Robust, Semi-Intelligible Isabelle Proofs from ATP Proofs

Steffen Smolka
Advisor: Jasmin Blanchette
well suited for large formalizations but require intensive manual labor

fully automatic but no proof management
well suited for large formalizations
but require intensive manual labor

Sledgehammer

fully automatic but no proof management
lemma "length (tl xs) ≤ length xs"

proof (prove): step 0

goal (1 subgoal):
  1. length (tl xs) ≤ length xs
lemma "length (tl xs) ≤ length xs"

Sledgehammering...
lemma "length (tl xs) ≤ length xs"

Sledgehammering...
"spass": Try this: by (metis diff_le_self length_tl) (17 ms).
lemma "length (tl xs) ≤ length xs"

sledgehammer

Sledgehammering...
"spass": Try this: by (metis diff_le_iff length_tl) (17 ms).
lemma "length (tl xs) ≤ length xs"
by (metis diff_le_self length_tl)
Exploit ATPs,
but don’t trust them.

**LCF Principle (Robin Milner):**
Have all proofs checked by the inference kernel.

⇒ ATP proofs must be *reconstructed* in Isabelle.
Approach A: Metis One-Liners

lemma "length (tl xs) ≤ length xs"
by (metis diff_le_le_self length_tl)

proof method

lemmas
**Approach A: Metis One-Liners**

```isar
lemma "length (tl xs) ≤ length xs"
  by (metis diff_le_self length_tl)
```

**proof method**

**lemmas**

**external ATPs:** find proof given 100s of facts

**Metis:** re-find proof given only necessary facts
**Approach A: Metis One-Liners**

**lemma** "length (tl xs) ≤ length xs"

**by** (metis diff_le_self length_tl)

**proof method**

**lemmas**

**external ATPs:** find proof given 100s of facts

**Metis:** re-find proof given only necessary facts

+ usually fast and reliable
+ lightweight
- cryptic
- sometimes slow (several seconds)
- on avg. 5% “loss”
Approach B: Detailed Isar Proofs

lemma "length (tl xs) ≤ length xs"
proof -
  have "∀x1 x2. (x1::nat) - x2 - x1 = 0 - x2"
    by (metis comm_monoid_diff_class.diff_cancel diff_right_commute)
  hence "length xs - 1 - length xs = 0"
    by (metis zero_diff)
  hence "length xs - 1 ≤ length xs"
    by (metis diff_is_0_eq)
  thus "length (tl xs) ≤ length xs"
    by (metis length_tl)
qed
Approach B: Detailed Isar Proofs

lemma "length (tl xs) ≤ length xs"
proof -
  have "∀x1 x2. (x1∷nat) - x2 - x1 = 0 - x2"
    by (metis comm_monoid_diff_class.diff_cancel diff_right_commute)
  hence "length xs - 1 - length xs = 0"
    by (metis zero_diff)
  hence "length xs - 1 ≤ length xs"
    by (metis diff_is_0_eq)
  thus "length (tl xs) ≤ length xs"
    by (metis length_tl)
qed

+ faster than one-liners
+ 100% reconstruction (in principle)
+ self-explanatory
- technically more challenging
Challenge 1:
Resolution proofs are by contradiction
"sin against mathematical exposition" (Knuth et al. 1989)

Jasmin Blanchette

Challenge 2:
Skolemization - introduce new symbols during proof

Challenge 3:
Type Annotations - make Isabelle understand its own output

Challenge 4:
Preplay & Optimization - test and improve proofs
Challenge 2: Skolemization
$\forall X. \exists Y. \ p(X, Y)$

$\forall X. \ p(X, y(X))$

Skolemization

signature is extended
\[
\forall X. \exists Y. p(X, Y) \quad \quad \quad \text{Skolemization}
\]

\[
\forall X. p(X, y(X))
\]

Signature is extended

\[
\forall X. \exists Y. p(X, Y) \quad \quad \quad \text{Ax. of Choice}
\]

\[
\exists y. \forall X. p(X, y(X))
\]
\[ \forall X. \exists Y. \ p(X, Y) \]
\[ \forall X. \ p(X, y(X)) \]

Skolemization

Signature is extended

\[ \forall X. \exists Y. \ p(X, Y) \]
\[ \exists y. \forall X. \ p(X, y(X)) \]

Ax. of Choice

Signature is extended

obtain \( y \) where \( \forall X. \ p(X, y(X)) \)
<steps with \textbf{reduced} sig.>

\[
\forall X. \exists Y. p(X, Y) \\
\exists y. \forall X. p(X, y(X)) \quad \text{Ax. of Choice}
\]

<steps with \textbf{extended} sig.>
<steps with **extended** sig.>

\[
\exists y. \forall X. \; p(X, y(X)) \\
\hline
\forall X. \; \exists Y. \; p(X, Y)
\]

Ax. of Choice

<steps with **reduced** sig.>
\[
\forall y. \exists X. \neg p(X, y(X)) \\
\exists X. \forall Y. \neg p(X, Y)
\]

Ax. of Choice

<steps with \textit{extended} sig.>

<steps with \textit{reduced} sig.>
∀y. ∃X. ¬p(X, y(X))  

Contrap. of

∃X. ∀Y. ¬p(X, Y)  Ax. of Choice

<steps with reduced sig.>
∀y. ∃X. ¬p(X, y(X))  

\[ \exists X. \forall Y. \neg p(X, Y) \]  

Contrap. of Ax. of Choice

{ fix y

<steps with extended sig.>

have ∃X. ¬p(X, y(X)) } 

hence ∃X. ∀Y. ¬p(X, Y)

<steps with reduced sig.>
Challenge 3:
Type Annotations
Make Isabelle understand its own output
2^{nat} + nat → nat → nat 2^{nat} = nat → nat → bool 4^{nat}

\[
\text{print}
\]

2 + 2 = 4
$$2^{nat} + nat \rightarrow nat \rightarrow nat \quad 2^{nat} = nat \rightarrow nat \rightarrow bool \quad 4^{nat}$$

print

$$2 + 2 = 4$$

parse

$$2^\alpha + \alpha \rightarrow \alpha \rightarrow \alpha \quad 2^\alpha = \alpha \rightarrow \alpha \rightarrow bool \quad 4^{\alpha}$$

where $\alpha : numeral$  

Unprovable
\[
\begin{align*}
2 \text{nat} & \rightarrow \text{nat} \rightarrow \text{nat} \\
2 \text{nat} & = \text{nat} \rightarrow \text{nat} \rightarrow \text{bool} \\
4 \text{nat} & \\
\end{align*}
\]
\[ 2 \text{nat} + \text{nat} \to \text{nat} \to \text{nat} = \text{nat} \to \text{nat} \to \text{bool} 4 \text{nat} \]

\[ 2 + 2 = 4 \]

\[ 2^\alpha + ^\alpha \to ^\alpha \to ^\alpha = ^\alpha \to ^\alpha \to \text{bool} 4^\alpha \]

where \( \alpha : \text{numeral} \)
\[ 2^{\text{nat}} + ^{\text{nat} \to \text{nat} \to \text{nat}} 2^{\text{nat}} = ^{\text{nat} \to \text{nat} \to \text{bool}} 4^{\text{nat}} \]

\[ \text{print} \]

\[ (2^{\text{nat}}) + 2 = 4 \]

\[ \text{parse} \]

\[ 2^{\text{nat}} + ^{\text{nat} \to \text{nat} \to \text{nat}} 2^{\text{nat}} = ^{\text{nat} \to \text{nat} \to \text{bool}} 4^{\text{nat}} \]
Goal: Calculate a set of annotations that is

(A) Complete: reparsing term must not change its type

(B) Minimal: annotations must impair readability as little as possible
\[
\begin{align*}
\text{type inference} & \quad \approx \quad \text{parsing} \\
& \quad \downarrow \quad \downarrow \\
& \quad f^{\text{nat} \to \text{int} \to \text{bool}} \quad x^{\text{nat}} \quad y^{\text{int}} \\
& \quad \text{type erasure} \quad \approx \quad \text{printing} \\
& \quad \downarrow \\
& \quad f^{-} \quad x^{-} \quad y^{-} \\
& \quad \text{type inference} \quad \approx \quad \text{parsing} \\
& \quad \downarrow \\
& \quad f^{\alpha \to \beta \to \gamma} \quad x^{\alpha} \quad y^{\beta} \\
& \quad \text{matching} \\
& \quad \downarrow \\
\sigma & = \{ \alpha \mapsto \text{nat}, \beta \mapsto \text{int}, \gamma \mapsto \text{bool} \} 
\end{align*}
\]
\[
\begin{align*}
\text{type inference} & \approx \text{parsing} \\
\text{type erasure} & \approx \text{printing} \\
\sigma = \{ \alpha \mapsto \text{nat}, \beta \mapsto \text{int}, \gamma \mapsto \text{bool} \}
\end{align*}
\]

\text{Set of ann. complete IFF it covers } \text{Dom}(\sigma)
\( f : \text{nat} \to \text{int} \to \text{bool} \)

\( \alpha \to \beta \to \gamma \)

\( f(x, y) \)
(f : nat -> int -> bool) x y
\((f : \text{nat} \to \text{int} \to \text{bool}) \: x \: y\)
(f : nat → int → bool) x y
f (x : nat) (y : int) : bool
\( f : \text{nat} \to \text{int} \to \text{bool} \)

\[
(f : \text{nat} \to \text{int} \to \text{bool}) \ x \ y \\
f (x : \text{nat}) (y : \text{int}) : \text{bool}
\]
\[(f: \text{nat} \rightarrow \text{int} \rightarrow \text{bool}) \ x \ y\]
\[f \ (x: \text{nat}) \ (y: \text{int}) : \text{bool}\]
\[(f \ (x: \text{nat}) : \text{int} \rightarrow \text{bool}) \ y\]
Which set of annotations is the best?

How do we compute it efficiently?
Which set of annotations is the best?

cost of $t^\tau :=$
Which set of annotations is the best?

\[
\text{cost of } t^\tau :=
\]

(size of \( \tau \), size of \( t \), postindex of \( t^\tau \)) → small annotations at the beginning
Which set of annotations is the best?

cost of $t^\tau :=$

$$(\text{size of } \tau, \text{size of } t, \text{postindex of } t^\tau) \rightarrow \text{small annotated terms}$$

$\leq \text{lexiographically} + \text{component-wise} \rightarrow \text{small annotations}$$
How do we compute it efficiently?
How do we compute it efficiently?

Instance of **Weighted Set Cover Problem**:

- Finite Universe $U$ \[\rightarrow \text{Dom}(\sigma)\]
- Family $S \subseteq 2^U$ \[\rightarrow \text{Possible Annotations}\]
How do we compute it efficiently?

Instance of **Weighted Set Cover Problem**:

- Finite Universe $U$
- Family $S \subseteq 2^U$
- Find $\{U_1,\ldots,U_n\} \subseteq S$ such that
  - $U_1 \cup \ldots \cup U_n = U$ → Completeness
  - cost $\{U_1,\ldots,U_n\}$ minimal → Readability

$\rightarrow \text{Dom}(\sigma)$
$\rightarrow$ Possible Annotations
SCP is **NP-complete**  $\implies$ settle for Approximation

**Reverse-Greedy Alg.** calculates local min:

- start with all annotations
- repeatedly remove the most expensive superfluous annotation
Challenge 4: Preplay & Optimization
Proof Preplay

Generated proofs are only useful if they...

• work
Proof Preplay

Generated proofs are only useful if they...

• work
• are reasonably fast
Proof Preplay

Generated proofs are only useful if they...
  • work
  • are reasonably fast

Let the computer find out!

→ Present proofs with “preplay” information
lemma "x ∪ -x = -x ∪ -(x)"

Sledgehammering...
lemma "x \cup -x = -x \cup (-x)"

sledgehammer

Try this: by (metis huntington sup_assoc sup_comm) (> 3 s).

timeout
lemma "x \cup -x = -x \cup -(x)"

sledgehammer

Try this: by (metis huntington sup_assoc sup_comm) (> 3 s).

Structured proof (43 steps, 1.34 s):

proof -

have f1: "\forall x_1 \ x_2. \ - (x_1 \cup x_2) \cup - (x_1 \cup -x_2) = x_1"
  by (metis huntington sup_assoc sup_comm)

have f2: "\forall x_1 \ x_2 \ x_3. \ x_1 \cup (x_2 \cup x_3) = x_3 \cup (x_1 \cup x_2)"
  by (metis sup_assoc sup_comm)

have f3: "\forall x_1 \ x_2 \ x_3. \ x_1 \cup (x_2 \cup x_3) = x_2 \cup x_1 \cup x_3"
  by (metis sup_assoc sup_comm)

have f4: "\forall x_1 \ x_2 \ x_3. \ x_1 \cup (x_2 \cup x_3) = x_3 \cup (x_2 \cup x_1)"

timeout
Approach A: Feed proof text to Isabelle

+ close to reality

- expensive

- no timings for individual steps
Approach A: Feed proof text to Isabelle
  + close to reality
  - expensive
  - no timings for individual steps

Approach B: Simulate replay on ML-level
  - not the real thing (no printing, no parsing)
  + timings for each step
Proof Compression

Try this: by (metis huntington sup_assoc sup_comm) (> 3 s).

Structured proof: (43 steps, 1.34 s):

proof -
    have f1: "\forall x_1 \ x_2. - (- x_1 \uplus x_2) \uplus - (- x_1 \uplus - x_2) = x_1" 
       by (metis huntington sup_comm)
    have f2: "\forall x_1 \ x_2 \ x_3. x_1 \uplus (x_2 \uplus x_3) = x_3 \uplus (x_1 \uplus x_2)"
Proof Compression

```
lemma "x ∪ -x = -x ∪ -(x)"
sledgehammer (huntington sup_assoc sup_comm)

Try this: by (metis huntington_sup_assoc sup_comm) (> 3 s).
Structured proof (33 steps, 754 ms):
proof -
  have f1: "∀x₁ x₂. (- x₁ ∪ x₂) ∪ (- x₁ ∪ - x₂) = x₁"
    by (metis huntington_sup_comm)
  have f2: "∀x₁ x₂ x₃. x₁ ∪ (x₂ ∪ x₃) = x₃ ∪ (x₁ ∪ x₂)"
```
A1 ⊢ I
I, A2 ⊢ C
$A_1 \vdash I$
$I, A_2 \vdash C$

merger

$A_1, A_2 \vdash C$
\[ t_1 + t_2 \geq t' \quad ? \]

Does merger save time?

\[
\begin{align*}
A_1 \vdash I \\
I, A_2 \vdash C \\
\text{merger} \\
A_1, A_2 \vdash C
\end{align*}
\]
A1 ⊢ I
I, A2 ⊢ C
merger
A1, A2 ⊢ C

(t1 + t2)(1+\text{bonus}) \geq t' ?

Does merger save time?
A1 ⊢ F1
A1, F1 ⊢ F2
A2, F1 ⊢ F3
F1, F2, F3 ⊢ C
A1 ⊢ F1
A1, F1 ⊢ F2
A2, F1 ⊢ F3
F1, F2, F3 ⊢ C
A1 ⊢ F1
A1, F1 ⊢ F2
A2, F1 ⊢ F3
F1, F2, F3 ⊢ C
A1 ⊢ F1
A1, F1 ⊢ F2
A2, F1 ⊢ F3
F1, F2, F3 ⊢ C

A1 ⊢ F1
A1, F1 ⊢ F2
F1, F2, A2 ⊢ C
A1 ⊢ F1
A1, F1 ⊢ F2
A2, F1 ⊢ F3
F1, F2, F3 ⊢ C
A1 ⊢ F1
A1, F1 ⊢ F2
F1, F2, A2 ⊢ C
A1 ⊢ F1
A1, F1 ⊢ F2
A2, F1 ⊢ F3
F1, F2, F3 ⊢ C

A1 ⊢ F1
A1, F1 ⊢ F2
F1, F2, A2 ⊢ C

F1, A1, A2 ⊢ C
Axioms

A1 \vdash F1
A1, F1 \vdash F2
A2, F1 \vdash F3
F1, F2, F3 \vdash C

A1 \vdash F1
A1, F1 \vdash F2
F1, F2, A2 \vdash C
F1, A1, A2 \vdash C
\[ A1 \vdash F1 \]
\[ A1, F1 \vdash F2 \]
\[ A2, F1 \vdash F3 \]
\[ F1, F2, F3 \vdash C \]
\[ A1, F1 \vdash F2 \]
\[ F1, F2, A2 \vdash C \]
\[ A1 \vdash F1 \]
\[ F1, A1, A2 \vdash C \]

A1, A2 \vdash C
Stop when given compression factor is reached
Stop when given compression factor is reached

Eliminate “large” steps first
Stop when given compression factor is reached

Eliminate “large” steps first

Generalizations:
- eliminate subproofs
- eliminate steps with k successors
Beyond Metis
“Sledgehammer Try0”

time

metis

→ method
Beyond Metis
“Sledgehammer Try0”

metis simp auto fastforce force arith blast

method
time
Beyond Metis
“Sledgehammer Try0”

metis  simp  auto  fastforce  force  arith  blast

FAIL

time

method
Beyond Metis
“Sledgehammer Try0”
Beyond Metis
“Sledgehammer Try0”

+ speedup  
+ robustness
Fact Minimization

Metis knows nothing.
Simp, Auto, ... know about lists, numbers, ...
Fact Minimization

**Metis** knows nothing.
**Simp, Auto, ...** know about lists, numbers, ...

```
... using g1 g2 l1 l2 by metis
```

```
... using g2 by simp
```
Fact Minimization

Metis knows nothing.
Simp, Auto, ... know about lists, numbers, ...

... using g1 g2 l1 l2 by metis

... using g2 by simp

+ may eliminate intermediate steps
+ speedup
lemma

 fixes a :: real and b :: real
 assumes a0: "0<a" and a1: "a<1"
        and b0: "0<b" and b1: "b<1"
 shows "a+b - a*b > 0"
lemma

fixes a :: real and b :: real
assumes a0: "0<a" and a1: "a<1"
    and b0: "0<b" and b1: "b<1"
shows "a+b - a*b > 0"
lemma

fixes a :: real and b :: real
assumes a0: "0<a" and a1: "a<1"
and b0: "0<b" and b1: "b<1"
shows "a+b - a*b > 0"
lemma
    fixes a :: real and b :: real
    assumes a0: "0<a" and a1: "a<1"
    and b0: "0<b" and b1: "b<1"
    shows "a+b - a*b > 0"

by (metis a0 a1 add_less_cancel_left b0
    comm_monoid_add_class.add.right_neutral
    comm_monoid_mult_class.mult.left_neutral
    comm_semiring_1_class.normalizing_semiring_rules(24)
    diff_add_cancel pos_add_strict real_mult_less_iff1)
lemma
   fixes a :: real and b :: real
   assumes a0: "0<a" and a1: "a<1" and b0: "0<b" and b1: "b<1"
   shows "a+b - a*b > 0"
lemma

    fixes a :: real and b :: real
    assumes a0: "0<a" and a1: "a<1"
        and b0: "0<b" and b1: "b<1"
    shows "a+b - a*b > 0"
lemma

fixes a :: real and b :: real
assumes a0: "0<a" and a1: "a<1"
and b0: "0<b" and b1: "b<1"
shows "a+b - a*b > 0"

proof -
  have "∀x2 x1. (x2::real) + (x1 - x2) = x1"
    by (metis comm_semiring_1_class.normalizing_semiring_rules(24) diff_add_cancel)
  hence f1: "∀x1 x2 x3. (x1::real) < x2 - x3 ∨ ¬ x3 + x1 < x2"
    by (metis add_less_cancel_left)
  have f2: "∀x1 x2. (x1::real) * x2 < x2 ∨ ¬ 0 < x2 ∨ ¬ x1 < 1"
    by (metis comm_monoid_mult_class.mult.left_neutral real_mult_less_iff1)
  have "0 < b ∧ a < 1"
    by (metis a1 b0)
  hence "a * b < b"
    using f2 by metis
  hence "0 < a ∧ a * b < b"
    by (metis a0)
  hence "a * b < a + b"
    by (metis pos_add_strict)
  hence "a * b + 0 < a + b"
    by (metis comm_monoid_add_class.add.right_neutral)
  thus "0 < a + b - a * b"
    using f1 by metis
qed
lemma
  fixes a :: real and b :: real
assumes a0: "0 < a" and a1: "a < 1"
  and b0: "0 < b" and b1: "b < 1"
shows "a + b - a * b > 0"

Sledgehammer
  • Isar Proof
  • Compression
lemma

fixes a :: real and b :: real

assumes a0: "0<a" and a1: "a<1"
and b0: "0<b" and b1: "b<1"

shows "a+b - a*b > 0"
lemma
  fixes a :: real and b :: real
  assumes a0: "0<a" and a1: "a<1"
    and b0: "0<b" and b1: "b<1"
  shows "a+b - a*b > 0"

proof -
  have "a * b < b"
    by (metis a1 b0 mult_strict_right_mono
        comm_semiring_1_class.normalizing_semiring_rules(11))
  hence "a * b < a + b"
    by (metis a0 pos_add_strict)
  thus "0 < a + b - a * b"
    by (metis add_less_imp_less_right diff_add_cancel
        comm_semiring_1_class.normalizing_semiring_rules(5))
qed
lemma

fixes a :: real and b :: real
assumes a0: "0<a" and a1: "a<1"
    and b0: "0<b" and b1: "b<1"
sshows "a+b - a*b > 0"

Sledgehammer

• Isar Proof
• Compression
• Try0 & Fact Minimization
lemma

fixes a :: real and b :: real
assumes a0: "0<a" and a1: "a<1"  
and b0: "0<b" and b1: "b<1"
shows "a+b - a*b > 0"
lemma

    fixes a :: real and b :: real
    assumes a0: "0<a" and a1: "a<1"
            and b0: "0<b" and b1: "b<1"
    shows "a+b - a*b > 0"

proof -

    have "a * b < b"
        using a1 b0 by simp
    hence "a * b < a + b"
        using a0 pos_add_strict by simp
    thus "0 < a + b - a * b"
        by simp

qed
lemma

    fixes \( a \) :: real and \( b \) :: real

    assumes \( a_0: 0 < a \) and \( a_1: a < 1 \)
    and \( b_0: 0 < b \) and \( b_1: b < 1 \)
    shows "\( a + b - a \cdot b > 0 \)"

proof -

    have "\( a \cdot b < b \)"
        using \( a_1 \) \( b_0 \) by simp

    hence "\( a \cdot b < a + b \)"
        using \( a_0 \) pos_add_strict by simp

    thus "\( 0 < a + b - a \cdot b \)"
        by simp

qed

sledgehammer[isar_proofs, isar_compress=2]
Robust, Semi-Intelligible Isabelle Proofs from ATP Proofs

Steffen Smolka
Advisor: Jasmin Blanchette