CS720

Logical Foundations of Computer Science

Lecture 8: Logical connectives in Coq

Tiago Cogumbreiro
Today we will...

- Recall the difference between value, type, Type, evidence, proposition, Prop

- Logical connectives in Coq

\[ \top \quad \bot \quad \neg P \quad P \iff Q \quad \exists x. P \]

Why are we learning this?

- The building blocks of any interesting property
Logic.v

Due Thursday, October 4, 11:59 EST
Recall product, conjunction

\begin{verbatim}
Inductive prod (A B : Type) : Type :=
| pair : A → B → prod A B.

Inductive and (P Q : Prop) : Prop :=
| conj : P → Q → and P Q.
\end{verbatim}
Recall product, conjunction

\begin{verbatim}
Inductive prod (A B : Type) : Type :=
| pair : A → B → prod A B.

Inductive and (P Q : Prop) : Prop :=
| conj : P → Q → and P Q.
\end{verbatim}

- P, Q are propositions (instances of Prop)
- A, B, and nat are types (instances of Type)
- A value is any instance of an instance of a Type (eg, 3 is a value)
- An evidence is any instance of an instance of a Prop (eg, if H:P and P:Prop, then H is an evidence)
- pair is a constructor (function) that builds values; conj is a constructor (function) that builds evidence
Recall a proof state

1 subgoal
T : Type
x : T
P : Prop
H1 : 1 = x
H2 : P

1 = 2 \land P

- All hypothesis are variables of a specific type, Type, or proposition
- Goals are (usually) propositions
- Propositions (instances of Prop) can mention values

Can a proposition mention pair, the constructor of prod? Can a proposition mention conj, the constructor of and?
Recall a proof state

1 subgoal
T : Type
x : T
P : Prop
H1 : 1 = x
H2 : P

\[\begin{align*}
1 &= 2 \\
1 &= 1 \land P
\end{align*}\]

- All hypothesis are variables of a specific type, Type, or proposition
- Goals are (usually) propositions
- Propositions (instances of Prop) can mention values

Can a proposition mention pair, the constructor of prod? Can a proposition mention conj, the constructor of and? Yes and no, respectively.
Where do constructors of propositions appear?

Theorem and_conj: forall P Q:Prop, P \rightarrow Q \rightarrow P \land Q.
Proof.
  intros P Q H1 H2.
  apply conj.
  - apply H1.
  - apply H2.
Qed.
Theorems are expressions too

Theorem and_conj: \( \forall P \; Q: \text{Prop}, \quad P \rightarrow Q \rightarrow P \land Q. \)

Proof.
\[ \text{intros} \; P \; Q \; H1 \; H2. \]
\[ \text{apply} \; (\text{conj} \; H1 \; H2). \]
Qed.

Proposition-constructors and theorems are functions whose parameters are evidences.
Truth

T
Truth

Truth can be encoded in Coq as a proposition that always holds, which can be described as a proposition type with a single constructor with 0-arity.

```
```

You will note that proposition `True` is not a very useful one.
Truth example

Goal  True.

(Done in class.)
Falsehood

⊥
So far we only seen results that are provable (eg, plus is commutative, equals is transitive)

How to encode falsehood in Coq?
Falsehood

Falsehood in Coq is represented by an *empty* type.

```
Inductive False : Prop :=.
```

- The only way to reach it is by using the exploding principle
- **No constructors available.** Thus, no way to build an inhabitant of \texttt{False}.
Example:

Goal $1 = 2 \Rightarrow False.$

Goal $False \Rightarrow 1 = 2.$

Goal $False.$

*(Done in class.)*
Negation

$\neg P$
Negation

The negation of a proposition \( \neg P \) is defined as

\[
\neg P = \neg \neg P
\]

(* As defined in Coq's stdlib *)

**Definition** definition not (H:Prop) := H → False.

**Goal** not (1 = 2).

**Outputs:**

1 subgoal

1 <> 2

(Done in class.)
Negation-related notations

- not $P$ is the same as $\sim P$, typeset as $\neg P$
- not $(x = y)$ is the same as $x \not\sim y$, typeset as $x \neq y$

Can we rewrite not with an inductive proposition?
Equivalence

$P \iff Q$
Logical equivalence

**Definition** iff $A \land B : \text{Prop} = (A \rightarrow B) \land (B \rightarrow A)$.

(* Notation $\leftrightarrow$ *)

**Goal** $(1 \leftrightarrow 1)$.

Tactics **rewrite**, **reflexivity**, and **symmetry** all handle equivalence as well.

Can we rewrite **iff** with an inductive proposition?
Equivalence exercise

Theorem mult_0 :
forall n m, n * m = 0 ↔ n = 0 ∨ m = 0.

Theorem or_assoc :
forall P Q R : Prop, P ∨ (Q ∨ R) ↔ (P ∨ Q) ∨ R.

Theorem mult_0_3 :
forall n m p, n * m * p = 0 ↔ n = 0 ∨ m = 0 ∨ p = 0.
Existential quantification

\[ \exists x. P \]
Existential quantification

\[
\text{Inductive } \text{ex} \ (A : \text{Type}) \ (P : A \to \text{Prop}) : \text{Prop} := \\
\mid \text{ex_intro} : \forall (x : A) \ (\_ : P\ x), \text{ex} \ P.
\]

Notation:

\text{exists } x:A, \ P \ x

- To conclude a goal \text{exists } x:A, \ P \ x we can use tactics \text{exist} x. which yields \ P \ x. Alternatively, we can use \text{apply ex_intro}.
- To use a hypothesis of type \text{H:exists } x:A, \ P \ x, you can use \text{destruct} \ H \ as \ (x,H), or \text{inversion} \ H
Equality

\[ X = Y \]
Equality

Even equality is defined as an inductive proposition

Inductive eq (A : Type): A → A → Prop :=
| eq_refl :
  forall x:A,
  eq x x.

Hide notations to see eq in action.
Programming with propositions

List membership

```ocaml
Fixpoint In {A : Type} (x : A) (l : list A) : Prop :=
  match l with
  | [] => False
  | x' :: l' => x' = x \ /
               In x l'
  end.
```
Example

Goal In 4 [1; 2; 3; 4; 5].
Example 3 stars

Takes as arguments two properties of numbers, \( P_{odd} \) and \( P_{even} \), and it should return a property \( P \) such that \( P \ n \) is equivalent to \( P_{odd} \ n \) when \( n \) is odd and equivalent to \( P_{even} \ n \) otherwise.

Definition combine_odd_even \((P_{odd} \ P_{even} : \text{n}at \rightarrow \text{Prop})\) : \text{n}at \rightarrow \text{Prop}

Theorem combine_odd_even_intro :
forall \((P_{odd} \ P_{even} : \text{n}at \rightarrow \text{Prop}) \ (n : \text{n}at),

(oddb \ n = \text{true} \rightarrow P_{odd} \ n) \rightarrow

(oddb \ n = \text{false} \rightarrow P_{even} \ n) \rightarrow

\text{combine_odd_even} \ P_{odd} \ P_{even} \ n.

Theorem combine_odd_even_elim_odd :
forall \((P_{odd} \ P_{even} : \text{n}at \rightarrow \text{Prop}) \ (n : \text{n}at),

\text{combine_odd_even} \ P_{odd} \ P_{even} \ n \rightarrow

oddb \ n = \text{true} \rightarrow

P_{odd} \ n.
Theorem combine_odd_even_elim_even :
  \forall (Podd Peven : \text{nat} \rightarrow \text{Prop}) (n : \text{nat}),
  \text{combine_odd_even} Podd Peven n \rightarrow 
  \text{oedb} n = \text{false} \rightarrow 
  \text{Peven} n.