Logical Foundations of Computer Science

Lecture 8: Logical connectives in Coq

Tiago Cogumbreiro
Today we will learn...

- more logic connectives
- constructive logic (and its relation to classical logic)
- building propositions with functions
- building propositions with inductive definitions
Logic connectives
Truth

T

T
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.
Truth example

**Goal** True.

*(Done in class.)*
Equivalence

\[ P \iff Q \]
Logical equivalence

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

(* Notation $\leftrightarrow$ *)
Split equivalence in goal

Goal ($1 = 1 \iff \text{True}$).

Theorem `mult_0`:

\[
\forall n m, n \times m = 0 \iff n = 0 \lor m = 0.
\]

Admitted.

- When induction is required, prove each side by induction independently. Split, and prove each side in its own theorem by induction.
Apply equivalence to assumption

Goal

\[
\text{forall } x \ y \ z, \\
x \ast (y \ast z) = 0 \Rightarrow \\
x \ast y = 0 \lor z = 0.
\]

Proof.

Admitted.
Interpret equivalence as equality

The Setoid library lets you treat an equivalence as an equals:
Tactics rewrite, reflexivity, and symmetry all handle equivalence as well.

```

Goal
  forall x y z, x * (y * z) = 0 <-> x = 0 / (y = 0 / z = 0).
Proof.
Admitted.
```
Existential quantification

$\exists x. P$
Existential quantification

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 \( P \ x \).
- To use a hypothesis of type \( H : \text{exists } x : A, \ P \ x \), you can use \( \text{destruct } H \text{ as } (x,H) \)
Use exist for existential in goal

To conclude a goal \( \exists x : A, \, P \, x \) we can use tactics \( \text{exist} \, x \). which yields \( P \, x \).

Goal
\[
\forall y, \exists x, \text{Nat.beq} \, x \, y = \text{true}.
\]

Goal
\[
\exists x \, y, \quad 3 + x = y.
\]

- Give the value that satisfies the equality.
- You can play around with exists to figure out what makes sense.
Destruct existential in assumption

Goal

\[
\text{forall } n, \\
(\exists m, n = 4 + m) \implies \\
(\exists o, n = 2 + o).
\]
Constructive logic is not classical logic
Constructive logic is not classical logic

- Coq implements a constructive logic
- Every proof consists of evidence that is constructed
- You cannot assume the law of the excluded middle (proofs that appear out of thin air)
- Truth tables may fail you!
  Especially if there are negations involved.

The following are unprovable in constructive logic (and therefore in Coq):

\begin{align*}
\text{Goal } & \forall (P:\text{Prop}), P \lor \sim P. \\
\text{Goal } & \forall P Q, ((P \to Q) \to P) \to P. \\
\text{Goal } & \forall (P Q:\text{Prop}), \sim (\sim P \lor \sim Q) \to P \lor Q.
\end{align*}
Building propositions with functions
Fixpoint replicate (P:Prop) (n:nat) :=
  match n with
  | 0 => True
  | S m => P \ replicate P m
end.

Print replicate (1 = 0) 3.

Goal forall P,
Replicate P 0 <-> True.

Goal forall P n,
P <-> Replicate (S n).
List membership example

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

- Computation cannot match on propositions
- Computations destruct types, not propositions
Building propositions with data structures (inductively)
Enumerated propositions
Recall enumerated types?

You can think of true as an enumerated type.

```plaintext
Inductive True : Prop :=
  I : True.
```
Many equivalent proofs

\texttt{Inductive Foo : Prop :=
| A : Foo
| B : Foo.}
Many equivalent proofs

Inductive Foo : Prop :=
  | A : Foo
  | B : Foo.

Yet, same as having one

Goal
  Foo ↔ True.

- We can prove Foo with A or with B, we still just have Foo
- What happens when we do a case analysis on Foo? Show when A holds, then show when B holds.
Falsehood

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

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

This explains why case analysis proves the following goal:

```coq
Goal
  False →
  1 = 0.
```
Composite inductive propositions
Disjunction

Inductive or (A B : Prop) : Prop :=
| or_introl :
  A ->
  or A B
| or_intror :
  B ->
  or A B
Conjunction

Inductive and (P Q : Prop) : Prop :=
| conj :
  P →
  Q →
  and P Q.
Adding parameters to predicates

$$\text{Inductive\ Bar} : \text{nat} \rightarrow \text{Prop} :=$$
$$\mid C : \text{Bar 1}$$
$$\mid D : \text{Bar 2}.$$
Adding parameters to predicates

\[
\text{Inductive } \text{Bar} : \text{nat} \to \text{Prop} := \\
| \text{C} : \text{Bar} \ 1 \\
| \text{D} : \forall n, \\
\quad \text{Bar} \ (\text{S} \ n).
\]

\text{Goal} \ \forall n, \\
\text{Bar} \ n \ \Rightarrow \\
\quad n \ \not\equiv \ 0.
Alternative definition of Bar

**Definition** $\text{Bar2 } n : \text{Prop} := n <> 0.$
Existential

\[
\text{Inductive sig} \ (A : \text{Type}) \ (P : A \rightarrow \text{Prop}) : \text{Type} := \\
| \text{exist} : \forall x : A, \\
\ P x \rightarrow \\
\ sig A P. \\
\]
Recursive inductive propositions
Defining In inductively

\textbf{Inductive} \texttt{In} \{A:Type\} : A \to \texttt{list A} \to \texttt{Prop} :=
Defining \( \text{In} \) inductively

\[
\text{Inductive } \text{In} \{A: \text{Type}\} : A \to \text{list } A \to \text{Prop} := \\
| \text{in_eq:} \\
\quad \text{forall } x \ l, \\
\quad \text{In } x \ (x::l) \\
| \text{in_cons:} \\
\quad \text{forall } x \ y \ l, \\
\quad \text{In } x \ l \to \\
\quad \text{In } x \ (y::l).
\]
Fixed parameters in inductive propositions

Inductive In \{A:\text{Type}\} (x: A) : list A \to Prop :=
| in_eq:
  \forall x \, l,
  \text{In} x (x::l)
| in_cons:
  \forall x \, y \, l,
  \text{In} x l \to
  \text{In} x (y::l).
Defining even numbers

```
Inductive Even : nat → Prop :=
| even_0 : Even 0
| even_s_s : forall n, Even n → Even (S (S n)).

Goal forall n, Even n → exists m, n = 2 * m.
```