# Preuves Interactives et Applications

Burkhart Wolff

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

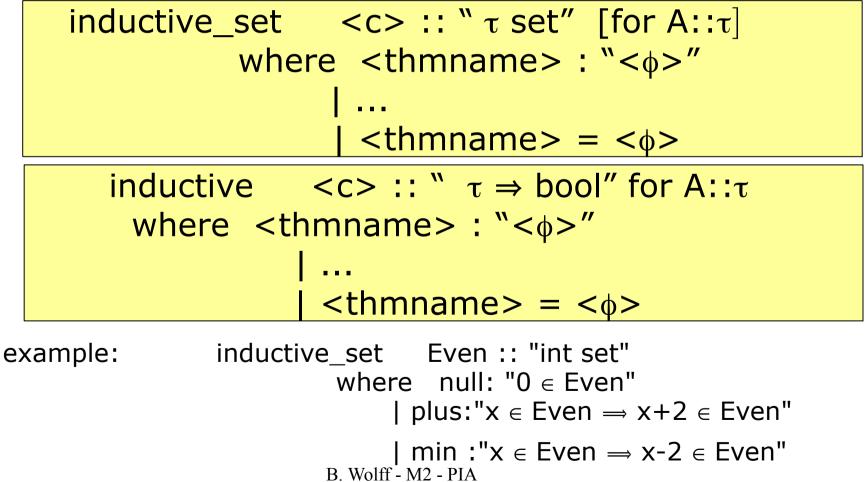
Université Paris-Saclay

#### Induction, Induction and Induction

### Outline

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

• Inductively Defined Sets:



- 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 :

If  $\mu :: (\alpha \text{ set} \Rightarrow \alpha \text{ set}) \Rightarrow \alpha \text{ set}^{"}$ 

which can be conservatively defined by:

"Ifp f = 
$$\bigcap \{u. f u \subseteq u\}$$
"

and which enjoys a constrained fixed-point property:

mono 
$$f \implies lfp f = f (lfp f)$$

• Example : Even (see before)

- the set Even is conservatively defined by:

- from which the properties:

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

can be derived automatically (Note that Isabelle/HOL Version 2016 proceeds differently for technical reasons) B. Wolff - M2 - PIA 5

• 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.

B. Wolff - M2 - PIA

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

$$\begin{bmatrix} a \in Even; P(a) \end{bmatrix}_a & \begin{bmatrix} a \in Even; P(a) \end{bmatrix}_a \\ \vdots & \vdots \\ P(a+2) & P(a-2) \\ \hline P(x) \\ \hline \end{bmatrix}$$
- Note that a is free and does only occur in

these sub-proof-trees

09/25/19

B. Wolff - M2 - PIA

• Example : Even (see before)

- Isabelle derives this as theorem from the lfp definition and displays it as follows:

$$\llbracket x \in Even; P 0; \land x. \llbracket x \in Even; P x \rrbracket \implies P (x + 2); \land x. \llbracket x \in Even; P x \rrbracket \implies P (x - 2) \rrbracket$$

$$\implies$$
 P x

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

• Example : Even (see before)

- or equivalently:

assumes " $x \in Even$ " and base: "P 0" and step1: " $\bigwedge x$ . [[ $x \in Even$ ; P x]]  $\Rightarrow$  P (x + 2)" and step2: " $\bigwedge x$ . [[ $x \in Even$ ; P x]]  $\Rightarrow$  P (x - 2)" shows "P x"

- Remarks
  - Induction schemes (closely related to fixpoints, recursion, and while-loops) are the major weapon in HOL proofs that can NO<sup>-</sup> 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.

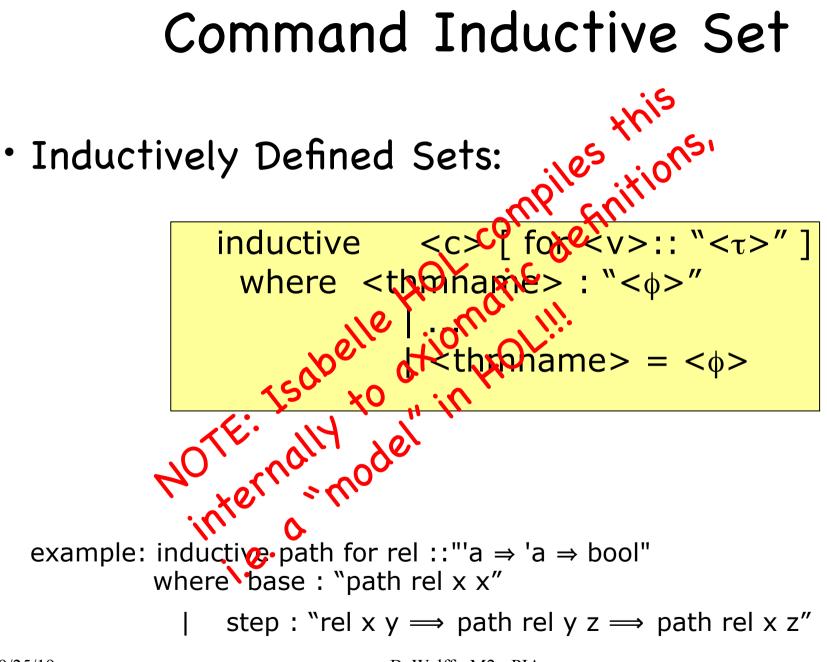
• Inductively Defined Sets:

inductive 
$$\langle c \rangle [ \text{ for } \langle v \rangle :: `` \langle \tau \rangle'' ]$$
  
where  $\langle thmname \rangle : `` \langle \phi \rangle''$   
 $| \dots$   
 $| \langle thmname \rangle = \langle \phi \rangle$ 

example: inductive path for rel ::"'a 
$$\Rightarrow$$
 'a  $\Rightarrow$  bool"  
where base : "path rel x x"

| step : "rel x y  $\implies$  path rel y z  $\implies$  path rel x z"

B. Wolff - M2 - PIA



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

path rel x y  

$$\Rightarrow \bigwedge x. P x x;$$
  
 $\Rightarrow \bigwedge x y z. [[rel x y; path rel y z; P y z]]  $\Rightarrow P x z$   
 $\Rightarrow P x y$$ 

• Textbook:

09/25/19

$$\begin{bmatrix} rel \ a \ b; path \ rel \ b \ c; P \ b \ c \end{bmatrix}_{a,b,c}$$

$$\vdots$$

$$P \ x \ y$$

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

path rel x y

 $\implies$  ( $\land$ x. P x x)

 $\implies (\land x \ y \ z. \ [ path rel x y; P x y; rel y z ] Px z)$  $\implies P x y$ 

• The choice of the induction scheme matters for the task ahead ... <sup>09/25/19</sup>

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

```
datatype ('a_1..'a_n) \Theta = 
<c> :: "<\tau>" | ... | <c> :: "<\tau>"
```

 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>"
```

 Datatype Definitions (similar SML): (Machinery behind : complex !) datatype ('a<sub>1</sub> <c> :: · Recursive Function Definitio (Machinery behind: Veeevy complex!)  $<\tau>''$  where <pattern> = <t>" "<c> <pattern> = <t>"

• Example: Induction Scheme from Datatype Definitions

 $\begin{array}{ll} - & (\land a. \ \mathsf{P} \ (\mathsf{leaf} \ a)) \\ \implies \land (\land a \ \mathsf{t} \ '. \ \mathsf{P} \ \mathsf{t} \implies \mathsf{P} \ \mathsf{t} \implies \mathsf{P} \ (\mathsf{node} \ a \ \mathsf{t} \ ')) \\ \implies \ \mathsf{P} \ \mathsf{tree} & \left[ P \ t; P \ t' \right]_{a,t,t'} \\ \vdots \\ \left[ P(\mathit{leaf} \ a) \right]_a & P(\mathit{node} \ a \ t \ t') \end{array}$ 

B. Wolff - M2 - PIA

• 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"

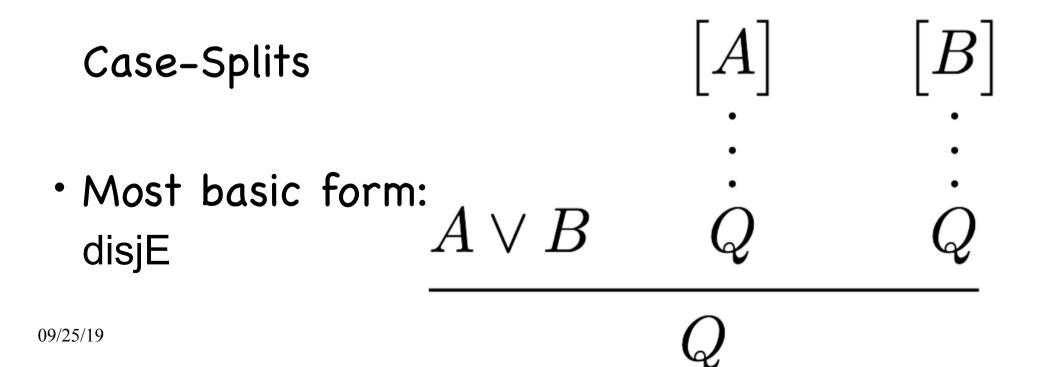
• Example Proof: Iemma "reflect(reflect t) = t":

 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:



#### Induction vs. Case-Split

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

#### Induction vs. Case-Split

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

$$\begin{bmatrix} \text{path rel a1 a2;} \\ \land x. & [[a1 = x; a2 = x]] \implies P; \\ \land x \ y \ z. & [[a1 = x; a2 = z; \\ & rel \ x \ y; \ path \ rel \ y \ z]] \implies P \\ \end{bmatrix} \implies P$$

## 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 ... >)

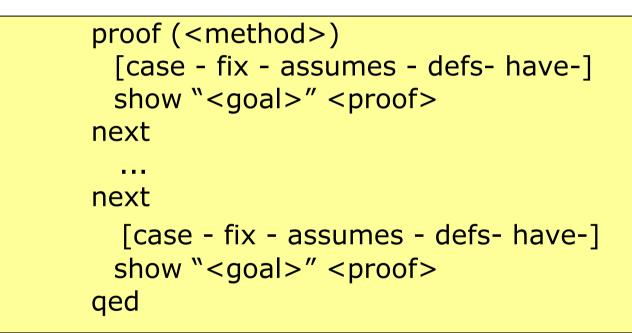
apply(case "<term>")

which require that parameters are "fixed".

B. Wolff - M2 - PIA

- A language for structured proofs: Isar – Intelligible semi-automated reasoning
- http://isabelle.in.tum.de/lsar/
- supporting a declarative proof-style (rather than a procedural one)
- presenting intermediate steps in a machine-checked, human readable format

• Core: the proof environment:

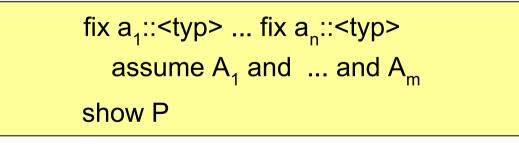


 ... a switch from procedural to declarative style can be done by rephrasing the goals

• Instead of the goal format:

$$\bigwedge a_1 \dots a_n \therefore A_1 \Longrightarrow \dots A_m \Longrightarrow P$$

the "ISAR"-format:



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

- 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 proofstructures (by induction, by case distinction,...)
- ... and by the way, ISAR paved the way for better portability and parallel proof-checking