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*
  $\vdots$
  have $formula_n$ **by** *blast*
  show $formula_{n+1}$ **by** $\ldots$
qed

proves $formula_0 \Longrightarrow formula_{n+1}$

Isar core syntax

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

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

step	=	**fix** variables	$(\bigwedge)$
	|	**assume** prop	$(\Longrightarrow)$
	|	[**from** fact$^+$] (**have** | **show**) prop proof	

prop	=	[name:] "formula"

fact	=	name | ...

Example: Cantor's theorem

lemma $\neg$ *surj(f :: 'a $\Rightarrow$ 'a 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

this	=	the previous proposition proved or assumed
then	=	**from** *this*
thus	=	**then show**
hence	=	**then have**

using and with

(**have**|**show**) prop **using** facts

=

from facts (**have**|**show**) prop

with facts

=

from facts *this*

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 $surj\ f \Longrightarrow False$
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

show *"R"*
proof *cases*
  assume *"P"*
  $\vdots$
  show *"R"* ...
next
  assume *"¬ P"*
  $\vdots$
  show *"R"* ...
qed

have *"P ∨ Q"* ...
then show *"R"*
proof
  assume *"P"*
  $\vdots$
  show *"R"* ...
next
  assume *"Q"*
  $\vdots$
  show *"R"* ...
qed

Proof patterns: Contradiction

show *"¬ P"*
proof
  assume *"P"*
  ⋮
  show *"False"* . . .
qed

show *"P"*
proof *(rule ccontr)*
  assume *"¬P"*
  ⋮
  show *"False"* . . .
qed

Proof patterns: $\longleftrightarrow$

show *"$P \longleftrightarrow Q$"*
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 *"$P(witness)$"* . . .
qed

Proof patterns: $\exists$ elimination: obtain

have $\exists\, x.\ P(x)$
then obtain x **where** p: $P(x)$ **by** *blast*
$\vdots$ x fixed local variable

Works for one or more x

obtain example

lemma $\neg$ *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 $a \notin f\, a \longleftrightarrow a \in f\, a$ **by** *blast*
  thus *False* **by** *blast*
qed

Proof patterns: Set equality and subset

show *"$A = B$"*
proof
  show *"$A \subseteq B$"* $\ldots$
next
  show *"$B \subseteq A$"* $\ldots$
qed

show *"$A \subseteq B$"*
proof
  fix x
  assume *"$x \in A$"*
  $\vdots$
  show *"$x \in B$"* $\ldots$
qed

Example: pattern matching

show $formula_1 \longleftrightarrow formula_2$ (**is** $?L \longleftrightarrow ?R$)
proof
  assume $?L$
  $\vdots$
  show $?R$ ...
next
  assume $?R$
  $\vdots$
  show $?L$ ...
qed

?thesis

show *formula* (*is ?thesis*)
proof -
$\vdots$
show *?thesis* ...
qed

Every show implicitly defines *?thesis*

let

Introducing local abbreviations in proofs:

let $?t$ $=$ "some-big-term"

$\vdots$

have "... $?t$ ..."

Quoting facts by value

By name:

have *x0: "x > 0" ...*

$\vdots$

from *x0 ...*

By value:

have *"x > 0" ...*

$\vdots$

from *\`x>0\` ...*

$\uparrow \quad \uparrow$

back quotes

Example

lemma
"(∃ ys zs. xs = ys @ zs ∧ length ys = length zs) ∨
(∃ ys zs. xs = ys @ zs ∧ 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
$\vdots$
moreover
have $"P_n"$ ...
ultimately
have $"P"$ ...

$\approx$

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

With names

Raw proof blocks

$\{$ **fix** $x_1 \ldots x_n$
assume $A_1 \ldots A_m$
$\vdots$
have B
$\}$

proves $[\![A_1; \ldots ; A_m]\!] \Longrightarrow B$
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):

$$\bigwedge x_1 \dots x_n \; [\![A_1; \dots ; A_m]\!] \Longrightarrow B$$

How to prove each subgoal:

fix $x_1 \dots x_n$
assume $A_1 \dots A_m$
$\vdots$
show B

Separated by **next**

Datatype case analysis

datatype t = $C_1\ \vec{\tau}$ | ...

proof *(cases "term")*
  case $(C_1\ x_1\ \ldots\ x_k)$
  ... x_j ...
next
$\vdots$
qed

where **case** $(C_i\ x_1\ \ldots\ x_k)$ $\equiv$

fix $x_1\ \ldots\ x_k$
assume $\underbrace{C_i\!:}_{\text{label}}\ \underbrace{term = (C_i\ x_1\ \ldots\ x_k)}_{\text{formula}}$

Structural induction for *nat*

```
show P(n)
proof (induction n)
  case 0                ≡   let ?case = P(0)
  ⋮
  show ?case
next
  case (Suc n)          ≡   fix n assume Suc: P(n)
  ⋮                         let ?case = P(Suc n)
  ⋮
  show ?case
qed
```

Structural induction with $\Longrightarrow$

show $A(n) \Longrightarrow P(n)$
proof *(induction n)*
  case *0* $\equiv$ **assume** $0{:}\ A(0)$
  ⋮ **let** $?case = P(0)$
  show *?case*
next
  case *(Suc n)* $\equiv$ **fix** n
  ⋮ **assume** $Suc{:}\quad A(n) \Longrightarrow P(n)$
                  $A(Suc\ n)$
  ⋮ **let** $?case = P(Suc\ n)$
  show *?case*
qed

Named assumptions

In a proof of

$$A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow B$$

by structural induction:
In the context of

case C

we have

$\boldsymbol{C.IH}$	the induction hypotheses
$\boldsymbol{C.prems}$	the premises A_i
$\boldsymbol{C}$	$C.IH + C.prems$

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

inductive $I :: \tau \Rightarrow \sigma \Rightarrow bool$
where
$rule_1$: . . .
$\vdots$
$rule_n$: . . .

show $I\ x\ y \Longrightarrow P\ x\ y$
proof *(induction rule: I.induct)*
  case $rule_1$
  ...
  show *?case*
next
$\vdots$
next
  case $rule_n$
  ...
  show *?case*
qed

Fixing your own variable names

case $(\mathit{rule}_i\ x_1\ \ldots\ x_k)$

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

Named assumptions

In a proof of

$$I \ldots \Longrightarrow A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow B$$

by rule induction on $I \ldots$:
In the context of

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

inductive *ev* :: "*nat* $\Rightarrow$ *bool*" **where**
ev0: "*ev 0*" |
evSS: "*ev n* $\Longrightarrow$ *ev(Suc(Suc n))*"

What can we deduce from *ev n* ?
That it was proved by either *ev0* or *evSS* !

$$ev\ n \Longrightarrow n = 0 \lor (\exists\, k.\ n = Suc\ (Suc\ k) \land ev\ k)$$

Rule inversion = case distinction over rules

Rule inversion template

```
from `ev n` have "P"
proof cases
  case ev0                                  n = 0
   ⋮
  show ?thesis ...
next
  case (evSS k)                             n = Suc (Suc k), ev 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.