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 ∈ \( \mathbb{N} \)
Exercise

Consider the following Coq excerpt:

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

- What is x?
Exercise

Consider the following Coq excerpt:

\[
\text{Definition } x := 5. \\
\]

- 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:

```coq
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:

```coq
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.
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?**

```coffeescript
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 $\text{Nat.eqb } 2 \ 2$ a proposition? No, $\text{Nat.eqb } 2 \ 2$ is an expression of type $\text{bool}$.
- 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?

```coq
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 $\text{Nat.eqb 2 2}$ a proposition? No, $\text{Nat.eqb 2 2}$ is an expression of type $\text{bool}$.
- Is the code below a proposition?

```latex
Lemma example: 2 = 2.
Proof.
\hspace{1em} \text{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 $2 = 2$? $\text{Prop}$.
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

$P \land Q$
What is $P \land Q$?

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 $P \land Q$?

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

1. What is the type of $P$? Prop
2. What is the type of $Q$? Prop
3. What is the type of $\land$? Prop $\rightarrow$ Prop $\rightarrow$ Prop
What is $P \land Q$?

Let and represent $\land$:

\[
\text{and}: \text{Prop} \to \text{Prop} \to \text{Prop}
\]

Recall how we defined a pair:

\[
\text{Inductive pair } (X:\text{Type}) (Y:\text{Type}) : \text{Type} := \ldots
\]

How would we define and?
Conjunction

**Inductive** \( \text{and} (P \ Q : \text{Prop}) : \text{Prop} := \)
\[
| \text{conj} : P \rightarrow Q \rightarrow \text{and} P \ Q. \\
\]

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

Notice that \( P \ \land \ Q \) 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 $A \land B$.

Goal forall $A$ $B$ : Prop, $A \rightarrow B \rightarrow A \land B$. 
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.)
Lemma correct_2 : \(\forall A B : \text{Prop}, A \land B \rightarrow A.\)
Proof.

Lemma correct_3 : \(\forall A B : \text{Prop}, A \land B \rightarrow B.\)
Proof.

(Done in class.)
Disjunction

\[ P \lor Q \]
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 $P \lor Q$?

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

1. What is the type of $P$? Prop
2. What is the type of $Q$? Prop
3. What is the type of $\lor$? Prop $\rightarrow$ Prop $\rightarrow$ Prop

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

\[ \text{Inductive } or \ (A \ B : \text{Prop}) : \text{Prop} := \]
\[ | \text{or_introl} : A \rightarrow or \ A \ B \]
\[ | \text{or_intror} : B \rightarrow or \ A \ B \]

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

Theorem or_1: \( \forall A, B : \text{Prop}, \ A \to A \lor B. \)

Theorem or_2: \( \forall A, B : \text{Prop}, \ B \to A \lor B. \)

*(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 :
\[ \forall n \ m : \text{nat}, \ n = 0 \lor m = 0 \rightarrow n \times m = 0. \]

Proof.
\begin{Verbatim}
intros n m Hor.
destruct Hor as [Heq | Heq].
\end{Verbatim}
Recall a proof state

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

All hypotheses 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: \( \forall P \ Q : \text{Prop}, \ P \rightarrow Q \rightarrow P \land Q. \)

Proof.
\begin{align*}
&\text{intros } P \ Q \ H1 \ H2. \\
&\quad \text{apply conj.} \\
&\quad - \text{apply } H1. \\
&\quad - \text{apply } H2. \\
&\text{Qed.}
\end{align*}
Theorems are expressions too

Theorem \textit{and\textunderscore conj}: \[\forall P \ Q : \text{Prop},\] \[P \rightarrow Q \rightarrow P \land Q.\]

Proof.
\begin{itemize}
  \item \texttt{intros} P Q H1 H2.
  \item \texttt{apply} (conj H1 H2).
\end{itemize}
Qed.

Proposition-constructors and theorems are \textit{functions} whose parameters are \textit{evidences}.
Truth

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.

\[
\text{Inductive } \text{True} : \text{Prop} \equiv \text{I} : \text{True}.
\]

You will note that proposition \text{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 in Coq is represented by an **empty** type.

```coq
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

\[
(* \text{ As defined in Coq's stdlib } *) \\
\text{Definition } \text{not } (\text{H:Prop}) := \text{H} \rightarrow \text{False}.
\]

\text{Goal } \text{not } (1 = 2).

Outputs:
1 subgoal

\[1 \leftrightarrow 2\]  
\text{(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 $x \not= y$, typeset as $x \neq y$

Can we rewrite not with an inductive proposition?