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

proves $formula_0 \Rightarrow formula_{n+1}$
Overview

- Basic Isar
- Propositional logic
- Predicate logic
**Isar core syntax**

\[
\text{proof} \quad = \quad \text{proof} \ [\text{method}] \ \text{statement}^* \ \text{qed} \\
| \quad \text{by} \ \text{method}
\]
Isar core syntax

proof = proof [method] statement* \text{qed}
\quad | \quad \text{by method}

method = (simp \ldots) \mid (blast \ldots) \mid (rule \ldots) \mid \ldots
**Isar core syntax**

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

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

**statement** = fix variables  (\wedge)
   | assume proposition  (\implies)
   | [from name^+] (have | show) proposition proof
**Isar core syntax**

\[
\begin{align*}
\text{proof} & \ = \ \text{proof} \ [\text{method}] \ \text{statement}^* \ \text{qed} \\
& \quad | \ \text{by} \ \text{method} \\
\text{method} & \ = \ (\text{simp} \ldots) | (\text{blast} \ldots) | (\text{rule} \ldots) | \ldots \\
\text{statement} & \ = \ \text{fix} \ \text{variables} \quad (\land) \\
& \quad | \ \text{assume} \ \text{proposition} \quad (\implies) \\
& \quad | \ [\text{from name}^+] \ (\text{have} | \text{show}) \ \text{proposition} \ \text{proof} \\
& \quad | \ \text{next} \quad (\text{separates subgoals})
\end{align*}
\]
Isar core syntax

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

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

statement = fix variables (∧)
           | assume proposition (⇒)
           | [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:
  \textbf{from \bar{a} have \textit{formula} proof}
Elimination rules / forward reasoning

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

- \textit{proof} alone abbreviates \textit{proof} \textit{rule}
Elimination rules / forward reasoning

• Elim rules are triggered by facts fed into a proof:
  
  \[
  \text{from } \vec{a} \text{ have } \text{formula } \text{proof}
  \]

• 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:
  \textit{from } \vec{a} \textit{ have } \textit{formula } \textit{proof}

• \textit{proof} alone abbreviates \textit{proof } \textit{rule}

• \textit{rule}: tries elim rules first (if there are incoming facts $\vec{a}$!)

• \textit{from } \vec{a} \textit{ have } \textit{formula } \textit{proof } (\textit{rule } \textit{rule})
Elimination rules / forward reasoning

• Elim rules are triggered by facts fed into a proof:
  \[ \text{from } \vec{a} \text{ have formula proof} \]
• 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:
  \( \text{from } \vec{a} \text{ have } \text{formula } \text{proof} \)
- 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 \( \text{formula} \) proof`

- `proof` alone abbreviates `proof rule`

- `rule`: tries elim rules first (if there are incoming facts \( \vec{a}! \))

- `from \( \vec{a} \) have \( \text{formula} \) 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

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>this</em></td>
<td>the previous proposition proved or assumed</td>
</tr>
<tr>
<td>then</td>
<td>from <em>this</em></td>
</tr>
<tr>
<td>thus</td>
<td>then show</td>
</tr>
<tr>
<td>hence</td>
<td>then have</td>
</tr>
<tr>
<td>with (\vec{a})</td>
<td>from (\vec{a}) <em>this</em></td>
</tr>
</tbody>
</table>
First the what, then the how:

(have|show) proposition using facts
First the what, then the how:

\[(\text{have}|\text{show}) \text{ proposition using facts} \]

\[=\]

\[\text{from facts (have|show) proposition}\]
First the what, then the how:

\[(\text{have}|\text{show}) \text{ proposition } \textit{using} \text{ facts} = \text{from facts } (\text{have}|\text{show}) \text{ proposition}\]

Can be mixed:

\[\text{from major-facts } (\text{have}|\text{show}) \text{ proposition } \textit{using} \text{ minor-facts}\]
First the what, then the how:

\[(\text{have}|\text{show}) \text{ proposition using facts} = \text{from facts (have|show) proposition}\]

Can be mixed:

\[\text{from major-facts (have|show) proposition using minor-facts} = \text{from major-facts minor-facts (have|show) proposition}\]
Demo: avoiding duplication
Schematic term variables

?A
Schematic term variables

?A

• Defined by pattern matching:

\[ x = 0 \land y = 1 \text{ (is } ?A \land _) \]
Schematic term variables

?A

• Defined by pattern matching:

\[ x = 0 \land y = 1 \ (\text{is} \ \ ?A \land _) \]

• Predefined: \( ?\text{thesis} \)
  The last enclosing show formula
Demo: predicate calculus
Syntax:

```
obtain variables where proposition proof
```
Mixing proof styles

from . . .

have . . .

apply - make incoming facts assumptions

apply( . . . )

::

apply( . . . )

done
Overview

- Case distinction
- Induction
- Calculational reasoning
Case distinction
Boolean case distinction

proof cases
  assume \( formula \)
  :
  next
  assume \( \neg formula \)
  :
qed
**Boolean case distinction**

proof cases

assume \( \text{formula} \)

:  

next

assume \( \neg \text{formula} \)

:  

qed

proof (cases \( \text{formula} \))

case True

:  

next

case False

:  

qed
### Boolean case distinction

proof cases
  assume \( \text{formula} \)
  :
next
  assume \( \neg \text{formula} \)
  :
qed

proof \((\text{cases } \text{formula})\)
  case \(\text{True}\)
  :
next
  case \(\text{False}\)
  :
qed

\[\begin{align*}
\text{case } \text{True} & \equiv \\
\text{assume } \text{True}: \text{formula}
\end{align*}\]
Demo: case distinction
Datatype case distinction

proof \((\text{cases } \text{term})\)

\begin{align*}
\text{case } \text{Constructor}_1 \\
\vdots \\
\text{next} \\
\vdots \\
\text{next} \\
\text{case } (\text{Constructor}_k \vec{x}) \\
\vdots \vec{x} \vdots
\end{align*}

qed
Datatype case distinction

proof (cases \textit{term})

\begin{align*}
\text{case } \text{Constructor}_1 \\
\vdots \\
\text{next} \\
\vdots \\
\text{next} \\
\text{case } (\text{Constructor}_k \vec{x}) \\
\vdots \vec{x} \vdots \\
\text{qed}
\end{align*}

\begin{align*}
\text{case } (\text{Constructor}_i \vec{x}) & \equiv \\
\text{fix } \vec{x} \text{ assume } \text{Constructor}_i : \text{term} = (\text{Constructor}_i \vec{x})
\end{align*}
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 $n$)
  ...
  ... $n$ ...
  show ?case
qed
**Structural induction for type nat**

\[
\begin{align*}
\text{show } & P(n) \\
\text{proof } & (\text{induction } n) \\
\text{case } & 0 \\
& \equiv \text{ let } ?\text{case} = P(0) \\
& \ldots \\
& \text{show } ?\text{case} \\
\text{next} \\
\text{case } & (\text{Suc } n) \\
& \ldots \\
& \ldots n \ldots \\
& \text{show } ?\text{case} \\
\text{qed}
\end{align*}
\]
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 = \( P(\text{Suc } n) \)
  ... \( n \) ...
  show ?case
qed
Demo: structural induction
Structural induction with $\Rightarrow$ and $\wedge$

\[
\begin{align*}
\text{show } & \forall x. A(n) \Rightarrow P(n) \\
\text{proof (induction } n) & \\
\text{ case 0 } & \\
\ldots & \\
\text{ show } & \text{ ?case} \\
\text{ next } & \\
\text{ case } (Suc n) & \\
\ldots & \\
\ldots & n \\
\ldots & \\
\text{ show } & \text{ ?case} \\
\text{ qed}
\end{align*}
\]
Structural induction with \(\implies\) and \(\wedge\)

show \(\forall x. A(n) \implies P(n)\)

proof (induction \(n\))
  case 0
    
    ...  
    show ?case
  next
  case \((\text{Suc } n)\)
    
    ...  
    ...  \(n\)  ...
    
    ...  
    show ?case
qed
Structural induction with $\implies$ and $\land$

show $\land x. A(n) \implies P(n)$

proof (induction $n$)
  case 0
    ...  
    show $\text{?case}$

next
  case $(\text{Suc } n)$
    ...  
    ... $n$ ...
    ...  
    show $\text{?case}$

qed
A remark on style

• case \( (\text{Suc } n) \ldots \text{show } \texttt{?case} \) 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 $\Rightarrow$ and $\land$
Rule induction
**Inductive definition**

inductive $S$

intros

$\text{rule}_1 : [ s \in S; A ] \implies s' \in S$

$\vdots$

$\text{rule}_n : \ldots$
Rule induction

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

proof (induct rule: $S.induct$)

  case $rule_1$

  ... 

  show $?case$

next

  ...

next

  case $rule_n$

  ... 

  show $?case$

qed
Implicit selection of induction rule

assume $A: x \in S$

:\

show $P(x)$

using $A$ proof *induct*

:\

qed
Implicit selection of induction rule

assume \( A: x \in S \)

\[ \vdots \]

show \( P(x) \)

using \( A \) proof \textit{induct}

\[ \vdots \]

qed
Renaming free variables in rule

\[ \text{case } (\text{rule}_i \ x_1 \ldots \ x_k) \]

Renames the (alphabetically!) first \( k \) variables in \( \text{rule}_i \) to \( x_1 \ldots x_k \).
Demo: rule induction
Definition:

\texttt{recdef } \texttt{f}

:
Definition:
\textit{recdef} \ f

Proof:
\textit{show} \ldots \ f(\ldots) \ldots
\textit{proof} (\textit{induction} \ \! x_1 \ldots \ x_k \ \textit{rule}: \ f.\textit{induct})
**Induction with recdef**

Definition:

recdef $f$

Proof:

show ... $f(...)$ ...

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

  case 1

  :
Induction with recdef

Definition:
recdef f
:

Proof:
show \ldots \ f(\ldots) \ldots
proof (induction x_1 \ldots x_k \ rule: f.induct)
  case 1
    :

Case \ i \ refers\ to\ equation \ i \ in\ the\ definition\ of \ f
Induction with recdef

Definition:
recdef $f$
:

Proof:
show ... $f(...)$ ...
proof (induction $x_1 \ldots x_k$ rule: $f.induct$)
  case 1
  :

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
.
moreover
have formula_n . . .
ultimately show . . .
— pipes facts formula_1 . . . formula_n into the proof
proof
.

moreover
also

have \( t_0 = t_1 \ldots \).
also
have \( \ldots = t_2 \ldots \).
also
\[
\vdots
\]
also
have \( \ldots = t_n \ldots \).
also

have \( t_0 = t_1 \ldots \)
also
have \( \ldots = t_2 \ldots \)
also
\vdots
also
have \( \ldots = t_n \ldots \)
also
also

\[ \text{have } t_0 = t_1 \ldots \]

also

\[ \text{have } \ldots = t_2 \ldots \]

\[ \equiv t_1 \]

also

\[ \vdots \]

also

\[ \text{have } \ldots = t_n \ldots \]

\[ \equiv t_{n-1} \]
also

have $t_0 = t_1 \ldots$
also
have $\ldots = t_2 \ldots$
also
$\vdots$
also
have $\ldots = t_n \ldots$

finally show $\ldots$
— pipes fact $t_0 = t_n$ into the proof
proof
$\vdots$
“...” is merely an abbreviation
Demo: moreover and also
Variations on also

Transitivity:

have $t_0 = t_1 \ldots$
also have $\ldots = t_2 \ldots$
also/finally $\sim\rightarrow$
Variations on also

Transitivity:

have $t_0 = t_1$ . . . .
also have . . . = $t_2$ . . . .
also/finally $\sim t_0 = t_2$
Variations on also

Transitivity:

\begin{align*}
\text{have } & t_0 = t_1 \ldots \\
\text{also have } & \ldots = t_2 \ldots \\
\text{also/finally } & \leadsto t_0 = t_2
\end{align*}

Substitution:

\begin{align*}
\text{have } & P(s) \ldots \\
\text{also have } & s = t \ldots \\
\text{also/finally } & \leadsto
\end{align*}
Variations on also

Transitivity:

\( \text{have } t_0 = t_1 \ldots \)
\( \text{also have } \ldots = t_2 \ldots \)
\( \text{also/finally } \leadsto t_0 = t_2 \)

Substitution:

\( \text{have } P(s) \ldots \)
\( \text{also have } s = t \ldots \)
\( \text{also/finally } \leadsto P(t) \)
Transitivity:

have \( t_0 \leq t_1 \ldots \)

also have \( \ldots \leq t_2 \ldots \)

also/finally \( \sim \)
Transitivity:

- have $t_0 \leq t_1 \ldots$
- also have $\ldots \leq t_2 \ldots$
- also/finally $\sim t_0 \leq t_2$
Transitivity:

have $t_0 \leq t_1 \ldots$
also have $\ldots \leq t_2 \ldots$
also/finally $\sim \Rightarrow t_0 \leq t_2$

Substitution:

have $r \leq f(s) \ldots$
also have $s < t \ldots$
also/finally $\sim \Rightarrow$
Transitivity:

have $t_0 \leq t_1 \ldots$
also have $\ldots \leq t_2 \ldots$
also/finally $\sim t_0 \leq t_2$

Substitution:

have $r \leq f(s) \ldots$
also have $s < t \ldots$
also/finally $\sim (\bigwedge x. x < y \implies f(x) < f(y)) \implies r < f(t)$
From $=$ to $\leq$ and $<$

Transitivity:

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

Substitution:

have $r \leq f(s)$ . . .
also have $s < t$ . . .
also/finally $\leadsto (\land x. x < y \implies f(x) < f(y)) \implies r < f(t)$

Similar for all other combinations of $=$, $\leq$ and $<$. 
To view all combinations in Proof General:
Isabelle/Isar → Show me → Transitivity rules
Demo: monotonicity reasoning