# Introduction to Interactive Proof of Software

J.-F. Monin

Univ. Joseph Fourier and
LIAMA-FORMES, Tsinghua Univ., Beijing

2012, Semester 1

Lecture 6

# Outline

# Outline

# Fixpoints



### Recursive calls

must be on a structurally smaller argument.

### Available for all inductive types

Not only natural numbers

### Induction is a special case of a fixpoint

Not only natural numbers
Computational interpretation
More secure
Subtleties on quantification

# Syntax of fixpoints



Consider a recursive function `f` with arguments `x`... `z`, including `y`

```
Fixpoint f (x:A)...(z:C) {struct y}: R :=
  ...
  match y with
    ...
    | Construct...y'... => ... (f...y'...) ...
    ...
  end
  ...
```

However, `{struct y}` can be omitted:
Coq tries to guess which is the structurally decreasing argument from the body of `f`

# Subtle inductions





Proofs by induction may need a strengthening of the statement

- additional conjuncts
- put more quantifications $\forall$ in the scope of the induction

# Outline

# Informative data types



## Informative Booleans: sumbool

```
Inductive sumbool (P Q: Prop) : Set :=
  | left  : forall p:P, sumbool P Q
  | right : forall q:Q, sumbool P Q.
```

Notation : `{P}+{Q}`

## Qualified values: sig

```
Inductive sig (A : Type) (P : A -> Prop) : Type :=
    exist : forall x : A, P x -> sig P.
```

Notation : `{x:A | P x}`

# Pragmatics of informative data types





Corresponding counterparts in Prop

| logic | data types |
|---|---|
| $P \vee Q$ | $\{P\} + \{Q\}$ |
| $\exists x, P\ x$ | $\{x : A \mid P\ x\}$ |

Easier to construct and to use in interactive mode

# Differences between Prop and Set (1)





In general, we don't care about normal form of proofs

E.g. in {x:nat | even x},
consider $(20 \times 15, p)$, where $p$ is a proof that $20 \times 15$ is even

- $20 \times 15$ reduces to $300$:
  useful, e.g., we may want to compute `pred` $(20 \times 15)$
- $p$ may rely on a lemma saying that $n \times m$ is even if $n$ is even; reducing $p$ to the constructors of `even` has no special interest

# Differences between Prop and Set (2)



### Bottom line

Case analysis on p:P:Prop to get a value in A:Set

**is not allowed**

### Can be read as confidentiality

The information contents of proofs in Prop is secret:

- it is visible only in other proofs in Prop
- it is hidden to the world of datatypes and computations Set (and Type)

# Differences between Prop and Set (3)



### Advanced (not discussed here)

Prop is impredicative while Set may be predicative

# Excluded middle



A consequence of the computational reading of disjunction

## Constructive (intuitionistic) logic

- $P \lor \neg P$ is not a theorem
- $\neg\neg(P \lor \neg P)$ is a theorem
- similar for $\{P\} + \{\neg P\}$

## Examples

- $\forall n\ m : nat,\ \{n = m\} + \{\neg n = m\}$ OK... with work
- $\forall f : nat \to nat,\ \{\exists n,\ f\ n = 0\} + \{\forall n, \neg f\ n = 0\}$
  just impossible

## Notes

- $\forall n, \neg P\ n$ is equivalent to $\neg \exists n, P\ n$
- $\forall f\ g : nat \to nat,\ f = g \lor \neg f = g$

# More on Excluded middle



## Admissible axioms

- $P \vee \neg P$ is admissible:
  `Require Import Classical.`
  Can be convenient, but often stronger than really needed
  Matter of taste...
- $\{P\} + \{\neg P\}$ is not admissible
  Consistent with confidentiality (see above)

# Outline

# Why induction matters



## Tool of choice for proving properties on an infinite (but countable) number of values

Other methods are

- either weaker (prove less properties)
- or rely on induction in a hidden way

## Required in many applications in computer science

- reasoning on data structures
- language syntax
- programming language semantics
- proofs of algorithms

# Strength of induction

Induction requires ingenuity, in general

- a consequence of Gödel incompleteness theorems
- support for induction is a discriminating criterium for automated provers
  Coq supports induction
- proof search $\neq$ proof checking

# Several forms of induction

- Basic induction on natural numbers ($\mathbb{N}$)
- Well-founded induction on $(\mathbb{N}, <)$
- Well-founded induction on $(S, R)$, where $S$ is an arbitrary set and $R$ a suitable relation on $S$
- Transfinite induction
- Structural induction

We will focus on structural induction, because it is

- a very natural extension of basic induction but on lists, trees, terms ... instead of $\mathbb{N}$
- close to computer science concerns
- yet powerful enough to embed all other kinds of induction

# Proving something on all natural numbers





Let us define $x \leq y \stackrel{\text{def}}{=\!=} \exists d, d + x = y$

Prove $\forall x, 2 + x \leq 5 + x$

- Take an arbitrary natural number $x$
- Remark that $3 + (2 + x) = 5 + x$
- Hence $\exists d, d + (2 + x) = 5 + x$
- By definition of $\leq$ we get: $2 + x \leq 5 + x$

This proof is uniform : it does not depend on the value of $x$

# Looking at $x$ : (non-uniform) proof by cases



Prove $\forall x, x \leq 4 \Rightarrow \exists y, x = 2y \vee x = 1 + 2y$

The proof is not uniform: different is each case

- Case $x = 0$: take $y = 0$, left, check $0 = 2.0$
- Case $x = 1$: take $y = 0$, right, check $1 = 1 + 2.0$
- Case $x = 2$: take $y = 1$, left, check $2 = 2.1$
- Case $x = 3$: take $y = 1$, right, check $3 = 1 + 2.1$
- Case $x = 4$: take $y = 2$, left, check $4 = 2.2$
- Case $x = 5 + n$: don't care

# What do you think of the following one?



$x \leq y \stackrel{\text{def}}{=\!=} \exists d, d + x = y$

Prove $\forall x, x \leq 3x$

- Take an arbitrary natural number $x$
- Remark that $2x + x = 3x$
- Hence $\exists d, d + x = 3x$
- That is $x \leq 3x$

Is this proof uniform? Yes: no case analysis on $x$

# Common scheme for a proof by cases on nat



## Basic scheme

$$\frac{P\,0 \qquad \forall n, P\,(S\;n)}{\forall x, P\,x}$$

## Variants

$$\frac{P\,0 \qquad P\,1 \qquad \forall n, P\,(S\,(S\;n))}{\forall x, P\,x}$$

$$\frac{P\,0 \qquad P\,1 \qquad P\,2 \qquad \forall n, P\,(S\,(S\,(S\;n)))}{\forall x, P\,x}$$

etc.

# Proof by cases on all natural numbers

$$\frac{P\,0 \quad P\,1 \quad \ldots \quad P\,n \ldots}{\forall x, P\,x}$$

*In order to prove* $\forall x, P\,x$*,*
*prove* $P$ *on each natural number* $n$

$\infty$ cases to consider

Does not work...

Unless we have a systematical way to construct a proof of $P\ n$ for each $n$?

# Constructing proofs of $P\ n$, with $n : nat$

1. Prove $P\ 0$
2. Prove $P\ 0 \Rightarrow P\ 1$
3. Prove $P\ 1 \Rightarrow P\ 2$
4. etc.

From 1. and 2. we get $P\ 1$
From the latter and 3. we get $P\ 2$
Etc.

At first sight, no progress:
infinite number of proof obligations

Unless ve prove (uniformly) 2. 3. 4. etc. at once:

$$\forall n,\ P\ n \Rightarrow P\ (S\ n)$$

# Induction on nat

$$\frac{P\,0 \qquad \forall n, P\ n \Rightarrow P\,(S\ n)}{\forall n, P\ n}$$

*P n* is called the *induction hypothesis*.

Remark: proof by cases

$$\frac{P\,0 \qquad \forall n, P\,(S\ n)}{\forall n, P\ n}$$

is a special case of induction – the induction hypothesis is not used.

# Primitive recursion



## Example: addition

Given some fixed natural $m$, what is to "add to $m$"?

- $0 + m = m$
- $S\ n + m = S(n + m)$

## Method for defining such functions $f$

- provide the returned value when the argument is $0$
- provide the returned value when the argument is $S\ n$
  this value may depend on $n$ and on $f\ n$

Note that $f$ may have other fixed arguments

Official name in the jargon of logic : *primitive recursion*

# Properties of $+$



(Almost all) basic properties of $+$ are proved by induction

- $\forall n, 0 + n = n \quad \ldots?$
- $\forall n, n + 0 = n \quad \ldots?$

Commutativity, associativity

Similarly for subtraction, multiplication…

Interest: foundations (Coq library); fundamental exercises

# Constructive (i.e. functional) reading

A proof of $\forall n,\ P\ n \Rightarrow P\,(S\ n)$ is a function which, given 2 arguments:

- a nat $n$
- a proof $p_n$ of $P\ n$

yields a proof of $P\,(S\ n)$

Let $f$ be such a proof.
Let $p_0$ be a proof of $P\,0$

Then

- $f\,1\,(f\,0\,p_0)$ is a proof of $P\,2$
- given any nat $n$, $f\ n\,(\ldots(f\,1\,(f\,0\,p_0))\ldots)$ is a proof of $P\,(S\ n)$

# Example: the product of 2 consecutive numbers is even

Formally: $\forall n, \underbrace{\exists k, n.(S\ n) = 2.k}_{P\ n}$

- For $n = 0$: we have $n.(S\ n) = 0.1 = 0 = 2.0$, taking $k = 0$ yields $P\ 0$
- (Uniform) proof of $\forall n,\ P\ n \Rightarrow P(S\ n)$
  - For an arbitrary $n \in nat$, assume $P\ n$ i.e. $n.(S\ n) = 2.y$ for some $y$
  - Then $$\begin{aligned}(S\ n).(S\ (S\ n)) &= (2 + n).(S\ n) \\ &= 2.(S\ n) + 2.y \\ &= 2.(S\ n + y)\end{aligned}$$
  - Taking $k = S\ n + y$, we get $P(S\ n)$, QED.

# Constructive (i.e. functional) reading

A proof of $\exists x, P\ x$ is a pair `(ex_intro w p)`,
written $(w, p)$ for short,
where $w$ is a value (the witness) and $p$ a proof of $P\ w$

Let $g$ be the previous proof of $\forall n, \underbrace{\exists k, n.(S\ n) = 2.k}_{P\ n}$

which uses $f$, a proof of $\forall n,\ P\ n \Rightarrow P(S\ n)$

Reducing a proof of $g$ 10 yields
$f\ 9\,(f\ 8\,(\ldots(f\ 0\ p_0)\ldots)$

which reduces to $(55, e_{110})$:

- $p_0 = (0, e_0)$
- $p_1 = f\ 0\ p_0$ reduces to $(1, e_2)$
- $p_2 = f\ 1\ p_1$ reduces to $(3, e_6)$
- ...

Where $e_i : i = i$ which reduces to reflexivity of equality on $i$

# Constructive reading in Set

However, reductions are not performed in Prop
(except for theorems finishing with **Defined** instead of **Qed**)

Using the existence in Set:
A proof of $\{x \mid P\ x\}$ is a pair `(exist w p)`,
written $(w, p)$ for short,
where $w$ is a value (the witness) and $p$ a proof of $P\ w$

Let $g$ be the previous proof of $\forall n, \underbrace{\{k \mid n.(S\ n) = 2.k\}}_{P\ n}$

which uses $f$, a proof of $\forall n,\ P\ n \Rightarrow P(S\ n)$

Reducing a proof of $g\ 10$ yields
$f\ 9\,(f\ 8\,(\ldots(f\ 0\ p_0)\ldots)$

which reduces to $(55, e_{110})$

The proof $e_i$ reduces, in principle, to reflexivity of equality on $i$,
but reductions are not performed there (but we don't care)

# About excluded middle



## In Prop

A proof of $\forall n, \underbrace{\texttt{even}\ n \lor \neg\texttt{even}\ n}_{P\ n}$

is a function $f$ which provides for each $n$ a precise answer:

- either yes: $n$ is even, here is a proof
- or no: $n$ is not even, here is a proof

E.g., reducing $f\ 10$ will answer: yes + proof of $\texttt{even}\ 10$

## 2 possibilities

- Cheating, using classical logic: $\forall P, P \lor \neg P$
- Really provide a proof, by induction on $n$

## In Set: testing functions returning additional knowledge

A proof of $\forall n, \underbrace{\{\texttt{even}\ n\} + \{\neg\texttt{even}\ n\}}_{P\ n}$ must be constructive

Excluded middle not allowed

# Outline

# Subtelties with induction



Consider the following version of addition
*Coq syntax for function application, see below why*

- $addt\ 0\ m = m$
- $addt\ (S\ n)\ m = addt\ n\ (S\ m)$

*Beyond primitive recursion, see explanation below*

Prove $addt\ n\ m = n + m$ forall $n$ and $m$

## First try

Prove $addt\ n\ m = n + m$ by induction on $n$
(Previous model) $\rightarrow$ Fails

## Second try

Prove $\forall m, addt\ n\ m = n + m$ by induction on $n$
Works

# Explanations on *addt*

- $addt\ 0\ m = m$
- $addt\ (S\ n)\ m = addt\ n\ (S\ m)$

Means

- $addt\ 0\ =\ fun\ m \Rightarrow m$
- $addt\ (S\ n)\ =\ fun\ m \Rightarrow addt\ n\ (S\ m)$

Official name in the jargon of logic :

*higher order primitive recursion*

# More advanced example (homework)

- $fib\ 0 = 1$
- $fib\ 1 = 1$
- $fib\ (S\ (S\ n)) = fib\ n + fib\ (S\ n)$

*Harmless shorthand for a truly primitive recursion, where we define $fib\ n$ and $fib\ (S\ n)$ at the same time.*

- $lfib\ 0\ a\ b = a$
- $lfib\ (S\ n)\ a\ b = lfib\ n\ b\ (a + b)$

Prove $\forall n, lfib\ n\ 1\ 1 = fib\ n$.

# Outline

# What if there is no zero?



## On nat

```
Inductive nat : Set :=
  |  O :  nat
  |  S :  nat -> nat.
```

$$\frac{P\ \mathtt{O} \qquad \forall n, P\ n \rightarrow P\,(\mathtt{S}\ n)}{\forall x, P\ x}$$

## On wrongnat

```
Inductive wrongnat : Set :=
  |  Swn :  wrongnat -> wrongnat.
```

$$\frac{\forall n, P\ n \rightarrow P\,(\mathtt{Swn}\ n)}{\forall x, P\ x}$$

# Interpretation





A value in an inductive type
is made with finitely many constructors

- A nat comes from `O`
- A wrongnat comes from nowhere
  The conclusion of

$$\frac{\forall n, P\ n \to P\,(\texttt{Swn}\ n)}{\forall x, P\ x}$$

  can only be applied to some wrongnat
  But assuming such a value is inconsistent !
- Application: take for $P$ the predicate constantly false:
  fun $n \to$ False