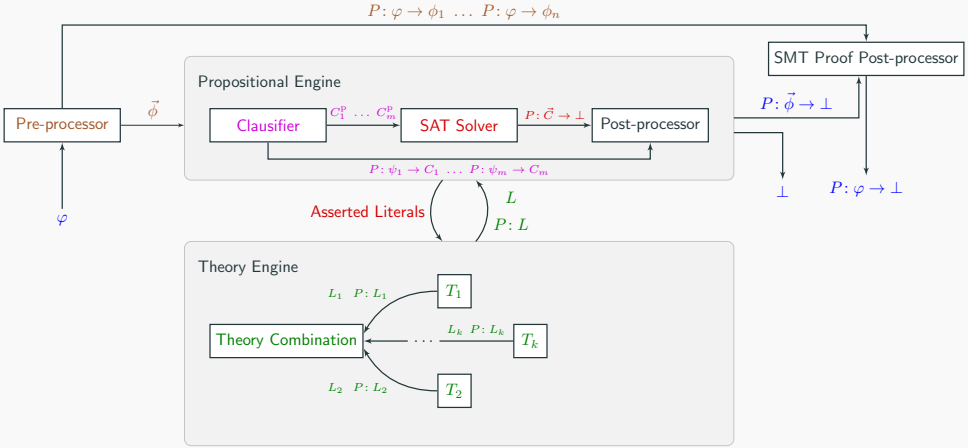
- Theory solvers check consistency in the theory

Proof module architecture for CDCL($\mathcal{T}$)



- Theory solvers check consistency in the theory

Proof module architecture for CDCL($\mathcal{T}$)



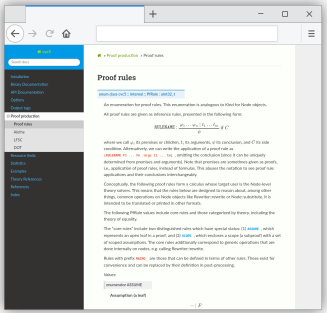
- Theory solvers check consistency in the theory

Main components: Internal proof calculus

- Rules for equality reasoning (congruence closure)
- Rules for rewriting, substitution
  - Coarse-grained rules for capturing multiple core utilities
- Rules for witness forms
  - Enable introduction and correct handling of new symbols
- Rules for scoped reasoning
  - Enable local reasoning, via assumptions and $\Rightarrow$-introduction
- Theory-specific rules
  - Boolean (clausification, resolution, ...)
  - Arithmetic (linear, non-linear, integer, rationals, transcendentals)
  - Arrays, Datatypes, Bit-vectors, Quantifiers, ...

Main components: Internal proof calculus

- Rules for equality reasoning (congruence closure)
- Rules for rewriting, substitution
  - Coarse-grained rules for capturing multiple core utilities
- Rules for witness forms
  - Enable introduction and correct handling of new symbols
- Rules for scoped reasoning
  - Enable local reasoning, via assumptions and $\Rightarrow$-introduction
- Theory-specific rules
  - Boolean (clausification, resolution, ...)
  - Arithmetic (linear, non-linear, integer, rationals, transcendentals)
  - Arrays, Datatypes, Bit-vectors, Quantifiers, ...

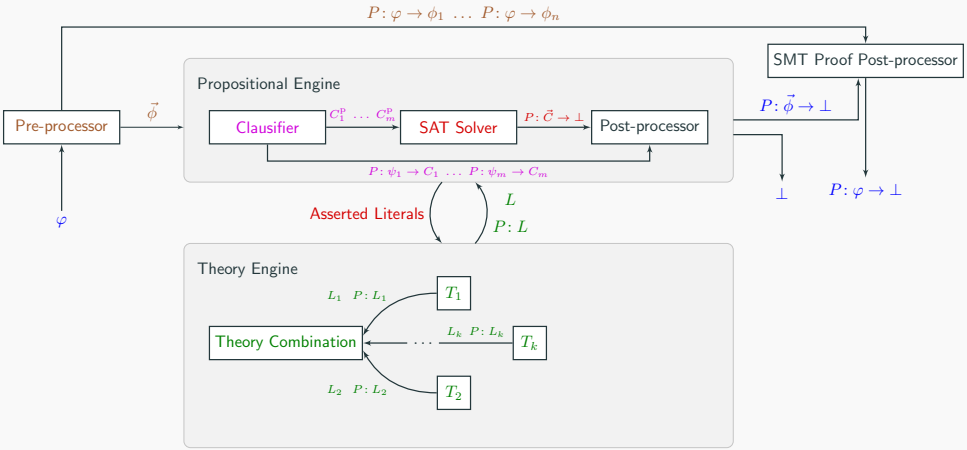


Documentation: `https://cvc5.github.io/docs/latest/proofs/proof_rules.html`

Main components: Library of proof generators

- Encapsulate common patterns for building proofs
- Solving components store information during solving
- Derived facts are distributed with associated proof generators
- When proof generator is requested for fact φ, its internal information is used to produce the proof $P : \varphi$.

Proof module architecture



- Actually, *proof generators* are transmitted between components
- Only during post-processing are proofs fully computed

Proof generation for substitution and rewriting

- Substitution and rewriting inferences recorded without further details
- No need to instrument utilities to track how terms are converted
  - Only macro steps and used rewrites rules are stored in generators

$$\frac{a \simeq 0 \qquad b \simeq 1}{(a > b \wedge F) \simeq \bot}\ \mathsf{SR}$$

Proof generation for substitution and rewriting

- Substitution and rewriting inferences recorded without further details
- No need to instrument utilities to track how terms are converted
  - Only macro steps and used rewrites rules are stored in generators

$$\frac{a \simeq 0 \qquad b \simeq 1}{(a > b \wedge F) \simeq \bot}\ \text{SR}$$

$$\frac{\dfrac{a \simeq 0 \qquad b \simeq 1}{(a > b \wedge F) \simeq (0 > 1 \wedge F)}\ \text{SUBS} \qquad \dfrac{}{(0 > 1 \wedge F) \simeq \bot}\ \text{RW}}{(a > b \wedge F) \simeq \bot}$$

Proof generation for substitution and rewriting

- Substitution and rewriting inferences recorded without further details
- No need to instrument utilities to track how terms are converted
  - Only macro steps and used rewrites rules are stored in generators

$$\frac{a \simeq 0 \qquad b \simeq 1}{(a > b \wedge F) \simeq \bot}\ \text{SR}$$

$$\dfrac{\dfrac{a \simeq 0 \qquad b \simeq 1}{(a > b \wedge F) \simeq (0 > 1 \wedge F)}\ \text{SUBS} \qquad \dfrac{}{(0 > 1 \wedge F) \simeq \bot}\ \text{RW}}{(a > b \wedge F) \simeq \bot}$$

$$\dfrac{\dfrac{\dfrac{}{0 > 1 \simeq \bot}\ \text{arith\_rw} \qquad \dfrac{}{F \simeq F}\ \text{refl}}{(0 > 1 \wedge F) \simeq (\bot \wedge F)}\ \text{cong} \qquad \dfrac{}{(\bot \wedge F) \simeq \bot}\ \text{bool\_rw}}{(0 > 1 \wedge F) \simeq \bot}\ \text{trans}$$

Proof generation for substitution and rewriting

- Substitution and rewriting inferences recorded without further details
- No need to instrument utilities to track how terms are converted
  - Only macro steps and used rewrites rules are stored in generators

$$\frac{a \simeq 0 \qquad b \simeq 1}{(a > b \wedge F) \simeq \bot}\ \text{SR}$$

$$\dfrac{\dfrac{a \simeq 0 \qquad b \simeq 1}{(a > b \wedge F) \simeq (0 > 1 \wedge F)}\ \text{SUBS} \qquad \dfrac{}{(0 > 1 \wedge F) \simeq \bot}\ \text{RW}}{(a > b \wedge F) \simeq \bot}$$

$$\dfrac{\dfrac{\dfrac{}{0 > 1 \simeq \bot}\ \text{arith\_rw} \qquad \dfrac{}{F \simeq F}\ \text{refl}}{(0 > 1 \wedge F) \simeq (\bot \wedge F)}\ \text{cong} \qquad \dfrac{}{(\bot \wedge F) \simeq \bot}\ \text{bool\_rw}}{(0 > 1 \wedge F) \simeq \bot}\ \text{trans}$$

- Heavily used for strings, preprocessing, bitblasting, and so on.

Demo!

Consider the following unsatisfiable SMT problem:

$$x \simeq y \wedge f(x) \not\simeq f(y)$$

which in SMT-LIB is:

```
(set-logic QF_UFLIA)

(declare-const x Int)
(declare-const y Int)

(declare-fun f (Int) Int)

(assert (= x y))
(assert (not (= (f x) (f y))))

(check-sat)
```

Integration with proof assistants

- Detailed proofs help *interoperability* with proof assistants

- The steps to discharge proof goals using SMT solvers:
  1. Encode proof goal as an SMT-LIB problem
  2. Solve the problem, produce proof
  3. Convert SMT proof into the proof assistant's format to prove original goal

- The last step can be performed in two ways:
  1. *certified*: bridge theorem between formats
  2. *certifying*: ad-hoc conversion between proofs

Lean-SMT

- A work-in-progress `smt` tactic to discharge Lean proof goals
- We apply a certifying approach to convert SMT proofs into Lean proofs
- Our goal is to (eventually...) reproduce the success of similar approaches in other proof assistants
  - `SMTCoq` in the Coq proof assistant
  - `Sledgehammer` in the Isabelle/HOL proof assistant
    - "The difference between walking and running" – Larry Paulson

Other current/future work

- Detailed proofs for challenging theories, such as non-linear arithmetic
- Extending Sledgehammer with bit-vectors
  - Mechanizing bit-blasting and bit-vector rewrites
  - Extending the reconstruction tactic
- Handling highly custom theory rewrites via a dedicated DSL
  - Automatic translation of rewrites specification into proof assistants
- Parallel proof checking
  - SMT proofs are highly amenable for parallel proof checking
  - Work-in-progress in this direction in the CARCARA proof checker for Alethe
- Development of a proof library
- ...

Proof Certificates in Satisfiability Modulo Theories

Clark Barrett, Stanford University