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
($\sim$ 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 $formula_1 \ldots formula_n$ into the proof

proof . . .

Demo: moreover/ultimately

General case distinctions

show $formula$

proof -

have $P_1 \vee P_2 \vee P_3$ **. . .**

General case distinctions

show $formula$

proof -

have $P_1 \vee P_2 \vee P_3 \ \ldots$

moreover

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

General case distinctions

show $formula$

proof -

  have $P_1 \vee P_2 \vee P_3 \ \ldots$

  moreover

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

  moreover

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

General case distinctions

show $formula$

proof -

have $P_1 \vee P_2 \vee P_3$ . . .

moreover

{ **assume** P_1 . . . **have** *?thesis* . . . }

moreover

{ **assume** P_2 . . . **have** *?thesis* . . . }

moreover

{ **assume** P_3 . . . **have** *?thesis* . . . }

General case distinctions

show $formula$
proof -
have *P*$_1 \vee$ *P*$_2 \vee$ *P*$_3$ **. . .**
moreover
{ **assume** *P*$_1$ **. . . have** *?thesis* **. . .** }
moreover
{ **assume** *P*$_2$ **. . . have** *?thesis* **. . .** }
moreover
{ **assume** *P*$_3$ **. . . have** *?thesis* **. . .** }
ultimately show *?thesis* **by** *blast*
qed

Chains of equations

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

Chains of equations

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

Chains of equations

- Keywords **also** and **finally**.
- $\ldots$: 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 "$\ldots$ = t_2" . . .

also

⋮

also

have "$\ldots$ = t_n" . . .

also/finally

have "$t_0 = t_1$" ...

also

have "$\ldots = t_2$" ...

also

$\vdots$

also

have "$\ldots = t_n$" ...

$t_0 = t_1$

also/finally

have "$t_0 = t_1$" ...

also $\quad t_0 = t_1$

have "$\ldots = t_2$" ...

also $\quad t_0 = t_2$

⋮

also

have "$\ldots = t_n$" ...

also/finally

have "$t_0 = t_1$" ...	
also	$t_0 = t_1$
have "$\ldots = t_2$" ...	
also	$t_0 = t_2$
$\vdots$	$\vdots$
also	$t_0 = t_{n-1}$
have "$\ldots = t_n$" ...	

also/finally

have "$t_0 = t_1$" . . .

also $t_0 = t_1$

have "$\ldots = t_2$" . . .

also $t_0 = t_2$

⋮ ⋮

also $t_0 = t_{\mathtt{n-1}}$

have "$\ldots = t_n$" . . .

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 $\bigwedge x.\ A \Longrightarrow C$

proof -

  fix x

  assume *Ass*: A

  $\vdots$

  from *Ass* **show** C ...

qed

Isar is based on contexts

theorem $\bigwedge x.\ A \Longrightarrow C$

proof -

fix x

assume Ass: A

⋮

from Ass **show** C ...

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: $e\ q_1\ \ldots\ q_n$

Change names of parameters in *e*.

Locale expressions

Locale name:	*n*
Rename:	$e\ q_1 \ldots q_n$
	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:	$e\ q_1\ \ldots\ q_n$
	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* $t_1 \ldots t_n$ instantiate locale *loc*.

Instantiation

Move from abstract to concrete.

instantiate *label* : *loc*

- From chained fact *loc* $t_1 \, \ldots \, t_n$ instantiate locale *loc*.
- Imports all theorems of *loc* into current context.

Instantiation

Move from abstract to concrete.

instantiate *label* : *loc*

- From chained fact *loc* $t_1 \ldots t_n$ instantiate locale *loc*.
- Imports all theorems of *loc* into current context.
  - Instantiates the parameters with $t_1 \ldots t_n$.
  - Interprets attributes of theorems.
  - Prefixes theorem names with *label*

Instantiation

Move from abstract to concrete.

instantiate *label* : *loc*

- From chained fact *loc* $t_1 \ldots t_n$ instantiate locale *loc*.
- Imports all theorems of *loc* into current context.
  - Instantiates the parameters with $t_1 \ldots t_n$.
  - 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