软件理论基础与实践 习题课

最后一次习题课

2022年6月9日（星期四）

Powerpoint is turing complete but coq isn't.

Programming with Coq

我们学了什么?

一套教材 Software Foundations 中的两本书

- LF（Logical Foundations）
- PLF（Programming Language Foundations）

Logical Foundations

- 函数式编程
- 逻辑基础
- Coq 编程

函数式编程：ADT

代数数据类型（Algebric data type）、*归纳类型*

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

- 构造器
- 模式匹配

函数式编程：多态

同一个类别/函数的定义代码对多种类型适用，给定不同的参数得到不同的具体类型。

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

- 参数化多态（面向对象中的泛型）
- `list` 可以看作是 `Type->Type` 的函数（依赖类型）
- 隐式类型参数

函数式编程：高阶函数

函数可以作为参数传递，也可以作为返回值

- 匿名函数 / λ表达式

```
fun x => x * x
```

- 柯里化

```
f: A -> B -> C
g: (A -> B) -> C
```

逻辑基础：形式系统

- 字符表（Alphabet）
- 文法（Grammar）
- 公理模式（Axiom Schemata）
- 推导规则（Inference Rules）

性质：

- 一致性（Consistency）
- 完备性（Completeness）

哥德尔不完备性定理

逻辑基础：数理逻辑基础

一阶谓词逻辑：

- 与、或、非、蕴含、双向蕴含
- 全称量词、存在量词

高阶逻辑：

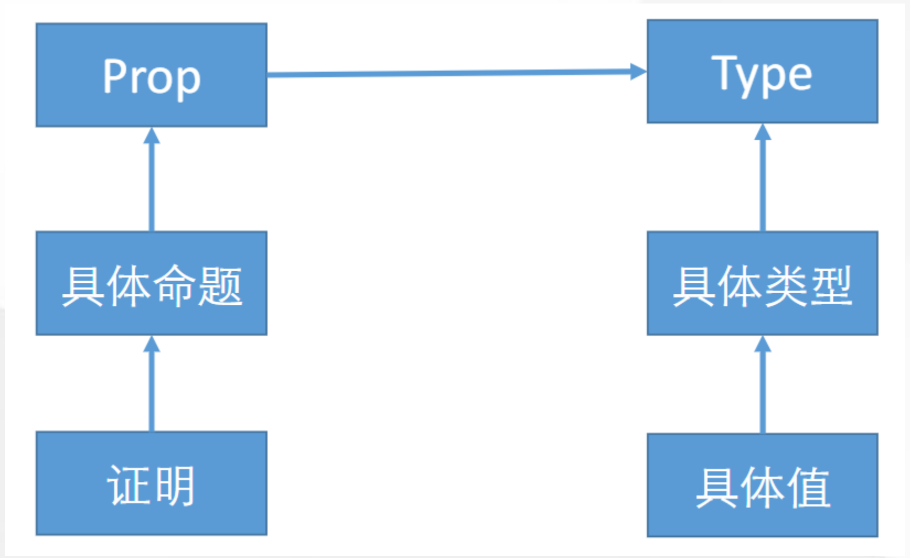
- 量词可以作用在谓词和函数上

直觉主义逻辑/构造逻辑：

- 没有排中律

Coq 编程：柯里-霍华德对应

柯里-霍华德同构、命题即类型



Coq 编程：命题

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

Inductive False : Prop := .
```

Coq 编程：命题

LANGUAGE WITH NULL AND UNDEFINED BEHAVIOR
NO UNDEFINED BEHAVIOR
NO NULL
NO FALSE

Coq 编程：命题

归纳命题:

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

Coq 编程：Tactics

等价关系

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

`reflexivity` 等价于 `apply eq_refl`。

Coq 编程：Tactics

逻辑与

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

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

Coq 编程：Tactics

逻辑与

```
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` 等价于 `apply conj`。

Coq 编程：Tactics

逻辑与

```
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`。

Coq 编程：Tactics

逻辑或

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

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

`left` 等价于 `apply or_introl`；
`right` 等价于 `apply or_intror`。

Coq 编程：Tactics

存在量词

```
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.
```

`exists n` 等价于 `apply ex_intro with (x:=n)`。

Coq 编程：Tactics

逻辑非

```
Definition not (P:Prop) := P -> False.
Notation "~ x" := (not x) : type_scope.
```

Coq 编程：Tactics

Solving simple goals:

- assumption
- reflexivity
- trival
- auto
- discriminate
- exact
- contradiction

Coq 编程：Tactics

Transforming goals:

- intros
- simpl
- unfold
- apply
- eapply
- rewrite

Coq 编程：Tactics

Transforming goals:

- inversion
- replace
- left/right
- exists
- exfalso

Coq 编程：Tactics

Transforming goals:

- revert
- generalize dependent
- remember
- clear
- subst

Coq 编程：Tactics

Breaking apart goals and hypotheses:

- split
- destruct
- induction

Coq Tactics Cheatsheet

Coq 编程：结构归纳法

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

Coq对每个递归定义的数据类型生成结构归纳法定理。

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

Programming Language Foundations

- 两种语言：IMP、Stlc
- 两种语义：操作语义（大步、小步）、公理语义
- 类型系统

IMP - 操作语义（大步）

$$\frac{}{\mathtt{st =[\ skip\ ]\Rightarrow\ st}}\quad\text{(E\_Skip)}$$

$$\frac{\mathtt{aeval\ st\ a = n}}{\mathtt{st =[\ x := a\ ]\Rightarrow\ (x\ !\!\!\rightarrow\ n\ ;\ st)}}\quad\text{(E\_Asgn)}$$

$$\frac{\begin{array}{c}\mathtt{st =[\ c_1\ ]\Rightarrow\ st'} \\ \mathtt{st' =[\ c_2\ ]\Rightarrow\ st''}\end{array}}{\mathtt{st =[\ c_1;c_2\ ]\Rightarrow\ st''}}\quad\text{(E\_Seq)}$$

IMP - 操作语义（大步）

$$\frac{\begin{array}{c}\texttt{beval st b = true} \\ \texttt{st =[} c_1 \texttt{]=> st'}\end{array}}{\texttt{st =[if b then } c_1 \texttt{ else } c_2 \texttt{ end]=> st'}} \quad \text{(E\_IfTrue)}$$

$$\frac{\begin{array}{c}\texttt{beval st b = false} \\ \texttt{st =[} c_2 \texttt{]=> st'}\end{array}}{\texttt{st =[if b then } c_1 \texttt{ else } c_2 \texttt{ end]=> st'}} \quad \text{(E\_IfFalse)}$$

$$\frac{\texttt{beval st b = false}}{\texttt{st =[while b do c end]=> st}} \quad \text{(E\_WhileFalse)}$$

$$\frac{\begin{array}{c}\texttt{beval st b = true} \\ \texttt{st =[c]=> st'} \\ \texttt{st' =[while b do c end]=> st''}\end{array}}{\texttt{st =[while b do c end]=> st''}} \quad \text{(E\_WhileTrue)}$$

IMP - 操作语义（小步）

AStep:

```
————————————————————————————————————————————————— (AS_Id)
             x / st --> ANum (st x)

                 a1 / st --> a1'
————————————————————————————————————————————————— (AS_Plus1)
          APlus a1 a2 / st --> APlus a1' a2

              aval v1    a2 / st --> a2'
————————————————————————————————————————————————— (AS_Plus2)
          APlus v1 a2 / st --> APlus v1 a2'

————————————————————————————————————————————————— (AS_Plus2)
APlus (ANum n1) (ANum n2) / st --> ANum (n1 + n2)

......
```

IMP - 操作语义（小步）

BStep:

```
                  a1 / st --> a1'
─────────────────────────────────────────────────── (BS_Eq1)
            BEq a1 a2 / st --> BEq a1' a2

               aval v1    a2 / st --> a2'
─────────────────────────────────────────────────── (BS_Eq2)
            BEq v1 a2 / st --> BEq v1 a2'

─────────────────────────────────────────────────── (BS_Eq)
         BEq (ANum n1) (ANum n2) / st --> ...

......
```

IMP - 操作语义（小步）

CStep:

```
                 a1 / st --> a1'
──────────────────────────────────────────────────── (CS_AssStep)
          i := a1 / st --> i := a1' / st

──────────────────────────────────────────────────── (CS_Ass)
    i := ANum n / st --> skip / (i !-> n ; st)

──────────────────────────────────────────────────── (CS_While)
           while b1 do c1 end / st -->
if b1 then c1; while b1 do c1 end else skip end / st

......
```

IMP - 操作语义（小步）

- 进展（Progress）
- 保持（Preservation）

IMP - 公理语义

$$\textsc{Skip}\ \frac{}{\{P\}\ \textbf{skip}\ \{P\}} \qquad \textsc{Assign}\ \frac{}{\{P[a/x]\}\ x := a\ \{P\}}$$

$$\textsc{Seq}\ \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}} \qquad \textsc{If}\ \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{Q\}}$$

$$\textsc{Consequence}\ \frac{\models (P \Rightarrow P') \qquad \{P'\}\ c\ \{Q'\} \qquad \models (Q' \Rightarrow Q)}{\{P\}\ c\ \{Q\}}$$

$$\textsc{While}\ \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{P \wedge \neg b\}}$$

IMP - 公理语义

循环不变式的条件:

- 足够弱：能被前条件推出
- 足够强：能推出后条件
- 能保持：每一次循环都保持条件

霍尔逻辑的性质:

- 正确性（Soundness）
- 完备性（Completeness）

IMP - 类型系统

```
Inductive tm : Type :=
  | tru : tm
  | fls : tm
  | test : tm -> tm -> tm -> tm
  | zro : tm
  | scc : tm -> tm
  | prd : tm -> tm
  | iszro : tm -> tm.
```

IMP - 类型系统

```
Inductive has_type : tm -> ty -> Prop :=
| T_Tru : |- tru \in Bool
| T_Fls : |- fls \in Bool
| T_Test : forall t1 t2 t3 T,
    |- t1 \in Bool ->
    |- t2 \in T ->
    |- t3 \in T ->
    |- test t1 t2 t3 \in T
| T_Zro : |- zro \in Nat
| T_Scc : forall t1,
    |- t1 \in Nat ->
    |- scc t1 \in Nat
| T_Prd : forall t1,
    |- t1 \in Nat ->
    |- prd t1 \in Nat
| T_Iszro : forall t1,
    |- t1 \in Nat ->
    |- iszro t1 \in Bool
```

Stlc - 操作语义

$$\frac{\texttt{value } v_2}{(\backslash x{:}T_2, t_1)\ v_2 \rightarrow [x{:=}v_2]t_1} \quad \text{(ST\_AppAbs)}$$

$$\frac{t_1 \rightarrow t_1'}{t_1\ t_2 \rightarrow t_1'\ t_2} \quad \text{(ST\_App1)}$$

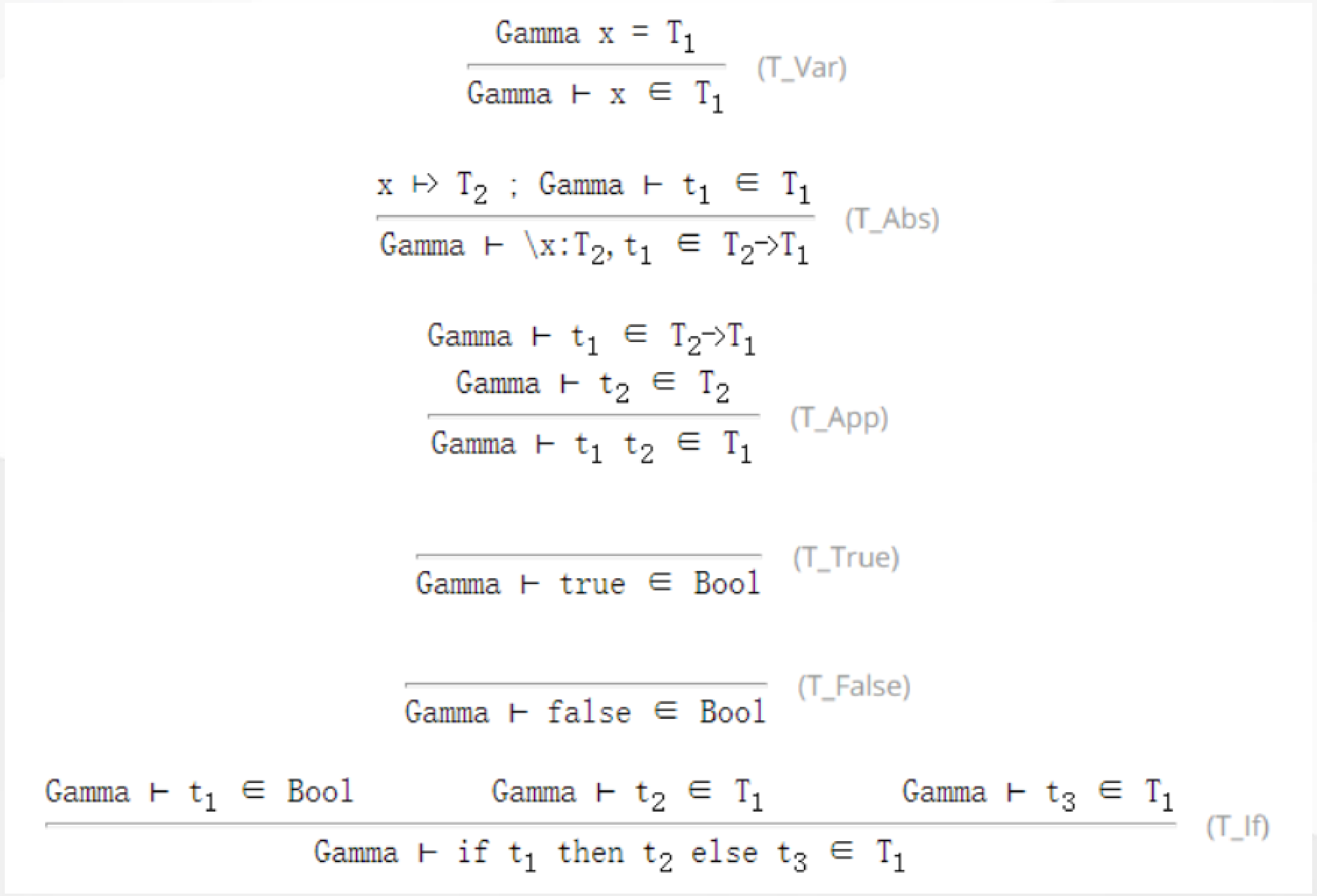
$$\frac{\begin{array}{c}\texttt{value } v_1 \\ t_2 \rightarrow t_2'\end{array}}{v_1\ t_2 \rightarrow v_1\ t_2'} \quad \text{(ST\_App2)}$$

$$\frac{}{(\texttt{if true then } t_1 \texttt{ else } t_2) \rightarrow t_1} \quad \text{(ST\_IfTrue)}$$

$$\frac{}{(\texttt{if false then } t_1 \texttt{ else } t_2) \rightarrow t_2} \quad \text{(ST\_IfFalse)}$$

$$\frac{t_1 \rightarrow t_1'}{(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3) \rightarrow (\texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3)} \quad \text{(ST\_If)}$$

Stlc - 类型推导规则



$$\frac{\texttt{Gamma x = T}_1}{\texttt{Gamma} \vdash \texttt{x} \in \texttt{T}_1} \quad \text{(T\_Var)}$$

$$\frac{\texttt{x} \mapsto \texttt{T}_2 \ ; \ \texttt{Gamma} \vdash \texttt{t}_1 \in \texttt{T}_1}{\texttt{Gamma} \vdash \texttt{\textbackslash x:T}_2\texttt{, t}_1 \in \texttt{T}_2\texttt{->T}_1} \quad \text{(T\_Abs)}$$

$$\frac{\begin{array}{c}\texttt{Gamma} \vdash \texttt{t}_1 \in \texttt{T}_2\texttt{->T}_1 \\ \texttt{Gamma} \vdash \texttt{t}_2 \in \texttt{T}_2\end{array}}{\texttt{Gamma} \vdash \texttt{t}_1 \ \texttt{t}_2 \in \texttt{T}_1} \quad \text{(T\_App)}$$

$$\frac{}{\texttt{Gamma} \vdash \texttt{true} \in \texttt{Bool}} \quad \text{(T\_True)}$$

$$\frac{}{\texttt{Gamma} \vdash \texttt{false} \in \texttt{Bool}} \quad \text{(T\_False)}$$

$$\frac{\texttt{Gamma} \vdash \texttt{t}_1 \in \texttt{Bool} \qquad \texttt{Gamma} \vdash \texttt{t}_2 \in \texttt{T}_1 \qquad \texttt{Gamma} \vdash \texttt{t}_3 \in \texttt{T}_1}{\texttt{Gamma} \vdash \texttt{if t}_1 \texttt{ then t}_2 \texttt{ else t}_3 \in \texttt{T}_1} \quad \text{(T\_If)}$$

Stlc - 扩展

加入更多的语言成分

- 自然数
- Let
- Pairs
- Unit
- Sums
- Lists
- 递归调用
- Records

Stlc - 再扩展

加入更多的语言成分

- 引用和赋值
- 子类型关系

欢迎（在研究生）选修编程语言的设计原理

为什么不让上本研合上课（怒

一些其他的 Topic

使用 Coq 做数学证明

定义点、线、面，及它们的关系

```
Parameter Point: Type.
Parameter Line: Type.
Parameter Plane: Type.

Parameter pt_on_l: Point -> Line -> Prop.
Parameter pt_on_pl: Point -> Plane -> Prop.
Parameter l_on_pl: Line -> Plane -> Prop.
```

使用 Coq 做数学证明

公理 I1：两点确定一条线段。

```
Axiom line: ∀ (A B: Point), A ≠ B -> Line.

Axiom line_exists: ∀ (A B: Point) (d: A ≠ B),
  let l := line A B d in A on l /\ B on l.

Axiom line_exists_l: ∀ (A B: Point) (d: A ≠ B), A on (line A B d).
Axiom line_exists_r: ∀ (A B: Point) (d: A ≠ B), B on (line A B d).
```

使用 Coq 做数学证明

定理：任意两点A、B，存在一点C在A和B之间（在线段AB上）。

教材：A5的纸，有图，6行。
Coq：86行。

```
A, C :  Point
dAC :  A ≠ C
lAC :=  line A C dAC : Line
HA_lAC :  A on line A C dAC
HC_lAC :  C on line A C dAC
E :  Point
HE_lAC :  ¬ E on line A C dAC
dEA :  E ≠ A
dAE :  A ≠ E
lAE :=  line A E dAE : Line
F :  Point
HF_lAE :  F on line A E dAE
HEbet :  E @ A -- F
dFA :  A ≠ F
HF_lAC :  ¬ F on lAC
dFC :  F ≠ C
lFC :=  line F C dFC : Line
```

关于继承

Inheritance is the base class of evil.

—— Sean Parent

- The requirement of a polymorphic type, by definition, comes from it's use.
- There are no polymorphic type, only a polymorphic use of similar types.

关于 Coq 的名字

The name Coq comes from the French word for rooster, CoC (the Calculus of Constructions) and Thierry Coquand, one of the initial authors of Coq. But it is also close to the word "cock" which has a slang meaning that some English speakers consider offensive (it also means a male bird or the firing lever in a gun). This similarity has already led to some women turning away from Coq and others getting harassed when they said they were working on Coq. It also makes some English conversations about Coq with lay persons simply more difficult.

答疑