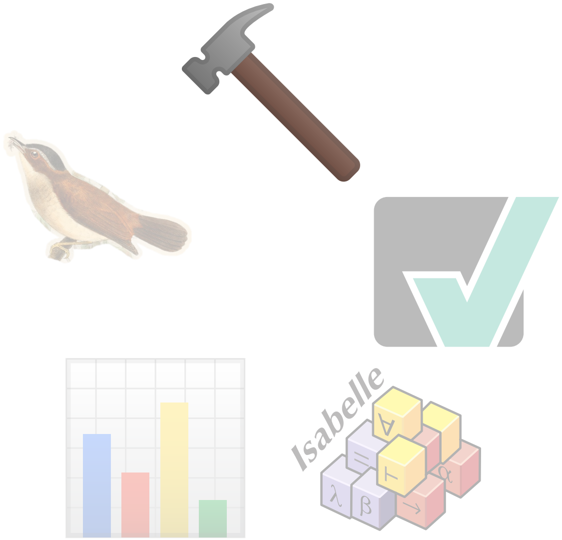
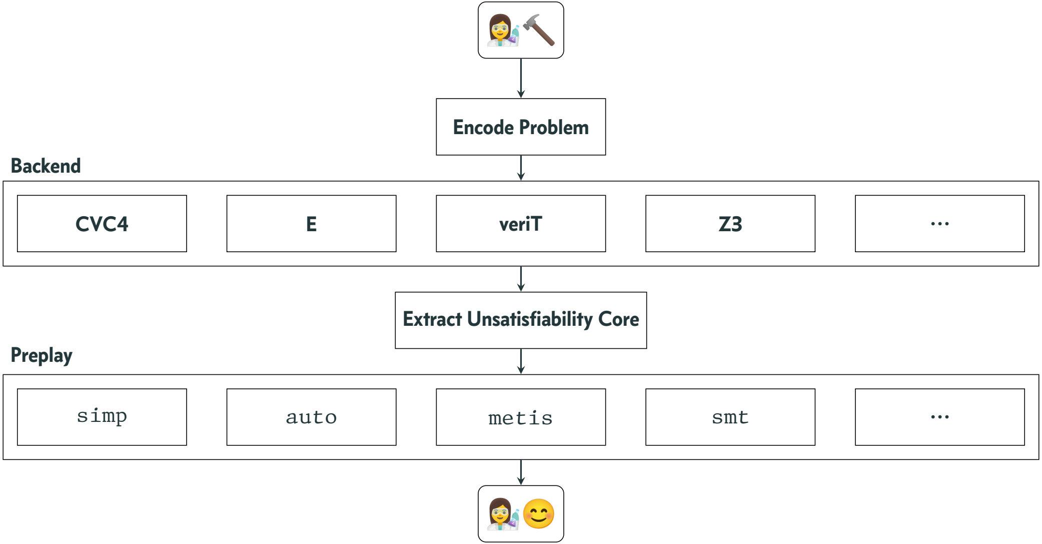
Reliable Reconstruction of Fine-Grained Proofs in a Proof Assistant

Hans-Jörg Schurr Mathias Fleury Martin Desharnais

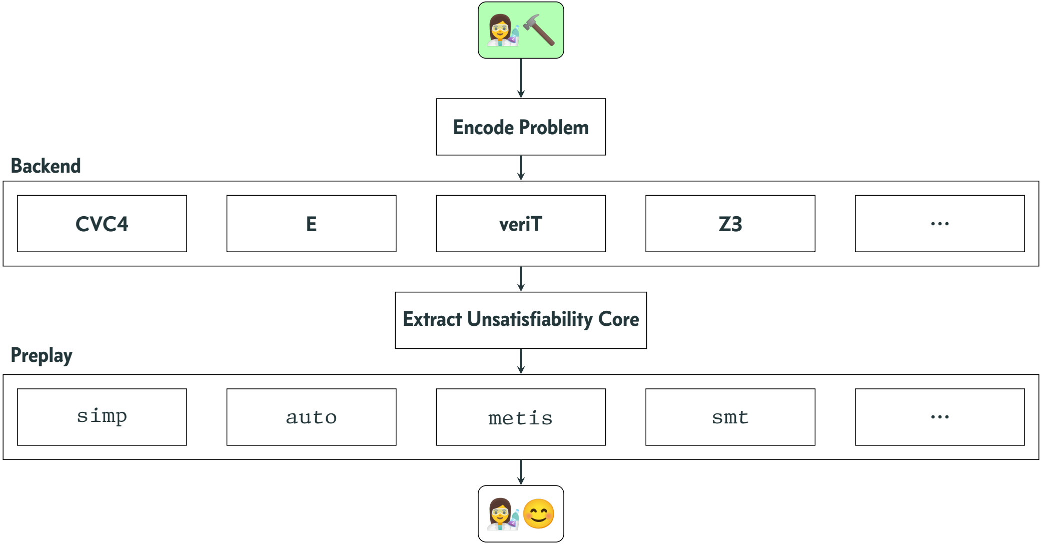
SMT 2021 (and CADE'28)


Isabelle

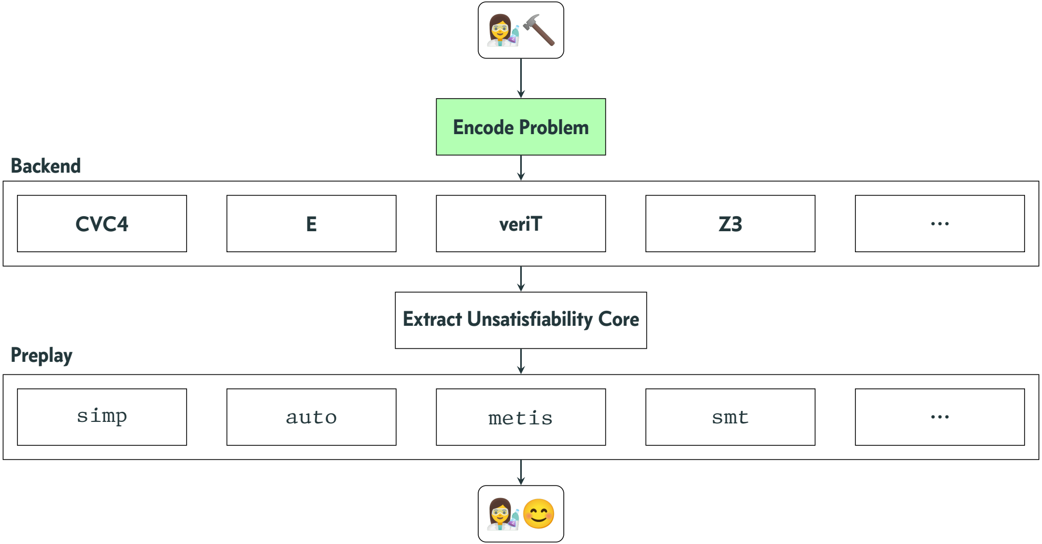
Interactive Theorem Proving with Sledgehammer



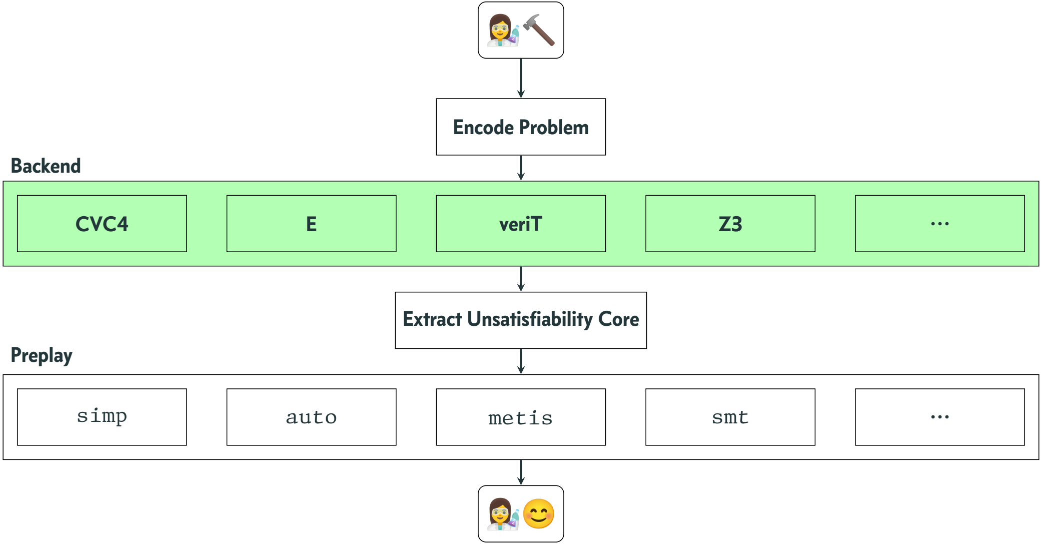
Interactive Theorem Proving with Sledgehammer



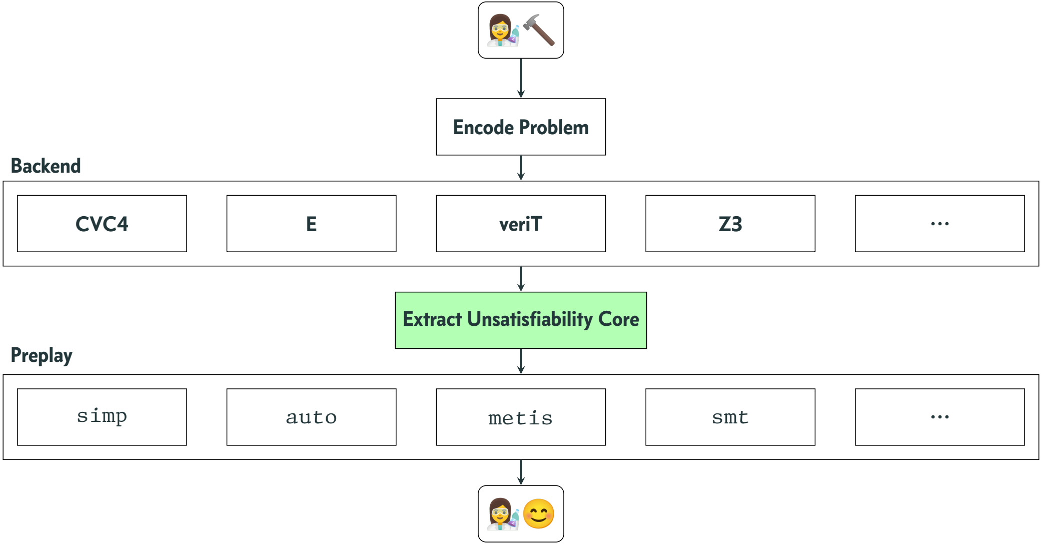
Interactive Theorem Proving with Sledgehammer



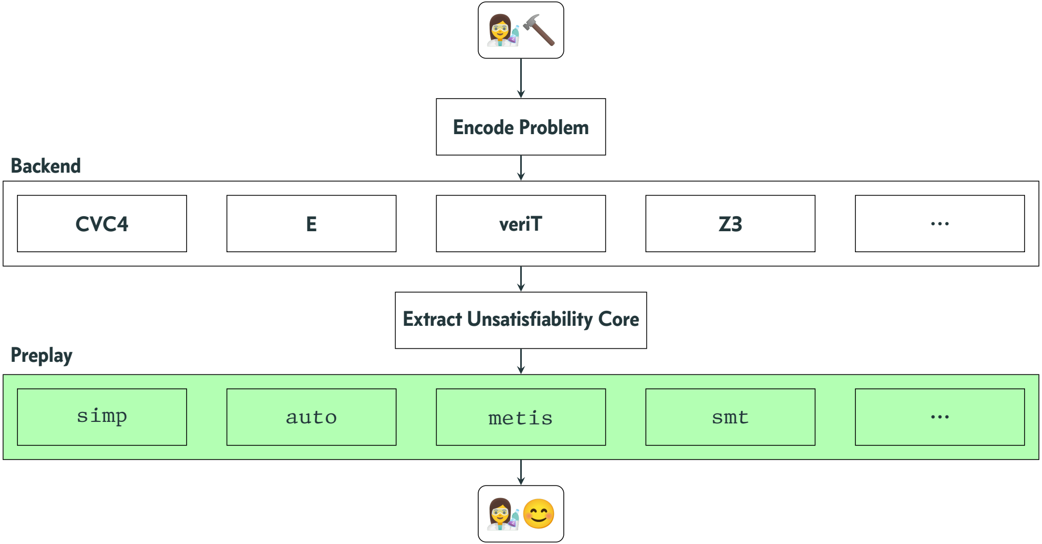
Interactive Theorem Proving with Sledgehammer



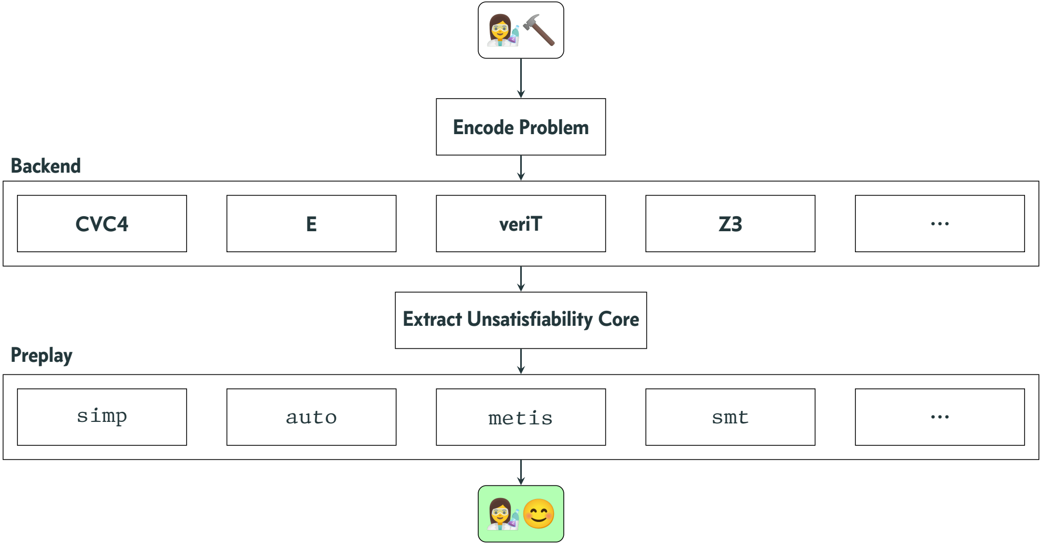
Interactive Theorem Proving with Sledgehammer



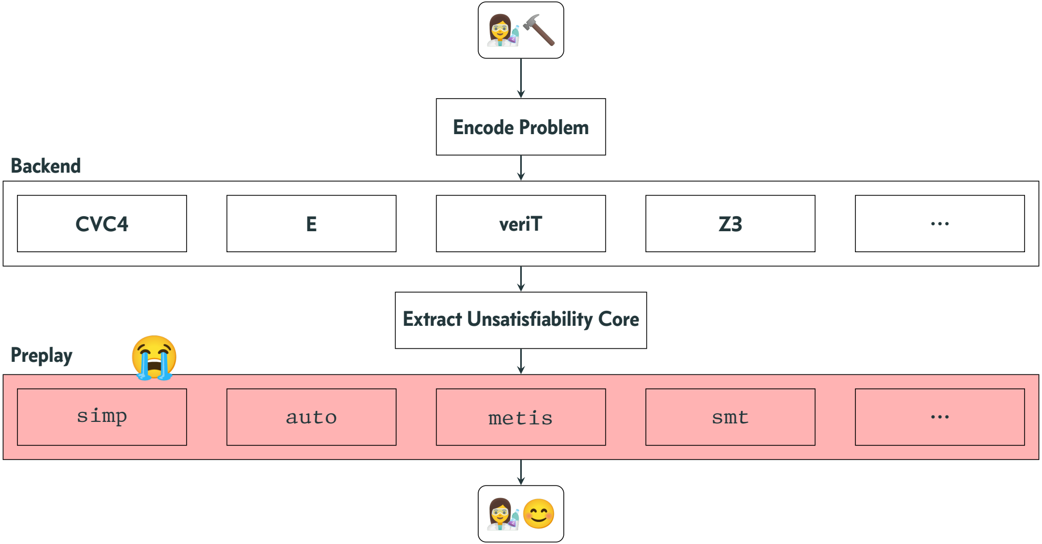
Interactive Theorem Proving with Sledgehammer



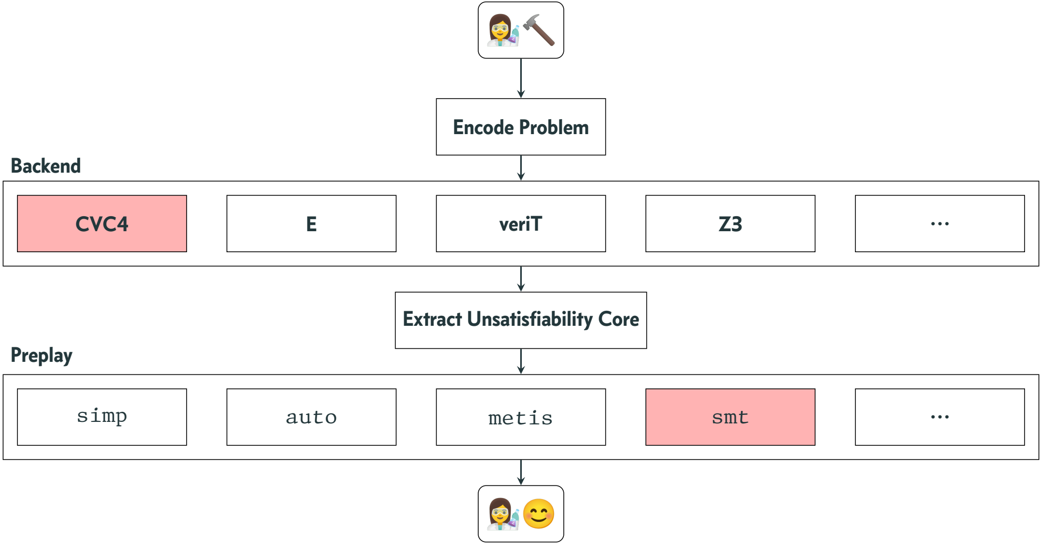
Interactive Theorem Proving with Sledgehammer



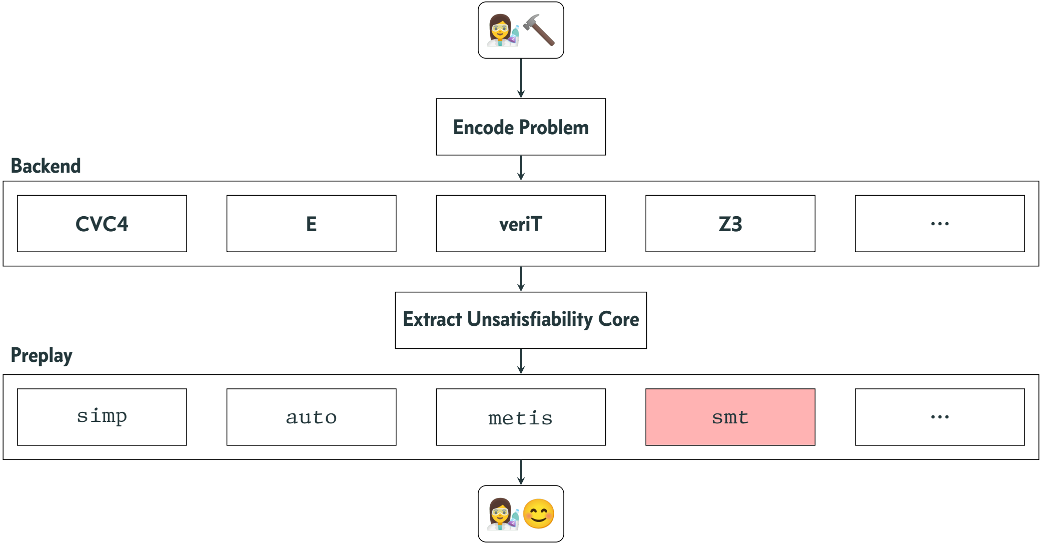
Interactive Theorem Proving with Sledgehammer



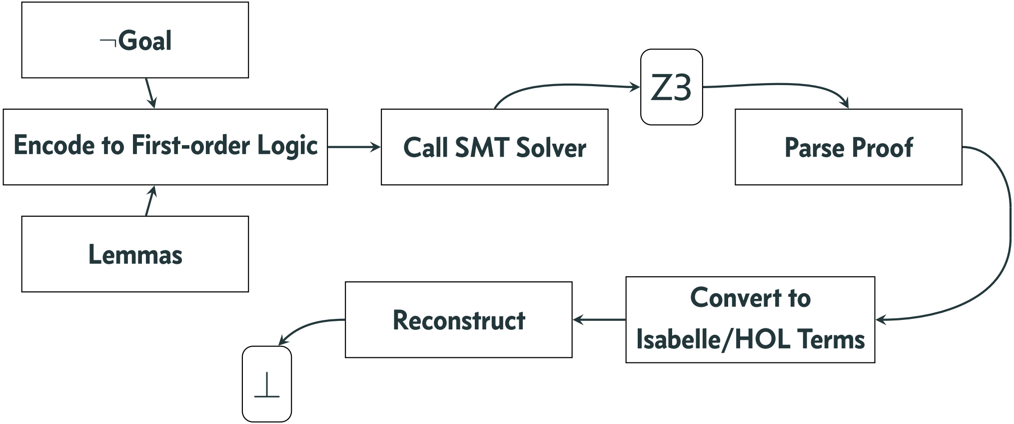
Interactive Theorem Proving with Sledgehammer



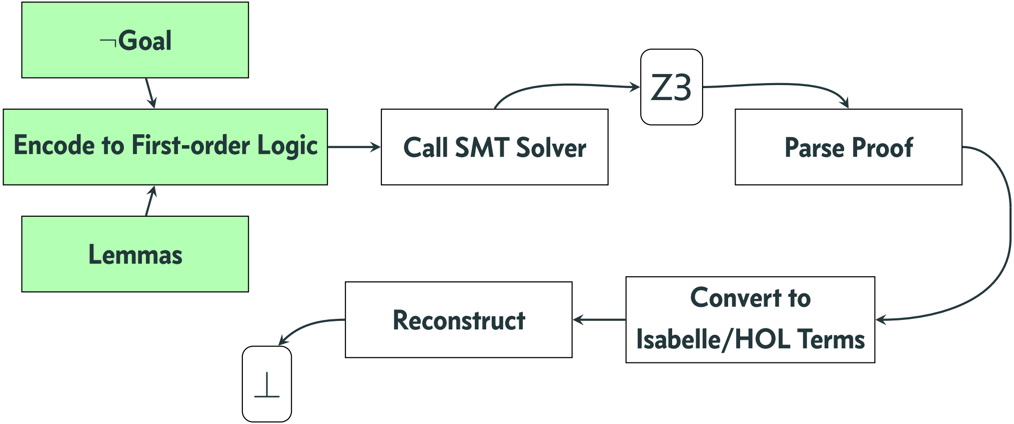
Interactive Theorem Proving with Sledgehammer



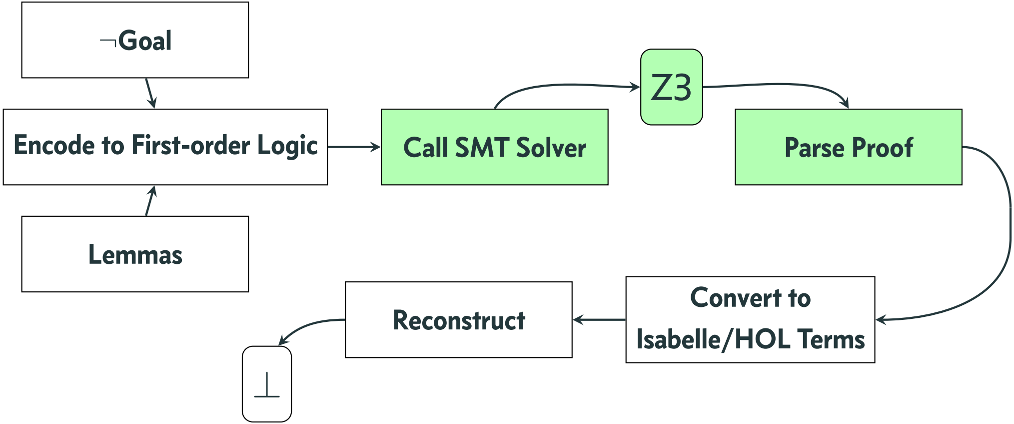
Inside the `smt` tactic



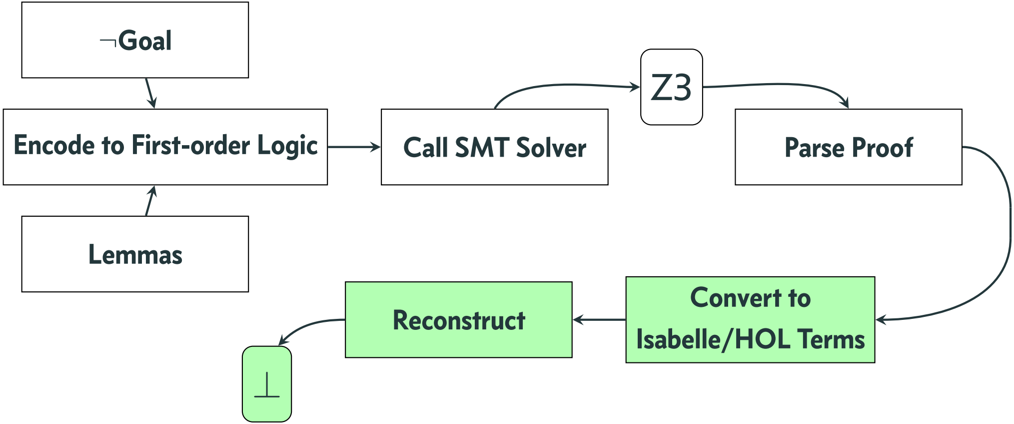
Inside the smt tactic



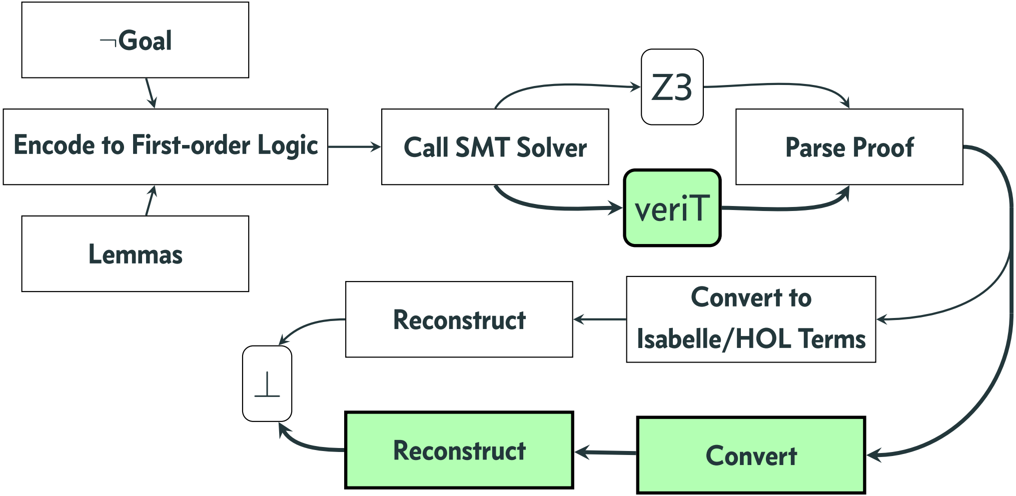
Inside the smt tactic



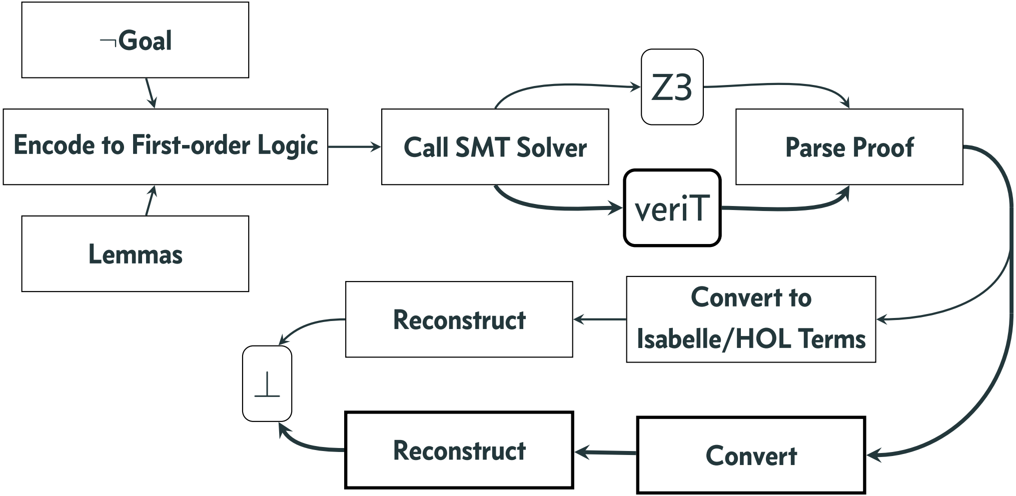
Inside the `smt` tactic



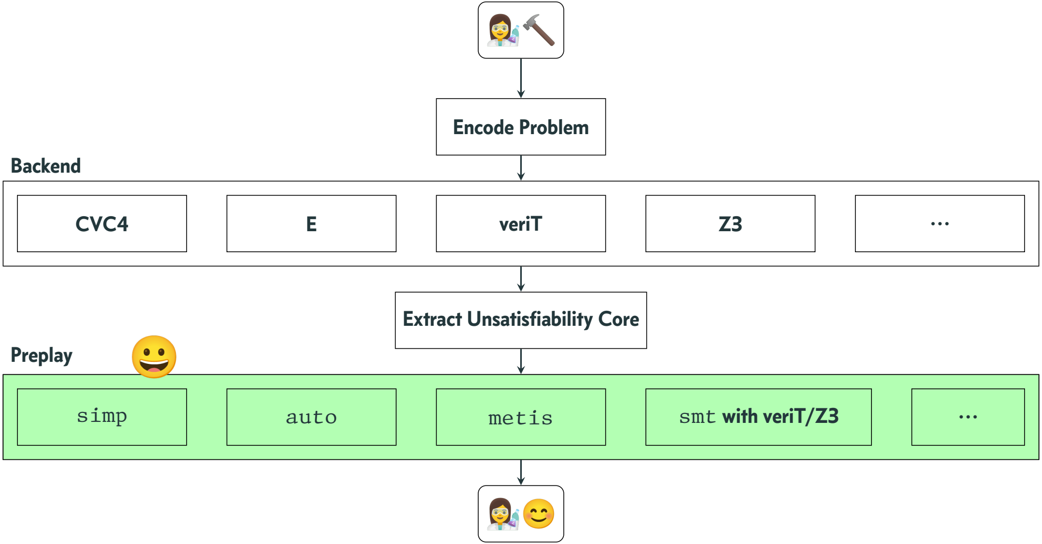
Inside the smt tactic

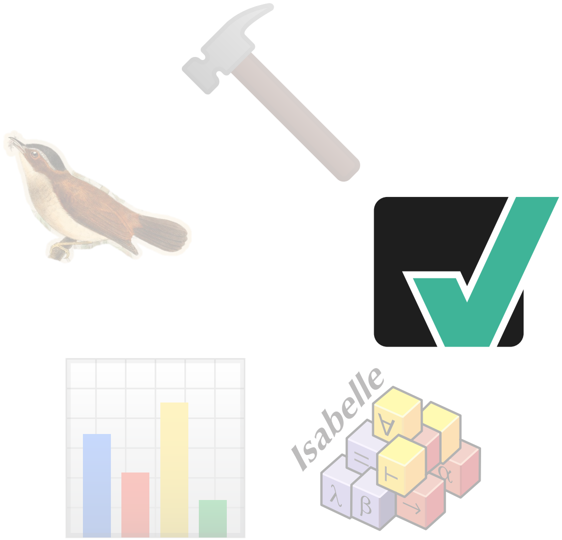


Inside the smt tactic



Interactive Theorem Proving with Sledgehammer




Isabelle

veriT

- Traditional CDCL(T) solver
- Supports:
  - Uninterpreted functions
  - Linear arithmetic
  - Quantifiers
  - …
- SMT-LIB input
- Lightweight
- BSD Licence
- Quantifier instantiation:
  - Conflicting instances
  - Trigger-based instantiation
  - Enumerative instantiation
- Proofs
  - Fine-grained
  - Proofs for transformations below quantifiers
  - Alethe output

Simplifications

Can the simplification rule be more fine grained?

Before single rule combining all simplifications, undocumented

Now one rule per transformation with a semantic 17 different rules

Before automatic proof tactics like `auto`, with known timeouts

Now directed applications of the simplifier
along `simp only: plus_simps`

Simplifications

Can the simplification rule be more fine grained?

Before single rule combining all simplifications, undocumented

Now one rule per transformation with a semantic 17 different rules

Before automatic proof tactics like `auto`, with known timeouts

Now directed applications of the simplifier along `simp only: plus_simps`

Simplifications

Can the simplification rule be more fine grained?

Before single rule combining all simplifications, undocumented

Now one rule per transformation with a semantic — 17 different rules

Before automatic proof tactics like `auto`, with known timeouts

Now directed applications of the simplifier
along `simp only: plus_simps`

Implicit Normalizations

Clauses like tautologies are simplified, why?

Before $\neg\neg t$ implicitly simplified to t in the solver

Before clauses with complementary literals simplified to $\top$

Before repeated literals implicitly eliminated

After patch the proof with, e.g, a step $\neg\neg\neg t \vee t$ and a resolution step

Before special case for every step!

Now no pollution in rule reconstruction
`(if` P `then` Q `else` R`) implies` $\neg P \vee Q$

Implicit Normalizations

Clauses like tautologies are simplified, why?

Before $\neg\neg t$ implicitly simplified to t in the solver

Before clauses with complementary literals simplified to $\top$

Before repeated literals implicitly eliminated

After patch the proof with, e.g, a step $\neg\neg\neg t \vee t$ and a resolution step

Before special case for every step!

Now no pollution in rule reconstruction

`(if` P `then` Q `else` R`) implies` $\neg P \vee Q$

Implicit Normalizations

Clauses like tautologies are simplified, why?

Before $\neg\neg t$ implicitly simplified to t in the solver

Before clauses with complementary literals simplified to $\top$

Before repeated literals implicitly eliminated

After patch the proof with, e.g, a step $\neg\neg\neg t \vee t$ and a resolution step

Before special case for every step!

Now no pollution in rule reconstruction

(if P then Q else R) implies $\neg P \vee Q$

Reconstructing Arithmetic

Isabelle fails on this LA tautology: $2x < 3 \leftrightarrow x \leq 1$ over $\mathbb{Z}$

Why? Strengthening!

Before no witness

Now witness in the proof, e.g., $1 / 2$

Now even typed witness

Before witness (Farkas's coefficients) derived again

Now reconstruction of the LA solver...

Now ... with same visibility $2 * \texttt{if True then 1 else 0}$

Reconstructing Arithmetic

Isabelle fails on this LA tautology: $2x < 3 \leftrightarrow x \leq 1$ over $\mathbb{Z}$

Why? Strengthening!

Before no witness

Now witness in the proof, e.g., $1 / 2$

Now even typed witness

Before witness (Farkas's coefficients) derived again

Now reconstruction of the LA solver...

Now ... with same visibility $2 * \texttt{if True then 1 else 0}$

Reconstructing Arithmetic

Isabelle fails on this LA tautology: $2x < 3 \leftrightarrow x \leq 1$ over $\mathbb{Z}$

Why? Strengthening!

Before	no witness
Now	witness in the proof, e.g., $1/2$
Now	even typed witness

Before	witness (Farkas's coefficients) derived again
Now	reconstruction of the LA solver…
Now	… with same visibility $2 *$ `if True then 1 else 0`

Step Skipping

Can we do better by understanding proofs globally?

- veriT normalizes every name `x` to `veriT_vr42` with a proof.
  But: $(\forall x.\, P\ x) = (\forall \mathrm{veriT\_vr42}.\, P\ \mathrm{veriT\_vr42})$ for Isabelle — De Brujn indices
  So: remove subproof.

- detect $P \neq Q \lor \neg P \lor Q$, $P = Q$, P implies Q. — used for every normalization pattern
  So: remove one step and specialize resolution step.
  But: conclusion of step must be known.

Both important for quantifiers — Skolemization: $\geq$ 8 to 3 steps

Step Skipping

Can we do better by understanding proofs globally?

- veriT normalizes every name `x` to `veriT_vr42` with a proof.
  But: $(\forall x.\, P\ x) = (\forall \mathrm{veriT\_vr42}.\, P\ \mathrm{veriT\_vr42})$ for Isabelle
  So: remove subproof.

De Brujn indices

- detect $P \neq Q \vee \neg P \vee Q, \quad P = Q, \quad P \quad$ implies Q.
  So: remove one step and specialize resolution step.
  But: conclusion of step must be known.

used for every normalization pattern

Both important for quantifiers

Skolemization: $\geq$ 8 to 3 steps

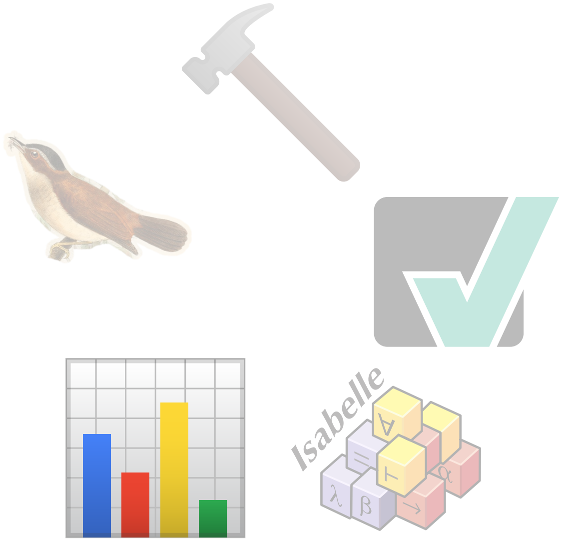
Step Skipping

Can we do better by understanding proofs globally?

- veriT normalizes every name `x` to `veriT_vr42` with a proof.
  But: $(\forall x.\, P\ x) = (\forall \mathrm{veriT\_vr42}.\, P\ \mathrm{veriT\_vr42})$ for Isabelle — De Brujn indices
  So: remove subproof.

- detect $P \neq Q \vee \neg P \vee Q, \quad P = Q, \quad P \quad$ implies Q. — used for every normalization pattern
  So: remove one step and specialize resolution step.
  But: conclusion of step must be known.

Both important for quantifiers — Skolemization: ≥ 8 to 3 steps


Isabelle

Mirabelle

Automatic tool to test Sledgehammer:

- calls Sledgehammer on all possible goals
- can produce the SMT files corresponding to the goals

Three outcomes for Sledgehammer/Mirabelle:

1. the backend found a proof and preplay worked 😊
2. the backend found a proof but preplay failed 😭
3. the backend did not find a proof our job cannot be fully automated!

Strategy Selection

veriT is highly configurable! Can we do better than the default strategy?

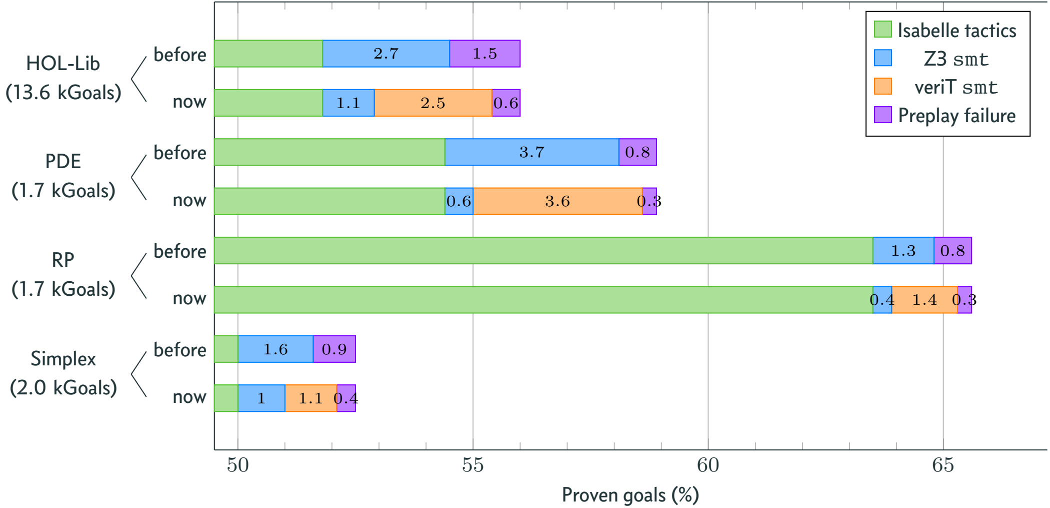
We found four strategies:

- the overall best
- three complementary strategies

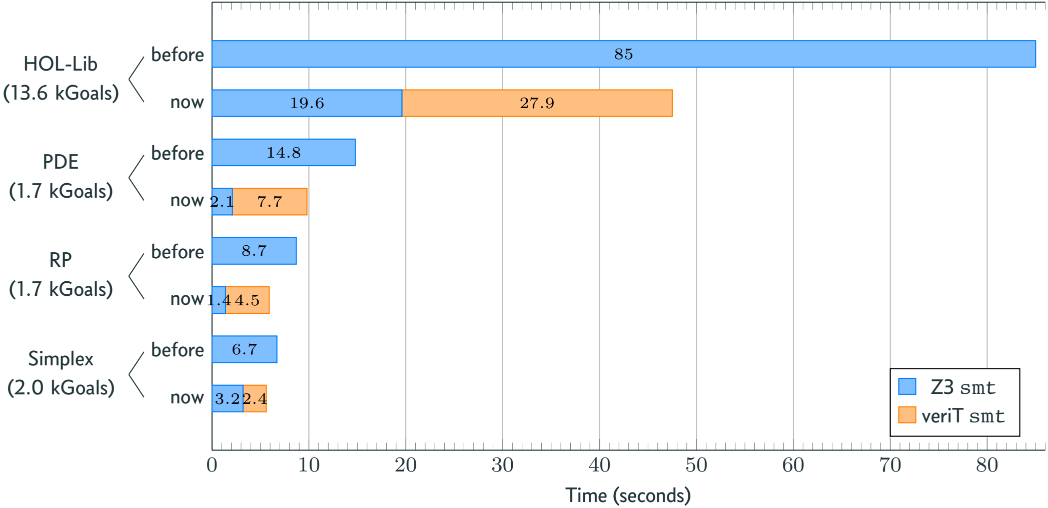
instantiation strategy varies

But: no scheduling in veriT `smt`, instead all tried during preplay.

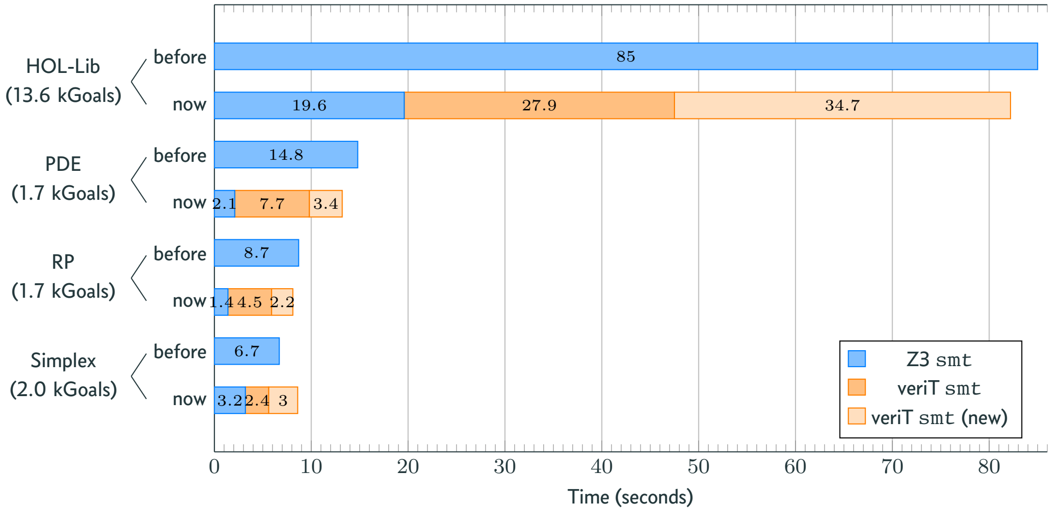
CVC4: Preplay Success Rate



CVC4: Preplay Time (smt only)



CVC4: Preplay Time (smt only)



Isabelle

Alethe Proof Format

Key elements:

- natural-deduction style
- avoids repetition — let-binding not expanded
- fine-grained quantifier reasoning — skolemization via Hilbert choice
- follows SMT-LIB when possible — S-expressions, commands, and annotations

Key idea: stack with context — Barbosa et al. CADE'26 and JAR'20

$$\frac{x = y \blacktriangleright P\,x = Q\,y}{\blacktriangleright (\forall x.\ P\ x) = (\forall y.\ P\ y)}$$

Alethe Proof Format

```
(assume a0 (exists ((x A)) (f x)))
(anchor :step t1 :args (:= x vr))
(step t1.t1 (cl (= x vr))                                           :rule cong)
(step t1.t2 (cl (= (f x) (f vr)))                                   :rule cong)
(step t1 (cl (= (exists ((x A)) (f x)) (exists ((vr A)) (f vr)))) :rule bind)
(step t2 (cl (not (= (exists ((vr A)) (f x)) (exists ((vr A)) (f vr))))
             (not (exists ((vr A)) (f x)))
             (exists ((vr A)) (f vr)))                              :rule equiv_pos1)
(step t3 (cl (exists ((vr A)) (f vr))) :premises (a0 t1 t2) :rule resolution)
(define-fun X () A (choice ((vr A)) (f vr)))
(step t4 (cl (= (exists ((vr A)) (f vr)) (f X)))                    :rule sko_ex)
(step t5 (cl (not (= (exists ((vr A)) (f vr)) (f X)))
             (not (exists ((vr A)) (f vr)))   (f X))                :rule equiv_pos1)
(step t6 (cl (f X))                   :premises (t3 t4 t5) :rule resolution)
```

Part of veriT. Ongoing work for inclusion in cvc5, formal specification, and standalone proof checker.

More details in our PxTP'21 talk

We can now reconstruct veriT proofs...

... as a user, just profit:

- part of Isabelle 2021
- improved Sledgehammer performance
- already 141 calls in the Archive of Formal Proofs 718cb448a456

... as a developper (futur work):

- wider support for `smt`
- better Isar proofs

We can now reconstruct veriT proofs...

... as a user, just profit:

- part of Isabelle 2021
- improved Sledg
- already 141 call

718cb448a456

... as a developper (f

- wider support f
- better Isar proofs

We can now reconstruct veriT proofs…

… as a user, just profit:

- part of Isabelle 2021
- improved Sledgehammer performance
- already 141 calls in the Archive of Formal Proofs 718cb448a456

… as a developper (futur work):

- wider support for `smt`
- better Isar proofs

CVC4 Results

	HOL-Library (13 562 goals)				PDE (1 715 goals)				RP (1 658 goals)				Simplex (1 982 goals)			
	SR	OL_v	OL_z	PF	SR	OL_v	OL_z	PF	SR	OL_v	OL_z	PF	SR	OL_v	OL_z	PF
Fact-filter prover: CVC4																
z-smt	54.5		2.7	1.5	33.1		3.7	0.8	64.8		1.3	0.8	51.6		1.6	0.9
v-smt+z-smt	55.5	2.5	1.1	0.5	33.6	3.6	0.6	0.3	65.3	1.4	0.4	0.3	52.1	1.1	1.0	0.4