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**
    ∀ 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 $\text{bit} \rightarrow \text{bit}$? example, $\text{fun } (b:\text{bit}) \Rightarrow \text{if } b \text{ then off else on}$
• What is a value of $\text{bool} \rightarrow \text{bit}$?
Examples

- What is a value of bit? example, off.
- What is a value of $\text{bit} \rightarrow \text{bit}$? example, \text{fun} (b:bit) \text{⇒} if b then off else on
- What is a value of bool $\rightarrow$ bit? example, \text{fun} (b:bool) \text{⇒} 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 $\rightarrow$ 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 $\rightarrow$ 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 $\leftrightarrow$ 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

\[
\text{Inductive } \text{or} (A \ B : \text{Prop}) : \text{Prop} := \\
\text{or}\text{-}\text{introl} : A \rightarrow \text{or} A \ B \\
\text{or}\text{-}\text{intror} : B \rightarrow \text{or} A \ B
\]
Conjunction

\[
\text{Inductive and } (P \ Q : \text{Prop}) : \text{Prop} := \\
\mid \text{conj} : \\
\quad P \rightarrow \\
\quad Q \rightarrow \\
\quad \text{and } P \ Q.
\]
Adding parameters to predicates

```
Inductive Bar : nat → Prop :=
  | C : Bar 1
  | D : Bar 2.
```
Adding parameters to predicates

```ocaml
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** \( \text{Bar2} \ n : \text{Prop} := n <> 0. \)
Existential

\[
\text{Inductive } \text{sig} \ (A \ : \ \text{Type}) \ (P \ : \ A \rightarrow \ \text{Prop}) \ : \ \text{Type} \ := \\
\quad | \ \text{exist} \ : \ \forall x : A, \ P \ x \ \\
\quad \text{sig} \ A \ P.
\]
Recursive inductive propositions
Recall the functional definition of \( \text{In} \):

\[
\text{Fixpoint } \text{In} \ {\{A : \text{Type}\}} \ (x : A) \ (l : \text{list A}) : \text{Prop} :=
\text{match } l \ \text{with}
    | \[] \ \Rightarrow \text{False}
    | x' :: l' \ \Rightarrow \ x' = x \ \lor \ \text{In} \ x \ l'
\text{end}.
\]
Defining In inductively

\[
\text{Inductive } \text{In } \{A:\text{Type}\} : A \to \text{list } A \to \text{Prop} :=
\]
Defining \texttt{In} inductively

\begin{verbatim}
Inductive \texttt{In} \{A:Type\} : A \to \text{list } A \to \text{Prop} :=
| \texttt{in_eq}:
  \forall x l, \texttt{In } x \text{ (}x::l\text{)}
| \texttt{in_cons}:
  \forall x y l, \texttt{In } x \text{ l} \Rightarrow \texttt{In } x \text{ (}y::l\text{)}.
\end{verbatim}
Fixed parameters in inductive propositions

\[
\text{Inductive } \text{In'} \{A: \text{Type}\} (x: A): \text{list } A \rightarrow \text{Prop} := \\
| \text{in_eq:} \\
\quad \forall l, \\
\quad \text{In'} x (x::l) \\
| \text{in_cons:} \\
\quad \forall y l, \\
\quad \text{In'} x l \rightarrow \\
\quad \text{In'} x (y::l).
\]
Proofs by induction on the derivation

Lemma \text{in\_in'}:
\[
\forall (A: \text{Type}) (x: \text{Type}) \, \text{In'} x 1 \rightarrow \text{In} x 1.
\]

Proof.
\begin{itemize}
  \item \text{intros.}
  \item \text{induction H.}
\end{itemize}
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:
  - \(\forall n,\)
  - \(n > 100 \rightarrow\)
  - McCarthy91 n (n - 10)
- mc_carthy_91_le:
  - \(\forall n o m,\)
  - \(n \leq 100 \rightarrow\)
  - McCarthy91 (n + 11) m \rightarrow
  - McCarthy91 m o \rightarrow
  - 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 \texttt{ev\_0} and \texttt{ev\_SS} of propositions can be called \textit{inference rules}.

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

\[
\begin{align*}
\text{ev\_0} & \quad \text{// } ev(0) \\
\text{ev\_SS} & \quad \text{// } ev(S(S(n)))
\end{align*}
\]
Proving that 4 is even

| ___________ ev_0          |
| ev 0         |
| ___________ ev_SS        |
| ev 2         |
| ___________ ev_SS        |
| ev 4         |

**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 $\sim ev\ 3.$

*(Done in class.)*
Proofs by induction

Goal \( \forall n, \text{ev } n \rightarrow \neg \text{ev } (S \ n) \).

(Done in class.)
Proofs by induction

Goal: forall n, ev n \rightarrow \sim ev (S n).

(Done in class.)

Notice the difference between induction on n and on judgment ev n.
Inductive le : nat → nat → Prop :=
  | le_n : ∀ n, le n n
  | le_S : ∀ n m, le n m → le n (S m).

Notation "n ≤ m" := (le n m).
Exercise

Goal $3 \leq 6$. 
Less-than

**Definition** \( \text{lt} \ (n \ m: \text{nat}) := \text{le} \ (\text{S} \ n) \ m. \)

How do we prove that this definition is correct?
Less-than

Definition \( \text{lt} \ (n \ m:\text{nat}) \ := \text{le} \ (S \ n) \ m. \)

How do we prove that this definition is correct?

Goal \( n \leq m \iff \text{lt} \ n \ m \lor n = m. \)
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