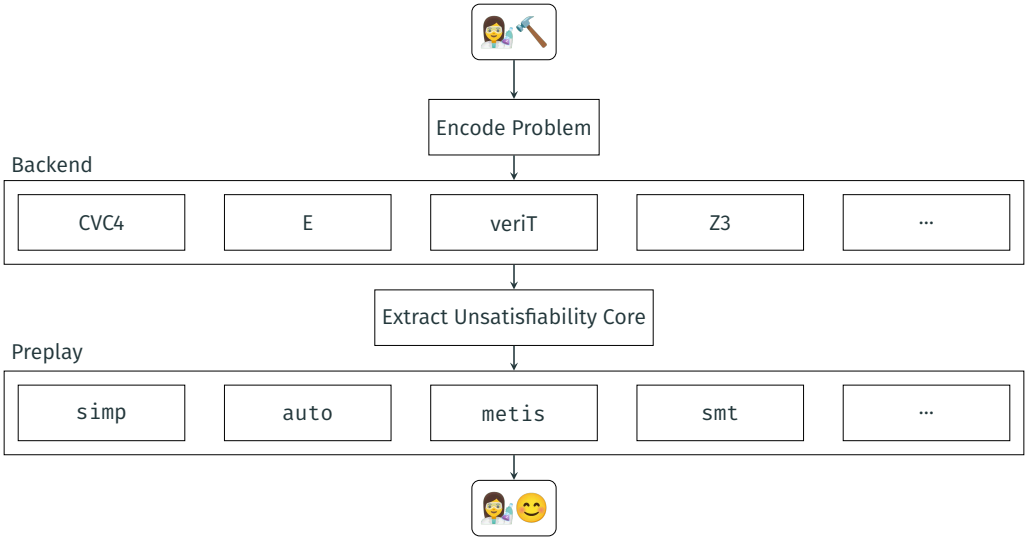
Reliable Reconstruction of Fine-Grained Proofs in a Proof Assistant

Bruno Andreotti Mathias Fleury

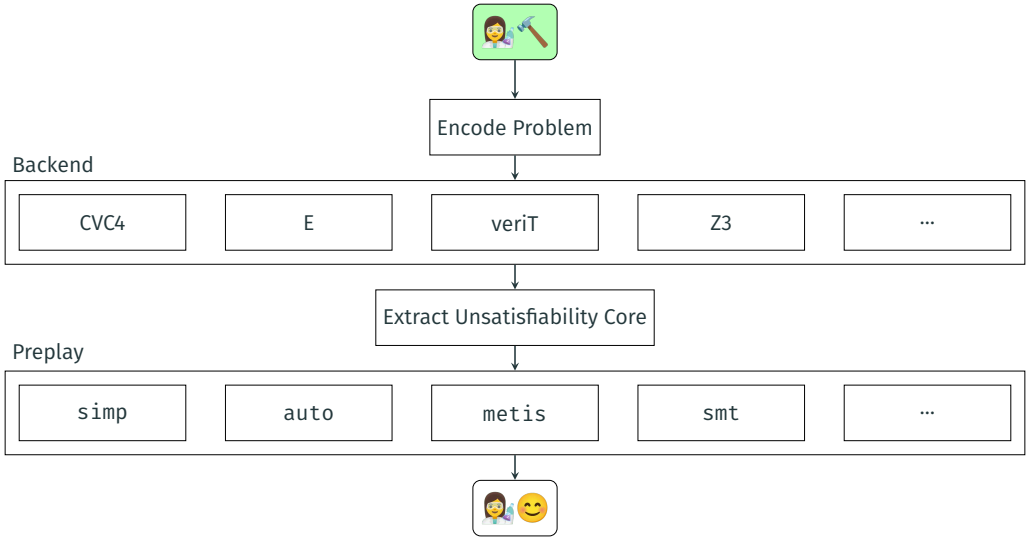
Joint work with Hanna Lachnitt, Martin Desharnais, Hans-Jörg Schurr, Haniel Barbosa, Jasmin Blanchette, Pascal Fontaine

Introduction

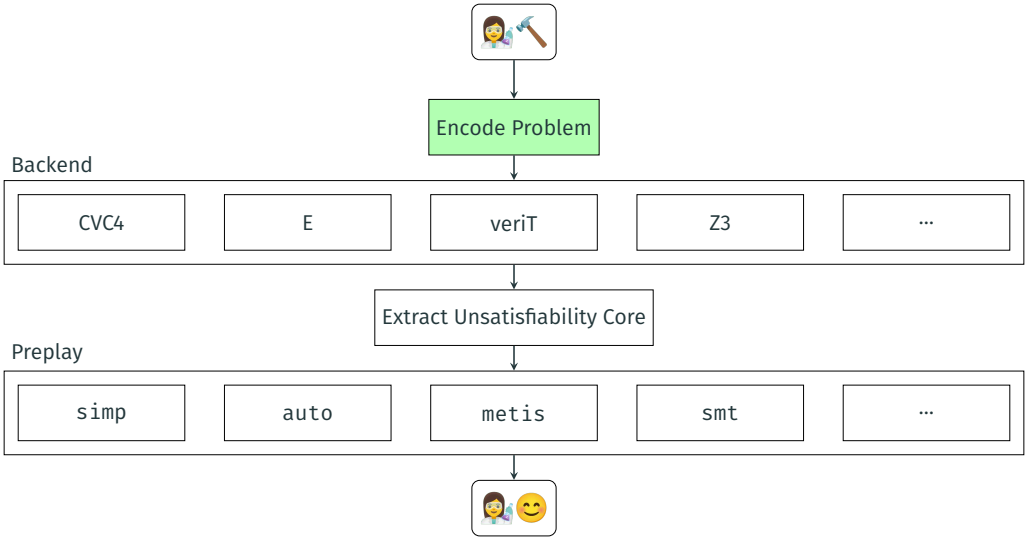
Interactive Theorem Proving with Sledgehammer



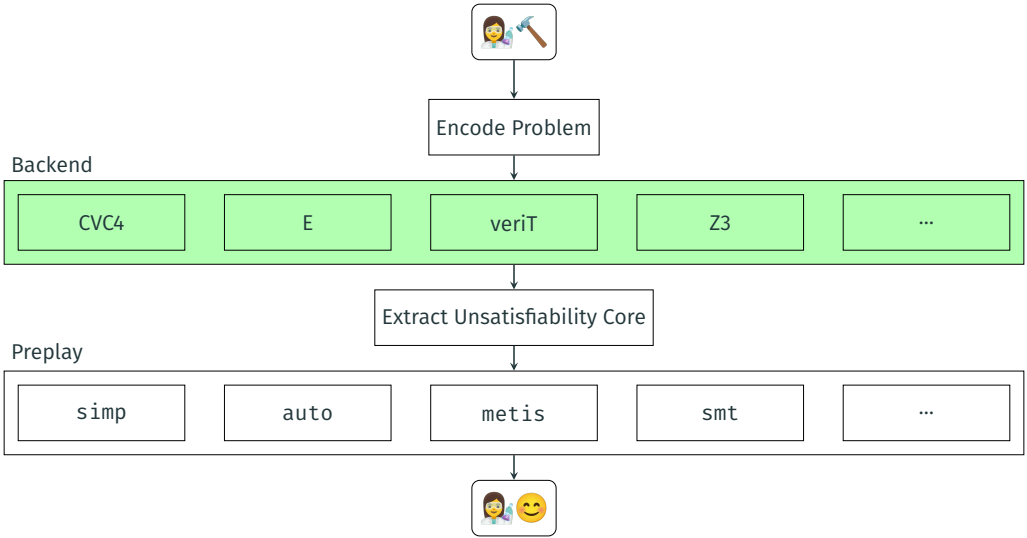
Interactive Theorem Proving with Sledgehammer



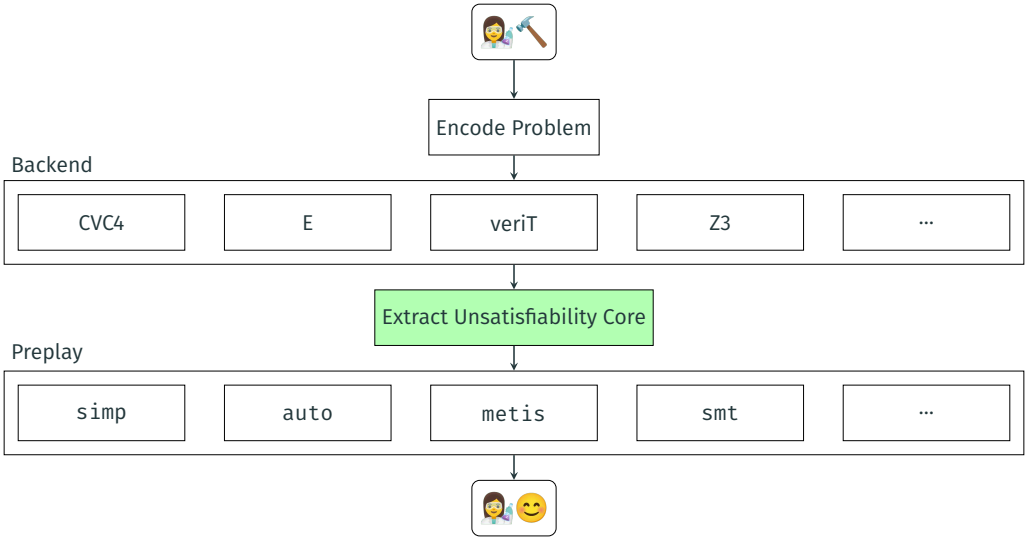
Interactive Theorem Proving with Sledgehammer



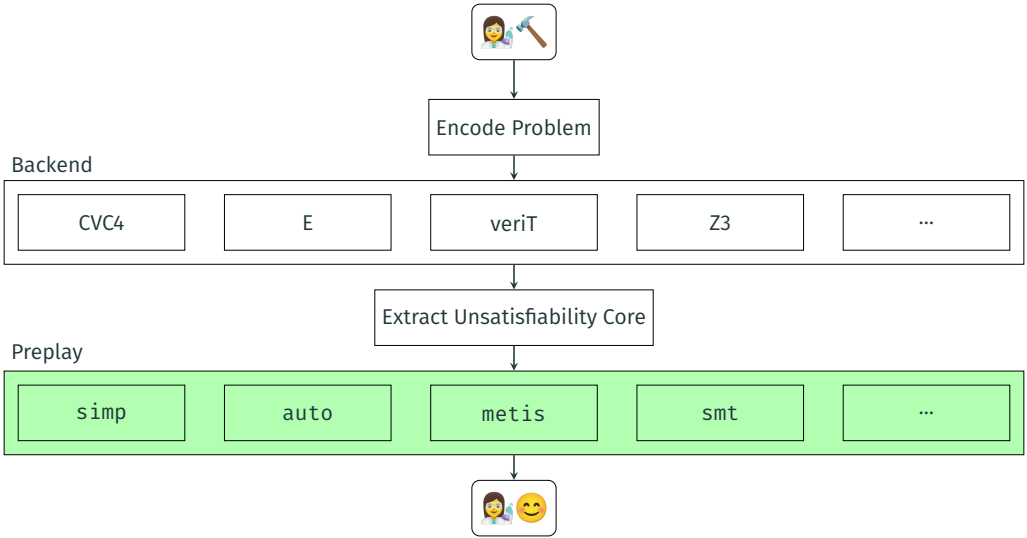
Interactive Theorem Proving with Sledgehammer



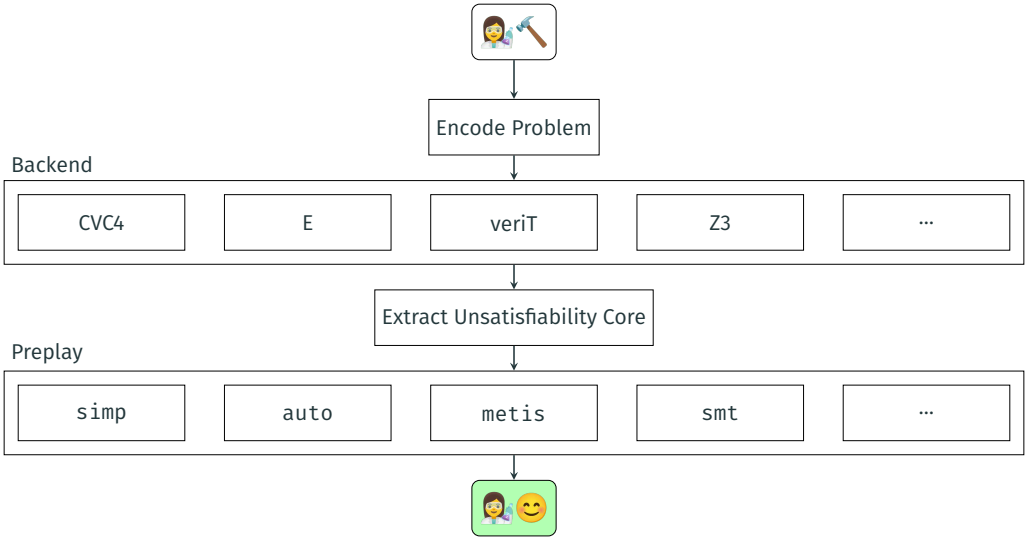
Interactive Theorem Proving with Sledgehammer



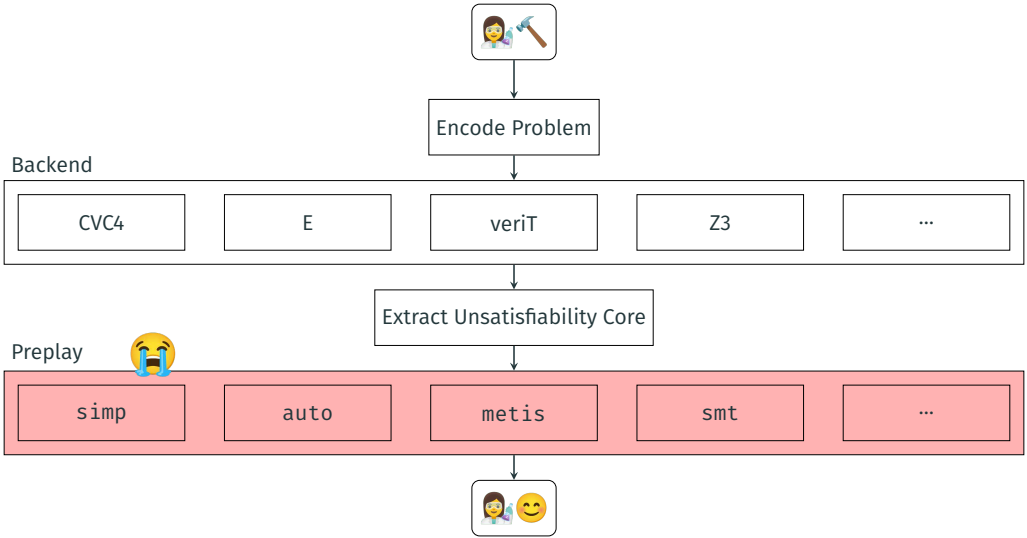
Interactive Theorem Proving with Sledgehammer



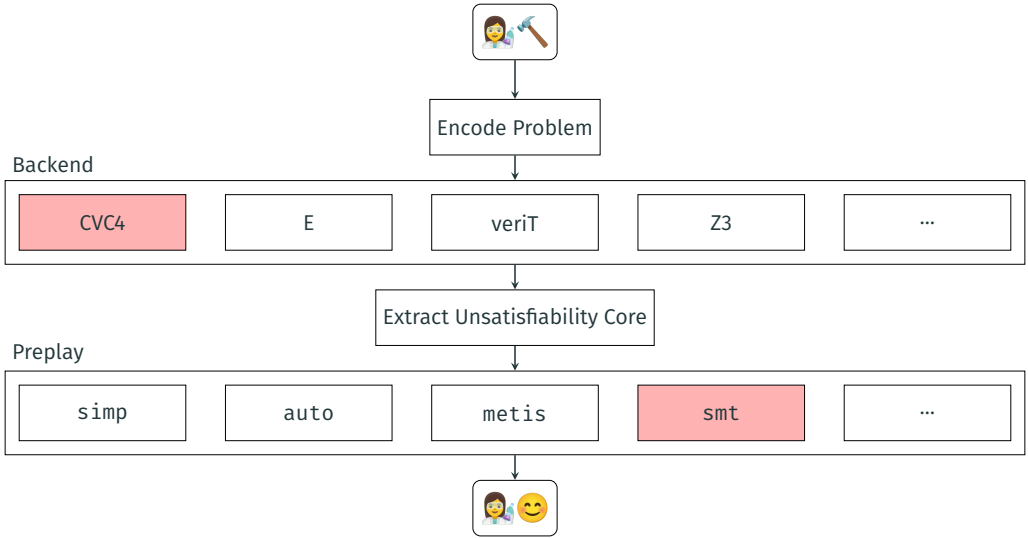
Interactive Theorem Proving with Sledgehammer



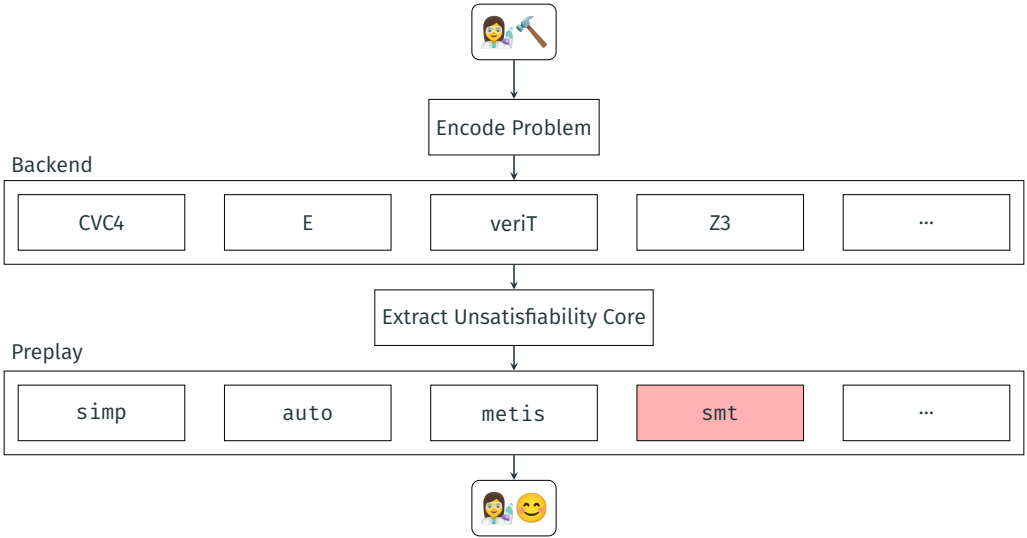
Interactive Theorem Proving with Sledgehammer



Interactive Theorem Proving with Sledgehammer



Interactive Theorem Proving with Sledgehammer



SMT Proof Formats

Old format:
- based on natural deduction
- used in Isabelle, Key, ...
- unmaintained

New format (ongoing work as far as I can see)

[Github discussion]

- more DRAT based
- much less detailed information
- no quantifier support for now
- no standard

Old format:
- based on natural deduction
- used in Isabelle, Key, ...
- unmaintained

New format (ongoing work as far as I can see)

[Github discussion]

- more DRAT based
- much less detailed information
- no quantifier support for now
- no standard

SMTInterpol

- purely resolution based
- not used as far as I am aware
- very verbose

[Hoenicke and Schindler, SMT'22]

LFSC	old
Alethe	new, the proof format of veriT (This talk!)
Alf	newest, based on Alethe (mostly compatible with Alethe)

LFSC old

Alethe new, the proof format of veriT This talk!

Alf newest, based on Alethe mostly compatible with Alethe

LFSC old

Alethe new, the proof format of veriT This talk!

Alf newest, based on Alethe mostly compatible with Alethe

Alethe

What do Users want?

First format developed

- with a semantics in mind
  `https://verit.loria.fr/documentation/alethe-spec.pdf`
- with a Zulip chat for questions `https://alethe.zulipchat.com`
- while working on reconstruction!

What is hard?

SAT Solver Resolution, $A \vee \ell$ and $B \vee \neg\ell$ implies $A \vee B$

- preserves logical equivalence but not all transformations do

Theory solvers lemmas like $x < y \vee x > y \vee x = y$ or $\neg(x = y) \vee f(x) = f(y)$

- not detailed enough for simplifications

Instantiation Module lemmas like $\neg(\forall x.\ \phi[x]) \vee \phi[t]$

Complicated code, many cases

What is hard?

SAT Solver Resolution, $A \vee \ell$ and $B \vee \neg\ell$ implies $A \vee B$

- preserves logical equivalence but not all transformations do

Theory solvers lemmas like $x < y \vee x > y \vee x = y$ or $\neg(x = y) \vee f(x) = f(y)$

- not detailed enough for simplifications

Instantiation Module lemmas like $\neg(\forall x.\ \phi[x]) \vee \phi[t]$

Complicated code, many cases

Idea: local transformations with binders. [Barbosa et al, JAR'20]

Skolemization $\neg(\forall x.\ \phi(x)) \simeq \neg p(\varepsilon x.\ \neg\phi(x))$

Let-elim $(\text{let } x = a \text{ in } p(x,x)) \simeq p(a,a)$

Theory simplification $k + 1 \times 0 < k \simeq k < k$

Challenge: efficient and sound manipulation of bound variables

Definition

A context Γ fixes variables and specifies substitutions:

$$\Gamma ::= \emptyset \mid \Gamma, x \mapsto s \mid \Gamma, x$$

substitution — bound variable

Rules have the form

$$\frac{\mathcal{D}_1' \quad \cdots \quad \mathcal{D}_n}{\Gamma \rhd t \simeq u}\ \mathsf{R}$$

premises — assumptions — Transformation

Semantics: proof of $\Gamma(t) = u$ for all variables fixed by Γ

Definition

A context Γ fixes variables and specifies substitutions:

$$\Gamma ::= \emptyset \mid \Gamma, x \mapsto s \mid \Gamma, x$$

substitution — bound variable

Rules have the form

$$\frac{\mathcal{D}_1' \quad \cdots \quad \mathcal{D}_n}{\Gamma \rhd t \simeq u}\ \mathsf{R}$$

premises — assumptions — Transformation

Semantics: proof of $\Gamma(t) = u$ for all variables fixed by Γ

The Rules of the Game (1)

$$\frac{}{\Gamma \rhd a \simeq a}\ \textsc{Refl}$$

$$\frac{}{\Gamma \rhd b \simeq a}\ \textsc{Taut}_{\mathcal{T}},\ \text{if } \vDash_{\mathcal{T}} \Gamma(b) = a$$

Let-Example

$$\dfrac{\dfrac{}{\rhd\ a \simeq a}\ \text{REFL} \qquad \dfrac{\dfrac{}{x \mapsto a\ \rhd\ x \simeq a}\ \text{REFL} \qquad \dfrac{}{x \mapsto a\ \rhd\ x \simeq a}\ \text{REFL}}{x \mapsto a\ \rhd\ p(x,x) \simeq p(a,a)}\ \text{CONG}}{\rhd\ (\text{let } x = a \text{ in } p(x,x)) \simeq p(a,a)}\ \text{LET}$$

Proof-producing contextual recursion

function *process*$(\Gamma,\ t)$
  match t
    case x*:*
      return *build_var*$(\Gamma,\ x)$
    case $\mathsf{f}(\bar{t}_n)$*:*
      $\bar{\Gamma}'_n \leftarrow (\mathit{ctx\_app}(\Gamma,\ \mathsf{f}, \bar{t}_n,\ i))_{i=1}^{n}$
      return $\mathit{build\_app}\big(\Gamma,\ \bar{\Gamma}'_n,\ \mathsf{f},\ \bar{t}_n,\ (\mathit{process}(\Gamma'_i,\ t_i))_{i=1}^{n}\big)$
    case $Qx.\ \varphi$*:*
      $\Gamma' \leftarrow \mathit{ctx\_quant}(\Gamma,\ Q,\ x,\ \varphi)$
      return $\mathit{build\_quant}(\Gamma,\ \Gamma',\ Q,\ x,\ \varphi,\ \mathit{process}(\Gamma',\ \varphi))$
    case $\mathrm{let}\ \bar{x}_n \simeq \bar{r}_n\ \mathrm{in}\ t'$*:*
      $\Gamma' \leftarrow \mathit{ctx\_let}(\Gamma,\ \bar{x}_n,\ \bar{r}_n,\ t')$
      return $\mathit{build\_let}(\Gamma,\ \Gamma',\ \bar{x}_n,\ \bar{r}_n,\ t',\ \mathit{process}(\Gamma',\ t'))$

Proof-producing contextual recursion

This is what developers like about Alethe: every call is on step you add to the proof!

Reconstruction in Isabelle

Communication with the Developers

I give you $\neg(\forall x.\ \phi[x]) \vee \phi[t]$, is it enough?

Can you also give me the instantiations? It is hard for me to

Communication with the Developers

I give you $\neg(\forall x.\ \phi[x]) \vee \phi[t]$, is it enough?

Can you also give me the instantiations? It is hard for me to guess them because you reorder equalities.

Sure!

Here is a case where QI is also doing CNF tranformation. Is that expected?

Yeah sorry. It is very hard to fix

Communication with the Developers

I give you $\neg(\forall x.\ \phi[x]) \vee \phi[t]$, is it enough?

Can you also give me the instantiations? It is hard for me to guess them because you reorder equalities.

Sure!

Here is a case where QI is also doing CNF tranformation. Is that expected?

Yeah sorry. It is very hard to fix

Communication with the Developers

I give you $\neg(\forall x.\ \phi[x]) \vee \phi[t]$, is it enough?

Can you also give me the instantiations? It is hard for me to guess them because you reorder equalities.

Sure!

Here is a case where QI is also doing CNF tranformation. Is that expected?

Yeah sorry. It is very hard to fix

Communication with the Developers

I give you $\neg(\forall x.\ \phi[x]) \vee \phi[t]$, is it enough?

Can you also give me the instantiations? It is hard for me to guess them because you reorder equalities.

Sure!

Here is a case where QI is also doing CNF tranformation. Is that expected?

Yeah sorry. It is very hard to fix

Communication with the Developers

I give you $\neg(\forall x.\ \phi[x]) \vee \phi[t]$, is it enough?

Can you also give me the instantiations? It is hard for me to guess them because you reorder equalities.

Sure!

Here is a case where QI is also doing CNF tranformation. Is that expected?

Yeah sorry. It is very hard to fix

Communication with the Developers

(6 months later)

Actually I have fixed it, now the CNF transformation is a separate step!

Communication with the Developers

(2 years later)

Actually let's change the syntax. Because we only instantiate all quantifiers from outside to inside

Are you sure?

Not fully actually

Communication with the Developers

(2 years later)

Actually let's change the syntax. Because we only instantiate all quantifiers from outside to inside

Are you sure?

Not fully actually

Communication with the Developers

(2 years later)

Actually let's change the syntax. Because we only instantiate all quantifiers from outside to inside

Are you sure?

Not fully actually

Communication with the Hanna, towards getting cvc5 in Isabelle

What the hell is your Isabelle code doing in the quantifier instantiation? Why is it so complicated?

Communication with the Hanna, towards getting cvc5 in Isabelle

What the hell is your Isabelle code doing in the quantifier instantiation? Why is it so complicated?

But there are subtle differences

1. Isabelle does not follow the context semantics. Instead of a λ, the equalities are part of the assumptions in the context [Schurr et al, CADE 28]
   - You do not want to re-automate a new symbol when you have equality
   - No type-safe representation possible of $\simeq$
2. The steps are sometimes still too coarse ongoing work on `lia_generic` in Carcara: one step for veriT = 60 for Z3
3. The reconstruction is very taylored towards veriT Equalities can be reordered (Carcara does it), but veriT is using a deterministic order

But there are subtle differences

1. Isabelle does not follow the context semantics. Instead of a λ, the equalities are part of the assumptions in the context [Schurr et al, CADE 28]
   - You do not want to re-automate a new symbol when you have equality
   - No type-safe representation possible of $\simeq$
2. The steps are sometimes still too coarse ongoing work on `lia_generic` in Carcara: one step for veriT = 60 for Z3
3. The reconstruction is very taylored towards veriT Equalities can be reordered (Carcara does it), but veriT is using a deterministic order

But there are subtle differences

1. Isabelle does not follow the context semantics. Instead of a λ, the equalities are part of the assumptions in the context [Schurr et al, CADE 28]
   - You do not want to re-automate a new symbol when you have equality
   - No type-safe representation possible of $\simeq$
2. The steps are sometimes still too coarse ongoing work on `lia_generic` in Carcara: one step for verit = 60 for Z3
3. The reconstruction is very taylored towards veriT Equalities can be reordered (Carcara does it), but veriT is using a deterministic order

But there are subtle differences

1. Isabelle does not follow the context semantics. Instead of a λ, the equalities are part of the assumptions in the context [Schurr et al, CADE 28]
   - You do not want to re-automate a new symbol when you have equality
   - No type-safe representation possible of $\simeq$

2. The steps are sometimes still too coarse ongoing work on `lia_generic` in Carcara: one step for verit = 60 for Z3

3. The reconstruction is very taylored towards veriT Equalities can be reordered (Carcara does it), but veriT is using a deterministic order

But there are subtle differences

1. Isabelle does not follow the context semantics. Instead of a λ, the equalities are part of the assumptions in the context [Schurr et al, CADE 28]
   - You do not want to re-automate a new symbol when you have equality
   - No type-safe representation possible of $\simeq$

2. The steps are sometimes still too coarse ongoing work on `lia_generic` in Carcara: one step for verit = 60 for Z3

3. The reconstruction is very taylored towards veriT Equalities can be reordered (Carcara does it), but veriT is using a deterministic order

The Developers Strike Back

We want RaRe Rules!

RaRE rules:

[Lachnitt et al., TACAS'24]

- rules not in the specification
- included during compilation

- but also sometimes useless
- but also sometimes buggy

Checking in Carcara

Carcara [Andreotti et al., TACAS'23]

- independent proof checker, written in Rust
- focused on performance and usability
- has a dedicated checking procedure for each rule in the format

(named after this bird ↑)

Proof Checking

- dealing with implicit equality reordering makes checking tricky
  - e.g., `(= a b)` and `(= b a)` are used interchangeably
- some rules don't provide all the details needed for checking
  - e.g., the resolution pivots are not provided
- some rules are simply too hard to check
  - e.g., checking `lia_generic` is an NP-hard problem

Proof Checking

- dealing with implicit equality reordering makes checking tricky
  - e.g., `(= a b)` and `(= b a)` are used interchangeably
- some rules don't provide all the details needed for checking
  - e.g., the resolution pivots are not provided
- some rules are simply too hard to check
  - e.g., checking `lia_generic` is an NP-hard problem

Proof Checking

- dealing with implicit equality reordering makes checking tricky
  - e.g., `(= a b)` and `(= b a)` are used interchangeably
- some rules don't provide all the details needed for checking
  - e.g., the resolution pivots are not provided
- some rules are simply too hard to check
  - e.g., checking `lia_generic` is an NP-hard problem

Proof Elaboration

- besides checking, Carcara is also a proof elaborator
- this means adding more detail to a proof and breaking down hard-to-check steps into more fine-grained ones
- the idea is to serve as a post-processing step, to make a proof easier to reconstruct in a proof assistant

Proof Elaboration

- besides checking, Carcara is also a proof elaborator
- this means adding more detail to a proof and breaking down hard-to-check steps into more fine-grained ones
- the idea is to serve as a post-processing step, to make a proof easier to reconstruct in a proof assistant

Proof Elaboration

- besides checking, Carcara is also a proof elaborator
- this means adding more detail to a proof and breaking down hard-to-check steps into more fine-grained ones
- the idea is to serve as a post-processing step, to make a proof easier to reconstruct in a proof assistant

Proof Elaboration

- remove the implicit reordering of equalities
- fill-in hard to check steps by calling external tools expanding `lia_generic` steps with an SMT solver
- completely remove some kinds of steps e.g. `reordering` steps

Carcara

- currently used by SMT solver developers to aid in adding support for Alethe
- proof elaboration also has use cases in translating Alethe proofs to different formats
- future work: extending to more theories, support for RaRe, SMT-LIB 3/Alf

Conclusion

- Alethe is a very nice proof format
- It is sometimes a bit verbous for my taste
- But has a very good proof checker

- RaRE rule to please developers of SMT solvers

extension on a per-solver based

- If you ever used `smt (verit, XXX)`, you actually used this work

CVC4: Preplay Success Rate