Improving the SMT Proof Reconstruction Pipeline in Isabelle/HOL

Hanna Lachnitt,[1] Mathias Fleury,[4] Haniel Barbosa,[2] Jibiana Jakpor,[1]
Bruno Andreotti,[2] Andrew Reynolds,[3] Hans-Jörg Schurr,[3] Clark Barrett,[1] Cesare Tinelli[3]

[1] Stanford University, [2] Universidade Federal de Minas Gerais, [3] The University of Iowa, [4] University of Freiburg

Motivation: Formal Verification

Isabelle/HOL:

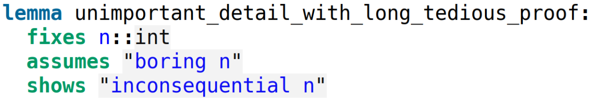
- *Interactive theorem prover*: Human and machine work together
- Expressive language (HOL) and large knowledge base of lemmas
- Offers a very high level of assurance
- Example: Verified SeL4 Microkernel

Motivation: Formal Verification

Isabelle/HOL:

- *Interactive theorem prover*: Human and machine work together
- Expressive language (HOL) and large knowledge base of lemmas
- Offers a very high level of assurance
- Example: Verified SeL4 Microkernel

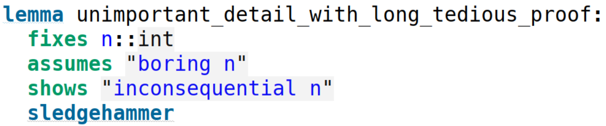
Proving lemmas can be tedious though...



```
lemma unimportant_detail_with_long_tedious_proof:
  fixes n::int
  assumes "boring n"
  shows "inconsequential n"
```

Proof Automation to the Rescue

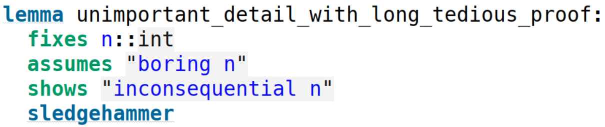
Isabelle’s Sledgehammer tool can help:



```
lemma unimportant_detail_with_long_tedious_proof:
  fixes n::int
  assumes "boring n"
  shows "inconsequential n"
  sledgehammer
```

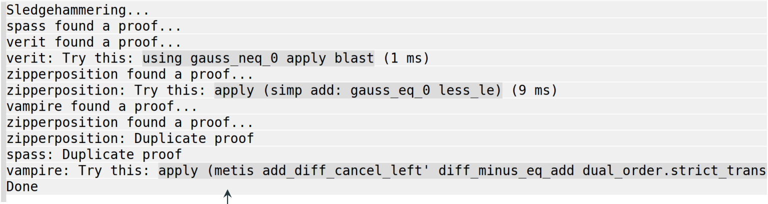
Proof Automation to the Rescue

Isabelle's Sledgehammer tool can help:



```
lemma unimportant_detail_with_long_tedious_proof:
  fixes n::int
  assumes "boring n"
  shows "inconsequential n"
  sledgehammer
```

Sledgehammer can call external ATPs including SMT solvers

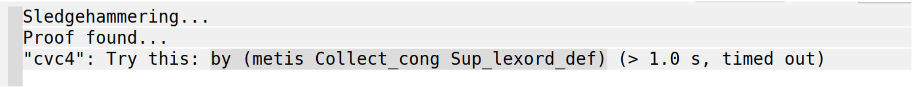


```
Sledgehammering...
spass found a proof...
verit found a proof...
verit: Try this: using gauss_neq_0 apply blast (1 ms)
zipperposition found a proof...
zipperposition: Try this: apply (simp add: gauss_eq_0 less_le) (9 ms)
vampire found a proof...
zipperposition found a proof...
zipperposition: Duplicate proof
spass: Duplicate proof
vampire: Try this: apply (metis add_diff_cancel_left' diff_minus_eq_add dual_order.strict_trans
Done
```

It learns which facts the solver used and finds an internal proof

Proof Reconstruction

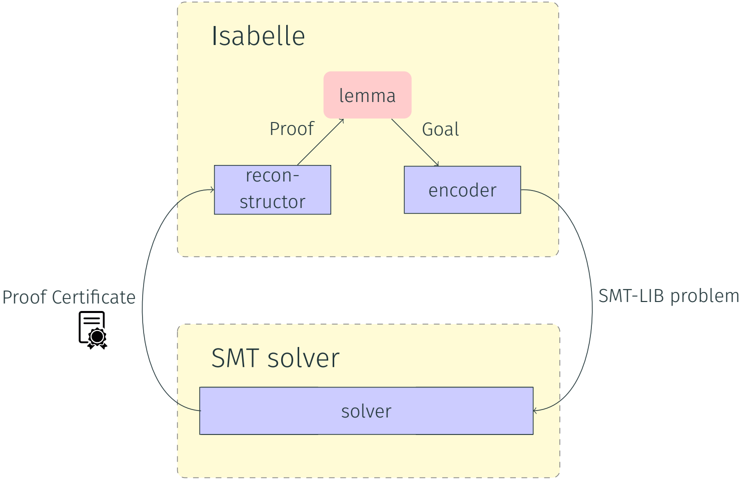
Finding an internal proof is not always possible even if the facts are known:



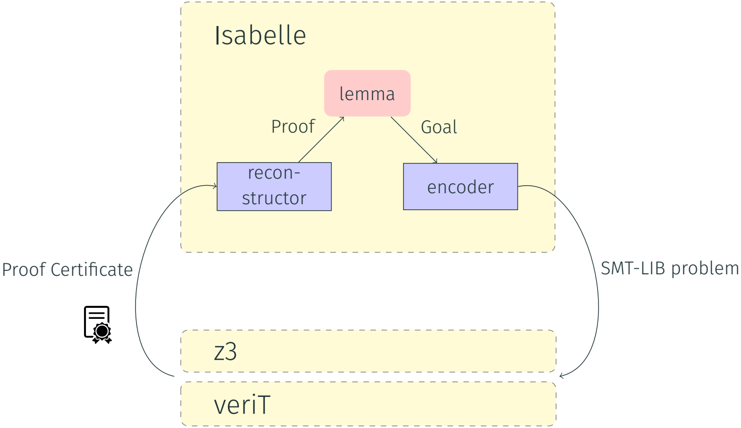
```
Sledgehammering...
Proof found...
"cvc4": Try this: by (metis Collect_cong Sup_lexord_def) (> 1.0 s, timed out)
```

If the solver explains its reasoning in the form of a proof certificate Isabelle can check each step. This increases the rate of success.

Proof Automation Circle



Currently Supported Solvers



Our goal: facilitate adding new proof-producing solvers to Isabelle

Agenda

Goal: add a new proof-producing solver to Isabelle

Running case study: integrating cvc5

1. required modifications on the solver-side
2. required modifications on the Isabelle-side
3. new debugging tools

Adding a New Solver

Adding reconstruction for a new format is very expensive!

- Isabelle already supports veriT’s and z3’s proof certificates

💡 Reconstruction should be gratis if the new solver outputs proofs in an existing format.

Adding a New Solver

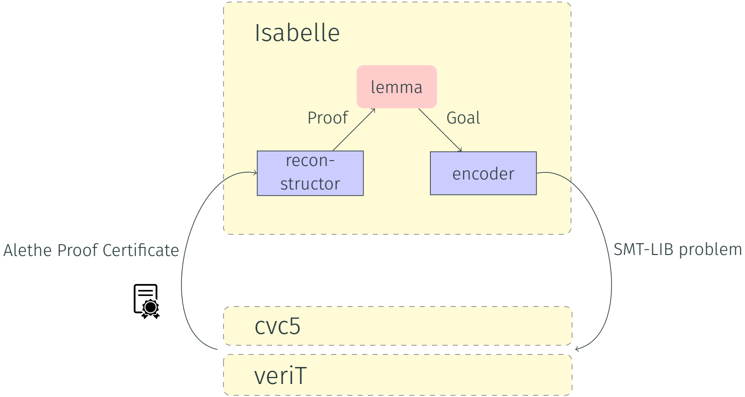
Adding reconstruction for a new format is very expensive!

- Isabelle already supports veriT’s and z3’s proof certificates

💡 Reconstruction should be gratis if the new solver outputs proofs in an existing format.

- z3 format is too coarse, not well documented, and has known bugs
- veriT’s output is now called the Alethe format, is generic and has detailed specification

Adding cvc5 using Alethe Proofs



Adding a New Solver

Adding reconstruction for a new format is very expensive!

- Isabelle already supports veriT's and z3's proof certificates

💡 Reconstruction should be gratis if the new solver outputs proofs in an existing format.

- z3 format is too coarse, not well documented, and has known bugs
- veriT's output is now called the Alethe format, is generic and has detailed specification

Adding a New Solver

Adding reconstruction for a new format is very expensive!

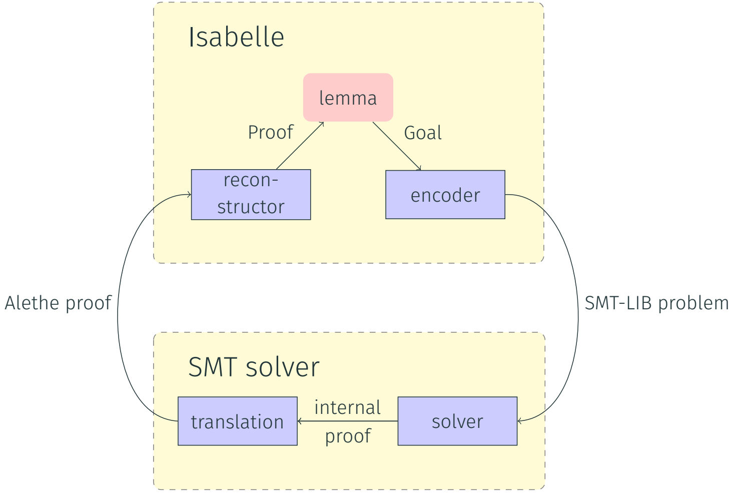
- Isabelle already supports veriT’s and z3’s proof certificates

💡 Reconstruction should be gratis if the new solver outputs proofs in an existing format.

- z3 format is too coarse, not well documented, and has known bugs
- veriT’s output is now called the Alethe format, is generic and has detailed specification

What if solver does not natively output Alethe proofs?

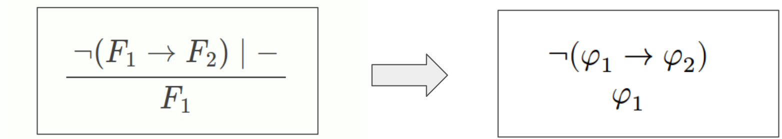
Proof Replay with Translation into Alethe



Translation Approach

We want to support cvc5 as a proof producing solver in Isabelle

- cvc5's internal proof format is called CPC
- Each proof rule in the original calculus needs to be translated into a Alethe rules
- This adds some overhead in time and proof size
- Most rules are easy to translate!



$$\frac{\neg(F_1 \rightarrow F_2) \mid -}{F_1}$$

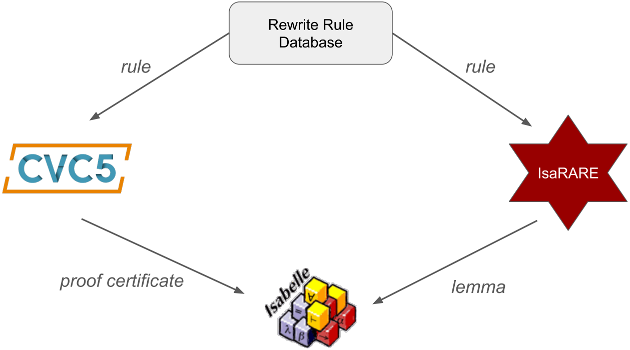
$$\begin{gathered}\neg(\varphi_1 \rightarrow \varphi_2)\\ \varphi_1\end{gathered}$$

CPC rule
NOT_IMPLIES_ELIM1

Alethe rule
not_implies1

Automated Translation

- cvc5 uses hundreds of rewrite rules that might be added or deleted. Translating them would be tedious!
- We use IsaRARE to automatically add them to the Isabelle Alethe reconstruction

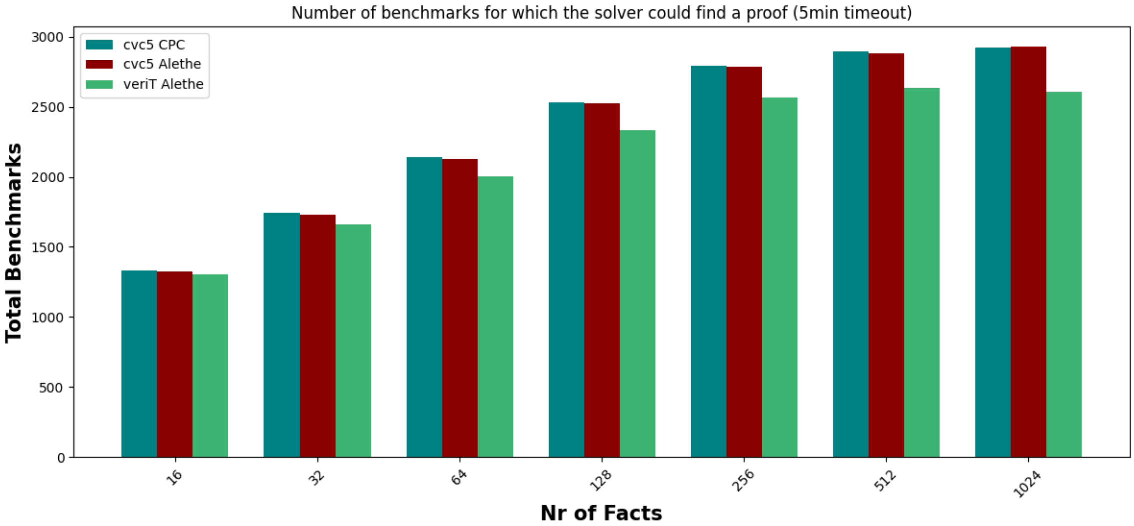


Making cvc5 Alethe Proof-Producing : Evaluation

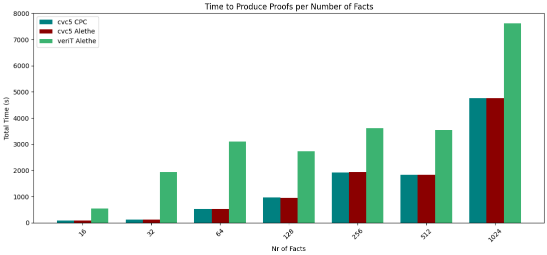
Evaluation

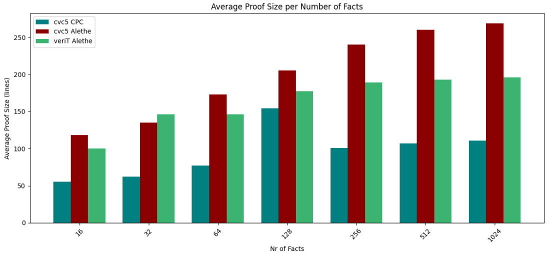
- SMT-LIB problems and those generated from Isabelle goals have a very different structure
- We test on Sledgehammer generated benchmarks from the seventeen provers paper [1]
    - Each set contains the same 5000 problems with different amount of facts included
- We compare cvc5's CPC with cvc5's and veriT's Alethe proofs

Solved Instances



Solving Time and Proof Size (on Benchmarks Solved by All Solvers)





Conclusion:

- cvc5 is faster even with the translation overhead to Alethe
- cvc5 Alethe proofs are larger than veriT's proofs.

I’ll just plug it into Isabelle and we are done.

Thanks for your attention!

Any questions?

I'll just plug it into Isabelle and we are done.

Thanks for your attention!

Any questions?

Problems with the Alethe Reconstruction

Not the whole standard was supported by Isabelle

- veriT produces only a subset of allowed Alethe proofs
- the structure of the problems restricted the fragment even more

Proofs outside of the standard were supported

- it was not clear what was expected by Isabelle

Even updating to a newer version of veriT was not possible without changes!

Problem Analysis

Insights:

- Parts of the Isabelle code were too coupled.
    - E.g., encoding and reconstruction
    - Developers needed to become experts in every part of the code to fix any bug
    - Efficient testing was not possible. E.g., only one instance of the onepoint rule

Problem Analysis

Insights:

- Parts of the Isabelle code were too coupled.
  - E.g., encoding and reconstruction
  - Developers needed to become experts in every part of the code to fix any bug
  - Efficient testing was not possible. E.g., only one instance of the onepoint rule
- The Alethe specification was sometimes unclear or wrong

Problem Analysis

Insights:

- Parts of the Isabelle code were too coupled.
  - E.g., encoding and reconstruction
  - Developers needed to become experts in every part of the code to fix any bug
  - Efficient testing was not possible. E.g., only one instance of the onepoint rule
- The Alethe specification was sometimes unclear or wrong
- New rules were not supported

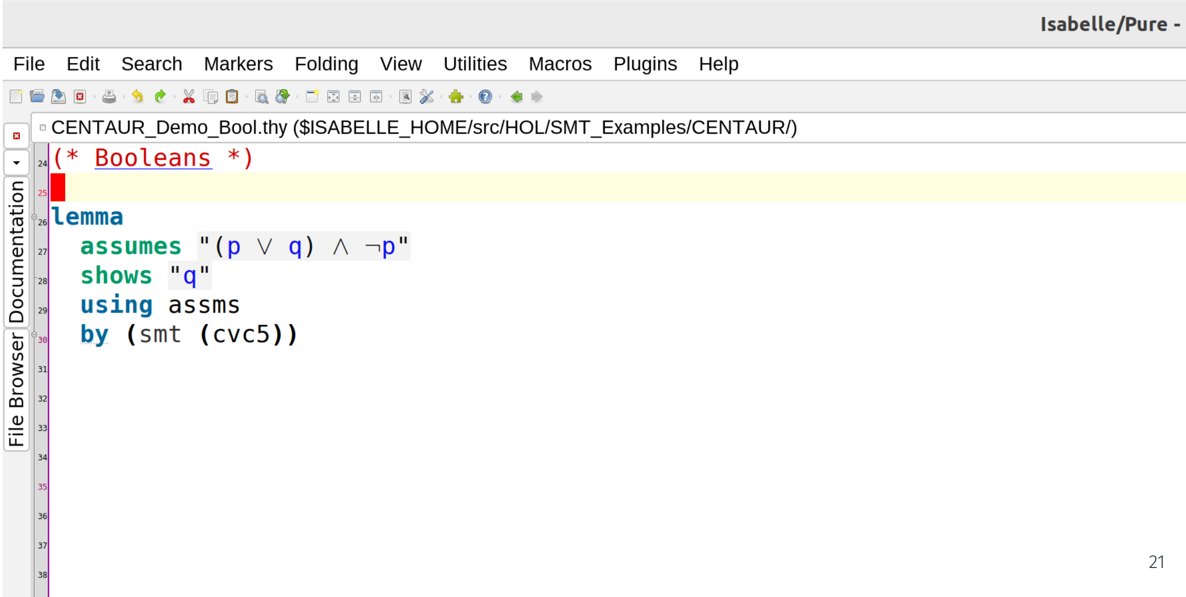
Problem Analysis

Insights:

- Parts of the Isabelle code were too coupled.
    - E.g., encoding and reconstruction
    - Developers needed to become experts in every part of the code to fix any bug
    - Efficient testing was not possible. E.g., only one instance of the onepoint rule
- The Alethe specification was sometimes unclear or wrong
- New rules were not supported

We extensively re-factored and extended the Isabelle code as well as the Alethe standard.
This was a huge team effort!

The smt Method: An Example Lemma



```
(* Booleans *)

lemma
  assumes "(p ∨ q) ∧ ¬p"
  shows "q"
  using assms
  by (smt (cvc5))
```

Testing Infrastructure

Great now cvc5 works on the examples we have but how do we know that we actually support Alethe proofs?

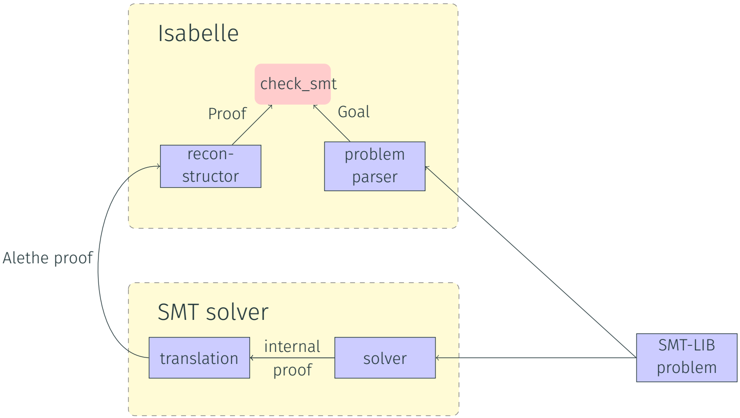
Previous experiments only included an extremely small set of benchmarks …

In this talk am going to focus on the biggest problem:

For any given Alethe proof rule, can all instances of it can be reconstructed?

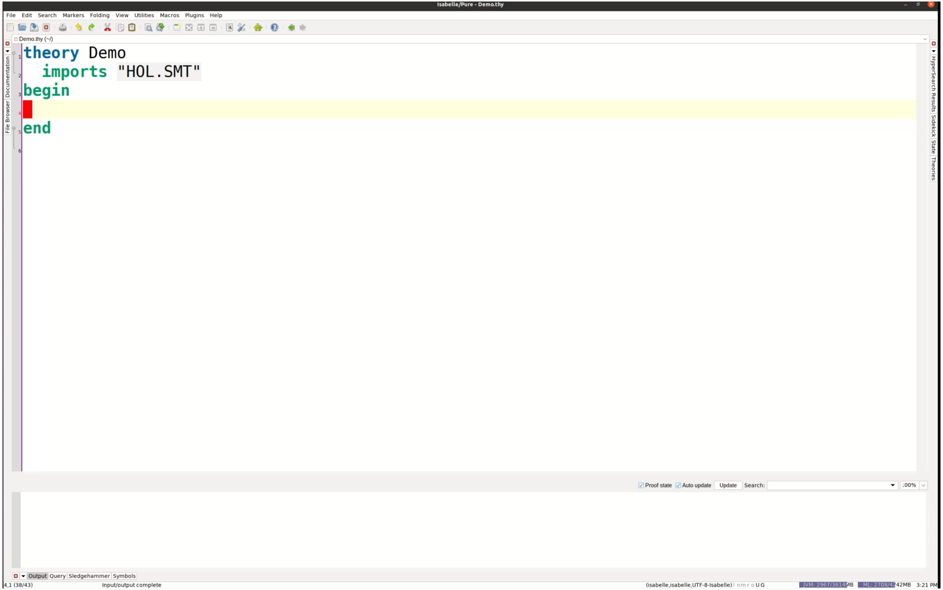
Testing Infrastructure
Enabling large-scale testing

Decoupling Encoding and Replay



Our tool check_smt can parse external SMT-LIB problems and check them against an Alethe proof certificate!

Demo: check_smt


Isabelle/Pure - Demo.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Demo.thy (~/)
File Browser Documentation
HyperSearch Results Sidekick State Theories
☑ Proof state ☑ Auto update Update Search:
100%
Output Query Sledgehammer Symbols
4,1 (38/43)
Input/output complete
(isabelle,isabelle,UTF-8-Isabelle) nmro UG
JVM: 2967/3614MB
ML: 2708/4242MB
3:21 PM

```
theory Demo
  imports "HOL.SMT"
begin

end
```

Evaluation

Table 1: Reconstruction Success. The average (reconstruction) time in ms only takes benchmarks into account that were solved both by veriT and cvc5 and does not take the solving time into account.

Benchmark Set	cvc5 Alethe				veriT Alethe			
	solved	rec.	unique rec.	Av. time	solved	rec.	unique rec.	Av. time
max facts 16	1327	1326	45	386	1305	1295	14	131
max facts 32	1732	1730	100	349	1658	1644	14	133
max facts 64	2130	2123	168	423	2003	1980	25	161
max facts 128	2523	2512	235	528	2335	2309	32	203
max facts 256	2788	2777	276	702	2565	2537	36	215
max facts 512	2881	2869	317	904	2637	2600	48	251
max facts 1024	2927	2916	411	1328	2607	2560	55	343

Carcara Slice Feature

It is still necessary to check the whole proof… This makes debugging painful.

Carcara Slice Feature

It is still necessary to check the whole proof... This makes debugging painful.

Original Problem

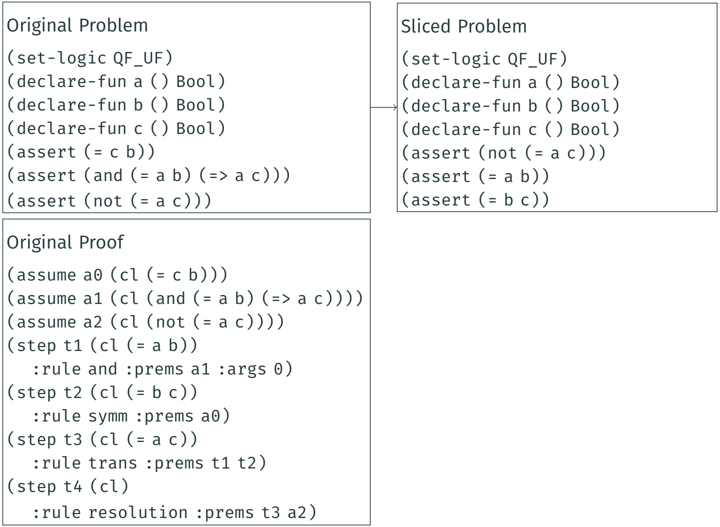
```
(set-logic QF_UF)
(declare-fun a () Bool)
(declare-fun b () Bool)
(declare-fun c () Bool)
(assert (= c b))
(assert (and (= a b) (=> a c)))
(assert (not (= a c)))
```

Original Proof

```
(assume a0 (cl (= c b)))
(assume a1 (cl (and (= a b) (=> a c))))
(assume a2 (cl (not (= a c))))
(step t1 (cl (= a b))
   :rule and :prems a1 :args 0)
(step t2 (cl (= b c))
   :rule symm :prems a0)
(step t3 (cl (= a c))
   :rule trans :prems t1 t2)
(step t4 (cl)
   :rule resolution :prems t3 a2)
```

Carcara Slice Feature

It is still necessary to check the whole proof... This makes debugging painful.



Original Problem

```
(set-logic QF_UF)
(declare-fun a () Bool)
(declare-fun b () Bool)
(declare-fun c () Bool)
(assert (= c b))
(assert (and (= a b) (=> a c)))
(assert (not (= a c)))
```

Sliced Problem

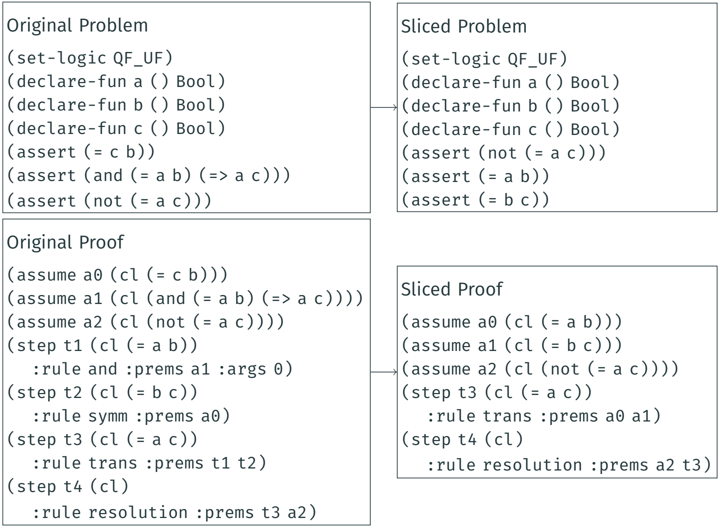
```
(set-logic QF_UF)
(declare-fun a () Bool)
(declare-fun b () Bool)
(declare-fun c () Bool)
(assert (not (= a c)))
(assert (= a b))
(assert (= b c))
```

Original Proof

```
(assume a0 (cl (= c b)))
(assume a1 (cl (and (= a b) (=> a c))))
(assume a2 (cl (not (= a c))))
(step t1 (cl (= a b))
   :rule and :prems a1 :args 0)
(step t2 (cl (= b c))
   :rule symm :prems a0)
(step t3 (cl (= a c))
   :rule trans :prems t1 t2)
(step t4 (cl)
   :rule resolution :prems t3 a2)
```

Carcara Slice Feature

It is still necessary to check the whole proof... This makes debugging painful.



Original Problem

```
(set-logic QF_UF)
(declare-fun a () Bool)
(declare-fun b () Bool)
(declare-fun c () Bool)
(assert (= c b))
(assert (and (= a b) (=> a c)))
(assert (not (= a c)))
```

Sliced Problem

```
(set-logic QF_UF)
(declare-fun a () Bool)
(declare-fun b () Bool)
(declare-fun c () Bool)
(assert (not (= a c)))
(assert (= a b))
(assert (= b c))
```

Original Proof

```
(assume a0 (cl (= c b)))
(assume a1 (cl (and (= a b) (=> a c))))
(assume a2 (cl (not (= a c))))
(step t1 (cl (= a b))
   :rule and :prems a1 :args 0)
(step t2 (cl (= b c))
   :rule symm :prems a0)
(step t3 (cl (= a c))
   :rule trans :prems t1 t2)
(step t4 (cl)
   :rule resolution :prems t3 a2)
```

Sliced Proof

```
(assume a0 (cl (= a b)))
(assume a1 (cl (= b c)))
(assume a2 (cl (not (= a c))))
(step t3 (cl (= a c))
   :rule trans :prems a0 a1)
(step t4 (cl)
   :rule resolution :prems a2 t3)
```

Contributions Summary

SMT solver and Alethe standard:

- Show that translation from industry-strength solver to Alethe is possible with little overhead
- Small number of holes remaining
- Correct, refine and extend Alethe standard

Isabelle:

- Refactor smt tactic implementation
- Support automatic lemma based reconstruction for simple rules
- Support new solver: cvc5

Testing:

- Isabelle mode for Carcara
- Slice tool for Carcara
- Isabelle internal tools and regression test library

Thank you for your attention

Want to learn more about the cvc5 proof reconstruction project? Check out my blog post on the cvc5 website (`https://cvc5.github.io/`)

Please feel free to contact me with any questions (lachnitt@stanford.edu):