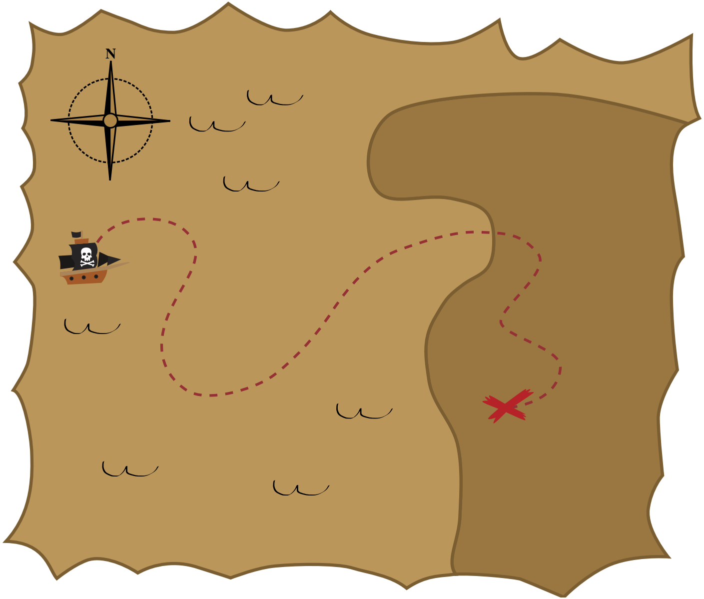
Better SMT Proofs for Easier Reconstruction

AITP 2019, Obergurgl – Austria

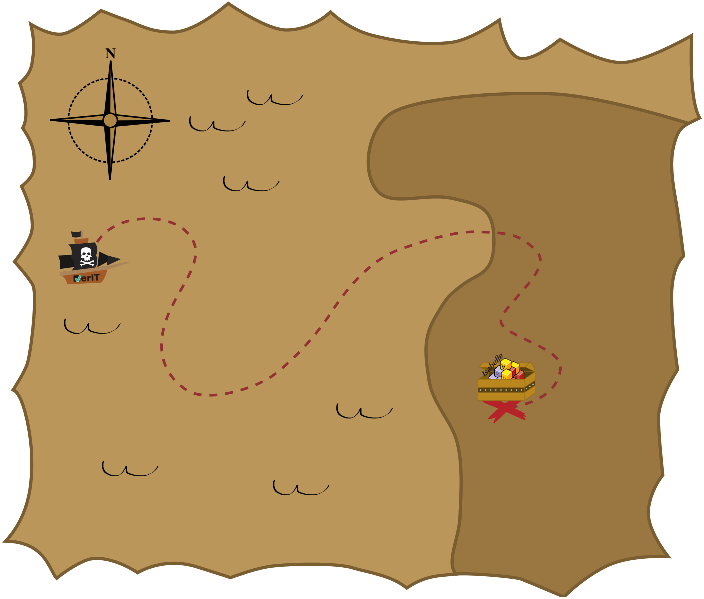
Haniel Barbosa, Jasmin Christian Blanchette, Mathias Fleury, Pascal Fontaine,
Hans-Jörg Schurr

April 10, 2019

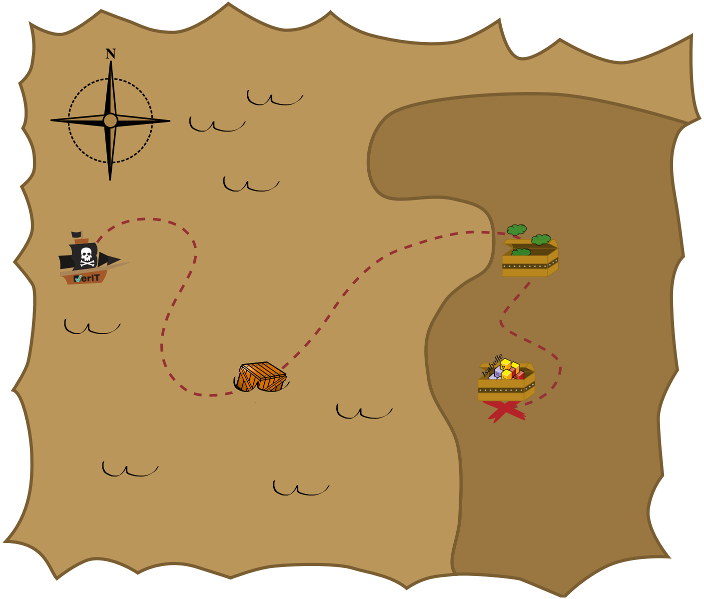
An Adventure



An Adventure

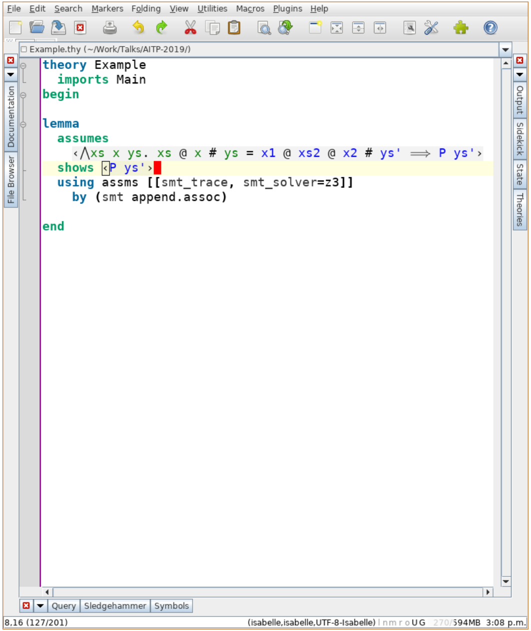


An Adventure



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` tactic 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

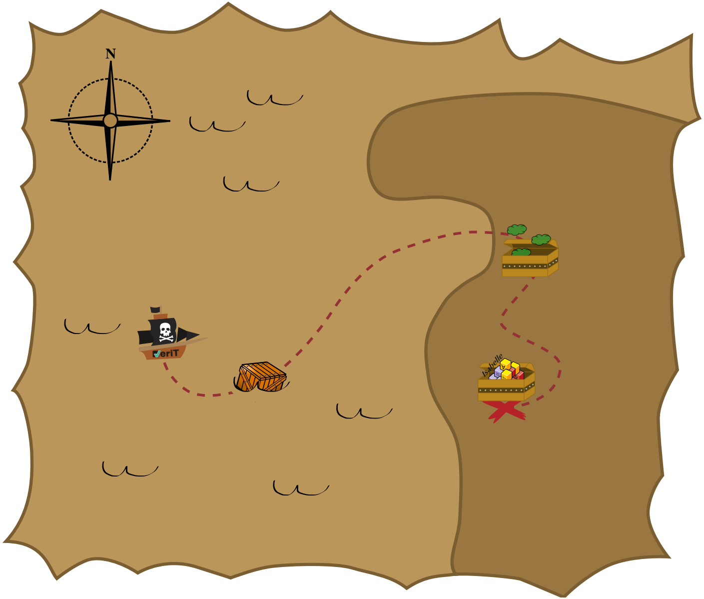
Use Cases

- Learning from proofs:
  - guidance: (FE)MaLeCoP, rlCoP (reinforcement learning), ...
  - see also Daniel's talk
- 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), veriT, 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))) (p a))
          :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

veriT's Proofs

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_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))) (p a))
          :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

veriT's Proofs

```
(assume h1 (not (p a)))
(assume h2 (forall ((z1 U ... ((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))) (p a))
          :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Simple step

veriT's Proofs

```
(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))) (p a))
          :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Name

veriT's Proofs

```
(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))) (p a))
          :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Introduced term

veriT's Proofs

```
(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))) (p a))
          :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Rule

```
(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 Premises (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))) (p a))
          :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

veriT's Proofs

```
(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)))                     ses (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))) (p a))
          :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Context annotation

veriT's Proofs

```
(assume h1 (not (p a)))
(assu                                                  )))
...
(ancl
(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))) (p a))
          :rule or :premises (t15))
(step t17 (cl) :rule resolution :premises (t16 h1 t14))
```

Skolemization is done by showing lemmas of the form

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

Setting Sails

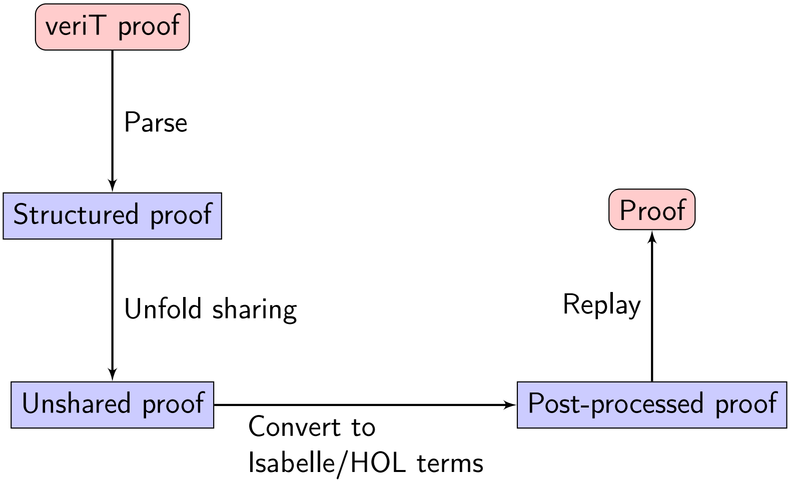
Collaborate

Given that we are both developers of the SMT solver and the reconstruction, many problems (bugs, unclarities, etc.) can be solved on short notice.

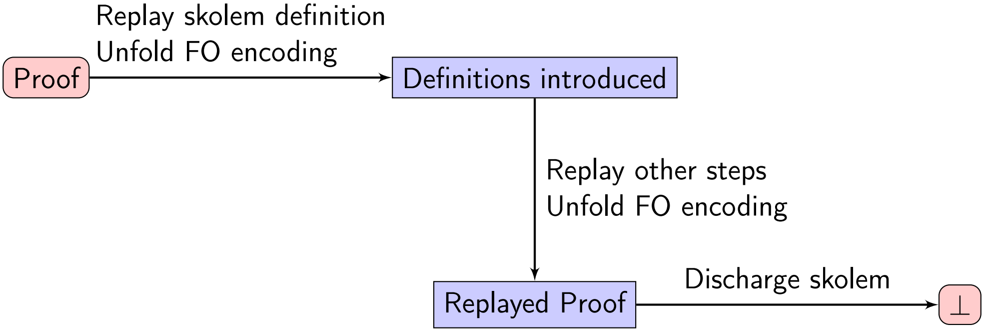
Documentation

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

The Reconstruction Inside Isabelle/HOL



The Reconstruction Inside Isabelle/HOL



Reconstruction

Direct Proof Rules

- Assume $A \Rightarrow B$ is applied
- We assume A
- We derive B'
- then `simp`/`fast`/`blast` to discharge $B' \Rightarrow B$

Hand-described Rules

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

Reconstruction

Direct Proof Rules

- Assume $A \Rightarrow B$ is applied
- We assume A
- We derive B'
- then `simp`/`fast`/`blast` to discharge $B' \Rightarrow B$

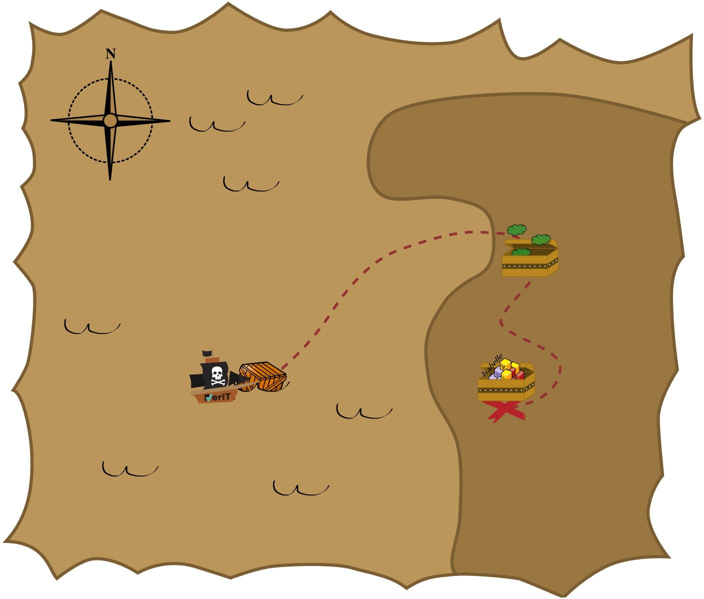
Hand-described Rules

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

Challenges

- `arith` is too weak to reliably reconstruct the current arithmetic step
- Skolemization
- The `connective_equiv` rule:
  - Encodes "trivial" truth about theory connectives
  - First attempt to solve on the propositional level
  - Then try automation
- Implicit steps
  - Order of $=$ is freely changed
  - Step simplification:
    $a \approx b \wedge a \approx b \Rightarrow f(a, a) \approx f(b, b)$
    $a \approx b \Rightarrow f(a, a) \approx f(b, b)$
  - Double negation is eliminated

Weight: Proof Size



Weight: Proof Size

- Proofs are often huge
- 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

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 where the name is used.

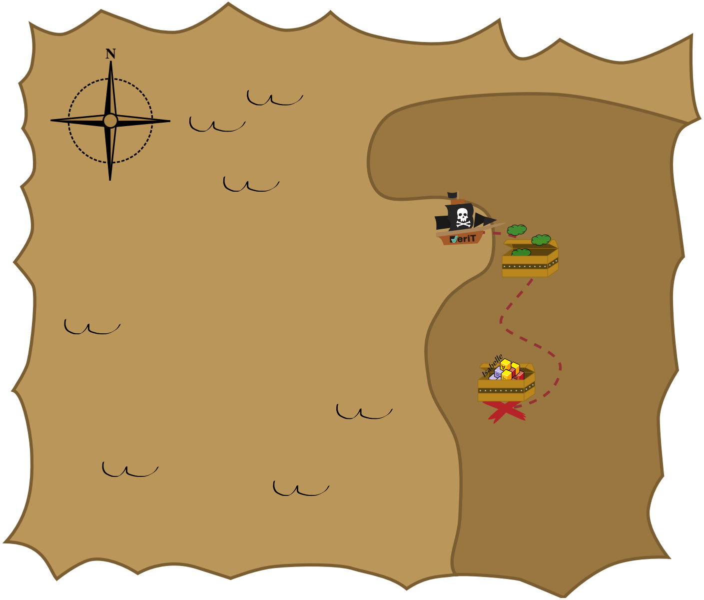
Proof Without Sharing

```
(assume h1 (and (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (= ?veriT.veriT__4 ?veriT.veriT__3)) (not (=
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.t1 t2.t2))
(step t2 (cl (= (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (= ?veriT.veriT__4 ?veriT.veriT__3)) (forall
((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.veriT__3))
(not (= c1 c2))) (and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c2))))) :rule
cong :premises (t2))
(step t4 (cl (not (= (and (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (= ?veriT.veriT__4 ?veriT.veriT__3))
(not (= c1 c2))) (and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c2))))) (not
(and (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (= ?veriT.veriT__4 ?veriT.veriT__3)) (not (= c1 c2))))
(and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c2)))) :rule equiv_pos2)
(step t5 (cl (and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c2)))) :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)) :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_vr2 Client)
(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 c2))) (and (forall
((veriT_vr2 Client) (veriT_vr3 Client)) (= veriT_vr2 veriT_vr3)) (not (= c1 c2))))) :rule cong :premises (t6))
(step t8 (cl (not (= (and (forall ((veriT_vr0 Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c2))) (and
(forall ((veriT_vr2 Client) (veriT_vr3 Client)) (= veriT_vr2 veriT_vr3)) (not (= c1 c2))))) (not (and (forall ((veriT_vr0
Client) (veriT_vr1 Client)) (= veriT_vr0 veriT_vr1)) (not (= c1 c2)))) (and (forall ((veriT_vr2 Client) (veriT_vr3 Client))
(= 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)))) :rule
th_resolution :premises (t5 t7 t8))
(step t10 (cl (forall ((veriT_vr2 Client) (veriT_vr3 Client)) (= veriT_vr2 veriT_vr3))) :rule and :premises (t9))
(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))) :rule forall_inst
:args ((:= veriT_vr2 c2) (:= veriT_vr3 c1)))
(step t13 (cl (not (forall ((veriT_vr2 Client) (veriT_vr3 Client)) (= veriT_vr2 veriT_vr3))) (= c1 c2)) :rule or :premises (t12))
```

Proof With Sharing

```
(assume h1 (! (and (! (forall ((?veriT.veriT__4 Client) (?veriT.veriT__3 Client)) (! (= ?veriT.veriT__4 ?veriT.veriT__3)
: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.t1 t2.t2))
(step t2 (cl (! (= @p_2 (! (forall ((veriT_vr0 Client) (veriT_vr1 Client)) @p_9) :named @p_11)) :named @p_10)) :rule bind)
(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 (t6.t1 t6.t2))
(step t6 (cl (! (= @p_11 (! (forall ((veriT_vr2 Client) (veriT_vr3 Client)) @p_19) :named @p_21)) :named @p_20)) :rule bind)
(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_vr2 c2) (:=
veriT_vr3 c1)))
(step t13 (cl @p_27 @p_5) :rule or :premises (t12))
(step t14 (cl) :rule resolution :premises (t13 t10 t11))
```

Proof Rot



At the beginning everything was fine and veriT produced the step:

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

At the beginning everything was fine and veriT produced the step:

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

Then: «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

Proof Rot

At the beginning everything was fine and veriT produced the step:

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

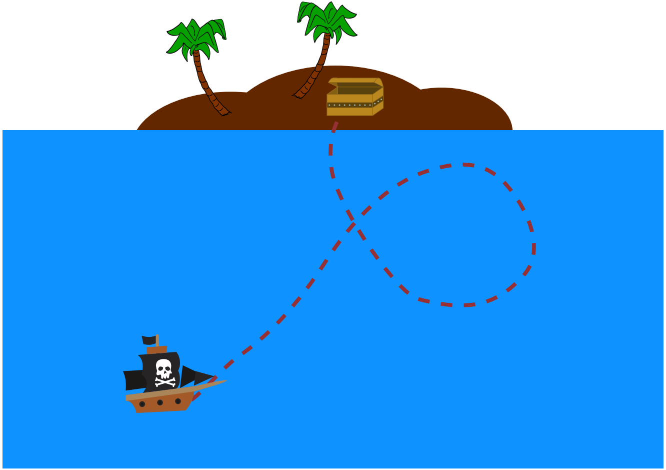
Then: «If we have $\forall x. \, (p_1 \wedge p_2 \wedge p_3)$ we can produce $\forall x. \, (p_1 \wedge p_2 \wedge p_3) \rightarrow p_i[t]$.»

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

Since then: Under some circumstances $p[x]$ is a CNF of another 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
- 498 calls, 447 proofs produced by veriT
- 443 proofs reconstructed
- Average solving time 303ms
- Average reconstruction time 679.4ms

Where We Are Now

Test on `smt` calls in the AFP:

- Hence, only theorems easy for Z3
- 498 calls, 447 proofs produced by veriT
- 443 proofs reconstructed
- Average solving time 303ms
- Average reconstruction time 679.4ms

Outlook

- Perfect reconstruction
- Isabelle/HOL as a certifier
- Long term: A widely accepted format

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*.