CS 565

Programming Languages (graduate) Spring 2025

Week 4

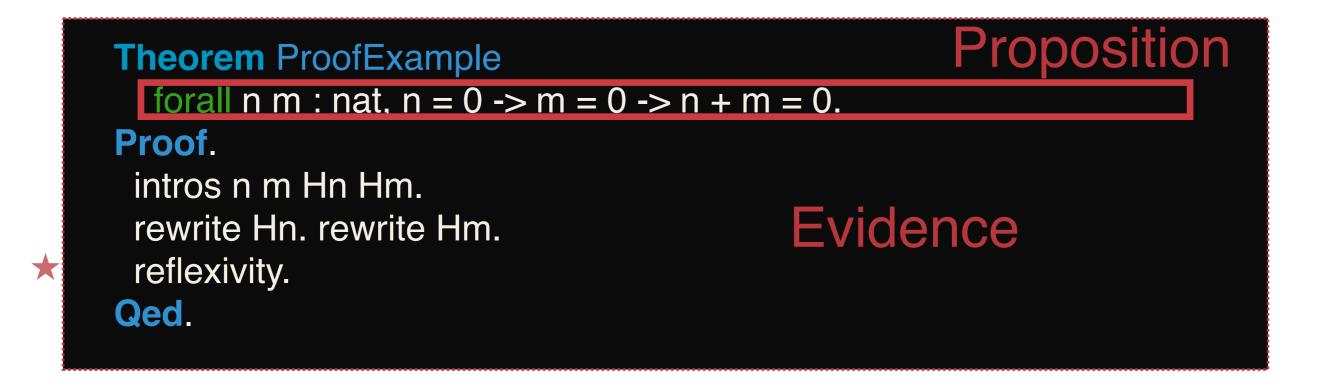
Propositions and Inductive Evidence

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.

3

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



Check (2 = 2).(* : Prop *)Check (3 = 2).(* : Prop *)Check $(3 = 2 \rightarrow 2 = 3)$.(* : Prop *)Check (forall n: nat, n = 2).(* : Prop *)

5

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 *)

6

Can have polymorphic predicates:

```
Definition injective {A B} (f : A -> B) : Prop :=
```

```
forall x y : A, f x = f y \rightarrow x = y.
```

```
Theorem plus1_inj : injective (plus 1).
```

```
Proof.
```

```
... (* unfold injective *)
```

Equality is a polymorphic binary predicate:

Check @eq. (* : $\forall A$: Type, $A \rightarrow A \rightarrow Prop$ *)

What is the type of the following expression?

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

pred (S O) = O

8

What is the type of the following expression?

A. Prop

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

 \forall n:nat, pred (S n) = n

What is the type of the following expression?

A. Prop

9

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

 \forall n:nat, S (pred n) = n

10

What is the type of the following expression?

A. Prop

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

∀ n:nat, S (pred n)

11

What is the type of the following expression?

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

fun n:nat => S (pred n)

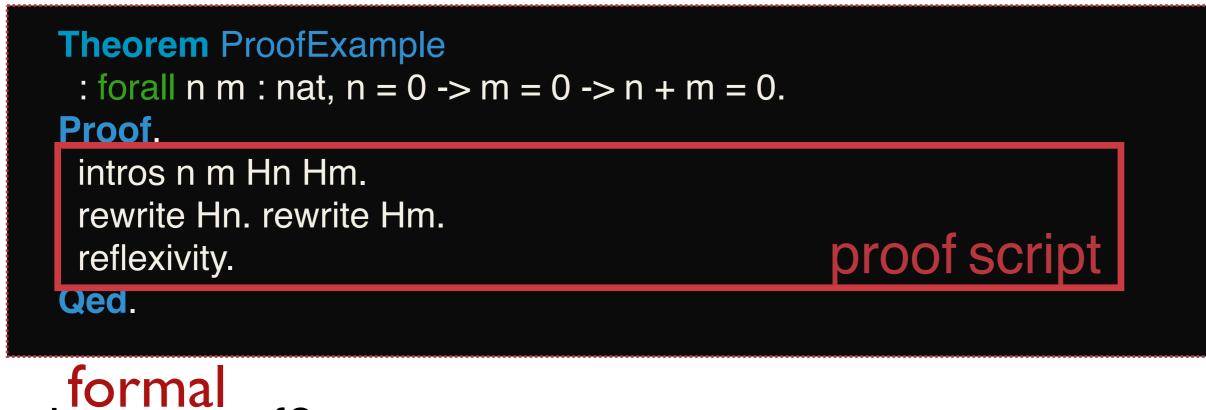
What is the type of the following expression?

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?



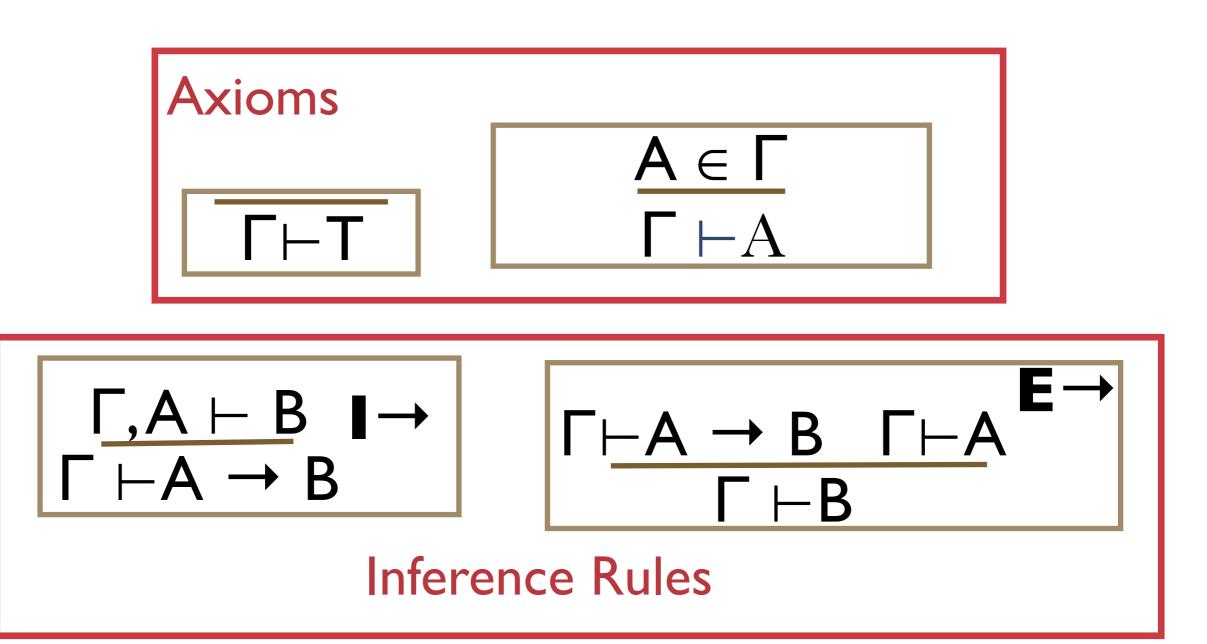
formal What is a ^ proof? 14

A judgement is a claim of a proof system

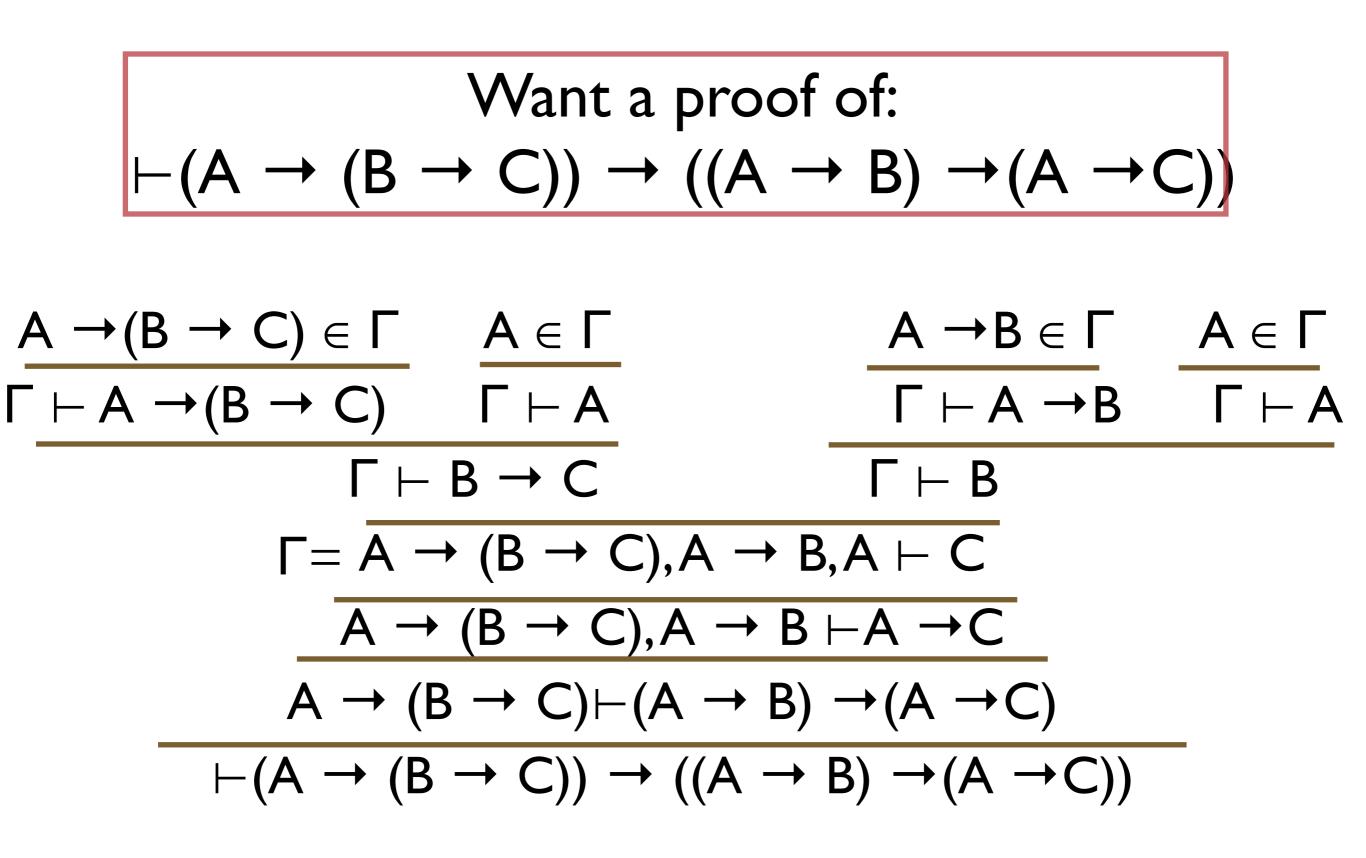
The judgement $\Gamma \vdash A$ is read as: "assuming the propositions in Γ are true, A is true". We'll see other judgements over the course of the semester:

Inference Rules

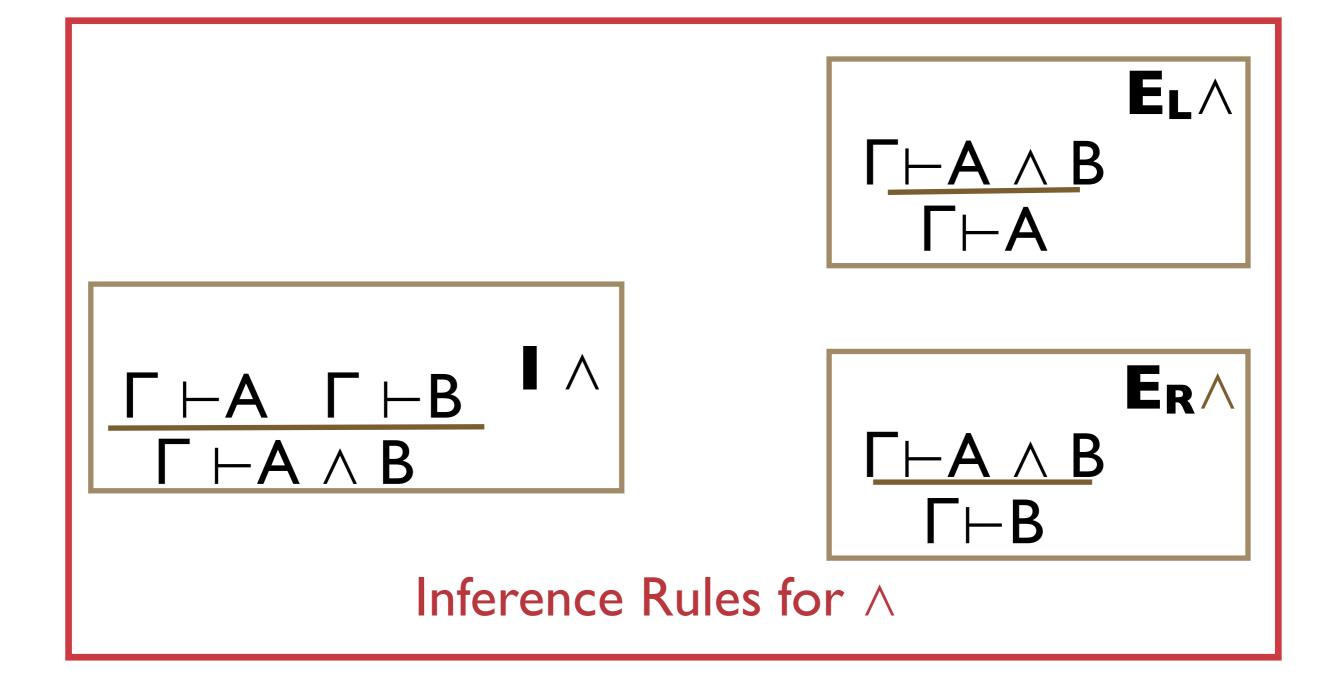
Proof systems construct evidence of judgements via inference rules:



Example Proof

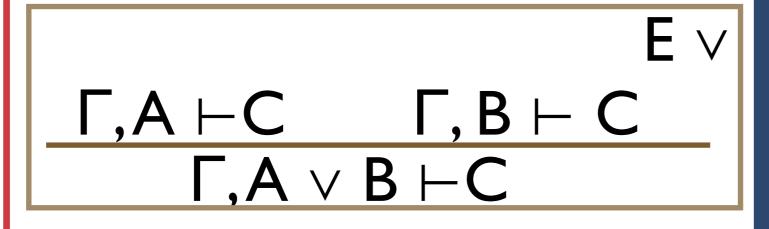


Symbol Pushing





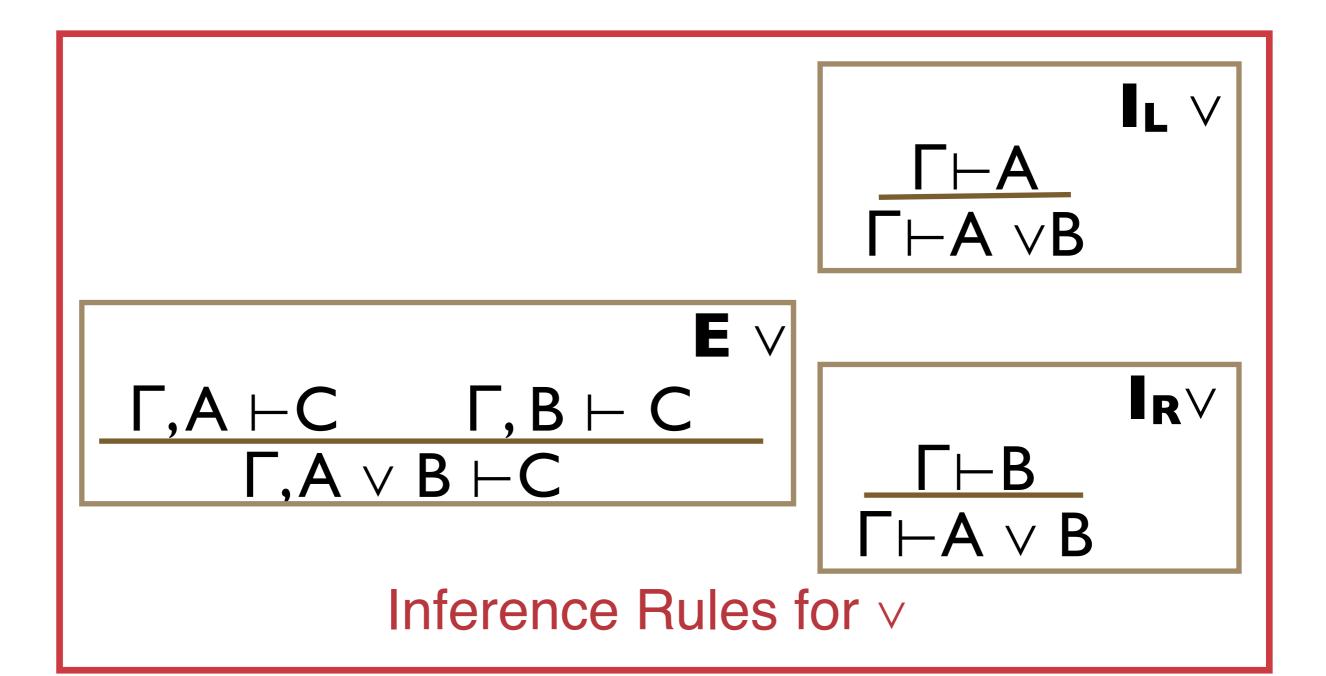
18



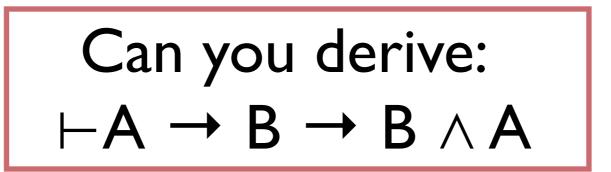
Introduction Rules for Or?

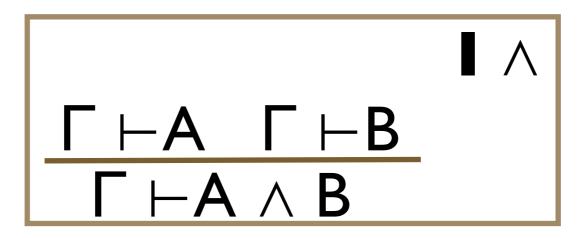
Inference Rules for \vee

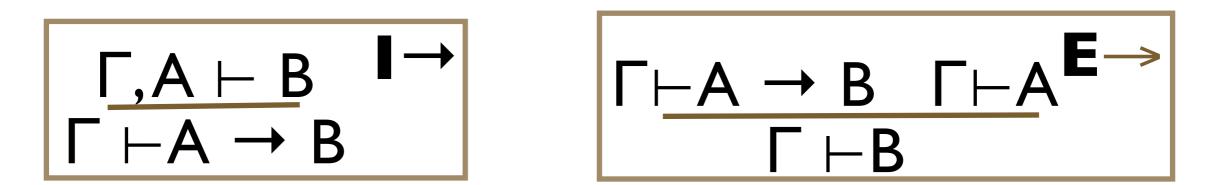










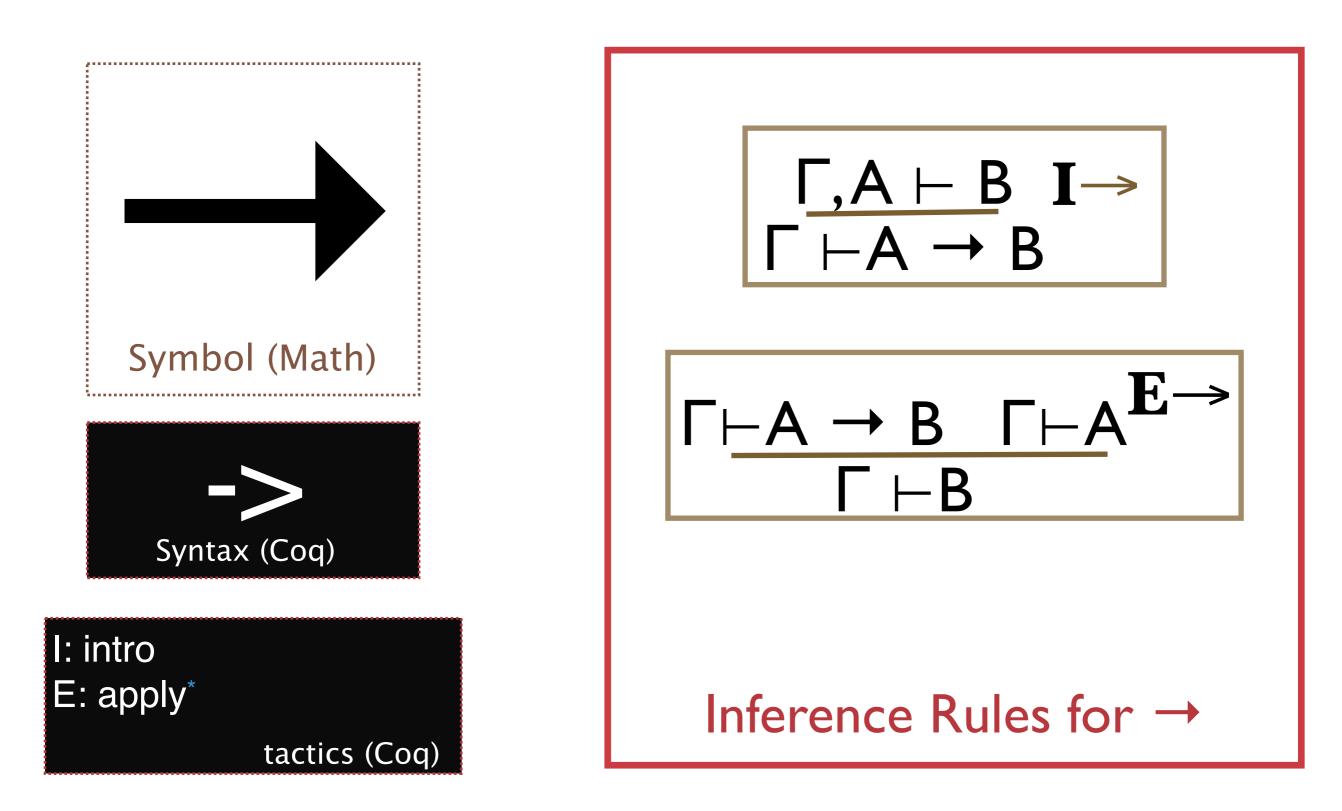


Haven't we already seen a number of proofs?

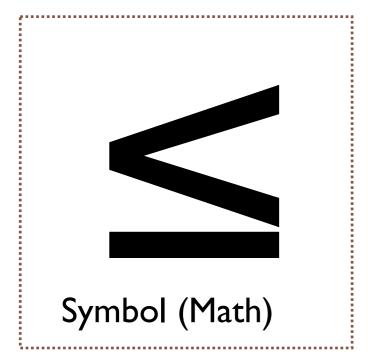
Theorem ProofExample : forall n m : nat, $n = 0 \rightarrow m = 0 \rightarrow n + m = 0$.	
Proof.	
intros n m Hn Hm.	proofscript
rewrite Hn. rewrite Hm.	prooiscript
reflexivity.	

What is a $_{\Lambda}$ proof? A proof tree in the Calculus of co-Inductive Constructions.

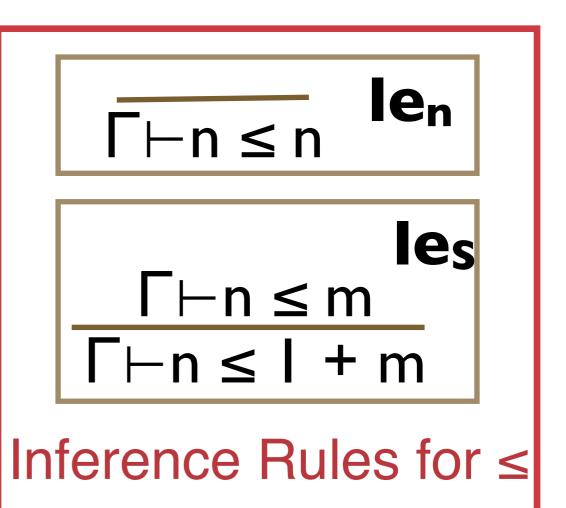
Implication



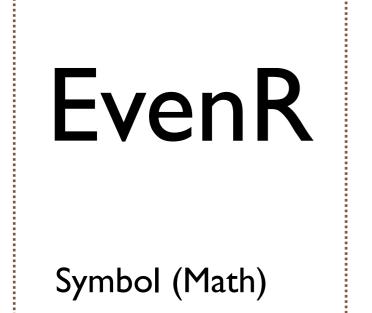
Less Than



 $n \le m \equiv \exists k. n+k \equiv m$ Definition of \leq



Even-ness





Definition of EvenR

 $\begin{array}{c} & \mathbf{ev_0} \\ \Gamma \vdash \operatorname{EvenR} 0 \end{array} \\ & \mathbf{ev_2} \\ \Gamma \vdash \operatorname{EvenR} n \\ \Gamma \vdash \operatorname{EvenR} (2+n) \end{array} \\ \end{array} \\ \begin{array}{c} & \text{Inference Rules for EvenR} \end{array} \\ \end{array}$

Less Than (Coq)

- <u>Goal</u>:

Binary relation on natural numbers



Form of evidence that two numbers belong to that relation

- <u>Step 0</u>: Name the relation:
- <u>Step 1</u>: Give the relation a signature:
- <u>Step 2</u>: Enumerate evidence:



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

Less Than (Coq)

- <u>G</u>oal:

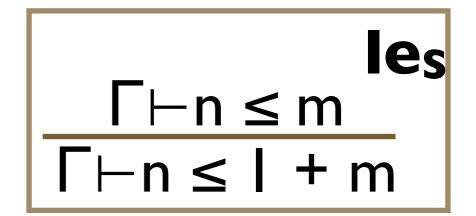
26

Binary relation on natural numbers



Form of evidence that two numbers belong to that relation

- <u>Step 0</u>: Name the relation:
- <u>Step 1</u>: Give the relation a signature:
- <u>Step 2</u>: Enumerate evidence:



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

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

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





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

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

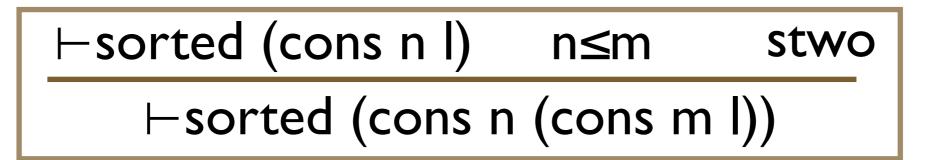
Sorted Lists (Coq)

29





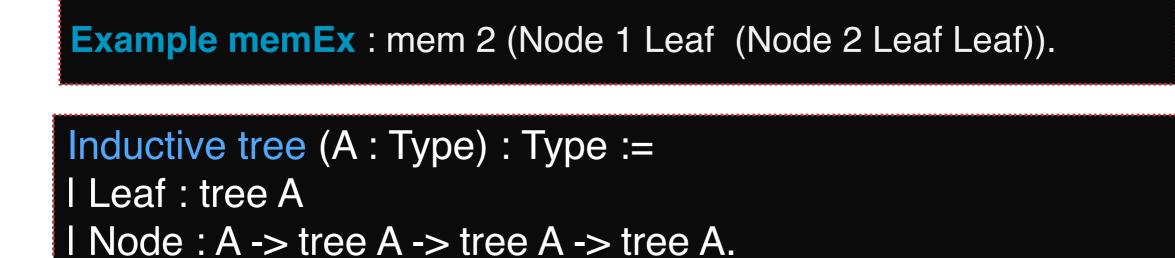
⊢sorted (cons n nil)



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

30

Give an inductively defined proposition capturing membership in a tree:



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