COMP4011/8011 Advanced Topics in Formal Methods and Programming Languages

– Software Verification with Isabelle/HOL –

Peter Höfner

August 6, 2024

Section 7

Isabelle/HOL
Isar (Part 1)
A Language for Structured Proofs

Motivation

Is this true: $(A \longrightarrow B) = (B \vee \neg A)$?

Motivation

Is this true: $(A \longrightarrow B) = (B \vee \neg A)$?

YES!

```
apply (rule iffI)
 apply (cases A)
  apply (rule disjI1)
  apply (erule impE)
   apply assumption
  apply assumption
 apply (rule disjI2)
 apply assumption
apply (rule impI)
apply (erule disjE)
 apply assumption
apply (erule notE)
apply assumption
done
```

or `by blast`

OK it's true. But WHY?

Motivation

WHY is this true: $(A \longrightarrow B) = (B \vee \neg A)$?

Demo

Isar

apply scripts	**What about..**
$\rightarrow$ hard to read	$\rightarrow$ Elegance?
$\rightarrow$ hard to maintain	$\rightarrow$ Explaining deeper insights?

No explicit structure.

Isar!

A typical Isar proof

proof
  assume $\mathit{formula}_0$
  have $\mathit{formula}_1$ **by** simp
  $\vdots$
  have $\mathit{formula}_n$ **by** blast
  show $\mathit{formula}_{n+1}$ **by** ...
qed

proves $\mathit{formula}_0 \Longrightarrow \mathit{formula}_{n+1}$

(analogous to **assumes**/**shows** in lemma statements)

Isar core syntax

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

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

statement = **fix** variables ($\bigwedge$)
| **assume** proposition ($\Longrightarrow$)
| [**from** name$^+$] (**have** | **show**) proposition proof
| **next** (separates subgoals)

proposition = [name:] formula

proof and qed

proof [method] statement* **qed**

lemma "$[\![A; B]\!] \Longrightarrow A \wedge B$"
proof (rule conjI)
  assume A: "A"
  from A **show** "A" **by** assumption
next
  assume B: "B"
  from B **show** "B" **by** assumption
qed

$\rightarrow$	**proof** (<method>)	applies method to the stated goal
$\rightarrow$	**proof**	applies a single rule that fits
$\rightarrow$	**proof -**	does nothing to the goal

How do I know what to Assume and Show?

Look at the proof state!

lemma "$[\![A; B]\!] \Longrightarrow A \wedge B$"
proof (rule conjI)

- **proof** (rule conjI) changes proof state to
  1. $[\![A; B]\!] \Longrightarrow A$
  2. $[\![A; B]\!] \Longrightarrow B$
- so we need 2 shows: **show** "A" and **show** "B"
- We are allowed to **assume** A,
  because A is in the assumptions of the proof state.

The Three Modes of Isar

- **[prove]**:
  goal has been stated, proof needs to follow.
- **[state]**:
  proof block has opened or subgoal has been proved,
  new *from* statement, goal statement or assumptions can follow.
- **[chain]**:
  from statement has been made, goal statement needs to follow.

lemma "$[\![A; B]\!] \Longrightarrow A \wedge B$" **[prove]**
proof (rule conjI) **[state]**
  assume A: "A" **[state]**
  from A **[chain]** **show** "A" **[prove]** **by** assumption **[state]**
next **[state]** **. . .**

Have

Can be used to make intermediate steps.

Example: **lemma** "$(x :: \mathsf{nat}) + 1 = 1 + x$"
proof -
  have A: "$x + 1 = \mathsf{Suc}\ x$" **by** simp
  have B: "$1 + x = \mathsf{Suc}\ x$" **by** simp
  show "$x + 1 = 1 + x$" **by** (simp only: A B)
qed

Demo

Backward and Forward

Backward reasoning: ... **have** "$A \wedge B$" **proof**

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: "$A \wedge B$"
from AB **have** "..." **proof**

- now **proof** picks an **elim** rule automatically
- triggered by **from**
- first assumption of rule must unify with AB

General case: **from** A_1 ... A_n **have** R **proof**

- first n assumptions of rule must unify with A_1 ... A_n
- conclusion of rule must unify with R

Fix and Obtain

- **fix** $v_1 \dots v_n$

  Introduces new arbitrary but fixed variables
  ($\sim$ parameters, $\bigwedge$)

- **obtain** $v_1 \dots v_n$ **where** $<$prop$>$ $<$proof$>$

  Introduces new variables together with property

Fancy Abbreviations

this	=	the previous fact proved or assumed
then	=	**from** this
thus	=	**then show**
hence	=	**then have**
with $A_1 \dots A_n$	=	**from** $A_1 \dots A_n$ this
?thesis	=	the last enclosing goal statement

Demo

Moreover and Ultimately

have X_1: P_1 ...	**have** P_1 ...
have X_2: P_2 ...	**moreover have** P_2 ...
$\vdots$	$\vdots$
have X_n: P_n ...	**moreover have** P_n ...
from $X_1 \ldots X_n$ **show** ...	**ultimately show** ...

wastes lots of brain power on names $X_1 \ldots X_n$

General Case Distinctions

show *formula*
proof -
  have $P_1 \vee P_2 \vee P_3$ <proof>
  moreover { **assume** P_1 ... **have** ?thesis <proof> }
  moreover { **assume** P_2 ... **have** ?thesis <proof> }
  moreover { **assume** P_3 ... **have** ?thesis <proof> }
  ultimately show ?thesis **by** blast
qed

{ ... } is a proof block similar to **proof ... qed**

{ **assume** P_1 ... **have** P <proof> }
stands for $P_1 \Longrightarrow P$

Mixing proof styles

from . . .
have . . .
  apply -   make incoming facts assumptions
  apply (. . .)
  ⋮
  apply (. . .)
  done

More on Automation

This can be automated

Automated methods (fast, blast, clarify etc) are not hardwired.
Safe/unsafe intro/elim rules can be declared.

Syntax:

[<kind>!]	for safe rules (<kind> one of intro, elim, dest)
[<kind>]	for unsafe rules

Application (roughly):
do safe rules first, search/backtrack on unsafe rules only

Example:

declare attribute globally	**declare** conjI [intro!] allE [elim]
remove attribute globally	**declare** allE [rule del]
use locally	**apply** (blast intro: someI)
delete locally	**apply** (blast del: conjI)

Demo: Automation

Exercises

- derive the classical contradiction rule $(\neg P \Longrightarrow \mathit{False}) \Longrightarrow P$ in Isabelle
- define **nor** and **nand** in Isabelle
- show $\;\mathsf{nor}\ x\ x \;=\; \mathsf{nand}\ x\ x$
- derive safe intro and elim rules for them
- use these in an automated proof of $\;\mathsf{nor}\ x\ x \;=\; \mathsf{nand}\ x\ x$