Script **generated by TTT**

Title: FDS (02.06.2017)

Date: Fri Jun 02 08:30:13 CEST 2017

Duration: 99:37 min

Pages: 46

TTT 100% Fri 08:30

When automation fails

Split proof up into smaller steps.

maximize

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

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

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

Local lemmas

have B **if** *name:* $A_1 \ \dots \ A_m$ **for** $x_1 \ \dots \ x_n$

Local lemmas

have B **if** *name:* $A_1 \ \dots \ A_m$ **for** $x_1 \ \dots \ x_n$

proves $\llbracket A_1; \dots ; A_m \rrbracket \Longrightarrow B$

Local lemmas

have B **if** *name:* $A_1 \ \dots \ A_m$ **for** $x_1 \ \dots \ x_n$

proves $\llbracket A_1; \dots ; A_m \rrbracket \Longrightarrow B$
where all x_i have been replaced by $?x_i$.

Proof state and Isar text

Proof state and Isar text

In general: **proof** *method*

Proof state and Isar text

In general: **proof** *method*

Applies *method* and generates subgoal(s):

$$\bigwedge x_1 \dots x_n.\ [\![A_1; \dots ; A_m]\!] \Longrightarrow B$$

Proof state and Isar text

In general: **proof** *method*

Applies *method* and generates subgoal(s):

$$\bigwedge x_1 \dots x_n.\ [\![A_1; \dots ; A_m]\!] \Longrightarrow B$$

How to prove each subgoal:

Proof state and Isar text

In general: **proof** *method*

Applies *method* and generates subgoal(s):

$$\bigwedge x_1 \dots x_n.\ [\![A_1; \dots ; A_m]\!] \Longrightarrow B$$

How to prove each subgoal:

fix $x_1 \dots x_n$
assume $A_1 \dots A_m$
$\vdots$
show B

Proof state and Isar text

In general: **proof** *method*

Applies *method* and generates subgoal(s):

$$\bigwedge x_1 \dots x_n.\ [\![A_1; \dots ; A_m]\!] \Longrightarrow B$$

How to prove each subgoal:

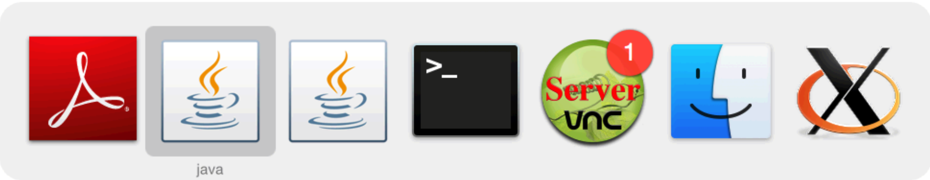
fix $x_1 \dots x_n$
assume $A_1 \dots A_m$
$\vdots$
show B

Separated by **next**

Isar_Induction_Demo.thy

Proof by cases

Isar_Induction_Demo.thy



Datatype case analysis

datatype $t = C_1\ \vec{\tau}\ \mid\ \ldots$

$$\boxed{\begin{array}{l}\textbf{proof}\ (cases\ "term")\\ \quad \textbf{case}\ (C_1\ x_1\ \ldots\ x_k)\\ \quad \ldots\ x_j\ \ldots\\ \textbf{next}\\ \vdots\\ \textbf{qed}\end{array}}$$

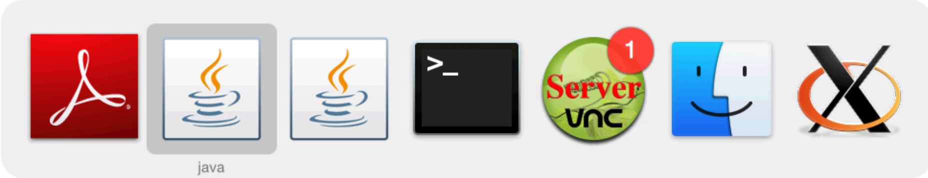
where $\quad \textbf{case}\ (C_i\ x_1\ \ldots\ x_k) \quad \equiv$

$$\begin{array}{l}\textbf{fix}\ x_1\ \ldots\ x_k\\ \textbf{assume}\ \underbrace{C_i\text{:}}_{\text{label}}\ \underbrace{term = (C_i\ x_1\ \ldots\ x_k)}_{\text{formula}}\end{array}$$

`Isar_Induction_Demo.thy`

Structural induction for nat

`Isar_Induction_Demo.thy`



Structural induction for nat

show $P(n)$
proof $(induction\ n)$
  case 0 $\equiv$ **let** $?case = P(0)$
  $\vdots$
  show $?case$
next
  case $(Suc\ n)$
  $\vdots$
  $\vdots$
  show $?case$
qed

Structural induction for nat

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

Structural induction for nat

show $P(n)$
proof $(induction\ n)$
  case 0 $\equiv$ **let** $?case = P(0)$

  ⋮ **let** $?case = P(Suc\ n)$
  ⋮
  show $?case$
qed

Structural induction with $\Longrightarrow$

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

Named assumptions

In a proof of

$$A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow B$$

by structural induction:
In the context of

  case C

we have

  $C.IH$ the induction hypotheses

Named assumptions

In a proof of

$$A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow B$$

by structural induction:
In the context of

  case C

we have

  $C.IH$ the induction hypotheses
  $C.prems$ the premises A_i

Named assumptions

In a proof of

$$A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow B$$

by structural induction:
In the context of

case C

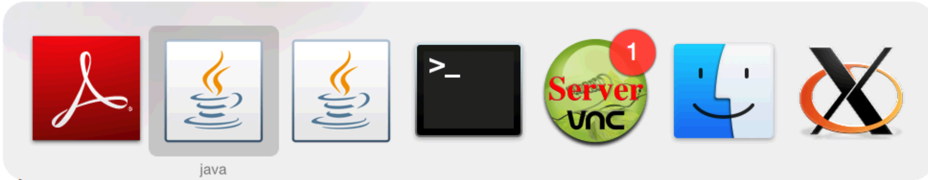
we have

$C.IH$	the induction hypotheses
$C.prems$	the premises A_i
C	$C.IH + C.prems$

Named assumptions

In a proof of

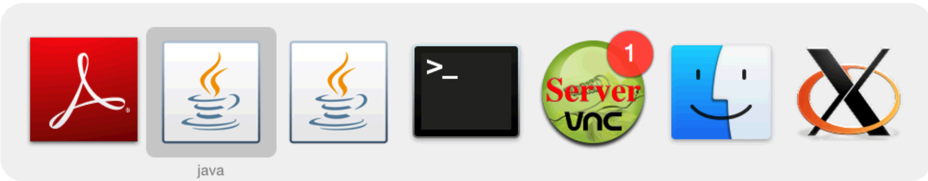
$$A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow B$$



we have

$C.IH$	the induction hypotheses
$C.prems$	the premises A_i
C	$C.IH + C.prems$

`Isar_Induction_Demo.thy`



Computation induction

If function f is defined by **fun** with n equations:

proof$(induction\ s\ t\ \ldots\ rule\text{:}\ f.induct)$

Generates cases named $i = 1 \ldots n$:

case $(i\ x\ y\ \ldots)$

Computation induction

If function f is defined by **fun** with n equations:

proof($induction\ s\ t\ ...\ rule\!: f.induct$)

Generates cases named $i = 1 \ldots n$:

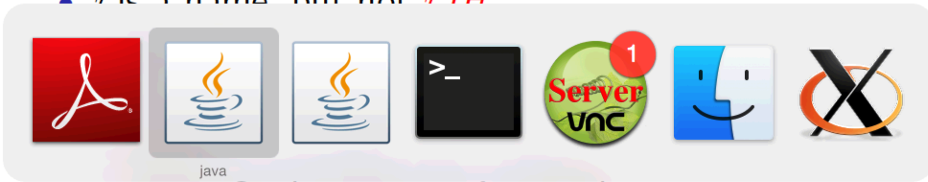
case $(i\ x\ y\ ...)$

Isabelle/jEdit generates Isar template for you!

Computation induction

Naming

- i is a name, but not $i.IH$



$\leadsto$ Isabelle instantiates overlapping equations
$\leadsto$ case names of the form "i_j"

$sorted :: ('a{::}linorder)\ list \Rightarrow bool$

$sorted\ [] = True$
$sorted\ (x \# xs) = ((\forall y{\in}set\ xs.\ x \leq y) \wedge sorted\ xs)$

Correctness of sorting

Specification of $sort :: ('a{::}linorder)\ list \Rightarrow 'a\ list$:

$sorted\ (sort\ xs)$

Correctness of sorting

Specification of $sort :: ('a::linorder)\ list \Rightarrow 'a\ list$:

$$sorted\ (sort\ xs)$$

Is that it?

Correctness of sorting

Specification of $sort :: ('a::linorder)\ list \Rightarrow 'a\ list$:

$$sorted\ (sort\ xs)$$

Is that it? How about

$$set\ (sort\ xs) = set\ xs$$

Correctness of sorting

Specification of $sort :: ('a::linorder)\ list \Rightarrow 'a\ list$:

$$sorted\ (sort\ xs)$$

Is that it? How about

$$set\ (sort\ xs) = set\ xs$$

Better: every x occurs as often in $sort\ xs$ as in xs.

More succinctly:

$$mset\ (sort\ xs) = mset\ xs$$

where $mset :: 'a\ list \Rightarrow 'a\ multiset$

What are multisets?

Sets with (possibly) repeated elements

Some operations:

$$\begin{array}{rcl} \{\#\} & :: & 'a\ multiset \\ add\_mset & :: & 'a \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \\ + & :: & 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \end{array}$$

What are multisets?

Sets with (possibly) repeated elements

Some operations:

$$\begin{array}{rcl} \{\#\} & :: & 'a\ multiset \\ add\_mset & :: & 'a \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \\ + & :: & 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \\ mset & :: & 'a\ list \Rightarrow 'a\ multiset \\ set\_mset & :: & 'a\ multiset \Rightarrow 'a\ set \end{array}$$

1. Correctness
2. Insertion Sort
3. Time
4. Merge Sort

Thys/Sorting.thy

Insertion sort correctness