CS420

Introduction to the Theory of Computation

Lecture 14: Case analysis & proof by induction

Tiago Cogumbreiro

Today we will...

- Rewriting terms: using equality assumption
- Case analysis: inspecting values
- Proofs by induction: generalizing case analysis

> Chapters `Basics.v` and `Induction.v`

Rewriting terms

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.
```

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)
n + n = m + m
```

How do we use `H`? **New tactic:** use `rewrite -> 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 -> m = o -> 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
  | O ⇒ match m with
         | O ⇒ true
         | S m' ⇒ false
         end
  | S n' ⇒ match m with
            | O ⇒ 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 `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`?

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.

Induction.v

Example: prove this lemma (1/4)

```
Theorem plus_n_O : forall n:nat,
  n = n + 0.
Proof.
```

Example: prove this lemma (1/4)

```
Theorem plus_n_O : forall n:nat,
  n = n + 0.
Proof.
```

Tactic `simpl` does nothing.

Example: prove this lemma (1/4)

```
Theorem plus_n_O : forall n:nat,
  n = n + 0.
Proof.
```

Tactic `simpl` does nothing. Tactic `reflxivity` fails.

Example: prove this lemma (1/4)

```
Theorem plus_n_O : forall n:nat,
  n = n + 0.
Proof.
```

Tactic `simpl` does nothing. Tactic `reflxivity` 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 `simpl` 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 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
______________________________________(1/2)
0 = 0 + 0
______________________________________(2/2)
S n = S n + 0
```

> 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?

Intermediary results

```
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.

Formal versus informal proofs

- The objective of a mechanical (formal) proofs is to appease the proof checker.
- The objective of an informal proof is to convince (logically) the reader.
- `ltac` proofs are imperative, assume the reader can step through
- In informal proofs we want to help the reader reconstruct the proof state.

Reading an `ltac` proof

```
Theorem plus_assoc : forall n m p : nat,
  n + (m + p) = (n + m) + p.
Proof.
  intros n m p. induction n as [| n' IHn'].
  - reflexivity.
  - simpl. rewrite -> IHn'. reflexivity. Qed.
```

1. The proof follows by induction on n.

Reading an `ltac` proof

```
Theorem plus_assoc : forall n m p : nat,
  n + (m + p) = (n + m) + p.
Proof.
  intros n m p. induction n as [| n' IHn'].
  - reflexivity.
  - simpl. rewrite -> IHn'. reflexivity. Qed.
```

1. The proof follows by induction on n.
2. In the base case, we have that $n = 0$. We need to show $0 + (m + p) = 0 + m + p$, which follows by the definition of $+$.

Reading an `ltac` proof

```
Theorem plus_assoc : forall n m p : nat,
  n + (m + p) = (n + m) + p.
Proof.
  intros n m p. induction n as [| n' IHn'].
  - reflexivity.
  - simpl. rewrite → IHn'. reflexivity. Qed.
```

1. The proof follows by induction on n.
2. In the base case, we have that $n = 0$. We need to show $0 + (m + p) = 0 + m + p$, which follows by the definition of $+$.
3. In the inductive case, we have $n = \texttt{S}\ n'$ and must show $Sn' + (m + p) = Sn' + m + p$.
   From the definition of $+$ it follows that $\texttt{S}\ (n' + (m + p)) = \texttt{S}\ (n' + m + p)$.
   The proof concludes by applying the induction hypothesis $n' + (m + p) = n' + m + p$
   .

Basic.v

- Learn: interplay between `forall`, `simpl`, and `reflexivity`
- New syntax: → to represent implication
- New tactic: `rewrite` to replace terms using equality
- New tactic: `destruct` to perform case analysis
- New tactic: bullets (`-`, `*`, and `+`) and scopes (`{` and `}`)

`Induction.v`

- Learn: induction principle for natural numbers.
- New tactic: `induction`
- New tactic: `assert`
- Learn: formal vs informal proofs

Ltac vocabulary

- `simpl`
- `reflexivity`
- `intros`
- `rewrite`
- `destruct`
- `induction`
- `assert`