The Coq proof assistant : principles and practice

J.-F. Monin

Université Grenoble Alpes

2016

Lecture 4

Outline

Outline

Propositions and proofs

More Logic

More on Prop and Set

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: $3 = 2 + 1 : \text{Prop}$

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

More on Prop and Set

Special Propositions

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

Inductive False: Prop := .
```

- No way to prove False
  in an empty environment
- From False we can get a proof of anything
- From False we can get an element in any type

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

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 20×15 is even

- 20×15 reduces to 300:
  useful, e.g., we may want to compute `pred` (20×15)
- 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