CS720

Logical Foundations of Computer Science

Lecture 7: Logical connectives in Coq

Tiago Cogumbreiro

Poly.v

Due Tuesday, September 25, 11:59 EST

Tactics.v

Due Thursday, September 27, 11:59 EST

Logic.v

Due Thursday, October 4, 11:59 EST

What have we learned so far

- Comparing if two expressions are equal syntactically: `e1 = e2`
- Implication `P -> Q`
- Universal quantifier `forall x, P`

> Is this all we can do?

What have we learned so far

- Comparing if two expressions are equal syntactically: `e1 = e2`
- Implication `P → Q`
- Universal quantifier `forall x, P`

> Is this all we can do? No.

We encoded predicates *computationally*:

- In `Basics.v` we defined `beq_nat: nat → nat → bool` to compare if two naturals are equal.
- In `Basics.v` we defined `evenb: nat → bool` to check if a natural number is even

> Computational predicates are limited in what they can describe (eg, functions in Coq have to be total), and are not very easy to reason about (ie, they are meant to compute/execute, not build logic statements).

Today we will...

- Logical connectives in Coq

$$P \wedge Q \qquad P \vee Q$$

Why are we learning this?

- The building blocks of any interesting property

Typing equality

> What is the type of an equality?

```
Check beq_nat 2 2 = true.

Check forall (n m : nat), n + m = m + n.
```

Typing equality

> What is the type of an equality?

```
Check beq_nat 2 2 = true.

Check forall (n m : nat), n + m = m + n.
```

Both of these expressions have type `Prop`, for proposition.

Are all propositions provable?

Are all propositions provable?

No. How do you prove this proposition:

```
Check 0 = 1. (* Prints: 0 = 1: Prop *)

Goal 0 = 1.
```

> Notice how we cannot conclude (write a proof for) a statement that does not hold.
> In Coq, we must show evidence of what holds. *(This is known as a constructive logic.)*

Propositions are still expressions (1/3)

What is the type of `ex0`:

```
Definition ex0 := beq_nat 2 2.
```

Propositions are still expressions (1/3)

What is the type of `ex0`:

```
Definition ex0 := beq_nat 2 2.
```

What is the type of `ex1`? How can we use `ex1`?

```
Definition ex1 (n:nat) := beq_nat 2 n = true.
```

For which `n` is `ex1 n` provable?

Propositions are still expressions (1/3)

What is the type of `ex0`:

```
Definition ex0 := beq_nat 2 2.
```

What is the type of `ex1`? How can we use `ex1`?

```
Definition ex1 (n:nat) := beq_nat 2 n = true.
```

For which `n` is `ex1 n` provable?

```
Lemma easy:
  forall n, n = 2 -> ex1 n.
Proof.
```

(Done in class.)

Propositions are still expressions (2/3)

> What is the difference between `ex1` and `ex2`?

```
Definition ex1 (n:nat) := beq_nat 2 n = true.

Theorem ex2: forall (n:nat), beq_nat 2 n = true.
```

Propositions are still expressions (2/3)

> What is the difference between `ex1` and `ex2`?

```
Definition ex1 (n:nat) := beq_nat 2 n = true.

Theorem ex2: forall (n:nat), beq_nat 2 n = true.
```

`ex1` defines a position (`Prop`), `ex2` is a theorem definition and is expecting a proof.

> What is the relation between `ex3` and `ex1`, `ex2`?

```
Definition ex3 (n:nat) : beq_nat 2 n = true.
```

Propositions are still expressions (2/3)

> What is the difference between `ex1` and `ex2`?

```
Definition ex1 (n:nat) := beq_nat 2 n = true.

Theorem ex2: forall (n:nat), beq_nat 2 n = true.
```

`ex1` defines a position (`Prop`), `ex2` is a theorem definition and is expecting a proof.

> What is the relation between `ex3` and `ex1`, `ex2`?

```
Definition ex3 (n:nat) : beq_nat 2 n = true.
```

- Recall that `Theorem` and `Definition` are synonyms!
- Thus, `ex2` and `ex3` are the same

Propositions are still expressions (3/3)

> What is the difference between `ex3` and `ex4`?

```
Definition ex1 (n:nat) := beq_nat 2 n = true.
Definition ex3 (n:nat) : beq_nat 2 n = true.

Theorem ex4: forall (n:nat), ex1 n.
```

Propositions are still expressions (3/3)

What is the difference between `ex3` and `ex4`?

```
Definition ex1 (n:nat) := beq_nat 2 n = true.
Definition ex3 (n:nat) : beq_nat 2 n = true.

Theorem ex4: forall (n:nat), ex1 n.
```

`ex3` and `ex4` are the same.

Are any of `ex2`, `ex3`, and `ex4` provable?

Propositions are still expressions (3/3)

> What is the difference between `ex3` and `ex4`?

```
Definition ex1 (n:nat) := beq_nat 2 n = true.
Definition ex3 (n:nat) : beq_nat 2 n = true.

Theorem ex4: forall (n:nat), ex1 n.
```

> `ex3` and `ex4` are the same.

> Are any of `ex2`, `ex3`, and `ex4` provable?

No, because not all numbers are equal to 2.

Summary: Propositions are still expressions

- `forall` versus arguments of a definition
- Definitions for propositions are just abbreviations for our own understanding

For instance, define `GreaterThan` in terms of `leb` so that it is easier to read:

```
Definition GreaterThan x y := leb x y = false.
(* which is the same as *)
Definition GreaterThan := forall x y, leb x y = false.
```

Coq from the ground up

Inductive propositions

We have seen how to define types inductively; propositions can also be defined inductively.

- instead of `Type` we use `Prop`
- the parameters are not just values, but propositions
- the idea is to build your logical argument as *structured data*

We will now encode various logical connectives using inductive definitions.

Conjunction

$$P \land Q$$

What is $P \wedge Q$?

1. What is the type of P?

What is $P \wedge Q$?

1. What is the type of P? `Prop`
2. What is the type of Q?

What is $P \wedge Q$?

1. What is the type of P? `Prop`
2. What is the type of Q? `Prop`
3. What is the type of $\wedge$?

What is $P \wedge Q$?

1. What is the type of P? `Prop`
2. What is the type of Q? `Prop`
3. What is the type of $\wedge$? `Prop → Prop → Prop`

What is $P \wedge Q$?

Let `and` represent $\wedge$:

```
and: Prop → Prop → Prop
```

Recall how we defined a pair:

```
Inductive pair (X:Type) (Y:Type) : Type := ...
```

How would we define `and`?

Conjunction

```
Inductive and (P Q : Prop) : Prop :=
| conj : P → Q → and P Q.
```

- **`apply conj` to solve a goal, `inversion` in a hypothesis**
- The `/\` operator represents a logical conjunction (usually typeset with $\wedge$)
- The `split` tactics is used to prove a goal of type `?X /\ ?Y`, where `?X` and `?Y` are propositions

> Notice that `P /\ Q` is a type (a proposition) and that `conj` is the only constructor of that type.

Conjunction example

```
Example and_example : 3 + 4 = 7 /\ 2 * 2 = 4.
Proof.
  apply conj.
```

(Done in class.)

Conjunction example 1

More generally, we can show that if we have propositions A and B, we can conclude that we have $A \wedge B$.

```
Goal forall A B : Prop, A → B → A /\ B.
```

Conjunction in the hypothesis

```
Example and_in_conj :
  forall x y,
  3 + x = y /\ 2 * 2 = x ->
  x = 4 /\ y = 7.
Proof.
  intros x y Hconj.
  destruct Hconj as [Hleft Hright].
```

(Done in class.)

Conjunction example 2

```
Lemma correct_2 : forall A B : Prop, A /\ B → A.
Proof.
```

```
Lemma correct_3 : forall A B : Prop, A /\ B → B.
Proof.
```

(Done in class.)

Disjunction

$$P \vee Q$$

What is $P \lor Q$?

1. What is the type of P?

What is $P \lor Q$?

1. What is the type of P? `Prop`
2. What is the type of Q?

What is $P \vee Q$?

1. What is the type of P? `Prop`
2. What is the type of Q? `Prop`
3. What is the type of $\vee$?

What is $P \vee Q$?

1. What is the type of P? `Prop`
2. What is the type of Q? `Prop`
3. What is the type of $\vee$? `Prop → Prop → Prop`

> How can we define an disjunction using an inductive proposition?

Disjunction

```
Inductive or (A B : Prop) : Prop :=
  | or_introl : A → or A B
  | or_intror : B → or A B
```

- **`apply or_introl` or `apply or_intror` to goal; `inversion` to hypothesis**
- The `\/` operator represents a logical disjunction (usually typeset with $\vee$)
- The `left` (`right`) tactics are used to prove a goal of type `?X \/ ?Y`, replacing it with a new goal `?X` (`?Y` respectively)

Disjunction example

```
Theorem or_1: forall A B : Prop,
  A -> A \/ B.

Theorem or_2: forall A B : Prop,
  B -> A \/ B.
```

(Done in class.)

Disjunction in the hypothesis

> Tactics `destruct` can break a disjunction into its two cases.
> Tactics `inversion` also breaks a disjunction, but leaves the original hypothesis in place.

```
Lemma or_example :
  forall n m : nat, n = 0 \/ m = 0 -> n * m = 0.
Proof.
  intros n m Hor.
  destruct Hor as [Heq | Heq].
```

Example

```
Theorem odd_or_even:
  forall n,
  evenb n = true \/ oddb n = true.
```

Example

```
Theorem odd_or_even:
  forall n,
  evenb n = true \/ oddb n = true.
```

Hint, prove this first:

```
Theorem evenb_flip:
  forall n,
  evenb n = negb (evenb (S n)).
```

Summary

- Propositions as expressions
- Inductive propositions
- $P \wedge Q$
- $P \vee Q$