



软件理论基础与实践

# Tactics: More Tactics

熊英飞  
北京大学



# apply策略

- apply策略直接应用假设完成结论推导

```
Theorem silly1 : forall (n m : nat),  
  n = m ->  
  n = m.
```

```
Proof. intros n m eq.
```

```
(** [Coq Proof View]
```

```
* 1 subgoal
```

```
*
```

```
*   n, m : nat
```

```
*   eq : n = m
```

```
*   =====
```

```
*   n = m
```

```
*)
```

```
apply eq. (** No more subgoals. *)
```

```
Qed.
```



# apply策略

- apply也可应用P->Q形式的假设把结论从Q变成P

```
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.
```

```
(** [Coq Proof View]  
* 1 subgoal  
*  
*   n, m, o, p : nat  
*   eq1 : n = m  
*   eq2 : n = m -> [n; o] = [m; p]  
*   =====  
*   [n; o] = [m; p]  
*)
```



# apply策略

- apply也可应用P->Q形式的假设把结论从Q变成P

```
apply eq2. (** [Coq Proof View]
* 1 subgoal
*
*   n, m, o, p : nat
*   eq1 : n = m
*   eq2 : n = m -> [n; o] = [m; p]
*   =====
*   n = m
*)
apply eq1. (** No more subgoals. *)
Qed.
```



# apply策略

- apply策略会自动替换全称量词

```
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.
```

```
(** [Coq Proof View]  
* 1 subgoal  
*  
*   n, m : nat  
*   eq1 : (n, n) = (m, m)  
*   eq2 : forall q r : nat, (q, q) = (r, r) -> [q] = [r]  
*   =====  
*   [n] = [m]  
*)
```



# apply策略

- apply策略会自动替换全称量词

```
apply eq2.(** [Coq Proof View]
* 1 subgoal
*
*   n, m : nat
*   eq1 : (n, n) = (m, m)
*   eq2 : forall q r : nat, (q, q) = (r, r) -> [q] = [r]
*   =====
*   (n, n) = (m, m)
*)
apply eq1.(** No more subgoals. *)
Qed.
```



# apply策略

- apply策略应用时假设和结论必须能完全匹配

```
Theorem silly3 : forall (n m : nat),  
  n = m ->  
  m = n.
```

```
Proof. intros n m H.
```

```
(**  H : n = m  
*   =====  
*   m = n  
*)
```

```
Fail apply H.
```

```
(** [Coq Proof View]  
* The command has indeed failed with message:  
* In environment  
* n, m : nat  
* H : n = m  
* Unable to unify "n = m" with "m = n".  
*)
```



# symmetry策略

- symmetry策略用于交换目标等式的左右两边

```
symmetry.  
(* [Coq Proof View]  
* 1 subgoal  
*  
*   n, m : nat  
*   H : n = m  
*   =====  
*   n = m  
*)  
apply H. (** No more subgoals. *)  
Qed.
```





# apply with策略

- 如果定理前提中有自由变量，apply策略会失败

```
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.
```

```
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.
```

```
Fail apply trans_eq.
```

```
(* Unable to find an instance for the variable m. *)
```



# apply with策略

- apply with指定自由变量的值

```
  apply trans_eq with (m:=[c;d]).
(** [Coq Proof View]
 * 2 subgoals
 *
 *   a, b, c, d, e, f : nat
 *   eq1 : [a; b] = [c; d]
 *   eq2 : [c; d] = [e; f]
 *   =====
 *   [a; b] = [c; d]
 *
 * subgoal 2 is:
 * [c; d] = [e; f]
 *)
  apply eq1. apply eq2.    Qed.
```



# transitivity策略

- transitivity x等价于apply trans\_eq with (m:=x)

```
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.  
transitivity [c;d].  
apply eq1. apply eq2.   Qed.
```



# 归纳类型定义的特点

- Injection: 同一个构造函数传不同参数时构造的值不同,

- 即构造函数为单射
- 即  $S\ n = S\ m \rightarrow n = m$

```
Inductive nat : Type :=  
  | 0  
  | S (n : nat).
```

- Disjointness: 不同的构造函数构造的值均不同,
  - 即  $S\ n = 0$  不可能成立
- 利用这些特点可以完成一些证明, Coq也提供了相应的策略支持



# 证明单射

- 单射可以通过定义函数来返回构造函数实参证明

```
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. }
```

```
  (* n, m : nat
```

```
  *   H1 : S n = S m
```

```
  *   H2 : n = Nat.pred (S n)
```

```
  *   =====
```

```
  *   n = m
```

```
  *)
```

```
rewrite H2.
```

```
(* Nat.pred (S n) = m *)
```

```
rewrite H1.
```

```
(* Nat.pred (S m) = m *)
```

```
reflexivity. Qed.
```

```
Definition pred (n : nat) : nat :=  
  match n with  
  | 0 => 0  
  | S n' => n'  
  end.
```



# injection策略

- injection策略根据构造函数的单射性推导参数的等价性

```
Theorem S_injective' : forall (n m : nat),  
  S n = S m ->  
  n = m.
```

```
Proof. intros n m H.  
(* [Coq Proof View]  
* 1 subgoal  
*  
*   n, m : nat  
*   H : S n = S m  
*   =====  
*   n = m  
*)
```



# injection策略

- injection策略根据构造函数的单射性推导参数的等价性

```
injection H as Hnm.  
(* [Coq Proof View]  
* 1 subgoal  
*  
*   n, m : nat  
*   Hnm : n = m  
*   =====  
*   n = m  
*)  
  apply Hnm.  
Qed.
```



# injection策略

- as部分可以省略，省略后推出的等式加入目标

```
injection H.  
(** [Coq Proof View]  
* 1 subgoal  
*  
*   n, m : nat  
*   H: S n = S m  
*   =====  
*   n = m -> n = m  
*)  
  intros Hnm. apply Hnm.  
Qed.
```





# injection策略

- 也可以递归推出多个等式

```
Theorem injection_ex1 : forall (n m o : nat),  
  [n;m] = [o;o] ->  
  n = m.
```

Proof.

```
  intros n m o H.  
  (** [Coq Proof View]  
  * 1 subgoal  
  *  
  *   n, m, o : nat  
  *   H : [n; m] = [o; o]  
  *   =====  
  *   n = m  
  *)
```

注意[n;m]等价于  
cons n (cons m nil)



# injection策略

- 也可以递归推出多个等式

```
injection H as H1 H2.  
(** [Coq Proof View]  
* 1 subgoal  
*  
*   n, m, o : nat  
*   H1 : n = o  
*   H2 : m = o  
*   =====  
*   n = m  
*)  
rewrite H1. rewrite H2. reflexivity.  
Qed.
```



# injection的逆: f\_equal

对任意函数都成立

**Theorem** f\_equal : forall (A B : Type) (f: A -> B) (x y: A),  
x = y -> f x = f y.

**Proof.** intros A B f x y eq. rewrite eq. 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. 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.

注意injection应用到假设上，  
f\_equal应用到目标上



# discriminate策略

- 如果假设包含不同构造函数构造的值形成了等式，则直接判断结论成立
  - 爆炸原理：False推导出任意结论

```
Theorem discriminate_ex1 : forall (n m : nat),  
  false = true ->  
  n = m.
```

```
Proof. intros n m contra.
```

```
(* 1 subgoal
```

```
*
```

```
*   n, m : nat
```

```
*   contra : false = true
```

```
*   =====
```

```
*   n = m
```

```
*)
```

```
discriminate contra. (** No more subgoals. *)
```

```
Qed.
```

参数可省略，discriminate  
会自动寻找矛盾的假设



# discriminate策略

- discriminate会自动应用simpl，并递归到深层构造函数

```
Theorem discriminate_ex2 : forall (n : nat),
```

```
  S n = 0 ->
```

```
  2 + 2 = 5.
```

```
Proof.
```

```
  intros n contra.
```

```
(** [Coq Proof View]
```

```
 * 1 subgoal
```

```
 *
```

```
 *   n : nat
```

```
 *   contra : S n = 0
```

```
 *   =====
```

```
 *   2 + 2 = 5
```

```
 *)
```

```
  discriminate. Qed.
```



# 将策略应用到假设

- 在适用的策略后面加上“in H”能将策略应用到假设H

```
Theorem S_inj : forall (n m : nat) (b : bool),  
  ((S n) =? (S m)) = b ->  
  (n =? m) = b.
```

```
Proof. intros n m b H.
```

```
(** [Coq Proof View]  
* 1 subgoal  
*  
*   n, m : nat  
*   b : bool  
*   H : (S n =? S m) = b  
*   =====  
*   (n =? m) = b  
*)
```



# 将策略应用到假设

- 在适用的策略后面加上“in H”能将策略应用到假设H

```
simpl in H.(** [Coq Proof View]
* 1 subgoal
*
*   n, m : nat
*   b : bool
*   H : (n =? m) = b
*   =====
*   (n =? m) = b
*)
apply H.(** No more subgoals. *)
Qed.
```



# 将策略应用到假设

- 等价变换策略应用到目标和假设上效果相同

```
Theorem silly4 : forall (n m p q : nat),  
  (n = m -> p = q) ->  
  m = n ->  
  q = p.
```

```
Proof. intros n m p q EQ H.  
(* [Coq Proof View]  
* 1 subgoal  
*  
*   n, m, p, q : nat  
*   EQ : n = m -> p = q  
*   H : m = n  
*   =====  
*   q = p  
*)
```





# 将策略应用到假设

- 等价变换策略应用到目标和假设上效果相同

```
symmetry in H. (** [Coq Proof View]
* 1 subgoal
*
*   n, m, p, q : nat
*   EQ : n = m -> p = q
*   H : n = m
*   =====
*   q = p
*)
```



# 将策略应用到假设

- 不等价变换应用时方向相反
  - 给定 $H:P \rightarrow Q$ ，`apply H`将目标从 $Q$ 替换为 $P$ ，`apply H in H1`将 $H1$ 从 $P$ 替换到 $Q$

```
apply EQ in H. (** [Coq Proof View]
* 1 subgoal
*
*   n, m, p, q : nat
*   EQ : n = m -> p = q
*   H : p = q
*   =====
*   q = p
*)
symmetry in H. apply H. Qed.
```



# 一个失败的归纳证明过程

**Theorem** `double_injective_FAILED` : `forall` `n m`,  
 `double n = double m ->`  
 `n = m.`

**Proof.**

```
intros n m. induction n as [| n' IHn'].  
- (* n = 0 *) simpl. intros eq. destruct m as [| m'] eqn:E.  
  + (* m = 0 *) reflexivity.  
  + (* m = S m' *) discriminate eq.  
- (* n = S n' *) intros eq. destruct m as [| m'] eqn:E.  
  + (* m = 0 *) discriminate eq.  
  + (* m = S m' *)
```



# 一个失败的归纳证明过程

```
(** [Coq Proof View]
* 1 subgoal
*
*   n', m, m' : nat
*   E : m = S m'
*   IHn' : double n' = double (S m') -> n' = S m'
*   eq : double (S n') = double (S m')
*   =====
*   S n' = S m'
*)
Abort.
```

前提不为真，自然可以有任意结论，该归纳假设完全无用



# 为什么失败

- 自然数上归纳证明 $P$ 的过程
  - 证明 $P(0)$
  - 证明 $\forall n, P(n) \rightarrow P(S n)$
- 这个例子中,  $P(n) \equiv \forall m, P'(n, m)$ 
  - 其中 $P'(n, m) \equiv \text{double } n = \text{double } m \rightarrow n = m$
- 即, 我们需要证明
  - $\forall m, P'(0, m)$
  - $\forall n, (\forall m, P'(n, m)) \rightarrow (\forall m, P'(S n, m))$
- 但实际我们证明的是
  - $\forall m, P'(0, m)$
  - $\forall n, \forall m, (P'(n, m) \rightarrow P'(S n, m)) \equiv$   
 $\forall n, \forall m, ((\text{double } n = \text{double } m \rightarrow n = m)$   
 $\rightarrow (\text{double } S n = \text{double } m \rightarrow S n = m))$
- Coq规则: 已经在假设区的变量不作为自由变量放入归纳假设



# 解决方案1

- 不主动intro额外的变量

intro n可以省略，  
induction n自动引入n  
和n之前的变量

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

```
Proof. intro n. induction n as [| n' IHn'].
```

- (\* n = 0 \*) simpl. intros m eq. destruct m as [| m'] eqn:E.
  - + (\* m = 0 \*) reflexivity.
  - + (\* m = S m' \*) discriminate eq.
- (\* n = S n' \*)

```
(** [Coq Proof View]
```

```
* 1 subgoal
```

```
*
```

```
* n' : nat
```

```
* IHn' : forall m : nat, double n' = double m -> n' = m
```

```
*
```

```
* =====  
* forall m : nat, double (S n') = double m -> S n' = m
```

```
* *)
```



# 解决方案1

- 该方法在归纳变量不在第一位时会出问题

**Theorem** `double_injective_take2_FAILED2` : `forall` n m,  
double n = double m -> n = m.

**Proof.** `induction` m.

- `(* m = 0 *) simpl. intros. destruct n as [| n'] eqn:E.`
  - + `(* n = 0 *) reflexivity.`
  - + `(* n = S n' *) discriminate H.`
- `(* n = S n' *)`

`(** [Coq Proof View]`

`* 1 subgoal`

`*`

`* n, m: nat`

`* IHm: double n = double m -> n = m`

`*`

`* =====`  
`* double n = double (S m) -> n = S m`

`* )`



# 解决方案2

- 采用generalize dependent策略

**Theorem** `double_injective_take2` : forall n m,  
double n = double m -> n = m.

**Proof.**

```
intros n m.  
(*   n, m : nat  
 *   =====  
 *   double n = double m -> n = m  
 *)  
  generalize dependent n.  
(*   m : nat  
 *   =====  
 *   forall n : nat, double n = double m -> n = m  
 *)
```





# 解决方案2

- 采用generalize dependent策略

```
induction m as [| m' IHm'].
- (* m = 0 *) simpl. intros n eq. destruct n as [| n'] eqn:E.
  + (* n = 0 *) reflexivity.
  + (* n = S n' *) discriminate eq.
- (* m = S m' *)
(** [Coq Proof View]
* 1 subgoal
*
*   m' : nat
*   IHm' : forall n : nat, double n = double m' -> n = m'
*   =====
*   forall n : nat, double n = double (S m') -> n = S m'
*)
```



# Unfold策略——动机

Definition square n := n \* n.

Lemma square\_mult : forall n m,  
square (n \* m) = square n \* square m.

Proof.

intros n m.

```
(*  n, m : nat
 *  =====
 *  square (n * m) = square n * square m
 *)
```

simpl.

```
(*  n, m : nat
 *  =====
 *  square (n * m) = square n * square m
 *)
```

为什么square没有被展开?  
Coq只在能展开match  
或者展开fixpoint的时候  
进行约简, 否则不变。



# Unfold策略

```
unfold square.
(** [Coq Proof View]
 * 1 subgoal
 *
 *   n, m : nat
 *   =====
 *   n * m * (n * m) = n * n * (m * m)
 *)
rewrite mult_assoc.
assert (H : n * m * n = n * n * m).
  { rewrite mul_comm. apply mult_assoc. }
rewrite H. rewrite mult_assoc. reflexivity.
Qed.
```

将目标中的square展开。  
也可以加上in H用于假设H。



# 更多simpl的例子

Definition foo (x: nat) := 5.

Fact silly\_fact\_1 : forall m, foo m + 1 = foo (m + 1) + 1.

Proof.

intros m.

```
(* m : nat
 * =====
 * foo m + 1 = foo (m + 1) + 1
 *)
```

simpl.

```
(* m : nat
 * =====
 * 6 = 6
 *)
```

结果是什么?



# 更多simpl的例子

```
Definition bar x :=  
  match x with  
  | 0 => 5  
  | S _ => 5  
  end.
```

Fact silly\_fact\_2\_FAILED : forall m, bar m + 1 = bar (m + 1) + 1.

Proof.

```
  intros m.  
  (*   m : nat  
   *   =====  
   *   bar m + 1 = bar (m + 1) + 1  
   *)  
  simpl.  
  (*   m : nat  
   *   =====  
   *   bar m + 1 = bar (m + 1) + 1  
   *)
```

结果是什么？



# 采用destruct分解表达式

```
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.
```

```
(** [Coq Proof View]  
* 1 subgoal  
*  
*   n : nat  
*   =====  
*   (if n =? 3 then false else if n =? 5 then false else false)  
*   = false  
*)
```

如何证明?



# 采用destruct分解表达式

- 虽然可以用destruct n证明，但过于麻烦

```
Theorem sillyfun_false : forall (n : nat),  
  sillyfun n = false.
```

```
Proof. intros n. unfold sillyfun.
```

```
destruct n. reflexivity.
```

```
destruct n. reflexivity.
```

```
destruct n. reflexivity.
```

```
destruct n. reflexivity.
```

```
destruct n. reflexivity.
```

```
destruct n. reflexivity.
```

```
reflexivity.
```

```
Qed.
```



# 采用destruct分解表达式

```
destruct (n =? 3) eqn:E1.
(** [Coq Proof View]
 * 2 subgoals
 *
 *   n : nat
 *   E1 : (n =? 3) = true
 *   =====
 *   false = false
 *
 * subgoal 2 is:
 * (if n =? 5 then false else false) = false
 *)
- (* n =? 3 = true *) reflexivity.
- (* n =? 3 = false *) destruct (n =? 5) eqn:E2.
  + (* n =? 5 = true *) reflexivity.
  + (* n =? 5 = false *) reflexivity. Qed.
```





# 分解表达式时eqn:H往往关键

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

```
Theorem sillyfun1_odd_FAILED : forall (n : nat),  
  sillyfun1 n = true -> odd n = true.
```

Proof.

```
  intros n eq. unfold sillyfun1 in eq.  
  destruct (n =? 3).
```

```
(*   n : nat  
*   eq : true = true  
*   =====  
*   odd n = true  
*  
* subgoal 2 is:  
*   odd n = true  
*)
```

Abort.

# 分解表达式时eqn:H往往关键



```
Theorem sillyfun1_odd : forall (n : nat),
  sillyfun1 n = true ->
  odd 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.
```



# 策略总结

- `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



# 策略总结

- `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 defined constant by its right-hand side in the goal
- `unfold ... in H: ...` or a hypothesis



# 策略总结

- `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 ... as ...`: reason by injectivity on equalities between values of inductively defined types



# 策略总结

- `discriminate`: reason by disjointness of constructors on equalities between values of inductively defined 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standard非optional且不属于Additional Exercises的8道习题
  - 请使用最新英文版教材