CS 565

Programming Languages (graduate)
Spring 2025

Week 4
Propositions and Inductive Evidence

Propositions

A **proposition** is a factual claim.

Have seen a couple of propositions (in Coq) so far:

- equalities: 0 + n = n
- implications: P -> Q
- universally quantified propositions: forall x, P

A **proof** is some evidence for the truth of a proposition

A **proof system** is a formalization of particular kinds of evidence.

Propositions

- We've already seen a number of propositions in Coq:

```
Theorem ProofExample
  forall n m : nat, n = 0 -> m = 0 -> n + m = 0.
Proof.
  intros n m Hn Hm.
  rewrite Hn. rewrite Hm.
  reflexivity.
Qed.
```

Proposition

Evidence

Propositions


```
Check (2 = 2).                (* : Prop *)
Check (3 = 2).                (* : Prop *)
Check (3 = 2 -> 2 = 3).       (* : Prop *)
Check (forall n: nat, n = 2). (* : Prop *)
```

Propositions

Propositions are first-class entities in Coq. Can name them:

```
Definition plus_claim : Prop := 2 + 2 = 4.
Theorem ProofExample : plus_claim.
Proof.
… (* unfold plus_claim*)
```

We can also write parameterized propositions (**predicates**)

```
Definition is_three (n : nat) : Prop := n = 3.
Theorem ProofExample2 : is_three 3.
Proof.
… (* unfold is_three *)
```

Propositions

Can have polymorphic predicates:

```
Definition injective {A B} (f : A -> B) : Prop :=
  forall x y : A, f x = f y -> x = y.
Theorem plus1_inj : injective (plus 1).
Proof.
… (* unfold injective *)
```

Equality is a polymorphic binary predicate:

```
Check @eq. (* : ∀ A : Type, A → A → Prop *)
```

Concept Check

What is the type of the following expression?

```
pred (S O) = O
```

A. Prop
B. nat→Prop
C. ∀ n:nat, Prop
D. nat→nat
E. Not typeable

Concept Check

What is the type of the following expression?

```
∀ n:nat, pred (S n) = n
```

A. Prop
B. nat→Prop
C. ∀ n:nat, Prop
D. nat→nat
E. Not typeable

Concept Check

What is the type of the following expression?

```
∀ n:nat, S (pred n) = n
```

A. Prop
B. nat→Prop
C. ∀ n:nat, Prop
D. nat→nat
E. Not typeable

Concept Check

What is the type of the following expression?

```
∀ n:nat, S (pred n)
```

A. Prop
B. nat→Prop
C. ∀ n:nat, Prop
D. nat→nat
E. Not typeable

Concept Check

What is the type of the following expression?

```
fun n:nat => S (pred n)
```

A. Prop
B. nat→Prop
C. ∀ n:nat, Prop
D. nat→nat
E. Not typeable

Concept Check

What is the type of the following expression?

```
fun n:nat => S (pred n) = n
```

A. Prop
B. nat→Prop
C. ∀ n:nat, Prop
D. nat→nat
E. Not typeable

Proofs

Haven't we already seen a bunch of proofs too?

```
Theorem ProofExample
  : forall n m : nat, n = 0 -> m = 0 -> n + m = 0.
Proof.
  intros n m Hn Hm.
  rewrite Hn. rewrite Hm.
  reflexivity.
Qed.
```

proof script

formal

What is a proof?

Judgement

A **judgement** is a claim of a proof system

The judgement $\boxed{\Gamma \vdash A}$ is read as:
"assuming the propositions in Γ are true, A is true".

We'll see other judgements over the course of the semester:

Inference Rules

Proof systems construct evidence of judgements via inference rules:

Axioms

$$\frac{}{\Gamma \vdash \top}$$

$$\frac{A \in \Gamma}{\Gamma \vdash A}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \mathbf{I}{\to}$$

$$\frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \mathbf{E}{\to}$$

Inference Rules

Example Proof

Want a proof of:

$\vdash (A \to (B \to C)) \to ((A \to B) \to (A \to C))$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \to (B \to C) \in \Gamma}{\Gamma \vdash A \to (B \to C)} \qquad \dfrac{A \in \Gamma}{\Gamma \vdash A}}{\Gamma \vdash B \to C} \qquad \dfrac{\dfrac{A \to B \in \Gamma}{\Gamma \vdash A \to B} \qquad \dfrac{A \in \Gamma}{\Gamma \vdash A}}{\Gamma \vdash B}}{\Gamma = A \to (B \to C), A \to B, A \vdash C}}{A \to (B \to C), A \to B \vdash A \to C}}{A \to (B \to C) \vdash (A \to B) \to (A \to C)}}{\vdash (A \to (B \to C)) \to ((A \to B) \to (A \to C))}
$$

Symbol Pushing

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \mathbf{I} \wedge$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \mathbf{E_L} \wedge$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \mathbf{E_R} \wedge$$

Inference Rules for $\wedge$

Example

$$\frac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}\ E\vee$$

Introduction Rules for Or?

Inference Rules for $\vee$

Example

$\mathbf{I_L} \vee$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}$$

$\mathbf{E} \vee$

$$\frac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}$$

$\mathbf{I_R} \vee$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

Inference Rules for $\vee$

Example

Can you derive:

$\vdash A \to B \to B \land A$

$\mathbf{I}\land$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B}$$

$\mathbf{I}\to$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

$\mathbf{E}\to$

$$\frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

Proof

Haven't we already seen a number of proofs?

```
Theorem ProofExample
  : forall n m : nat, n = 0 -> m = 0 -> n + m = 0.
Proof.
  intros n m Hn Hm.
  rewrite Hn. rewrite Hm.
  reflexivity.
```

proofscript

What is a formal proof?

A proof tree in the Calculus of co-Inductive Constructions.

Implication

$$\rightarrow$$

Symbol (Math)

->

Syntax (Coq)

I: intro

E: apply*

tactics (Coq)

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}\ \mathbf{I}\!\rightarrow$$

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \mathbf{E}\!\rightarrow$$

Inference Rules for $\rightarrow$

Less Than

$\leq$

Symbol (Math)

$$n \leq m \equiv \exists k.\, n+k = m$$

Definition of $\leq$

$$\frac{}{\Gamma \vdash n \leq n}\ \mathbf{le_n}$$

$$\frac{\Gamma \vdash n \leq m}{\Gamma \vdash n \leq 1 + m}\ \mathbf{le_S}$$

Inference Rules for $\leq$

Even-ness

EvenR

Symbol (Math)

$$\text{EvenR } n \equiv \exists k.\ n = k + k$$

Definition of EvenR

$$\frac{}{\Gamma \vdash \text{EvenR } 0}\ \mathbf{ev_0}$$

$$\frac{\Gamma \vdash \text{EvenR } n}{\Gamma \vdash \text{EvenR } (2+n)}\ \mathbf{ev_2}$$

Inference Rules for EvenR

Less Than (Coq)

$\leq$

- Goal:

  Binary relation on natural numbers

  Form of evidence that two numbers belong to that relation
- Step 0: Name the relation:
- Step 1: Give the relation a signature:
- Step 2: Enumerate evidence:

$$\frac{}{\Gamma \vdash n \leq n}\ \mathbf{le_n}$$

```
      le : nat -> nat -> Prop
le_n : forall n : nat, le n n
```

Less Than (Coq)

$\leq$

- Goal:
  Binary relation on natural numbers
  Form of evidence that two numbers belong to that relation
- Step 0: Name the relation:
- Step 1: Give the relation a signature:
- Step 2: Enumerate evidence:

$$\frac{\Gamma \vdash n \leq m}{\Gamma \vdash n \leq 1 + m}\ \textbf{leS}$$

```
le : nat -> nat -> Prop
le_n : forall n : nat, le n n
le_S : forall n m : nat, le n m -> le n (S m)
```

Inductively Defined Propositions

- Goal:
  N-ary relation on natural numbers
  Form of evidence of membership in that relation
- Step 0: Name the ~~relation~~ type:
- Step 1: Give the ~~relation~~ type a ~~signature~~ type:
- Step 2: Enumerate ~~evidence~~ constructors:

```
Inductive even : nat -> Prop :=
  | ev_O : even O
  | even_2 : forall n : nat, even n ->
                              even (S (S n)).
```

EvenL (Both)

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash \text{EvenL}\ [\,]}\ \mathbf{I_{nil}}\ \text{EL}$$

$$\frac{\Gamma \vdash \text{Even n} \quad \Gamma \vdash \text{EvenL l}}{\Gamma \vdash \text{EvenL (n :: l)}}\ \mathbf{I_{cons}}\ \text{EL}$$

Inference Rules for a "list only has even numbers" (EvenL)

```
Inductive EvenL : list nat -> Prop :=
   EvenL_nil : Forall P [ ]
  | EvenL_cons : forall (n : nat) (l : list nat),
              Even n -> EvenL l -> EvenL (x :: l).
```

Sorted Lists (Coq)

$$\frac{}{\vdash \text{sorted } []}\ \text{sempty}$$

$$\frac{}{\vdash \text{sorted (cons n nil)}}\ \text{sone}$$

$$\frac{\vdash \text{sorted (cons n l)} \qquad n \leq m}{\vdash \text{sorted (cons n (cons m l))}}\ \text{stwo}$$

```
Inductive sorted {A : Type} (R : A -> A -> Prop) : list A ->  Prop :=
  | sempty : sorted R nil
  | sone : forall a : A, sorted R (cons a nil)
  | stwo : forall (a b : A) (l : list A),
            R a b -> sorted R (cons b l) ->
            sorted R (cons a (cons b l)).
```

Concept Check

Give an inductively defined proposition capturing membership in a tree:

```
Example memEx : mem 2 (Node 1 Leaf  (Node 2 Leaf Leaf)).
```

```
Inductive tree (A : Type) : Type :=
| Leaf : tree A
| Node : A -> tree A -> tree A -> tree A.
```

```
Inductive InTree {A : Type} (x : A) : tree A -> Prop :=
 | InRoot : forall (l r : tree A), InTree x (Node x l r)
 | InLeft : forall (y : A) (l r : tree A), InTree x l -> InTree x (Node y l r)
 | InRight : forall (y : A) (l r : tree A), InTree x r -> InTree x (Node y l r).
```