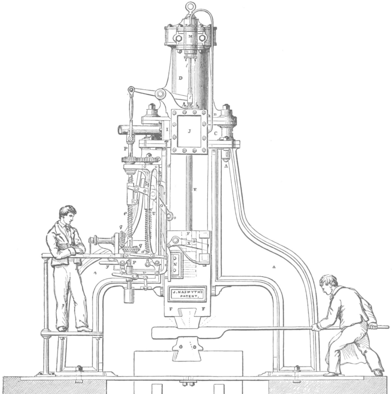
SMT Proof Certificates With AletheLF

Fast, Flexible, and Familiar

Hans-Jörg Schurr
CS Extras – Grinnell College
March 7 2024



Starting Point

Many human pursuits demand precise and correct reasoning.

Starting Point

Many human pursuits demand precise and correct reasoning.

Starting Point

Many human pursuits demand precise and correct reasoning.

Starting Point

Many human pursuits demand precise and correct reasoning.

- Our tool: formal logic.
- It's unfeasible to write formal proofs by hand:

Reliability mistakes happen easily
Effort horribly time consuming

Software Supported Proof Construction

Automated Theorem Provers

"Push Button"

Usually refute problems and produce proofs.

Software Supported Proof Construction

Automated Theorem Provers

"Push Button"

Usually refute problems and produce proofs.

Satisfiability Modulo Theories

Propositional reasoning + theories.

- Functions
- Linear Arithmetic
- Quantifiers

Examples:

- **cvc5**
- **veriT**
- Z3

Software Supported Proof Construction

Automated Theorem Provers

"Push Button"

Usually refute problems and produce proofs.

Satisfiability Modulo Theories

Propositional reasoning + theories.

- Functions
- Linear Arithmetic
- Quantifiers

Examples:

- **cvc5**
- **veriT**
- Z3

Proof Assistants

Reliability trusted kernel

Effort proof construction routines

Examples:

- **Isabelle/HOL**
- Coq
- Lean

Software Supported Proof Construction

Automated Theorem Provers

"Push Button"

Usually refute problems and produce proofs.

Satisfiability Modulo Theories

Propositional reasoning + theories.

- Functions
- Linear Arithmetic
- Quantifiers

Examples:

- **cvc5**
- **veriT**
- Z3

Proof Assistants

Reliability trusted kernel

Effort proof construction routines

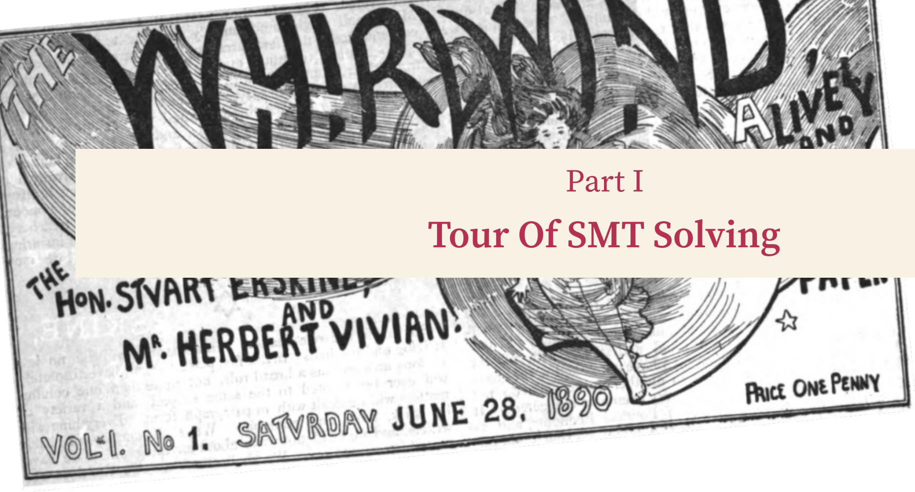
Examples:

- **Isabelle/HOL**
- Coq
- Lean

Automation

Must build upon the kernel.

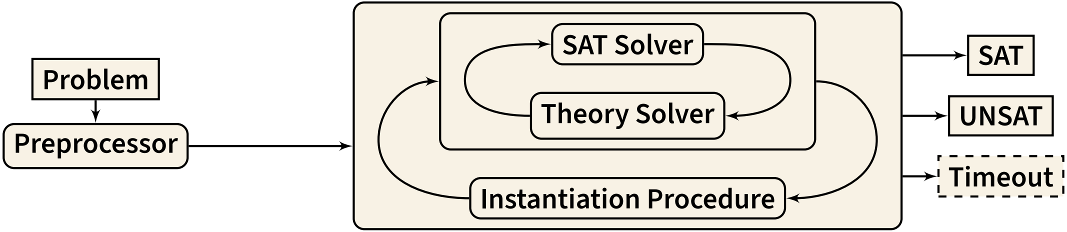
- Simplifier: replaces equal by equal.
- Integration of automated theorem provers.



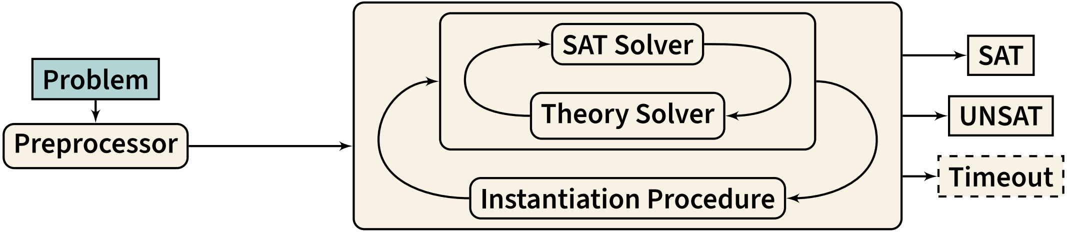
Part I

Tour Of SMT Solving

SMT Solving As A Diagram



SMT Solving As A Diagram



An Example: Problem Specification

1. We produce 1L, 2L, and 3L bottles.
2. The price of a bottle is the volume plus four times the wall thickness (in mm).
3. The price must be less than 4\$.
4. If the new machine is broken, we cannot produce 3L bottles, and the wall thickness must be more than 1mm.
5. The new machine is broken.
6. For all bottle sizes, the wall thickness in millimetre can at most be the volume in liters.

An Example: Problem Specification

1. We produce 1L, 2L, and 3L bottles.
2. The price of a bottle is the volume plus four times the wall thickness (in mm).
3. The price must be less than 4\$.
4. If the new machine is broken, we cannot produce 3L bottles, and the wall thickness must be more than 1mm.
5. The new machine is broken.
6. For all bottle sizes, the wall thickness in millimetre can at most be the volume in liters.

1. $v = 1$ or $v = 2$ or $v = 3$

An Example: Problem Specification

1. We produce 1L, 2L, and 3L bottles.
2. The price of a bottle is the volume plus four times the wall thickness (in mm).
3. The price must be less than 4\$.
4. If the new machine is broken, we cannot produce 3L bottles, and the wall thickness must be more than 1mm.
5. The new machine is broken.
6. For all bottle sizes, the wall thickness in millimetre can at most be the volume in liters.

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$

An Example: Problem Specification

1. We produce 1L, 2L, and 3L bottles.
2. The price of a bottle is the volume plus four times the wall thickness (in mm).
3. The price must be less than 4\$.
4. If the new machine is broken, we cannot produce 3L bottles, and the wall thickness must be more than 1mm.
5. The new machine is broken.
6. For all bottle sizes, the wall thickness in millimetre can at most be the volume in liters.

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$

An Example: Problem Specification

1. We produce 1L, 2L, and 3L bottles.
2. The price of a bottle is the volume plus four times the wall thickness (in mm).
3. The price must be less than 4\$.
4. If the new machine is broken, we cannot produce 3L bottles, and the wall thickness must be more than 1mm.
5. The new machine is broken.
6. For all bottle sizes, the wall thickness in millimetre can at most be the volume in liters.

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$
4. If b then: not $v = 3$ and $t > 1$

An Example: Problem Specification

1. We produce 1L, 2L, and 3L bottles.
2. The price of a bottle is the volume plus four times the wall thickness (in mm).
3. The price must be less than 4\$.
4. If the new machine is broken, we cannot produce 3L bottles, and the wall thickness must be more than 1mm.
5. The new machine is broken.
6. For all bottle sizes, the wall thickness in millimetre can at most be the volume in liters.

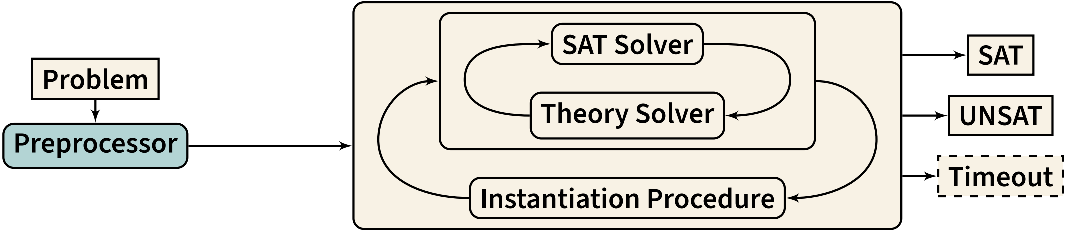
1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$
4. If b then: not $v = 3$ and $t > 1$
5. b!

An Example: Problem Specification

1. We produce 1L, 2L, and 3L bottles.
2. The price of a bottle is the volume plus four times the wall thickness (in mm).
3. The price must be less than 4\$.
4. If the new machine is broken, we cannot produce 3L bottles, and the wall thickness must be more than 1mm.
5. The new machine is broken.
6. For all bottle sizes, the wall thickness in millimetre can at most be the volume in liters.

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$
4. If b then: not $v = 3$ and $t > 1$
5. b!
6. For all z: if $v = z$ then $t \leq z$

SMT Solving As A Diagram



An Example: Preprocessing

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$
4. If b then: not $v = 3$ and $t > 1$

5. b!
6. For all z: if $v = z$ then $t \leq z$

An Example: Preprocessing

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$
4. If b then: not $v = 3$ and $t > 1$

5. b!
6. For all z: if $v = z$ then $t \leq z$

1. $v = 1$ or $v = 2$ or $v = 3$

An Example: Preprocessing

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$
4. If b then: not $v = 3$ and $t > 1$

5. b!
6. For all z: if $v = z$ then $t \leq z$

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < 4$

An Example: Preprocessing

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$
4. If b then: not $v = 3$ and $t > 1$
5. b!
6. For all z: if $v = z$ then $t \leq z$

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < 4$
3.
4. not b or not $v = 3$

An Example: Preprocessing

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$
4. If b then: not $v = 3$ and $t > 1$
5. b!
6. For all z: if $v = z$ then $t \leq z$

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < 4$
3.
4. not b or not $v = 3$

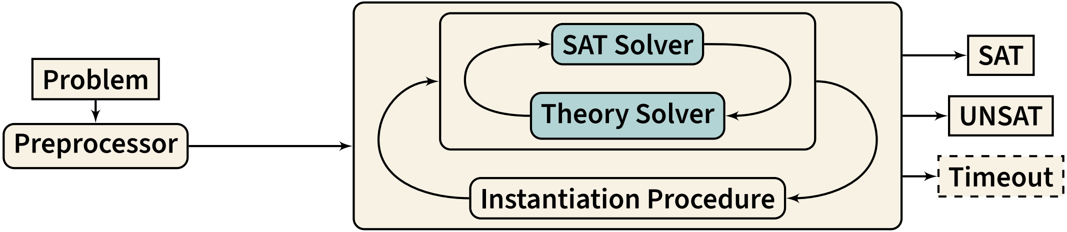
   not b or $t > 1$
5. b

An Example: Preprocessing

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < p$
3. $p = 4$
4. If b then: not $v = 3$ and $t > 1$
5. b!
6. For all z: if $v = z$ then $t \leq z$

1. $v = 1$ or $v = 2$ or $v = 3$
2. $v + 2t < 4$
3.
4. not b or not $v = 3$
   not b or $t > 1$
5. b
6. For all z: not $v = z$ or not $t > z$

SMT Solving As A Diagram



An Example: The Ground Solver

- $v = 1$ or $v = 2$ or $v = 3$
- $v + 2t < 4$
- not b or not $v = 3$
- not b or $t > 1$
- b
- For all z: not $v = z$ or not $t > z$

An Example: The Ground Solver

- $v = 1$ or $v = 2$ or $v = 3$
- $v + 2t < 4$
- not b or not $v = 3$
- not b or $t > 1$
- b
- ~~For all z: not $v = z$ or not $t > z$~~

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b

An Example: The Ground Solver

- $v = 1$ or $v = 2$ or $v = 3$
- $v + 2t < 4$
- not b or not $v = 3$
- not b or $t > 1$
- b
- ~~For all z: not $v = z$ or not $t > z$~~

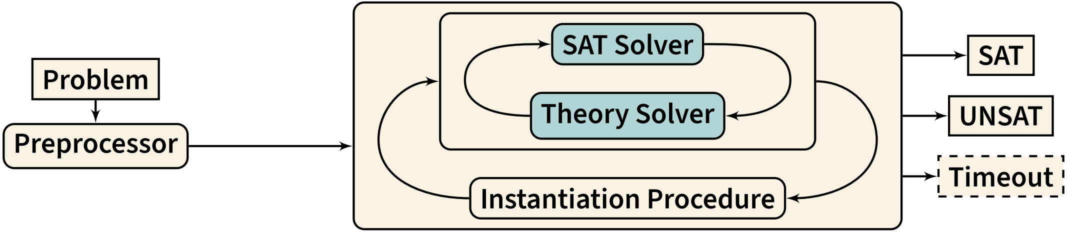
SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

SMT Solving As A Diagram



An Example: The SAT Solver and the Theory Solver

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

An Example: The SAT Solver and the Theory Solver

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b

SAT Solver

I pick b, p_2, p_4, and p_5 😊

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

An Example: The SAT Solver and the Theory Solver

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b

SAT Solver

I pick b, p_2, p_4, and p_5 😊

Linear Arithmetic Solver

1. I get $v = 2$, $v + 2t < 4$, and $t > 1$

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

An Example: The SAT Solver and the Theory Solver

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b
- not p_2 or not p_4 or not p_5

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

SAT Solver

I pick b**,** p_2**,** p_4**, and** p_5 ☺️

Linear Arithmetic Solver

1. I get $v = 2$**,** $v + 2t < 4$**, and** $t > 1$
2. Doesn't work: **not** $v = 2$ **or not** $v + 4t < 4$ **or not** $t > 1$ 🤔

An Example: The SAT Solver and the Theory Solver

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b
- not p_2 or not p_4 or not p_5

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

SAT Solver

I pick b, p_2, p_4, **and** p_5 ☺️

Linear Arithmetic Solver

1. I get $v = 2$, $v + 2t < 4$, **and** $t > 1$
2. Doesn't work: **not** $v = 2$ **or not** $v + 4t < 4$ **or not** $t > 1$ 🤔

SAT Solver

I have to pick b, p_1, p_4, **and** p_5 🤞

An Example: The SAT Solver and the Theory Solver

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b
- not p_2 or not p_4 or not p_5

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

SAT Solver

I pick b, p_2, p_4, and p_5 ☺️

Linear Arithmetic Solver

1. I get $v = 2$, $v + 2t < 4$, and $t > 1$
2. Doesn't work: **not** $v = 2$ **or not** $v + 4t < 4$ **or not** $t > 1$ 🤔

SAT Solver

I have to pick b, p_1, p_4, and p_5 🤞

Linear Arithmetic Solver

1. I get $v = 1$, $v + 2t < 4$, and $t > 1$

An Example: The SAT Solver and the Theory Solver

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b
- not p_2 or not p_4 or not p_5

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

SAT Solver

I pick b, p_2, p_4, and p_5 ☺️

Linear Arithmetic Solver

1. I get $v = 2$, $v + 2t < 4$, and $t > 1$
2. Doesn't work: **not** $v = 2$ **or not** $v + 4t < 4$ **or not** $t > 1$ 🤔

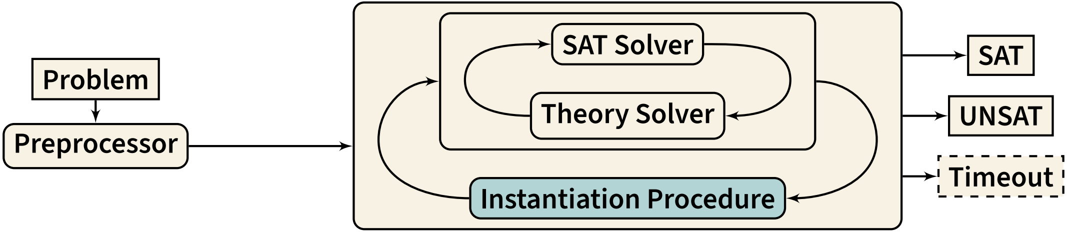
SAT Solver

I have to pick b, p_1, p_4, and p_5 🤞

Linear Arithmetic Solver

1. I get $v = 1$, $v + 2t < 4$, and $t > 1$
2. That works! 🎉

SMT Solving As A Diagram



An Example: Quantifier Instantiation

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b

Instantiation Procedure

- I have **For all z: not** $v = z$ **or not** $t > z$

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

An Example: Quantifier Instantiation

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b

Instantiation Procedure

- I have **For all z: not** $v = z$ **or not** $t > z$
- What happens if I pick $z \leftarrow 1$? 😈

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

An Example: Quantifier Instantiation

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b

Instantiation Procedure

- I have **For all z: not** $v = z$ **or not** $t > z$
- What happens if I pick $z \leftarrow 1$? 😈
- That's **not** $v = 1$ **or not** $t > 1$

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

An Example: Quantifier Instantiation

SAT Problem

- p_1 or p_2 or p_3
- p_4
- not b or not p_3
- not b or p_5
- b
- not p_2 or not p_4 or not p_5

Theory Literals

- p_1 is $v = 1$, p_2 is $v = 2$, p_3 is $v = 3$
- p_4 is $v + 2t < 4$
- p_5 is $t > 1$

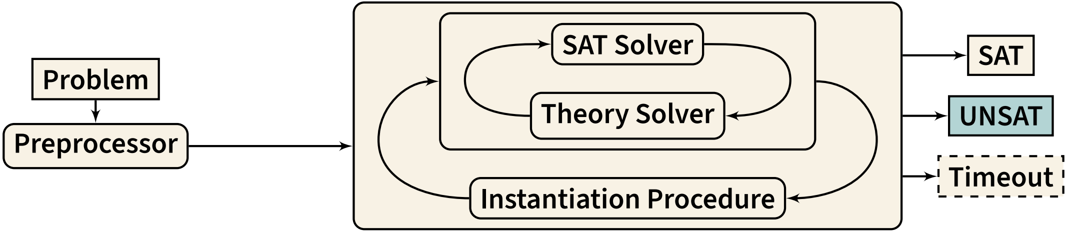
Instantiation Procedure

- I have **For all z: not** $v = z$ **or not** $t > z$
- What happens if I pick $z \leftarrow 1$? 😈
- That's **not** $v = 1$ **or not** $t > 1$

SAT Solver

- That's **not** p_1 **or not** p_5
- Oh no 😢

SMT Solving As A Diagram



Do we get an argument for the contradiction?

1. Since the new machine is broken, the volume cannot be 3L, and the wall thickness is > 1mm.
2. If the volume would be 2L, and the thickness is larger than 1L, then we get a contradiction with the price bound $v + 2t < 4$.
3. Since only 1L, 2L, and 3L bottles are produced, the volume must be 1L.
4. Because, the wall thickness must be smaller than the volume in liters, the wall thickness must be < 1mm.
5. This is a contradiction with the fact that we can only produce bottles with a wall thickness > 1mm.

Part II

SMT Proofs in Use: Alethe in Isabelle/HOL

The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

1. $f(x+5) = f(5+x)$ by ×_unit
2. $x + 5 = 5 + x$ by cong
3. $x + 5 = x + 5$ by +_com
4. $\top$ by refl

The Sledgehammer Pipeline

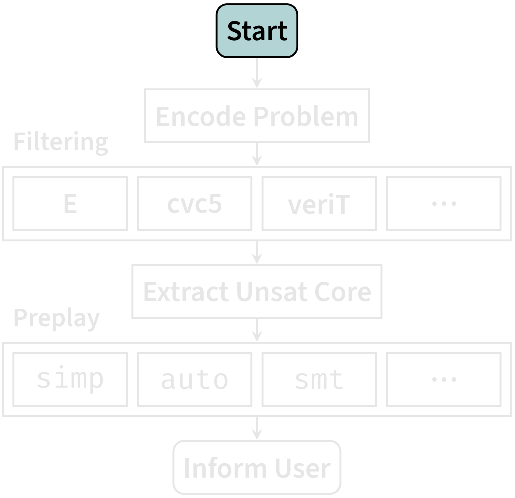
lemma $f(x+5) = f((1 \times 5) + x)$

🔨

The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$



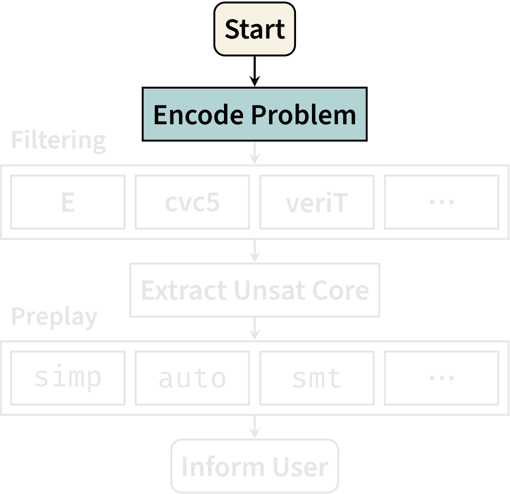
The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

- - -

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add B := {×_unit, ×_com, ×_assoc, +_unit, +_com, +_assoc, cong, refl, ...}



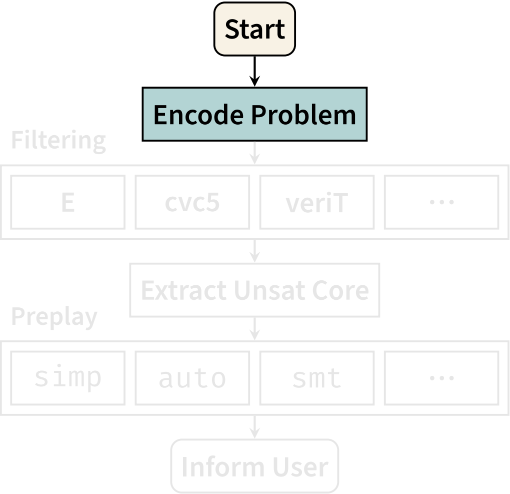
The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

🔨

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add B $:=$ $\{$**×_unit**, ×_com, ×_assoc, +_unit, **+_com**, +_assoc, **cong**, **refl**, ...$\}$

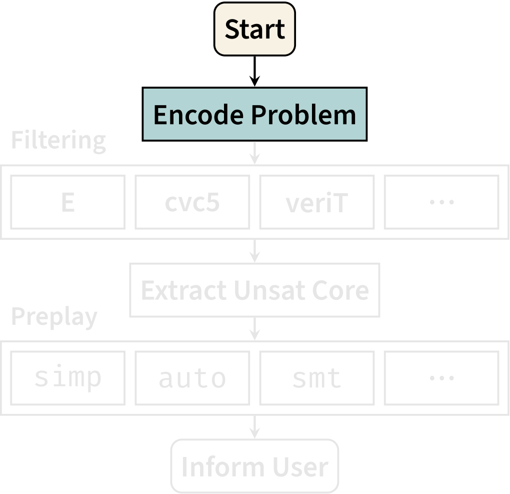


The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add B := {×_unit, **×_com**, **×_assoc**, **+_unit**, +_com, **+_assoc**, cong, refl, **...**}

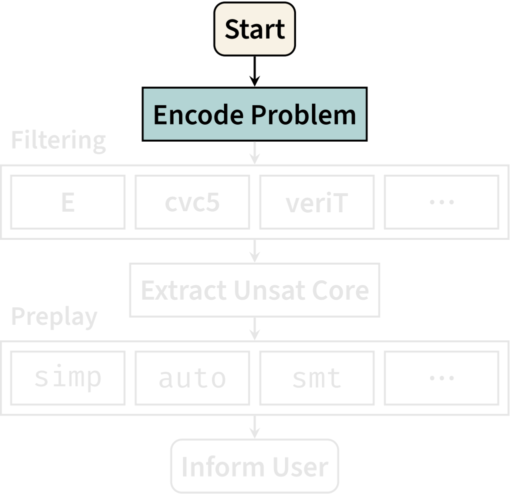


The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B := \{$x_unit, x_com, x_assoc, ...$\}$



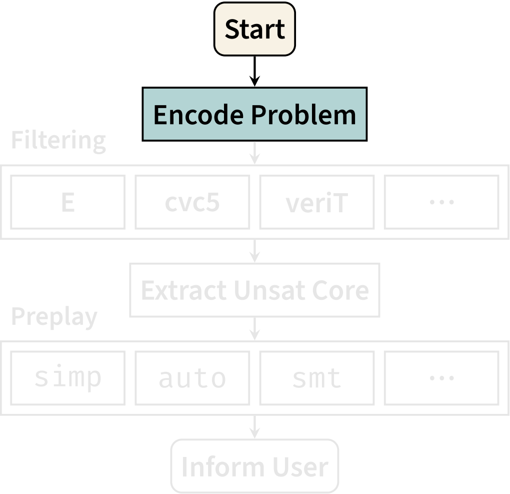
The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$



The Sledgehammer Pipeline

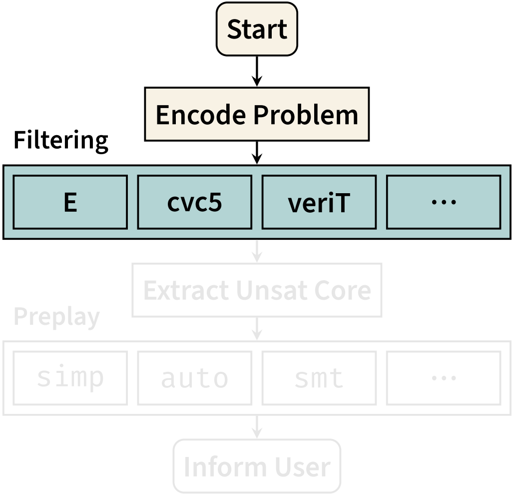
lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {x_unit, x_com, x_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ...



The Sledgehammer Pipeline

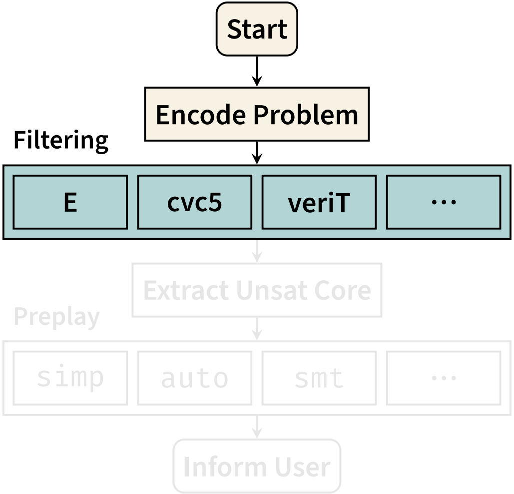
lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

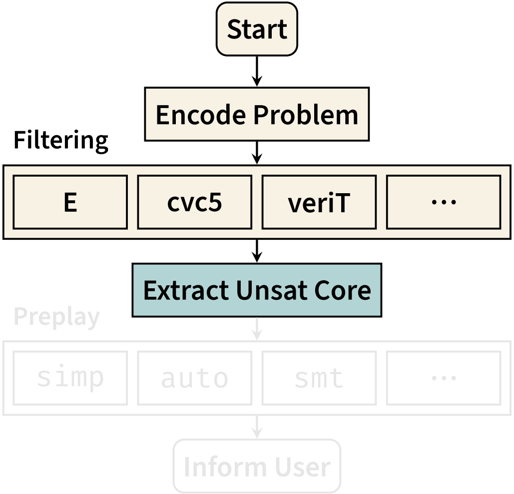
Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

Core $C :=$ {×_unit, +_com, cong, refl}



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

🔨

Goal $L := f(x+5) = f((1 \times 5) + x)$

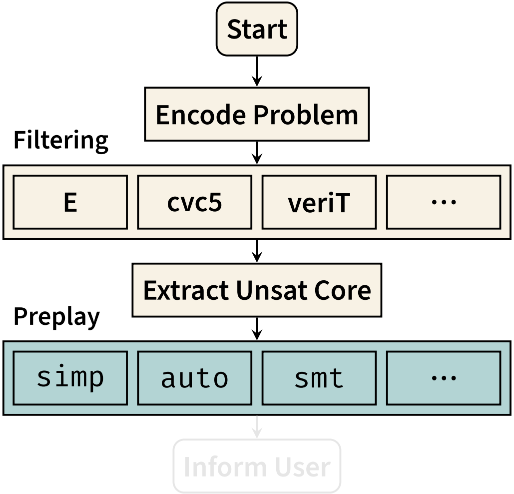
Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

Core $C :=$ {×_unit, +_com, cong, refl}

Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, ...



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

🔨

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, …}

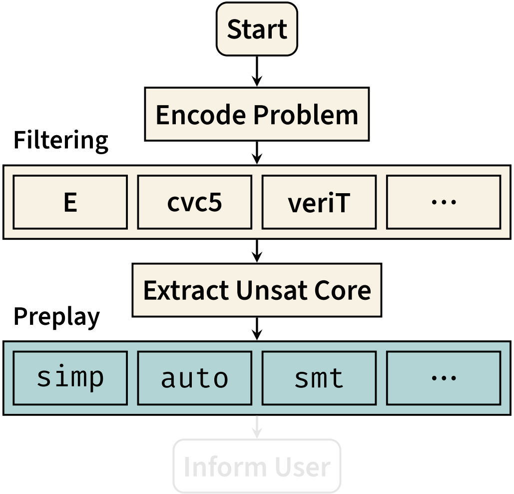
Encode $B \wedge \neg L$

Try E, cvc5, veriT, … veriT shows unsat!

Core $C :=$ {×_unit, +_com, cong, refl}

Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, …

`smt` shows unsat!



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

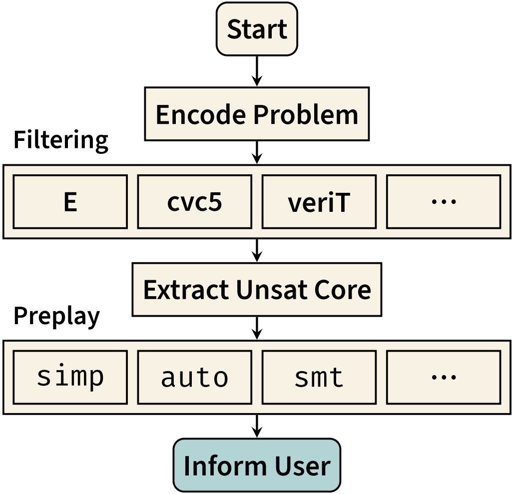
Core $C :=$ {×_unit, +_com, cong, refl}

Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, ...

`smt` shows unsat!

Done!

Try `smt`: ×_unit, cong, +_com, refl



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

1. $\top$ by `smt`: ×_unit, cong, +_com, refl

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B :=$ {×_unit, ×_com, ×_assoc, ...}

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

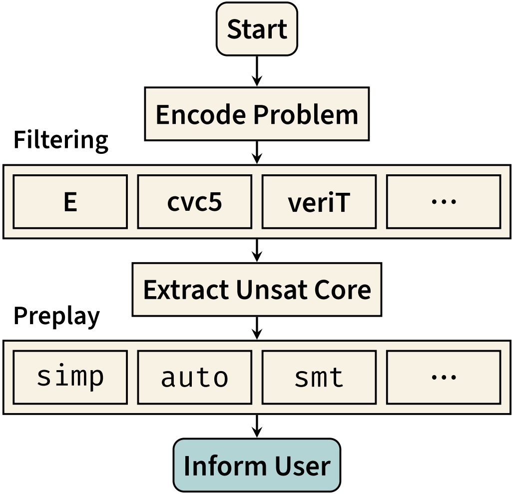
Core $C :=$ {×_unit, +_com, cong, refl}

Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, ...

`smt` shows unsat!

Done!

Try `smt`: ×_unit, cong, +_com, refl



The Sledgehammer Pipeline

lemma $f(x+5) = f((1 \times 5) + x)$

1. $\top$ by `smt`: ×_unit, cong, +_com, refl

Goal $L := f(x+5) = f((1 \times 5) + x)$

Add $B := \{$×_unit, ×_com, ×_assoc, ...$\}$

Encode $B \wedge \neg L$

Try E, cvc5, veriT, ... veriT shows unsat!

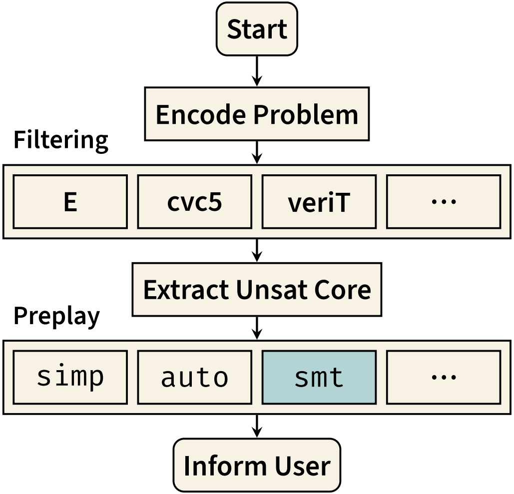
Core $C := \{$×_unit, +_com, cong, refl$\}$

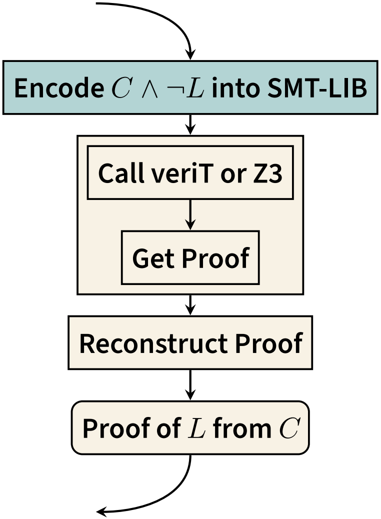
Preplay `simp`, `auto`, `smt` on $C \wedge \neg L$, ...

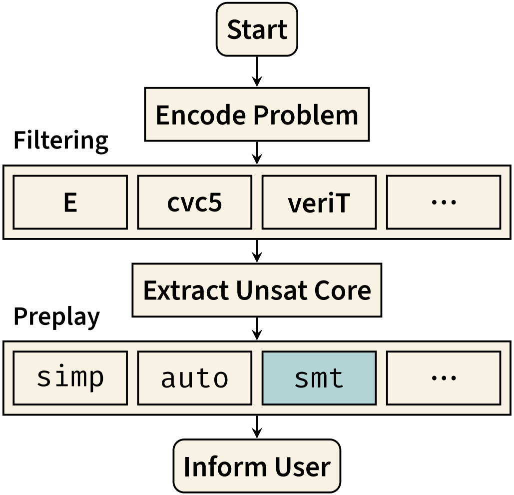
`smt` shows unsat!

Done!

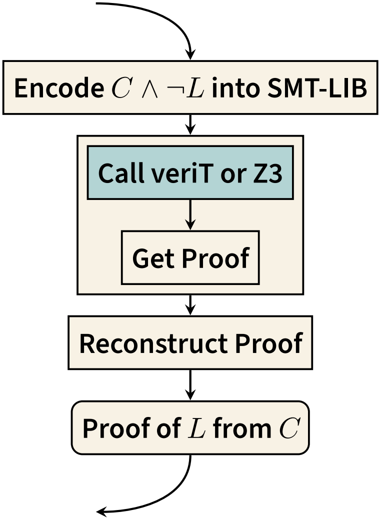
Try `smt`: ×_unit, cong, +_com, refl

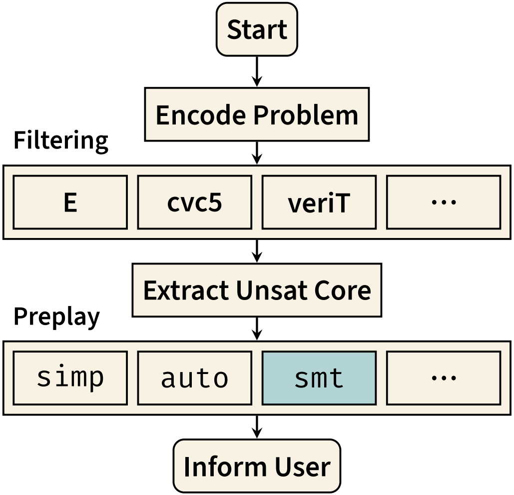



Encode $C \wedge \neg L$ into SMT-LIB
Call veriT or Z3
Get Proof
Reconstruct Proof
Proof of L from C

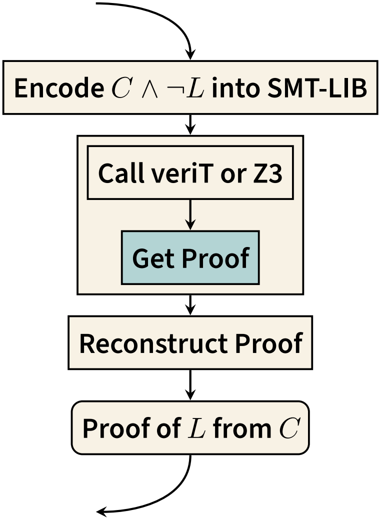

Start
Encode Problem
Filtering
E
cvc5
veriT
...
Extract Unsat Core
Preplay
simp
auto
smt
...
Inform User

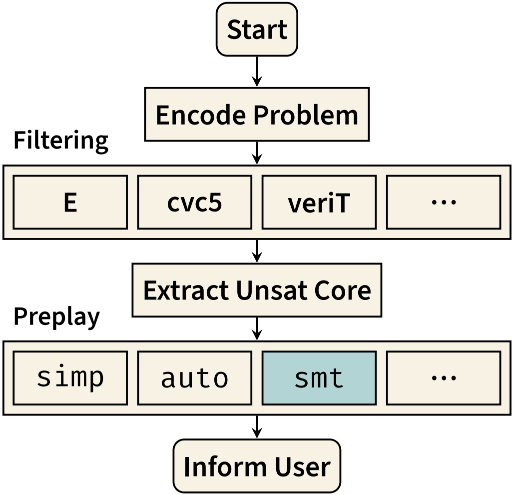
Using Proofs: `smt`



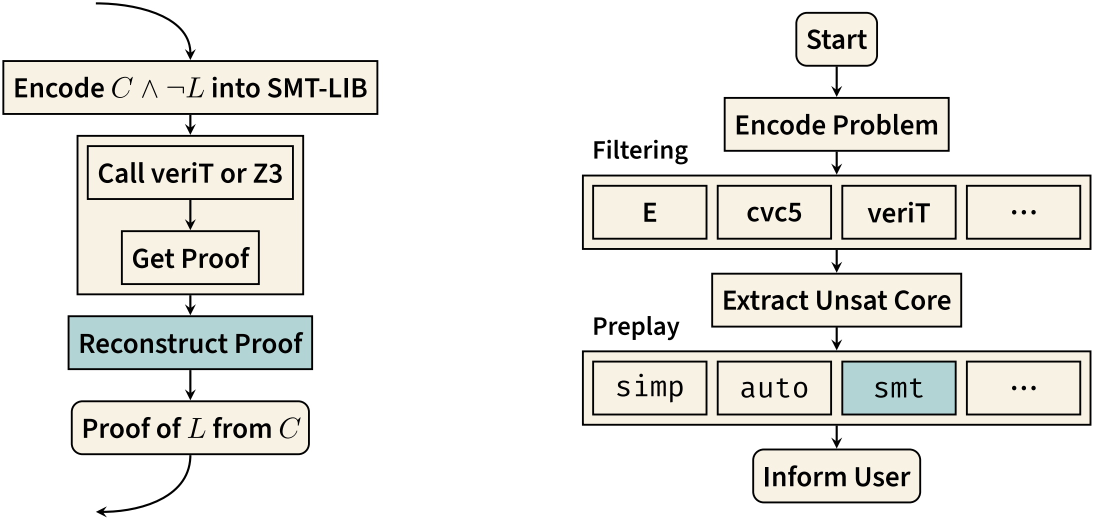


Using Proofs: `smt`

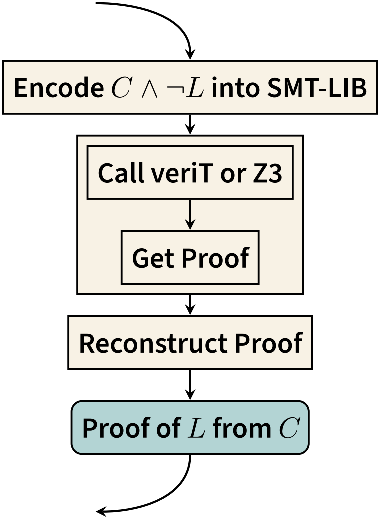


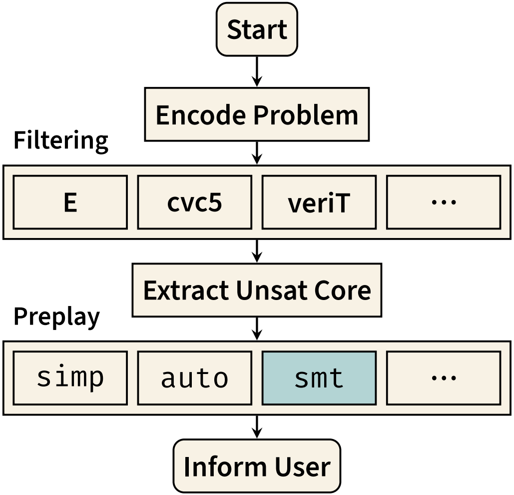


Using Proofs: `smt`



Using Proofs: `smt`





SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Macro rules, different philosophy.

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Macro rules, different philosophy.

Questions

- Can we make the proofs more rigorous?
- What can we learn from doing reconstruction?
- Is veriT's fine-grained proof & quantifier support useful?

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Macro rules, different philosophy.

Questions

- Can we make the proofs more rigorous? **Yes: Alethe!**
- What can we learn from doing reconstruction?
- Is veriT's fine-grained proof & quantifier support useful?

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Macro rules, different philosophy.

Questions

- Can we make the proofs more rigorous? **Yes: Alethe!**
- What can we learn from doing reconstruction? **Next Part.**
- Is veriT's fine-grained proof & quantifier support useful?

SMT Proof Reconstruction Circa 2018

veriT Proofs

- New in 2017: reasoning about binders. [Barbosa, et al. 2017]
- Reconstruction prototype by Fleury for validation. [Barbosa, et al. 2020]
- Philosophy: fine-grained proofs, natural deduction style.

The `smt` tactic: Z3 only

- From 2009, by Böhme, et al.
- Stable, but bound to a specific Z3 version.
- Macro rules, different philosophy.

Questions

- Can we make the proofs more rigorous? **Yes: Alethe!**
- What can we learn from doing reconstruction? **Next Part.**
- Is veriT's fine-grained proof & quantifier support useful? **Yes: veriT `smt`!**

Alethe Proofs: Basic Structure

$$\dfrac{\dfrac{\dfrac{t_2}{t_3}}{\vdots}\quad t_1 \qquad \neg t_1}{\bot}\ \text{resolution}$$

$$t_1, t_2 \vdash \bot$$

```
(assume a0 t1)
(assume a1 t2)
(step  s1 (cl t3)
        :premises (a1)     :rule rule1)
...
(step s20 (cl (not t1))
        :premises (s19)    :rule rule2)
(step s21 (cl )
        :premises (a0 s20) :rule resolution)
```

Alethe Proofs: Subproofs With Assumptions

$$
\dfrac{\dfrac{t_1}{t_2} \qquad \dfrac{\begin{array}{c}[t_2] \\ \vdots \\ t_3\end{array}}{\neg t_2, t_3}\;\text{subproof}}{t_3}\;\text{resolution}
$$

$$t_1 \vdash t_3$$

```
(assume a0   t1)
(step s1 (cl t2)
        :premises (a0)    :rule rule1)
(anchor :step s2)
  (assume s2.a1      t2)
  ...
  (step   s2.s10 (cl t3)
        :premises (s2.s9) :rule rule2)
(step s2 (cl (not t2) t3) :rule subproof)
(step s3 (cl t3)
        :premises (s1 s2) :rule resolution)
```

Alethe Proofs: Subproofs With Assumptions

$$\dfrac{\dfrac{t_1}{t_2} \qquad \dfrac{\begin{matrix}[t_2]\\ \vdots \\ t_3\end{matrix}}{\neg t_2, t_3}\,\text{subproof}}{t_3}\,\text{resolution}$$

$$t_1 \vdash t_3$$

```
(assume a0   t1)
(step s1 (cl t2)
        :premises (a0)    :rule rule1)
(anchor :step s2)
  (assume s2.a1      t2)
  ...
  (step   s2.s10 (cl t3)
        :premises (s2.s9) :rule rule2)
(step s2 (cl (not t2) t3) :rule subproof)
(step s3 (cl t3)
        :premises (s1 s2) :rule resolution)
```

$$\dfrac{\dfrac{\overline{x \mapsto y \rhd \quad x = y}\ \text{refl}}{x \mapsto y \rhd f(x) = f(y)}\ \text{cong}}{\forall x.\, f(x) = \forall y.\, f(y)}\ \text{bind}$$

$$\vdash \forall x.\, f(x) = \forall y.\, f(y)$$

```
(anchor :step s2 :args ((:= (x S) y)))
  (step s2.s1 (cl (= x y))     :rule refl)
  (step s2.s2 (cl (= (f x) (f y)))
                               :rule cong)
(step s2 (cl (= (forall ((x S)) (f x))
                (forall ((y S)) (f y))))
                               :rule bind)
```

Improving Alethe for Reconstruction

Important Hurdles Solved

- Clear term simplifications.
- No implicit clause normalizations.
- Certificates for linear arithmetic.

Improving Alethe for Reconstruction

Important Hurdles Solved

- Clear term simplifications.
- No implicit clause normalizations.
- Certificates for linear arithmetic.

Other Improvements

- **Complete documentation of the format.**
- Rigorous handling of quantifiers.
  - No implicit clausification.
  - $\forall$-instantiation certificate: explicit substitution.
- Proper printing of number constants depending on theory.
- A better algorithm for proof pruning.
- Clever term sharing.
- ...

Clear Term Simplifications

Can we improve proofs of preprocessing?

Clear Term Simplifications

Can we improve proofs of preprocessing?

Proofs

Before a single rule combining all simplifications, **undocumented**

$$\vDash_T \Gamma \rhd t = u$$

Now 17 rules arranged by operators. **Documented** as rewrite rules.
e.g. $x + 0 \rightarrow x$ in sum_simplify.

Clear Term Simplifications

Can we improve proofs of preprocessing?

Proofs

Before a single rule combining all simplifications, **undocumented**

$$\vDash_T \Gamma \triangleright t = u$$

Now 17 rules arranged by operators. **Documented** as rewrite rules.
e.g. $x + 0 \rightarrow x$ in sum_simplify.

Reconstruction

Before automatic proof tactics are necessary, with tweaked timeouts.

Now directed use of the simplifier parameterized with the rewrite rules.

No Implicit Clause Normalizations

Clauses in conclusions are sometimes simplified, why?

No Implicit Clause Normalizations

Clauses in conclusions are sometimes simplified, why?

Proofs

Before $\neg\neg\varphi$ implicitly simplified to φ **in the proof module**

Before clauses with complementary literals simplified to $\top$

Before repeated literals implicitly eliminated

Now patch every **proof step**, e.g, add step $\neg\neg\neg\varphi \vee \varphi$ and a resolution step

No Implicit Clause Normalizations

Clauses in conclusions are sometimes simplified, why?

Proofs

Before $\neg\neg\varphi$ implicitly simplified to φ **in the proof module**

Before clauses with complementary literals simplified to $\top$

Before repeated literals implicitly eliminated

Now patch every **proof step**, e.g, add step $\neg\neg\neg\varphi \lor \varphi$ and a resolution step

Reconstruction

Before special case possible at every step!

rule $(\textbf{if}\,\varphi\,\textbf{then}\,\psi_1\,\textbf{else}\,\psi_2) \Rightarrow \neg\varphi \lor \psi_1$

step $(\textbf{if}\,\varphi\,\textbf{then}\,\neg\varphi\,\textbf{else}\,\psi_2) \Rightarrow \neg\varphi$

Now no pollution in rule reconstruction.

step $(\textbf{if}\,\varphi\,\textbf{then}\,\neg\varphi\,\textbf{else}\,\psi_2) \Rightarrow \neg\varphi \lor \neg\varphi$

Reconstruction fails on this LA tautology: $(2x < 3) = (x \leq 1)$ over $\mathbb{Z}$

Why? Strengthening!

Certificates for Linear Arithmetic

Reconstruction fails on this LA tautology: $(2x < 3) = (x \leq 1)$ over $\mathbb{Z}$

Why? Strengthening!

Proofs

Before just a clause of inequalities, no certificate.

Now strengthening documented.

$$(2x < 3) = (x \leq 1)$$

Strengthened: $(2x \leq 2) = (x \leq 1)$

Now certificate: coefficient. Here: $\frac{1}{2}$ and 1.

Certificates for Linear Arithmetic

Reconstruction fails on this LA tautology: $(2x < 3) = (x \leq 1)$ over $\mathbb{Z}$
Why? Strengthening!

Proofs

Before just a clause of inequalities, no certificate.

Now strengthening documented.

$$(2x < 3) = (x \leq 1)$$

$$\text{Strengthened: } (2x \leq 2) = (x \leq 1)$$

Now certificate: coefficient. Here: $\frac{1}{2}$ and 1.

Reconstruction

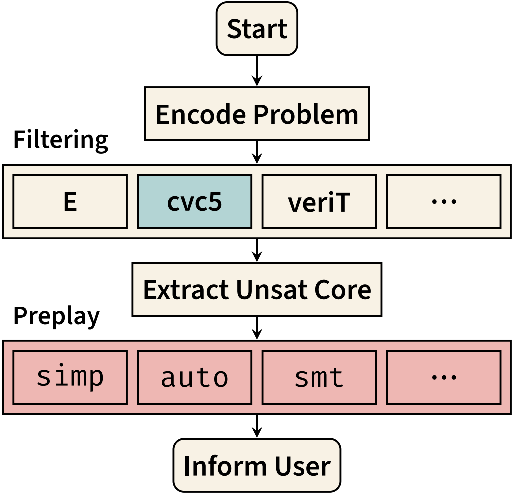
Before certificate derived again.

Now reconstruction amounts to calculations.

Now can abstract nested terms: $2 \times (\textbf{if} \top \textbf{ then } 1 \textbf{ else } 0)$ treated as $2 \times x$.

Evaluating `smt`

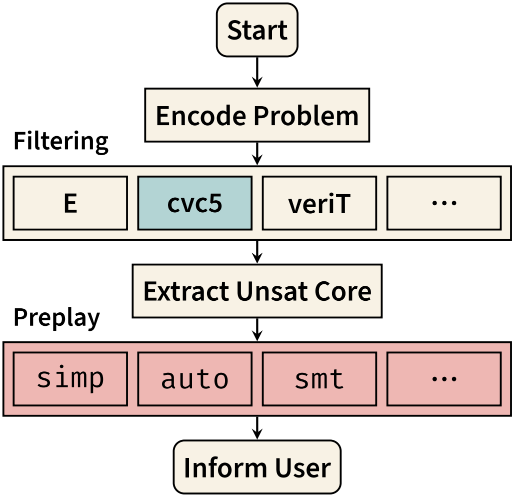
1. Pick an existing theory.
2. Try Sledgehammer on each obligation.



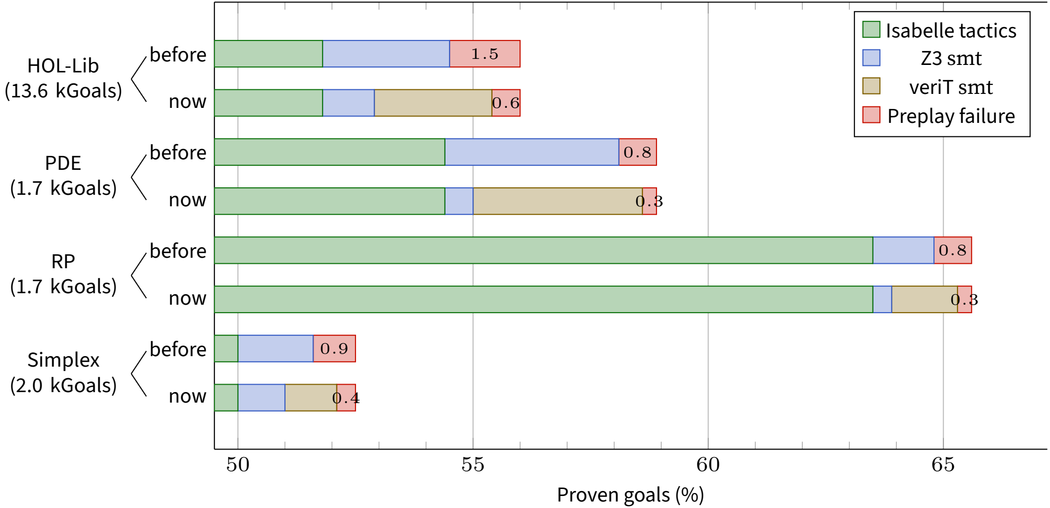
Evaluating `smt`

1. Pick an existing theory.
2. Try Sledgehammer on each obligation.

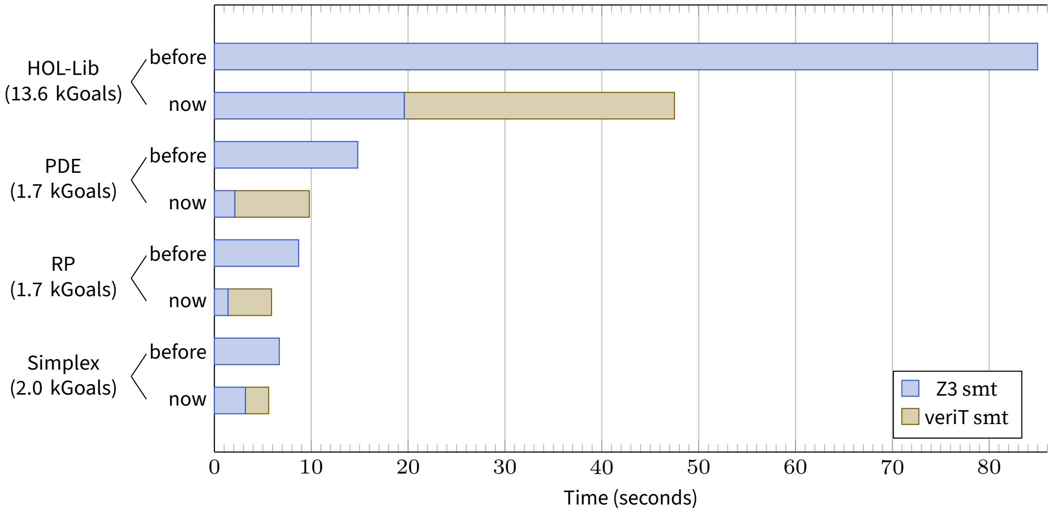
- Did Sledgehammer succeed?
- Which tactic did preplay suggest?
- Preplay failure: there is a proof, but it's not usable!
- Also: how long does the tactic run?



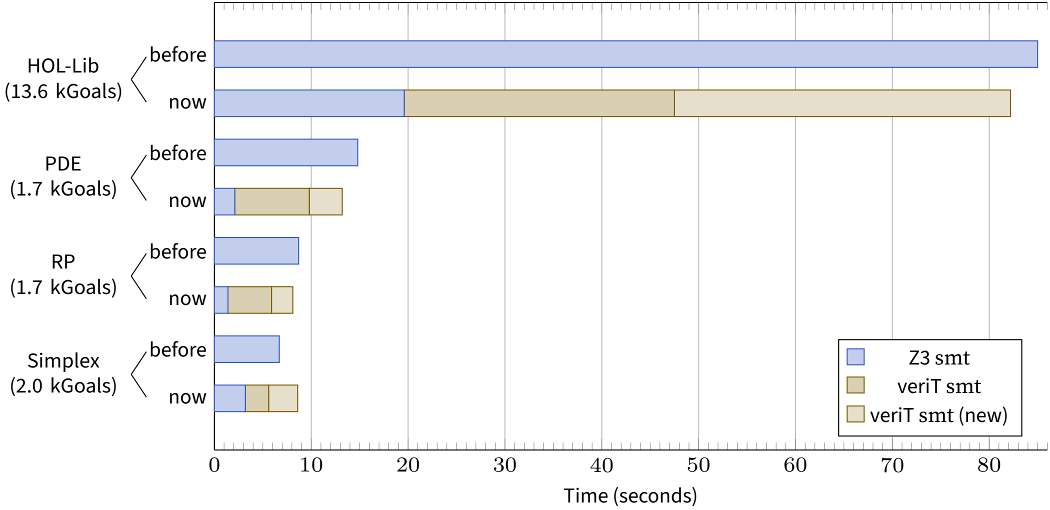
CVC4: Preplay Success Rate



CVC4: Preplay Time (smt only)



CVC4: Preplay Time (smt only)



Conclusion

Reconstruction

- 611 smt-veriT calls in AFP.
- Granular proofs matter.
- Proof size is critical.

SMT Proofs

- Danger of “Proof Rot.”
- Proof checking can prevent this.
- The familiar SMT-LIB syntax reduced debugging pain.
- Solver design leaks to the format.

Part III

The Future: AletheLF

What do we want?

Staying in Sync

- Documentation
- Proof production
- Proof checking

Designed for SMT Solvers

- Feels like using SMT-LIB
- Flexible enough to capture solver design details
- Fast!

What do we want?

Staying in Sync

- Documentation
- Proof production
- Proof checking

Designed for SMT Solvers

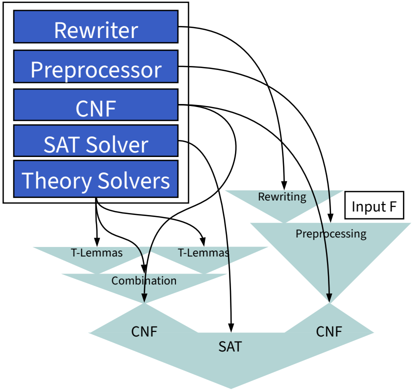
- Feels like using SMT-LIB
- Flexible enough to capture solver design details
- Fast!

Integrated in cvc5!

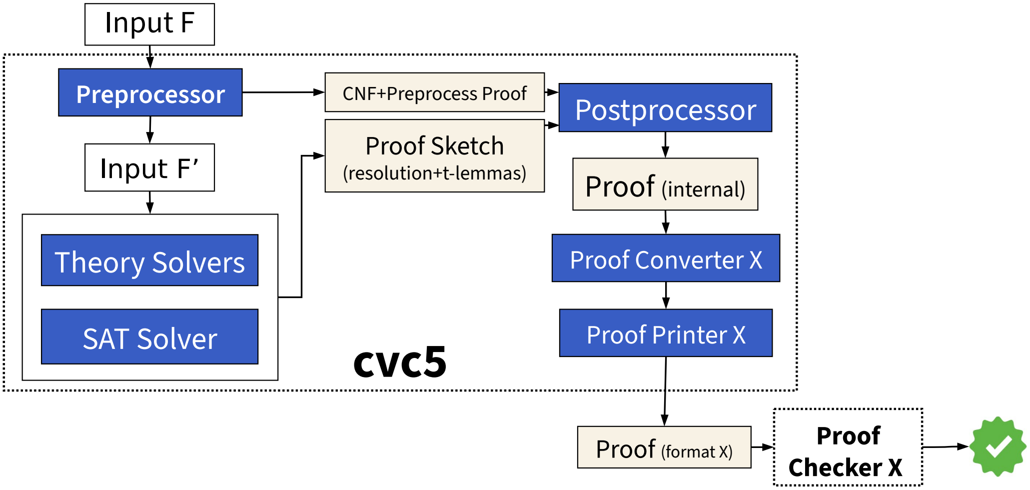
Proofs in cvc5

- Internal proof calculus
  - Roughly 142 rules (132 core + 10 macro)
  - Native proof checker in cvc5's core
  - Original focus was on theory of strings
  - Multiple backends
- Evaluated on many SMT-LIB theories

  [Barbosa, et al. 2022]



Proofs in cvc5

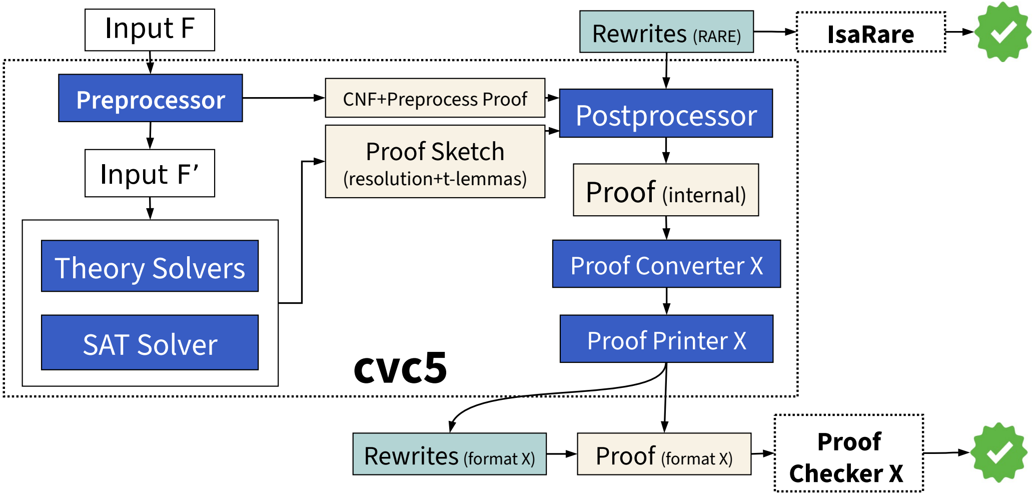


RARE: a DSL for Proofs of Rewrites

- Key element of preprocessing: **rewriting**
- A DSL to express rewrite rules
- Automatic elaboration during post-processing
- Large library of rewrites (strings, bitvectors, ...)
- Translation pipeline to Isabelle/HOL

```
(define-rule* str-concat-unify
 ((s1 String)
  (s2 String) (s3 String :list)
  (t2 String) (t3 String :list))
  (= (str.++ s1 s2 s3)
     (str.++ s1 t2 t3))
  (= (str.++    s2 s3)
     (str.++    t2 t3)))
```

Proofs with RARE in cvc5



Why not Alethe?

- ⛔ A pen-and-paper standard: danger of proof rot
- ⛔ Limited set of theories.
- ⛔ No rule to proof testing pipeline.
- ⛔ Does not capture cvc5's type system, proof calculus
  - incurs large post-processing cost

Why not LFSC?

What is LFSC?

- Dedicated logical framework (LF) for SMT proofs. [Oe, et al. 2009], [Stump, et al. 2013], [Hadarean, et al. 2015], [Katz, et al. 2016]
- Based on Edinburg Logical Framework (LF) extended with side conditions.
- Allows user defined proof rules.

⛔ Performance.

⛔ Proof rules must encoded at a low level.

⛔ Syntax for terms does not match SMT-LIB.

⛔ Limited tooling support.

Why not LFSC?

What is LFSC?

- Dedicated logical framework (LF) for SMT proofs. [Oe, et al. 2009], [Stump, et al. 2013], [Hadarean, et al. 2015], [Katz, et al. 2016]
- Based on Edinburg Logical Framework (LF) extended with side conditions.
- Allows user defined proof rules.

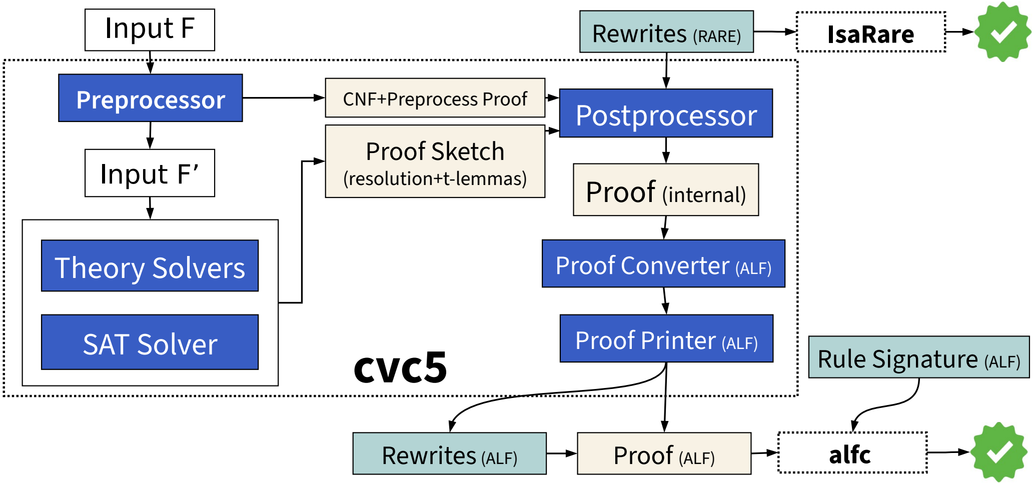
⛔ Performance.

⛔ Proof rules must encoded at a low level.

⛔ Syntax for terms does not match SMT-LIB.

⛔ Limited tooling support.

AletheLF:
Alethe x LFSC


Input F
Rewrites (RARE)
IsaRare
Preprocessor
CNF+Preprocess Proof
Postprocessor
Proof Sketch
(resolution+t-lemmas)
Input F’
Proof (internal)
Theory Solvers
Proof Converter (ALF)
SAT Solver
Proof Printer (ALF)
cvc5
Rule Signature (ALF)
Rewrites (ALF)
Proof (ALF)
alfc

Example: Symmetry of Equality

```
(declare-const = (-> (! Type :var T :implicit) T T Bool))
(declare-rule symm ((T Type) (t T) (s T))
    :premises ((= t s))
    :conclusion (= s t)
)

(declare-sort S 0)
(declare-const a S)
(declare-const b S)

(assume @a0 (= a b))
(step @s1 (= b a) :rule symm :premises (@a0))
```

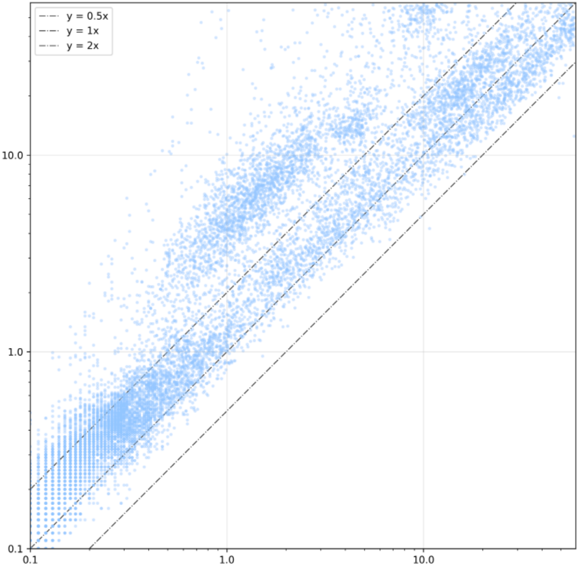
Example: And-Elimination

```
(declare-const and (-> Bool Bool Bool))
(program select ((i Int) (l Bool) (r Bool))
        (Int Bool) Bool
        (
                ((select 1 (and l r)) l)
                ((select 2 (and l r)) r)
        )
)
(declare-rule and-elim ((l Bool) (r Bool) (i Int))
    :premises ((and l r))
    :args (i)
    :conclusion (select i (and l r))
)

(declare-const p Bool)
(declare-const q Bool)

(assume @a0 (and p q))
(step @s1 q :rule and-elim :premises (@a0) :args (2))
```

Evaluation: AletheLF vs. LFSC



- 97348 benchmarks
- 60s timeout
- All quantifier-free SMT-LIB logics with
  - strings
  - linear arithmetic
  - uninterpreted functions
- alfc 1.56x faster
  - Due to flexibility in AletheLF (e.g. uses chain resolution)

What about SAT proofs?

Why do we care?

- Proof production in for SAT solvers has been successful:
  - DRAT is everywhere!
- cvc5 now uses configurable SAT solver
  - via the IPASIR-UP API (IPASIR with user propagators)
  - Notably, cvc5 supports CaDiCaL

What about SAT proofs?

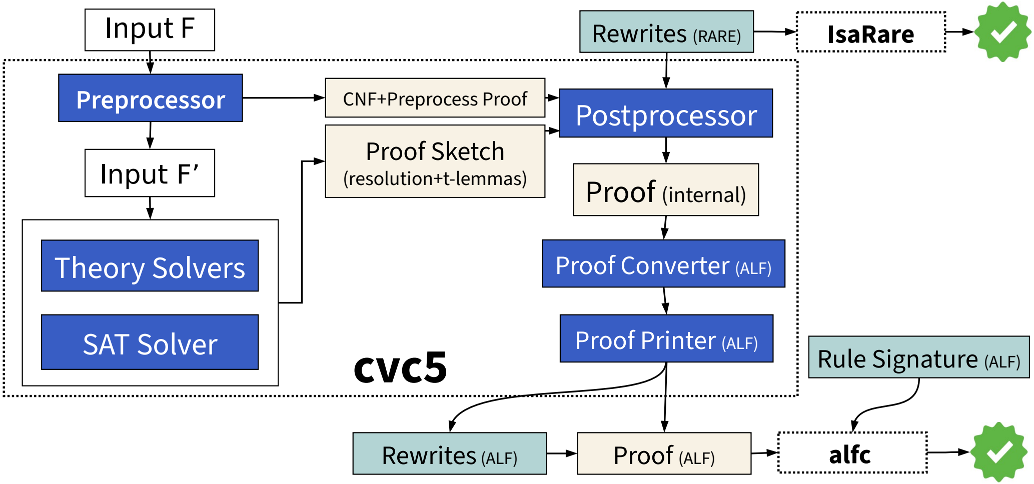
Why do we care?

- Proof production in for SAT solvers has been successful:
  - DRAT is everywhere!
- cvc5 now uses configurable SAT solver
  - via the IPASIR-UP API (IPASIR with user propagators)
  - Notably, cvc5 supports CaDiCaL

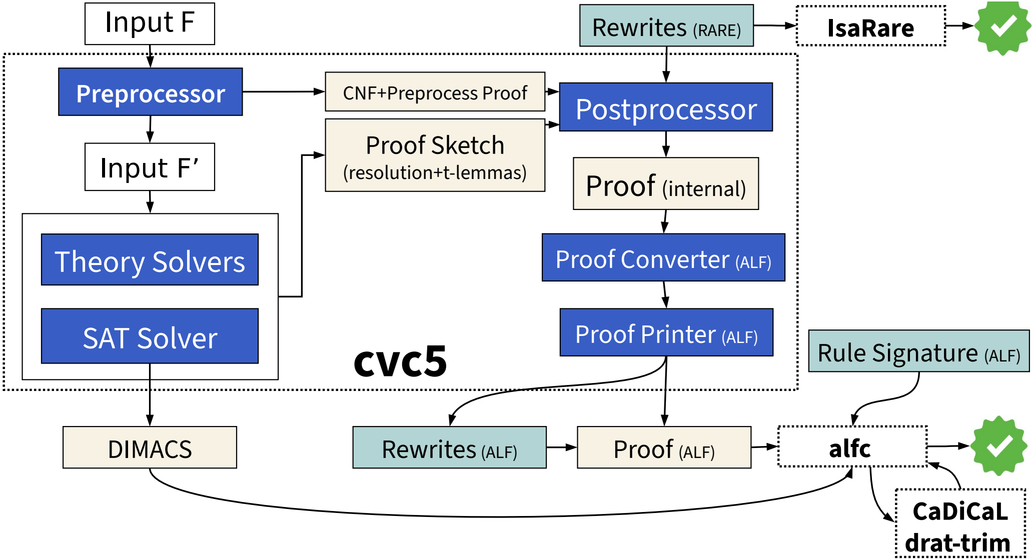
In AletheLF

- Integrate via **oracles**
- Use `declare-oracle-fun` to declare an interface with an external program.
- Communication using SMT-LIB syntax.

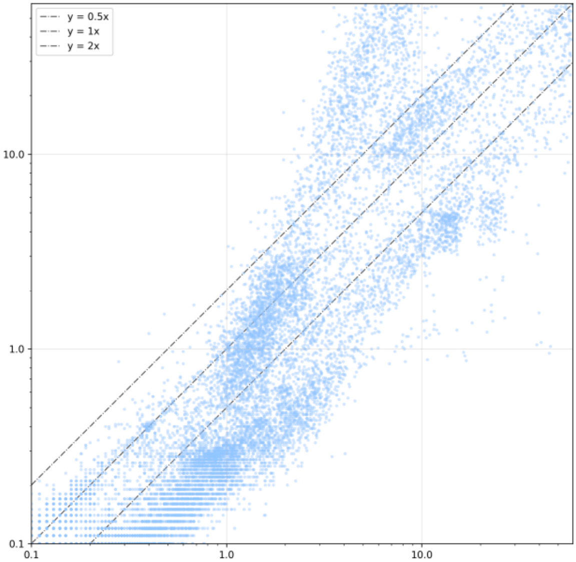
Proofs from cvc5 with SAT proofs



Proofs from cvc5 with SAT proofs

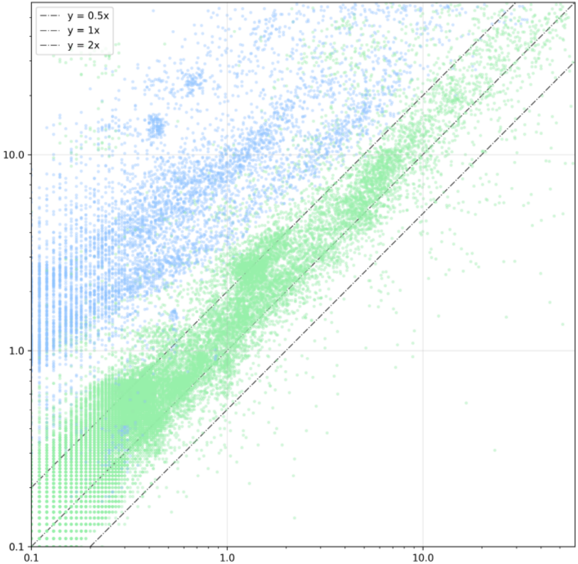


Evaluation: AletheLF with DRAT vs. resolution

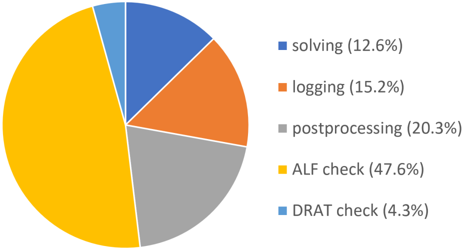


- DRAT scales better than res on harder examples
  - DRAT 1.34x faster for benchmarks >5s

Evaluation: Proof Overhead



- 7.9x overhead solving+proof checking (unsat)
  - 1.4x for SAT problems

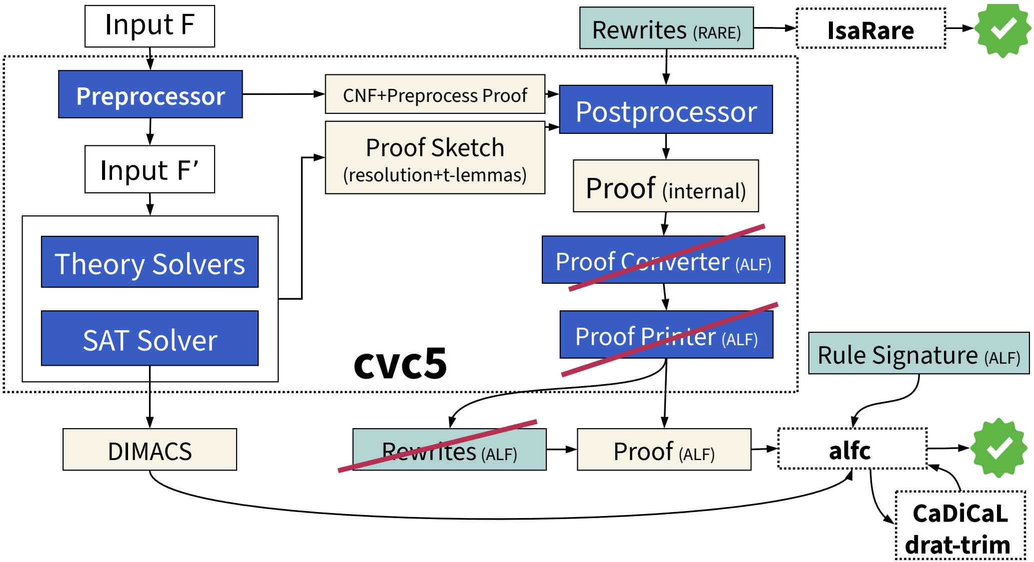


The Current Status

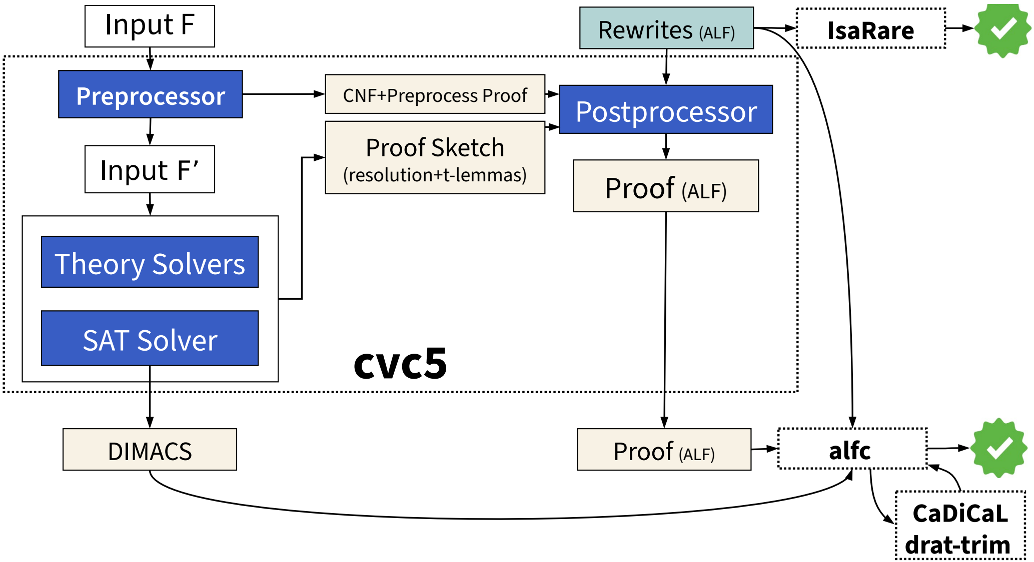
Ongoing Work

- Polish alfc for release.
- Use AletheLF as the internal format for cvc5 proofs.

AletheLF for internal proofs?



AletheLF for internal proofs?



The Current Status

Ongoing Work

- Polish alfc for release.
- Use AletheLF as the internal format for cvc5 proofs.
- Express the Alethe calculus in AletheLF.
- Formalize the core of AletheLF in Agda.

The Current Status

Ongoing Work

- Polish alfc for release.
- Use AletheLF as the internal format for cvc5 proofs.
- Express the Alethe calculus in AletheLF.
- Formalize the core of AletheLF in Agda.

Try it!

- cvc5: `https://cvc5.github.io`
  - use `--dump-proofs --proof-format=alf`
- alfc with documentation: `https://github.com/cvc5/alfc`
- Alethe in AletheLF: `https://github.com/cvc5/AletheInAlf`

Thank You!