CS420

Logical Foundations of Computer Science

Lecture 6: Logical connectives

Tiago Cogumbreiro

Today we will learn...

- What are proofs?
- Logical connectives
- Inductive propositions

What are proofs?

What is a type? What is a value?

- `nat` is a type

What is a type? What is a value?

- `nat` is a type
- `5` is a value of type `nat`

What is a type? What is a value?

- `nat` is a type
- `5` is a value of type `nat`
- Notations `5 : nat` means `5` has type `nat`

What is a type? What is a value?

- `nat` is a type
- `5` is a value of type `nat`
- Notations `5 : nat` means `5` has type `nat`
- ***Types can be thought of as sets***
  `5 : nat` a programming notation $5 \in \mathcal{N}$

Exercise

Consider the following Coq excerpt:

```
Definition x := 5.
```

- What is `x`?

Exercise

Consider the following Coq excerpt:

```
Definition x := 5.
```

- What is `x`? A variable.
- What is the value of `x`?

Exercise

Consider the following Coq excerpt:

```
Definition x := 5.
```

- What is `x`? A variable.
- What is the value of `x`? `5`
- What is the type of `x`?

Exercise

Consider the following Coq excerpt:

```
Definition x := 5.
```

- What is `x`? A variable.
- What is the value of `x`? `5`
- What is the type of `x`? `nat`
- How do I query the type of `x` in Coq?

Exercise

Consider the following Coq excerpt:

```
Definition x := 5.
```

- What is `x`? A variable.
- What is the value of `x`? `5`
- What is the type of `x`? `nat`
- How do I query the type of `x` in Coq? Using `Check`.
- How do I query the value of `x` in Coq?

Exercise

Consider the following Coq excerpt:

```
Definition x := 5.
```

- What is `x`? A variable.
- What is the value of `x`? `5`
- What is the type of `x`? `nat`
- How do I query the type of `x` in Coq? Using `Check`.
- How do I query the value of `x` in Coq? Using `Print`.

What is a proof? What is a proposition?

- **A proof** (or a proof object): a ***completed*** proof of some goal
  - usually written using tactics
  - a proof object is a ***value*** of a proposition

What is a proof? What is a proposition?

- **A proof** (or a proof object): a ***completed*** proof of some goal
  - usually written using tactics
  - a proof object is a ***value*** of a proposition
- **A proposition**: is a formula written in some logic
  - Propositions are of type `Prop`
  - You can confirm that something is a proposition using `Check`

What is a proof? What is a proposition?

- **A proof** (or a proof object): a ***completed*** proof of some goal
  - usually written using tactics
  - a proof object is a ***value*** of a proposition
- **A proposition**: is a formula written in some logic
  - Propositions are of type `Prop`
  - You can confirm that something is a proposition using `Check`
- **A truthful proposition**: a proposition that contains a proof
  - `Proof : Proposition`
  - We also say that the proposition **holds** (if there is some proof of it)

What is a proof? What is a proposition?

- **A proof** (or a proof object): a ***completed*** proof of some goal
  - usually written using tactics
  - a proof object is a ***value*** of a proposition
- **A proposition**: is a formula written in some logic
  - Propositions are of type `Prop`
  - You can confirm that something is a proposition using `Check`
- **A truthful proposition**: a proposition that contains a proof
  - `Proof : Proposition`
  - We also say that the proposition **holds** (if there is some proof of it)
- **Assumption:** a synonym of a proof

What is a proof? What is a proposition?

- **A proof** (or a proof object): a ***completed*** proof of some goal
  - usually written using tactics
  - a proof object is a ***value*** of a proposition
- **A proposition**: is a formula written in some logic
  - Propositions are of type `Prop`
  - You can confirm that something is a proposition using `Check`
- **A truthful proposition**: a proposition that contains a proof
  - `Proof : Proposition`
  - We also say that the proposition **holds** (if there is some proof of it)
- **Assumption:** a synonym of a proof
- **Proof state:** zero or more assumptions and 1 or more goals we need to prove
  - Each assumption is an implication to the current goal
  - Each sub-goal is a conjunctions

Exercise

- Is `10` a proposition?

Exercise

- Is `10` a proposition? No. `10` is a natural number.
- Is `2 = 2` a proposition?

Exercise

- Is `10` a proposition? No. `10` is a natural number.
- Is `2 = 2` a proposition? Yes.
- Is `Nat.eqb 2 2` a proposition?

Exercise

- Is `10` a proposition? No. `10` is a natural number.
- Is `2 = 2` a proposition? Yes.
- Is `Nat.eqb 2 2` a proposition? No, `Nat.eqb 2 2` is an expression of type `bool`.
- Is the code below a proposition?

```
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a **proof** of formula `2 = 2`.

- What is `example`?

Exercise

- Is `10` a proposition? No. `10` is a natural number.
- Is `2 = 2` a proposition? Yes.
- Is `Nat.eqb 2 2` a proposition? No, `Nat.eqb 2 2` is an expression of type `bool`.
- Is the code below a proposition?

```
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a **proof** of formula `2 = 2`.

- What is `example`? A proof of `2 = 2`.
- What is the value of `example`?

Exercise

- Is `10` a proposition? No. `10` is a natural number.
- Is `2 = 2` a proposition? Yes.
- Is `Nat.eqb 2 2` a proposition? No, `Nat.eqb 2 2` is an expression of type `bool`.
- Is the code below a proposition?

```
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a **proof** of formula `2 = 2`.

- What is `example`? A proof of `2 = 2`.
- What is the value of `example`? `reflexivity.` (actually `eq_refl`)
- What is the type of `example`?

Exercise

- Is `10` a proposition? No. `10` is a natural number.
- Is `2 = 2` a proposition? Yes.
- Is `Nat.eqb 2 2` a proposition? No, `Nat.eqb 2 2` is an expression of type `bool`.
- Is the code below a proposition?

```
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a **proof** of formula `2 = 2`.

- What is `example`? A proof of `2 = 2`.
- What is the value of `example`? `reflexivity.` (actually `eq_refl`)
- What is the type of `example`? `2 = 2`.
- What is the type of `2 = 2`?

Exercise

- Is `10` a proposition? No. `10` is a natural number.
- Is `2 = 2` a proposition? Yes.
- Is `Nat.eqb 2 2` a proposition? No, `Nat.eqb 2 2` is an expression of type `bool`.
- Is the code below a proposition?

```
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a **proof** of formula `2 = 2`.

- What is `example`? A proof of `2 = 2`.
- What is the value of `example`? `reflexivity.` (actually `eq_refl`)
- What is the type of `example`? `2 = 2`.
- What is the type of `2 = 2`? `Prop`.

Exercise

- Is `10` a proposition? No. `10` is a natural number.
- Is `2 = 2` a proposition? Yes.
- Is `Nat.eqb 2 2` a proposition? No, `Nat.eqb 2 2` is an expression of type `bool`.
- Is the code below a proposition?

```
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a **proof** of formula `2 = 2`.

- What is `example`? A proof of `2 = 2`.
- What is the value of `example`? `reflexivity.` (actually `eq_refl`)
- What is the type of `example`? `2 = 2`.
- What is the type of `2 = 2`? `Prop`.

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

$$P \wedge Q$$

What is $P \wedge Q$?

1. What is the type of P?

What is $P \land Q$?

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].
```

Recall a proof state

```
1 subgoal
T : Type
x : T
P : Prop
H1 : 1 = x
H2 : P
______________________________________(1/1)
1 = 2 /\ 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
______________________________________(1/1)
1 = 2 /\ 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`? Yes and no, respectively.

Where do constructors of propositions appear?

```
Theorem and_conj: forall P Q:Prop,
  P -> Q -> P /\ Q.
Proof.
  intros P Q H1 H2.
   apply conj.
  - apply H1.
  - apply H2.
Qed.
```

Theorems are expressions too

```
Theorem and_conj: forall P Q:Prop,
  P -> Q -> P /\ Q.
Proof.
  intros P Q H1 H2.
  apply (conj H1 H2).
Qed.
```

> Proposition-constructors and theorems are **functions** whose parameters are **evidences**.

Truth

$$\top$$

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.

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

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

Truth example

```
Goal True.
```

(Done in class.)

Falsehood

$$\perp$$

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 `False`.

Example:

```
Goal 1 = 2 -> False.

Goal False -> 1 = 2.

Goal False.
```

(Done in class.)

Negation

$$\neg P$$

Negation

> The negation of a proposition $\neg P$ is defined as

```
(* As defined in Coq's stdlib *)
Definition not (H:Prop) := H → False.

Goal not (1 = 2).
```

Outputs:

```
1 subgoal
______________________________________(1/1)
1 <> 2
```

(Done in class.)

Negation-related notations

- `not P` is the same as `~ P`, typeset as $\neg P$
- `not (x = y)` is the same as `x <> y`, typeset as $x \neq y$

> Can we rewrite `not` with an inductive proposition?