CS720

Logical Foundations of Computer Science

Lecture 6: Tactics (continued)

Tiago Cogumbreiro

Today we will…

- Take a deeper look at proofs by induction
- Unfolding definitions
- Simplifying expressions
- Destructing compound expressions

Why are we learning this?

- To make your proofs smaller/simpler
- Many interesting properties require what we will learn today about induction

Varying the Induction Hypothesis (1/2)

Varying the Induction Hypothesis (1/2)

```
Fixpoint double (n:nat) := match n with | O ⇒ O | S n' ⇒ S (S (double n')) end.

Theorem double_injective_FAILED : forall n m,
     double n = double m →
     n = m.
Proof.
  intros n m. induction n as [| n'].
  - (* n = O *) simpl. intros eq. destruct m as [| m'].
    + (* m = O *) reflexivity.
    + (* m = S m' *) discriminate eq.
  - (* n = S n' *) intros eq.
```

(Proof state in the next slide.)

Varying the Induction Hypothesis (2/2)

```
1 subgoal
n', m : nat
IHn' : double n' = double m → n' = m
eq : double (S n') = double m
______________________________________(1/1)
S n' = m
```

0. Know that: $S(n') = n$, thus $double(n)$ became $double(S(n'))$
1. Know that: If $double(n') = double(m)$, then $n' = m$ ☚ Can we prove the pre?
2. Know that: $double(\underbrace{S(n')}_{n}) = double(m)$, thus $S(S(double(n'))) = double(m)$
3. Show that: $S(n') = m$

> Where do we go from this? How can we use the induction hypothesis?

Recall the induction principle of nats

We performed induction on `n` and our goal is `double n = double m → n = m`
That is, prove `P(n) := double n = double m → n = m` by induction on `n`.

- Prove `P(0)`, thus replace `n` by `0` in `P(n)`:
  Prove `double 0 = double m → 0 = m`
- Prove that `P(n)` implies `P(n+1)`:
  Given `double n = double m → n = m` prove that `double (n + 1) = double m → n = m`.

> What is impeding our proof?

Recall the induction principle of nats

We performed induction on `n` and our goal is `double n = double m → n = m`
That is, prove `P(n) := double n = double m → n = m` by induction on `n`.

- Prove `P(0)`, thus replace `n` by `0` in `P(n)`:
  Prove `double 0 = double m → 0 = m`
- Prove that `P(n)` implies `P(n+1)`:
  Given `double n = double m → n = m` prove that `double (n + 1) = double m → n = m`.

> What is impeding our proof?

The problem is that the goal we are proving fixes the `m`, however in the expression `double n = double m` the `n` and the `m` are **related**!

Since the induction variable `n` "influences" `m`, then we must generalize `m`.

How do we fix it?

How do we generalize a variable?

We perform induction on `n` and our goal `P(n)` becomes:

```
forall m, double n = double m → n = m
```

By performing induction on `n` we get:

- `P(0) = forall m, double 0 = double m → 0 = m`
- `P(n) → P(n+1) =`
  `(forall m, double n = double m → n = m) →`
  (forall m, double (n + 1) = double m)`

Let us try again

```
Theorem double_injective : forall n m,
     double n = double m →
     n = m.
Proof.
  intros n. induction n as [| n'].
```

(Done in class.)

Second try

```
Theorem double_injective : forall m n,
     double n = double m ->
     n = m.
Proof.
  intros m n eq1.
```

Notice how `m` and `n` are switched.

(Done in class.)

Second try

```
Theorem double_injective : forall m n,
     double n = double m →
     n = m.
Proof.
  intros m n eq1.
```

Notice how `m` and `n` are switched.

(Done in class.)

> - `generalize dependent n`: generalizes (abstracts) variable `n`
> - ***Takeaway:*** the induction variable should be the left-most in a `forall` binder

Destruct compound expressions

Destruct compound expressions

Destruct works for any expressions, not just variables

```
Definition sillyfun (n : nat) : bool :=
  if Nat.eqb n 3 then false
  else if Nat.eqb n 5 then false
  else false.

Theorem sillyfun_false : forall (n : nat),
  sillyfun n = false.
Proof.
  intros n. unfold sillyfun.
  destruct (Nat.eqb n 3).
```

(Completed in class.)

Destruct compound expressions

Destruct works for any expressions, not just variables

```
Definition sillyfun1 (n : nat) : bool :=
  if Nat.eqb n 3 then true
  else if Nat.eqb n 5 then true
  else false.

Theorem sillyfun1_odd : forall (n : nat),
     sillyfun1 n = true ->
     oddb n = true.
Proof.
  intros n eq1. unfold sillyfun1 in eq1.
  destruct (Nat.eqb n 3).
```

Destruct compound expressions

Destruct works for any expressions, not just variables

```
Definition sillyfun1 (n : nat) : bool :=
  if Nat.eqb n 3 then true
  else if Nat.eqb n 5 then true
  else false.

Theorem sillyfun1_odd : forall (n : nat),
     sillyfun1 n = true →
     oddb n = true.
Proof.
  intros n eq1. unfold sillyfun1 in eq1.
  destruct (Nat.eqb n 3).
```

> What happened here? We lost our knowledge. Use `destruct PATTERN eqn:H`.

Unfolding Definitions

Unfolding Definitions

```
Definition square n := n * n.

Lemma square_mult : forall n m, square (n * m) = square n * square m.
Proof.
  intros n m.
  simpl.
```

How do we prove this?

Unfolding Definitions

```
Definition square n := n * n.

Lemma square_mult : forall n m, square (n * m) = square n * square m.
Proof.
  intros n m.
  simpl.
```

> How do we prove this?

Use `unfold square` to "open" the definition.

> Function `square` is not "simplifiable". A "simplifiable" function performs a match in the argument ***and*** inspects the structure of the argument.

Simplifiable expressions

Which of `e`, `f 0`, `g 5`, `i 5`, and `h 5` simplify?

```
Definition e := 5.

Definition f (x:nat) := 5.

Definition g (x:nat) := x.

Definition i (x:nat) := match x with _ ⇒ x end.

Definition h (x:nat) :=
  match x with
  | S _ ⇒ x
  | 0 ⇒ x
  end.
```

Non-simplifiable expressions

```
Definition e := 5.
Goal f = 5. Proof. simpl. Abort.
Definition f (x:nat) := 5.
Goal f 0 = 5. Proof. simpl. Abort.
(* no match, simplify cannot unfold *)
Definition g (x:nat) := x.
Goal g 5 = 5. Proof. simpl. Abort.
(* match, but no inspection *)
Definition i (x:nat) := match x with _ ⇒ x end.
Goal i 5 = 5. Proof. simpl. Abort.
(* match inspects the argument *)
Definition h (x:nat) :=
  match x with
  | S _ ⇒ x | 0 ⇒ x
  end.
Goal h 5 = 5. Proof. simpl. reflexivity. Qed.
```

If `simpl` does nothing, try unfolding the definition, to understand why `simpl` is stuck.