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 \( f_{\text{ormula}_0} \)

have \( f_{\text{ormula}_1} \) by simp

: 

have \( f_{\text{ormula}_n} \) by blast

show \( f_{\text{ormula}_{n+1}} \) by \ldots 

qed
A typical Isar proof

proof
  assume \( \text{formula}_0 \)
  have \( \text{formula}_1 \) by simp
  
  
  have \( \text{formula}_n \) by blast
  show \( \text{formula}_{n+1} \) by \ldots
  qed

proves \( \text{formula}_0 \implies \text{formula}_{n+1} \)
Overview

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

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

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

\[\text{method} = (\text{simp} \ldots) \ | \ (\text{blast} \ldots) \ | \ (\text{rule} \ldots) \ | \ \ldots\]
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**

\[
\text{proof} = \begin{cases} 
\text{proof} \ [\text{method}] \ \text{statement}^* \ \text{qed} \\
\text{by} \ \text{method}
\end{cases}
\]

\[
\text{method} = (\text{simp} \ldots) \mid (\text{blast} \ldots) \mid (\text{rule} \ldots) \mid \ldots
\]

\[
\text{statement} = \begin{cases} 
\text{fix variables} \\
\text{assume proposition} \\
[\text{from name}^+] (\text{have} \mid \text{show}) \ \text{proposition} \ \text{proof} \\
\text{next}
\end{cases}
\]

\(\begin{array}{c}
\text{(separates subgoals)}
\end{array}\)
Isar core syntax

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

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

statement = fix variables (\wedge)
  | assume proposition (\Rightarrow)
  | [from name^+] (have | show) proposition proof
  | next (separates subgoals)

proposition = [name:] formula
Demo: propositional logic, introduction rules
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 \tilde{a}
apply a rule in \tilde{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 \( f_{ormula} \) \( p\)roof
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: 
  \textit{from } \vec{a}\textit{ have formula proof}
- proof alone abbreviates proof \textit{rule}
- \textit{rule}: tries elim rules first (if there are incoming facts } \vec{a}!\textit{)
Elimination rules / forward reasoning

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

- proof alone abbreviates proof rule

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

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

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

- \texttt{proof} alone abbreviates \texttt{proof rule}

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

- \texttt{from } \vec{a} \texttt{ have } formula \texttt{ proof (rule rule)}
  \vec{a} \texttt{ must prove the first } n \texttt{ premises of } rule,
Elimination rules / forward reasoning

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

• proof alone abbreviates proof rule

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

• \texttt{from } \vec{a} \texttt{ have } \textit{formula} \texttt{ proof (rule rule)}
  \vec{a} \texttt{ must prove the first } n \texttt{ premises of rule, in the right order}
Elimination rules / forward reasoning

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

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

\[=\]

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

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

\[= \]

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

Can be mixed:

\[\text{from major-facts (have|show) proposition } \text{using} \text{ minor-facts} \]
using

First the what, then the how:

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

Can be mixed:

\[\text{from major-facts (have}|\text{show}) \text{ proposition using minor-facts} = \text{from major-facts minor-facts (have}|\text{show}) \text{ proposition}\]
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: ?thesis
  The last enclosing show formula
Demo: predicate calculus
obtain

Syntax:

\texttt{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 \( f_{\text{ormula}} \)
  :
next
  assume \( \neg f_{\text{ormula}} \)
  :
qed
**Boolean case distinction**

\[
\text{proof cases} \\
\quad \text{assume } formula \\
\quad \vdash \\
\quad \text{next} \\
\quad \text{assume } \neg formula \\
\quad \vdash \\
\text{qed}
\]

\[
\text{proof } \left( \text{cases } formula \right) \\
\quad \text{case } True \\
\quad \vdash \\
\quad \text{next} \\
\quad \text{case } False \\
\quad \vdash \\
\text{qed}
\]
**Boolean case distinction**

\[\begin{align*}
\text{proof cases} & \quad \text{proof (cases } \text{formula}) \\
\text{assume } \text{formula} & \quad \text{case } \text{True} \\
\vdots & \quad \vdots \\
\text{next} & \quad \text{next} \\
\text{assume } \neg \text{formula} & \quad \text{case } \text{False} \\
\vdots & \quad \vdots \\
\text{qed} & \quad \text{qed} \\
\end{align*}\]

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

proof \((cases \ term)\)
  
  case \(Constructor_1\)
    
    : 

  next

  : 

  next

  case \((Constructor_k \ \vec{x})\)
    
    \(\ldots \ \vec{x} \ \ldots\)

qed
Datatype case distinction

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

\[
\begin{aligned}
\text{case } \text{Constructor}_1
\end{aligned}
\]

\[
\begin{aligned}
\vdots
\end{aligned}
\]

next

\[
\begin{aligned}
\vdots
\end{aligned}
\]

next

\[
\begin{aligned}
\text{case } (\text{Constructor}_k \ \vec{x})
\end{aligned}
\]

\[
\begin{aligned}
\vdots \ \vec{x} \ \vdots
\end{aligned}
\]

qed

\[
\begin{aligned}
\text{case } (\text{Constructor}_i \ \vec{x}) \quad \equiv
\end{aligned}
\]

fix \( \vec{x} \) assume \( \text{Constructor}_i : \ \text{term} = (\text{Constructor}_i \ \vec{x}) \)
Induction
Overview

- Structural induction
- Rule induction
- Induction with fun
show $P(n)$
proof (induction $n$)
  case 0
    ...
    ...
    show ?case
next
  case $(Suc\ n)$
    ...
    ...
    $n$ ...
    show ?case
qed
Structural induction for type nat

show $P(n)$
proof (induction $n$)
  case 0
  \[ \equiv \text{ let } ?\text{case} = P(0) \]
  \[ \cdots \]
  show ?case
next
  case (Suc $n$)
  \[ \cdots \]
  \[ \cdots n \cdots \]
  show ?case
qed
Structural induction for type nat

show $P(n)$
proof (induction $n$)
  case 0
  ...  
  show $?case$
next
  case (Suc $n$)
  ...  
  ... $n$ ...
  show $?case$
qed
Demo: structural induction
Structural induction with $\implies$ and $\land$

show $\land x. A(n) \implies P(n)$
proof (induction $n$)
  case 0
    ...
    ...
    show $?case$
next
  case (Suc n)
    ...
    $n$ ...
    ...
    show $?case$
qed
Structural induction with \( \Rightarrow \) and \( \wedge \)

\[ \forall x. A(n) \Rightarrow P(n) \]

proof (induction \( n \))

\[ \text{case } 0 \]
\[ \quad \ldots \]
\[ \quad \text{show } \text{?case} \]
\[ \text{next} \]
\[ \quad \text{case } (\text{Suc } n) \]
\[ \quad \ldots \]
\[ \quad \ldots n \ldots \]
\[ \quad \ldots \]
\[ \quad \text{show } \text{?case} \]
\[ \text{qed} \]

\[ \equiv \quad \text{fix } x \quad \text{assume } 0: A(0) \]
\[ \quad \text{let } \text{?case } = P(0) \]
Structural induction with $\Rightarrow$ and $\wedge$

show $\forall x. A(n) \Rightarrow P(n)$

proof (induction n)

  case 0
  
  ... 

  show $?case$

next

  case (Suc n)
  
  ... n ...

  ... 

  show $?case$

qed

case 0

= fix x assume 0: A(0)
let $?case = P(0)$

case (Suc n)

= fix n x
assume Suc: $\forall x. A(n) \Rightarrow P(n)$

\[ A(Suc n) \]

let $?case = P(Suc n)$
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 $\Rightarrow$ and $\land$
Rule induction
**Inductive definition**

inductive_set $S$

intros

$\text{rule}_{1}: [ s \in S; A ] \implies s' \in S$

: 

$\text{rule}_{n}: \ldots$
Rule induction

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

proof \((\text{induct rule: } S.\text{induct})\)

\begin{align*}
\text{case } & rule_1 \\
\ldots
\end{align*}

\begin{align*}
\text{show } & \ ?case \\
\text{next}
\end{align*}

\begin{align*}
\vdots
\text{next}
\end{align*}

\begin{align*}
\text{case } & rule_n \\
\ldots
\end{align*}

\begin{align*}
\text{show } & \ ?case \\
\text{qed}
\end{align*}
Implicit selection of induction rule

assume $A : x \in S$

::

show $P(x)$
using $A$ proof $induct$

::

qed
Implicit selection of induction rule

assume $A : x \in S$

using $A$ proof $induct$

show $P(x)$

using $A$ proof $induct$

qed

lemma assumes $A : x \in S$ shows $P(x)$

using $A$ proof $induct$

qed
Renaming free variables in rule

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

Renames the (alphabetically!) first \( k \) variables in \( rule_i \) to \( x_1 \ldots x_k \).
Demo: rule induction


Definition:

\[ \text{fun } f \]

\[ : \]
**Induction with fun**

Definition:

```plaintext
fun f
```

Proof:

```plaintext
show ... f(...) ...
proof (induction x₁ ... xₖ rule: f.induct)
```
Induction with fun

Definition:
fun f
:

Proof:
show ... f(...) ...
proof (induction x_1 ... x_k rule: f.induct)
  case 1
  :
  :
**Induction with fun**

Definition:

```plaintext
fun f
:
```

Proof:

```plaintext
show ... f(...) ...
proof (induction x₁ ... xₖ rule: f.induct)
  case 1
    :
```

Case $i$ refers to equation $i$ in the definition of $f$
**Induction with fun**

Definition:

```
fun f

: 
```

Proof:

```
show ... f(...)

proof (induction x_1 ... 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 fun
Calculational Reasoning
Overview

• Accumulating facts
• Chains of equations and inequations
moreover

have \( formula_1 \ldots \)
moreover
have \( formula_2 \ldots \)
moreover
\vdots
moreover
have \( formula_n \ldots \)
ultimately show \ldots
— pipes facts \( formula_1 \ldots formula_n \) into the proof
proof
\vdots
also

have \( t_0 = t_1 \ldots \).

also

have \ldots = t_2 \ldots .

also

\vdots

also

have \ldots = t_n \ldots .
also

\[ t_0 = t_1 \ldots \]

also

\[ \ldots = t_2 \ldots \]

\[ \ldots \equiv t_1 \]

also

\[ \vdots \]

also

\[ \ldots = t_n \ldots \]
also

have $t_0 = t_1$ . . .
also
have . . . = $t_2$ . . .
also
:
also
have . . . = $t_n$ . . .

also

\[ \ldots \equiv t_1 \]
\[ \ldots \equiv t_{n-1} \]
also

have \( t_0 = t_1 \) \ldots \\
also \\

have \( \ldots = t_2 \) \ldots \ldots \ldots \equiv t_1 \\
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 \( \sim \)
Variations on also

Transitivity:

**have** $t_0 = t_1 \ldots$

**also have** $\ldots = t_2 \ldots$

**also/finally** $\sim t_0 = t_2$
Variations on also

Transitivity:

have \( t_0 = t_1 \ldots \)
also have \( \ldots = t_2 \ldots \)
also/finally \( \sim \) \( t_0 = t_2 \)

Substitution:

\[ \text{have } P(s) \ldots \]
also have \( s = t \ldots \)
also/finally \( \sim \)
Variations on also

Transitivity:

have \( t_0 = t_1 \) \ldots
also have \( \ldots = t_2 \) \ldots
also/finally \( \rightsquigarrow t_0 = t_2 \)

Substitution:

have \( P(s) \) \ldots
also have \( s = t \) \ldots
also/finally \( \rightsquigarrow P(t) \)
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 \( \sim \) \( t_0 \leq t_2 \)

Substitution:

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

- have \( t_0 \leq t_1 \) . . . .
- also have . . . \( \leq t_2 \) . . . .
- also/finally \( \sim \) \( t_0 \leq t_2 \)

Substitution:

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

\textbf{From} = \textbf{to} \leq \textbf{and} <
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** $\leadsto (\forall x. x < y \implies f(x) < f(y)) \implies r < f(t)$

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