Automated Reasoning

Lecture 9: Isar – A Language for Structured Proofs

Jacques Fleuriot
jdf@inf.ed.ac.uk

Acknowledgement: Tobias Nipkow kindly provided the slides for this lecture
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
proof
  \textbf{assume} \ formula_0
  \textbf{have} \ formula_1 \quad \textbf{by simp}
  \quad \quad \vdots
  \quad \textbf{have} \ formula_n \quad \textbf{by blast}
  \textbf{show} \ formula_{n+1} \quad \textbf{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 ...

qed

proves $formula_0 \implies formula_{n+1}$
Isar core syntax

proof = proof [method] step* qed
| by method
Isar core syntax

\[\text{proof} \equiv \begin{align*} &\text{proof } \text{[method]} \ \text{step}^* \ \text{qed} \\ &\text{by method} \end{align*} \]

\[\text{method} \equiv (\text{simp } \cdots) \mid (\text{blast } \cdots) \mid (\text{induction } \cdots) \mid \ldots \]
Isar core syntax

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

\[
\text{method} = (\text{simp . . .}) | (\text{blast . . .}) | (\text{induction . . .}) | ... 
\]

\[
\text{step} = \text{fix variables} \quad (\land) \\
| \quad \text{assume prop} \quad (\equiv)
| \quad [\text{from fact}^+] \quad (\text{have} | \text{show}) \quad \text{prop} \quad \text{proof}
\]
Isar core syntax

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

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

step = fix variables (\lor)
    | assume prop (\iff)
    | [from fact+] (have | show) prop proof

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

\[
\text{proof} = \begin{array}{ll}
\text{proof} & \text{method} \quad \text{step} \quad \text{qed} \\
\mid & \text{by} \quad \text{method}
\end{array}
\]

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

\[
\text{step} = \begin{array}{ll}
\text{fix} & \text{variables} \\
\mid & (\land)
\end{array}
\begin{array}{ll}
\text{assume} & \text{prop} \\
\mid & (\equiv)
\end{array}
\begin{array}{ll}
[\text{from} & \text{fact}^+] \\
\mid & (\text{have} \mid \text{show}) \quad \text{prop} \quad \text{proof}
\end{array}
\]

\[
\text{prop} = \begin{array}{ll}
[\text{name:}] & \text{"formula"}
\end{array}
\]

\[
\text{fact} = \begin{array}{ll}
\text{name} \mid \ldots
\end{array}
\]
Example: Cantor’s theorem

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

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

\text{proof}
Example: Cantor’s theorem

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

\begin{verbatim}
lemma surj(f :: 'a \Rightarrow 'a set)
proof  default proof: assume surj, show False
  assume a: surj f
\end{verbatim}
Example: Cantor’s theorem

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

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

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

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

proof  
  default 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
Example: Cantor’s theorem

lemma ¬ surj(f :: ’a ⇒ ’a set)
proof default proof: assume surj, show False
  assume a: surj f
  from a have b: ∀ A. ∃ a. A = f a
    by(simp add: surj_def)
  from b have c: ∃ a. {x. x ∉ f x} = f a
    by blast
Example: Cantor’s theorem

lemma ¬ surj(f :: 'a ⇒ 'a set)
proof default proof: assume surj, show False
  assume a: surj f
from a have b: ∀ A. ∃ a. A = f a
  by(simp add: surj_def)
from b have c: ∃ a. {x. x ∉ f x} = f a
  by blast
from c show False
Example: Cantor’s theorem

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

proof  default 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 \( \neg \text{surj}(f :: \ 'a \Rightarrow \ 'a \text{ set}) \)

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

assume \( a: \text{surj } f \)

from \( a \) have \( b: \forall A. \exists a. A = f a \)

by\((\text{simp add: surj_def})\)

from \( b \) have \( c: \exists a. \{x. x \notin f x\} = f a \)

by blast

from \( c \) show \( \text{False} \)

by blast

qed
Abbreviations

\[\text{this} = \text{the previous proposition proved or assumed}\]
\[\text{then} = \text{from this}\]
\[\text{thus} = \text{then show}\]
\[\text{hence} = \text{then have}\]
using and with

(have\|show) \text{ prop using facts}
using and with

\[(\text{have}|\text{show}) \text{ prop using facts} \]
\[= \]
\[\text{from facts (have}|\text{show}) \text{ prop} \]
using and with

(\texttt{have}|\texttt{show}) \text{ prop using facts} \\
= \\
\texttt{from facts (have}|\texttt{show}) \text{ prop} \\
\text{} \\
\text{} \\
\textbf{with facts} \\
= \\
\texttt{from facts this}
lemma
  fixes \( f :: \text{"}a \Rightarrow \text{'}a \text{ set}" \)
  assumes \( s : \text{"}surj f" \)
  shows \( \text{"}False" \)
lemma
  fixes f :: "'a ⇒ 'a set"
  assumes s: "surj f"
  shows "False"
proof -
Structured lemma statement

lemma
  fixes $f :: "'a \Rightarrow 'a set"$
  assumes $s: \text{"surj } f\"$
  shows \text{"False"}
proof - no automatic proof step
lemma
  fixes $f :: "'a \Rightarrow 'a set"$
  assumes $s: "\text{surj } f"$
  shows "False"
proof -
  no automatic proof step
  have "\exists a. \{x. x \notin f \cdot x\} = f \cdot a" using $s$
  by (auto simp: surj_def)
lemma
  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
lemma
  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
Structured lemma statement

lemma  
  fixes $f :: \text{"a} \Rightarrow \text{'}a \text{ set}\)  
  assumes $s: \text{"surj} f$  
  shows $\text{"False}$  
proof -  
  no automatic proof step  
  have $\exists \ a. \ {x. \ x \notin f \ x} = f \ a$ using $s$  
    by (auto simp: surj_def)  
  thus $\text{"False}$ by blast  
qed  

Proves $\text{surj} f \Rightarrow \text{False}$  
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

\textbf{fixes} \( x : \tau_1 \ \text{and} \ y : \tau_2 \ \ldots \)
\textbf{assumes} \( a : P \ \text{and} \ b : Q \ \ldots \)
\textbf{shows} \( R \)
Structured lemma statements

\textbf{fixes} \ x :: \ \tau_1 \ \textbf{and} \ y :: \ \tau_2 \ \ldots
\textbf{assumes} \ a :: \ P \ \textbf{and} \ b :: \ Q \ \ldots
\textbf{shows} \ R

- \textbf{fixes} and \textbf{assumes} sections optional
Structured lemma statements

\textbf{fixes} \ x :: \ \tau_1 \ \textbf{and} \ y :: \ \tau_2 \ \ldots
\textbf{assumes} \ a: \ P \ \textbf{and} \ b: \ Q \ \ldots
\textbf{shows} \ R

\begin{itemize}
  \item \textbf{fixes} and \textbf{assumes} sections optional
  \item \textbf{shows} optional if no \textbf{fixes} and \textbf{assumes}
\end{itemize}
Proof patterns: Case distinction

show "R"
proof cases
  assume "P"
  :
  show "R" ...
next
  assume "¬ P"
  :
  show "R" ...
qed
Proof patterns: Case distinction

show "R"

proof cases
  assume "P"
  :
  show "R" ...
next
  assume "¬ P"
  :
  show "R" ...
qed

have "P ∨ Q" ...
then show "R"
proof
  assume "P"
  :
  show "R" ...
next
  assume "Q"
  :
  show "R" ...
qed
Proof patterns: Contradiction

show \( \neg P \)
proof
  assume \( P \)
  :
  show \( False \)  
qed
Proof patterns: Contradiction

show \( \neg P \)
proof
  assume \( P \)
  
  show \( \text{False} \) \ldots
qed

show \( P \)
proof (rule ccontr)
  assume \( \neg P \)
  
  show \( \text{False} \) \ldots
qed
Proof patterns:

show "P 🜀 Q"

proof
  assume "P"
  
  show "Q" ...

next
  assume "Q"
  
  show "P" ...

qed
Proof patterns: $\forall$ and $\exists$ introduction

show “$\forall x. P(x)$”
proof
  fix $x$  local fixed variable
  show “$P(x)$”  
Qed
Proof patterns: $\forall$ and $\exists$ introduction

\begin{verbatim}
show "\(\forall x. P(x)\)"
proof
  fix \(x\)  local fixed variable
  show "\(P(x)\)" . . .
qed

show "\(\exists x. P(x)\)"
proof
  :  
  show "\(P(witness)\)" . . .
qed
\end{verbatim}
Proof patterns: \( \exists \) elimination: obtain
Proof patterns: \( \exists \) elimination: obtain

have \( \exists x. P(x) \)
then obtain \( x \) where \( p: P(x) \) by blast
\[ x \] fixed local variable
Proof patterns: $\exists$ elimination: obtain

\[\text{have } \exists x. P(x)\]
\[\text{then obtain } x \text{ where } p: P(x) \text{ by blast}\]
\[\vdash x \text{ fixed local variable}\]

Works for one or more $x$
lemmata \rightarrow \text{surj}(f :: 'a \\rightarrow 'a set)

proof
  \text{assume } \text{surj } f
  \text{hence } \exists a. \{x. x \notin f \ x\} = f a \text{ by (auto simp: surj_def)}
lemma \( \rightarrow \) 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
lemma \( \vdash \text{surj}(f :: 'a \Rightarrow 'a \text{ set}) \)

proof

assume \( \text{surj } f \)

hence \( \exists a. \{x. x \not\in f x\} = f a \) by (auto simp: surj_def)

then obtain \( a \) where \( \{x. x \not\in f x\} = f a \) by blast

hence \( a \not\in f a \longleftrightarrow a \in f a \) by blast
lemma \neg \text{surj}(f :: 'a \Rightarrow 'a 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 \iff a \in f a by blast
  thus False by blast
qed
Proof patterns: Set equality and subset

show "A = B"
proof
  show "A ⊆ B" ...
next
  show "B ⊆ A" ...
qed
Proof patterns: Set equality and subset

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
Example: pattern matching

show $formula_1 \leftrightarrow formula_2$ (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
Every show implicitly defines thesis
show formula (is \textit{thesis})
proof -
  :
  show \textit{thesis} ...
qed
Every show implicitly defines \(\text{thesis}\)
Introducing local abbreviations in proofs:

```plaintext
let ℓ = "some-big-term"
have "...?t ...
```
Quoting facts by value

By name:

```prolog
have x0: "x > 0" ...
```

From `x0` ...
Quoting facts by value

By name:

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

By value:

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

By name:

```has 
x0: "x > 0" ...
```

```from x0 ...
```

By value:

```has "x > 0" ...
```

```from 'x>0' ...
```

↑   ↑

back quotes
Example

lemma

“(∃ ys zs. xs = ys @ zs ∧ length ys = length zs) ∨
(∃ ys zs. xs = ys @ zs ∧ length ys = length zs + 1)”
Example

Lemma

“$(\exists \ ys \ zs. \ xs = ys \ @ \ zs \land \ length \ ys = length \ zs) \lor
(\exists \ ys \ zs. \ xs = ys \ @ \ zs \land \ length \ ys = length \ zs + 1)$”

Proof ???
When automation fails

Split proof up into smaller steps.
When automation fails

Split proof up into smaller steps.

Or explore by apply:
When automation fails

Split proof up into smaller steps.

Or explore by **apply**:

```
have ... using ...
```
When automation fails

Split proof up into smaller steps.

Or explore by *apply*:

- *have* ... *using* ...
- *apply* - to make incoming facts part of proof state

At the end:

- done
- Better: convert to structured proof
When automation fails

Split proof up into smaller steps.

Or explore by apply:

- have ... using ...
- apply - to make incoming facts part of proof state
- apply auto or whatever

At the end:
▶ done
▶ Better: convert to structured proof
When automation fails

Split proof up into smaller steps.

Or explore by **apply**:

- **have ... using ...**
- **apply** - to make incoming facts part of proof state
- **apply** *auto* or whatever
- **apply** ...

At the end:

▶ done
▶ Better: convert to structured proof
When automation fails

Split proof up into smaller steps.

Or explore by **apply**:

- **have** ... **using** ...

- **apply** - to make incoming facts part of proof state

- **apply** *auto* or whatever

- **apply** ...

At the end:
When automation fails

Split proof up into smaller steps.

Or explore by **apply**:

```
  have ... using ...
  apply -  to make incoming facts
           part of proof state
  apply *auto*  or whatever
  apply ...
```

At the end:

▶ **done**
When automation fails

Split proof up into smaller steps.

Or explore by **apply**: 

- `have ... using ...`
- `apply` to make incoming facts part of proof state
- `apply auto` or whatever
- `apply ...`

At the end:

- ▶ **done**
- ▶ Better: *convert to structured proof*
Moreover—ultimately

have “$P_1$” . . .
moreover
have “$P_2$” . . .
moreover
.:.
moreover
have “$P_n$” . . .
ultimately
have “$P$” . . .
moreover—ultimately

have “$P_1$” ...
moreover
have “$P_2$” ...
moreover
:
moreover
have “$P_n$” ...
ultimately
have “$P$” ...

have $lab_1$: “$P_1$” ...
have $lab_2$: “$P_2$” ...
:
have $lab_n$: “$P_n$” ...
from $lab_1\ lab_2$ ...
have “$P$” ...

With names
\{ \textbf{fix} \ x_1 \ldots \ x_n \\
\textbf{assume} \ A_1 \ldots \ A_m \\
\ldots \\
\textbf{have} \ B \}
\[
\{\text{fix } x_1 \ldots x_n \\
\text{assume } A_1 \ldots A_m \\
\vdots \\
\text{have } B
\}
\]

proves \( [A_1; \ldots; A_m] \Rightarrow B \)
Raw proof blocks

\{ \textbf{fix} \ x_1 \ldots \ x_n \\
\textbf{assume} \ A_1 \ldots \ A_m \\
\vdots \\
\textbf{have} \ B \}

proves \left[ A_1; \ldots ; A_m \right] \implies B

where all \( x_i \) have been replaced by \( ?x_i \).
Proof state and Isar text

In general:
proof method
Applies
method
and generates subgoal(s):
\( \land \ x_1 : : x_n [ [ A_1 ; : : ; A_m ] ] = B \)

How to prove each subgoal:
fix
x_1 : : x_n
assume
A_1 ; : : ; A_m
...
show
B
Separated by
next
Proof state and Isar text

In general:

proof method

\[ \land x_1 : : : x_n \begin{array}{c} A_1 ; : : ; A_m \end{array} = B \]

How to prove each subgoal:

fix \[ x_1 : : : x_n \]

assume \[ A_1 ; : : ; A_m \]

show \[ B \]

Separated by next
In general: proof \textit{method}

Applies \textit{method} and generates subgoal(s):

\[ \land x_1 \ldots x_n [A_1; \ldots ; A_m] \Longrightarrow B \]
Proof state and Isar text

In general: **proof method**

Applies *method* and generates subgoal(s):

\[ \land x_1 \ldots x_n \left[ A_1; \ldots ; A_m \right] \Rightarrow B \]

How to prove each subgoal:
Proof state and Isar text

In general: \textbf{proof} \textit{method}

Applies \textit{method} and generates subgoal(s):
\[ \land x_1 \ldots x_n \, [\, A_1; \ldots ; A_m \, ] \implies B \]

How to prove each subgoal:

\begin{verbatim}
fix  x_1 \ldots x_n
assume A_1 \ldots A_m
:
show  B
\end{verbatim}
Proof state and Isar text

In general: \textbf{proof method}

Applies \textit{method} and generates subgoal(s):

\[ \land x_1 \ldots x_n \left[ A_1; \ldots ; A_m \right] \Rightarrow B \]

How to prove each subgoal:

\begin{verbatim}
  fix  x_1 \ldots x_n 
  assume  A_1 \ldots A_m 
  : 
  show  B 
\end{verbatim}

Separated by \textbf{next}
Datatype case analysis

\texttt{datatype } t = C_1 \vec{r} \mid \ldots
Datatype case analysis

datatype \( t = C_1 \overrightarrow{\tau} \mid \ldots \)

proof (cases "term")
    case \((C_1 x_1 \ldots x_k)\)
        ... \(x_j\) ...
    next
    : 
    : 
    qed
Datatype case analysis

\[ \text{datatype } t = C_1 \vec{r} | \ldots \]

proof (cases "term")
   case \((C_1 x_1 \ldots x_k)\)
       \ldots x_j \ldots
   next
   :
   qed

where case \((C_i x_1 \ldots x_k)\) \equiv

fix \(x_1 \ldots x_k\)
assume \(C_i: \text{term} = (C_i x_1 \ldots x_k)\)
Structural induction for $nat$

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

proof \((\text{induction } n)\)

\[
\begin{align*}
\text{case } 0 & \quad \equiv \quad \text{let } ?\text{case} = P(0) \\
\vdots \\
\text{show } ?\text{case}
\end{align*}
\]

next

\[
\begin{align*}
\text{case } (\text{Suc } n) \\
\vdots \\
\vdots \\
\vdots \\
\text{show } ?\text{case}
\end{align*}
\]

qed
Structural induction for $nat$

\begin{align*}
& \text{show } P(n) \\
& \text{proof (induction n)} \\
& \hspace{1em} \text{case } 0 \quad \equiv \quad \text{let } \ ?\text{case} = P(0) \\
& \hspace{2em} \vdots \\
& \hspace{1em} \text{show } ?\text{case} \\
& \text{next} \\
& \hspace{1em} \text{case } (Suc\ n) \quad \equiv \quad \text{fix } \ n \ \text{assume } Suc: P(n) \\
& \hspace{2em} \text{let } ?\text{case} = P(Suc\ n) \\
& \hspace{2em} \vdots \\
& \hspace{2em} \vdots \\
& \hspace{1em} \text{show } ?\text{case} \\
& \text{qed}
\end{align*}
show \( A(n) \implies P(n) \)

proof (induction \( n \))
  case 0
    
    
    show ?case

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

qed
Structural induction with $\implies$

show $A(n) \implies P(n)$
proof (induction $n$)
  case 0
    : show ?case
next
  case (Suc $n$)
    : show ?case
qed
Structural induction with $\implies$

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

proof (induction $n$)
  case 0
    :=
    show ?case
  next
  case (Suc $n$)
    :=
    fix $n$
    assume Suc: $A(n) \implies P(n)$
    $A(Suc \, n)$
    let ?case = $P(Suc \, n)$

qed
Named assumptions

In a proof of

$$A_1 \implies \ldots \implies A_n \implies B$$

by structural induction:
Named assumptions

In a proof of

\[ A_1 \implies \ldots \implies A_n \implies B \]

by structural induction:
In the context of

\textbf{case} \ C
Named assumptions

In a proof of

\[ A_1 \implies \ldots \implies A_n \implies B \]

by structural induction:

In the context of

\textbf{case} C

we have

\textbf{C.IH} the induction hypotheses
Named assumptions

In a proof of

\[ A_1 \implies \ldots \implies A_n \implies B \]

by structural induction:

In the context of

\textbf{case} \ C

we have

\textbf{C.IH} the induction hypotheses

\textbf{C.prems} the premises \( A_i \)
Named assumptions

In a proof of
\[ A_1 \implies \ldots \implies A_n \implies B \]

by structural induction:
In the context of
\textbf{case} \ C

we have
\begin{align*}
\textbf{C.IH} & \text{ the induction hypotheses} \\
\textbf{C.prem}\text{s} & \text{ the premises } A_i \\
\textbf{C} & \text{ C.IH + C.prem}\text{s}
\end{align*}
A remark on style

- **case** \((\text{Suc } n)\) …**show** \(?case\)
  
is easy to write and maintain
A remark on style

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

- **fix** \( n \) **assume** \( \text{formula} \ ... \text{show} \ \text{formula}' \)
  is easier to read:
  - all information is shown locally
  - no contextual references (e.g. \(?\text{case}\)
Rule induction

\textbf{inductive} \( I :: \tau \Rightarrow \sigma \Rightarrow \text{bool} \)
\textbf{where}
\( \text{rule}_1 : \ldots \)
\( \vdots \)
\( \text{rule}_n : \ldots \)
Rule induction

\[
\text{inductive } I :: \tau \Rightarrow \sigma \Rightarrow \text{bool} \quad \text{show } I x y \Rightarrow P x y
\]

where

\[
\text{rule}_1 : \ldots
\]

\[
\vdots
\]

\[
\text{rule}_n : \ldots
\]
Rule induction

\textbf{inductive} \( I :: \tau \Rightarrow \sigma \Rightarrow \text{bool} \)

\textbf{where}

\textit{rule} \( _1 : \ldots \)

\vdots

\textit{rule} \( _n : \ldots \)

\textbf{show} \( I \ x \ y \ \Rightarrow \ P \ x \ y \)

\textbf{proof} \ (\textit{induction rule:} \( I\text{-induct} \))
Rule induction

inductive $I :: \tau \Rightarrow \sigma \Rightarrow \text{bool}$
where
$\text{rule}_1 : \ldots$

$\vdots$

$\text{rule}_n : \ldots$

show $I \ x \ y \Rightarrow P \ x \ y$
proof (induction rule: $I$.induct)
  case $\text{rule}_1$
    $\ldots$
  show $\ ? \ case$
next
  $\vdots$
next
  case $\text{rule}_n$
  $\ldots$
  show $\ ? \ case$
qed
Fixing your own variable names

\textbf{case} \ (\textit{rule}_i \ x_1 \ldots \ x_k)

Renames the first $k$ variables in $\textit{rule}_i$ (from left to right) to $x_1 \ldots x_k$. 
Named assumptions

In a proof of

\[ I \ldots \implies A_1 \implies \ldots \implies A_n \implies B \]

by rule induction on \( I \ldots \):
Named assumptions

In a proof of

\[ I \ldots \Rightarrow A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow B \]

by rule induction on \( I \ldots \):

In the context of

**case** \( R \)
Named assumptions

In a proof of

\[ I \ldots \implies A_1 \implies \ldots \implies A_n \implies B \]

by rule induction on \( I \ldots \):

In the context of

\textbf{case } \textit{R}

we have

\textbf{R.IH} \quad \text{the induction hypotheses}
Named assumptions

In a proof of

\[ I \ldots \implies A_1 \implies \ldots \implies A_n \implies B \]

by rule induction on \( I \ldots : \)
In the context of

**case** \( R \)

we have

*\( R.IH \) the induction hypotheses*

*\( R.hyps \) the assumptions of rule \( R \)*
Named assumptions

In a proof of

\[ I \ldots \Rightarrow A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow B \]

by rule induction on \( I \ldots \):

In the context of

\textbf{case} \( R \)

we have

- \textit{R.IH} the induction hypotheses
- \textit{R.hyps} the assumptions of rule \( R \)
- \textit{R.prems} the premises \( A_i \)
Named assumptions

In a proof of
\[ I \ldots \implies A_1 \implies \ldots \implies A_n \implies B \]
by rule induction on \( I \ldots : \)
In the context of
\[ \text{case } R \]
we have
\[ R.IH \quad \text{the induction hypotheses} \]
\[ R.hyps \quad \text{the assumptions of rule } R \]
\[ R.prems \quad \text{the premises } A_i \]
\[ R \quad R.IH + R.hyps + R.prems \]
Rule inversion

inductive ev :: “nat ⇒ bool” where
  ev0: “ev 0” |
  evSS: “ev n ⇒ ev(Suc(Suc n))”

What can we deduce from ev n?
Rule inversion

\begin{verbatim}
inductive ev :: "nat ⇒ bool" where
  ev0: "ev 0" |
  evSS: "ev n =⇒ ev(Suc(Suc n))"
\end{verbatim}

What can we deduce from \( ev n \)?
That it was proved by either \( ev0 \) or \( evSS \)!
Rule inversion

\textbf{inductive} ev :: “nat \Rightarrow bool” \textbf{where}
\begin{align*}
  \text{ev0:} & \quad \text{“} ev \ 0 \text{”} \\
  \text{evSS:} & \quad \text{“} ev \ n \Rightarrow ev(Suc(Suc \ n)) \text{”}
\end{align*}

What can we deduce from \( ev \ n \)?
That it was proved by either \( ev0 \) or \( evSS \)!

\( ev \ n \Rightarrow n = 0 \lor \exists k. n = Suc(Suc \ k) \land ev \ k \)
Rule inversion

\begin{itemize}
\item \textbf{inductive} \textit{ev} :: \textit{"nat \Rightarrow bool" where}
\item \textit{ev0: \textit{"ev 0" /}}
\item \textit{evSS: \textit{"ev n \Rightarrow ev(Suc(Suc n))"}}
\end{itemize}

What can we deduce from \textit{ev n}?
That it was proved by either \textit{ev0} or \textit{evSS}!

\textit{ev n \Rightarrow n = 0 \lor (\exists k. n = Suc (Suc k) \land ev k)}

\textbf{Rule inversion = case distinction over rules}
Rule inversion template

from 'ev n' have "P"
proof cases
  case ev0
   
   show ?thesis ...  
next
  case (evSS k)  
   
   show ?thesis ...  
qed

Impossible cases disappear automatically
Rule inversion template

from 'ev n' have "P"

proof cases
  case ev0
  |
  |
  show ?thesis . . .

next
  case (evSS k)
  |
  |
  show ?thesis . . .

qed

Impossible cases disappear automatically
Summary

- Introduction to Isar and to some common proof patterns e.g. case distinction, contradiction, etc.
- Structured proofs are becoming the norm for Isabelle as they are more readable and easier to maintain.
- Mastering structured proof takes practice and it is usually better to have a clear proof plan beforehand.
- Useful resource: Isar quick reference manual (see AR web page).
- Reading: N&K (Concrete Semantics), Chapter 5.