Eisbach: An Isabelle Proof Method Language

Daniel Matichuk

Makarius Wenzel, Toby Murray

ITP 2014
Size distribution of AFP entries in lines of proof, sorted by submission date.
Size distribution of AFP entries in lines of proof, sorted by submission date
Proof Engineering

Size distribution of AFP entries in lines of proof, sorted by submission date
Size distribution of AFP entries in lines of proof, sorted by submission date
Outline

Isabelle Concepts
- Isar
- Proof Methods

Eisbach
- Easy Custom Proof Methods
- Demo

Evaluation/Future
- Existing method rewritten
- Tracing/Debugging…
Isabelle Concepts

- Isar
- Proof Methods

Eisbach
- Easy Custom Proof Methods
- Demo

Evaluation/Future
- Existing method rewritten
- Tracing/Debugging…
Isabelle/Isar

**theorem** Knaster-Tarski:
- **assumes** mono: \( \land x \ y. \ x \leq y \implies f \ x \leq f \ y \)
- **shows** \( f (\bigcap \{x. \ f \ x \leq x\}) = \bigcap \{x. \ f \ x \leq x\} \) (is \( f \ ?a = \ ?a \))

**proof**
- **have** \(*:: f \ ?a \leq \ ?a\) (is - \leq \bigcap \ ?H)

**proof**
- **fix** \( x \) **assume** \( H: \ x \in \ ?H \)
- **then have** \( \ ?a \leq \ ?a \) ..
- **also from** \( H\) **have** \( f \ldots \leq \ ?a \) ..
- **moreover note** mono **finally show** \( f \ ?a \leq \ ?a \)

**qed**

**also have** \( \ ?a \leq f \ ?a \)

**proof**
- **from** mono **and** \( *\) **have** \( f (f \ ?a) \leq f \ ?a \)
- **then show** \( f \ ?a \in \ ?H \) ..

**qed**

**finally show** \( f \ ?a = \ ?a \)

**qed**
Isabelle/Isar

**theorem** Knaster-Tarski:

- **assumes** mono: \( \forall x \ y. \ x \leq y \implies f \ x \leq f \ y \)
- **shows** \( f \ (\bigcap \{x. f \ x \leq x\}) = \bigcap \{x. f \ x \leq x\} \) (is \( ?a = ?a \))

**proof** —

- **have** \( *: f \ ?a \leq ?a \) (is \( - \leq \bigcap ?H \))

**proof**

- **fix** \( x \) **assume** \( H: \ x \in ?H \)
- **then** **have** \( ?a \leq x \) ..
- **also from** \( H \) **have** \( f \ldots \leq x \) ..
- **moreover note** mono **finally show** \( f \ ?a \leq x \).

**qed**

**also have** \( ?a \leq f \ ?a \)

**proof**

- **from** mono **and** \( * \) **have** \( f \ (f \ ?a) \leq f \ ?a \).
- **then show** \( f \ ?a \in ?H \) ..

**qed**

**finally show** \( f \ ?a = ?a \).

**qed**
theorem Knaster-Tarski':

  assumes mono[intro!]: \( \forall x \ y. \, x \leq y \implies f \, x \leq f \, y \)
  shows \( f \left( \bigcap \{ x. \, f \, x \leq x\} \right) = \bigcap \{ x. \, f \, x \leq x\} \) (is f ?a = ?a)

proof –
  have *: f ?a \leq ?a by (clarsimp, rule order.trans, fastforce)
  also have ?a \leq f ?a by (fastforce intro!: *)
  finally show f ?a = ?a.
qed
theorem \textit{Knaster-Tarski}'

assumes \textit{mono}[intro!]: $\forall x\ y. \ x \leq y \implies f\ x \leq f\ y$

shows $f\ (\bigcap\ \{\ x. \ f\ x \leq x\}\ ) = \bigcap\ \{\ x. \ f\ x \leq x\\}$ (is $f\ ?a = ?a$)

proof –

have *: $f\ ?a \leq ?a$ by (clarsimp,rule order.trans, fastforce)
also have $?a \leq f\ ?a$ by (fastforce intro!: *)
finally show $f\ ?a = ?a$.

qed
Proof Methods

have *: f ?a ≤ ?a by (clarsimp, rule order.trans, fastforce)

Goal    Method    Combinator

also have ?a ≤ f ?a by (fastforce intro!: *)

Method Parameter
theorem Knaster-Tarski': \((\forall x \ y. \ x \leq y \Longrightarrow f \ x \leq f \ y) \Longrightarrow f \ (\bigcap \ \{x. \ f \ x \leq x\}) = \bigcap (\{x. \ f \ x \leq x\})\)
apply (tactic \((\text{EqSubst.eqsubst-tac @\{context\} [0] @\{thms order-eq-iff\} 1})\)
THEN (Tactic.resolve-tac @\{thms context-conjI\} 1)
THEN (Tactic.resolve-tac @\{thms Inf-greatest\} 1)
THEN (Tactic.forward-tac @\{thms Inf-lower\} 1)
THEN (Clasimp.fast-force-tac @\{context\} 1)
THEN (Tactic.resolve-tac @\{thms Inf-lower\} 1)
THEN (Clasimp.fast-force-tac @\{context\} 1)
\))
done
Isabelle’s AFP

Number of files in AFP

- **ML**: 50
- **Isar (.thy)**: 1,663
seL4 - our experience

• Full functional correctness proof
  – Source code and Proof going open source!
  – [http://seL4.systems](http://seL4.systems) for more info
  – July 29

• Isabelle proof methods developed
  – WP/WPC - vcg for monadic hoare logic
  – sep-* - automating separation logic

• Proof Engineers want more!
  – Languages like Ltac show this
Eisbach

Isabelle Concepts
- Isar
- Proof Methods

Eisbach
- Easy Custom Proof Methods
- Demo

Evaluation/Future
- Existing method rewritten
- Tracing/Debugging…
Language Elements

• Integrates existing/new methods
  – fastforce, simp, auto…

• Abstract over Terms/Facts/Methods

• Attributes for method hints
  – simp, intro, my_vcg_rules…

• Matching provides control flow
  – Match and bind higher-order patterns against focused subgoal elements
Eisbach

method-definition induct-list facts simp =
  (match ?concl in ?P (?x :: 'a list) ⇒ (induct ?x ↦ fastforce simp: simp))

lemma length (xs @ ys) = length xs + length ys by induct-list
Eisbach - Design goals

• Easy for beginners and experts
  – Familiar method syntax from Isar

• Limited functionality - leave complexity to Isabelle/ML

• Integration with other Isabelle languages

• Readable proof procedures
Eisbach - Combinators

• **Standard Isar Method Combinators**
  - “|” - alternative composition
  - “,” - sequential composition
  - “?” - suppress failure (try)
  - “+” - repeated application

• **New Combinator**
  - “⇒” - compose with emerging subgoals

```
method-definition prop-solver₁ = ((rule impI, (erule conjE)?) | assumption)+
```
Eisbach - Abstraction

- Parameterize over facts, terms, and methods

Method “Signature”

```
method-definition prop-solver facts intro elim =
  ((rule intro, (erule elim)?) | assumption)+
```

Abstracted Facts
Eisbach - Abstraction

- Parameterize over facts, terms, and methods

**Method “Signature”**

```plaintext
method-definition prop-solver₂ facts intro elim =
((rule intro, (erule elim)?) | assumption) +
```

**Abstracted Facts**

**Fact Arguments**

```plaintext
lemma P ∧ Q → P by (prop-solver₂ intro: impI elim: conjE)
```
Eisbach - Attributes

• New command: **declare-attributes**

```
declare-attributes intro elim
```

- Managed with the usual Isar `declare` command

```
declar impI [intro] and conjE [elim]
```

- Used at run-time by methods

```
method-definition prop-solver3 facts [intro] [elim] =
((rule intro, (erule elim)?) | assumption)+
```
Eisbach - Attributes

- New command: declare-attributes

\[ \text{declare-attributes intro elim} \]

- Managed with the usual Isar declare command

\[ \text{declare impI [intro] and conjE [elim]} \]

- Used at run-time by methods

\[ \text{method-definition prop-solver\_3 facts [intro] [elim] =} \]
\[ (\text{((rule intro, (erule elim)?)} | \text{assumption})+) \]

Square brackets indicate fact parameter is managed by attribute
• New command: **declare-attributes**

```
declare-attributes intro elim
```

– Managed with the usual Isar `declare` command

```
declare impI [intro] and conjE [elim]
```

– Used at run-time by methods

```
method-definition prop-solver3 facts [intro] [elim] = ((rule intro, (erule elim)?) | assumption)+
```

Square brackets indicate fact parameter is managed by attribute

Contains `impI`

Contains `conjE`
Eisbach - Attributes

• New command: `declare-attributes`

```
declare-attributes intro elim
```

– Managed with the usual Isar `declare` command

```
declare impI [intro] and conjE [elim]
```

– Used at run-time by methods

```
method-definition prop-solver3 facts [intro] [elim] =
((rule intro, (erule elim)?) | assumption)+
```

Contains `impI`
Contains `conjE`

```
lemma P ∧ Q → P by prop-solver3
```
Eisbach - Matching

- Higher-order matching for control flow
  - Bind matched patterns

```
method-definition solve-ex =
  (match ?concl in \exists x. ?Q x \Rightarrow
    (match prems in U: Q ?y \Rightarrow (rule exI [where x = y and P = Q, OF U])))
```
Eisbach - Matching

• Higher-order matching for control flow
  – Bind matched patterns

Special term ?concl is current subgoal

method-definition solve-ex =
(match ?concl in \exists x. ?Q x \implies
  (match prems in U: ?Q ?y \implies (rule exI [where x = y and P = Q, OF U])))
Eisbach - Matching

• Higher-order matching for control flow
  – Bind matched patterns

**Special term** $\text{?concl}$

is current subgoal

**Matched pattern** $\text{?Q}$

is bound

```
method-definition solve-ex =
  (match ?concl in \( \exists x. \ ?Q \ x \Rightarrow \))
  (match prems in U: Q ?y \Rightarrow (rule exI [where x = y and P = Q, OF U])))
```
Eisbach - Matching

• Higher-order matching for control flow
  – Bind matched patterns

\[
\text{method-definition } \text{solve-ex} = \\
(\text{match } ?\text{concl} \text{ in } \exists x. \ ?Q \ x \Rightarrow \\
(\text{match } \text{prems in } U : Q \ ?y \Rightarrow (\text{rule exI [where } x = y \ \text{and } P = Q, \ OF U])))
\]

Special term \(?\text{concl}\) is current subgoal

Matched pattern \(?Q\) is bound

Special fact \(?\text{prems}\) is current premises
Eisbach - Matching

• Higher-order matching for control flow
  – Bind matched patterns

Method-definition `solve-ex` =

\[
\text{match } ?\text{concl} \text{ in } \exists x. \ ?Q \ x \Rightarrow
\text{match } \text{prems in } U: \ Q \ ?y \Rightarrow (\text{rule exI [where } x = y \ \text{and } P = Q, \ OF \ U])
\]

Special term `?concl` is current subgoal

Matched pattern `?Q` is bound

Special fact `prems` is current premises

Matching singleton fact `U` is bound
Focus/Matching

• Problem: Raw subgoals are unstructured

$$\bigwedge x. \ A \ x \implies B \ x \implies A \ x \land B \ x$$
Focus/Matching

- Problem: Raw subgoals are unstructured

\[ \forall x. \ A \ x \implies B \ x \implies A \ x \land B \ x \]

by \( \text{rule conjI}[\text{OF assms}(1) \ \text{assms}(2)] \)
Focus/Matching

• Problem: Raw subgoals are unstructured

\[ \forall x. \ A \ x \implies B \ x \implies A \ x \land B \ x \]

by \((\text{rule conj}[\text{OF assms(1)} \ \text{assms(2)}])]\)
Focus/Matching

• Problem: Raw subgoals are unstructured

\[ \forall x. A\ x \Rightarrow B\ x \Rightarrow A\ x \land B\ x \]

by \( \text{rule conjI[OF assms(1) assms(2)]} \)

• Goal:

```plaintext
method-definition solve-conj =
  (match ?concl in ?P \land ?Q \Rightarrow
    (match prems in U: P and U': Q \Rightarrow
      (rule conjI[OF U U'])))
```
Focus/Matching

• Problem: Raw subgoals are unstructured

\[ \forall x. A \ x \implies B \ x \implies A \ x \land B \ x \]

by \((\text{rule conjI[\text{OF assms(1)} \ \text{assms(2)}]})\)

• Goal:

\text{method-definition} \ solve-conj =

\begin{align*}
\text{(match } ?\text{concl in } ?P \land ?Q & \Rightarrow \text{(match } \text{prems in } U: P \ \text{and } U': Q & \Rightarrow \\
\text{(rule conjI[OF } U U'])}))\end{align*}

\text{Find and name assumptions through matching}
Focus

• **Solution: Focusing**
  – Based on existing work

\[ \land x. \ A x \implies B x \implies A x \land B x \]
Focus

- **Solution: Focusing**
  - Based on existing work

\[ \land x. \ A \ x \implies B \ x \implies A \ x \land B \ x \]

- **fixes** \( x \)
- **assumes** \( A \ x \) **and** \( B \ x \)
- **shows** \( A \ x \land B \ x \)
Focus

- **Solution: Focusing**
  - Based on existing work

\[ \forall x. A\ x \iff B\ x \iff A\ x \land B\ x \]

- fixes \( x \)
- assumes \( A\ x \) and \( B\ x \)
- shows \( A\ x \land B\ x \)
Demo
Evaluation/Future work

- Isabelle Concepts
  - Isar
  - Proof Methods

- Eisbach
  - Easy Custom Proof Methods
  - Demo

- Evaluation/Future
  - Existing method rewritten
  - Tracing/Debugging…
Tactic Languages are not new

- **Ltac**
  - Untyped High-level tactic language for Coq
  - Goal matching, iteration, recursion

- **VeriML**
  - Dependently typed tactic language
  - Provides strong static guarantees

- **Mtac**
  - Typed tactic language for Coq
  - Leverages built-in Coq notion of computation
  - Strong static guarantees
Current Results

• **Eisbach**
  – Extension of Isar, Isabelle’s proof language
  – Integrates with existing Isar syntax
    • methods
    • attributes

• **Evaluation**
  – Existing methods rewritten in Eisbach
    • WP, WPC: l4.verified invariant proof successfully checked

• **Future Work**
  – Tracing/Debugging
  – Optimisations
Conclusion

- Proof Engineers need tools
  - to write proofs at scale
- Isar provides structure/syntax for proofs
  - Most Isabelle users most familiar with Isar
- Eisbach provides easy mechanisms for writing automation
  - abstraction
  - matching
  - backtracking
  - recursion
- Coming soon…
Thank You!