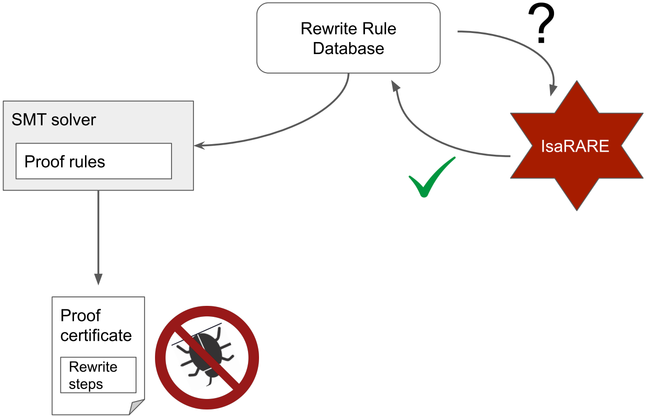
Automatic Verification of SMT Rewrites in Isabelle/HOL

Hanna Lachnitt,[1] Mathias Fleury,[4] Leni Aniva,[1] Andrew Reynolds,[3]
Haniel Barbosa,[2] Andres Nötzli,[1] Clark Barrett,[1] Cesare Tinelli[3]

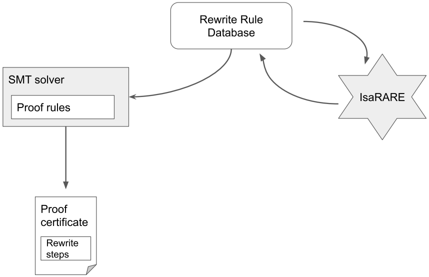
[1] Stanford University, [2] Universidade Federal de Minas Gerais, [3] The University of Iowa, [4] University of Freiburg

Our Contribution: IsaRARE a tool to verify rewrite rules



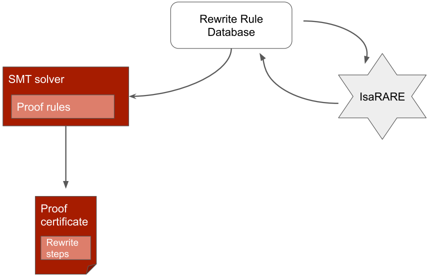
Agenda

- SMT Proofs
- Rewrites & RARE Language
- IsaRARE: Translation into Isabelle
- Evaluation on cvc5 Rewrite Database



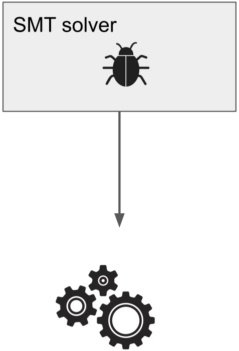
Agenda

- SMT Proofs
- Rewrites & RARE Language
- IsaRARE: Translation into Isabelle
- Evaluation on cvc5 Rewrite Database

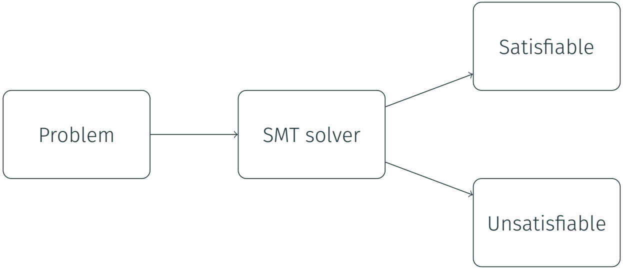


Bugs in SMT solvers

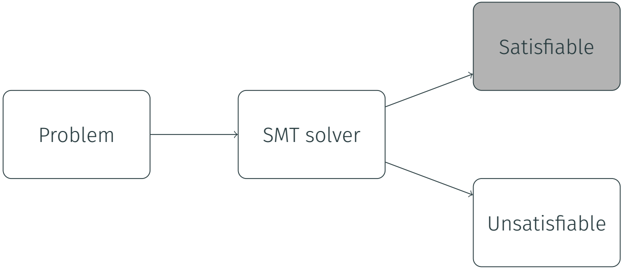
- SMT solvers often used in safety and security critical applications
  - E.g., billions of calls a day at AWS
- SMT solvers are
  - large and complex software projects
  - under active development
  - not practicable to be verified completely
- Despite best efforts they contain bugs
  - Disagreement between solvers on same benchmarks
  - Fuzzing tools often find bugs



Increasing Trust in SMT solvers



Increasing Trust in SMT solvers



Problem

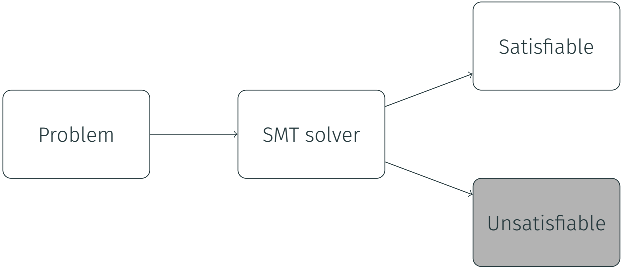
SMT solver

Satisfiable

Unsatisfiable

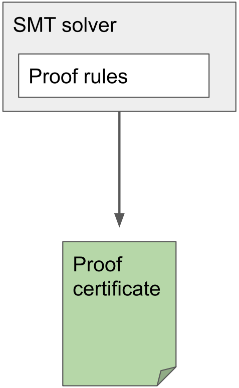
```
(model
(define-fun y () Int 0)
(define-fun x () Int (- 3))
(define-fun z () Int 2)
)
```

Increasing Trust in SMT solvers



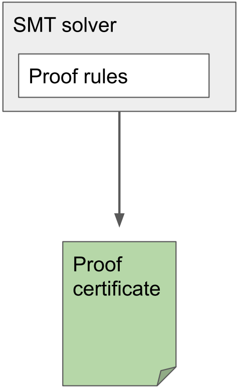
SMT Proofs

- Record of reasoning steps the solver did to reach unsat
  - Proof steps are instances of the solver’s internal proof rule calculus



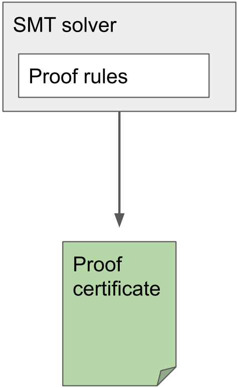
SMT Proofs

- Record of reasoning steps the solver did to reach unsat
  - Proof steps are instances of the solver’s internal proof rule calculus
- Proof can be checked independently:
  - Proof checking is usually easier than solving
  - Checker can be small enough to be formally verified



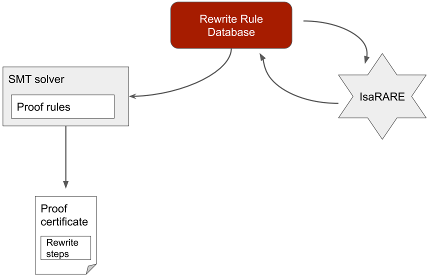
SMT Proofs

- Record of reasoning steps the solver did to reach unsat
  - Proof steps are instances of the solver's internal proof rule calculus
- Proof can be checked independently:
  - Proof checking is usually easier than solving
  - Checker can be small enough to be formally verified
- Proofs can have different granularities
  - Tradeoff between checking and solving



Agenda

- SMT Proofs
- **Rewrites & RARE Language**
- IsaRARE: Translation into Isabelle
- Evaluation on cvc5 Rewrite Database



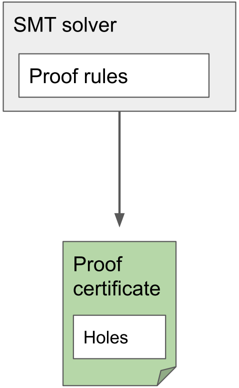
Rewrites

- Modern SMT solver implement hundred of rewrites for better performance

  (bvneg (bvneg x)) $\rightsquigarrow$ x

- Making this code proof producing is difficult and tedious
  - Requires a different proof rule for every rewrite

- Barbosa et al. (2022) and Nötzli et al. (2022) present a flexible infrastructure for proof production:
  - Low granularity: proof with holes for rewrites

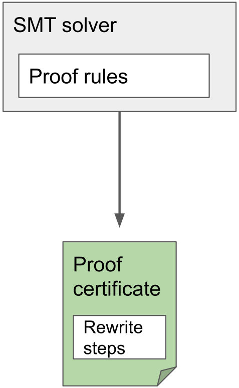


Rewrites

- Modern SMT solver implement hundred of rewrites for better performance

  (bvneg (bvneg x)) ⇝ x

- Making this code proof producing is difficult and tedious
  - Requires a different proof rule for every rewrite
- Barbosa et al. (2022) and Nötzli et al. (2022) present a flexible infrastructure for proof production:
  - Low granularity: proof with holes for rewrites
  - High granularity: rewrites are filled in in post-processing step



Rewrites

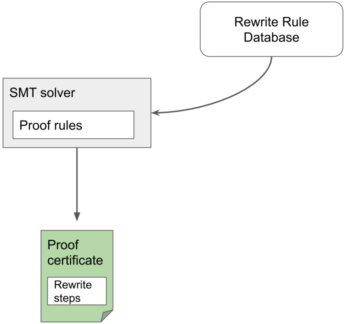
- Modern SMT solver implement hundred of rewrites for better performance

  (bvneg (bvneg x)) ⇝ x

- Making this code proof producing is difficult and tedious
  - Requires a different proof rule for every rewrite

- Barbosa et al. (2022) and Nötzli et al. (2022) present a flexible infrastructure for proof production:
  - Low granularity: proof with holes for rewrites
  - High granularity: rewrites are filled in in post-processing step

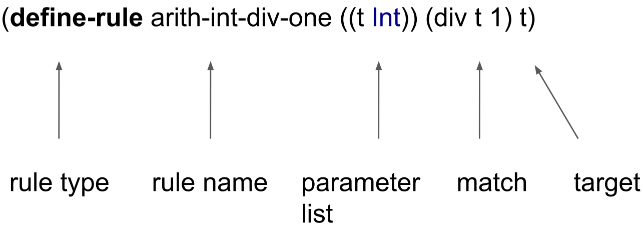
- Advantages:
  - Separate databases with rewrite rules can be used
  - Changes in rewrite code are easy



Specifying Rewrites in the RARE Language

RARE:

- syntax is an extension of SMT-LIB 3
- proof reconstructor uses one or more rewrite rules to fill a hole



(**define-rule** arith-int-div-one ((t Int)) (div t 1) t)

rule type

rule name

parameter list

match

target

Specifying Rewrites in the RARE Language

RARE:

- rules can be conditional
- supports matching n-ary functions using list arguments

```
(define-cond-rule str-eq-ctn-false
   ((x1 String :list) (x String)       ; parameter
    (x2 String :list) (y String))      ; list
  (= (str.contains y x) false)         ; condition
  (= (str.++ x1 x x2) y)               ; match
  false                                ; target
)
```

Specifying Rewrites in the RARE Language

RARE:

- Unlike SMT-LIB, we support gradual typing
- Fixed-point rules give a hint to the reconstructor

```
(define-rule bv-sub-eliminate
   ((x ?BitVec) (y ?BitVec))     ; parameter list
  (bvsub x y)                    ; match
  (bvadd x (bvneg y))            ; target
)
```

One Problem

A mistake in a RARE rule can have fatal consequences:

- The hole may not be fillable → incomplete proof
- An error in the code base might be covered up → checkable proof but unsound result

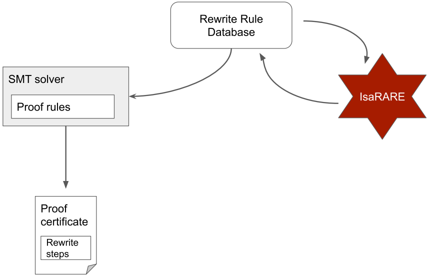
Proof checkers will directly use rewrite database

⇒ If we want trust, every aspect of our toolchain must be trustworthy!

⇒ Solution: Verify each rewrite rule in a trusted environment

Agenda

- SMT Proofs
- Rewrites & RARE Language
- **IsaRARE: Translation into Isabelle**
- Evaluation on cvc5 Rewrite Database



IsaRARE Overview I

We present the IsaRare a plug-in for Isabelle/HOL which:

- provides the **parse_rare_file** command (taking in a Rare file)
- generates a lemma for each rewrite rule
- suggests a proof skeleton

The user only has to prove any lemma that is not proven automatically.

If a lemma is proven the corresponding Rare rule is sound!

IsaRARE Example

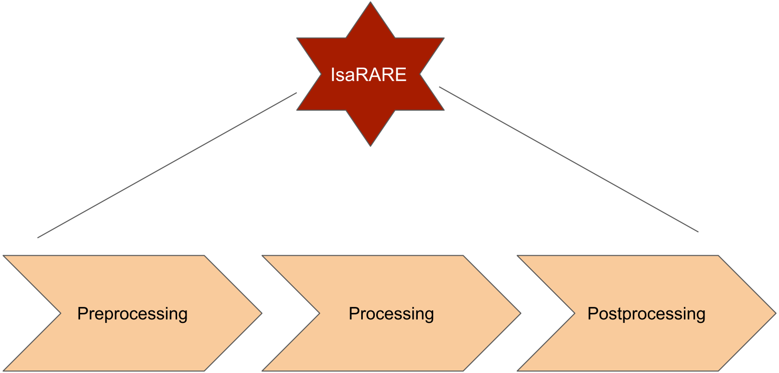
IsaRARE re-uses the SMT-LIB parser in Isabelle/HOL whenever possible

```
(define-rule ite-then-true
   ((c Bool) (x Bool))
  (ite c true x)
  (or c x))
```

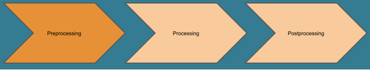
becomes:

```
lemma [ite_then_true]:
fixes c::”bool” and x::”bool”
shows ”(if c then True else x) = (c ∨ x)”
```

IsaRARE



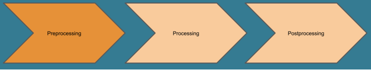
IsaRARE: Preprocessing



First, the rule is parsed into an AST using Isabelle's SMT-LIB parser. Then, IsaRARE:

- eliminates define-rule* rules
- adds implicit conditions (obeyed in cvc5 due to SMT-LIB syntax)

IsaRARE: Preprocessing



First, the rule is parsed into an AST using Isabelle’s SMT-LIB parser. Then, IsaRARE:

- eliminates define-rule* rules
- adds implicit conditions (obeyed in cvc5 due to SMT-LIB syntax)

Example of generated conditions with gradual types:

```
(define-rule bv-extract-extract
((x ?BitVec) (i Int)
(j Int) (k Int) (l Int)))
(extract l k (extract j i x)))
(extract (+ i l) (+ i k) x))
```

(a) Rare rule

```
t0 = (extract j i x) ∧
size t0 = j + 1 - i ∧
t1 = (extract l k t0) ∧
size t1 = l + 1 - k ∧
t2 = (extract (i+l) (i+k) x) ∧
size t2 = (i+l) + 1 - (i+k) ∧
j ≤ size x ∧ 0 ≤ i ∧ i ≤ j ∧
l ≤ size t0 ∧ 0 ≤ k ∧ k ≤ l ∧
(i+l) ≤ size x ∧ 0 ≤ (i+k) ∧
(i+k) ≤ (i+l)
```

(b) Additional Assumptions

IsaRARE: Processing

Every SMT-LIB term is mapped to an Isabelle term (E.g., *and* is mapped to *HOL.conj(∧)*)

- We extended the Isabelle bitvector term parser and add a new string term parser
- For gradual types we assign a dummy type
- Then, we re-infer types

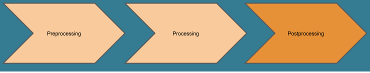
```
if (x = y) then (bvand z u) else w
```

(a) Rare term

```
x::” 'a::len word”
y::” 'a::len word”
z::” 'b::len word”
u::” 'b::len word”
w::” 'b::len word”
```

(b) Most general types

IsaRARE: Postprocessing



Preprocessing

Processing

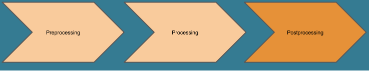
Postprocessing

- SMT-LIB uses n-ary operators (e.g, and, concat) but Isabelle does not
- During parsing the operators are binarised
- This does not work for lists! E.g., (and xs y)
- Many special cases, what if xs is empty?

```
(define-rule bool-and-false ((xs Bool :list) (ys Bool :list)) (and xs false ys) false)
```

$\rightsquigarrow (xs \wedge (false \wedge ys))$

IsaRARE: Postprocessing



```
datatype 'a rare_ListVar = ListVar "'a list"
datatype 'a rare_ListOp = ListOp "'a ⇒ 'a ⇒ 'a" "'a"
```

(a) Encapsulating Lists in a new Datatype

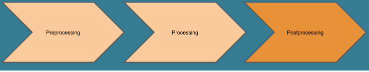
```
lemma rare_list_left_transfer
shows "rare_list_left op (ListVar xs) y = foldr op xs y"
```

(b) Transfer between new Definitions and Fold

```
lemma bool_and_true:
 fixes xs::"bool rare_ListVar" and ys::"bool rare_ListVar"
shows "rare_list_left (∧) xs (rare_list_right (∧) True ys)
= rare_list_both (∧) True xs ys"
```

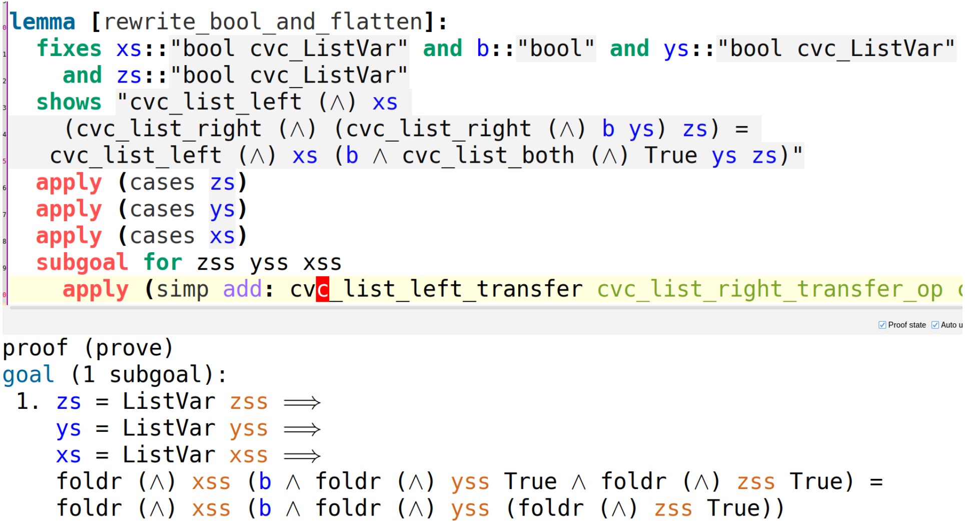
(c) Lemma with RARE Lists

IsaRARE: Postprocessing


Preprocessing

Processing

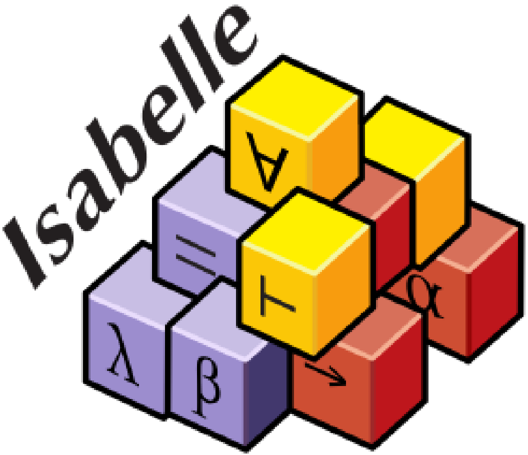
Postprocessing


```
lemma [rewrite_bool_and_flatten]:
  fixes xs::"bool cvc_ListVar" and b::"bool" and ys::"bool cvc_ListVar"
    and zs::"bool cvc_ListVar"
  shows "cvc_list_left (∧) xs
    (cvc_list_right (∧) (cvc_list_right (∧) b ys) zs) =
   cvc_list_left (∧) xs (b ∧ cvc_list_both (∧) True ys zs)"
  apply (cases zs)
  apply (cases ys)
  apply (cases xs)
  subgoal for zss yss xss
    apply (simp add: cvc_list_left_transfer cvc_list_right_transfer_op c
```

☑ Proof state ☑ Auto u

```
proof (prove)
goal (1 subgoal):
 1. zs = ListVar zss ⟹
    ys = ListVar yss ⟹
    xs = ListVar xss ⟹
    foldr (∧) xss (b ∧ foldr (∧) yss True ∧ foldr (∧) zss True) =
    foldr (∧) xss (b ∧ foldr (∧) yss (foldr (∧) zss True))
```

IsaRARE: Demo



Evaluation

Support all theories cvc5 has rewrites for:

- Added theory of SMT-LIB strings
- Added new term parsers to Isabelle (Strings, Sets, Arrays)
- Extended bit-vector term parser

Verify cvc5 RARE rules:

- Verify existing rules
- Iteratively use IsaRARE to develop new rewrite rules for bitvectors (85% reconstruction rate)

Results

	rewrites				
theory	old	new	untranslated	proven	automatic proof
Core	22	43	0	43	85%
Arithmetic	23	23	0	32	74%
Sets	0	7	0	7	100%
Arrays	0	4	0	4	100%
Strings	40	57	0	57	69%
Bit-vectors	0	168	3	122	66%

Table 1: Translation and verification rates per theory

Lessons Learned

- IsaRare discovered several bugs in the Rare rules used in cvc5
- Nitpick was particularly helpful in this respect
- For example:

```
(define-cond-rule str-substr-empty-range
((x String) (n Int) (m Int))
(>= n m)
(str.substr x n m)
"")
```

Lessons Learned

- IsaRare discovered several bugs in the Rare rules used in cvc5
- Nitpick was particularly helpful in this respect
- For example:

```
(define-cond-rule str-substr-empty-range
((x String) (n Int) (m Int))
~~(>= n m)~~
(str.substr x n m)
"")
```

Lessons Learned

- IsaRare discovered several bugs in the Rare rules used in cvc5
- Nitpick was particularly helpful in this respect
- For example:

```
(define-cond-rule str-substr-empty-range
((x String) (n Int) (m Int))
(>= 0 m)
(str.substr x n m)
"")
```

Contributions & Further work

Contributions Summary:

- Extended the RARE language and added 217 new RARE rules (mainly bit-vectors)
    - 85% of bv proof could be reconstructed from the go
- Developed IsaRARE, a plug-in for Isabelle
    - Efficiently re-uses existing SMT support in Isabelle
- Proved rules from all theories except BV correct
    - Many lemmas are proven automatically
    - The lemmas will be used in the reconstruction of cvc5 proofs in Isabelle

Parts of our work will be included in the official Isabelle distribution. The rest of IsaRARE will be submitted to the Archive of Formal Proofs.

Contributions & Further work

Further work:

- Prove all bitvector rewrites in Isabelle (25% remaining)
- This work is part of a bigger project of reconstructing cvc5 proofs in Isabelle/HOL. The generated lemmas will be used for this!
  - We use the Alethe proof format (some reconstruction already exists)
  - We added an Alethe back-end to cvc5
  - We prove Isabelle lemmas by showing every step in a corresponding Alethe proof holds
  - We also added functionality to use Isabelle as a proof checker for Alethe proof

Any questions?

Please feel free to contact me with any questions:

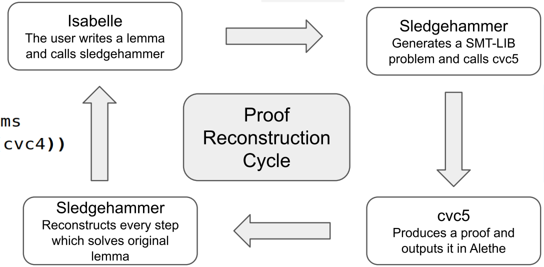
(a) Me, sometime this year

(b) A QR code

Figure 4: Means to contact me

Reconstruction of cvc5 proofs in Isabelle/HOL

```
lemma
  assumes "¬a ∧ b"
  shows "a ⟶ b"
```



Isabelle
The user writes a lemma and calls sledgehammer

Sledgehammer
Generates a SMT-LIB problem and calls cvc5

cvc5
Produces a proof and outputs it in Alethe

Sledgehammer
Reconstructs every step which solves original lemma

Proof Reconstruction Cycle

```
using assms
by (smt (cvc4))
```

```
(set-logic AUFLIA)
(declare-fun a$ () Bool)
(declare-fun b$ () Bool)
(assert (! (and (not a$) b$)
(assert (! (not (=> a$ b$))
(check-sat)
(get-proof)
```

```
unsat
(assume a0 (not (=> a$ b$)))
(assume a1 (and (not a$) b$))
(step t1 (cl a$) :rule not_implies1 :premises (a0))
(step t2 (cl (not a$)) :rule and :premises (a1))
(step t3 (cl) :rule resolution :premises (t1 t2))
```

Questions?
Message me at:
lachnitt@cs.stanford.edu