Automated Reasoning

Lecture 10: Isar – A Language for Structured Proofs

Jacques Fleuriot
jdf@inf.ed.ac.uk

Acknowledgement: Tobias Nipkow kindly provided the slides for this lecture
Apply scripts

- unreadable
- hard to maintain
- do not scale

No structure!
Apply scripts versus Isar proofs

Apply script = assembly language program

Isar proof = structured program with comments

But: apply still useful for proof exploration
A typical Isar proof

proof
  assume $formula_0$
  have $formula_1$ by simp
  : 
  have $formula_n$ by blast
  show $formula_{n+1}$ by \\
qed

proves $formula_0 \implies formula_{n+1}$
Isar core syntax

proof = proof [method] step* qed
   | by method

method = (simp ...) | (blast ...) | (induction ...) | ...

step = fix variables (\land)
   | assume prop (\equiv)
   | [from fact^+] (have | show) prop proof

prop = [name:] "formula"

fact = name | ...
Example: Cantor’s theorem

lemma \( \neg surj(f :: 'a \Rightarrow 'a \text{ set}) \)
proof default proof: assume \( surj \), show \( False \)
  assume \( a: surj f \)
  from \( a \) have \( b: \forall A. \exists a. A = f a \)
    by (simp add: surj_def)
  from \( b \) have \( c: \exists a. \{x. x \notin f x\} = f a \)
    by blast
  from \( c \) show \( False \)
    by blast
qed
**Abbreviations**

\[ \begin{align*}
\textit{this} & = \text{the previous proposition proved or assumed} \\
\textit{then} & = \textit{from \ this} \\
\textit{thus} & = \textit{then show} \\
\textit{hence} & = \textit{then have} \\
\end{align*} \]
using and with

\[
\begin{align*}
(have|show) \text{ prop } & \text{using } \text{facts} \\
= & \\
\text{from facts (have|show) prop} \\

\text{with facts} \\
= & \\
\text{from facts this}
\end{align*}
\]
Structured lemma statement

lemma
  fixes f :: "'a ⇒ 'a set"
  assumes s: "surj f"
  shows "False"
proof -
  no automatic proof step
  have "∃ a. {x. x ∉ f x} = f a" using s
    by (auto simp: surj_def)
  thus "False" by blast
qed

  Proves surj f ⟷ False
  but surj f becomes local fact s in proof.
The essence of structured proofs

Assumptions and intermediate facts can be named and referred to explicitly and selectively
Structured lemma statements

- **fixes** $x :: \tau_1$ and $y :: \tau_2$ ...
- **assumes** $a : P$ and $b : Q$ ...
- **shows** $R$

- **fixes** and **assumes** sections optional
- **shows** optional if no **fixes** and **assumes**
Proof patterns: Case distinction

```isar
show "R"
proof cases
  assume "P"
  :
  show "R" 
next
  assume "¬ P"
  :
  show "R" 
qed

have "P ∨ Q" 
then show "R"
proof
  assume "P"
  :
  :
  show "R" 
next
  assume "Q"
  :
  :
  show "R" 
qed
```
Proof patterns: Contradiction

\[
\begin{align*}
\text{show } & \neg P \\
\text{proof} & \\
\text{assume } & P \\
\vdots & \\
\text{show } & \text{False} \\
\text{qed} & \\
\end{align*}
\]

\[
\begin{align*}
\text{show } & P \\
\text{proof } & (\text{rule ccontr}) \\
\text{assume } & \neg P \\
\vdots & \\
\text{show } & \text{False} \\
\text{qed} & \\
\end{align*}
\]
Proof patterns: 

\[ \text{show } "P \iff Q" \]

proof

assume "P"

\[ \vdots \]

show "Q" \ldots

next

assume "Q"

\[ \vdots \]

show "P" \ldots

qed
Proof patterns: \( \forall \) and \( \exists \) introduction

show "\( \forall x. P(x) \)"
proof
  fix \( x \)  local fixed variable
  show "\( P(x) \)"  
qed

show "\( \exists x. P(x) \)"
proof
  :  
    show "\( P(\text{witness}) \)"  
qed
Proof patterns: $\exists$ elimination: obtain

\[
\begin{align*}
\textbf{have } & \exists x. P(x) \\
\textbf{then obtain } & x \textbf{ where } p : P(x) \textbf{ by blast} \\
\textbf{:} & x \textbf{ fixed local variable}
\end{align*}
\]

Works for one or more $x$
obtain example

lemma "surj(f :: 'a ⇒ 'a set)
proof
  assume surj f
  hence \( \exists a. \{x. x \notin f x\} = f a \) by (auto simp: surj_def)
  then obtain a where \( \{x. x \notin f x\} = f a \) by blast
  hence \( a \notin f a \) by blast
  thus False by blast
qed
Proof patterns: Set equality and subset

show "A = B"
proof
  show "A ⊆ B"  
next
  show "B ⊆ A"  
qed

show "A ⊆ B"
proof
  fix x
  assume "x ∈ A"
  ;
  show "x ∈ B"  
qed
Example: pattern matching

\[ \text{show } formula_1 \leftrightarrow formula_2 \quad (\text{is } ?L \leftrightarrow ?R) \]

proof
  \begin{align*}
  & \text{assume } ?L \\
  & \text{...} \\
  & \text{show } ?R \quad ... \\
  \end{align*}

next
  \begin{align*}
  & \text{assume } ?R \\
  & \text{...} \\
  & \text{show } ?L \quad ... \\
  \end{align*}

qed
Every show implicitly defines $\text{thesis}$
Introducing local abbreviations in proofs:

```plaintext
let ?t = "some-big-term"

: 

have "...?t ...
```
Quoting facts by value

By name:

```plaintext
have x0: "x > 0" ...
:
from x0 ...
```

By value:

```plaintext
have "x > 0" ...
:
from 'x>0' ...
```

↑  ↑
back quotes
Example

lemma

"(\exists \ ys \ zs. \ xs = ys @ zs \land \ length \ ys = length \ zs) \lor
(\exists \ ys \ zs. \ xs = ys @ zs \land \ length \ ys = length \ zs + 1)"

proof ???
When automation fails

Split proof up into smaller steps.

Or explore by apply:

have ... using ...

apply - to make incoming facts
  part of proof state

apply auto or whatever

apply ...

At the end:

- done

- Better: convert to structured proof
moreover—ultimately

have "P_1" ...
moreover
have "P_2" ...
moreover
... 
moreover
have "P_n" ...
ultimately
have "P" ...

have lab_1: "P_1" ...
have lab_2: "P_2" ...
...
have lab_n: "P_n" ...
from lab_1 lab_2 ...
have "P" ...

With names
{ \textbf{fix } x_1 \ldots x_n \\
\textbf{assume } A_1 \ldots A_m \\
\vdots \\
\textbf{have } B 
}\}

proves \([ A_1; \ldots ; A_m ] \Rightarrow B\)

where all \( x_i \) have been replaced by \( ?x_i \).
Proof state and Isar text

In general: **proof method**

Applies *method* and generates subgoal(s):

\[ \bigwedge x_1 \ldots x_n \left[ A_1; \ldots ; A_m \right] \Rightarrow B \]

How to prove each subgoal:

- **fix** \( x_1 \ldots x_n \)
- **assume** \( A_1 \ldots A_m \)
- **show** \( B \)

Separated by **next**
Datatype case analysis

datatype \( t = C_1 \vec{x} \mid \ldots \)

proof (cases "term")
  case \((C_1 x_1 \ldots x_k)\)
    \[\ldots x_j \ldots\]
  next
  \[\vdots\]
  qed

where

\[
\text{case } (C_i x_1 \ldots x_k) \equiv
\]

fix \( x_1 \ldots x_k \)

assume \( C_i : \) [\(\text{term} = (C_i x_1 \ldots x_k)\)]

\[\text{label}\]

\[\text{formula}\]
Structural induction for \( \text{nat} \)

\[
\text{show } P(n) \\
\text{proof } (\text{induction } n) \\
\quad \text{case } 0 \quad \equiv \quad \text{let } ?\text{case} = P(0) \\
\quad \vdots \\
\quad \text{show } ?\text{case} \\
\text{next} \\
\quad \text{case } (\text{Suc } n) \quad \equiv \quad \text{fix } n \text{ assume } \text{Suc: } P(n) \\
\quad \vdots \\
\quad \text{let } ?\text{case} = P(\text{Suc } n) \\
\quad \vdots \\
\quad \text{show } ?\text{case} \\
\text{qed}
\]
**Structural induction with**

\[
\text{show } A(n) \implies P(n) \\
\text{proof } (\text{induction } n) \\
\quad \text{case } 0 \\
\quad \quad \vdash \\
\quad \quad \text{show } ?\text{case} \\
\text{next} \\
\quad \text{case } (\text{Suc } n) \\
\quad \quad \vdash \\
\quad \quad \text{show } ?\text{case} \\
\text{qed}
\]

\[
\equiv \quad \text{assume } 0: A(0) \\
\quad \text{let } ?\text{case} = P(0) \\
\equiv \quad \text{fix } n \\
\quad \text{assume } \text{Suc: } A(n) \implies P(n) \\
\quad \quad A(\text{Suc } n) \\
\quad \text{let } ?\text{case} = P(\text{Suc } n)
\]
In a proof of
\[ A_1 \implies \ldots \implies A_n \implies B \]
by structural induction:
In the context of
\textbf{case} \ C
we have
\begin{align*}
& \textbf{C.IH} \quad \text{the induction hypotheses} \\
& \textbf{C.prems} \quad \text{the premises } A_i \\
& \quad \quad \quad \textbf{C} \quad \textbf{C.IH} + \textbf{C.prems}
\end{align*}
A remark on style

- **case** `(Suc n) ...show ?case`
  is easy to write and maintain

- **fix n assume formula ...show formula'**
  is easier to read:
  - all information is shown locally
  - no contextual references (e.g. ?case)
Rule induction

\textbf{inductive} \( I :: \tau \Rightarrow \sigma \Rightarrow \text{bool} \)

\textbf{where}

\textit{rule}_1 : \ldots

\textbf{show} \( I \ x \ y \Rightarrow P \ x \ y \)

\textbf{proof} \textit{(induction rule: I.induct)}

\textbf{case} \( \text{rule}_1 \)

\ldots

\textbf{show} \ ?\text{case}

\textbf{next}

\textbf{case} \( \text{rule}_n \)

\ldots

\textbf{show} \ ?\text{case}

\textbf{qed}
Fixing your own variable names

\[
\text{case } (rule_i \ x_1 \ldots \ x_k)
\]

Renames the first \( k \) variables in \( rule_i \) (from left to right) to \( x_1 \ldots x_k \).
Named assumptions

In a proof of

\[ I \implies A_1 \implies \ldots \implies A_n \implies B \]

by rule induction on \( I \ldots \):

In the context of

\textbf{case} \( R \)

we have

- \( R.IH \) the induction hypotheses
- \( R.hyps \) the assumptions of rule \( R \)
- \( R.prems \) the premises \( A_i \)

\( R \) \( R.IH + R.hyps + R.prems \)
Rule inversion

\textbf{inductive} \ ev :: "nat} \Rightarrow \ bool" \ \textbf{where}

\begin{align*}
\text{ev0: } & \text{"ev 0"} \\
\text{evSS: } & \text{"ev } n \Rightarrow \text{ ev(Suc(Suc } n))"
\end{align*}

What can we deduce from \( \text{ev } n \)?
That it was proved by either \text{ev0} or \text{evSS}!

\[
\text{ev } n \Rightarrow \ n = 0 \lor (\exists \ k. \ n = \text{Suc (Suc } k) \land \text{ev } k)
\]

\text{Rule inversion } = \text{ case distinction over rules}
from `ev n` have "P"
proof cases
  case ev0
    :  
    show ?thesis . . .
next
  case (evSS k)  
    :  
    show ?thesis . . .
qed

Impossible cases disappear automatically
Summary

- Introduction to Isar and to some common proof patterns e.g. case distinction, contradiction, etc.
- Structured proofs are becoming the norm for Isabelle as they are more readable and easier to maintain.
- Mastering structured proof takes practice and it is usually better to have a clear proof plan beforehand.
- Useful resource: Isar quick reference manual (see AR web page).
- Reading: N&K (Concrete Semantics), Chapter 5.