软件理论基础与实践

ProofObjects: The Curry-Howard Correspondence
IndPrinciples: Induction Principles

胡振江 熊英飞
北京大学

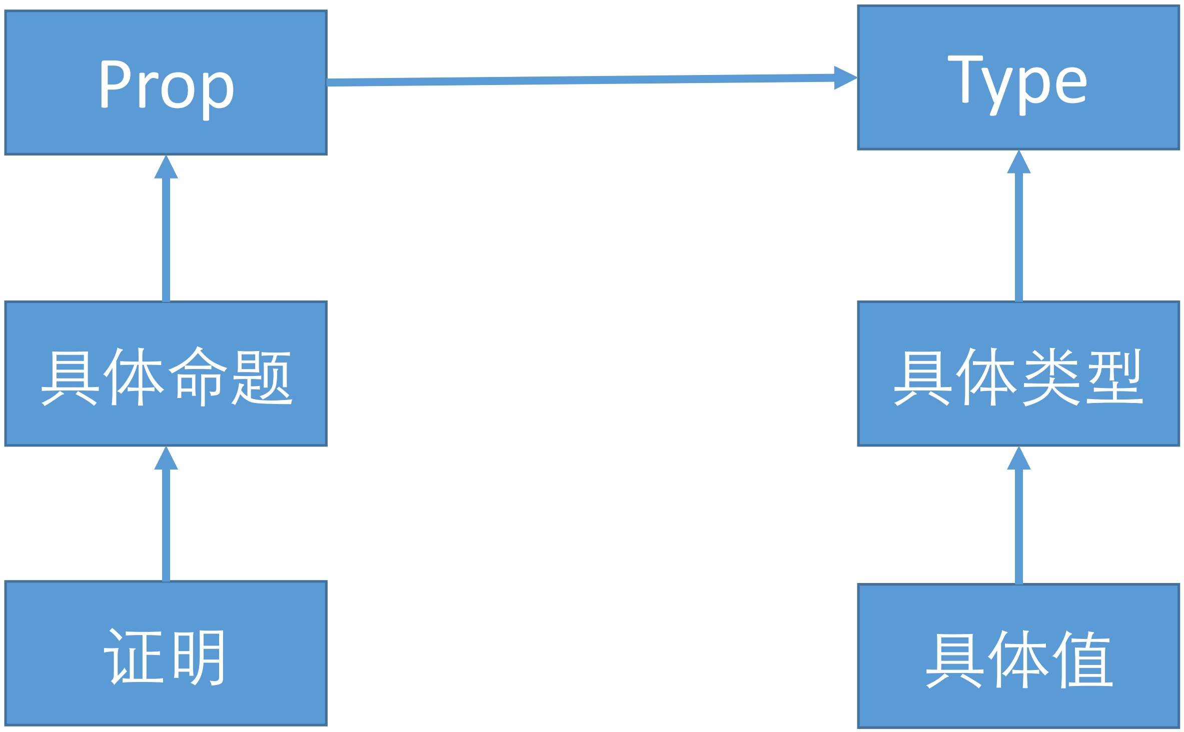
Curry-Howard Correspondence

- 由Haskell Brooks Curry和William Alvin Howard在1934-1969年间三个主要发现构成
- 命题 ⇔ 类型
  - A->B ⇔ A->B
- 证明 ⇔ 值

Haskell Brooks Curry

Logic side	Programming side
universal quantification	generalised product type (Π type)
existential quantification	generalised sum type (Σ type)
implication	function type
conjunction	product type
disjunction	sum type
true formula	unit type
false formula	bottom type
Hilbert-style deduction system	type system for combinatory logic
natural deduction	type system for lambda calculus

Curry-Howard Correspondence in Coq



箭头表示“S的类型是T”

Coq中的类型定义

```
Inductive bool : Type :=
   | true
   | false.
```

Coq中的命题定义

```
Inductive bool : Type :=
  | true
  | false.
```

```
Inductive True : Prop :=
  | I : True.
```

定义类型的值

```
Inductive bool : Type :=
  | true
  | false.
```

```
Coq < Compute true.
     = true
     : bool
```

定义命题的证明

```
Inductive bool : Type :=
  | true
  | false.
```

```
Coq < Compute true.
     = true
     : bool
```

```
Inductive True : Prop :=
  | I : True.
```

```
Coq < Compute I.
     = I
     : True
```

Tactic: 生成证明的命令

```
Lemma True_is_true : True.
Proof.
   apply I.
   Show Proof.
   (* I *)
Qed.
```

```
Print True_is_true.
(* True_is_true = I : True *)
```

```
Definition True_is_true := I.
```

定理=命题+证明

Tactic: 生成程序的命令

```
Definition FalseTerm : bool.
apply false.
Defined.
```

```
Print FalseTerm.
(* FalseTerm = false : bool *)
```

```
Definition FalseTerm := false.
```

带参数的归纳类型定义

```
Inductive list (X:Type) : Type :=
  | nil
  | cons (x : X) (l : list X).
```

首行冒号左边的参数叫做parameter，用于所有的constructor

```
Inductive nnlist : bool -> Type :=
| nnnil : nnlist false
| nncons {b:bool} (x : nat) (l : nnlist b) : nnlist true.

Definition fst(l:nnlist true) :=
  match l with
  | nncons x l => x
  end.
Fail Compute (fst nnnil).
(* The term "nnnil" has type "nnlist false" while it is
expected to have type "nnlist true". *)
```

表示列表非空

首行冒号右边的参数叫做index或者annotation，可由具体的constructor填充

带参数的归纳命题定义

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

```
Theorem ev_4 : ev 4.
Proof.
  apply ev_SS. Show Proof.
  (* (ev_SS 2 ?Goal) *)
  apply ev_SS. Show Proof.
  (* (ev_SS 2 (ev_SS 0 ?Goal)) *)
  apply ev_0. Show Proof.
  (* (ev_SS 2 (ev_SS 0 ev_0)) *)
Qed.
```

```
Definition ev_4 := ev_SS 2 (ev_SS 0 ev_0).
```

apply: 生成对应函数调用，将输入参数标记为Goal

回顾常见逻辑运算符和相关策略

量词、蕴含和函数

```
Theorem ev_plus4 : forall n, ev n -> ev (4 + n).
Proof.
  intros n H.
  (* (fun (n : nat) (H : ev n) => ?Goal) *)
  apply ev_SS.
  apply ev_SS.
  apply H.
Qed.
```

```
Definition ev_plus4' : forall n, ev n -> ev (4 + n) :=
  fun (n : nat) => fun (H : ev n) =>
    ev_SS (S (S n)) (ev_SS n H).
```

intros: 生成函数声明

等价和自反性

```
Inductive eq {X:Type} : X -> X -> Prop :=
| eq_refl : forall x, eq x x.

Notation "x = y" := (eq x y)
                      (at level 70, no associativity)
                    : type_scope.
```

该版本和标准库版本略有区别以帮助理解

```
Lemma four: 2 + 2 = 1 + 3.
Proof.
  reflexivity.
Qed.
```

```
Definition four' : 2 + 2 = 1 + 3 :=
  eq_refl 4.
```

reflexivity: 等价于apply eq_refl

逻辑与：定义

```
Inductive and (P Q : Prop) : Prop :=
| conj : P -> Q -> and P Q.

Arguments conj [P] [Q].

Notation "P /\ Q" := (and P Q) : type_scope.
```

```
Inductive prod (X Y : Type) : Type :=
| pair (x : X) (y : Y).

Arguments pair {X} {Y} _ _.

Notation "( x , y )" := (pair x y).

Notation "X * Y" := (prod X Y) : type_scope.
```

逻辑与：split

```
Lemma and_intro' : forall A B : Prop, A -> B -> A /\ B.
Proof.
  intros A B HA HB. split. Show Proof.
  (* (fun (A B : Prop) (HA : A) (HB : B) => conj ?Goal ?Goal0) *)
  - apply HA.
  - apply HB.
  Show Proof.
  (* (fun (A B : Prop) (HA : A) (HB : B) => conj HA HB) *)
Qed.
```

split: 如果目标只有一个constructor，生成该constructor，并将参数类型定义为目标

逻辑与： split

```
Lemma and_intro' : forall A B : Prop, A -> B -> A /\ B.
Proof.
  intros A B HA HB. apply conj. Show Proof.
  (* (fun (A B : Prop) (HA : A) (HB : B) => conj ?Goal ?Goal0) *)
  - apply HA.
  - apply HB.
  Show Proof.
  (* (fun (A B : Prop) (HA : A) (HB : B) => conj HA HB) *)
Qed.
```

split等价于apply constructor

逻辑与： split

```
Lemma truth : True.
Proof.
  split.
Qed.
```

split也不一定会产生新的目标

逻辑与： destruct

```
Theorem proj1' : forall P Q,
    P /\ Q -> P.
Proof.
  intros P Q HPQ. Show Proof.
  (* (fun (P Q : Prop) (HPQ : P /\ Q) => ?Goal) *)
  destruct HPQ as [HP HQ]. Show Proof.
  (* (fun (P Q : Prop) (HPQ : P /\ Q) => match HPQ with
                                          | conj HP HQ => ?Goal
                                            end) *)
  apply HP.
Qed.
```

destruct: 根据参数的归纳定义生成match

逻辑与: destruct

destruct有可能形成多个分支

```
Definition somefun: nat->bool.
intros H.
destruct H.
- apply true.
- apply false.
Defined.
```

以上代码定义了什么?

```
Definition iszero :=
fun H : nat => match H with
               | 0 => true
               | S _ => false
               end.
```

False

```
Inductive False : Prop := .
```

没有Constructor所以永远构造不出False的证明

```
Definition false_implies_zero_eq_one : False -> 0 = 1.
Proof.
  intros.
  destruct H.
Qed.
```

```
Definition false_implies_zero_eq_one : False -> 0 = 1 :=
  fun contra => match contra with end.
```

没有分支的match表达式具有任意类型

逻辑或：定义

```
Inductive or (P Q : Prop) : Prop :=
| or_introl : P -> or P Q
| or_intror : Q -> or P Q.

Arguments or_introl [P] [Q].
Arguments or_intror [P] [Q].

Notation "P \/ Q" := (or P Q) : type_scope.
```

逻辑或：left, right

```
Theorem inj_l' : forall (P Q : Prop), P -> P \/ Q.
Proof.
  intros P Q HP. left. apply HP.
Qed.
```

```
Definition inj_l : forall (P Q : Prop), P -> P \/ Q :=
  fun P Q HP => or_introl HP.
```

left：等价于apply or_introl
right：等价于apply or_intror

存在量词：定义

```
Inductive ex {A : Type} (P : A -> Prop) : Prop :=
| ex_intro : forall x : A, P x -> ex P.

Notation "'exists' x , p" :=
  (ex (fun x => p))
    (at level 200, right associativity) : type_scope.
```

P: 给定一个值，构造一个命题（即存在量词的body）
P x: 对于某个具体x值的证明

存在量词：exists策略

```
Theorem some_nat_is_even : exists n, ev n.
Proof.
  exists 0.
  apply ev_0.
Qed.
```

```
Definition some_nat_is_even' : exists n, ev n :=
  ex_intro ev 0 ev_0.
```

exists：根据当前目标构造ex_intro

存在量词：exists策略

```
Theorem some_nat_is_even : exists n, ev n.
Proof.
  apply ex_intro with (x:=0).
  apply ev_0.
Qed.
```

```
Definition some_nat_is_even' : exists n, ev n :=
  ex_intro ev 0 ev_0.
```

exists n等价于apply ex_intro with (x:=n)

apply with：为apply应用过程中所需要的类型参数提供值

逻辑非:定义

复习

```
Definition not (P:Prop) := P -> False.

Notation "~ x" := (not x) : type_scope.
```

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

回顾其他常见策略

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

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：coq根据inductive定义自动生成和证明结构归纳法定理，induction应用该定理

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.
```

injection

```
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.
```

```
Definition S_injective'' : forall (n m : nat),
  S n = S m -> n = m :=
  fun (n m : nat) (H : S n = S m) =>
    f_equal (fun e : nat => match e with
      | 0 => n
      | S n0 => n0
    end) H.
```

injection：生成从复杂结构到简单结构的函数并调用f_equal

复习：f_equal

```
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.
```

作业

- 完成ProofObject，IndPrinciples中standard非optional的习题
  - 请使用最新英文版教材
  - 注意IndPrinciples中Induction Principles for Propositions的部分将在下次课介绍，该部分无习题