Introduction to the Theory of Computation

Lecture 3: Induction principle

Tiago Cogumbreiro
Today we will learn...

- Rewriting tactics
- Case analysis tactics
- Induction tactics
- Induction principle
Rewriting terms
Theorem plus_id_example : \( \forall \) n m:nat,
\[ n = m \rightarrow \]
\[ n + n = m + m. \]
Proof.
\begin{align*}
\text{intros } n. \\
\text{intros } m.
\end{align*}
Multiple pre-conditions in a lemma

Theorem plus_id_example : forall n m:nat, 
  n = m -> 
  n + n = m + m.

Proof. 
  intros n. 
  intros m. 

yields

1 subgoal
n, m : nat
----------------------------------(1/1)
 n = m -> n + n = m + m
Multiple pre-conditions in a lemma

applying intros H yields

1 subgoal
n, m : nat
H : n = m

(1/1)

1 + 1 = 2 + 2

How do we use H? **New tactic:** use rewrite \(\rightarrow\) H (lhs becomes rhs)

1 subgoal
n, m : nat
H : n = m

(1/1)

m + m = m + m

How do we conclude? Can you write a Theorem that replicates the proof-state above?
Let us prove this example

Theorem plus_id_exercise : forall n m o : nat, 
    n = m \rightarrow m = o \rightarrow n + m = m + o.
Proof.

(Done in class...)
Comparing naturals

Consider this recursive function that tests if two naturals are equal.

```
Fixpoint beq_nat (n m : nat) : bool :=
  match n with
  | 0 ⇒ match m with
    | 0 ⇒ true
    | S m' ⇒ false
  end
  | S n' ⇒ match m with
    | 0 ⇒ false
    | S m' ⇒ beq_nat n' m'
  end
end.
```
How do we prove this example?

**Theorem** `plus_1_neq_0_firsttry` : `forall n : nat, beq_nat (plus n 1) 0 = false.``

**Proof.**

```
intros n.
```

yields

```
1 subgoal
n : nat

-------------------------(1/1)
beq_nat (plus n 1) 0 = false
```
How do we prove this example?

**Theorem** `plus_1_neq_0_firsttry` : `forall` `n` : nat,
  `beq_nat (plus n 1) 0` = false.

**Proof.**
  `intros` `n`.

yields

1 subgoal
n : nat

-----------------------------------------------(1/1)
`beq_nat (plus n 1) 0` = false

**Apply simpl and it does nothing. Apply reflexivity:**

In environment
n : nat
Unable to **unify** "false" with "`beq_nat (plus n 1) 0"."
Why does simpl fail?

Q: Why can't beq_nat (n + 1) be simplified? (Hint: inspect its definition.)
Why does simpl fail?

Q: Why can't beq_nat (n + 1) be simplified? (Hint: inspect its definition.)
A: beq_nat expects the first parameter to be either 0 or S ?n, but we have an expression n + 1 (or plus n 1).
Why does simpl fail?

Q: Why can't beq_nat (n + 1) be simplified? (Hint: inspect its definition.)
A: beq_nat expects the first parameter to be either 0 or S ?n, but we have an expression n + 1 (or plus n 1).
Q: Can we simplify plus n 1?
Why does simpl fail?

Q: Why can't \text{beq_nat} (n + 1) be simplified? (Hint: inspect its definition.)
A: \text{beq_nat} expects the first parameter to be either 0 or \text{S } ?n, but we have an expression \text{n + 1} (or plus \text{n 1}).

Q: Can we simplify plus \text{n 1}?
A: No because plus decreases on the first parameter, not on the second!
Case analysis
Case analysis (1/3)

Let us try to inspect value n. Use: destruct n as [n'].

2 subgoals

-----------------------------(1/2)
beq_nat (0 + 1) 0 = false

-----------------------------(2/2)
beq_nat (S n' + 1) 0 = false

Now we have two goals to prove!

1 subgoal

------------------------(1/1)
beq_nat (0 + 1) 0 = false

How do we prove this?
Case analysis (2/3)

After we conclude the first goal we get:
This subproof is complete, but there are some unfocused goals:

______________________________________(1/1)
beq_nat (S n' + 1) 0 = false
Use another bullet (-).

| 1 subgoal |
n' : nat

______________________________________(1/1)
beq_nat (S n' + 1) 0 = false

And prove the goal above as well.

Why can the latter be simplified?
Case analysis (3/3)

- Use: `destruct n as [| n']` when you want to explicitly name the variables being introduced.
- Otherwise, use: `destruct n` and let Coq automatically name the variables.

Using automatically generated variable names makes the proofs more brittle to change.
Example: prove this lemma (1/4)

**Theorem** `plus_n_0` : \( \forall n: \text{nat}, \ n = n + 0. \)

**Proof.**
Example: prove this lemma (1/4)

\textbf{Theorem} \texttt{plus\_n\_0} : \texttt{forall} n: \texttt{nat}, \\
\hspace{1cm} n = n + 0. \\
\textbf{Proof}. \\

\texttt{Tactic simpl does nothing.}
Example: prove this lemma (1/4)

Theorem plus_n_0 : forall n:nat,  
    n = n + 0.  
Proof.

Tactic simpl does nothing. Tactic reflexivity fails.
Example: prove this lemma (1/4)

Theorem plus_n_0 : forall n:nat, n = n + 0.

Proof.

Tactic simpl does nothing. Tactic reflexivity fails. Apply destruct n.

2 subgoals

1/2

0 = 0 + 0

2/2

S n = S n + 0
Example: prove this lemma (2/4)

After proving the first, we get

```
1 subgoal
n : nat
------------------------------------------(1/1)
S n = S n + 0
```

Applying `simplify` yields:

```
1 subgoal
n : nat
------------------------------------------(1/1)
S n = S (n + 0)
```
Example: prove this lemma (2/4)

After proving the first, we get

```
1 subgoal
n : nat
_____________________________(1/1)
S n = S n + 0
```

Applying `simpl` yields:

```
1 subgoal
n : nat
_____________________________(1/1)
S n = S (n + 0)
```

Tactic reflexivity fails and there is nothing to rewrite.
We need an induction principle of $\text{nat}$

For some property $P$ we want to prove.

- Show that $P(0)$ holds.
- Given the induction hypothesis $P(n)$, show that $P(n + 1)$ holds.

Conclude that $P(n)$ holds for all $n$. 
Example: prove this lemma (3/4)

Apply induction \( n \).

2 subgoals

\( 0 = 0 + 0 \) (1/2)

\( S \ n = S \ n + 0 \) (2/2)

How do we prove the first goal?

Compare induction \( n \) with destruct \( n \).
Example: prove this lemma (4/4)

After proving the first goal we get
1 subgoal
n : nat
IHn : n = n + 0
______________________________________(1/1)
S n = S n + 0
applying simpl yields
1 subgoal
n : nat
IHn : n = n + 0
______________________________________(1/1)
S n = S (n + 0)

How do we conclude this proof?
Theorem mult_0_plus': forall n m : nat, (0 + n) * m = n * m.

Proof.
intros n m.
assert (H: 0 + n = n). { reflexivity. }
rewrite H.
reflexivity. Qed.

- H is a variable name, you can pick whichever you like.
- Your intermediary result will capture all of the existing hypothesis.
- It may include forall.
- We use braces { and } to prove a sub-goal.