universität innsbruck

Summer Term 2019

Interactive Theorem Proving using Isabelle/HOL

Session 3

Christian Sternagel

Department of Computer Science

Topics

calculational reasoning, case analysis, code generation, computation induction, data type invariants, document preparation, finding theorems, first steps, functional programming in HOL, higher-order logic, history and motivation, induction, inductive definitions, Isabelle basics, Isabelle/Isar, Isabelle/ML, IsaFoR/CeTA, locales, manual termination proofs, multisets, natural deduction, notation, proof methods, PSL: a high-level proof strategy language, rule induction, rule inversion, session management, sets, simplification, sledgehammer, structural induction, structured proof, The Archive of Formal Proofs, the certification approach, total recursive functions, type classes, type definitions, well-foundedness

Topics

calculational reasoning, case analysis, code generation, computation induction, data type invariants, document preparation, finding theorems, first steps, functional programming in HOL, higher-order logic, history and motivation, induction, inductive definitions, Isabelle basics, Isabelle/Isar, Isabelle/ML, IsaFoR/CeTA, locales, manual termination proofs, multisets, natural deduction, notation, proof methods, PSL: a high-level proof strategy language, rule induction, rule inversion, session management, sets, simplification, sledgehammer, structural induction, structured proof, The Archive of Formal Proofs, the certification approach, total recursive functions, type classes, type definitions, well-foundedness

Overview

- Finding Existing Results
- Single Step Proving (aka Natural Deduction)
- Case Analysis and Structural Induction
- Exercises

Finding Existing Results

Isabelle/jEdit's Query Panel

enter query in "Find:" input field of "Find Theorems" tab

Isabelle/jEdit's Query Panel

enter query in "Find:" input field of "Find Theorems" tab

Search Criteria

- `name: foo` – search for facts whose name contains substring "`foo`"
- `"`*pattern*`"` – search for facts that match *pattern*
- prefix criterion by "–" to exclude facts that match
- combine several criteria by juxtaposition

Isabelle/jEdit's Query Panel

enter query in "Find:" input field of "Find Theorems" tab

Search Criteria

- `name: foo` – search for facts whose name contains substring "`foo`"
- `"`*pattern*`"` – search for facts that match *pattern*
- prefix criterion by "–" to exclude facts that match
- combine several criteria by juxtaposition

Search Patterns

HOL terms with schematic variables `?x`, `?y`, ... or _ instead of free variables

Isabelle/jEdit's Query Panel

enter query in "Find:" input field of "Find Theorems" tab

Search Criteria

- `name: foo` – search for facts whose name contains substring "`foo`"
- "*pattern*" – search for facts that match *pattern*
- prefix criterion by "–" to exclude facts that match
- combine several criteria by juxtaposition

Search Patterns

HOL terms with schematic variables `?x`, `?y`, ... or _ instead of free variables

Examples

query	finds facts mentioning	query	finds facts mentioning

Isabelle/jEdit's Query Panel

enter query in "Find:" input field of "Find Theorems" tab

Search Criteria

- `name: foo` – search for facts whose name contains substring "`foo`"
- `"`*pattern*`"` – search for facts that match *pattern*
- prefix criterion by "–" to exclude facts that match
- combine several criteria by juxtaposition

Search Patterns

HOL terms with schematic variables `?x`, `?y`, ... or _ instead of free variables

Examples

query	finds facts mentioning	query	finds facts mentioning
`_ + _`	addition		

Isabelle/jEdit's Query Panel

enter query in "Find:" input field of "Find Theorems" tab

Search Criteria

- `name: foo` – search for facts whose name contains substring "`foo`"
- `"`*pattern*`"` – search for facts that match *pattern*
- prefix criterion by "–" to exclude facts that match
- combine several criteria by juxtaposition

Search Patterns

HOL terms with schematic variables `?x`, `?y`, ... or _ instead of free variables

Examples

query	finds facts mentioning	query	finds facts mentioning
`_ + _`	addition	`map`	`map` function

Isabelle/jEdit's Query Panel

enter query in "Find:" input field of "Find Theorems" tab

Search Criteria

- `name: foo` – search for facts whose name contains substring "`foo`"
- `"`*pattern*`"` – search for facts that match *pattern*
- prefix criterion by "–" to exclude facts that match
- combine several criteria by juxtaposition

Search Patterns

HOL terms with schematic variables `?x`, `?y`, ... or _ instead of free variables

Examples

query	finds facts mentioning	query	finds facts mentioning
`_ + _`	addition	`map`	`map` function
`?x + ?x`	addition of same value		

Isabelle/jEdit's Query Panel

enter query in "Find:" input field of "Find Theorems" tab

Search Criteria

- `name: foo` – search for facts whose name contains substring "`foo`"
- `"`*pattern*`"` – search for facts that match *pattern*
- prefix criterion by "–" to exclude facts that match
- combine several criteria by juxtaposition

Search Patterns

HOL terms with schematic variables `?x`, `?y`, ... or _ instead of free variables

Examples

query	finds facts mentioning	query	finds facts mentioning
`_ + _`	addition	`map`	`map` function
`?x + ?x`	addition of same value	`map (λx. x) ?xs = ?xs`	equation

Customizing Keyboard Shortcuts

- go to Utilities → Global Options . . . → jEdit → Shortcuts

Customizing Keyboard Shortcuts

- go to Utilities → Global Options . . . → jEdit → Shortcuts
- choose Action Set: Plugin: Isabelle

Customizing Keyboard Shortcuts

- go to Utilities → Global Options ... → jEdit → Shortcuts
- choose Action Set: Plugin: Isabelle
- customize shortcuts for commands

Customizing Keyboard Shortcuts

- go to Utilities → Global Options . . . → jEdit → Shortcuts
- choose Action Set: Plugin: Isabelle
- customize shortcuts for commands

Example

set Primary shortcut of command "Query panel (Toggle)" to CTRL + e q

Customizing Keyboard Shortcuts

- go to Utilities → Global Options . . . → jEdit → Shortcuts
- choose Action Set: Plugin: Isabelle
- customize shortcuts for commands

Example

set Primary shortcut of command "Query panel (Toggle)" to CTRL+e q

Isabelle/jEdit Gesture – "Control Click/Hover"

- while pressing "control key" hover with mouse pointer: CTRL+
- reveals further information (if available)
- if box appears around target, click jumps to "origin"

Single Step Proving (aka Natural Deduction)

Example – Natural Deduction Inference Rules

connective	introduction	elimination
$\neg$	$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\,\phi}$	$\dfrac{\neg\,\phi \qquad \phi}{\psi}$
$\wedge$	$\dfrac{\phi \qquad \psi}{\phi \;\wedge\; \psi}$	$\dfrac{\phi \;\wedge\; \psi}{\phi}$ $\quad$ $\dfrac{\phi \;\wedge\; \psi}{\psi}$
$\vee$	$\dfrac{\phi}{\phi \;\vee\; \psi}$ $\quad$ $\dfrac{\psi}{\phi \;\vee\; \psi}$	$\dfrac{\phi \;\vee\; \psi \qquad \boxed{\begin{array}{c}\phi\\ \vdots\\ \chi\end{array}} \qquad \boxed{\begin{array}{c}\psi\\ \vdots\\ \chi\end{array}}}{\chi}$

Example – Natural Deduction Inference Rules (cont'd)

connective	introduction	elimination
$\longrightarrow$	$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \longrightarrow \psi}$	$\dfrac{\phi \longrightarrow \psi \qquad \phi}{\psi}$
$\forall$	$\dfrac{\boxed{\begin{array}{c}x_0\\ \vdots\\ \phi(x_0)\end{array}}}{\forall \mathtt{x}.\ \phi(\mathtt{x})}$	$\dfrac{\forall \mathtt{x}.\ \phi(\mathtt{x})}{\phi(\mathtt{a})}$
$\exists$	$\dfrac{\phi(\mathtt{a})}{\exists \mathtt{x}.\ \phi(\mathtt{x})}$	$\dfrac{\exists \mathtt{x}.\ \phi(\mathtt{x}) \qquad \boxed{\begin{array}{c}x_0\phi(x_0)\\ \vdots\\ \psi\end{array}}}{\psi}$

Example Proof

$$A \longrightarrow (\forall x.\ P(x))$$

$$\forall x.\ A \longrightarrow P(x)$$

Example Proof

$$\mathtt{A} \longrightarrow (\forall \mathtt{x}.\ \mathtt{P}(\mathtt{x}))$$

$$\boxed{\begin{array}{l} x_0 \\ \\ \mathtt{A} \longrightarrow \mathtt{P}(x_0) \end{array}}$$

$$\forall \mathtt{x}.\ \mathtt{A} \longrightarrow \mathtt{P}(\mathtt{x})$$

$$\frac{\boxed{\begin{array}{c} x_0 \\ \vdots \\ \phi(x_0) \end{array}}}{\forall \mathtt{x}.\ \phi(\mathtt{x})}$$

Example Proof

1	$\mathtt{A} \longrightarrow (\forall \mathtt{x}.\ \mathtt{P}(\mathtt{x}))$
2	x_0
	$\mathtt{A} \longrightarrow \mathtt{P}(x_0)$
	$\forall \mathtt{x}.\ \mathtt{A} \longrightarrow \mathtt{P}(\mathtt{x})$ $\quad$ $\forall$ introduction 2–

$$\frac{\boxed{\begin{array}{c} x_0 \\ \vdots \\ \phi(x_0) \end{array}}}{\forall \mathtt{x}.\ \phi(\mathtt{x})}$$

Example Proof

1	$\mathtt{A} \longrightarrow (\forall \mathtt{x}.\ \mathtt{P}(\mathtt{x}))$	
2	x_0	
3	$\mathtt{A}$	assumption
	$\mathtt{P}(x_0)$	
	$\mathtt{A} \longrightarrow \mathtt{P}(x_0)$	$\longrightarrow$ introduction 3–
	$\forall \mathtt{x}.\ \mathtt{A} \longrightarrow \mathtt{P}(\mathtt{x})$	$\forall$ introduction 2–

$$\frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \longrightarrow \psi}$$

Example Proof

$$\frac{\phi \longrightarrow \psi \qquad \phi}{\psi}$$

1	$\mathtt{A} \longrightarrow (\forall \mathtt{x}.\ \mathtt{P}(\mathtt{x}))$	
2	x_0	
3	$\mathtt{A}$	assumption
4	$\forall \mathtt{x}.\ \mathtt{P}(\mathtt{x})$	$\longrightarrow$ elimination 1, 3
	$\mathtt{P}(x_0)$	
	$\mathtt{A} \longrightarrow \mathtt{P}(x_0)$	$\longrightarrow$ introduction 3–
	$\forall \mathtt{x}.\ \mathtt{A} \longrightarrow \mathtt{P}(\mathtt{x})$	$\forall$ introduction 2–

Example Proof

$$\frac{\forall x.\ \phi(x)}{\phi(a)}$$

1	$A \longrightarrow (\forall x.\ P(x))$	
2	x_0	
3	A	assumption
4	$\forall x.\ P(x)$	$\longrightarrow$ elimination 1, 3
5	$P(x_0)$	$\forall$ elimination 4
6	$A \longrightarrow P(x_0)$	$\longrightarrow$ introduction 3–5
	$\forall x.\ A \longrightarrow P(x)$	$\forall$ introduction 2–6

Example Proof

1	$\mathtt{A} \longrightarrow (\forall \mathtt{x}.\ \mathtt{P}(\mathtt{x}))$	
2	x_0	
3	$\mathtt{A}$	assumption
4	$\forall \mathtt{x}.\ \mathtt{P}(\mathtt{x})$	$\longrightarrow$ elimination 1, 3
5	$\mathtt{P}(x_0)$	$\forall$ elimination 4
6	$\mathtt{A} \longrightarrow \mathtt{P}(x_0)$	$\longrightarrow$ introduction 3–5
	$\forall \mathtt{x}.\ \mathtt{A} \longrightarrow \mathtt{P}(\mathtt{x})$	$\forall$ introduction 2–6

Corresponding Isabelle/HOL Fact

$$\mathtt{?A} \longrightarrow (\forall \mathtt{x}.\ \mathtt{?P\ x}) \Longrightarrow \forall \mathtt{x}.\ \mathtt{?A} \longrightarrow \mathtt{?P\ x}$$

Current Facts

- inside proof current facts are bound to special name `this`
- indicated in proof state by "`using this:`"

Current Facts

- inside proof current facts are bound to special name `this`
- indicated in proof state by "`using this:`"

The rule Method

- remember: each fact/proposition is inference rule
- `rule` *fact* – if current facts empty, apply (fact) as intro rule, otherwise as elim rule

Current Facts

- inside proof current facts are bound to special name `this`
- indicated in proof state by "`using this:`"

The rule Method

- remember: each fact/proposition is inference rule
- `rule` *fact* – if current facts empty, apply (fact) as intro rule, otherwise as elim rule

Rule Application

- given rule $P_1 \implies \ldots \implies P_n \implies C$

Current Facts

- inside proof current facts are bound to special name `this`
- indicated in proof state by "`using this:`"

The rule Method

- remember: each fact/proposition is inference rule
- `rule` *fact* – if current facts empty, apply (fact) as intro rule, otherwise as elim rule

Rule Application

- given rule $\mathtt{P}_1 \implies \ldots \implies \mathtt{P}_n \implies \mathtt{C}$
- intro – unify $\mathtt{C}$ with conclusion of current subgoal and add correspondingly instantiated premises $\mathtt{P}_1\sigma, \ldots, \mathtt{P}_n\sigma$ as new subgoals

Current Facts

- inside proof current facts are bound to special name `this`
- indicated in proof state by "`using this:`"

The rule Method

- remember: each fact/proposition is inference rule
- `rule` *fact* – if current facts empty, apply (fact) as intro rule, otherwise as elim rule

Rule Application

- given rule $\mathtt{P}_1 \implies \ldots \implies \mathtt{P}_n \implies \mathtt{C}$
- intro – unify $\mathtt{C}$ with conclusion of current subgoal and add correspondingly instantiated premises $\mathtt{P}_1\sigma, \ldots, \mathtt{P}_n\sigma$ as new subgoals
- elim – unify major premise $\mathtt{P}_1$ of rule with first of current facts; unify remaining current facts with remaining premises; add rest of premises correspondingly instantiated as new subgoals

Example

```
lemma "A ⟶ (∀x. P x) ⟹ ∀x. A ⟶ P x"
proof -
  assume "A ⟶ (∀x. P x)"
  {
    fix x
    {
      assume "A"
      from ‹A ⟶ (∀x. P x)› and ‹A›
      have "∀x. P x" by (rule mp) — ‹?P ⟶ ?Q ⟹ ?P ⟹ ?Q›
      from this have "P x" by (rule spec) — ‹∀x. ?P x ⟹ ?P ?x›
    } — ‹(?P ⟹ ?Q) ⟹ ?P ⟶ ?Q›
    from this have "A ⟶ P x" by (rule impI)
  } — ‹(⋀x. ?P x) ⟹ ∀x. ?P x›
  from this show "∀x. A ⟶ P x" by (rule allI)
qed
```

Example – “then” abbreviates “from this”

```
lemma "A ⟶ (∀x. P x) ⟹ ∀x. A ⟶ P x"
proof -
  assume "A ⟶ (∀x. P x)"
  {
    fix x
    {
      assume "A"
      from ‹A ⟶ (∀x. P x)› and ‹A›
      have "∀x. P x" by (rule mp)
      then have "P x" by (rule spec)
    }
    then have "A ⟶ P x" by (rule impI)
  }
  then show "∀x. A ⟶ P x" by (rule allI)
qed
```

Example – ".." applies default intro/elim rule

```
lemma "A ⟶ (∀x. P x) ⟹ ∀x. A ⟶ P x"
proof -
  assume "A ⟶ (∀x. P x)"
  {
    fix x
    {
      assume "A"
      from ‹A ⟶ (∀x. P x)› and ‹A›
      have "∀x. P x" ..
      then have "P x" ..
    }
    then have "A ⟶ P x" ..
  }
  then show "∀x. A ⟶ P x" ..
qed
```

Example – The Same Proof But Less Forward

```
lemma "A ⟶ (∀x. P x) ⟹ ∀x. A ⟶ P x"
proof -
  assume "A ⟶ (∀x. P x)"
  show "∀x. A ⟶ P x"
  proof (rule allI)
    fix x show "A ⟶ P x"
    proof (rule impI)
      assume "A"
      from ‹A ⟶ (∀x. P x)› and ‹A›
      have "∀x. P x" by (rule mp)
      then show "P x" by (rule spec)
    qed
  qed
qed
```

Example – using implicit standard method

```
lemma "A ⟶ (∀x. P x) ⟹ ∀x. A ⟶ P x"
proof -
  assume "A ⟶ (∀x. P x)"
  show "∀x. A ⟶ P x"
  proof
    fix x show "A ⟶ P x"
    proof
      assume "A"
      from ‹A ⟶ (∀x. P x)› and ‹A›
      have "∀x. P x" ..
      then show "P x" ..
    qed
  qed
qed
```

Case Analysis and Structural Induction

Manual Case Analysis via Plain Natural Deduction

- `case_split: (?P ⟹ ?Q) ⟹ (¬ ?P ⟹ ?Q) ⟹ ?Q`
- apply `case_split` as intro rule for proof by case analysis `proof (rule case_split)`
- separate resulting subgoals by `next` (works in any situation with multiple subgoals)
- schematic `?Q` fixed by conclusion of current subgoal
- schematic `?P` implicitly fixed by first `assume`

Manual Case Analysis via Plain Natural Deduction

- `case_split: (?P ⟹ ?Q) ⟹ (¬ ?P ⟹ ?Q) ⟹ ?Q`
- apply `case_split` as intro rule for proof by case analysis `proof (rule case_split)`
- separate resulting subgoals by `next` (works in any situation with multiple subgoals)
- schematic `?Q` fixed by conclusion of current subgoal
- schematic `?P` implicitly fixed by first `assume`

Example - The Law of Excluded Middle

```
lemma LEM: "A ∨ ¬ A"
proof (rule case_split)
  assume A
  then show "A ∨ ¬ A" ..
next
  assume "¬ A"
  then show "A ∨ ¬ A" ..
qed
```

Drawback

several copies of (potentially huge) case formula (`A` in example)

Drawback

several copies of (potentially huge) case formula (`A` in example)

The cases Method

- `cases` – Boolean case split (on implicit formula)
- `cases "`*term*`"` – case split on constructors of type of *term*

Drawback

several copies of (potentially huge) case formula (`A` in example)

The cases Method

- `cases` – Boolean case split (on implicit formula)
- `cases "`*term*`"` – case split on constructors of type of *term*

The case Keyword

- use `case` statement according to grammar

statement	::=	`case` *name*	named case
	|	(`case` *name* (*name* | _)*)	named case with instantiations

  where first *name* is specific to applied rule (e.g., constructor names of data type)
- makes assumption(s) corresponding to current case available with name *name*

Drawback

several copies of (potentially huge) case formula (`A` in example)

The cases Method

- `cases` – Boolean case split (on implicit formula)
- `cases "`*term*`"` – case split on constructors of type of *term*

The case Keyword

- use `case` statement according to grammar

statement	::=	`case` *name*	named case
	|	(`case` *name* (*name* | _)*)	named case with instantiations

  where first *name* is specific to applied rule (e.g., constructor names of data type)
- makes assumption(s) corresponding to current case available with name *name*
- refer to conclusion of current subgoal by `?thesis` (not specific to case analysis)

Drawback

several copies of (potentially huge) case formula (`A` in example)

The cases Method

- `cases` – Boolean case split (on implicit formula)
- `cases "`*term*`"` – case split on constructors of type of *term*

The case Keyword

- use `case` statement according to grammar

statement	::=	`case` *name*	named case
	|	(`case` *name* (*name* | _)*)	named case with instantiations

  where first *name* is specific to applied rule (e.g., constructor names of data type)
- makes assumption(s) corresponding to current case available with name *name*
- refer to conclusion of current subgoal by `?thesis` (not specific to case analysis)
- proof outline available in “Output” panel (insert by clicking on it)

Examples

```
lemma "A ∨ ¬ A"
proof (cases A)
  case True
  then show ?thesis ..
next
  case False
  then show ?thesis ..
qed
```

Examples

```
lemma "A ∨ ¬ A"
proof (cases A)
  case True
  then show ?thesis ..
next
  case False
  then show ?thesis ..
qed
```

```
fun prefixes :: "'a list ⇒ 'a list list"
  where
    "prefixes [] = [[]]"
  | "prefixes (x # xs) =
      [] # map ((#) x) (prefixes xs)"
```

Examples

```
lemma "A ∨ ¬ A"
proof (cases A)
  case True
  then show ?thesis ..
next
  case False
  then show ?thesis ..
qed
```

```
fun prefixes :: "'a list ⇒ 'a list list"
  where
    "prefixes [] = [[]]"
  | "prefixes (x # xs) =
      [] # map ((#) x) (prefixes xs)"

lemma "prefixes xs ≠ []"
proof (cases xs)
  case Nil
  then show ?thesis by auto
next
  case (Cons y ys)
  then show ?thesis by auto
qed
```

Remark

use infix operator `op` as prefix function by `(op)`

Remark

use infix operator `op` as prefix function by `(op)`

Isabelle Symbol – Inequality

symbol	internal	auto completion
$\neq$	`\<noteq>`	~ =

Remark

use infix operator `op` as prefix function by `(op)`

Isabelle Symbol – Inequality

symbol	internal	auto completion
$\neq$	`\<noteq>`	~ =

Automatic Case Analysis on Case-Expressions – The split Modifier

- each data type `d` comes with (case) split rule `d.split`

Remark

use infix operator `op` as prefix function by `(op)`

Isabelle Symbol – Inequality

symbol	internal	auto completion
$\neq$	`\<noteq>`	~ =

Automatic Case Analysis on Case-Expressions – The split Modifier

- each data type `d` comes with (case) split rule `d.split`
- for if-expressions there is `if_split`

Remark

use infix operator `op` as prefix function by `(op)`

Isabelle Symbol – Inequality

symbol	internal	auto completion
$\neq$	`\<noteq>`	~ =

Automatic Case Analysis on Case-Expressions – The split Modifier

- each data type `d` comes with (case) split rule `d.split`
- for if-expressions there is `if_split`
- `(auto split:` *split-rules*`)` applies *split-rules* to replace corresponding case-expressions

Remark

use infix operator `op` as prefix function by `(op)`

Isabelle Symbol – Inequality

symbol	internal	auto completion
≠	`\<noteq>`	~ =

Automatic Case Analysis on Case-Expressions – The split Modifier

- each data type `d` comes with (case) split rule `d.split`
- for if-expressions there is `if_split`
- `(auto split:` *split-rules*`)` applies *split-rules* to replace corresponding case-expressions

Example

```
lemma "(case n of 0 ⇒ None | Suc _ ⇒ m div n) =
    (if n = 0 then None else Some (m div n))"
  by (auto split: nat.split)
```

Manual Induction via Plain Natural Deduction

- `nat.induct: ?P 0` $\Longrightarrow$ `(`$\bigwedge$`n. ?P n` $\Longrightarrow$ `?P (Suc n))` $\Longrightarrow$ `?P ?n`
- apply `nat.induct` as intro rule for proof by induction `proof (rule nat.induct)`
- explicit instantiation of schematics (from left to right; skip position by _) in *fact* by *fact* `[of` x_1 ... x_n`]`

Manual Induction via Plain Natural Deduction

- `nat.induct: ?P 0 ⟹ (⋀n. ?P n ⟹ ?P (Suc n)) ⟹ ?P ?n`
- apply `nat.induct` as intro rule for proof by induction `proof (rule nat.induct)`
- explicit instantiation of schematics (from left to right; skip position by _) in *fact* by *fact* `[of` x_1 ... x_n`]`

Example - Gauß's Formula $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$

```
lemma "∑{..n::nat} = n * (n + 1) div 2"
proof (rule nat.induct [of _ n])
  show "∑{..0::nat} = 0 * (0 + 1) div 2" by auto
next
  fix n
  assume "∑{..n::nat} = n * (n + 1) div 2"
  then show "∑{..Suc n::nat} = Suc n * (Suc n + 1) div 2" by auto
qed
```

Isabelle Symbol - Big Sum

symbol	internal	abbreviation
$\sum$	`\<Sum>`	S U M

Isabelle Symbol - Big Sum

symbol	internal	abbreviation
$\sum$	`\<Sum>`	S U M

Drawbacks

- manual instantiation of induction rule

Isabelle Symbol - Big Sum

symbol	internal	abbreviation
$\sum$	`\<Sum>`	S U M

Drawbacks

- manual instantiation of induction rule
- several copies of variants of property (original statement, base case, IH, step case, ...)

The induction Method

- `induction x` – induction on parameter `x` (rule chosen according to type of `x`)
- use `case` to start case
- separate cases by `next`
- `?case` abbreviates goal of current case

The induction Method

- `induction x` – induction on parameter `x` (rule chosen according to type of `x`)
- use `case` to start case
- separate cases by `next`
- `?case` abbreviates goal of current case

Example

```
lemma "∑{..n::nat} = n * (n + 1) div 2"
proof (induction n)
  case 0
  show ?case by auto
next
  case (Suc n)
  then show ?case by auto
qed
```

Generalization

- sometimes proving `P x y` by induction on `x` yields too weak IH (since `y` is fixed)

Generalization

- sometimes proving `P x y` by induction on `x` yields too weak IH (since `y` is fixed)
- in such cases, we can either prove (equivalent) ∀`y`. `P x y` instead

Generalization

- sometimes proving `P x y` by induction on `x` yields too weak IH (since `y` is fixed)
- in such cases, we can either prove (equivalent) `∀y. P x y` instead
- or explicitly make `y` arbitrary during induction using
  `proof (induction x arbitrary: y)`

Generalization

- sometimes proving `P x y` by induction on `x` yields too weak IH (since `y` is fixed)
- in such cases, we can either prove (equivalent) `∀y. P x y` instead
- or explicitly make `y` arbitrary during induction using
  `proof (induction x arbitrary: y)`

`Demo03.thy` – Iterative Reverse

```
fun itrev :: "'a list ⇒ 'a list ⇒ 'a list"
  where
    "itrev [] acc = acc"
  | "itrev (x # xs) acc = itrev xs (x # acc)"

lemma itrev_rev_conv: "itrev xs [] = rev xs"
  sorry
```

Exercises

Exercises (start from `Exercises03.thy`)

URL

http://cl-informatik.uibk.ac.at/teaching/ss19/itp/thys/Exercises03.thy

Important Concepts

- `..` (*command*)
- `?case` (*variable*)
- `?thesis` (*variable*)
- `arbitrary` (*modifier*)
- `case` (*command*)
- `cases` (*method*)
- current facts
- finding theorems
- `induction` (*method*)
- intro/elim rules
- `next` *command*
- `of` (*attribute*)
- `rule` (*method*)
- `split` (*modifier*)
- `then` (*command*)
- `this` (*fact*)