CS720

Logical Foundations of Computer Science

Lecture 9: Inductive propositions

Tiago Cogumbreiro

Building propositions with data structures (inductively)

Enumerated propositions

Types vs propositions

```
Inductive bit : Type := on | off.
Definition bool_to_bit (b:bool) : bit :=
    match b with
    | true => on
    | false => off
    end.
Definition bit_to_bool (b:bit) : bool :=
  match b with
  | on => true
  | off => false
  end.
Goal
    forall b,
    bool_to_bit (bit_to_bool b) = b.
```

Examples

- What is a value of `bit`?

Examples

- What is a value of `bit`? example, `off`.
- What is a value of `bit → bit`?

Examples

- What is a value of `bit`? example, `off`.
- What is a value of `bit → bit`? example, `fun (b:bit) ⇒ if b then off else on`
- What is a value of `bool → bit`?

Examples

- What is a value of `bit`? example, `off`.
- What is a value of `bit → bit`? example, `fun (b:bit) ⇒ if b then off else on`
- What is a value of `bool → bit`? example, `fun (b:bool) ⇒ if b then on else off`

Enumerated propositions

```
Inductive Bit : Prop := On | Off.

Definition bool_to_Bit (b:bool) : Bit :=
  match b with
  | true => On
  | false => Off
  end.

Definition Bit_to_bool (b:Bit) : bool :=
  match b with
  | On => true
  | Off => false
  end.
```

- Propositions cannot be the target of match

Examples of propositions and their proofs

- `Goal Bit.`

Examples of propositions and their proofs

- `Goal Bit.` You can always prove bit. Example, on
- `Goal Bit -> Bit.`

Examples of propositions and their proofs

- `Goal Bit.` You can always prove bit. Example, on
- `Goal Bit -> Bit.` If you have bit, then you can conclude bit. Example, `intros H. apply H.`
- `Goal forall b:Bit, b.`

Examples of propositions and their proofs

- `Goal Bit.` You can always prove bit. Example, on
- `Goal Bit → Bit.` If you have bit, then you can conclude bit. Example, `intros H. apply H.`
- `Goal forall b:Bit, b.` **Error!** Variable `b` is a value of `Bit`, an evidence. Cannot be used as a proposition (`Bit` is a proposition!)
- `Goal forall b:Bit, Bit.`

Examples of propositions and their proofs

- `Goal Bit.` You can always prove bit. Example, on
- `Goal Bit -> Bit.` If you have bit, then you can conclude bit. Example, `intros H. apply H.`
- `Goal forall b:Bit, b.` **Error!** Variable `b` is a value of `Bit`, an evidence. Cannot be used as a proposition (`Bit` is a proposition!)
- `Goal forall b:Bit, Bit.` If you have bit, then you can conclude bit. Example, `intros H. apply H.`
- `Goal Bit <-> True.`

Examples of propositions and their proofs

- `Goal Bit.` You can always prove bit. Example, on
- `Goal Bit -> Bit.` If you have bit, then you can conclude bit. Example, `intros H. apply H.`
- `Goal forall b:Bit, b.` **Error!** Variable `b` is a value of `Bit`, an evidence. Cannot be used as a proposition (`Bit` is a proposition!)
- `Goal forall b:Bit, Bit.` If you have bit, then you can conclude bit. Example, `intros H. apply H.`
- `Goal Bit <-> True.` Whenever you have `Bit`, you can conclude `True`, and vice versa. We are **not** saying that `Bit` **is** `True`.

Insights

- Propositions are restricted in how you can
- Equivalence between A and B, means A is provable whenever B is provable.
- Theorems are just definitions, where we don't care about how it was proved (the code), just that it ***can*** be proved

Composite inductive propositions

Disjunction

```
Inductive or (A B : Prop) : Prop :=
  | or_introl :
    A →
    or A B
  | or_intror :
    B →
    or A B
```

Conjunction

```
Inductive and (P Q : Prop) : Prop :=
| conj :
    P →
    Q →
    and P Q.
```

Adding parameters to predicates

```
Inductive Bar : nat → Prop :=
| C : Bar 1
| D : Bar 2.
```

Adding parameters to predicates

```
Inductive Bar : nat → Prop :=
| C : Bar 1
| D : forall n,
    Bar (S n).

Goal forall n,
    Bar n →
    n <> 0.
```

Alternative definition of Bar

```
Definition Bar2 n : Prop := n <> 0.
```

Existential

```
Inductive sig (A : Type) (P : A → Prop) : Type :=
  | exist : forall x : A,
    P x →
    sig A P.
```

Recursive inductive propositions

Recall the functional definition of `In`

```
Fixpoint In {A : Type} (x : A) (l : list A) : Prop :=
  match l with
  | [] ⇒ False
  | x' :: l' ⇒ x' = x \/ In x l'
  end.
```

Defining In inductively

```
Inductive In {A:Type} : A → list A → Prop :=
```

Defining In inductively

```
Inductive In {A:Type} : A → list A → Prop :=

| in_eq:
    forall x l,
    In x (x::l)
| in_cons:
    forall x y l,
    In x l →
    In x (y::l).
```

Fixed parameters in inductive propositions

```
Inductive In' {A:Type} (x: A) : list A → Prop :=
| in_eq:
    forall l,
    In' x (x::l)
| in_cons:
    forall y l,
    In' x l →
    In' x (y::l).
```

Proofs by induction on the derivation

```
Lemma in_in':
  forall (A:Type) (x:Type) l,
  In' x l →
  In x l.
Proof.
  intros.
  induction H.
```

McCarthy 91 function

- McCarthy's 91 function

$$M(n) = n - 10 \text{ if } n > 100$$
$$M(n) = M(M(n + 11)) \text{ if } n \leq 100$$

```
Inductive McCarthy91: nat → nat → Prop :=
| mc_carthy_91_gt:
  forall n,
  n > 100 →
  McCarthy91 n (n - 10)
| mc_carthy_91_le:
  forall n o m,
  n ≤ 100 →
  McCarthy91 (n + 11) m →
  McCarthy91 m o →
  McCarthy91 n o.
```

Exercise

Let us define even numbers inductively...

> In the world of propositions, what is a signature of a number being even?

Exercise

Let us define even numbers inductively...

> In the world of propositions, what is a signature of a number being even?

```
Inductive ev: nat → Prop
```

Exercise

Let us define even numbers inductively…

> In the world of propositions, what is a signature of a number being even?

```
Inductive ev: nat → Prop
```

- 0 is even
- If n is even, then $2 + n$ is also even.

Inductively defined even

- In Logic, the constructors `ev_0` and `ev_SS` of propositions can be called ***inference rules***.

```
Inductive ev: nat → Prop :=
(* Rule 1: *)
| ev_0:
  ev 0
(* Rule 2: *)
| ev_SS: forall n,
  ev n →
(*-----------*)
  ev (S (S n)).
```

Which can be typeset as an inductive definition with the following notation:

$$\frac{}{\mathrm{ev}(0)}\,\texttt{ev\_0} \qquad \frac{\mathrm{ev}(n)}{\mathrm{ev}(\mathrm{S}(\mathrm{S}(n)))}\,\texttt{ev\_SS}$$

Proving that 4 is even

$$\frac{}{\mathbf{ev}\ 0}\ \texttt{ev\_0}$$

$$\frac{}{\mathbf{ev}\ 2}\ \texttt{ev\_SS}$$

$$\frac{}{\mathbf{ev}\ 4}\ \texttt{ev\_SS}$$

Backward style: From `ev_SS` we can conclude that 4 is even, if we can show that 2 is even, which follows from `ev_SS` and the fact that 0 is even (by `ev_0`).

Forward style: From the fact that 0 is even (`ev_0`), we use theorem `ev_SS` to show that 2 is even; so, applying theorem `ev_SS` to the latter, we conclude that 4 is even.

```
Goal ev 4.
Proof. (* backward style proof *)
  apply eq_SS.
  apply eq_SS.
  apply ev_0.
Qed.

Goal ev 4.
Proof. (* forward style proof *)
  apply (ev_SS 2 (ev_SS 0 ev_0)).
Qed.
```

Reasoning about inductive propositions

```
Theorem evSS : forall n,
  ev (S (S n)) → ev n.
```

(Done in class.)

Example

```
Goal ~ ev 3.
```

(Done in class.)

Proofs by induction

```
Goal forall n, ev n -> ~ ev (S n).
```

(Done in class.)

Proofs by induction

```
Goal forall n, ev n -> ~ ev (S n).
```

(Done in class.)

> Notice the difference between induction on `n` and on judgment `ev n`.

Relations in Coq

```
Inductive le : nat → nat → Prop :=
  | le_n :
    forall n,
    le n n

  | le_S :
    forall n m,
    le n m →
    le n (S m).
Notation "n ≤ m" := (le n m).
```

$$\frac{}{n \leq n}\,\texttt{le\_n} \qquad \frac{n \leq m}{n \leq \mathtt{S}\ m}\,\texttt{le\_S}$$

Exercise

```
Goal 3 ≤ 6.
```

Less-than

```
Definition lt (n m:nat) := le (S n) m.
```

> How do we prove that this definition is correct?

Less-than

```
Definition lt (n m:nat) := le (S n) m.
```

> How do we prove that this definition is correct?

```
Goal n ≤ m ↔ lt n m \/ n = m.
```

Less-than

How can we define Less-Than inductively?

Less-than

How can we define Less-Than inductively?

```
Inductive lt : nat → nat → Prop :=
  | lt_base :
    forall n,
    lt n (S n)

  | lt_S :
    forall n m,
    lt n m →
    lt n (S m).
Notation "n < m" := (lt n m).
```

How do we prove that this definition is correct?

Exercises on Less-Than

Prove that

1. $<$ is transitive
2. $<$ is irreflexive
3. $<$ is asymmetric
4. $<$ is decidable