Introduction to Interactive Proof of Software

J.-F. Monin

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

2012, Semester 1

Lecture 4

Outline

Outline

Another way to look at definitions and types

```
Definition funny :
  forall (r: rgb), Set_of r :=
  fun (r: rgb) => some body
```

```
Theorem plus_id_example :
  ∀ n m:nat, n = m -> n + n = m + m.
```

Or, equivalently:

```
Theorem plus_id_example :
  ∀ n m:nat, ∀ e:n = m,  n + n = m + m.
```

Its proof is a function

- taking as arguments n, m and e a proof of $n = m$
- returning a proof of $n + n = m + m$

Proofs are trees!

Theorems are just definitions

Hypotheses are just variables

The type of propositions is called Prop

Example: $3 = 2 + 1 :$ Prop

WARNING

Prop is at the same level as Set, **not** bool

Some subtle differences between Prop and Set to be discussed later

Correspondance


```
Section my_propositional_logic.
Variables P Q: Prop.
Inductive P_or_Q: Prop :=
  | P_or_Q_intro_left  : forall p:P, P_or_Q
  | P_or_Q_intro_right : forall q:Q, P_or_Q.
```

We have

`P_or_Q_intro_left : P_or_Q`	`P_or_Q : Prop`
`true : bool`	`bool : Set`

`P_or_Q` is like `bool`:

- *Enriched* version of `bool`, where each constructor embeds an *additional proof tree*
- Minor difference: it is in `Prop` instead of `Set`

Parameterized inductive types

An inductive type may have parameters as follows:

```
Inductive list (A Set) : Set :=
  | Nil  : list A
  | Cons : forall (h:A) (t:list A), list A
.
```

Full definition of disjunction (standard library)

```
Inductive or (P Q: Prop) : Prop :=
  | or_intro_left  : forall p:P, or P Q
  | or_intro_right : forall q:Q, or P Q
.
```

Next, instead of or P Q, use the usual infix notation P \/ Q

Curry-Howard

Logic	Proposition	Proof	Lemma inlining
Programming	Type	Term	Reduction

A little bit of history

In the 20th century, logic and functionnal programming were developed separately

Actually the same ideas have been discovered twice with different names

Curry-Howard in practice

Logic	$\vee$	$\wedge$	$\forall \ \rightarrow$	`False`
Programming	Sum	product	function	empty

Note: the negation $\neg$`P` of a proposition `P` is defined as `P` $\rightarrow$ `False`. For instance, $\neg$`False` is easy to prove...

Correctness proofs of functions follow their shape

match $\longrightarrow$ case or destruct
fixpoint $\longrightarrow$ induction or fix

Choose convenient definitions
$1 + n$ or $S\ n$ better than $n + 1$

Outline

Propositions and proofs

More Logic

Special Propositions


```
Inductive True: Prop :=
  | I  : True.

Inductive False: Prop := .
```

Existential Quantifier


```
Inductive ex (A : Type) (P : A -> Prop) : Prop :=
  | ex_intro : forall x : A, P x -> ex P
```

A proof of $\exists x : A, P\ x$ is a pair made of

- a witness x
- a proof of $P\ x$

Selection of values


```
Inductive P248 : nat -> Prop :=
  | is2 : P248 2
  | is4 : P248 4
  | is8 : P248 8.
```

Elimination principle?
$P\ 2 \to P\ 4 \to P\ 8 \to \forall n, P248\ n \to P\ n$

Remark

- (P248 2) has a proof – it is like True
- similar for 2 and 4
- (P248 1) has no proof – it is like False
  but not that easy