Preuves Interactives et Applications

Burkhart Wolff

https://www.lri.fr/~wolff/teach-material/2017-18/M2-CSMR/index.html

Université Paris-Saclay

Induction, Induction and Induction

Outline

- Inductive Sets and lfp-Fixed Points
- (Inductive) Datatypes
- Induction forms in logics and Isabelle/Isar

Command Inductive Set

- Inductively Defined Sets:

```
inductive_set      <c> :: " τ set"  [for A::τ]
            where  <thmname> : "<ϕ>"
                   | ...
                   | <thmname> = <ϕ>
```

```
inductive      <c> :: "  τ ⇒ bool" for A::τ
      where  <thmname> : "<ϕ>"
                   | ...
                   | <thmname> = <ϕ>
```

example:

```
inductive_set     Even :: "int set"
          where   null: "0 ∈ Even"
                  | plus:"x ∈ Even ⟹ x+2 ∈ Even"
                  | min :"x ∈ Even ⟹ x-2 ∈ Even"
```

Command Inductive Set

- These are not buit-in constructs, rather they are based on a series of definitions and typedefs.
- The machinery behind is based on a fixed-point combinator on sets :

$$\text{lfp} :: \text{“}('\alpha\ \text{set} \Rightarrow '\alpha\ \text{set}) \Rightarrow '\alpha\ \text{set}\text{”}$$

which can be conservatively defined by:

$$\text{"lfp } f = \bigcap \{u.\ f\ u \subseteq u\}\text{"}$$

and which enjoys a constrained fixed-point property:

$$\text{mono } f \Longrightarrow \text{lfp } f = f\ (\text{lfp } f)$$

Command Inductive Set

- Example : Even (see before)
  - the set Even is conservatively defined by:

$$\text{Even} = \text{lfp}\ (\lambda X.\ \{0\} \cup (\lambda x.\ x + 2)\ `\ X \cup (\lambda x.\ x - 2)\ `\ X)$$

  - from which the properties:

null: "0 $\in$ Even"
plus:"x $\in$ Even $\Longrightarrow$ x+2 $\in$ Even"
min :"x $\in$ Even $\Longrightarrow$ x-2 $\in$ Even"

can be derived automatically (Note that Isabelle/HOL Version 2016 proceeds differently for technical reasons)

Command Inductive Set

- Example : Even (see before)
  - More important: it derives an

    Induction scheme

    for the Even set.
  - That is: if we know that
    - some x is in Even
    - and some property P over some arbitrary a is maintained (invariant) for a+2 and a-2
    - P x holds.

Command Inductive Set

- Example : Even (see before)
  - In Textbooks on Natural Deduction (like van Dalens Book) we might find this formalized in:

$$\frac{x \in Even \quad P(0) \quad \begin{array}{c}[a \in Even; P(a)]_a \\ \vdots \\ P(a+2)\end{array} \quad \begin{array}{c}[a \in Even; P(a)]_a \\ \vdots \\ P(a-2)\end{array}}{P(x)}$$

  - Note that a is free and does only occur in these sub-proof-trees

Command Inductive Set

- Example : Even (see before)
  - Isabelle derives this as theorem from the lfp definition and displays it as follows:

$$[\![x \in Even; P\ 0; \bigwedge x.\ [\![x \in Even; P\ x]\!] \Longrightarrow P\ (x + 2); \bigwedge x.\ [\![x \in Even; P\ x]\!] \Longrightarrow P\ (x - 2)]\!]$$

$$\Longrightarrow P\ x$$

  - or equivalently:

$$x \in Even$$
$$\Longrightarrow P\ 0$$
$$\Longrightarrow \bigwedge x.\ [\![x \in Even; P\ x]\!] \Longrightarrow P\ (x + 2)$$
$$\Longrightarrow \bigwedge x.\ [\![x \in Even; P\ x]\!] \Longrightarrow P\ (x - 2)$$
$$\Longrightarrow P\ x$$

Command Inductive Set

- Example : Even (see before)
  - or equivalently:

assumes “$x \in$ Even”

and base: “P 0”

and step1: “$\bigwedge x. \llbracket x \in \text{Even}; P\ x \rrbracket \Longrightarrow P\ (x + 2)$“

and step2: “$\bigwedge x. \llbracket x \in \text{Even}; P\ x \rrbracket \Longrightarrow P\ (x - 2)$“

shows “P x”

Command Inductive Set

- Remarks
  - Induction schemes (closely related to fixpoints, recursion, and while-loops) are the major weapon in HOL proofs that can NOT be done by automated provers
  - they can refer to (inductive) datatypes, sets and therefore relations and are always the means of choice if we want to express that something is „closed under a set of rules"
  - Usually there are several choices of induction schemes, their instantiation, and the target they are applied on.
  - Like invariants of while-loops, it may be that some generalization of a property can be proven inductively, the concrete property, however, not directly.

Command Inductive Set

- Inductively Defined Sets:

```
   inductive    <c> [ for <v>:: "<τ>" ]
     where  <thmname> : "<ϕ>"
                    | ...
                    | <thmname> = <ϕ>
```

example: inductive path for rel ::"'a ⇒ 'a ⇒ bool"
where base : "path rel x x"
| step : "rel x y ⟹ path rel y z ⟹ path rel x z"

Command Inductive Set

- Inductively Defined Sets:

```
inductive    <c> [ for <v>:: "<τ>" ]
  where  <thmname> : "<φ>"
               | ...
               | <thmname> = <φ>
```

NOTE: Isabelle HOL compiles this internally to axiomatic definitions, i.e. a "model" in HOL!!!

example: inductive path for rel ::"'a ⇒ 'a ⇒ bool"
where base : "path rel x x"
| step : "rel x y ⟹ path rel y z ⟹ path rel x z"

Command Inductive Set

- Inductively Defined Sets: Example path. Isabelle/HOL:

  path rel x y
  $\Longrightarrow \bigwedge$x. P x x;
  $\Longrightarrow \bigwedge$x y z. ⟦rel x y; path rel y z; P y z⟧ $\Longrightarrow$ P x z
  $\Longrightarrow$ P x y

- Text-book:

$$\dfrac{path\ rel\ x\ y \quad [P\ a\ a]_a \quad \begin{matrix}[rel\ a\ b; path\ rel\ b\ c; P\ b\ c]_{a,b,c} \\ \vdots \\ P\ a\ c\end{matrix}}{P\ x\ y}$$

Command Inductive Set

- Note: an equivalent (appending) induction scheme with the same power:

path rel x y

$\Longrightarrow (\bigwedge x.\ P\ x\ x)$

$\Longrightarrow (\bigwedge x\ y\ z.\ [\![\ path\ rel\ x\ y;\ P\ x\ y;\ rel\ y\ z\ ]\!] \Longrightarrow P\ x\ z)$

$\Longrightarrow P\ x\ y$

- The choice of the induction scheme matters for the task ahead …

Command Inductive Datatype

- Datatype Definitions (similar SML):
  (Machinery behind : complex series of const and typedefs !)

datatype ('a_1..'a_n) Θ =
<c> :: "<τ>" | ... | <c> :: "<τ>"

- Recursive Function Definitions:
  (Machinery behind: Veeery complex series of const and typedefs and automated proofs!)

fun <c> ::"<τ>" where
"<c> <pattern> = <t>"
| ...
| "<c> <pattern> = <t>"

Command Inductive Datatype

- Datatype Definitions (similar SML):
  (Machinery behind : complex !)

```
datatype ('a1..'an) Θ =
 <c> :: "<τ>"  | ... |  <c> :: "<τ>"
```

- Recursive Function Definitions:
  (Machinery behind: Veeery complex!)

```
fun <c> ::"<τ>" where
    "<c> <pattern> = <t>"
 | ...
 |  "<c> <pattern> = <t>"
```

NOTE: Isabelle HOL compiles this internally to conservqtive definitions, i.e. a "model" in HOL!!!

Command Inductive Datatype

- Example: Induction Scheme from Datatype Definitions
  - $(\bigwedge a.\ P\ (leaf\ a))$

    $\Longrightarrow (\bigwedge a\ t\ t'.\ P\ t \Longrightarrow P\ t' \Longrightarrow P\ (node\ a\ t\ t'))$

    $\Longrightarrow P\ tree$
  - Textbook:

$$\frac{[P(leaf\ a)]_a \qquad \begin{matrix}[P\ t; P\ t']_{a,t,t'} \\ \vdots \\ P(node\ a\ t\ t')\end{matrix}}{P\ tree}$$

Command Inductive Datatype

- Example: Recursive Function Definition

```
fun reflect :: "'a tree ⇒ 'a tree"
   where  a : "reflect (leaf x) = leaf x"
          | b : "reflect (node x t t') = node x t' t"
```

- Example Proof: lemma “reflect(reflect t) = t”:
  - Proof by induction (apply style; since tree.induct is just an ordinary (introduction) rule, *this works by rule*)

```
apply(rule_tac tree=t in tree.induct)
 apply(simp add: a)
apply(simp add: b)
done
```

Induction vs. Case-Split

- The commands inductive, inductive_set and datatype generate another important schema of rules which is an important weapon:

Case-Splits

- Most basic form: disjE

$$\frac{A \vee B \qquad \begin{matrix}[A] \\ \vdots \\ Q\end{matrix} \qquad \begin{matrix}[B] \\ \vdots \\ Q\end{matrix}}{Q}$$

Induction vs. Case-Split

- For the datatype tree, this rule present itself like this:

$$(\bigwedge a.\ y = \text{leaf } a \Longrightarrow Q)$$

$$\Longrightarrow (\bigwedge x\ t\ t'.\ y = \text{node } x\ t\ t' \Longrightarrow Q)$$

$$\Longrightarrow Q$$

$$\frac{\begin{matrix}[x = (leaf\ a)]_a & & [x = node\ a\ t\ t']_{a,t,t'} \\ \vdots & & \vdots \\ Q & & Q\end{matrix}}{Q}$$

Induction vs. Case-Split

- For the inductive sets, the case split rule path.cases presents itself like this:

$$
\begin{array}{l}
[\![\text{path rel a1 a2};\\
\;\bigwedge x.\qquad [\![a1 = x;\ a2 = x]\!] \Longrightarrow P;\\
\;\bigwedge x\ y\ z.\ [\![a1 = x;\ a2 = z;\\
\qquad\qquad \text{rel } x\ y;\ \text{path rel } y\ z]\!] \Longrightarrow P\\
]\!] \Longrightarrow P
\end{array}
$$

Induction and Case-Splitting Support

- induction and case-splitting were supported by specific methods attempting to figure out automatically which rule to use
- There are apply-style proof methods:

```
apply(induct_tac „<term>“)
```

```
apply(case_tac „<term>“)
```

which work with arbitrary open parameters of a subgoal …

Induction and Case-Splitting Support

- induction and case-splitting were supported by specific methods attempting to figure out automatically which rule to use
- There are "blue-style" proof methods giving support for an own structured proof-language Isar

```
apply(induct „<term>“ <options ... >)
```

```
apply(case „<term>“)
```

which require that parameters are "fixed".

Introduction to Isar Advanced Proof Techniques

- A language for structured proofs:

  Isar - Intelligible semi-automated reasoning

- http://isabelle.in.tum.de/Isar/
- supporting a declarative proof-style (rather than a procedural one)
- presenting intermediate steps in a machine-checked, human readable format

Introduction to Isar
Advanced Proof Techniques

- Core: the proof environment:

```
proof (<method>)
  [case - fix - assumes - defs- have-]
  show "<goal>" <proof>
next
   ...
next
   [case - fix - assumes - defs- have-]
  show "<goal>" <proof>
qed
```

- … a switch from procedural to declarative style can be done by rephrasing the goals

Introduction to Isar Advanced Proof Techniques

- Instead of the goal format:

$$\bigwedge a_1 \ldots a_n.\ A_1 \Longrightarrow \ldots A_m \Longrightarrow P$$

the "ISAR"-format:

fix a_1::<typ> ... fix a_n::<typ>
assume A_1 and ... and A_m
show P

is preferable (better labelling, control of goal parameters, intermediate steps "have", abbreviations, pattern-matching, support for cases, ...)

Introduction to Isar
Advanced Proof Techniques

- The methods induct and cases produce a list of local contexts (shown by the diagnostic command print_cases) with the appropriate fix'es and assume's
- Example:

```
 lemma "reflect(refect t) = t"
proof(induct t) print_cases
  case (leaf x) then show ?case sorry
next
  case (node x1a t1 t2) then show ?case sorry
qed
```

Conclusion

- Induction is at the heart of interactive proving; this requires the most human ingenuity
- Isabelle offers support for inductive and case-distinction based proofs
- the ISAR-language paves the way for adequate presentation of common proof-structures (by induction, by case distinction,…)
- … and by the way, ISAR paved the way for better portability and parallel proof-checking