Better SMT proofs for certifying compliance

(or what a bunch of people achieve in 1.5 years writing from scratch a proof module for an state-of-the-art SMT solver)

Haniel Barbosa

FACC 2021

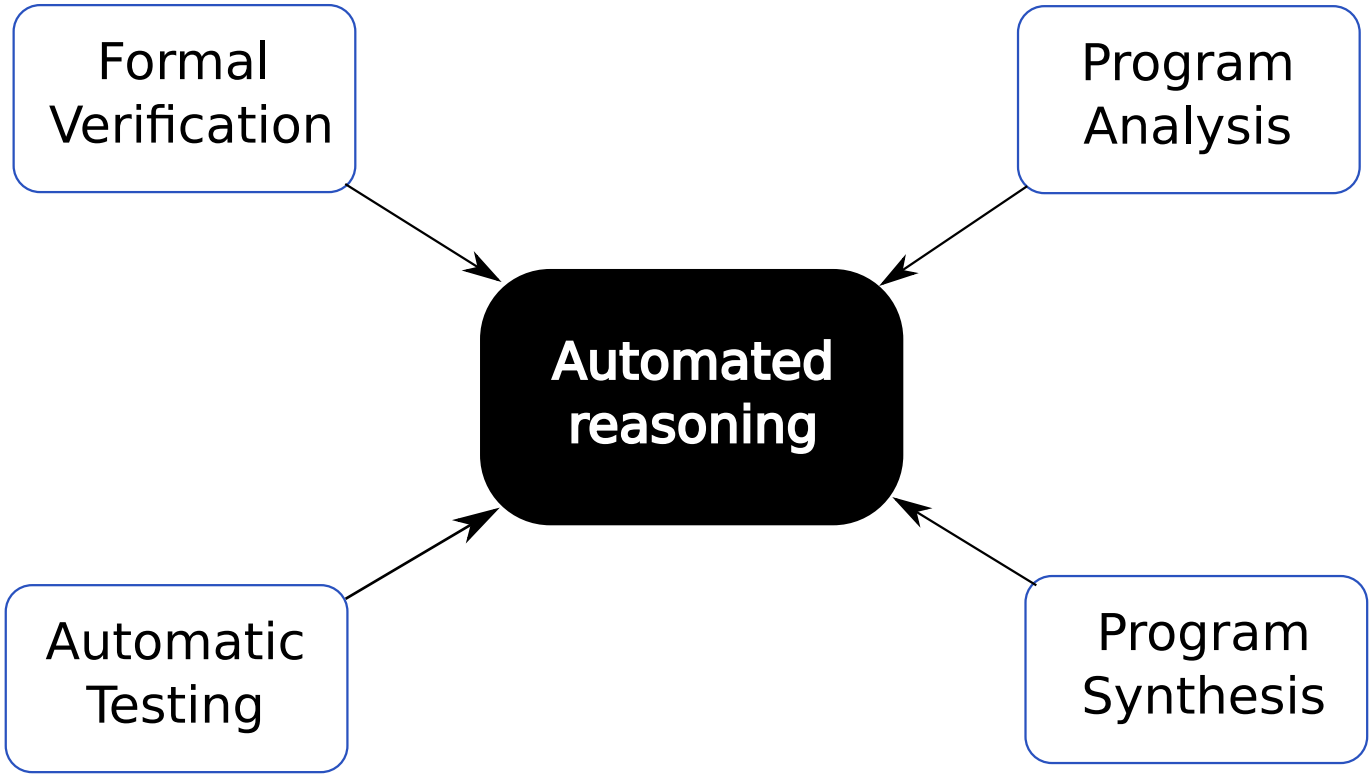
2021–07–19, The Internet

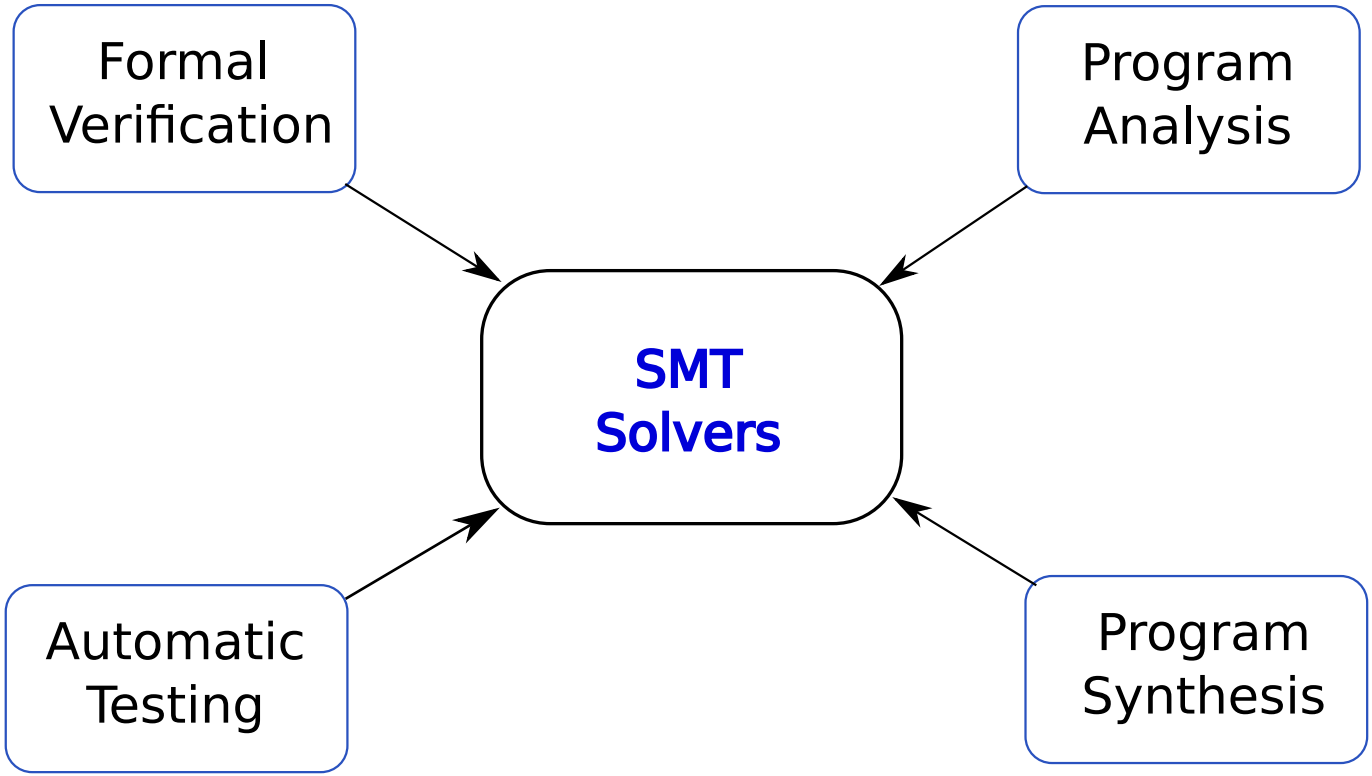
Formal Verification

Program Analysis

Automatic Testing

Program Synthesis


Formal Verification
Program Analysis
Automated reasoning
Automatic Testing
Program Synthesis


Formal
Verification
Program
Analysis
SMT
Solvers
Automatic
Testing
Program
Synthesis

SMT solvers can be hard to trust

In many settings one cannot just assume the solver is correct. But:

- Despite the best effort of developers, bugs remain
- Every year SMT-COMP has numerous disagreements between solvers
- Fuzzing tools often find bugs in solvers

Why don't we just certify/qualify 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*
  - And generally more efficiently than solving the problem
- Confidence in results is decoupled from the solver's implementation

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*
  - And generally more efficiently than solving the problem
- Confidence in results is decoupled from the solver's implementation
- Why are proofs not common place in compliance involving solvers?

Challenges for SMT proofs

- Collecting and storing proofs efficiently

  many attempts, no panacea

  [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 some decision procedures (e.g. CAD, strings)

  open

- Standardizing a proof format

  open

- Scalable, trustworthy checking

  many attempts, no panacea

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

The cvc5 team's plan

- CVC4's old proof module struggled with many of those challenges
- Since early 2020, the cvc5 team has been redesigning its proof module *from scratch*
  - Haniel Barbosa, Andrew Reynolds, Alex Ozdemir, Hanna Lachnitt, Scott Viteri, Diego Camargos, Gereon Kremer, Aina Niemetz, Mathias Preiner, Andres Nötzli, Yoni Zohar, Cesare Tinelli, Clark Barrett

The cvc5 team's plan

- CVC4's old proof module struggled with many of those challenges
- Since early 2020, the cvc5 team has been redesigning its proof module *from scratch*
  - Haniel Barbosa, Andrew Reynolds, Alex Ozdemir, Hanna Lachnitt, Scott Viteri, Diego Camargos, Gereon Kremer, Aina Niemetz, Mathias Preiner, Andres Nötzli, Yoni Zohar, Cesare Tinelli, Clark Barrett
- Our goals:
  - Producing proofs should not significantly change the behavior of the solver
    - Incorporate (almost) all relevant optimizations
    - Acceptable performance overhead ($\sim 20\%$)

The cvc5 team's plan

- CVC4's old proof module struggled with many of those challenges
- Since early 2020, the cvc5 team has been redesigning its proof module *from scratch*
  - Haniel Barbosa, Andrew Reynolds, Alex Ozdemir, Hanna Lachnitt, Scott Viteri, Diego Camargos, Gereon Kremer, Aina Niemetz, Mathias Preiner, Andres Nötzli, Yoni Zohar, Cesare Tinelli, Clark Barrett
- Our goals:
  - Producing proofs should not significantly change the behavior of the solver
    - Incorporate (almost) all relevant optimizations
    - Acceptable performance overhead ($\sim 20\%$)
  - Every rule must have an internal proof checker, part of the cvc5 code base
  - Modular infrastructure allowing fine-grained error localization

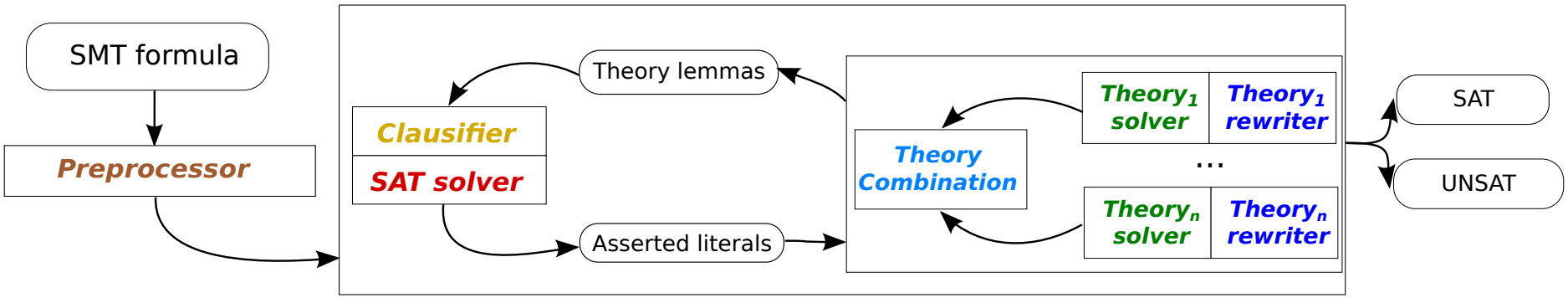
The cvc5 team's plan

- CVC4's old proof module struggled with many of those challenges
- Since early 2020, the cvc5 team has been redesigning its proof module *from scratch*
  - Haniel Barbosa, Andrew Reynolds, Alex Ozdemir, Hanna Lachnitt, Scott Viteri, Diego Camargos, Gereon Kremer, Aina Niemetz, Mathias Preiner, Andres Nötzli, Yoni Zohar, Cesare Tinelli, Clark Barrett
- Our goals:
  - Producing proofs should not significantly change the behavior of the solver
    - Incorporate (almost) all relevant optimizations
    - Acceptable performance overhead ($\sim 20\%$)
  - Every rule must have an internal proof checker, part of the cvc5 code base
  - Modular infrastructure allowing fine-grained error localization
  - Allow custom eager/lazy generation of proofs

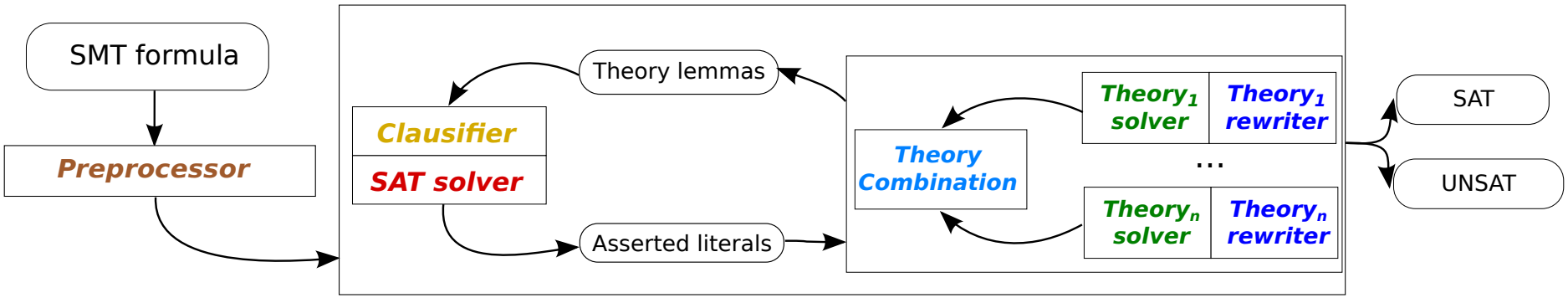
The cvc5 team's plan

- CVC4's old proof module struggled with many of those challenges
- Since early 2020, the cvc5 team has been redesigning its proof module *from scratch*
  - Haniel Barbosa, Andrew Reynolds, Alex Ozdemir, Hanna Lachnitt, Scott Viteri, Diego Camargos, Gereon Kremer, Aina Niemetz, Mathias Preiner, Andres Nötzli, Yoni Zohar, Cesare Tinelli, Clark Barrett
- Our goals:
  - Producing proofs should not significantly change the behavior of the solver
    - Incorporate (almost) all relevant optimizations
    - Acceptable performance overhead ($\sim 20\%$)
  - Every rule must have an internal proof checker, part of the cvc5 code base
  - Modular infrastructure allowing fine-grained error localization
  - Allow custom eager/lazy generation of proofs
  - Support different proof formats (and different external proof checkers)

How an SMT solver works: CDCL($\mathcal{T}$) calculus



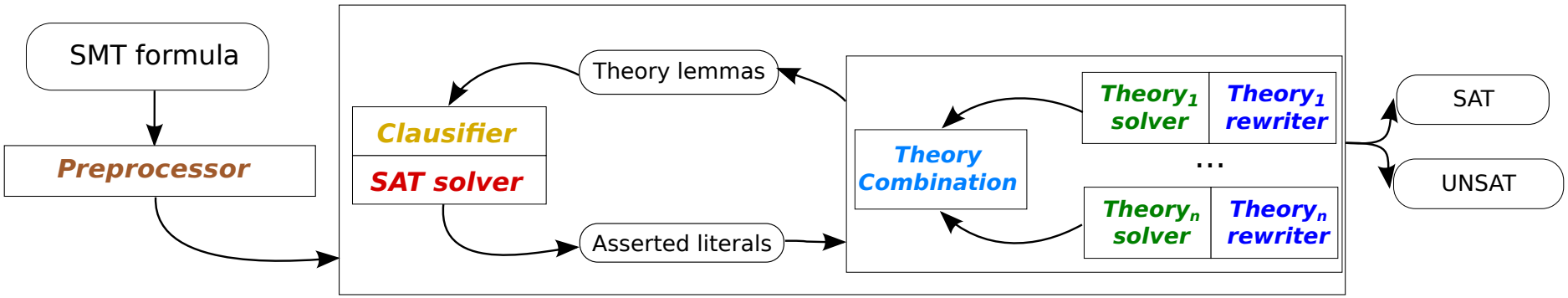
How an SMT solver works: CDCL($\mathcal{T}$) calculus



- ▷ Preprocessor simplifies formula globally

$x \simeq t \wedge F[x] \;\mapsto\; F[t]$ $\qquad$ $F[(ite\ P\ t_1\ t_2)] \;\mapsto\; F[t'] \wedge t' \simeq (ite\ P\ t_1\ t_2)$

How an SMT solver works: CDCL($\mathcal{T}$) calculus

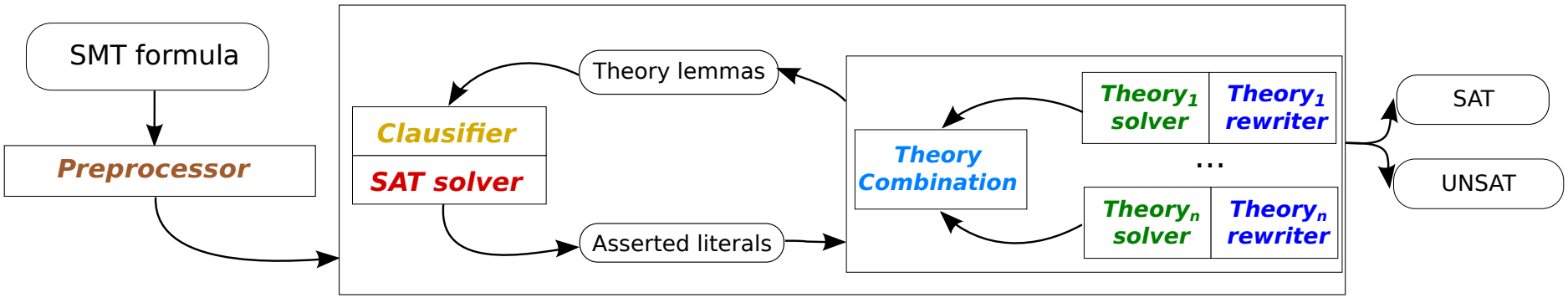


- ▷ Preprocessor simplifies formula globally

  $x \simeq t \wedge F[x] \;\mapsto\; F[t]$ $\qquad$ $F[(ite\ P\ t_1\ t_2)] \;\mapsto\; F[t'] \wedge t' \simeq (ite\ P\ t_1\ t_2)$

- ▷ Clausifier converts to Conjunctive Normal Form (CNF)

How an SMT solver works: CDCL($\mathcal{T}$) calculus

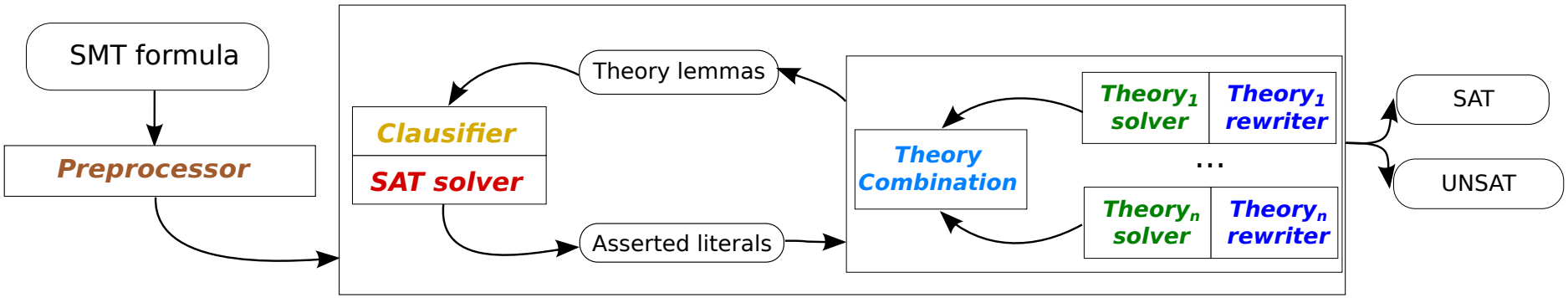


- ▷ Preprocessor simplifies formula globally

  $x \simeq t \wedge F[x] \;\mapsto\; F[t]$ $\qquad$ $F[(ite\ P\ t_1\ t_2)] \;\mapsto\; F[t'] \wedge t' \simeq (ite\ P\ t_1\ t_2)$

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

How an SMT solver works: CDCL($\mathcal{T}$) calculus



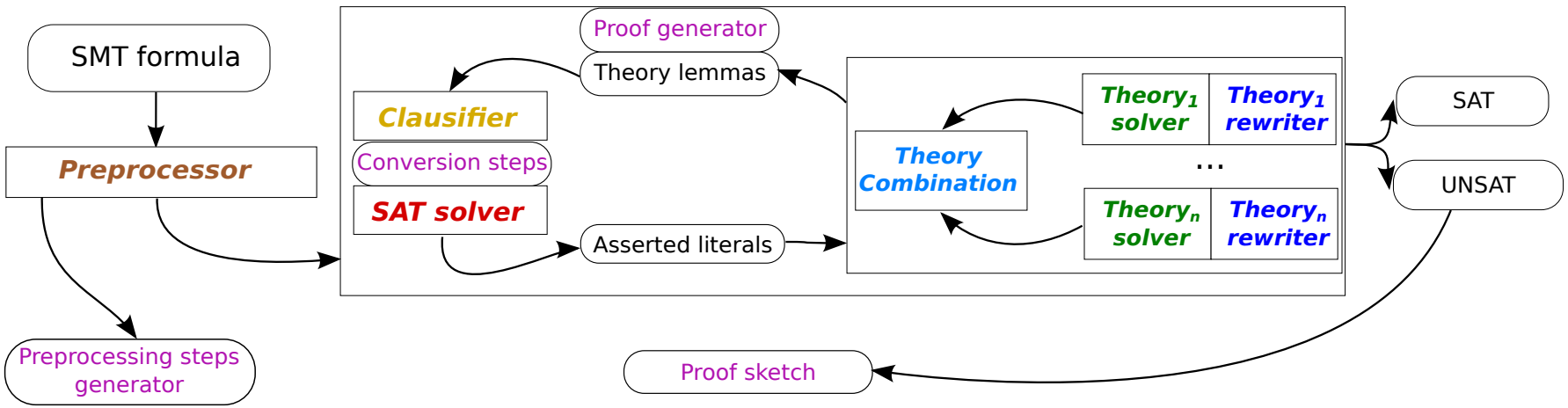
- ▷ Preprocessor simplifies formula globally

  $x \simeq t \wedge F[x] \;\mapsto\; F[t]$ $\qquad$ $F[(ite\ P\ t_1\ t_2)] \;\mapsto\; F[t'] \wedge t' \simeq (ite\ P\ t_1\ t_2)$

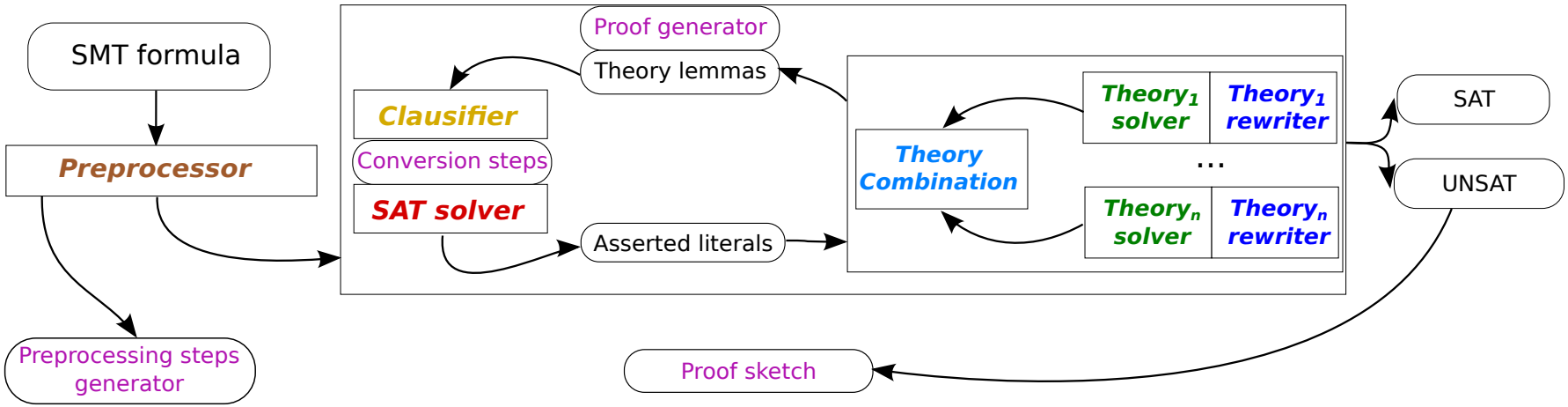
- ▷ Clausifier converts to Conjunctive Normal Form (CNF)
- ▷ SAT solver asserts literals that must hold based on Boolean abstraction
- ▷ Theory solvers check consistency in the theory
- ▷ Theory rewriters simplify terms of their theories

  $x + 0 \to x$ $\qquad$ $\mathsf{a} \not\simeq \mathsf{a} \to \bot$ $\qquad$ $(\text{str.replace}\ x\ (\text{str.++}\ x\ x)\ y) \to x$

Proof module architecture



Proof module architecture



- Proof sketch contains a *resolution* proof of unsatisfiability of CNF
  - Leafs are theory lemmas and preprocessed assertions

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

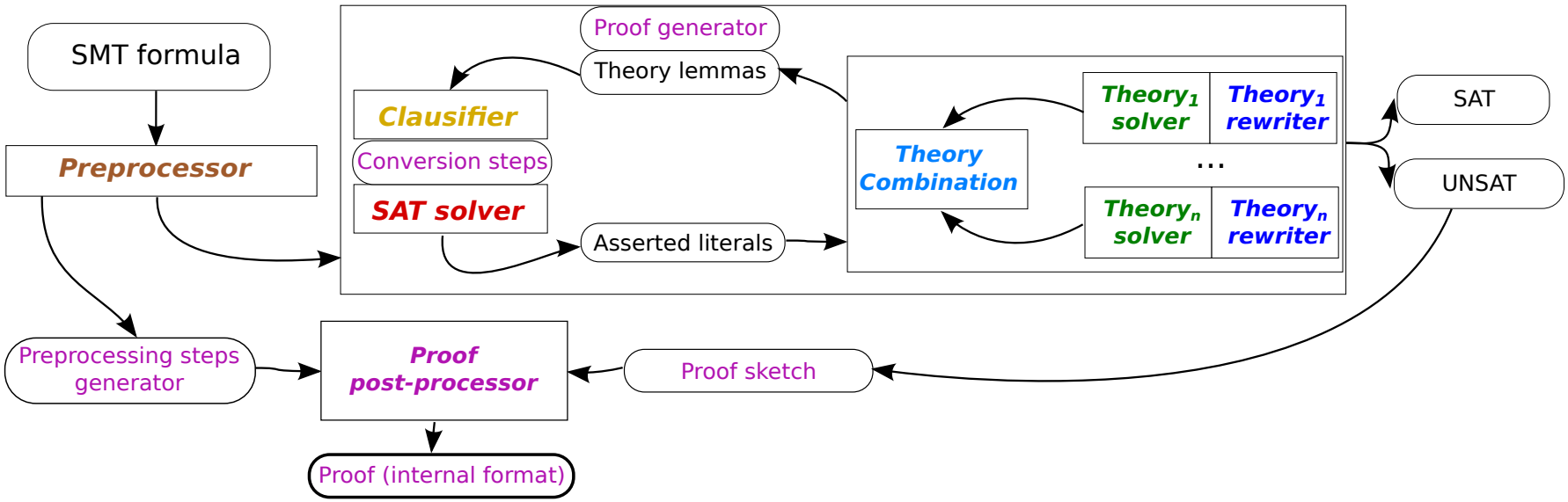
$$\neg(a \simeq b) \vee f(a) \simeq f(b)$$
$$\neg(y > 1) \vee \neg(x < 1) \vee y > x$$

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

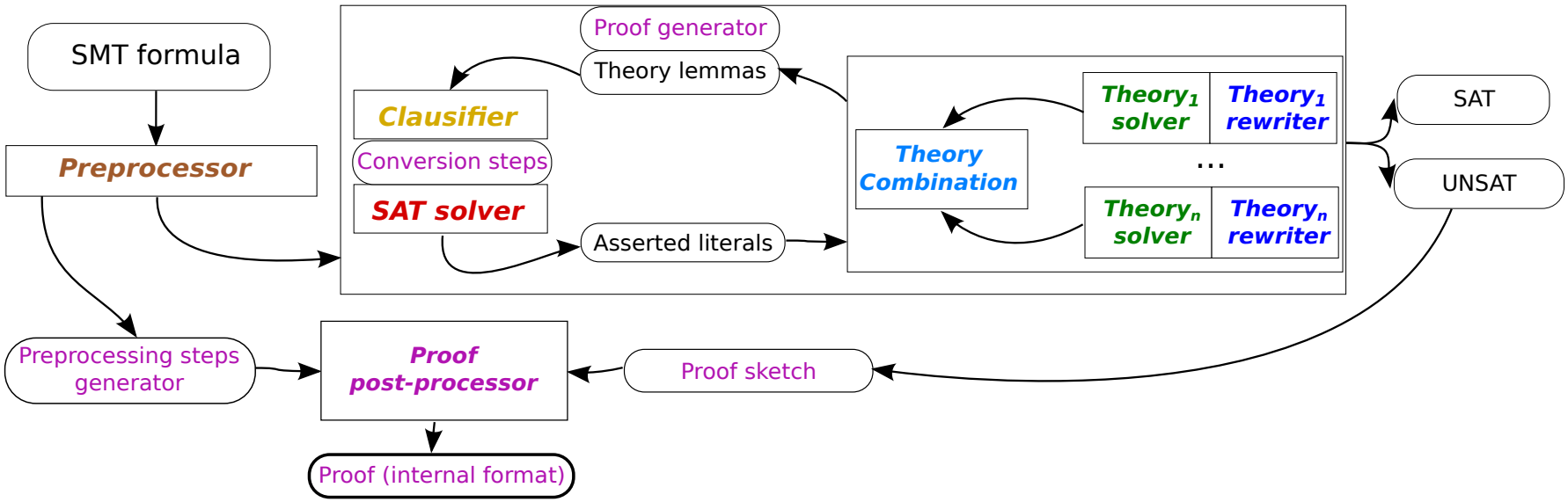
$$\neg(\varphi_1 \wedge \cdots \wedge \varphi_n) \vee \varphi_i$$

- Generators are hooks to proof-producing modules in preprocessor/theory reasoners

Proof module architecture



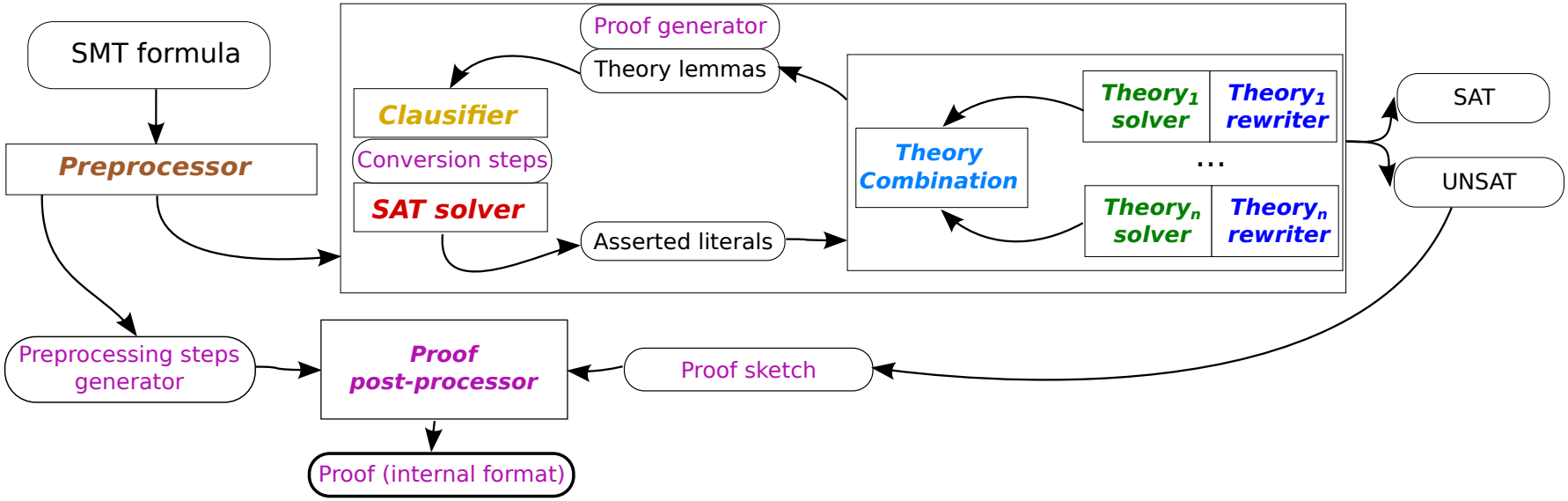
Proof module architecture



- ▷ Post-processor invokes proof generators of leaves in sketch
- ▷ Also (optionally) expands coarse-grained steps into fine-grained ones

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

Proof module architecture

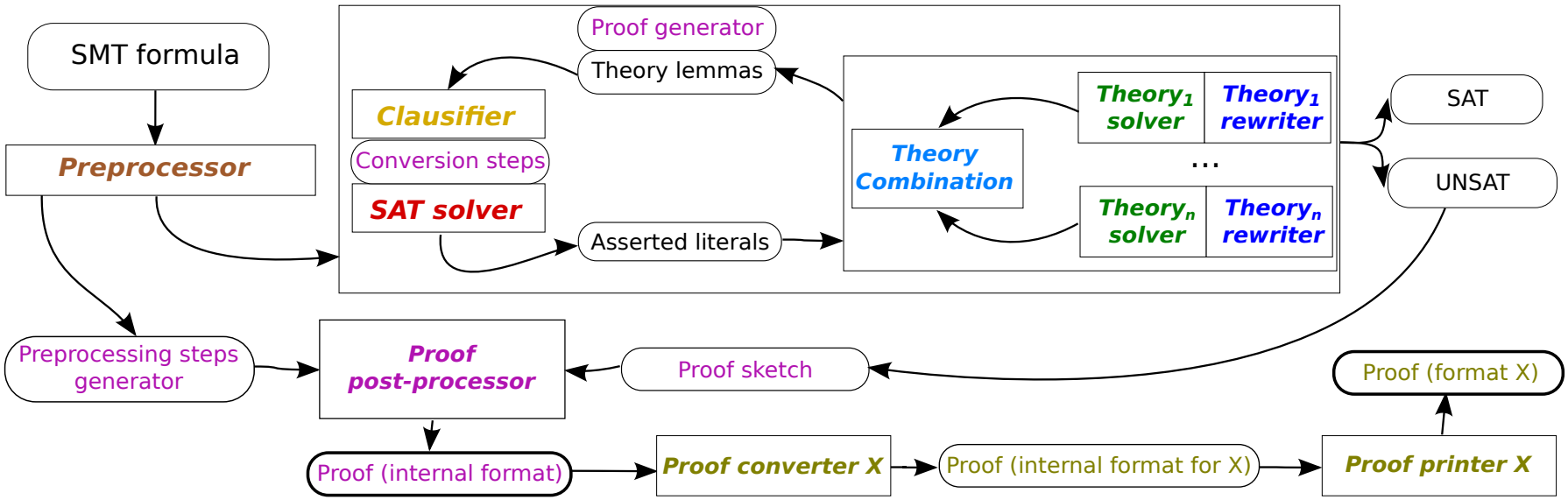


- ▷ Post-processor invokes proof generators of leaves in sketch
- ▷ Also (optionally) expands coarse-grained steps into fine-grained ones

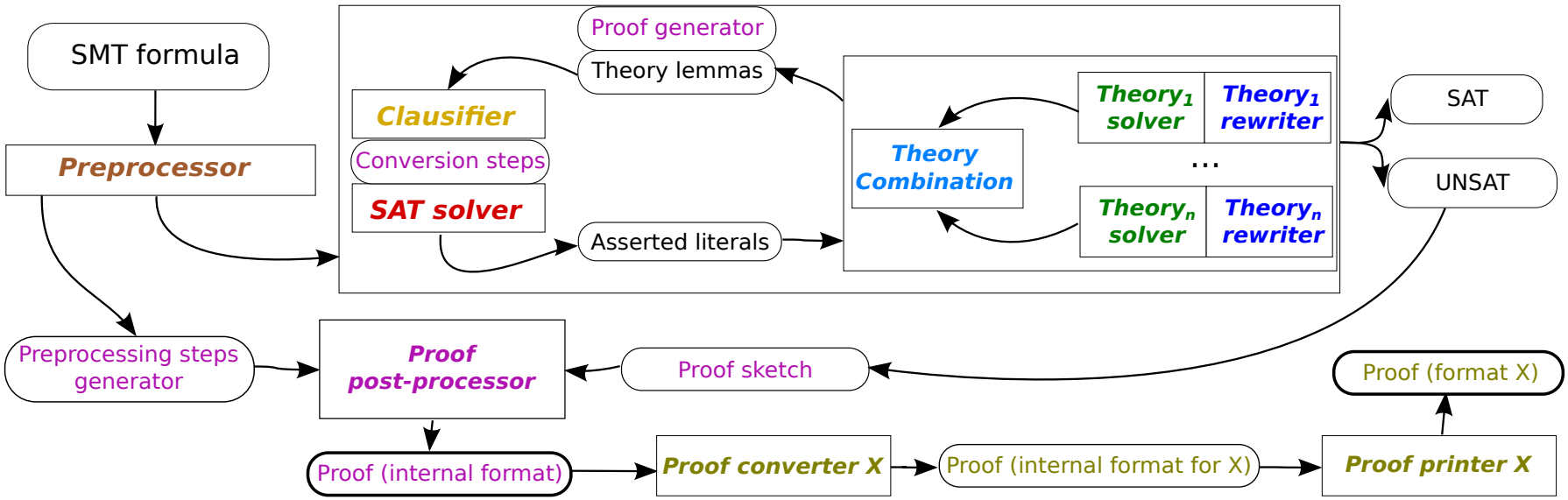
$$\frac{a \simeq 0 \qquad b \simeq 1}{(a > b \wedge F) \simeq \bot}\ \text{MACRO}$$

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

Proof module architecture



Proof module architecture



- Internal proof can be converted and printed to different proof formats
  - Convert the AST of internal proofs based on X
  - Pretty printer that proof in format X
- No internal proof checking for X, since it'll be checked externally

Demo!

Consider the following unsatisfiable SMT problem

```
(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-fun f (U) U)

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

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

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

Ongoing work

- Conversions to different proof formats
  - LFSC: custom checker via LFSC's type checking
  - Lean: custom checker via Lean4's type checking
  - Alethe
    - proof reconstruction in Isabelle/HOL via Sledghammer
    - proof reconstruction in Coq via SMTCoq
    - custom checker in Rust
  - Dot: proof visualization
- Fine-grained reconstruction of theory rewrite steps
  - Simple rewrite rules specified in DSL
  - Compilation into unification goals in cvc5
  - Elaboration of complex rewrite steps in terms of simple ones from DSL
- Incorporating DRAT proofs for BV lemmas from black-box SAT solvers

Conclusion

- For 1.5 years the cvc5 team has been hard at work to produce better proofs
- Fine-grained proofs are now available for most of cvc5's reasoning
  - Proofs for the strings solver has been a special milestone
- Multiple proof formats are supported
- Integration into multiple proof checkers are ongoing
  - Including the formalization of new calculi in Lean, LFSC, Isabelle/HOL
- We expect the high-quality proofs produced by cvc5 to facilitate automated compliance

Better SMT proofs for certifying compliance

(or what a bunch of people achieve in 1.5 years writing from scratch a proof module for an state-of-the-art SMT solver)

Haniel Barbosa

FACC 2021

2021–07–19, The Internet

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, 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.

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.

References

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 Progran 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, U 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

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.

References

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.

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.

References

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.

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.