CS 3110

Logic in Coq

Prof. Clarkson
Fall 2017

Today's music: *Autologic* by Rage Against The Machine

Review

Previously in 3110:

- Functional programming in Coq
- Proofs about simple programs

Today:

- Logic in Coq, at the CS 2800 level

TYPES

Propositions

```
Check list.
list : Type -> Type

Check 42 = 42.
42 = 42 : Prop

Check 42 = 3110.
42 = 3110 : Prop
```

Propositions

```
Theorem one_plus_one_is_two : 1 + 1 = 2.
Proof. trivial. Qed.

Check one_plus_one_is_two.
one_plus_one_is_two : 1 + 1 = 2

Print one_plus_one_is_two.
one_plus_one_is_two = eq_refl : 1 + 1 = 2

Check 1 + 1 = 2.
1 + 1 = 2 : Prop
```

Propositions

```
Check Prop.
```

Prop : Type

```
Check Type.
```

Type : Type

Logical formulas.
Things that can be proved.

Sets

```
Let x := 42.
Check x.
x : nat

Check nat.
nat : Set

Check bool.
bool: Set
```

Sets

```
Check Set.
```

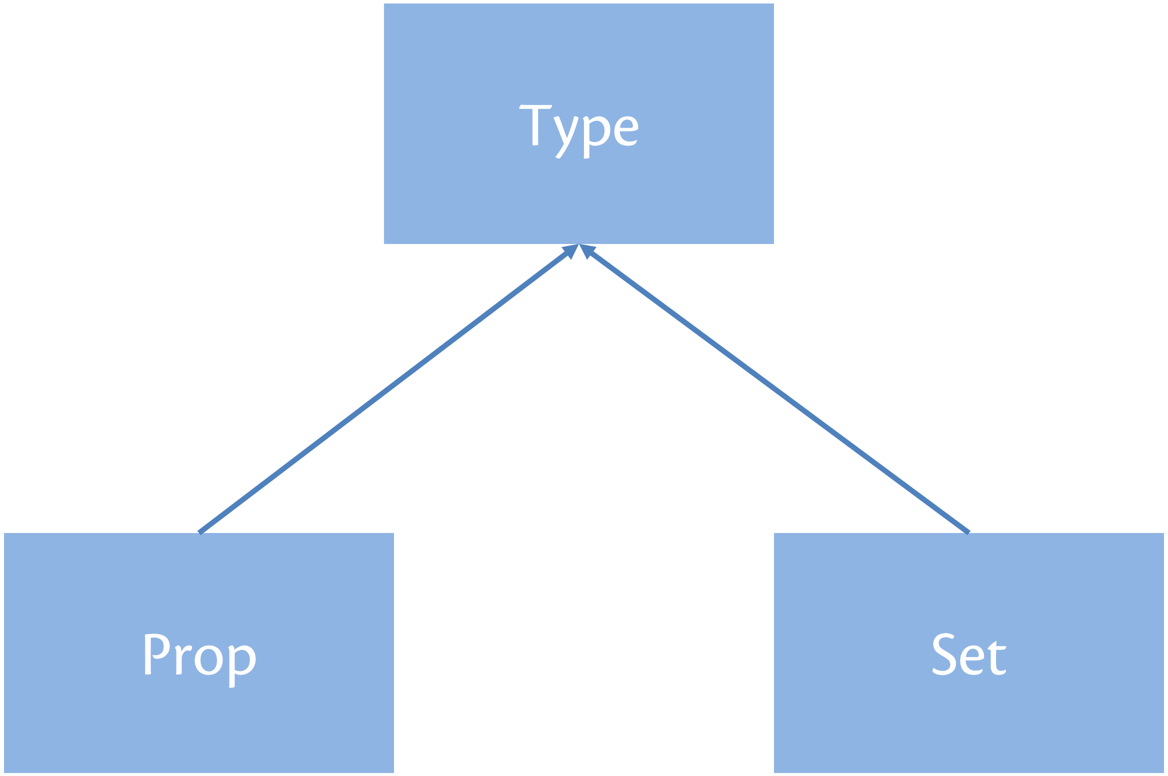
`Set : Type`

Program data types. Things that can be computed.

```
Check Type.
```

`Type : Type`

Type hierarchy



LOGIC

Logical connectives

- Implication: `p -> p`
- Conjunction: `p /\ p`
- Disjunction: `p \/ p`
- Negation: `~p`

Implication

```
Theorem p_implies_p : forall P:Prop, P -> P.
Proof.
  intros P. intros P_assumed. assumption.
Qed.

Check p_implies_p.
p_implies_p : forall P:Prop, P -> P
```

Implication

output is that proof

```
Print p_implies_p.
p_implies_p =
fun (P : Prop) (P_assumed : P) => P_assumed
     : forall P : Prop, P -> P
```

`p_implies_p` is a function

first input is a proposition

second input is proof of first input

Coq proofs

are

functional programs

Conjunction

```
Theorem and_fst : forall P Q, P /\ Q -> P.
Proof.
  intros P Q PandQ.
  destruct PandQ as [P_holds Q_holds].
  assumption.
Qed.
```

case analysis on why hypothesis holds

names for pieces of evidence for hypothesis

Conjunction

```
Check and_fst.
and_fst : forall P Q : Prop, P /\ Q -> P

Print and_fst.
and_fst =
fun (P Q : Prop) (PandQ : P /\ Q) =>
match PandQ with
| conj P_holds _  => P_holds
end
     : forall P Q : Prop, P /\ Q -> P
```

Conjunction

```
Ch                .
ar          al                      /
Pri  and_fst.
and_fst =
fun (P Q : Prop) (PandQ : P /\ Q) =>
match PandQ with
| conj P_holds _ => P_holds
end
     : for             op, P /\          -> P
```

and_fst is a function

takes two propositions as input

and proof of their conjunction

pattern matches to extract first proof

returns that proof

Conjunction

```
Theorem and_ex : 42=42 /\ 43=43.
Proof. split. trivial. trivial. Qed.
```

split conjunction
into two subgoals

```
Print and_ex.
and_ex = conj eq_refl eq_refl
     : 42 = 42 /\ 43 = 43
```

`and_ex` applies `conj`
to two uses of
reflexivity of =

Conjunction

```
Theorem and_comm: forall P Q,
  P /\ Q -> Q /\ P.
Proof.
  intros P Q PandQ.
  destruct PandQ as [P_holds Q_holds].
  split.
  all: assumption.
Qed.
```

case analysis on why `P /\ Q` holds

Conjunction

```
Print and_comm.
and_comm =
fun (P Q : Prop) (PandQ : P /\ Q) =>
match PandQ with
| conj P_holds Q_holds =>
  conj Q_holds P_holds
end
     : forall P Q : Prop, P /\ Q -> Q /\ P
```

swap evidence

Disjunction

```
Theorem or_left :
  forall (P Q : Prop), P -> P \/ Q.
Proof.
  intros P Q P_holds. left. assumption.
Qed.

Print or_left.
or_left =
fun (P Q : Prop) (P_holds : P) =>
  or_introl P_holds
     : forall P Q : Prop, P -> P \/ Q
```

choose to prove left side

Disjunction

proves left side

```
Check or_introl.
or_introl
     : forall A B : Prop, A -> A \/ B

Print or_introl.
Inductive or (A B : Prop) : Prop :=
     or_introl : A -> A \/ B
   | or_intror : B -> A \/ B
```

two constructors, one to prove each side

Disjunction

```
Theorem or_comm :
  forall P Q, P \/ Q -> Q \/ P.
Proof.
intros P Q PorQ. destruct PorQ.
- right. assumption.
- left. assumption.
Qed.
```

choose to prove left side

case analysis on why P \/ Q holds

choose to prove right side

Disjunction

```
Print or_comm.
or_comm =
fun (P Q : Prop) (PorQ : P \/ Q) =>
match PorQ with
| or_introl H => or_intror H
| or_intror H => or_introl H
end
     : forall P Q : Prop, P \/ Q -> Q \/ P
```

swap constructors

Negation

```
Theorem explosion : forall P, False -> P.
Proof.
  intros P false_holds.
  contradiction.
Qed.
```

detects contradiction in hypotheses

Negation

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

False has no constructors

```
Print not.
not = fun A : Prop => A -> False
     : Prop -> Prop
```

not is a function:
not P is P -> False

Negation

```
Theorem contra_implies_anything :
  forall P Q, P /\ ~P -> Q.
Proof.
  intros P Q PandnotP.
  destruct PandnotP as [P_holds notP_holds].
  contradiction.
Qed.
```

give names to the two pieces of evidence for why `P /\ ~P` holds

Upcoming events

- N/A

This is logical.

THIS IS 3110