软件理论基础与实践

IndPrinciples: Induction Principles

熊英飞
北京大学

结构归纳法定理 Induction Principle

- Coq对每个递归定义的数据类型生成结构归纳法定理

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

```
Check nat_ind :
  forall P : nat -> Prop,
    P 0 ->
    (forall n : nat, P n -> P (S n)) ->
    forall n : nat, P n.
```

结构归纳法定理的证明

- 直接按归纳定义展开，递归应用传入的证明即可

```
Print nat_ind.
(*
 * nat_ind =
 * fun (P : nat -> Prop) (f : P 0)
 *     (f0 : forall n : nat, P n -> P (S n)) =>
 * fix F (n : nat) : P n :=
 *   match n as n0 return (P n0) with
 *   | 0 => f
 *   | S n0 => f0 n0 (F n0)
 * end
 *)
```

更多结构归纳法定理

```
Inductive time : Type :=
  | day
  | night.
Check time_ind :
  forall P : time -> Prop,
    P day ->
    P night ->
    forall t : time, P t.
```

```
Inductive tree (X:Type) : Type :=
  | leaf (x : X)
  | node (t1 t2 : tree X).
Check tree_ind :
  forall (X : Type) (P : tree X -> Prop),
  (forall x : X, P (leaf X x)) ->
  (forall t1 : tree X,
    P t1 -> forall t2 : tree X, P t2 -> P (node X t1 t2)) ->
  forall t : tree X, P t.
```

参数的位置影响结构归纳法定理

```
Inductive tree' : Type -> Type :=
  | leaf' (X: Type) (x : X): tree' X
  | node' (X: Type) (t1 t2 : tree' X) : tree' X.


Check tree'_ind :
   forall P : forall T : Type, tree' T -> Prop,
   (forall (X : Type) (x : X), P X (leaf' X x)) ->
   (forall (X : Type) (t1 : tree' X),
   P X t1 -> forall t2 : tree' X, P X t2 -> P X (node' X t1 t2)) ->
   forall (T : Type) (t : tree' T), P T t.
```

induction

```
Theorem plus_n_O : forall n:nat, n = n + 0.
Proof.
  intros n. induction n as [| n' IHn']. Show Proof.
  (* (fun n : nat =>
      nat_ind (fun n0 : nat => n0 = n0 + 0) ?Goal
        (fun (n' : nat) (IHn' : n' = n' + 0) => ?Goal0) n) *)
  Print nat_ind.
  (* nat_ind : forall P : nat -> Prop,
     P 0 -> (forall n : nat, P n -> P (S n)) -> forall n : nat, P n *)
  - (* n = 0 *)     reflexivity.
  - (* n = S n' *) simpl. rewrite <- IHn'. reflexivity.  Qed.
```

induction：应用结构归纳法定理

命题对应的结构归纳法定理

根据一般归纳类型的结构归纳法定理猜测：

```
Inductive ev : nat -> Prop :=
| ev_0 : ev 0
| ev_SS (n : nat) (H : ev n) : ev (S (S n)).
```

```
ev_ind :
  forall P : forall (n:nat), ev n -> Prop,
    P 0 ev_0 ->
    (forall (n:nat) (E:ev n), P n E ->
     P (S (S n)) (ev_SS n E))->
    forall (n':nat) (E':ev n) -> P n' E'.
```

P的第二个参数（即ev n的证明）没有用，因为Coq不允许从证明构建命题，实践中也不会有命题的形式（而非证明）依赖于另外一个命题的证明。

尝试从证明构造命题

```
Fail Definition strange (H:forall n, ev n) (m:nat) :=
  match H m with
  | ev_0 => forall n m, n = m
  | ev_SS _ _ => forall n m, n <> m
  end.
```

```
The command has indeed failed with message:
Incorrect elimination of "H m" in the inductive type "ev":
the return type has sort "Type" while it should be "SProp" or "Prop".
Elimination of an inductive object of sort Prop
is not allowed on a predicate in sort Type
because proofs can be eliminated only to build proofs.
```

命题对应的结构归纳法定理

```
Check ev_ind :
  forall P : nat -> Prop,
    P 0 ->
    (forall n : nat, ev n -> P n -> P (S (S n))) ->
    forall n : nat, ev n -> P n.
```

去掉命题本身的证明作为参数
P作用在ev的参数上，ev n作为前提出现

关系对应的结构归纳法定理

```
Inductive le : nat -> nat -> Prop :=
  | le_n (n : nat)                    : le n n
  | le_S (n m : nat) (H : le n m) : le n (S m).
```

```
Check le_ind : forall P : nat -> nat -> Prop,
  (forall n : nat, P n n) ->
  (forall n m : nat, le n m -> P n m -> P n (S m)) ->
   forall n n0 : nat, le n n0 -> P n n0.
```

课本上定义的le_ind，P作用在le的所有参数上

关系对应的结构归纳法定理

```
Print le.
(* ====> Inductive le (n : nat) : nat -> Prop :=
            le_n : n <= n
          | le_S : forall m : nat, n <= m -> n <= S m *)
```

```
Check le_ind:
  forall (n : nat) (P : nat -> Prop),
    P n ->
    (forall m : nat, n <= m -> P m -> P (S m)) ->
    forall n0 : nat, n <= n0 -> P n0.
```

标准库中的le_ind，P只作用在index上，归纳形式更简单

eq_ind定理

```
Inductive eq (A : Type) (x : A) : A -> Prop
  :=  eq_refl : x = x
```

```
Check eq_ind :
  forall (A: Type) (x:A) (P : A -> Prop),
    P x -> forall y : A, x=y -> P y.
```

逻辑非:discriminate

```
Theorem zero_not_one : 1 <> 0.
Proof.
  intros contra.
  discriminate contra.
Qed.
```

```
Definition zero_not_one' : 0 <> 1 :=
  fun contra : 0 = 1 => eq_ind 0
    (fun e:nat => match e with
                  | 0 => True
                  | S _ => False
                  end) I 1 contra.
```

```
eq_ind : forall (A : Type) (x : A) (P : A -> Prop),
         P x -> forall y : A, x = y -> P y
```

discriminate：基于两个不同Constructor相等的证明构造False的证明

rewrite

```
Theorem plus_id_example : forall n m:nat,
  n=m -> n+n=m+m.
Proof.
  intros n m H.
  rewrite -> H.
  reflexivity.
Qed.
```

```
Definition plus_id_example' : forall n m:nat,
  n=m -> n+n=m+m :=
  fun (n m : nat) (H : n=m) =>
  eq_ind n (fun (m:nat) => n+n=m+m) eq_refl m H.
```

rewrite：利用eq_ind实现相等内容的替换

作业

- 完成IndPrinciples中standard非optional的3道习题
  - 请使用最新英文版教材