Automated Reasoning

Coursework lecture:
Proving and Reasoning in Isabelle/HOL

Imogen I. Morris

20/10/2017
Coursework overview

• Part 1: Propositional and first-order proofs [40%]
• Part 2: Geometry with order and signed areas [60%]
Part 1: Propositional and first-order proofs

- Procedural proofs (sequence of rule applications).
Part 1: Propositional and first-order proofs

- Procedural proofs (sequence of rule applications).
- You are given a list of rules you may use.
Part 1: Propositional and first-order proofs

- Procedural proofs (sequence of rule applications).
- You are given a list of rules you may use.
- View them using \texttt{thm rule}.
Knights and Knaves problems

- You meet two inhabitants: Sue and Zippy. Sue says that Zippy is a knave. Zippy says, ‘I and Sue are knights.’
- Can you determine who is a knight and who is a knave?
Knights and Knaves problems

- You meet two inhabitants: Sue and Zippy. Sue says that Zippy is a knave. Zippy says, ‘I and Sue are knights.’
- Can you determine who is a knight and who is a knave?
- The most natural way to solve this problem is to reason by cases.
Knights and Knaves problems

• You meet two inhabitants: Sue and Zippy. Sue says that Zippy is a knave. Zippy says, ‘I and Sue are knights.’
• Can you determine who is a knight and who is a knave?
• The most natural way to solve this problem is to reason by cases.
• In Isabelle we can use case_tac. E.g. (case_tac "V x"). We then have two subgoals: V x \implies goal and \neg V x \implies goal.
Knights and Knaves problems

- We formalise ‘a is a knave’ as $V a$ and ‘a is a knight’ as $G a$. 
Knights and Knaves problems

- We formalise ‘a is a knave’ as $V \alpha$ and ‘a is a knight’ as $G \alpha$.
- ‘Person $a$ says statement $P$’ is formalised as $S_n \alpha = P$, where $n$ is some natural number.
Knights and Knaves problems

• We formalise ‘a is a knave’ as $V a$ and ‘a is a knight’ as $G a$.

• ‘Person a says statement $P$’ is formalised as $S n a = P$, where $n$ is some natural number.

• We index the statement by $n$ because one person may make more than one statement.
Knights and Knaves problems

• We formalise ‘a is a knave’ as $V \ a$ and ‘a is a knight’ as $G \ a$.

• ‘Person a says statement $P$’ is formalised as $S \ n \ a = P$, where $n$ is some natural number.

• We index the statement by $n$ because one person may make more than one statement.

• We are also assuming that the domain of the quantifiers is all the inhabitants of the island (so you, as a visitor to the island, are not included).
Knights and Knaves problems

- We can formalise the previous problem:
  - $S_1 s = V z$ and $S_1 z = K s \land K z$.

- Solution: Zippy cannot be a knight, because if what he said was true, then Sue would be telling a lie and then she is not a knight - contradiction. Hence Zippy is a knave, and as Sue is telling the truth, she is a knight.

- Suppose we get the formalisation wrong:
  - $S_1 s = V z$ and $S_1 z = K s \land K z$.
  
  - The previous analysis holds, thus Zippy is a knave, yet he makes a true statement (that Sue is a knight), so Zippy is a knight. Hence the problem is unsolvable.
Knights and Knaves problems

• We can formalise the previous problem:

• \[ S_1 s = V z \text{ and } S_1 z = K s \land K z. \]

• Solution: Zippy cannot be a knight, because if what he said was true, then Sue would be telling a lie and then she is not a knight - contradiction. Hence Zippy is a knave, and as Sue is telling the truth, she is a knight.

• Suppose we get the formalisation wrong:

• \[ S_1 s = V z \text{ and } S_1 z = K s \land K z. \]

• The previous analysis holds, thus Zippy is a knave, yet he makes a true statement (that Sue is a knight), so Zippy is a knight. Hence the problem is unsolvable.
Knights and Knaves problems

• We can formalise the previous problem:
• $S_1 s = V z$ and $S_1 z = K s \land K z$.

• Solution: Zippy cannot be a knight, because if what he said was true, then Sue would be telling a lie and then she is not a knight - contradiction. Hence Zippy is a knave, and as Sue is telling the truth, she is a knight.

• Suppose we get the formalisation wrong:
• $S_1 s = V z$ and $S_1 z = K s$ and $S_2 z = K z$. 
Knights and Knaves problems

- We can formalise the previous problem:
  - $S_1 s = V z$ and $S_1 z = K s \land K z$.
- Solution: Zippy cannot be a knight, because if what he said was true, then Sue would be telling a lie and then she is not a knight - contradiction. Hence Zippy is a knave, and as Sue is telling the truth, she is a knight.
- Suppose we get the formalisation wrong:
  - $S_1 s = V z$ and $S_1 z = K s$ and $S_2 z = K z$.
  - The previous analysis holds, thus Zippy is a knave, yet he makes a true statement (that Sue is a knight), so Zippy is a knight. Hence the problem is unsolvable.
Part 2: Structured proofs & powerful reasoning tools

- Isabelle has a lot of machinery built in for presentation, interaction and automation.
Part 2: Structured proofs & powerful reasoning tools

• Isabelle has a lot of machinery built in for presentation, interaction and automation.

• Structured proofs (also called *declarative*); the name of the language is *Isar*. 
Part 2: Structured proofs & powerful reasoning tools

• Isabelle has a lot of machinery built in for presentation, interaction and automation.

• Structured proofs (also called *declarative*); the name of the language is *Isar*.

• Powerful automatic tools: simp, auto, safe, blast, fast, force, fastforce, linarith, arith, presburger, algebra, meson,metis.
Part 2: Structured proofs & powerful reasoning tools

- Isabelle has a lot of machinery built in for presentation, interaction and automation.
- Structured proofs (also called *declarative*); the name of the language is *Isar*.
- Powerful automatic tools: simp, auto, safe, blast, fast, force, fastforce, linarith, arith, presburger, algebra, meson, mesit.
- A link to external provers: sledgehammer.
Reasoning with equality (=)

Rules:

\[
\begin{align*}
\frac{t = s}{Pt} & \quad \frac{Ps}{P s} \quad \text{subst} \quad \frac{s = t}{Pt} & \quad \frac{Ps}{ssubst} \\
\hline
\frac{t = t}{\text{refl}} & \quad \frac{s = t}{\text{sym}} & \quad \frac{r = s \quad s = t}{\text{trans}}
\end{align*}
\]

\[
\frac{\forall x. f x = g x}{f = g} \quad \text{ext}
\]
Reasoning with equality (=)

Rules:

\[
\frac{t = s}{P t} \quad \frac{P s}{\text{subst}}
\]

\[
\frac{s = t}{P t} \quad \frac{P s}{\text{ssubst}}
\]

\[
\frac{t = t}{\text{refl}}
\]

\[
\frac{s = t}{t = s} \quad \frac{t = s}{\text{sym}}
\]

\[
\frac{r = s}{s = t} \quad \frac{s = t}{\text{trans}}
\]

\[
\frac{\forall x. f x = g x}{f = g} \quad \frac{\text{ext}}
\]

Are all of these rules necessary, or can some of them be derived from the others?
Reasoning with equality (=)

Output:
1. ∀c. ∃a b. a + 3 * b = c

lemma “∀c :: int. ∃a b. a + 3 * b = c”

\[
\begin{align*}
\text{mult\_zero\_right:} & \quad a \times 0 = 0 \\
\text{add\_0:} & \quad 0 + a = a \\
\text{add\_commute:} & \quad a + b = b + a
\end{align*}
\]
Reasoning with equality (=)

Output:
1. \( \forall c. \exists a \ b. \, a + 3 \times b = c \)

**Lemma** \( \forall c :: \text{int}. \exists a \ b. \, a + 3 \times b = c \)

apply (rule allI)
Reasoning with equality (=)

Output:
1. $\forall c. \exists b. c + 3 \times b = c$

lemma “$\forall c :: \text{int}. \exists a b. a + 3 \times b = c$”
apply (rule allI)
apply (rule_tac x = c in exI)

Reasoning with equality (=)

Output:
1. $\forall c. c + 3 \times 0 = c$

lemma "\(\forall c :: \text{int}. \exists a b. a + 3 \times b = c\)"
apply (rule allI)
apply (rule_tac x = c in exI)
apply (rule_tac x = 0 in exI)
apply (rule_tac s = 0 and t = "3 \times 0" in ssubst)
apply (rule mult_zero_right)
apply (rule_tac s = "0 + c" and t = "c + 0" in ssubst)
apply (rule add.commute)
apply (rule add_0)
done
Reasoning with equality (=)

Output:
1. $\land c. 3 \times 0 = 0$
2. $\land c. c + 0 = c$

\begin{align*}
\text{mult\_zero\_right:} & \quad a \times 0 = 0 \\
\text{add\_0:} & \quad 0 + a = a \\
\text{add\_commute:} & \quad a + b = b + a
\end{align*}

\textbf{lemma} “\begin{align*}
\forall c :: \text{int}. \exists a \ b. a + 3 \times b = c
\end{align*}”

apply (rule allI)
apply (rule_tac x = c in exI)
apply (rule_tac x = 0 in exI)
apply (rule_tac s = 0 and t = “3 \times 0” in subst)
Reasoning with equality (=)

Output:
1. $\forall c. c + 0 = c$

**lemma** “$\forall c :: \text{int}. \exists a b. a + 3 * b = c$”

apply (rule allI)
apply (rule_tac x = c in exI)
apply (rule_tac x = 0 in exI)
apply (rule_tac s = 0 and t = “3 * 0” in ssubst)
apply (rule mult_zero_right)
apply (rule_tac s = “0 + c” and t = “c + 0” in ssubst)
apply (rule add.commute)
apply (rule add_0)
done

**mult_zero_right:** $a * 0 = 0$
**add_0:** $0 + a = a$
**add.commute:** $a + b = b + a$
Reasoning with equality (=)

Output:
1. $\forall c. c + 0 = 0 + c$
2. $\forall c. 0 + c = c$

**lemma** "\(\forall c :: \text{int}. \exists a b. a + 3 \ast b = c\)"
   - apply (rule allI)
   - apply (rule_tac x = c in exI)
   - apply (rule_tac x = 0 in exI)
   - apply (rule_tac s = 0 and t = "3 \ast 0" in ssubst)
   - apply (rule mult_zero_right)
   - apply (rule_tac s = "0 + c" and t = "c + 0" in ssubst)

**Rules:**
- **mult_zero_right:** \(a \ast 0 = 0\)
- **add_0:** \(0 + a = a\)
- **add.commute:** \(a + b = b + a\)
Reasoning with equality (=)

Output:
1. $\forall c. 0 + c = c$

lemma “$\forall c : \text{int}. \exists a b. a + 3 \times b = c$”
apply (rule allI)
apply (rule_tac x = c in exI)
apply (rule_tac x = 0 in exI)
apply (rule_tac s = 0 and t = “3 \times 0” in ssubst)
apply (rule mult_zero_right)
apply (rule_tac s = “0 + c” and t = “c + 0” in ssubst)
apply (rule add.commute)
Reasoning with equality (=)

Output:
No subgoals!

\[
\text{mult\_zero\_right: } a \times 0 = 0 \\
\text{add\_0: } 0 + a = a \\
\text{add\_commute: } a + b = b + a
\]

\textbf{lemma} \textbf{“}∀c :: \textbf{int}. \exists a b. a + 3 \times b = c”

apply (rule allI)
apply (rule_tac x = c in exI)
apply (rule_tac x = 0 in exI)
apply (rule_tac s = 0 and t = “3 \times 0” in ssubst)
apply (rule mult\_zero\_right)
apply (rule_tac s = “0 + c” and t = “c + 0” in ssubst)
apply (rule add\_commute)
apply (rule add\_0)
Reasoning with equality (=)

Output: No subgoals!

```
lemma “∀c :: int. ∃a b. a + 3 * b = c”
  apply (rule allI)
  apply (rule_tac x = c in exI)
  apply (rule_tac x = 0 in exI)
  apply (rule_tac s = 0 and t = “3 * 0” in ssubst)
  apply (rule mult_zero_right)
  apply (rule_tac s = “0 + c” and t = “c + 0” in ssubst)
  apply (rule add.commute)
  apply (rule add_0)
  done
```
Reasoning with equality (=)

Output:
No subgoals!

\[ \text{mult\_zero\_right: } a \times 0 = 0 \]
\[ \text{add\_0: } 0 + a = a \]
\[ \text{add\_commute: } a + b = b + a \]

**Lemma** "\( \forall c : \text{int}. \exists a b. a + 3 \times b = c \)"

```
apply (rule allI)
apply (rule_tac x = c in exI)
apply (rule_tac x = 0 in exI)
apply (subst mult_zero_right)
apply (subst add.commute)
apply (rule add_0)
done
```

We can save all that variable instantiation using **subst**: rewriting.
But we will be using Isar:

Without `subst`:

```isar
lemma "∀c :: int. ∃a b. a + 3 * b = c"
proof
  fix c :: int
  have "c + 3 * 0 = c + 0"
    by (rule_tac s = 0 and t = "3 * 0" in ssubst, rule mult_zero_right, rule refl)
  also have "... = 0 + c"
    by (rule add.commute)
  also have "... = c"
    by (rule add_0)
  finally have "c + 3 * 0 = c"
    by (rule trans, rule_tac refl)
  then have "∃b. c + 3 * b = c"
    by (rule exI)
  then show "∃a b. a + 3 * b = c"
    by (rule exI)
qed
```

With `subst`:

```isar
lemma "∀c :: int. ∃a b. a + 3 * b = c"
proof
  fix c :: int
  have "c + 3 * 0 = c + 0"
    by (subst mult_zero_right, rule refl)
  also have "... = 0 + c"
    by (rule add.commute)
  also have "... = c" by (rule add_0)
  finally have "c + 3 * 0 = c"
    by (rule trans, rule_tac refl)
  then have "∃b. c + 3 * b = c"
    by (rule exI)
  then show "∃a b. a + 3 * b = c"
    by (rule exI)
qed
```
Useful attributes to use with subst

**symmetric**: This swaps the left and right hand sides of the equality in `theorem`.
Usage: `subst theorem [symmetric]`

**asm**: This allows substitution into the assumption rather than the conclusion.
Usage: `subst(asm) theorem`

**n, where n is a natural number**: This allows substitution with the \( n \)th occurrence in the goal of an expression that can be unified with the left-hand side of `theorem`.
Usage: `subst(n) theorem`
Reasoning with equality (=)

Output:
No subgoals!

lemma \(\forall c :: \text{int}. \exists a b. a + 3 \times b = c\)

apply (rule allI)
apply (rule_tac x = c in exI)
apply (rule_tac x = 0 in exI)
apply (simp only: mult_zero_right add.commute add_0)
done

Method simp does substitution automatically (given the right rules!).
Reasoning with equality (=)

Output:
No subgoals!

mult_zero_right: \( a \times 0 = 0 \)

add_0: \( 0 + a = a \)

add.commute: \( a + b = b + a \)

**lemma** "\( \forall c :: int. \exists a b. a + 3 \times b = c \)"

apply (rule allI)
apply (rule_tac x = c in exI)
apply (rule_tac x = 0 in exI)
apply simp
done

Method **simp** does substitution automatically (given the right rules!).

...and the right rules are already in the Main library.
Isabelle’s powerful tools

• **simp**: rewriting using equations.
  
  Uses: ```apply simp```
  ```apply (simp add: eq_1 ... eq_n)```
  ```apply (simp only: eq_1 ... eq_n)```
  ```apply (simp del: eq_1 ... eq_n)```
Isabelle’s powerful tools

- **simp**: rewriting using equations.
  Uses: `apply simp`
  `apply (simp add: eq_1 ... eq_n)`
  `apply (simp only: eq_1 ... eq_n)`
  `apply (simp del: eq_1 ... eq_n)`

- **auto**: rewriting + proof search (using classical logic).
  Uses: `apply auto`
  `apply (auto simp add: eq_1 ... eq_n)`
  `apply (auto simp only: eq_1 ... eq_n)`
  `apply (auto simp del: eq_1 ... eq_n)`
Isabelle’s powerful tools

- **simp**: rewriting using equations.
  
  Uses: `apply simp`
  
  `apply (simp add: eq₁ ... eqₙ)`
  
  `apply (simp only: eq₁ ... eqₙ)`
  
  `apply (simp del: eq₁ ... eqₙ)`

- **auto**: rewriting + proof search (using classical logic).
  
  Uses: `apply auto`
  
  `apply (auto simp add: eq₁ ... eqₙ)`
  
  `apply (auto simp only: eq₁ ... eqₙ)`
  
  `apply (auto simp del: eq₁ ... eqₙ)`

- Others: **blast, fast, force, fastforce, safe, algebra, linarith, arith, presburger, meson, meson, meson, meson**.
Isabelle’s powerful tools

Sledgehammer

\[ \text{lemma } \inj\_\text{on } f \ A = \Rightarrow \exists g. g' f A \subseteq A \land (\forall a \in A. g'(f(a)) = a) \land (\forall b \in A. f(g'(f(b))) = f(b)) \]
Isabelle’s powerful tools

Sledgehammer

• Tool for invoking external provers.
Isabelle’s powerful tools

Sledgehammer

• Tool for invoking external provers.
• Isabelle should not just trust external provers.
Isabelle’s powerful tools

Sledgehammer

- Tool for invoking external provers.
- Isabelle should not just trust external provers.
- Sledgehammer tries to reconstruct proof inside Isabelle.
Isabelle’s powerful tools

Sledgehammer

- Tool for invoking external provers.
- Isabelle should not just trust external provers.
- Sledgehammer tries to reconstruct proof inside Isabelle.
- Usually, \texttt{metis} will do the job, given a list of lemmas suggested by sledgehammer.
Isabelle’s powerful tools

**Sledgehammer**

- Tool for invoking external provers.
- Isabelle should not just trust external provers.
- Sledgehammer tries to reconstruct proof inside Isabelle.
- Usually, `metis` will do the job, given a list of lemmas suggested by sledgehammer.

**Lemma** "`inj_on f A` \(\implies\) \(\exists g. g' f' A \subseteq A \land (\forall a \in A. g(f a) = a) \land (\forall b \in A. f(g(f b)) = f b)\)"

`sledgehammer`
Isabelle’s powerful tools

**Sledgehammer**

- Tool for invoking external provers.
- Isabelle should not just trust external provers.
- Sledgehammer tries to reconstruct proof inside Isabelle.
- Usually, `metis` will do the job, given a list of lemmas suggested by sledgehammer.

**Lemma** “\( \text{inj\_on } f A \implies \exists g. g' f' A \subseteq A \land (\forall a \in A. g(f a) = a) \land (\forall b \in A. f(g(f b)) = f b) \)”

**by** `(metis order_refl the_inv_into_f_f the_inv_into_onto)`
Isabelle's powerful tools

Useful commands:
Isabelle’s powerful tools

Useful commands:

- **try0**: tries a bunch of internal provers (auto, simp, ...).
Isabelle’s powerful tools

Useful commands:

- **try0**: tries a bunch of internal provers (**auto**, **simp**, ...).
- **try**: try0 + sledgehammer + counterexample checkers!
Isabelle’s powerful tools

Useful commands:

- **try0**: tries a bunch of internal provers (**auto, simp, ...**).
- **try**: try0 + sledgehammer + counterexample checkers!
- Use them just like sledgehammer.
Part 2: Geometry with order and signed areas [60%]
Part 2: Geometry with order and signed areas [60%]

• Getting familiar with axiomatic systems.
Part 2: Geometry with order and signed areas [60%]

- Getting familiar with axiomatic systems.
- In particular, Isabelle’s locales.
Part 2: Geometry with order and signed areas [60%]

• Getting familiar with axiomatic systems.
• In particular, Isabelle’s locales.
• We will define familiar geometric objects in terms of new concepts (order, signed area).
Part 2: Geometry with order and signed areas [60%]

- Getting familiar with axiomatic systems.
- In particular, Isabelle’s *locales*.
- We will define familiar geometric objects in terms of new concepts (order, signed area).
- It will help if we relate the formal statements to our geometric intuition.
Signed area

- You will be given a locale defining a function $\Delta$. 

The signed area of a triangle is just the area of that triangle, multiplied by $-1$ if the points of that triangle are traversed clockwise, and by $1$ otherwise.
Signed area

- You will be given a locale defining a function $\Delta$.
- We can interpret $\Delta x y z$ as the signed area of a triangle defined by the three arguments, $x$, $y$ and $z$, of $\Delta$. 
Signed area

- You will be given a locale defining a function $\Delta$.
- We can interpret $\Delta x y z$ as the signed area of a triangle defined by the three arguments, $x$, $y$ and $z$, of $\Delta$.
- The signed area of a triangle is just the area of that triangle, multiplied by $-1$ if the points of that triangle are traversed clockwise, and by 1 otherwise.
Relating the formal statement to geometry

- Take as an example Axiom 2 from the locale:
  "\( x \neq y \implies \exists z. (\mathbb{R}::\text{real})= \Delta x \ y \ z \)."
Relating the formal statement to geometry

• Take as an example Axiom 2 from the locale:
  "x \neq y \implies \exists z. \ (R::real)= \Delta x \ y \ z".

• Geometrically it says given two distinct points we can construct a triangle with any area (even negative)
Relating the formal statement to geometry

- Take as an example Axiom 2 from the locale: 
  "\( x \neq y \implies \exists z. (\mathbb{R}::\text{real})= \Delta x \ y \ z \)."
- Geometrically it says given two distinct points we can construct a triangle with any area (even negative)
Hints for proving together with Isabelle

- Always solve the problems in your head (or on paper), before applying rules!
Hints for proving together with Isabelle

- Always solve the problems in your head (or on paper), before applying rules!
- If in your proof in paper it’s clear that results P and Q are used in the proof, then try `using P Q sledgehammer`
- This gives the provers a hint.
Hints for proving together with Isabelle

- Always solve the problems in your head (or on paper), before applying rules!
- If in your proof in paper it’s clear that results P and Q are used in the proof, then try `using P Q  sledgehammer`
- This gives the provers a hint.
- Preinstantiate variables when trying to use a result in a proof: `using P[where x = “some term”] Q  sledgehammer`.
Hints for proving together with Isabelle

- Always solve the problems in your head (or on paper), before applying rules!
- If in your proof in paper it’s clear that results $P$ and $Q$ are used in the proof, then try using $P \land Q$ sledgehammer
- This gives the provers a hint.
- Preinstantiate variables when trying to use a result in a proof: using $P[\text{where } x = \text{“some term”}] \land Q$ sledgehammer.
- When in doubt add brackets.
Hints for proving together with Isabelle

- Always solve the problems in your head (or on paper), before applying rules!
- If in your proof in paper it’s clear that results P and Q are used in the proof, then try `using P Q sledgehammer`
- This gives the provers a hint.
- Preinstantiate variables when trying to use a result in a proof: `using P[where x = “some term”] Q sledgehammer`.
- When in doubt *add brackets*.
- When in doubt *add type constraints*. 
Hints for proving together with Isabelle

- Always solve the problems in your head (or on paper), before applying rules!
- If in your proof in paper it’s clear that results P and Q are used in the proof, then try using P Q  sledgehammer
- This gives the provers a hint.
- Preinstantiate variables when trying to use a result in a proof: using P[where x = “some term”] Q sledgehammer.
- When in doubt add brackets.
- When in doubt add type constraints.
- During a proof, if you know your goal is unprovable (e.g., false), go back one step!
Hints for proving together with Isabelle

- Always solve the problems in your head (or on paper), before applying rules!
- If in your proof in paper it’s clear that results P and Q are used in the proof, then try using P Q sledgehammer
- This gives the provers a hint.
- Preinstantiate variables when trying to use a result in a proof: using P[where x = “some term”] Q sledgehammer.
- When in doubt add brackets.
- When in doubt add type constraints.
- During a proof, if you know your goal is unprovable (e.g., false), go back one step!
- Counterexample checkers (Quickcheck, Nitpick) can help you realise you made a wrong turn. Either call them directly (typing quickcheck or nitpick), or simply type try. (Especially important for knights and knaves).
More hints

• Start early.
• Go to the lab sessions.
• Contact me by email: I.I.Morris@sms.ed.ac.uk.
More hints

- Start early.
More hints

• Start early.
• Go to the lab sessions.

Contact me by email: I.I.Morris@sms.ed.ac.uk.
More hints

- Start early.
- Go to the lab sessions.
- Contact me by email: I.I.Morris@sms.ed.ac.uk.