Robust, Semi-Intelligible Isabelle Proofs from ATP Proofs

Steffen Juilf Smolka
Jasmin Christian Blanchette
ITPs

well suited for large formalizations
but require intensive manual labor

ATPs

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

ATPs are fully automatic but no proof management.
lemma "length (tl xs) \leq \text{length } xs"

proof (prove): step 0

goal (1 subgoal):
1. length (tl xs) \leq \text{length } xs
lemma "length (tl xs) ≤ length xs"

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

sledgehammer

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_self 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

\textbf{lemma} "length (tl xs) ≤ length xs"
\textbf{by} (metis diff_le_le_self length_tl)

-proof method-

-lemmas-
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
Approach A: Metis One-Liners

Lemma "length (tl xs) ≤ length xs"
by (metis diff_le_self length_tl)

Proof method: usually fast and reliable

External ATPs: find proof given 100s of facts

Metis: re-find proof given only necessary facts

Lemmas:
+ lightweight
- cryptic
- sometimes slow (several seconds)
- on avg. 5% > 30 seconds
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 & Compression - test and optimize proofs
Challenge 2: Skolemization
\[ \forall X. \exists Y. \ p(X, Y) \] \\
\[ \Downarrow \] \\
\[ \forall X. \ p(X, y(X)) \] \\

Skolemization

Signature is extended
\[ \forall X. \exists Y. p(X, Y) \]

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

Skolemization

Signature is extended

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

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

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

Skolemization

Ax. of Choice

Signature is extended

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

$$\forall x. \exists y. p(x, y)$$

$$\exists y. \forall x. p(x, y(x))$$  Ax. of Choice

<steps with extended sig.>
\[ \forall X. \exists Y. p(X, Y) \]

<steps with **extended** sig.>

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

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

Ax. of Choice

<steps with **reduced** sig.>
<steps with extended sig.>

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

$$\frac{\exists X. \forall Y. \neg p(X, Y)}{\exists X. \forall Y. \neg p(X, Y)}$$  Ax. of Choice

<steps with reduced sig.>
<steps with **extended** sig.>

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

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

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

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

\[ \text{print} \]

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

\[ 2 + 2 = 4 \]

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

where \( \alpha : \text{numeral} \quad \text{Un-provable} \)
\begin{align*}
2 \text{nat} + \text{nat} \to \text{nat} & \Rightarrow 2 \text{nat} = \text{nat} \to \text{nat} \to \text{bool} \quad 4 \text{nat} \\
\text{print} & \\
(2: \text{nat}) \ (+: \text{nat} \to \text{nat} \to \text{nat}) \ (2: \text{nat}) \\
& \Rightarrow (=: \text{nat} \to \text{nat} \to \text{bool}) \ (4: \text{nat}) \\
\text{parse} & \\
2 \text{nat} + \text{nat} \to \text{nat} & \Rightarrow 2 \text{nat} = \text{nat} \to \text{nat} \to \text{bool} \quad 4 \text{nat}
\end{align*}
$$2^{\text{nat}} + \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \quad 2^{\text{nat}} = \text{nat} \rightarrow \text{nat} \rightarrow \text{bool} \quad 4^{\text{nat}}$$

$2 + 2 = 4$

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

where $\alpha : \text{numeral}$
\(2 \text{nat} + \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \quad 2 \text{nat} = \text{nat} \rightarrow \text{nat} \rightarrow \text{bool} \quad 4 \text{nat}\)

\[\text{print}\]

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

\[\text{parse}\]

\(2 \text{nat} + \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \quad 2 \text{nat} = \text{nat} \rightarrow \text{nat} \rightarrow \text{bool} \quad 4 \text{nat}\)
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
\[ f : \text{nat} \to \text{int} \to \text{bool} \quad x : \text{nat} \quad y : \text{int} \]

\text{type erasure} \quad \approx \quad \text{printing}

\[ f^- \quad x^- \quad y^- \]

\text{type inference} \quad \approx \quad \text{parsing}

\[ f^{\alpha \to \beta \to \gamma} \quad x^{\alpha} \quad y^{\beta} \]

\text{matching}

\[ \sigma = \{ \alpha \mapsto \text{nat}, \beta \mapsto \text{int}, \gamma \mapsto \text{bool} \} \]
\[ f : \text{nat} \rightarrow \text{int} \rightarrow \text{bool} \quad x : \text{nat} \quad y : \text{int} \]

Type inference:
- \( f \sim \text{parsing} \)

Type erasure:
- \( \approx \text{printing} \)

\[ f^- : x^- \quad y^- \]

Matching:
- \( f^{\alpha \rightarrow \beta \rightarrow \gamma} : x^{\alpha} \quad y^{\beta} \)

\[ \sigma = \{ \alpha \mapsto \text{nat}, \beta \mapsto \text{int}, \gamma \mapsto \text{bool} \} \]

Set of annotations complete \( \iff \) it covers \( \text{Dom}(\sigma) \)
\[ f : \text{nat} \to \text{int} \to \text{bool} \]

\[ \alpha \to \beta \to \gamma \]

\[ x : \beta \to \gamma \]

\[ y : \text{int} \]

\[ \text{bool} \]

\[ f \ x \ y \]
\[ f : \text{nat} \to \text{int} \to \text{bool} \]

\[ \alpha \to \beta \to \gamma \]
\[(f : \text{nat} \rightarrow \text{int} \rightarrow \text{bool}) \, x \, y\]
\((f : \text{nat} \to \text{int} \to \text{bool}) \ x \ y\)
\[(f : \text{nat} \to \text{int} \to \text{bool}) \, x \, y\]

\[f (x : \text{nat}) \, (y : \text{int}) : \text{bool}\]
$(f : \text{nat} \to \text{int} \to \text{bool}) \ x \ y$

$f (x : \text{nat}) \ (y : \text{int}) : \text{bool}$
\[ (f : \text{nat} \rightarrow \text{int} \rightarrow \text{bool}) \ x \ y \]

\[ f (x : \text{nat}) (y : \text{int}) : \text{bool} \]

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

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

\textbf{cost of } t^\tau :=
Which set of annotations is the best?

cost of $t^\tau$ :=

(size of $\tau$, $\rightarrow$ small annotations)
Which set of annotations is the best?

cost of $t^\tau :=$

(size of $\tau$, size of $t$, → small annotations
→ small annotated terms)
Which set of annotations is the best?

\[
\text{cost of } t^\tau := \\
\text{(size of } \tau, \text{ size of } t, \text{ postindex of } t^\tau) \rightarrow \text{small annotations at the beginning}
\]
Which set of annotations is the best?

\[
\text{cost of } t^\tau :=
\begin{align*}
\text{(size of } \tau, & \rightarrow \text{small annotations} \\
\text{size of } t, & \rightarrow \text{small annotated terms} \\
\text{postindex of } t^\tau & \rightarrow \text{annotations at the beginning}
\end{align*}
\leq \text{lexiographically}
+ \text{component-wise}
\]
How do we compute it efficiently?
How do we compute it efficiently?

Instance of **Weighted Set Cover Problem:**

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

Instance of **Weighted Set Cover Problem**:

- Finite Universe $U$ $\rightarrow \text{Dom}(\sigma)$
- Family $S \subseteq 2^U$ $\rightarrow \text{Possible Annotations}$
- Find $\{U_1,\ldots,U_n\} \subseteq S$ such that
  - $U_1 \cup \ldots \cup U_n = U$ $\rightarrow \text{Completeness}$
  - cost $\{U_1,\ldots,U_n\}$ minimal $\rightarrow \text{Readability}$
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 & Compression
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 ∪ -x = -x ∪ -(x)"

Sledgehammer

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

Timeout
lemma "x U -x = -x U -(x)"
sledgehammer

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

Structured proof (43 steps, 1.34 s):
proof -
  have f1: "∀x₁ x₂. -(-x₁ U x₂) U -(-x₁ U -x₂) = x₁"
    by (metis huntington sup_comm)
  have f2: "∀x₁ x₂ x₃. x₁ U (x₂ U x₃) = x₃ U (x₁ U x₂)"
    by (metis sup_assoc sup_comm)
  have f3: "∀x₁ x₂ x₃. x₁ U (x₂ U x₃) = x₂ U x₁ U x₃"
    by (metis sup_assoc sup_comm)
  have f4: "∀x₁ x₂ x₃. x₁ U (x₂ U x₃) = x₃ U (x₂ U x₁)"
    by (metis sup_assoc sup_comm)
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

lemma "\( x \sqcup -x = -x \sqcup -(-x) \)"

sledgehammer (huntington sup_assoc sup_comm)

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

Structured proof: (43 steps, 1.34 s):

proof -

have f1: "\( \forall x_1 \ x_2. \ -(-x_1 \sqcup x_2) \sqcup -(-x_1 \sqcup -x_2) = x_1 \)"

by (metis huntington_sup_comm)

have f2: "\( \forall x_1 \ x_2 \ x_3. \ x_1 \sqcup (x_2 \sqcup x_3) = x_3 \sqcup (x_1 \sqcup x_2) \)"
Proof Compression

```
lemma "\( \exists \ x . -x = -x \uplus -(-x) \)"

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: "\( \forall x_1 \ x_2 . \ (-x_1 \uplus x_2) \uplus (-x_1 \uplus -x_2) = x_1 \)"
    by (metis huntington sup_comm)
  have f2: "\( \forall x_1 \ x_2 \ x_3 . \ x_1 \uplus (x_2 \uplus x_3) = x_3 \uplus (x_1 \uplus x_2) \)"
```
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 $\vdash$ F1
A1, F1 $\vdash$ F2
A2, F1 $\vdash$ F3
F1, F2, F3 $\vdash$ C
A1 ⊢ F1
A1, F1 ⊢ F2
A2, F1 ⊢ F3
F1, F2, F3 ⊢ C

A1 ⊢ F1
A1, F1 ⊢ F2
A2, F1 ⊢ F3
F1, F2, A2 ⊢ 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
\[
\begin{align*}
A1 & \vdash F1 \\
A1, F1 & \vdash F2 \\
A2, F1 & \vdash F3 \\
F1, F2, F3 & \vdash C \\
\end{align*}
\]
A1 ⊢ F1
A1,F1 ⊢ F2
A2,F1 ⊢ F3
F1,F2,F3 ⊢ C

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

F1,F2,A2 ⊢ C

F1,A1,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
F1, A1, A2 ⊢ C
Does merger save time? → Preplay

Have we reached a given compression factor?
Robust, Semi-Intelligible Isabelle Proofs from ATP Proofs

Steffen Juilf Smolka
Jasmin Christian Blanchette