Reconstructing veriT Proofs in Isabelle/HOL

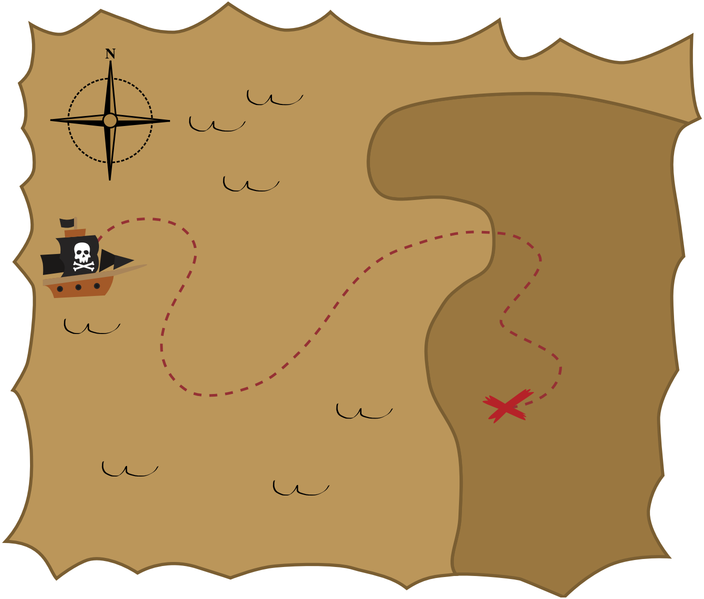
PxTP 2019

Natal – Brasil

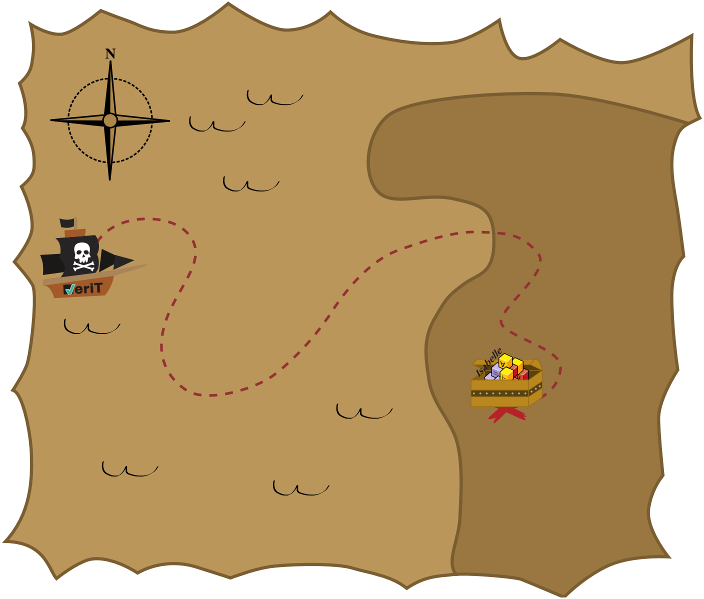
Mathias Fleury, **Hans-Jörg Schurr**

August 26, 2019

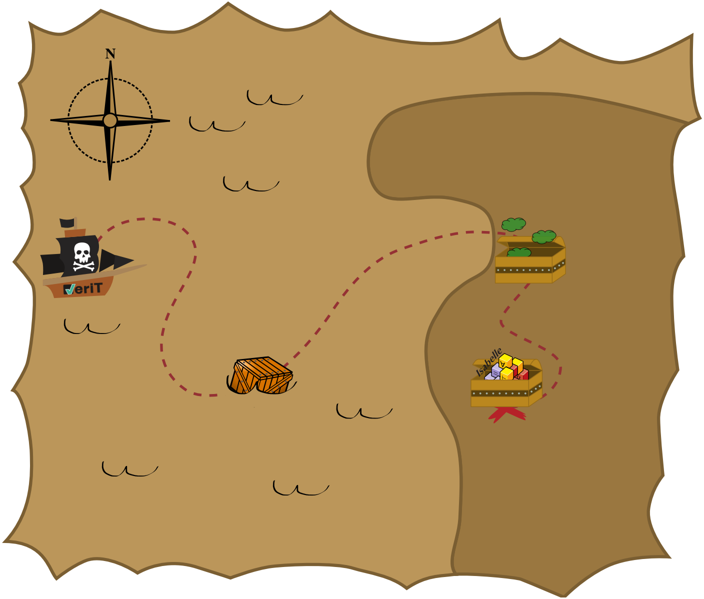
An Adventure



An Adventure

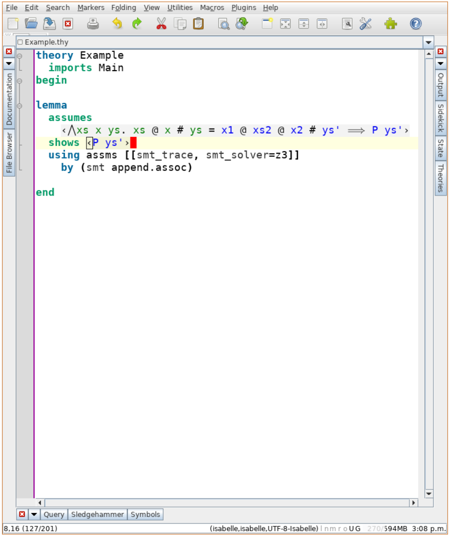


An Adventure



Proof Reconstruction in Isabelle/HOL

- Proof automation allows faster proof development
- One approach:
  1. Encode proof obligation into SMT-LIB
  2. Call an ATP
  3. Reconstruct the resulting proof
- Implemented by the `smt` method in Isabelle/HOL using Z3
  - Reconstruction can fail
  - Restricted to Z3
  - We want perfect reconstruction



Assisting Proof Construction

- Built-in methods
  - LCF approach
  - Checked by the prover kernel
  - In Isabelle: `auto`, `metis`, ...
- External automation:
  - `smt` with Z3 in Isabelle, SMTCoq
  - Hammers: Sledgehammer, HOL(y)Hammer, CoqHammer

The SMT Solver veriT

- Traditional CDCL(T) solver
- Supports:
  - Uninterpreted functions
  - Linear Arithmetic
  - Non-Linear Arithmetic
  - Quantifiers
  - ...
- Proof producing
- SMT-LIB input

```
(set-option :produce-proofs true)
(set-logic AUFLIA)
(declare-sort A$ 0)
(declare-sort A_list$ 0)
(declare-fun p$ (A_list$) Bool)
(declare-fun x1$ () A_list$)
(declare-fun x2$ () A$)
(declare-fun ys$ () A_list$)
(declare-fun xs2$ () A_list$)
(declare-fun cons$ (A$ A_list$) A_list$)
(declare-fun append$ (A_list$ A_list$) A_list$)
(assert (! (forall ((?v0 A_list$) (?v1 A_list$)
 (?v2 A_list$)) (= (append$ (append$ ?v0 ?v1) ?v2)
 (append$ ?v0 (append$ ?v1 ?v2)))) :named a0))
(assert (! (forall ((?v0 A_list$) (?v1 A$)
 (?v2 A_list$)) (=>
 (= (append$ ?v0 (cons$ ?v1 ?v2))
 (append$ x1$ (append$ xs2$ (cons$ x2$ ys$))))
 (p$ ys$))) :named a1))
(assert (! (not (p$ ys$)) :named a2))
(check-sat)
(get-proof)
```

Proofs from SMT Solvers

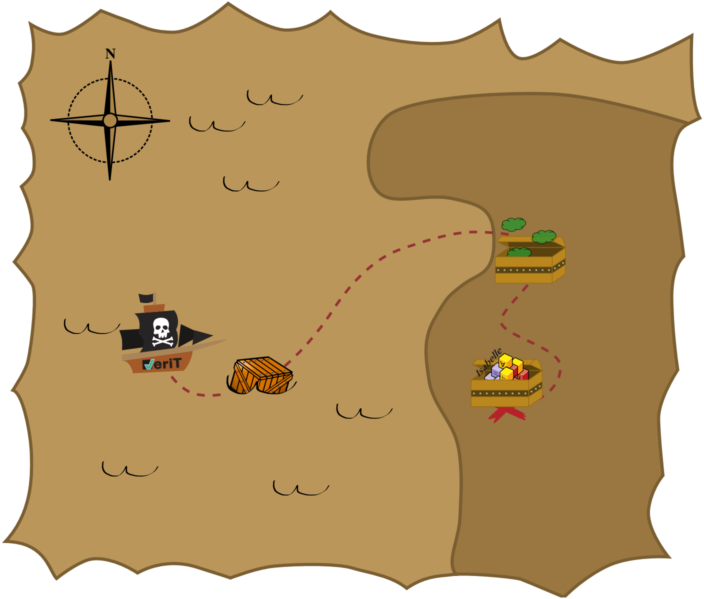
Use Cases

- Learning from proofs:
  - Guidance: (FE)MaLeCoP, rlCoP (reinforcement learning), ...
  - Instance filtering
- Unsatisfiable cores
- Finding interpolants
- Result certification if the problem is unsatisfiable
- Debugging

Proof Generating SMT Solvers

CVC4 (LFSC, no proofs for quantifiers), Z3 (SMT-LIB based proof trees, coarser steps, esp. for Skolemization), ArchSAT, ZenonModulo (Deducti), ...

Setting Sails



```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U)) (forall ((z2 U)) (p z2))))
...
(anchor :step t9 :args ((:= z2 veriT_vr4)))
(step t9.t1 (cl (= z2 veriT_vr4)) :rule refl)
(step t9.t2 (cl (= (p z2) (p veriT_vr4)))
  :rule cong :premises (t9.t1))
(step t9 (cl (= (forall ((z2 U)) (p z2))
  (forall ((veriT_vr4 U)) (p veriT_vr4)))) :rule bind)
...
(step t14 (cl (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule th_resolution :premises (t11 t12 t13))
(step t15 (cl (or (not (forall ((veriT_vr5 U))
  (p veriT_vr5))) (p a))) :rule forall_inst
  :args (:= veriT_vr5 a)))
(step t16 (cl (not (forall ((veriT_vr5 U)) (p veriT_vr5))) (
  :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U)) (forall ((z2 U)) (p z2))))
...
(anchor :step t9 :args ((:= z2 veriT_vr4)))
(step t9.t1 (cl (= z2 veriT_vr
(step t9.t2 (cl (= (p z2) (p veriT_vr4)))
  :rule cong :premises (t9.t1))
(step t9 (cl (= (forall ((z2 U)) (p z2))
  (forall ((veriT_vr4 U)) (p veriT_vr4)))) :rule bind)
...
(step t14 (cl (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule th_resolution :premises (t11 t12 t13))
(step t15 (cl (or (not (forall ((veriT_vr5 U))
  (p veriT_vr5))) (p a))) :rule forall_inst
  :args (:= veriT_vr5 a)))
(step t16 (cl (not (forall ((veriT_vr5 U)) (p veriT_vr5))) (
  :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Input assumptions

```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U)) (forall ((z2 U)) (p z2))))
...
(anchor :step t9 :args ((:= z2 veriT_vr4)))
(step t9.t1 (cl (= z2 veriT_ Simple step e refl)
(step t9.t2 (cl (= (p z2) (p veriT_vr4)))
  :rule cong :premises (t9.t1))
(step t9 (cl (= (forall ((z2 U)) (p z2))
  (forall ((veriT_vr4 U)) (p veriT_vr4)))) :rule bind)
...
(step t14 (cl (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule th_resolution :premises (t11 t12 t13))
(step t15 (cl (or (not (forall ((veriT_vr5 U))
  (p veriT_vr5))) (p a))) :rule forall_inst
  :args (:= veriT_vr5 a)))
(step t16 (cl (not (forall ((veriT_vr5 U)) (p veriT_vr5))) (
  :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U)) (forall ((z2 U)) (p z2))))
...
(anchor :step t9 :args ((:= z2 veriT_vr4)))
(step t9.t1 (cl (= z2 veriT_vr4)) :rule refl)
(step t9.t2 (cl (= (p z2) (p veriT_vr4)))
  :rule cong :premises (t9.t1))
(step t9 (cl (= (forall ((z2 U)) (p z2))
  (forall ((veriT_vr4 U)) ( Name T_vr4)))) :rule bind)
...
(step t14 (cl (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule th_resolution :premises (t11 t12 t13))
(step t15 (cl (or (not (forall ((veriT_vr5 U))
  (p veriT_vr5))) (p a))) :rule forall_inst
  :args (:= veriT_vr5 a)))
(step t16 (cl (not (forall ((veriT_vr5 U)) (p veriT_vr5))) (
  :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U)) (forall ((z2 U)) (p z2))))
...
(anchor :step t9 :args ((:= z2 veriT_vr4)))
(step t9.t1 (cl (= z2 veriT_vr4)) :rule refl)
(step t9.t2 (cl (= (p z2) (p veriT_vr4)))
  :rule cong :premises (t9.t1))
(step t9 (cl (= (forall ((z2 U)) (p z2))
  (forall ((veriT_vr4 U)) (p veriT_vr4)))) :rule bind)
...
(step t14 (cl (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule th_resolution :premises (t11 t12 t13))
(step t15 (cl (or (not (forall ((veriT_vr5
  (p veriT_vr5))) (p a))) :rule forall_inst
  :args (:= veriT_vr5 a)))
(step t16 (cl (not (forall ((veriT_vr5 U)) (p veriT_vr5))) (
  :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Introduced term

```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U)) (forall ((z2 U)) (p z2))))
...
(anchor :step t9 :args ((:= z2 veriT_vr4)))
(step t9.t1 (cl (= z2 veriT_vr4)) :rule refl)
(step t9.t2 (cl (= (p z2) (p veriT_vr4)))
  :rule cong :premises (t9.t1))
(step t9 (cl (= (forall ((z2 U Rule z2))
  (forall ((veriT_vr4 U)) (p veriT_vr4)))) :rule bind)
...
(step t14 (cl (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule th_resolution :premises (t11 t12 t13))
(step t15 (cl (or (not (forall ((veriT_vr5 U))
  (p veriT_vr5))) (p a))) :rule forall_inst
  :args (:= veriT_vr5 a)))
(step t16 (cl (not (forall ((veriT_vr5 U)) (p veriT_vr5))) (
  :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U)) (forall ((z2 U)) (p z2))))
...
(anchor :step t9 :args ((:= z2 veriT_vr4)))
(step t9.t1 (cl (= z2 veriT_vr4)) :rule refl)
(step t9.t2 (cl (= (p z2) (p ve Premises )))
  :rule cong :premises (t9.t1))
(step t9 (cl (= (forall ((z2 U)) (p z2))
  (forall ((veriT_vr4 U)) (p veriT_vr4)))) :rule bind)
...
(step t14 (cl (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule th_resolution :premises (t11 t12 t13))
(step t15 (cl (or (not (forall ((veriT_vr5 U))
  (p veriT_vr5))) (p a))) :rule forall_inst
  :args (:= veriT_vr5 a)))
(step t16 (cl (not (forall ((veriT_vr5 U)) (p veriT_vr5))) (
  :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U)) (forall ((z2 U)) (p z2))))
...
(anchor :step t9 :args ((:= z2 veriT_vr4)))
(step t9.t1 (cl (= z2 veriT_vr4)) :rule refl)
(step t9.t2 (cl (= (p z2) (p veriT_vr4)))
  :rule cong :premises (t9.t1))
(step t9 (cl (= (forall ((z2 U)) (p z2))
  (forall ((veriT_vr4 U)) (p veriT_vr4)))) :rule bind)
...
(step t14 (cl (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule th_resolution :premises (t11 t12 t13))
(step t15 (cl (or (not (forall ((veriT_vr5 U))
  (p veriT_vr5))) (p a))) :rule forall_inst
  :args (:= veriT_vr5 a)))
(step t16 (cl (not (forall ((veriT_vr5 U)) (p veriT_vr5))) (
  :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Start subproof

```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U)) (forall ((z2 U)) (p z2))))
...
(anc
(ste
(step t9.t2 (cl (= (p z2) (p veriT_vr4)))
  :rule cong :premises (t9.t1))
(step t9 (cl (= (forall ((z2 U)) (p z2))
  (forall ((veriT_vr4 U)) (p veriT_vr4)))) :rule bind)
...
(step t14 (cl (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule th_resolution :premises (t11 t12 t13))
(step t15 (cl (or (not (forall ((veriT_vr5 U))
  (p veriT_vr5))) (p a))) :rule forall_inst
  :args (:= veriT_vr5 a)))
(step t16 (cl (not (forall ((veriT_vr5 U)) (p veriT_vr5)))
  :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Skolemization is done by showing lemmas of the form

$$(\exists x.P[x]) = P[(\epsilon x.P)/x]$$

Implicit Steps

Some transformations are performed implicitly without generating steps:

- Symmetry of equality is applied: `(= a b)` becomes `(= b a)`
- Doubled negation is removed: `(cl (not (not a)))` becomes `(cl a)`
- Repeated literals are removed from clauses: `(cl (= a b) (= a b) (= (f a a) (f b b)))` becomes `(cl (= a b) (= (f a a) (f b b)))`

While those transformations are simple, they prohibit reconstruction by simple syntactic matching.

Setting Sails

Documentation

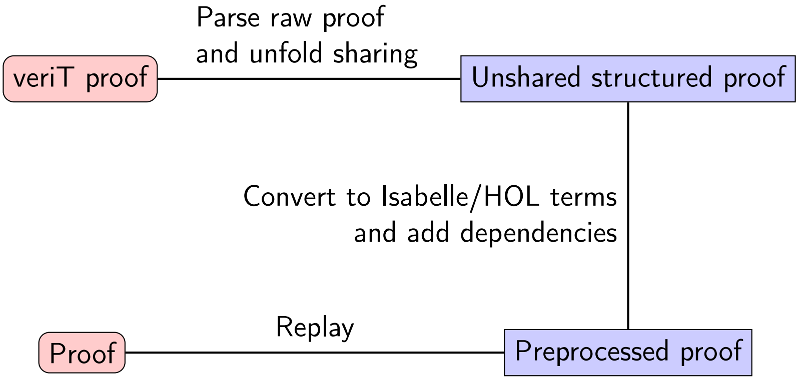
- Automatically generated: `--proof-format-and-exit`
  - Necessarily contains all rules
- Past publications (Besson et al. 2011, Déharbe et al. 2011, Barbosa et al. 2019)
- Collaboration helped a lot

Code Reuse

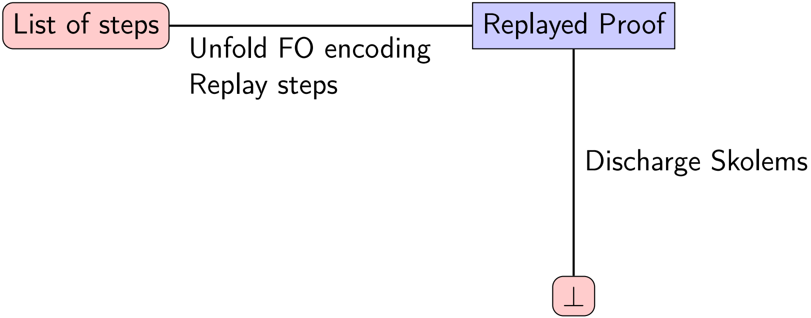
We could reuse some code of the reconstruction of Z3 proofs

- SMT-LIB parser
- Converter from SMT-LIB terms to Isabelle terms
- Reconstruction of resolution steps

The Reconstruction Inside Isabelle/HOL



The Reconstruction Inside Isabelle/HOL



Reconstruction

Direct Proof Rules

- Rule encoded as Isabelle thorems
- Reconstruct the rule upto implicit steps
- For $A \Rightarrow B$: We assume A and derive B'
- then we use `simp/fast/blast` to discharge $B' \Rightarrow B$
- Sometimes multiple versions of a lemma are tried:
  - $(\textit{if } \varphi \textit{ then } \psi_1 \textit{ else } \psi_2) \lor \neg\varphi \lor \psi_2$.
  - $(\textit{if } \neg\varphi \textit{ then } \psi_1 \textit{ else } \psi_2) \lor \varphi \lor \psi_2$

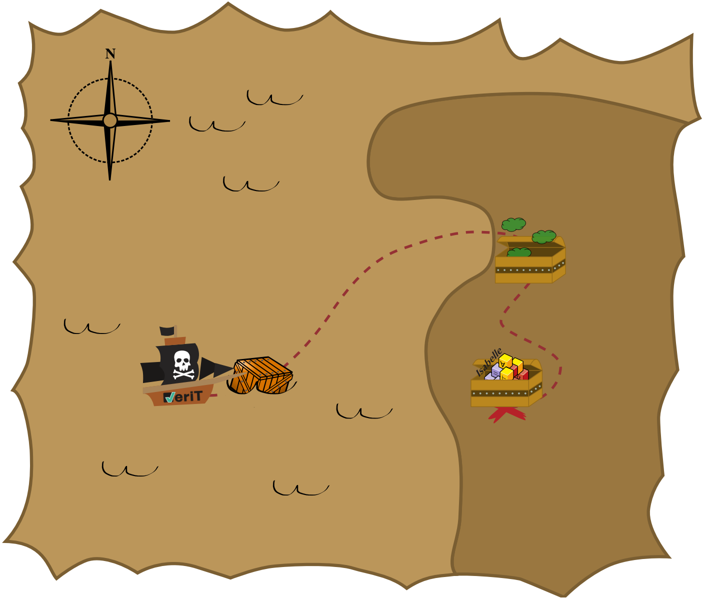
Hand-described Rules

- Call specific tactic for specific rules
- Some simplification (for speed)
- Terminal tactics

Challenges

- `arith` is too weak to reliably reconstruct the arithmetic rule
- The `connective_equiv` rule:
  - Encodes "trivial" truth about theory connectives
  - First attempt to solve on the propositional level
  - Then try automation
- Skolemization
- Implicit steps
- Bugs

Term Sharing



Term Sharing

- Proofs can be quite large
- Linear presentation unrolls shared terms
  - The choice terms introduced by Skolemization can be huge
- veriT proofs support optional sharing
- Utilizes `(!` t `:named` n`)` syntax of SMT-LIB

Proof Without Sharing

```
(assume h1 (and (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (= ?veriT.veriT__4 ?veriT.veri
c1 c2))))
(anchor :step t2 :args ((:= ?veriT.veriT__4 veriT_vr0) (:= ?veriT.veriT__3 veriT_vr1)))
(step t2.t1 (cl (= ?veriT.veriT__4 veriT_vr0)) :rule refl)
(step t2.t2 (cl (= ?veriT.veriT__3 veriT_vr1)) :rule refl)
(step t2.t3 (cl (= (= ?veriT.veriT__4 ?veriT.veriT__3) (= veriT_vr0 veriT_vr1))) :rule cong :premises (t2.
(step t2 (cl (= (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (= ?veriT.veriT__4 ?veriT.veri
((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)))) :rule bind)
(step t3 (cl (= (and (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (= ?veriT.veriT__4 ?veriT
(not (= c1 c2))) (and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1
cong :premises (t2))
(step t4 (cl (not (= (and (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (= ?veriT.veriT__4 ?
(not (= c1 c2))) (and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1
(and (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (= ?veriT.veriT__4 ?veriT.veriT__3)) (not
(and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c2)))) :rule equi
(step t5 (cl (and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c2))
th_resolution :premises (h1 t3 t4))
(anchor :step t6 :args ((:= veriT_vr0 veriT_vr2) (:= veriT_vr1 veriT_vr3)))
(step t6.t1 (cl (= veriT_vr0 veriT_vr2)) :rule refl)
(step t6.t2 (cl (= veriT_vr1 veriT_vr3)) :rule refl)
(step t6.t3 (cl (= (= veriT_vr0 veriT_vr1) (= veriT_vr2 veriT_vr3))) :rule cong :premises (t6.t1 t6.t2))
(step t6 (cl (= (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (forall ((veriT_v
(veriT_vr3 Client)) (= veriT_vr2 veriT_vr3)))) :rule bind)
(step t7 (cl (= (and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c
((veriT_vr2 Client) (veriT_vr3 Client)) (= veriT_vr2 veriT_vr3)) (not (= c1 c2))))) :rule cong :premises (
(step t8 (cl (not (= (and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (=
(forall ((veriT_vr2 Client) (veriT_vr3 Client)) (= veriT_vr2 veriT_vr3)) (not (= c1 c2))))) (not (and (for
Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c2)))) (and (forall ((veriT_vr2 Client) (v
(= veriT_vr2 veriT_vr3)) (not (= c1 c2)))) :rule equiv_pos2)
(step t9 (cl (and (forall ((veriT_vr2 Client) (veriT_vr3 Client)) (= veriT_vr2 veriT_vr3)) (not (= c1 c2))
th_resolution :premises (t5 t7 t8))
(step t10 (cl (forall ((veriT_vr2 Client) (veriT_vr3 Client)) (= veriT_vr2 veriT_vr3))) :rule and :premise
(step t11 (cl (not (= c1 c2))) :rule and :premises (t9))
(step t12 (cl (or (not (forall ((veriT_vr2 Client) (veriT_vr3 Client)) (= veriT_vr2 veriT_vr3))) (= c1 c2
```

Proof With Sharing

```
(assume h1 (! (and (! (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (! (= ?veriT.veriT__4 ?v
:named @p_3)) :named @p_2) (! (not (! (= c1 c2) :named @p_5)) :named @p_4)) :named @p_1))
(anchor :step t2 :args ((:= ?veriT.veriT__4 veriT_vr0) (:= ?veriT.veriT__3 veriT_vr1)))
(step t2.t1 (cl (! (= ?veriT.veriT__4 veriT_vr0) :named @p_6)) :rule refl)
(step t2.t2 (cl (! (= ?veriT.veriT__3 veriT_vr1) :named @p_7)) :rule refl)
(step t2.t3 (cl (! (= @p_3 (! (= veriT_vr0 veriT_vr1) :named @p_9)) :named @p_8)) :rule cong :premises (t2
(step t2 (cl (! (= @p_2 (! (forall ((veriT_vr0 Client) (veriT_vr1 Client)) @p_9) :named @p_11)) :named @p_
(step t3 (cl (! (= @p_1 (! (and @p_11 @p_4) :named @p_13)) :named @p_12)) :rule cong :premises (t2))
(step t4 (cl (! (not @p_12) :named @p_14) (! (not @p_1) :named @p_15) @p_13) :rule equiv_pos2)
(step t5 (cl @p_13) :rule th_resolution :premises (h1 t3 t4))
(anchor :step t6 :args ((:= veriT_vr0 veriT_vr2) (:= veriT_vr1 veriT_vr3)))
(step t6.t1 (cl (! (= veriT_vr0 veriT_vr2) :named @p_16)) :rule refl)
(step t6.t2 (cl (! (= veriT_vr1 veriT_vr3) :named @p_17)) :rule refl)
(step t6.t3 (cl (! (= @p_9 (! (= veriT_vr2 veriT_vr3) :named @p_19)) :named @p_18)) :rule cong :premises (
(step t6 (cl (! (= @p_11 (! (forall ((veriT_vr2 Client) (veriT_vr3 Client)) @p_19) :named @p_21)) :named @
(step t7 (cl (! (= @p_13 (! (and @p_21 @p_4) :named @p_23)) :named @p_22)) :rule cong :premises (t6))
(step t8 (cl (! (not @p_22) :named @p_24) (! (not @p_13) :named @p_25) @p_23) :rule equiv_pos2)
(step t9 (cl @p_23) :rule th_resolution :premises (t5 t7 t8))
(step t10 (cl @p_21) :rule and :premises (t9))
(step t11 (cl @p_4) :rule and :premises (t9))
(step t12 (cl (! (or (! (not @p_21) :named @p_27) @p_5) :named @p_26)) :rule forall_inst :args ((:= veriT_
veriT_vr3 c1)))
(step t13 (cl @p_27 @p_5) :rule or :premises (t12))
(step t14 (cl) :rule resolution :premises (t13 t10 t11))
```

Term Sharing In Practice

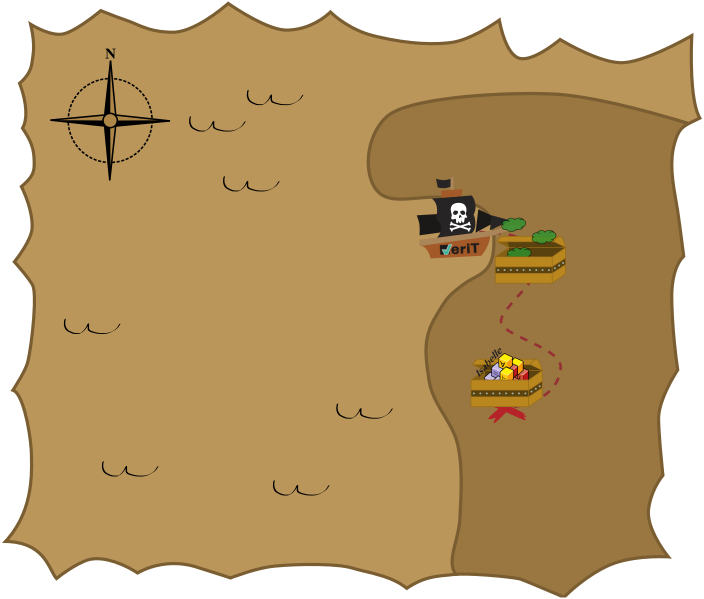
Where to introduce names?

- Perfect solution is hard to find
- Approximate: Terms which appear with two different parents get a name
  - $f(h(a), j(x, y)), g(h(a)), g(f(h(a), j(x, y)))$
  - $[f([h(a)]_{p_2}, j(x, y))]_{p_1}, [g(p_2)]_{p_3}, [g(p_1)]_{p_4}$
- Can be done in linear time thanks to perfect sharing

Isabelle/HOL side

- Isabelle/HOL unfolds everything except for Skolem terms
- Unfolding is currently done upfront
  - Better: Pipeline
- We introduced an optional syntax for Skolems as defined contants

Proof Rot



Proof Rot

Unintended, but sound divergences from the documented calculus.

Proof Rot

Unintended, but sound divergences from the documented calculus.

Example

The instantiation rule:

$$\forall x. p[x] \rightarrow p[t]$$

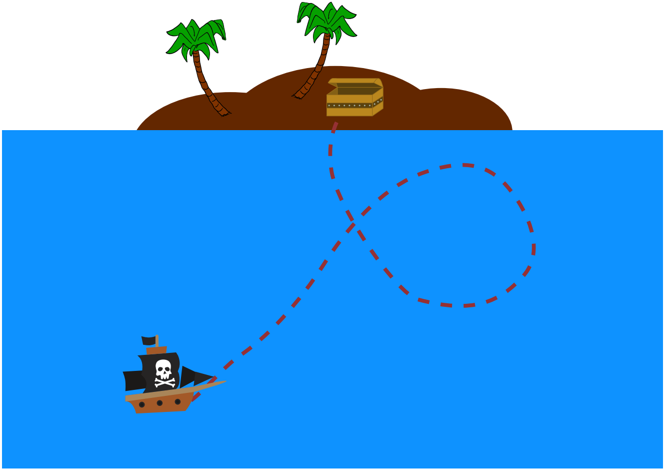
«If we have $\forall x.\ (p_1 \wedge p_2 \wedge p_3)$ we can produce $\forall x.\ (p_1 \wedge p_2 \wedge p_3) \rightarrow p_i[t]$.»

- Only a few lines of code change
- This change was done a while ago
- Without reconstruction we would never have known

Under some circumstances $p[x]$ is even a conjunctive normal form of a formula.

- Reconstruction forces you to stay honest

Where We Are Now



Land in sight!

Where We Are Now

Test on `smt` calls in the AFP:

- Hence, only theorems easy for Z3
- 502 calls, 451 proofs produced by veriT
- 4 proofs not reconstructed
- Average solving time 303ms
- Average reconstruction time 679.4ms

Sledgehammer test:

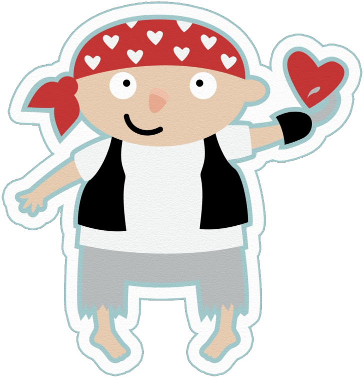
Theory	Ord. Res. Prover	Formal SSA
Found proofs	5019	5961
Z3-powered	90	109
veriT-powered	25	4
Oracle	9	63

Current and Future Work

- Fix Roadblocks
  - Linear arithmetic rules
  - The `connective_equiv` preprocessing rules
  - Refactoring
- Tool to debug proofs
  - Query subproofs
  - Selective unsharing
  - ...

Philosophical question: When do we trust our proofs?

- Verified checker
- Reconstruction in trusted kernel
- Unverified checker
- Human checkable
- Steps reproducible by other systems



Thank you for your attention!

- Questions? Suggestions?
- What would you like to see in the generated proofs?

References I

- Besson, Frédéric, Pascal Fontaine, and Laurent Théry (2011). "A Flexible Proof Format for SMT: a Proposal". In: *PxTP 2011*. Ed. by Pascal Fontaine and Aaron Stump, pp. 15–26.
- Déharbe, David, Pascal Fontaine, and Bruno Woltzenlogel Paleo (2011). "Quantifier Inference Rules for SMT proofs". In: *PxTP 2011*. Ed. by Pascal Fontaine and Aaron Stump, pp. 33–39. URL: `https://hal.inria.fr/hal-00642535`.
- Barbosa, Haniel et al. (2019). "Scalable Fine-Grained Proofs for Formula Processing". In: *J. Automated Reasoning*.