Isar — A language for structured proofs

Apply scripts

- unreadable

Apply scripts

- unreadable
- hard to maintain

Apply scripts

- unreadable
- hard to maintain
- do not scale

Apply scripts

- unreadable
- hard to maintain
- do not scale

No structure!

Apply scripts versus Isar proofs

Apply script = assembly language program

Apply scripts versus Isar proofs

Apply script = assembly language program

Isar proof = structured program with comments

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

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 $formula_0 \Longrightarrow formula_{n+1}$

Overview

- Basic Isar
- Propositional logic
- Predicate logic

Isar core syntax

proof	=	**proof** [method] statement* **qed**
	$\mid$	**by** method

Isar core syntax

$$\begin{array}{lcl} \text{proof} & = & \textbf{proof}\ [\text{method}]\ \text{statement}^*\ \textbf{qed} \\ & | & \textbf{by}\ \text{method} \end{array}$$

$$\text{method} \quad = \quad (\textit{simp}\ \ldots) \mid (\textit{blast}\ \ldots) \mid (\textit{rule}\ \ldots) \mid \ldots$$

Isar core syntax

$$
\begin{array}{lcl}
\text{proof} & = & \textbf{proof}\ [\text{method}]\ \text{statement}^*\ \textbf{qed} \\
 & | & \textbf{by}\ \text{method}
\end{array}
$$

$$
\text{method} \quad = \quad \textit{(simp} \ldots\textit{)} \mid \textit{(blast} \ldots\textit{)} \mid \textit{(rule} \ldots\textit{)} \mid \ldots
$$

$$
\begin{array}{lcll}
\text{statement} & = & \textbf{fix}\ \text{variables} & (\bigwedge) \\
 & | & \textbf{assume}\ \text{proposition} & (\Longrightarrow) \\
 & | & [\textbf{from}\ \text{name}^+]\ (\textbf{have} \mid \textbf{show})\ \text{proposition proof} &
\end{array}
$$

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)

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

Demo: propositional logic, introduction rules

Basic proof methods

Basic atomic proof:

by $method$

apply $method$, then prove all subgoals by assumption

Basic proof methods

Basic atomic proof:

by $method$
apply $method$, then prove all subgoals by assumption

Basic proof method:

rule $\vec{a}$
apply a rule in $\vec{a}$;

Basic proof methods

Basic atomic proof:

by $method$
apply $method$, then prove all subgoals by assumption

Basic proof method:

rule $\vec{a}$
apply a rule in $\vec{a}$;
if $\vec{a}$ is empty: apply a standard elim or intro rule.

Basic proof methods

Basic atomic proof:

by $method$
apply $method$, then prove all subgoals by assumption

Basic proof method:

rule $\vec{a}$
apply a rule in $\vec{a}$;
if $\vec{a}$ is empty: apply a standard elim or intro rule.

Abbreviations:

.	=	**by** do-nothing
..	=	**by** *rule*

Demo: propositional logic, elimination rules

Elimination rules / forward reasoning

- Elim rules are triggered by facts fed into a proof:
  from $\vec{a}$ **have** $formula$ **proof**

Elimination rules / forward reasoning

- Elim rules are triggered by facts fed into a proof:
  from $\vec{a}$ **have** $formula$ **proof**
- **proof** alone abbreviates **proof** *rule*

Elimination rules / forward reasoning

- Elim rules are triggered by facts fed into a proof:
  from $\vec{a}$ **have** $formula$ **proof**
- **proof** alone abbreviates **proof** *rule*
- *rule*: tries elim rules first (if there are incoming facts $\vec{a}$!)

Elimination rules / forward reasoning

- Elim rules are triggered by facts fed into a proof:
  from $\vec{a}$ **have** $formula$ **proof**
- **proof** alone abbreviates **proof** *rule*
- *rule*: tries elim rules first (if there are incoming facts $\vec{a}$!)
- **from** $\vec{a}$ **have** $formula$ **proof** *(rule* $rule$*)*

Elimination rules / forward reasoning

- Elim rules are triggered by facts fed into a proof:
  from $\vec{a}$ **have** $formula$ **proof**
- **proof** alone abbreviates **proof** *rule*
- *rule*: tries elim rules first (if there are incoming facts $\vec{a}$!)
- **from** $\vec{a}$ **have** $formula$ **proof** *(rule* $rule$*)*
  $\vec{a}$ must prove the first n premises of $rule$,

Elimination rules / forward reasoning

- Elim rules are triggered by facts fed into a proof:
  from $\vec{a}$ **have** $formula$ **proof**
- **proof** alone abbreviates **proof** *rule*
- *rule*: tries elim rules first (if there are incoming facts $\vec{a}$!)
- **from** $\vec{a}$ **have** $formula$ **proof** *(rule* $rule$*)*
  $\vec{a}$ must prove the first n premises of $rule$, in the right order

Elimination rules / forward reasoning

- Elim rules are triggered by facts fed into a proof:
  from $\vec{a}$ **have** $formula$ **proof**
- **proof** alone abbreviates **proof** *rule*
- *rule*: tries elim rules first (if there are incoming facts $\vec{a}$**!**)
- **from** $\vec{a}$ **have** $formula$ **proof** *(rule* $rule$*)*
  $\vec{a}$ must prove the first n premises of $rule$, in the right order
  the others are left as new subgoals

Abbreviations

this	=	the previous proposition proved or assumed
then	=	**from** *this*
thus	=	**then show**
hence	=	**then have**
with $\vec{a}$	=	**from** $\vec{a}$ *this*

using

First the what, then the how:

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

using

First the what, then the how:

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

=

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

using

First the what, then the how:

(**have**|**show**) proposition **using** facts
=
from facts (**have**|**show**) proposition

Can be mixed:

from major-facts (**have**|**show**) proposition **using** minor-facts

using

First the what, then the how:

(**have**|**show**) proposition **using** facts
=
from facts (**have**|**show**) proposition

Can be mixed:

from major-facts (**have**|**show**) proposition **using** minor-facts
=
from major-facts minor-facts (**have**|**show**) proposition

Demo: avoiding duplication

Schematic term variables

$?A$

Schematic term variables

$$?A$$

- Defined by pattern matching:

$$x = 0 \wedge y = 1 \ (\textbf{is} \ ?A \wedge \_)$$

Schematic term variables

$$?A$$

- Defined by pattern matching:

  $$x = 0 \wedge y = 1 \; (\textbf{is} \; ?A \wedge \_)$$

- Predefined: *?thesis*
  The last enclosing **show** formula

Demo: predicate calculus

obtain

Syntax:

obtain variables **where** proposition proof

Mixing proof styles

from . . .

have . . .

  apply - make incoming facts assumptions

  apply(. . .)

  ⋮

  apply(. . .)

  done

Advanced Isar

Overview

- Case distinction
- Induction
- Calculational reasoning

Case distinction

Boolean case distinction

proof *cases*

  assume $formula$

  $\vdots$

next

  assume $\neg formula$

  $\vdots$

qed

Boolean case distinction

proof *cases*
  assume $formula$
  $\vdots$
next
  assume $\neg formula$
  $\vdots$
qed

proof (*cases* $formula$)
  case *True*
  $\vdots$
next
  case *False*
  $\vdots$
qed

Boolean case distinction

proof *cases*

  assume $formula$

  $\vdots$

next

  assume $\neg formula$

  $\vdots$

qed

proof *(cases* $formula$*)*

  case *True*

  $\vdots$

next

  case *False*

  $\vdots$

qed

case *True* $\equiv$

assume *True:* $formula$

Demo: case distinction

Datatype case distinction

proof *(cases* $term$*)*

  case *Constructor* $_1$

  ⋮

next

⋮

next

  case *(Constructor* $_k$ $\vec{x}$*)*

  $\cdots\ \vec{x}\ \cdots$

qed

Datatype case distinction

proof *(cases* $term$*)*

  case *Constructor* $_1$

  ⋮

next

⋮

next

  case *(Constructor* $_k$ $\vec{x}$*)*

  $\cdots$ $\vec{x}$ $\cdots$

qed

case *(Constructor* $_i$ $\vec{x}$*)* $\quad\equiv$

fix $\vec{x}$ **assume** *Constructor* $_i$*:* $term$ = *(Constructor* $_i$ $\vec{x}$*)*

Induction

Overview

- Structural induction
- Rule induction
- Induction with recdef

Structural induction for type nat

show $P(n)$
proof *(induction n)*
  case *0*
  ...
  show *?case*
next
  case *(Suc n)*
  ...
  $\cdots$ *n* $\cdots$
  show *?case*
qed

Structural induction for type nat

show $P(n)$

proof *(induction n)*

  case *0* $\equiv$ **let** *?case* = $P(0)$

  ...

  show *?case*

next

  case *(Suc n)*

  ...

  $\cdots$ *n* $\cdots$

  show *?case*

qed

Structural induction for type nat

show $P(n)$
proof *(induction n)*
  case *0* $\equiv$ **let** *?case* = $P(0)$
  . . .
  show *?case*
next
  case *(Suc n)* $\equiv$ **fix** *n* **assume** *Suc:* $P(n)$
  . . . **let** *?case* = $P(Suc\ n)$
  $\cdots$ *n* $\cdots$
  show *?case*
qed

Demo: structural induction

Structural induction with $\Longrightarrow$ and $\bigwedge$

show $\bigwedge x.\ A(n) \Longrightarrow P(n)$

proof *(induction n)*

  case *0*

  $\dots$

  show *?case*

next

  case *(Suc n)*

  $\dots$

  $\cdots\ n\ \cdots$

  $\dots$

  show *?case*

qed

Structural induction with $\Longrightarrow$ and $\bigwedge$

show $\bigwedge x.\ A(n) \Longrightarrow P(n)$

proof *(induction n)*

case *0* $\equiv$ **fix** x **assume** *0:* $A(0)$

let *?case* $= P(0)$

$\ldots$

show *?case*

next

case *(Suc n)*

$\ldots$

$\cdots\ n\ \cdots$

$\ldots$

show *?case*

qed

Structural induction with $\Longrightarrow$ and $\bigwedge$

show $\bigwedge x.\ A(n) \Longrightarrow P(n)$
proof *(induction n)*

case 0	$\equiv$	**fix** x **assume** $0{:}\ A(0)$
...		**let** $?case = P(0)$
show $?case$		

next

case $(Suc\ n)$	$\equiv$	**fix** $n\ x$
...		**assume** $Suc{:}\ \bigwedge x.\ A(n) \Longrightarrow P(n)$
$\cdots\ n\ \cdots$		$A(Suc\ n)$
...		**let** $?case = P(Suc\ n)$
show $?case$		

qed

A remark on style

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

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*)

Demo: structural induction with $\Longrightarrow$ and $\bigwedge$

Rule induction

Inductive definition

inductive S

intros

rule$_1$: $[\![$ *s* $\in$ *S; A* $]\!] \Longrightarrow$ *s'* $\in$ *S*

$\vdots$

rule$_n$: $\ldots$

Rule induction

show $x \in S \Longrightarrow P(x)$

proof *(induct rule: S.induct)*

  case $rule_1$

  ...

  show *?case*

next

$\vdots$

next

  case $rule_n$

  ...

  show *?case*

qed

Implicit selection of induction rule

assume A: $x \in S$

$\vdots$

show $P(x)$

using A **proof** *induct*

$\vdots$

qed

Implicit selection of induction rule

assume A: $x \in S$
$\vdots$
show $P(x)$
using A **proof** *induct*
$\vdots$
qed

lemma assumes A: $x \in S$ **shows** $P(x)$
using A **proof** *induct*
$\vdots$
qed

Renaming free variables in rule

case *(rule*$_i$ *x*$_1$ $\ldots$ *x*$_k$*)*

Renames the (alphabetically!) first k variables in *rule*$_i$ to *x*$_1$ $\ldots$ *x*$_k$.

Demo: rule induction

Induction with recdef

Definition:

recdef *f*

⋮

Induction with recdef

Definition:

recdef *f*

$\vdots$

Proof:

show $\ldots$ *f(*$\ldots$*)* $\ldots$

proof *(induction* $x_1 \ldots x_k$ *rule: f.induct)*

Induction with recdef

Definition:

recdef *f*

$\vdots$

Proof:

show $\ldots$ *f*($\ldots$) $\ldots$

proof *(induction x* $_1$ $\ldots$ *x* $_k$ *rule: f.induct)*

  case *1*

  $\vdots$

Induction with recdef

Definition:

recdef *f*

$\vdots$

Proof:

show $\ldots$ *f(*$\ldots$*)* $\ldots$

proof *(induction x*$_1$ $\ldots$ *x*$_k$ *rule: f.induct)*

case *1*

$\vdots$

Case i refers to equation i in the definition of *f*

Induction with recdef

Definition:

recdef *f*

$\vdots$

Proof:

show $\ldots$ *f(*$\ldots$*)* $\ldots$

proof *(induction x*$_1$ $\ldots$ *x*$_k$ *rule: f.induct)*

  case *1*

  $\vdots$

Case i refers to equation i in the definition of *f*
More precisely: to equation i in *f.simps*

Demo: induction with recdef

Calculational Reasoning

Overview

- Accumulating facts
- Chains of equations and inequations

moreover

have $formula_1$ **. . .**

moreover

have $formula_2$ **. . .**

moreover

$\vdots$

moreover

have $formula_n$ **. . .**

ultimately show . . .

— pipes facts $formula_1 \ldots formula_n$ into the proof

proof

$\vdots$

also

have "t_0 = t_1" . . . .

also

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

also

$\vdots$

also

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

also

have "t_0 = t_1" . . . .

also

have "$\ldots$ = t_2" . . . . $\ldots \equiv t_1$

also

$\vdots$

also

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

also

have "t_0 = t_1" . . . .

also

have "$\dots$ = t_2" . . . . $\dots \equiv t_1$

also

$\vdots$

also

have "$\dots$ = t_n" . . . . $\dots \equiv t_{n-1}$

also

have "t_0 = t_1" . . . .

also

have "$\ldots$ = t_2" . . . . $\ldots \equiv t_1$

also

$\vdots$

also

have "$\ldots$ = t_n" . . . . $\ldots \equiv t_{n-1}$

finally **show** . . . .

— pipes fact t_0 = t_n into the proof

proof

$\vdots$

...

"..." is merely an abbreviation

Demo: moreover and also

Variations on also

Transitivity:

have "t_0 = t_1" . . . .

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

also/finally $\leadsto$

Variations on also

Transitivity:

have "t_0 = t_1" . . . .

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

also/finally $\leadsto$ t_0 = t_2

Variations on also

Transitivity:

have "t_0 = t_1" . . . .

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

also/finally $\leadsto$ t_0 = t_2

Substitution:

have "$P(s)$" . . . .

also have "s = t" . . . .

also/finally $\leadsto$

Variations on also

Transitivity:

have "t_0 = t_1" . . . .
also have "$\ldots$ = t_2" . . . .
also/finally $\leadsto$ t_0 = t_2

Substitution:

have "$P(s)$" . . . .
also have "s = t" . . . .
also/finally $\leadsto$ $P(t)$

From $=$ *to* $\leq$ *and* $<$

Transitivity:

have "$t_0 \leq t_1$" **. . . .**

also have "$\ldots \leq t_2$" **. . . .**

also/finally $\leadsto$

From $=$ to $\leq$ and $<$

Transitivity:

have "$t_0 \leq t_1$" **. . . .**

also have "$\ldots \leq t_2$" **. . . .**

also/finally $\leadsto t_0 \leq t_2$

From $=$ to $\leq$ and $<$

Transitivity:

have "$t_0 \leq t_1$" **. . . .**

also have "$\ldots \leq t_2$" **. . . .**

also/finally $\leadsto t_0 \leq t_2$

Substitution:

have "$r \leq f(s)$" **. . . .**

also have "s **<** t" **. . . .**

also/finally $\leadsto$

From $=$ *to* $\leq$ *and* $<$

Transitivity:

have "$t_0 \leq t_1$" . . . .

also have "$\ldots \leq t_2$" . . . .

also/finally $\leadsto t_0 \leq t_2$

Substitution:

have "$r \leq f(s)$" . . . .

also have "$s < t$" . . . .

also/finally $\leadsto (\bigwedge x.\ x < y \Longrightarrow f(x) < f(y)) \Longrightarrow r < f(t)$

From $=$ *to* $\leq$ *and* $<$

Transitivity:

have "$t_0 \leq t_1$" **. . . .**
also have "$\ldots \leq t_2$" **. . . .**
also/finally $\leadsto t_0 \leq t_2$

Substitution:

have "$r \leq f(s)$" **. . . .**
also have "$s < t$" **. . . .**
also/finally $\leadsto (\bigwedge x.\ x < y \Longrightarrow f(x) < f(y)) \Longrightarrow r < f(t)$

Similar for all other combinations of $=$, $\leq$ and $<$.

All about also

To view all combinations in Proof General:

Isabelle/Isar $\rightarrow$ Show me $\rightarrow$ Transitivity rules

Demo: monotonicity reasoning