Logic Foundations
Tactics: More Basic Tactics

熊英飞 胡振江
信息学院计算机系

2021年3月26日

The apply Tactic

A Silly Example

We often encounter situations where the goal to be proved is exactly the same as some hypothesis in the context or some previously proved lemma.

```
Theorem silly1 : forall (n m o p : nat),
    n = m  ->
    [n;o] = [n;p] ->
    [n;o] = [m;p].
Proof.
  intros n m o p eq1 eq2.
  rewrite <- eq1.
  rewrite -> eq2.  reflexivity.
Qed.
```

A Silly Example

We often encounter situations where the goal to be proved is exactly the same as some hypothesis in the context or some previously proved lemma.

```
Theorem silly1 : forall (n m o p : nat),
    n = m  ->
    [n;o] = [n;p] ->
    [n;o] = [m;p].
Proof.
  intros n m o p eq1 eq2.
  rewrite <- eq1.
  apply eq2.
Qed.
```

Another Silly Example

The apply tactic also works with conditional hypotheses and lemmas: if the statement being applied is an implication, then the premises of this implication will be added to the list of subgoals needing to be proved.

```
Theorem silly2 : forall (n m o p : nat),
   n = m ->
   (n = m -> [n;o] = [m;p]) ->
   [n;o] = [m;p].
Proof.
 intros n m o p eq1 eq2.
 apply eq2. apply eq1.  Qed.

Theorem silly2a : forall (n m : nat),
    (n,n) = (m,m)  ->
    (forall (q r : nat), (q,q) = (r,r) -> [q] = [r]) ->
    [n] = [m].
Proof.
 intros n m eq1 eq2.
 apply eq2. apply eq1.  Qed.
```

The "apply with" Tactics

```
Example trans_eq_example : forall (a b c d e f : nat),
    [a;b] = [c;d] ->
    [c;d] = [e;f] ->
    [a;b] = [e;f].
Proof.
 intros a b c d e f eq1 eq2.
 rewrite -> eq1. rewrite -> eq2. reflexivity.  Qed.
```

⬇ generalization

```
Theorem trans_eq : forall (X:Type) (n m o : X),
 n = m -> m = o -> n = o.
Proof.
 intros X n m o eq1 eq2. rewrite -> eq1. rewrite -> eq2.
 reflexivity.  Qed.
```

The “apply with” Tactics

```
Example trans_eq_example : forall (a b c d e f : nat),
    [a;b] = [c;d] ->
    [c;d] = [e;f] ->
    [a;b] = [e;f].
Proof.
  intros a b c d e f eq1 eq2.
  apply trans_eq with (m:=[c;d]).
  apply eq1. apply eq2.  Qed.
```

↑ application

```
Theorem trans_eq : forall (X:Type) (n m o : X),
  n = m -> m = o -> n = o.
Proof.
  intros X n m o eq1 eq2. rewrite -> eq1. rewrite -> eq2.
  reflexivity.  Qed.
```

The injection and discriminate Tactics

Injection: Injectivity of Constuctors

```
Theorem S_injective : forall (n m : nat),
 S n = S m ->
 n = m.
Proof.
 intros n m H1.
 assert (H2: n = pred (S n)). { reflexivity. }
 rewrite H2. rewrite H1. reflexivity.
Qed.
```

Constructor is inversible (destructor)

```
Theorem S_injective' : forall (n m : nat),
 S n = S m ->
 n = m.
Proof.
 intros n m H.
 injection H as Hnm. apply Hnm.
Qed.
```

Injection: : Injectivity of Constuctors

```
Theorem injection_ex1 : forall (n m o : nat),
  [n; m] = [o; o] ->
  [n] = [m].
Proof.
  intros n m o H.
  injection H as H1 H2.
  rewrite H1. rewrite H2. reflexivity.
Qed.
```

Discriminate: Disjointness of Constructors

```
Theorem eqb_0_l : forall n,
  0 =? n = true -> n = 0.
Proof.
 intros n.
 destruct n as [| n'] eqn:E.
 - (* n = 0 *)
  intros H. reflexivity.
 - (* n = S n' *)
  simpl.
  intros H. discriminate H.
Qed.
```

```
1 subgoal
n, n' : nat
E : n = S n'
H : false = true
______________
(1/1)
S n' = 0
```

Principle of explosion (爆炸原理)：从矛盾中推出一切

The "f_equal" Tactic

```
Theorem f_equal : forall (A B : Type) (f g: A -> B) (x y: A),
 f = g -> x = y -> f x = g y.
Proof. intros A B f g x y eq1 eq2. rewrite eq1. rewrite eq2.  reflexivity.  Qed.

Theorem eq_implies_succ_equal : forall (n m : nat),
   n = m -> S n = S m.
Proof. intros n m H. apply f_equal. reflexivity. apply H. Qed.


Theorem eq_implies_succ_equal' : forall (n m : nat),
   n = m -> S n = S m.
Proof. intros n m H. f_equal. apply H. Qed.
```

Using Tactics on Hypotheses

Forward Reasoning

```
Theorem silly3' : forall (n : nat),
  (n =? 5 = true -> (S (S n)) =? 7 = true) ->
  true = (n =? 5)  ->
  true = ((S (S n)) =? 7).
Proof.
  intros n eq H.
  symmetry in H. apply eq in H. symmetry in H.
  apply H.  Qed.
```

```
1 subgoal
n : nat
eq : (n =? 5) = true ->
    (S (S n) =? 7) = true
H : (n =? 5) = true
______________________(1/1)
true = (S (S n) =? 7)
```

→

```
1 subgoal
n : nat
eq : (n =? 5) = true ->
    (S (S n) =? 7) = true
H : (S (S n) =? 7) = true
______________________(1/1)
true = (S (S n) =? 7)
```

Revisit: Backward Reasoning

```
Theorem silly2 : forall (n m o p : nat),
   n = m ->
   (n = m -> [n;o] = [m;p]) ->
   [n;o] = [m;p].
Proof.
  intros n m o p eq1 eq2.
  apply eq2. apply eq1.  Qed.
```

```
1 subgoal
n, m : nat
eq1 : (n, n) = (m, m)
eq2 : forall q r : nat,
    (q, q) = (r, r) ->
    [q] = [r]
______________________(1/1)
[n] = [m]
```

➡

```
1 subgoal
n, m : nat
eq1 : (n, n) = (m, m)
eq2 : forall q r : nat,
    (q, q) = (r, r) ->
    [q] = [r]
______________________(1/1)
(n, n) = (m, m)
```

Varying the Induction Hypothesis

Problem 1: Introducing variable too early

```
Theorem double_injective_FAILED : forall n m,
    double n = double m ->
    n = m.
Proof.
 intros n m. induction n as [| n' IHn'].
 - (* n = O *) simpl. intros eq. destruct m as [| m'] eqn:E.
   + (* m = O *) reflexivity.
   + (* m = S m' *) discriminate eq.
 - (* n = S n' *) intros eq. destruct m as [| m'] eqn:E.
   + (* m = O *) discriminate eq.
   + (* m = S m' *) apply f_equal.
Abort.
```

"if double n = double m then n = m" implies
"if double (S n) = double m then S n = m"

Induction on n when m is already in the context doesn't work because we are then trying to prove a statement involving every n but just a single m.

A Solution

```
Theorem double_injective : forall n m,
    double n = double m ->
    n = m.
Proof.
 intros n. induction n as [| n' IHn'].
 - (* n = O *) simpl. intros m eq. destruct m as [| m'] eqn:E.
   + (* m = O *) reflexivity.
   + (* m = S m' *) discriminate eq.
 - (* n = S n' *) simpl.
   intros m eq.
   destruct m as [| m'] eqn:E.
   + (* m = O *)
     discriminate eq.
   + (* m = S m' *)
     apply f_equal. reflexivity.
     apply IHn'. simpl in eq. injection eq as goal. apply goal.
Qed.
```

Generalize: Quantified Variable Rearrangement

```
Theorem double_injective_take2 : forall n m,
  double n = double m ->
  n = m.
Proof.
 intros n m.
 generalize dependent n.
 induction m as [| m' IHm'].
 - (* m = O *) simpl. intros n eq. destruct n as [| n'] eqn:E.
  + (* n = O *) reflexivity.
  + (* n = S n' *) discriminate eq.
 - (* m = S m' *) intros n eq. destruct n as [| n'] eqn:E.
  + (* n = O *) discriminate eq.
  + (* n = S n' *) apply f_equal.
   apply IHm'. injection eq as goal. apply goal. Qed.
```

```
1 subgoal
n, m : nat
______________________(1/1)
double n = double m ->
n = m
```

→

```
1 subgoal
m : nat
______________________(1/1)
forall n : nat,
double n = double m -> n =
m
```

Unfolding Definitions

Manual Unfolding

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

Lemma square_mult : forall n m, square (n * m) = square n * square m.
Proof.
  intros n m.
  simpl.
  unfold square.
  rewrite mult_assoc.
  assert (H : n * m * n = n * n * m).
    { rewrite mult_comm. apply mult_assoc. }
  rewrite H. rewrite mult_assoc. reflexivity.
Qed.
```

Conservative Automatic Unfolding

```
Definition foo (x: nat) := 5.

Fact silly_fact_1 : forall m, foo m + 1 = foo (m + 1) + 1.
Proof.
  intros m.
  simpl.
  reflexivity.
Qed.
```

Conservative Automatic Unfolding

```
Definition bar x :=
 match x with
 | O => 5
 | S _ => 5
 end.

Fact silly_fact_2_FAILED : forall m, bar m + 1 = bar (m + 1) + 1.
Proof.
 intros m.
 simpl. (* Does nothing! *)
Abort.

Fact silly_fact_2 : forall m, bar m + 1 = bar (m + 1) + 1.
Proof.
 intros m.
 unfold bar. (* can be omitted *)
 destruct m eqn:E.
 - simpl. reflexivity.
 - simpl. reflexivity.
Qed.
```

Using destruct on Compound Expressions

Case Analysis on "Results"

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

Theorem sillyfun_false : forall (n : nat),
  sillyfun n = false.
Proof.
  intros n. unfold sillyfun.
  destruct (n =? 3) eqn:E1.
    - (* n =? 3 = true *) reflexivity.
    - (* n =? 3 = false *) destruct (n =? 5) eqn:E2.
      + (* n =? 5 = true *) reflexivity.
      + (* n =? 5 = false *) reflexivity.  Qed.
```

Using "Results”

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

Theorem sillyfun1_odd : forall (n : nat),
   sillyfun1 n = true ->
   oddb n = true.
Proof.
 intros n eq. unfold sillyfun1 in eq.
 destruct (n =? 3) eqn:Heqe3.
  - (* e3 = true *) apply eqb_true in Heqe3.
   rewrite -> Heqe3. reflexivity.
  - (* e3 = false *)
    destruct (n =? 5) eqn:Heqe5.
     + (* e5 = true *)
      apply eqb_true in Heqe5.
      rewrite -> Heqe5. reflexivity.
     + (* e5 = false *) discriminate eq.  Qed.
```

Tactics Review

List of Tactics

- intros: move hypotheses/variables from goal to context
- reflexivity: finish the proof (when the goal looks like e = e)
- apply: prove goal using a hypothesis, lemma, or constructor
- apply… in H: apply a hypothesis, lemma, or constructor to a hypothesis in the context (forward reasoning)
- apply… with…: explicitly specify values for variables that cannot be determined by pattern matching
- simpl: simplify computations in the goal
- simpl in H: … or a hypothesis

- rewrite: use an equality hypothesis (or lemma) to rewrite the goal
- rewrite … in H: … or a hypothesis
- symmetry: changes a goal of the form t=u into u=t
- symmetry in H: changes a hypothesis of the form t=u into u=t
- transitivity y: prove a goal x=z by proving two new subgoals, x=y and y=z
- unfold: replace a de!ned constant by its right-hand side in the goal
- unfold… in H: … or a hypothesis

- destruct… as…: case analysis on values of inductively defined types
- destruct… eqn:…: specify the name of an equation to be added to the context, recording the result of the case analysis induction… as…: induction on values of inductively defined types
- injection: reason by injectivity on equalities between values of inductively defined types
- discriminate: reason by disjointness of constructors on equalities between values of inductively de!ned types

- assert (H: e) (or assert (e) as H): introduce a "local lemma" e and call it H
- generalize dependent x: move the variable x (and anything else that depends on it) from the context back to an explicit hypothesis in the goal formula
- f_equal: change a goal of the form f x = f y into x = y

作业

- 完成Tactics.v中的至少10个练习题。