
Session III

More about Isar

Overview

- ▶ Abbreviations
- ▶ Predicate Logic
- ▶ Accumulating facts
- ▶ Reasoning with chains of equations
- ▶ Locales: the module system

Abbreviations

this = the previous proposition proved or assumed

then = from *this*

with \vec{a} = from \vec{a} *this*

?thesis = the last enclosing **show** formula

Mixing proof styles

from ...

have ...

apply - make incoming facts assumptions

apply(...)

:

apply(...)

done

Demo: Abbreviations

Predicate Calculus

fix

Syntax:

fix variables

Introduces new arbitrary but fixed variables
(~ parameters)

obtain

Syntax:

obtain variables where proposition proof

Introduces new variables together with property

Demo: predicate calculus

moreover/ultimately

have *formula*₁ . . .

have *formula*₂ . . .

⋮

have *formula*_{*n*} . . .

show . . .

proof . . .

moreover/ultimately

have *formula*₁ . . .

moreover

have *formula*₂ . . .

moreover

⋮

moreover

have *formula*_{*n*} . . .

show . . .

proof . . .

moreover/ultimately

have *formula*₁ . . .

moreover

have *formula*₂ . . .

moreover

⋮

moreover

have *formula*_{*n*} . . .

ultimately

show . . .

proof . . .

moreover/ultimately

have *formula*₁ . . .

moreover

have *formula*₂ . . .

moreover

⋮

moreover

have *formula*_{*n*} . . .

ultimately

show . . .

— pipes facts *formula*₁ . . . *formula*_{*n*} into the proof

proof . . .

Demo: moreover/ultimately

General case distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3 \dots$

General case distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3 \dots$

moreover

{ **assume** $P_1 \dots$ **have** *?thesis* \dots }

General case distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3 \dots$

moreover

{ **assume** $P_1 \dots$ **have** *?thesis* \dots }

moreover

{ **assume** $P_2 \dots$ **have** *?thesis* \dots }

General case distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3 \dots$

moreover

{ **assume** $P_1 \dots$ **have** *?thesis* \dots }

moreover

{ **assume** $P_2 \dots$ **have** *?thesis* \dots }

moreover

{ **assume** $P_3 \dots$ **have** *?thesis* \dots }

General case distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3 \dots$

moreover

{ **assume** $P_1 \dots$ **have** *?thesis* \dots }

moreover

{ **assume** $P_2 \dots$ **have** *?thesis* \dots }

moreover

{ **assume** $P_3 \dots$ **have** *?thesis* \dots }

ultimately **show** *?thesis* **by** *blast*

qed

Chains of equations

- ▶ Keywords **also** and **finally**.

Chains of equations

- ▶ Keywords **also** and **finally**.
- ▶ `...`: predefined schematic term variable, refers to the **right hand side of the last expression**.

Chains of equations

- ▶ Keywords **also** and **finally**.
- ▶ `...`: predefined schematic term variable, refers to the **right hand side of the last expression**.
- ▶ Uses transitivity rule.

also/finally

have " $t_0 = t_1$ " ...

also

have "... = t_2 " ...

also

⋮

also

have "... = t_n " ...

also/finally

have " $t_0 = t_1$ " ...

also

$$t_0 = t_1$$

have "... = t_2 " ...

also

⋮

also

have "... = t_n " ...

also/finally

have " $t_0 = t_1$ " ...

also

$$t_0 = t_1$$

have "... = t_2 " ...

also

$$t_0 = t_2$$

⋮

also

have "... = t_n " ...

also/finally

have " $t_0 = t_1$ " ...

also

$$t_0 = t_1$$

have "... = t_2 " ...

also

$$t_0 = t_2$$

⋮

⋮

also

$$t_0 = t_{n-1}$$

have "... = t_n " ...

also/finally

have " $t_0 = t_1$ " ...

also

$$t_0 = t_1$$

have "... = t_2 " ...

also

$$t_0 = t_2$$

⋮

⋮

also

$$t_0 = t_{n-1}$$

have "... = t_n " ...

finally show ...

— pipes fact $t_0 = t_n$ into the proof

proof

⋮

More about also

- ▶ Works for all combinations of $=$, \leq and $<$.

More about also

- ▶ Works for all combinations of $=$, \leq and $<$.
- ▶ Uses rules declared as `[trans]`.

More about also

- ▶ Works for all combinations of $=$, \leq and $<$.
- ▶ Uses rules declared as `[trans]`.
- ▶ To view all combinations in Proof General:
Isabelle/Isar \rightarrow Show me \rightarrow Transitivity rules

Demo: also/finally

Locales

Isabelle's Module System

Isar is based on contexts

theorem $\bigwedge x. A \implies C$

proof -

fix x

assume $ASS: A$

\vdots

from ASS **show** $C \dots$

qed

Isar is based on contexts

theorem $\bigwedge x. A \implies C$

proof -

fix x

assume $Ass: A$

\vdots

from Ass **show** $C \dots$

qed

x and Ass are visible
inside this context

Beyond Isar contexts

Locales are extended contexts

Beyond Isar contexts

Locales are extended contexts

- ▶ Locales are **named**

Beyond Isar contexts

Locales are extended contexts

- ▶ Locales are **named**
- ▶ Fixed variables may have **syntax**

Beyond Isar contexts

Locales are extended contexts

- ▶ Locales are **named**
- ▶ Fixed variables may have **syntax**
- ▶ It is possible to **add** and **export** theorems

Beyond Isar contexts

Locales are extended contexts

- ▶ Locales are **named**
- ▶ Fixed variables may have **syntax**
- ▶ It is possible to **add** and **export** theorems
- ▶ Locale expression: **combine** and **modify** locales

Context elements

Locales consist of **context elements**.

Context elements

Locales consist of **context elements**.

fixes Parameter, with syntax

Context elements

Locales consist of **context elements**.

fixes Parameter, with syntax

assumes Assumption

Context elements

Locales consist of **context elements**.

fixes	Parameter, with syntax
assumes	Assumption
defines	Definition

Context elements

Locales consist of **context elements**.

fixes	Parameter, with syntax
assumes	Assumption
defines	Definition
notes	Record a theorem

Declaring locales

locale *loc* =
loc1 +
fixes ...
assumes ...

Declaring locales

locale *loc* =
loc1 +
fixes ...
assumes ...

Declares **named locale** *loc*.

Declaring locales

locale *loc* =
loc1 + Import
fixes ...
assumes ...

Declares **named locale** *loc*.

Declaring locales

locale *loc* =

loc1 +

fixes ...

Context elements

assumes ...

Declares **named locale** *loc*.

Declaring locales

Theorems may be stated relative to a named locale.

```
lemma (in loc) P [simp]: proposition  
  proof
```

Declaring locales

Theorems may be stated relative to a named locale.

lemma (in *loc*) *P* [simp]: *proposition*
proof

- ▶ Adds theorem *P* to context *loc*.

Declaring locales

Theorems may be stated relative to a named locale.

lemma (in *loc*) *P* [simp]: *proposition*
proof

- ▶ Adds theorem *P* to context *loc*.
- ▶ Theorem *P* is in the simpset in context *loc*.

Declaring locales

Theorems may be stated relative to a named locale.

lemma (in *loc*) *P* [simp]: *proposition*
proof

- ▶ Adds theorem *P* to context *loc*.
- ▶ Theorem *P* is in the simpset in context *loc*.
- ▶ Exported theorem *loc.P* visible in the entire theory.

Demo: locales 1

Parameters must be consistent!

- ▶ Parameters in `fixes` are distinct.

Parameters must be consistent!

- ▶ Parameters in **fixes** are distinct.
- ▶ Free variables in **assumes** and **defines** occur in preceding **fixes**.

Parameters must be consistent!

- ▶ Parameters in **fixes** are distinct.
- ▶ Free variables in **assumes** and **defines** occur in preceding **fixes**.
- ▶ Defined parameters must neither occur in preceding **assumes** nor **defines**.

Locale expressions

Locale name: *n*

Locale expressions

Locale name: n

Rename: $e q_1 \dots q_n$

Change names of parameters in e .

Locale expressions

Locale name: n

Rename: $e \ q_1 \ \dots \ q_n$

Change names of parameters in e .

Merge: $e_1 + e_2$

Context elements of e_1 , then e_2 .

Locale expressions

Locale name: n

Rename: $e \ q_1 \ \dots \ q_n$

Change names of parameters in e .

Merge: $e_1 + e_2$

Context elements of e_1 , then e_2 .

- ▶ Syntax is lost after rename (**currently**).

Demo: locales 2

Normal form of locale expressions

Locale expressions are converted to flattened lists of locale names.

Normal form of locale expressions

Locale expressions are converted to flattened lists of locale names.

- ▶ With full parameter lists

Normal form of locale expressions

Locale expressions are converted to flattened lists of locale names.

- ▶ With full parameter lists
- ▶ Duplicates removed

Normal form of locale expressions

Locale expressions are converted to flattened lists of locale names.

- ▶ With full parameter lists
- ▶ Duplicates removed

Allows for multiple inheritance!

Instantiation

Move from **abstract** to **concrete**.

Instantiation

Move from **abstract** to **concrete**.

instantiate *label* : *loc*

Instantiation

Move from **abstract** to **concrete**.

instantiate *label* : *loc*

- ▶ From chained fact *loc t₁ ... t_n* instantiate locale *loc*.

Instantiation

Move from **abstract** to **concrete**.

instantiate *label* : *loc*

- ▶ From chained fact *loc t₁ ... t_n* instantiate locale *loc*.
- ▶ Imports all theorems of *loc* into current context.

Instantiation

Move from **abstract** to **concrete**.

instantiate *label* : *loc*

- ▶ From chained fact *loc t₁ ... t_n* instantiate locale *loc*.
- ▶ Imports all theorems of *loc* into current context.
 - ▶ Instantiates the parameters with *t₁ ... t_n*.
 - ▶ Interprets attributes of theorems.
 - ▶ Prefixes theorem names with *label*

Instantiation

Move from **abstract** to **concrete**.

instantiate *label* : *loc*

- ▶ From chained fact *loc* $t_1 \dots t_n$ instantiate locale *loc*.
- ▶ Imports all theorems of *loc* into current context.
 - ▶ Instantiates the parameters with $t_1 \dots t_n$.
 - ▶ Interprets attributes of theorems.
 - ▶ Prefixes theorem names with *label*
- ▶ **Currently only works inside Isar contexts.**

Demo: locales 3

Practical Session III

**The sun spills darkness
A dog howls after midnight
Goals remain unsolved.**

— Chris Owens