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 \( \text{formula}_0 \)
  have \( \text{formula}_1 \) by simp
  
  have \( \text{formula}_n \) by blast
  show \( \text{formula}_{n+1} \) 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
- Isar by example
- Proof patterns
- Streamlining proofs
Isar core syntax

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

\[
\text{proof} \ = \ \text{proof} \ [\text{method}] \ \text{statement}^* \ \text{qed} \\
\quad | \quad \text{by} \ \text{method} \\
\text{method} \ = \ (\text{simp} \ldots) \ | \ (\text{blast} \ldots) \ | \ (\text{rule} \ldots) \ | \ldots
\]
**Isar core syntax**

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

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

statement = fix variables (∧)
   | assume prop (⇒)
   | [from fact+] (have | show) prop proof
Isar core syntax

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

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

statement  =  fix variables          (∧)
          |  assume prop             (⇒)
          |  [from fact+] (have | show) prop proof
          |  next (separates subgoals)
Isar core syntax

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

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

statement = fix variables (\land)
   | assume prop (\Rightarrow)
   | [from fact^+] (have | show) prop proof
   | next (separates subgoals)

prop = [name:] "formula"
Isar core syntax

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

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

statement = fix variables (∧)
    | assume prop (⇒)
    | [from fact+] (have | show) prop proof
    | next (separates subgoals)

prop = [name:] "formula"

fact = name | name[OF fact+] | ‘formula‘
Isar by example
Example: Cantor’s theorem

Lemma Cantor: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)
Example: Cantor’s theorem

lemma Cantor: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)

proof
Example: Cantor’s theorem

lemma Cantor: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)
proof assume surj, show False
Example: Cantor’s theorem

lemma Cantor: \( \neg \text{surj}(f : \forall a \rightarrow a \rightarrow \text{set}) \)

proof
  assume surj, show False
  assume a: surj f
Example: Cantor’s theorem

lemma Cantor: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)
proof  
  assume surj, show False
  assume a: surj f
  from a have b: \( \forall A. \exists a. A = f a \)
Example: Cantor’s theorem

lemma Cantor: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)

proof  
  assume surj, show False  
  assume a: surj f  
  from a have b: \( \forall A. \exists a. A = f a \)  
  by (simp add: surj_def)
Example: Cantor’s theorem

lemma Cantor: \( \neg \text{surj}(f :: \text{'}a \Rightarrow \text{'}a \text{ set}) \)

proof  assume \( \text{surj} \), show False
  assume \( a: \text{surj } f \)
  from \( a \) have \( b: \forall A. \exists a. A = f a \)
    by (simp add: surj_def)
  from \( b \) have \( c: \exists a. \{x. x \notin f x\} = f a \)
Example: Cantor’s theorem

lemma Cantor: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)
proof  assume surj, show False
  assume a: surj f
  from a have b: \( \forall A. \exists a. A = f a \)
    by (simp add: surj_def)
  from b have c: \( \exists a. \{x. x \notin f x\} = f a \)
    by blast
Example: Cantor’s theorem

lemma Cantor: ¬ \text{surj}(f :: 'a ⇒ 'a set)
proof  assume \text{surj}, show False
  assume a: \text{surj} f
  from a have b: \forall A. \exists a. A = f a
    by (simp add: \text{surj\_def})
  from b have c: \exists a. \{x. x \notin f x\} = f a
    by blast
  from c show False
Example: Cantor’s theorem

lemma Cantor: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)

proof  assume \( \text{surj} \), show False

  assume \( a: \text{surj} \ f \)
  from \( a \) have \( b: \forall A. \exists a. A = f a \)
    by (simp add: surj_def)
  from \( b \) have \( c: \exists a. \{x. x \notin f x\} = f a \)
    by blast
  from \( c \) show False
    by blast
Example: Cantor’s theorem

lemma Cantor: \( \neg \text{surj}(f :: \text{'a} \Rightarrow \text{'a set}) \)
proof  
  assume \( \text{surj} \), show \( \text{False} \)
  assume \( a: \text{surj} f \)
  from \( a \) have \( b: \forall A. \exists a. A = f a \)
    by (simp add: surj_def)
  from \( b \) have \( c: \exists a. \{x. x \notin f x\} = f a \)
    by blast
  from \( c \) show \( \text{False} \)
    by blast
qed
Demo: this, then etc
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>this</em></td>
<td>the previous proposition proved or assumed</td>
</tr>
<tr>
<td><em>then</em></td>
<td>from <em>this</em></td>
</tr>
<tr>
<td><em>thus</em></td>
<td>then show</td>
</tr>
<tr>
<td><em>hence</em></td>
<td>then have</td>
</tr>
</tbody>
</table>
First the what, then the how:

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

\[(\text{have}|\text{show}) \text{ prop using facts} = \text{from facts (have}|\text{show}) \text{ prop}\]
Example: Structured lemma statement

lemma \textit{Cantor'}:
\begin{itemize}
\item fixes $f :: 'a \Rightarrow 'a \text{ set}$
\item assumes $s :: \text{ surj } f$
\item shows $\text{ False}$
\end{itemize}
Example: Structured lemma statement

lemma Cantor':
  \( \text{fixes } f :: \ 'a \Rightarrow \ 'a \text{ set} \)
  \( \text{assumes } s: \text{surj } f \)
  \( \text{shows } \text{False} \)
proof -
Example: Structured lemma statement

lemma \textit{Cantor'}:
  \begin{itemize}
  \item \textbf{fixes} \( f :: 'a \Rightarrow 'a \text{ set} \)
  \item \textbf{assumes} \( s : \text{surj } f \)
  \item \textbf{shows} \( \text{False} \)
  \end{itemize}
proof - no automatic proof step
Example: Structured lemma statement

lemma Cantor':
  fixes f :: 'a ⇒ 'a set
  assumes s: surj f
  shows False
proof -  no automatic proof step
  have ∃ a. {x. x ∉ f x} = f a using s
    by(auto simp: surj_def)
Example: Structured lemma statement

lemma \textit{Cantor'}:
  
  \textbf{fixes} \( f \colon 'a \Rightarrow 'a \text{ set} \)
  
  \textbf{assumes} \( s \colon \text{surj } f \)
  
  \textbf{shows} False

\textbf{proof} - no automatic proof step

\textbf{have} \( \exists a. \{x. x \notin f x\} = f a \) \text{ using } \( s \)

\textbf{by} (auto simp: surj_def)

\textbf{thus} False \textbf{by} blast

qed
Example: Structured lemma statement

lemma Cantor':
  fixes f :: 'a ⇒ 'a set
  assumes s: surj f
  shows False
proof -  no automatic proof step
  have ∃ a. {x. x /∈ f x} = f a using s
    by (auto simp: surj_def)
  thus False by blast
qed

  Proves surj f ⇒ False
Example: Structured lemma statement

lemma \textit{Cantor'}:
- \textbf{fixes} \( f :: 'a \Rightarrow 'a \text{ set} \)
- \textbf{assumes} \( s: \text{surj} f \)
- \textbf{shows} \( \text{False} \)

proof - no automatic proof step
- \textbf{have} \( \exists a. \{x. x \notin f x\} = f a \) using \( s \)
  - \textbf{by} (auto simp: surj_def)
- \textbf{thus} \( \text{False} \) \textbf{by blast}

qed

Proves \( \text{surj} f \implies \text{False} \)
but \( \text{surj} f \) becomes local fact \( s \) in proof.
Assumptions and intermediate facts can be named and referred to explicitly and selectively
Structured lemma statements

fixes \( x :: \tau_1 \) and \( y :: \tau_2 \) ... 
assumes a: \( P \) and b: \( Q \) ... 
shows \( R \)
Structured lemma statements

fixes $x :: \tau_1$ and $y :: \tau_2$ . . .
assumes a: $P$ and b: $Q$ . . .
shows $R$

• fixes and assumes sections optional
Structured lemma statements

fixes \( x :: \tau_1 \) and \( y :: \tau_2 \) . . .  
assumes a: \( P \) and b: \( Q \) . . .  
shows \( R \)

• fixes and assumes sections optional
• shows optional if no fixes and assumes
Proof patterns
show $P \iff Q$

proof
  assume $P$
  
  \[ \vdots \]
  show $Q$ . . .

next
  assume $Q$
  
  \[ \vdots \]
  show $P$ . . .

qed
Propositional proof patterns

show $P \leftrightarrow Q$
proof
  assume $P$
  :
  show $Q$ . . .
next
  assume $Q$
  :
  show $P$ . . .
qed

show $A = B$
proof
  show $A \subseteq B$ . . .
next
  show $B \subseteq A$ . . .
qed
Propositional proof patterns

show $P \iff Q$
proof
  assume $P$
  :  
  show $Q$  ... 
next
  assume $Q$
  :  
  show $P$  ... 
qed

show $A = B$
proof
  show $A \subseteq B$  ... 
next
  show $B \subseteq A$  ... 
qed

show $A \subseteq B$
proof
  fix $x$
  assume $x \in A$
  :  
  show $x \in B$  ... 
qed
Propositional proof patterns

show $R$
proof cases
  assume $P$
  ...
  show $R$ ...
next
  assume $\neg P$
  ...
  show $R$ ...
qed

Case distinction
**Propositional proof patterns**

- show $R$
  - proof cases
    - assume $P$
      - show $R$ ...
  - next
    - assume $\neg P$
      - show $R$ ...
- qed

- have $P \lor Q$ ...
  - then show $R$
    - proof
      - assume $P$
        - show $R$ ...
  - next
    - assume $Q$
      - show $R$ ...
- qed

Case distinction  Case distinction
Propositional proof patterns

show $R$
proof cases
  assume $P$
  :
  show $R$ ...
next
  assume $\neg P$
  :
  show $R$ ...
qed

have $P \lor Q$ ...
then show $R$
proof
  assume $P$
  :
  show $R$ ...
next
  assume $Q$
  :
  show $R$ ...
qed

show $P$
proof (rule ccontr)
  assume $\neg P$
  :
  show False ...
qed

Case distinction  Case distinction  Contradiction
Quantifier introduction proof patterns

show $\forall x. \ P(x)$
proof
  fix $x$  {local fixed variable}
  show $P(x)$  
qed
Quantifier introduction proof patterns

show $\forall x. P(x)$
proof
  fix $x$ \textit{local fixed variable}
  show $P(x)$ \ldots
qed

show $\exists x. P(x)$
proof
  : 
  : 
  show $P(\text{witness})$ \ldots
qed
∃ elimination: obtain
\exists \textit{elimination: obtain}

have $\exists x. P(x)$
then \textit{obtain} $x$ where $p: P(x)$ by blast

\vdash x \text{ local fixed variable}
∃ elimination: obtain

have ∃x. P(x)
then obtain x where p: P(x) by blast

∵ x local fixed variable

Works for one or more x
obtain example

lemma *Cantor”*: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)
proof
  assume \( \text{surj } f \)
  hence \( \exists a. \{x. x \notin f x\} = f a \) by (auto simp: surj_def)
lemma *Cantor’’*: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)

proof
  assume \( \text{surj} \ f \)
  hence \( \exists \ a. \{x. \ x \notin f \ x\} = f \ a \) \ by (auto simp: surj_def)
  then obtain \( a \) where \( \{x. \ x \notin f \ x\} = f \ a \) \ by blast
lemma *Cantor”*: \( \neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)

proof
  assume *surj* \( f \)
  hence \( \exists \ a. \{x. \ x \notin f \ x\} = f \ a \) by(auto simp: surj_def)
  then obtain \( a \) where \( \{x. \ x \notin f \ x\} = f \ a \) by blast
  hence \( a \notin f \ a \leftrightarrow a \in f \ a \) by blast
lemma \(\text{Cantor}''\): \(\neg \text{surj}(f :: 'a \Rightarrow 'a \text{ set})\)

proof

assume \(\text{surj } f\)

hence \(\exists a. \{x. x \notin f x\} = f a\) by (auto simp: surj_def)

then obtain \(a\) where \(\{x. x \notin f x\} = f a\) by blast

hence \(a \notin f a \iff a \in f a\) by blast

thus \(\text{False}\) by blast

qed
proof method
Applies method and generates subgoal(s):

1. $\forall x_1 \ldots x_n \left[ A_1; \ldots ; A_m \right] \implies A$
Applies method and generates subgoal(s):

1. $\bigwedge x_1 \ldots x_n \left[ A_1; \ldots ; A_m \right] \Rightarrow A$

How to prove each subgoal:
Applies method and generates subgoal(s):

1. $\bigwedge x_1 \ldots x_n \left[ A_1; \ldots ; A_m \right] \implies A$

How to prove each subgoal:

- **fix** $x_1 \ldots x_n$
- **assume** $A_1 \ldots A_m$
- : 
- **show** $A$
proof method

Applies method and generates subgoal(s):

1. \( \bigwedge x_1 \ldots x_n [A_1; \ldots ; A_m] \Rightarrow A \)

How to prove each subgoal:

fix \( x_1 \ldots x_n \)
assume \( A_1 \ldots A_m \)
::
show A

Separated by next
Demo: proof
Streamlining proofs: Pattern matching and Quotations
Example: pattern matching

\[ \text{show } formula_1 \leftrightarrow formula_2 \quad (\text{is } ?L \leftrightarrow ?R) \]
Example: pattern matching

show $formula_1 \leftrightarrow formula_2$ (is $?L \leftrightarrow ?R$)

proof
  assume $?L$
  :
  show $?R$ . . .

next
  assume $?R$
  :
  show $?L$ . . .

qed
show $formula$
proof -

:\

show $?thesis \ldots$
qed
show \textit{formula} (is \textit{thesis})
proof -
:  
  show \textit{thesis} ... 
qed
show $formula$ (is $thesis$)
proof -
  :
  show $thesis$ . . .
qed

Every show implicitly defines $thesis$
Quoting facts by value

By name:

```
have x0: "x > 0" . . .
```
```
from x0 . . .
```
Quoting facts by value

By name:

\[
\text{have } x_0: "x > 0" \ldots \\
\vdots \\
\text{from } x_0 \ldots 
\]

By value:

\[
\text{have } "x > 0" \ldots \\
\vdots \\
\text{from } x > 0 \ldots 
\]
Quoting facts by value

By name:

```plaintext
have x0: "x > 0" . . .
:
from x0 . . .
```

By value:

```plaintext
have "x > 0" . . .
:
from 'x>0' . . .
back quotes
```
Demo: pattern matching and quotations
Advanced Isar
Overview

- Case distinction
- Induction
- Chains of (in)equations
Case distinction
Demo: case distinction
Datatype case distinction

datatype $t = C_1 \bar{\tau} \mid \ldots$
Datatype case distinction

datatype \( t = C_1 \vec{x} \mid \ldots \)

proof (cases \textit{term})

\textbf{case} (\textit{C}_1 \vec{x})

\ldots \\vec{x} \ldots

next

: :

qed
Datatype case distinction

datatype \( t = C_1 \vec{x} | \ldots \)

proof (cases term)
    case \((C_1 \vec{x})\)
        \[\ldots \vec{x} \ldots\]
    next
    ::
    qed

where case \((C_i \vec{x})\) \(\equiv\)

fix \(\vec{x}\)

assume \(\begin{cases}
    \text{label} & C_i : \\
    \text{formula} & \text{term} = (C_i \vec{x})
\end{cases}\)
Induction
Overview

• Structural induction
• Rule induction
• Induction with fun
Structural induction for type nat

show \( P(n) \)
proof (induct \( n \))
  case 0
  ...
  show ?case
next
  case (Suc \( n \))
  ...
  ... \( n \) ...
  show ?case
qed
Structural induction for type nat

show $P(n)$
proof (induct $n$)
  case 0
    let ?case = $P(0)$
  ...
  show ?case
next
  case (Suc $n$)
  ...
  ... $n$ ...
  show ?case
qed
Structural induction for type nat

show $P(n)$

proof (induct $n$)

case 0

\[ \equiv \text{let } ?\text{case} = P(0) \]

\[ \ldots \]

show ?case

next

case (Suc $n$)

\[ \equiv \text{fix } n \text{ assume Suc: } P(n) \]

let ?case = $P(Suc \ n)$

\[ \ldots \]

\[ \ldots n \ldots \]

show ?case

qed
Demo: structural induction
show $A(n) \implies P(n)$

proof (induct $n$)
  case $0$
    \ldots
  \ldots
  show ?case

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

qed
Structural induction with $\implies$

show $A(n) \implies P(n)$

proof $(induct \ n)$
  case 0
    ...  
  show $?case$

next
  case $(Suc \ n)$
    ...  
    ... $n$ ...
    ...
  show $?case$

qed
**Structural induction with**

show $A(n) \Rightarrow P(n)$

proof (*induct* $n$)

  case 0
  
  \[
  \begin{align*}
  \text{fix } x \\
  \text{assume } 0: A(0) \\
  \text{let } ?\text{case} = P(0)
  \end{align*}
  \]

  show ?case

next

  case $(\text{Suc } n)$
  
  \[
  \begin{align*}
  \text{fix } n \\
  \text{assume } \text{Suc}: A(n) \Rightarrow P(n) \\
  A(\text{Suc } n) \\
  \text{let } ?\text{case} = P(\text{Suc } n)
  \end{align*}
  \]

  show ?case

qed
A remark on style

• \texttt{case \ (Suc\ n) \ldots show \ ?case}
  is easy to write and maintain
A remark on style

- **case** \((Suc \ n) \ldots \ show \ ?case\)** is easy to write and maintain
- **fix \ n \ assume \ formula \ldots \ show \ formula'\)** is easier to read:
  - all information is shown locally
  - no contextual references (e.g. \(?case\))
Demo: structural induction with
Rule induction
Inductive definition

inductive_set $S$

intros

$rule_1: \left[ s \in S; A \right] \implies s' \in S$

$\vdots$

$rule_n: \ldots$
Rule induction

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

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

\[
\begin{align*}
\text{case } & \text{rule}_1 \\
\ldots & \\
\text{show } & ?\text{case} \\
\text{next} & \\
\vdots & \\
\text{next} & \\
\text{case } & \text{rule}_n \\
\ldots & \\
\text{show } & ?\text{case} \\
\text{qed}
\end{align*}
\]
assume $A: x \in S$

::

show $P(x)$

using $A$ proof induct

::

qed
Implicit selection of induction rule

assume $A: x \in S$  

::

show $P(x)$  

using $A$ proof $induct$

::

qed

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

using $A$ proof $induct$

::

qed
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:

fun f

::
Induction with fun

Definition:
fun f
  : 

Proof:
show ... f(...) ...
proof (induct x₁ ... xₖ rule: f.induct)
**Induction with fun**

Definition:

```
fun f 
:
```

Proof:

```
show ... f(...) ...
proof (induct x₁ ... xₖ rule: f.induct)
  case 1
  :
```
\textbf{Induction with fun}

Definition:
\begin{verbatim}
fun f
:

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

Case \(i\) refers to equation \(i\) in the definition of \(f\)
**Induction with fun**

Definition:

```haskell
fun f
```

Proof:

```haskell
show ... f(...) ...
proof (induct x₁ ... xₖ, 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
Chains of (in)equations
also

have  "\( t_0 = t_1 \)"  ...
also

have  "\( t_0 = t_1 \)"  . . .
also
have  " . . . = t_2"  . . .
also

have "\( t_0 = t_1 \)" . . .
also
have "\( \ldots = t_2 \)" . . . \( \ldots \equiv t_1 \)
also

have   
  \[ t_0 = t_1 \]   

also

have   
  \[ \ldots = t_2 \]   

also

\[ \vdots \]

also

have   
  \[ \ldots = t_n \]   

\[=t_1\]
also

have \( t_0 = t_1 \) \ldots
also
have \( \ldots = t_2 \) \ldots \quad \ldots \equiv t_1
also
\vdots
also
have \( \ldots = t_n \) \ldots \quad \ldots \equiv t_{n-1}
also

have \( t_0 = t_1 \) \ldots

also

have \( \ldots = t_2 \) \ldots \quad \ldots \equiv t_1

also

\vdots

also

have \( \ldots = t_n \) \ldots \quad \ldots \equiv t_{n-1}

finally show \ldots
also

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

have "\( \ldots = t_2 \)" \ldots \ldots \equiv t_1
also

\vdots
also

have "\( \ldots = t_n \)" \ldots \ldots \equiv t_{n-1}
finally show \ldots

— like from \( t_0 = t_n \) show
also

- “…” is merely an abbreviation
• “…” is merely an abbreviation
• also works for other transitive relations ($<$, $\le$, …)
Demo: also
Accumulating facts
moreover

have $\mathit{formula}_1 \ldots$
moreover

have $\text{formula}_1 \ldots$
moreover
have $\text{formula}_2 \ldots$
moreover

have $formula_1 \ldots$
moreover
have $formula_2 \ldots$
moreover
::
moreover
have $formula_n \ldots$
moreover

have $formula_1$ . . .
moreover
have $formula_2$ . . .
moreover

::
moreover
have $formula_n$ . . .
ultimately show . . .
moreover

have \( \text{formula}_1 \ldots \)
moreover
have \( \text{formula}_2 \ldots \)
moreover

\vdots

moreover
have \( \text{formula}_n \ldots \)
ultimately show \ldots

— like from \( f_1 \ldots f_n \) show but needs no labels
Demo: moreover