Better SMT proofs for certifying compliance and correctness

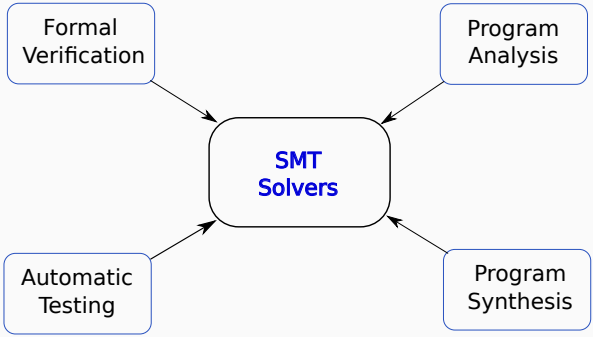
Haniel Barbosa, Universidade Federal de Minas Gerais

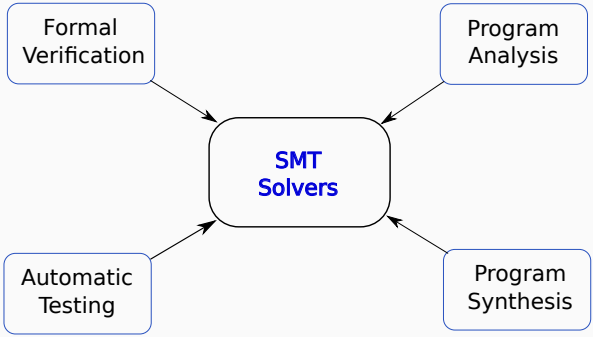
Agenda

- SMT solvers matter. Moreover we'd like to trust them.
- Producing SMT proofs
  - Challenges
  - Our approach
- Leveraging SMT proofs to help proof assistants
- Current and future work

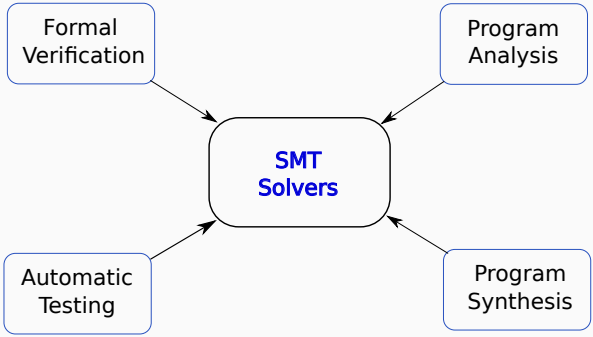
Joint work with:

- cvc5 proofs: Clark Barrett, 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: Vinícius Braga, Bruno Andreotti
- Isabelle/HOL integration: Hanna Lachnitt
- Lean integration: Tomaz Mascarenhas, Abdalrhman Mohamed, Wojciech Nawrocki


Formal Verification
Program Analysis
SMT Solvers
Automatic Testing
Program Synthesis

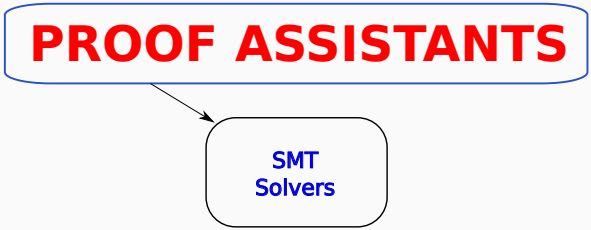


For example, SMT solvers called billions of times a day at AWS in a security critical setting...



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 bad



For example, SMT solvers called several times a day from proof assistants

- Any 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 [FBL18; Fle19]
- Certifying/qualifying a system freezes it, potentially blocking improvements
  - Working around adding new features slow and costly [BD18]

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, finding 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, finding 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:

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

Given a formula φ in propositional logic, finding 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, finding 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:

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

Given a formula φ in propositional logic, finding 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:

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

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*?

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

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:

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

Challenges for SMT proofs

- Collecting and storing proofs efficiently

  Many attempts, no silver bullet

  [SZS04; KBT+16; HBR+15; Mos08; MB08; Sch13; KV13; WDF+09; BODF09]
- Proofs for sophisticated preprocessing and rewriting techniques

  Initial progress but many challenges remain

  [BBFF20]
- Proofs for complex procedures in theory solvers (e.g., CAD, strings)

  Open
- Standardizing a proof format

  Open
- Scalable, trustworthy checking

  Many attempts, no silver bullet

  [BBP13; SOR+13; EMT+17; BBFF20; SFD21]

The Journey

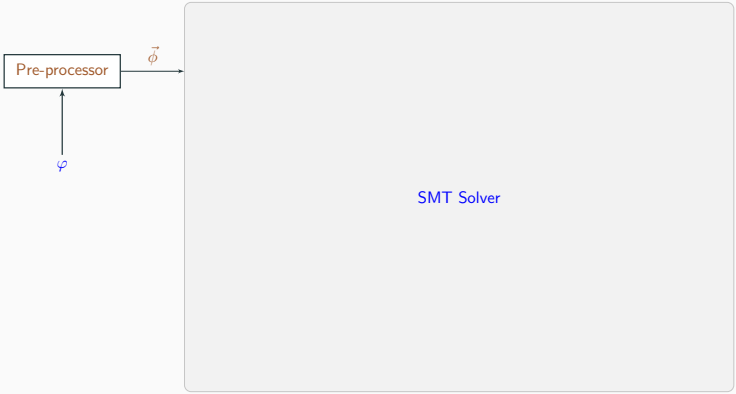
- CVC4's old proof module struggled with many of those challenges
- For two years, we reimplemented its proof module *from scratch*
  - Producing proofs should not significantly change the behavior of the solver
    - Incorporate (almost) all relevant optimizations
    - Coarse-grained steps for non-supported 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($\mathcal{T}$)

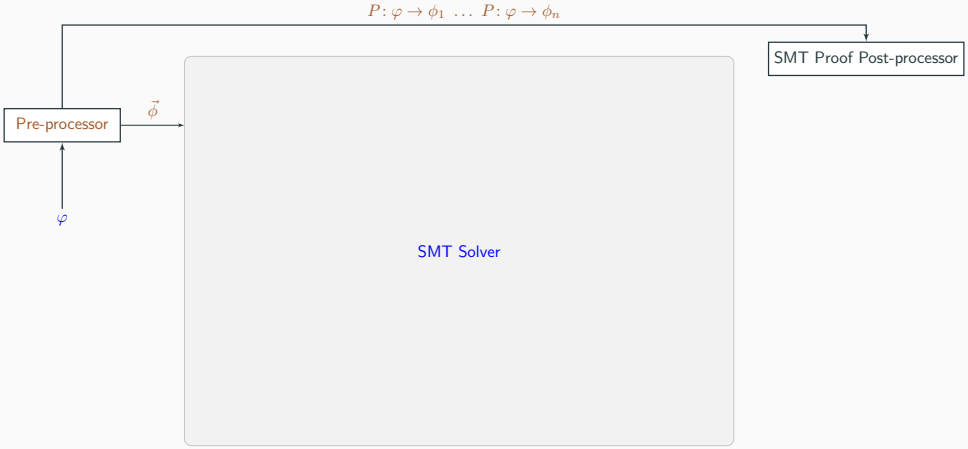
Proof module architecture for CDCL($\mathcal{T}$)

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



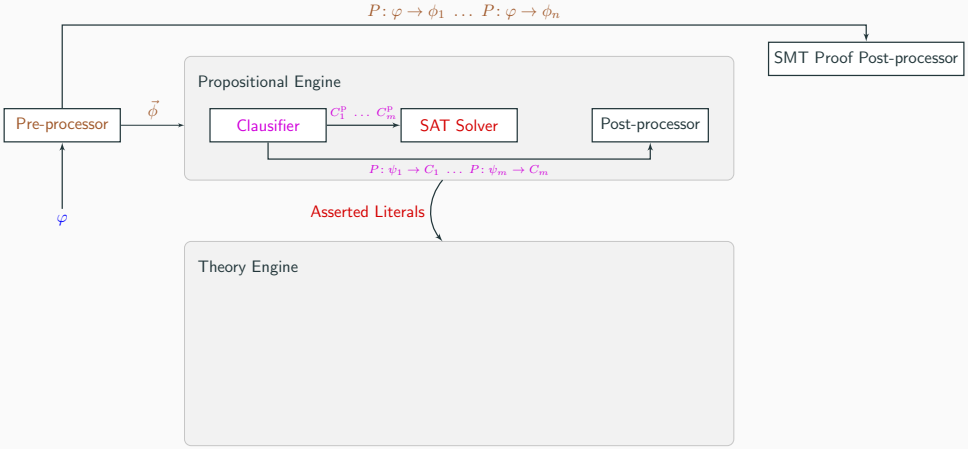
- Preprocessor simplifies formula globally:
  $x \simeq t \wedge F[x] \;\mapsto\; F[t] \qquad F[(ite\ P\ t_1\ t_2)] \;\mapsto\; F[t'] \wedge P \to t' \simeq t_1 \wedge \neg P \to t' \simeq t_2$

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



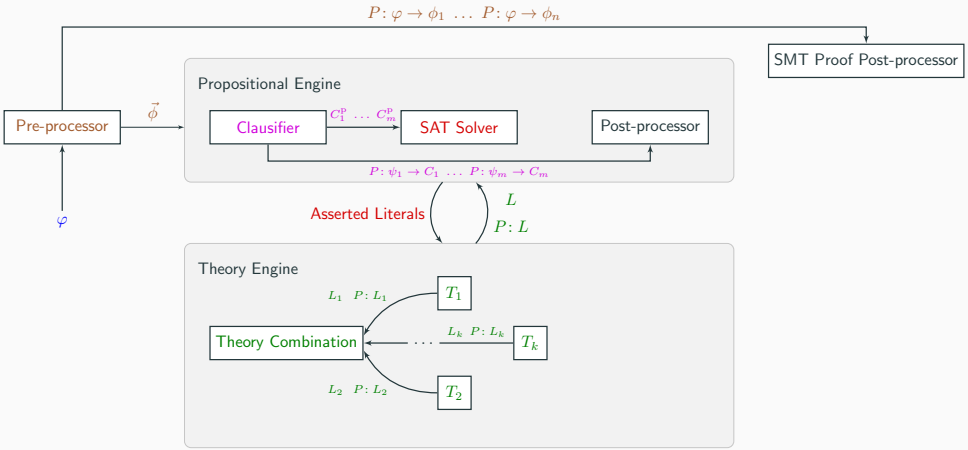
- Preprocessor simplifies formula globally:
  $x \simeq t \wedge F[x] \ \mapsto \ F[t] \qquad F[(ite\ P\ t_1\ t_2)] \ \mapsto \ F[t'] \wedge P \to t' \simeq t_1 \wedge \neg P \to t' \simeq t_2$

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



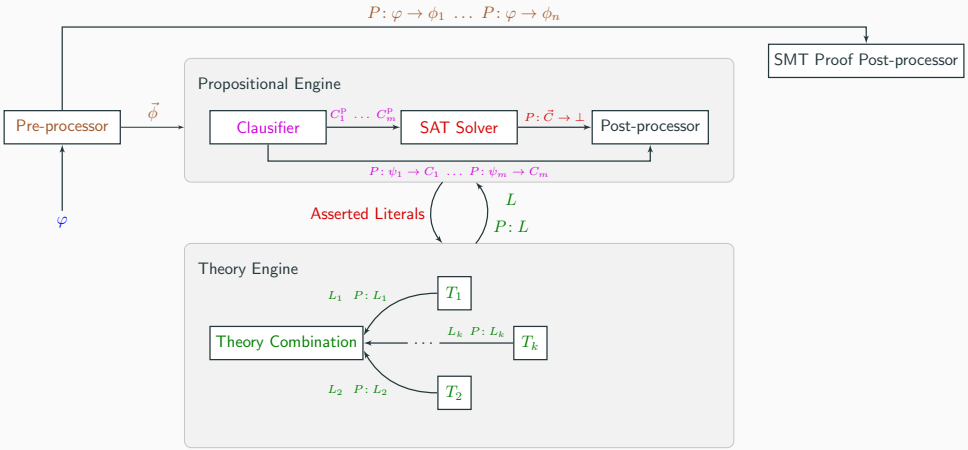
- Clausifier converts to Conjunctive Normal Form (CNF)
  SAT solver asserts literals that must hold based on Boolean abstraction

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



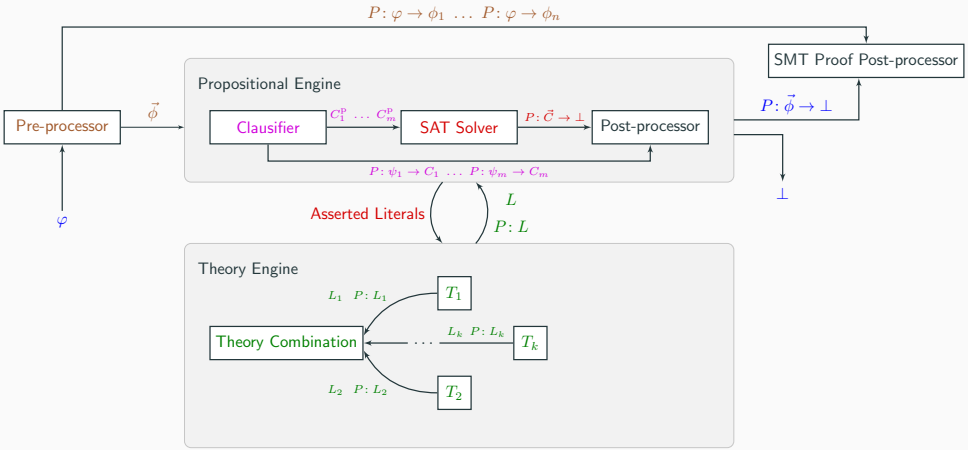
- Theory solvers check consistency in the theory

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



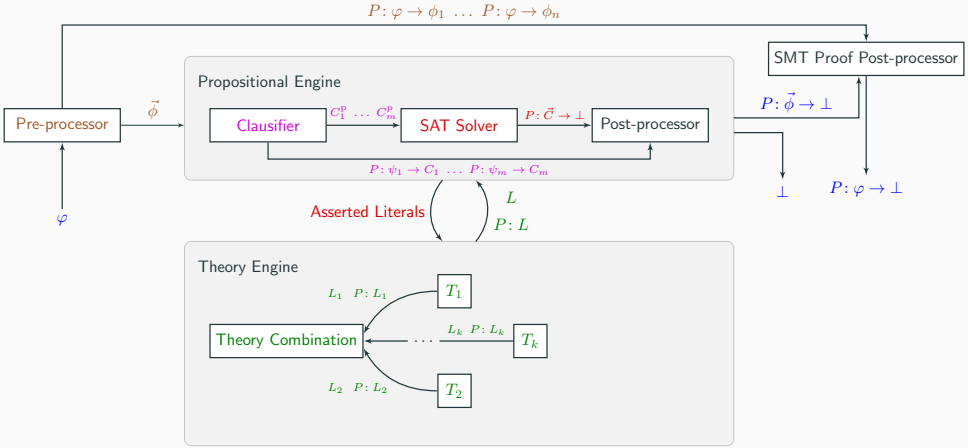
- Theory solvers check consistency in the theory

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



- Theory solvers check consistency in the theory

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



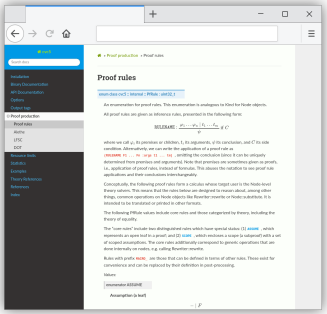
- Theory solvers check consistency in the theory

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, ...

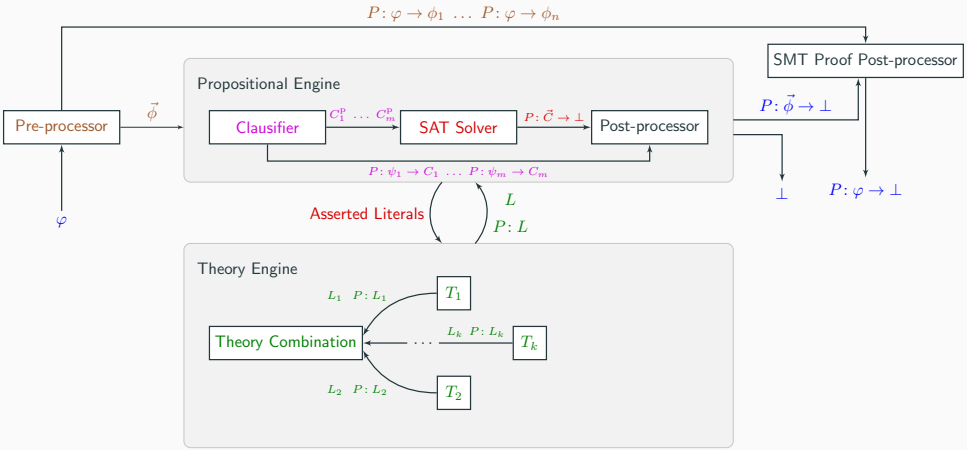


Documentation: `https://cvc5.github.io/docs/latest/proofs/proof_rules.html`

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 at the post-processors are proofs requested (and 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

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

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

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:

$$a \simeq b \wedge c \simeq d \wedge (p_1 \wedge \top) \wedge ((\neg p_1) \vee (p_2 \wedge p_3)) \wedge (\neg p_3 \vee (f(a,\, c) \not\simeq f(b,\, d)))$$

which in SMT-LIB is:

```
(set-logic QF_UF)
(declare-sort U 0)
(declare-const p1 Bool)
(declare-const p2 Bool)
(declare-const p3 Bool)
(declare-const a U)
(declare-const b U)
(declare-const c U)
(declare-const d U)
(declare-fun f (U U) U)

(assert (= a b))
(assert (= c d))
(assert (and p1 true))
(assert (or (not p1) (and p2 p3)))
(assert (or (not p3) (not (= (f a c) (f b d)))))
(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

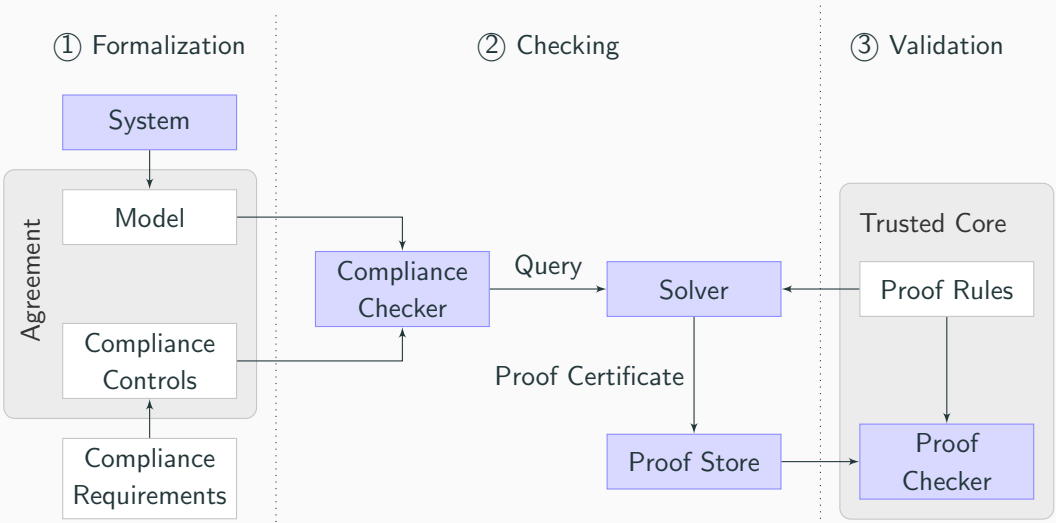
Other current/future work

- 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
- ...

Better SMT proofs for certifying compliance and correctness

Haniel Barbosa, Universidade Federal de Minas Gerais

Applications of SMT Proofs: Compliance



Anecdotes

- Internal proof checker is highly valuable for development
- Error localization for proofs is important
- Formalization of proof rules uncovers existing issues
- Performance issues
  - In a few cases, proof checker indicated it could prove something stronger
- Soundness issues
  - Cannot write proper proof checker if the reasoning of the solver is wrong
- Proofs are also valuable for debugging
  - Soundness bug reported, proofs used to easily isolate the incorrect rewrite
- Combination of approaches for proof generation

Conclusion

- Proofs are integral for the trustworthiness SMT solvers (and have other applications)
- Fine-grained proofs are now available for most of CVC5's reasoning
  - Combination of instrumentation and reconstruction
  - Strings and simplification under global assumptions were special milestones
- Multiple proof formats are supported
  - Integration into multiple proof checkers are ongoing
  - Formalization of new calculi in Lean, LFSC, Isabelle/HOL
  - DOT format and web-based proof visualizer

How some proofs look like

$$\frac{A \vee \ell \qquad B \vee \overline{\ell}}{A \vee B}$$

$$\frac{\varphi_1 \wedge \cdots \wedge \varphi_n}{\varphi_i}$$

$$\neg(a \simeq b) \vee f(a) \simeq f(b)$$
$$\neg(y > 1) \vee \neg(x < 1) \vee y > x$$

$$\neg(\varphi_1 \wedge \cdots \wedge \varphi_n) \vee \varphi_i$$

A particular challenge has been String solving

- Preprocessing
- Clausification
- SAT solving
- UF theory solver
- Linear Arithmetic solver
- Theory combination
- Quantifier instantiation
- Rewriting
  - Including complex string methods [RNBT19]
- Strings theory solver
  - Core calculus [LRT+14]
  - Extended function reductions [RWB+17]
  - Regular expression unfolding

References

Haniel Barbosa, Jasmin Christian Blanchette, Mathias Fleury, et al. "Scalable Fine-Grained Proofs for Formula Processing". In: Journal of Automated Reasoning 64.3 (2020), pp. 485–510.

Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. "Extending Sledgehammer with SMT Solvers". In: Journal of Automated Reasoning 51.1 (2013), pp. 109–128.

Lilian Burdy and David Déharbe. "Teaching an Old Dog New Tricks - The Drudges of the Interactive Prover in Atelier B". In:
Abstract State Machines, Alloy, B, TLA, VDM, and Z - 6th International Conference, ABZ 2018, Southampton, UK, June 5-8, 2
Ed. by Michael J. Butler, Alexander Raschke, Thai Son Hoang, et al. Vol. 10817. Lecture Notes in Computer Science. Springer, 2018, pp. 415–419.

Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, et al. "veriT: An Open, Trustable and Efficient SMT-Solver". In: Proc. Conference on Automated Deduction (CADE). Ed. by Renate A. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer, 2009, pp. 151–156.

References ii

Burak Ekici, Alain Mebsout, Cesare Tinelli, et al. “SMTCoq: A Plug-In for Integrating SMT Solvers into Coq”. In: Computer Aided Verification (CAV). Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10427. Lecture Notes in Computer Science. Springer, 2017, pp. 126–133.

Mathias Fleury, Jasmin Christian Blanchette, and Peter Lammich. “A verified SAT solver with watched literals using imperative HOL”. In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, Ed. by June Andronick and Amy P. Felty. ACM, 2018, pp. 158–171.

Mathias Fleury. “Optimizing a Verified SAT Solver”. In: NASA Formal Methods - 11th International Symposium, NFM 2019, Houston, TX, USA, May 7-9, 2019, Proceedings. Ed. by Julia M. Badger and Kristin Yvonne Rozier. Vol. 11460. Lecture Notes in Computer Science. Springer, 2019, pp. 148–165.

Liana Hadarean, Clark W. Barrett, Andrew Reynolds, et al. “Fine Grained SMT Proofs for the Theory of Fixed-Width Bit-Vectors”. In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Martin Davis, Ansgar Fehnker, Annabelle McIver, et al. Vol. 9450. Lecture Notes in Computer Science. Springer, 2015, pp. 340–355.

Guy Katz, Clark W. Barrett, Cesare Tinelli, et al. “Lazy proofs for DPLL(T)-based SMT solvers”. In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Ruzica Piskac and Muralidhar Talupur. IEEE, 2016, pp. 93–100.

References iii

Laura Kovács and Andrei Voronkov. "First-Order Theorem Proving and Vampire". English. In: Computer Aided Verification (CAV). Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 1–35.

Tianyi Liang, Andrew Reynolds, Cesare Tinelli, et al. "A DPLL(T) Theory Solver for a Theory of Strings and Regular Expressions". In: Computer Aided Verification (CAV). Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 646–662.

Leonardo Mendonça de Moura and Nikolaj Bjørner. "Proofs and Refutations, and Z3". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) Workshops. Ed. by Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, et al. Vol. 418. CEUR Workshop Proceedings. CEUR-WS.org, 2008.

Michał Moskal. "Rocket-Fast Proof Checking for SMT Solvers". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS). Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 486–500.

Andrew Reynolds, Andres Nötzli, Clark W. Barrett, et al. "High-Level Abstractions for Simplifying Extended String Constraints in SMT". In: Computer Aided Verification (CAV), Part II. Ed. by Isil Dillig and Serdar Tasiran. Vol. 11562. Lecture Notes in Computer Science. Springer, 2019, pp. 23–42.

References iv

Andrew Reynolds, Maverick Woo, Clark Barrett, et al. "Scaling Up DPLL(T) String Solvers Using Context-Dependent Simplification". In: Computer Aided Verification (CAV). Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10427. Lecture Notes in Computer Science. Springer, 2017, pp. 453–474.

Stephan Schulz. "System Description: E 1.8". English. In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Ken McMillan, Aart Middeldorp, and Andrei Voronkov. Vol. 8312. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 735–743.

Hans-Jörg Schurr, Mathias Fleury, and Martin Desharnais. "Reliable Reconstruction of Fine-grained Proofs in a Proof Assistant". In: Proc. Conference on Automated Deduction (CADE). Ed. by André Platzer and Geoff Sutcliffe. Vol. 12699. Lecture Notes in Computer Science. Springer, 2021, pp. 450–467.

Aaron Stump, Duckki Oe, Andrew Reynolds, et al. "SMT proof checking using a logical framework". In: Formal Methods in System Design 42.1 (2013), pp. 91–118.

Geoff Sutcliffe, Jürgen Zimmer, and Stephan Schulz. "TSTP Data-Exchange Formats for Automated Theorem Proving Tools". In: Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems. Ed. by Weixiong Zhang and Volker Sorge. Vol. 112. Frontiers in Artificial Intelligence and Applications. IOS Press, 2004, pp. 201–215.

References v

Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, et al. “SPASS Version 3.5”. English. In: Proc. Conference on Automated Deduction (CADE). Ed. by RenateA. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 140–145.