软件理论基础与实践

Logic: Logic in Coq

熊英飞
北京大学

数理逻辑历史

- 逻辑学的发源一般追溯到亚里斯多德
  - 古希腊三圣：苏格拉底、柏拉图、亚里斯多德
  - 一般也被认为是科学的发源
- 19世纪数学家们试图给数学建立基础理论
  - 1854年，英国数学家布尔(George Boole)提出布尔代数
    - 注意：没有boolean这个词，只有Boolean
    - 约等于命题逻辑
  - 1879年，德国数学家戈弗雷格(Gottlob Frege)引入量词
  - 1913年，英国数学家怀特海(Alfred North Whitehead)和罗素(Bertrand Russell)引入了公里和推导规则

形式系统

- 逻辑通过形式系统来定义
- 形式系统包括以下四个部分*
  - 字母表Alphabet：一个符号的集合Σ
  - 文法Grammar：一组文法规则，定义Σ^*的一个子集，为该形式系统中可以写的命题集合
  - 公理模式Axiom Schemata：一组公理模板，定义命题集合的一个子集，代表为真的命题
  - 推导规则Inference Rules：一组推导规则，用于推导出公理以外为真的命题

- 注意形式系统是纯语法定义

* 部分定义（如北大版《离散数学》）也省略公理模式和推导规则，直接定义公理和推导式

一阶逻辑——字母表

- 一阶逻辑是目前最广泛使用的形式系统之一
- 除了一阶逻辑之外还存在很多其他逻辑
  - 命题逻辑、二阶逻辑、高阶逻辑、时序逻辑、霍尔逻辑、分离逻辑……
- 一阶逻辑字母表
  - 量词符号：$\forall, \exists$
  - 逻辑连接符：$\wedge, \vee, \rightarrow, \neg$
  - 括号：(,)
  - 变量符号的集合，记作$x_1, x_2, \ldots$
  - 谓词符号的集合，记作$p_1, p_2, \ldots$
  - 函数符号的集合，记作$f_1, f_2, \ldots$
  - 假设以上各部分交集为空

文法

- 一个文法包括：
  - 一组非终结符，用大写字母表示，如S, A, B
  - 一组终结符，用小写字母表示，如a, b
  - 一个非终结符作为开始符号，用字母S表示
  - 一组产生式，表示把一个字符串中的左边部分替换成右边部分，如
    - S->AB
    - aA->aaAb
    - A→ϵ （ϵ表示空串）

    - S->A, S->B可简写为S->A|B

文法生成的例子

- G2 = ({S, A}, {a, b}, S, {S → aAb, aA → aaAb, A → ε })
  - S ⇒ aAb　　　　　　应用S → aAb
    ⇒ aaAbb　　　　　应用aA → aAb
    ⇒ aaaAbbb　　　　应用aA → aaAb
    ⇒ aaabbb　　　　　应用A → ε
- 所有从S开始，应用任意多次产生式可以生成的所有终结符序列构成了文法对应语言
  - 即一个形式系统的语法所允许的所有命题
- 逻辑系统中通常采用上下文无关文法，即产生式左边只有一个非终结符

一阶逻辑——文法

- $Prop \ \rightarrow \ p_k(Term_1, \dots, Term_n)$
  - 该规则对任意谓词符号p_k有一个实例，n是和p_k有关的数
- $Prop \rightarrow \neg Prop$
- $Prop \rightarrow Prop_1 \wedge Prop_2$
- $Prop \rightarrow Prop_1 \vee Prop_2$
- $Prop \rightarrow \forall X \ Prop$
- $Prop \rightarrow \exists X \ Prop$
- $Term \rightarrow X$
- $Term \rightarrow f_k(Term_1, \dots, Term_n)$
  - 该规则对任意函数符号f_k有一个实例，n是和f_k有关的数
- $X \rightarrow x_1 \mid x_2 \mid \cdots$

一阶逻辑——文法

- 数学文献中存在很多不在上述写法中的简写
  - a + b代表plus(a, b)
  - $\forall a: Nat.\, P(a)$代表$\forall a\big(contains(a, Nat()) \rightarrow P(a)\big)$

- 严格来说，文法应该是一个定义了运算符优先级的无歧义版本（即所有命题都只有唯一产生方式），大多数文献中为了简单采用以上有歧义的文法。

推导式

- 推导式写作
  - $P_1, \dots, P_n \Rightarrow P$
- 或写作
  - $\frac{P_1 \quad P_2 \cdots P_n}{P}$
- 其中$P, P_1, \dots, P_n$是形式系统的命题

- 证明序列：给定一个前提集合和一组推导式集合I，如果存在命题序列$P_1, \dots, P_n$，使得对任意P_i满足
  - 要么P_i是前提
  - 要么存在一个推导式$P_{i1}, P_{i2}, \dots P_{ik} \Rightarrow P_i$，满足$i1, \dots, ik < i$
- 则$P_1, \dots, P_n$称为一个证明序列，记为
  - $P_{j1}, \dots, P_{jn} \vdash_I P_n$，
  - 其中$P_{j1}, \dots, P_{jn}$为该证明序列中的前提
  - P_n称为该证明序列的结论

公理模式

- 存在大量形式相同的公理
  - $A \wedge B \rightarrow A, B \wedge C \rightarrow B, A \wedge C \rightarrow A$
- 公理模式对同类公理进行统一定义
  - 对任意命题P, Q，$P \wedge Q \rightarrow P$
- 公理模式的文法：
  - 对原文法中任意非终结符E
  - 添加$E \rightarrow T_1 \mid T_2 \mid \cdots$
  - 其中$T_1, T_2, \ldots$是类别为E的元变量
  - 同时公理可能对元变量有额外约束
- 替换σ：一个从元变量到Σ^*的映射，其中$\sigma(T)$必须能从T对应的非终结符产生，并满足公理的额外约束
- 公理模式能产生的公理：对任意替换σ，将公理模式中的每一元变量T替换成$\sigma(T)$之后得到的命题

推导规则

- 和公理模式类似，推导规则用于产生一组推导式
  - 如对任意命题P, Q，$P \wedge Q \Rightarrow P$

- 一个形式系统的公理模式和推导规则统称推导系统。

- 给定形式系统FS，从其公理模式所能产生的公理、推导规则所能产生的推导式所能得出的结论F称为该形式系统的定理，记为$\vdash_{FS} F$

推导系统的性质

- 推导系统的一致性（Consistency）
  - 对任意命题P，P和$\neg P$中最多一个可以被推出来
- 推导系统的完备性（Completeness）
  - 对任意命题P，P和$\neg P$中至少一个可以被推出来

- 哥德尔不完备定理：一个能表达自然数的形式系统一定不同时具备这两个性质

推导系统的风格

- 希尔伯特风格演绎系统
  - 推导规则尽量少，极端情况只有一条推导规则
    - $P, P \to Q \Rightarrow Q$
  - 其他全是公理，比如
    - $A \wedge B \to \mathrm{A}$
- 自然演绎系统
  - 将和逻辑有关的公理放入推导规则，更接近自然推理，如：
    - $A \wedge B \Rightarrow \mathrm{A}$
  - 本质上和希尔伯特风格演绎系统等价
- 有时希尔伯特风格演绎系统指所有证明系统的全集，而自然演绎系统是其特例

Coq的逻辑

- Coq的逻辑系统称为Calculus of Inductive Constructions
  - 是通过Curry-Howard Correspondence在类型系统上实现的逻辑系统
  - 是一种高阶逻辑，包含了一阶逻辑的内容，但量词可以作用在谓词和函数上
- Coq的证明系统更为接近自然演绎系统
  - 逻辑符号在类型系统上大多数有特殊处理
  - 证明策略包含自然演绎系统的推导规则
    - 也包含一部分公理
  - 如apply对应规则$P, P \to Q \Rightarrow Q$
  - intro对应规则$P \Rightarrow \forall x\ P$，其中x是P中的自由变量
  - rewrite对应公理：
    - $t_1 = t_2 \to p\ t_1 \to p\ t_2$

形式系统的语义

- 一个形式系统的语义将形式系统的符号映射到数学的符号
- 一阶逻辑的语义将一阶逻辑映射到集合论
- 一阶逻辑的解释(Interpretation)：
  - 一个对象的定义域D，定义量词下变量的取值范围
  - 一个从函数符号到D上的函数的映射
  - 一个从谓词符号到D^n的子集的映射
- 一阶逻辑的语义：对逻辑运算符定义一组运算规则，使得给定一个解释，计算每个命题在集合论下的真值

推导系统的正确性和完备性

- 如果一个命题在所有解释下都为真，那么称为命题有效的（valid）
- 推导系统的正确性（Soundness）
  - 推出来的定理是都是有效命题
- 推导系统的完备性（Completeness）
  - 所有有效命题都能推出来
  - 在正确性的前提下，该属性是前一个完备性的推论，所以名字一样

理论(Theory)和理论的性质

- 理论：
  - 一个函数符号的子集
  - 一个谓词符号的子集
  - 一个关于这组函数符号和谓词符号的解释
  - 一组公理
- 理论的性质
  - 正确性(soundness)：基于该组公理推出来的所有定理都是该理论解释下的有效命题
  - 完备性(completeness)：该理论解释下的有效命题都能基于该组公理推出来

Coq中的命题

复习：Check命令

- Check可以用来返回表达式的类型
  - Check (negb true) : bool.
- Check也可以用来返回命题的定义
  - Check plus_id_example.
  - (**
  - * plus_id_example
  - *   : forall n m : nat, n = m -> n + n = m + m
  - *)
- 命题定义本身的类型是什么？

命题是表达式的一种

- 每个命题在Coq中是一个Prop类型的实例，无论是否成立

```
Check (3 = 3) : Prop.

Check (forall n m : nat, n + m = m + n) : Prop.

Check 3 = 2 : Prop.

Check forall n : nat, n = 2 : Prop.
```

命题也可以单独定义

```
Definition plus_claim : Prop := 2 + 2 = 4.
Check plus_claim : Prop.

Theorem plus_claim_is_true :
  plus_claim.
Proof. reflexivity.  Qed.

Definition injective {A B} (f : A -> B) :=
  forall x y : A, f x = f y -> x = y.

Lemma succ_inj : injective S.
Proof.
  intros n m H. injection H as H1. apply H1.
Qed.
```

- Coq中的谓词：返回命题的函数

谓词=

- =实际上是函数eq

```
Check @eq : forall A : Type, A -> A -> Prop.
```

- eq函数的具体定义方法在下一章介绍

逻辑连接

逻辑与

- 写作∧，实际上是一个函数
  - Check and : Prop -> Prop -> Prop.
- 目标中有逻辑与如何证明?

```
Example and_example : 3 + 4 = 7 /\ 2 * 2 = 4.

Proof.
  split.
  - (* 3 + 4 = 7 *) reflexivity.
  - (* 2 * 2 = 4 *) reflexivity.
Qed.
```

$$\frac{A\ \mathrm{true} \qquad B\ \mathrm{true}}{(A \wedge B)\ \mathrm{true}} \wedge_I$$

自然演绎系统中对应规则

逻辑与

- 该自然演绎规则是可以证明的

```
Lemma and_intro : forall A B : Prop, A -> B -> A /\ B.
Proof.
  intros A B HA HB. split.
  - apply HA.
  - apply HB.
Qed.

Example and_example' : 3 + 4 = 7 /\ 2 * 2 = 4.
Proof.
  apply and_intro.
  - (* 3 + 4 = 7 *) reflexivity.
  - (* 2 + 2 = 4 *) reflexivity.
Qed.
```

逻辑与

- 假设中有逻辑与如何使用?

```
Lemma and_example2 :
  forall n m : nat, n = 0 /\ m = 0 -> n + m = 0.
Proof.
  (* WORKED IN CLASS *)
  intros n m H.
  destruct H as [Hn Hm].
  rewrite Hn. rewrite Hm.
  reflexivity.
Qed.
```

复习：destruct之前是用来做什么的?

$$\frac{A \wedge B \text{ true}}{A \text{ true}} \wedge_{E1} \qquad \frac{A \wedge B \text{ true}}{B \text{ true}} \wedge_{E2}$$

自然演绎系统中对应规则

逻辑与

- destruct可以直接并入intro中

```
Lemma and_example2' :
  forall n m : nat, n = 0 /\ m = 0 -> n + m = 0.
Proof.
  intros n m [Hn Hm].
  rewrite Hn. rewrite Hm.
  reflexivity.
Qed.
```

$$\frac{A \wedge B \text{ true}}{A \text{ true}} \wedge_{E1} \qquad \frac{A \wedge B \text{ true}}{B \text{ true}} \wedge_{E2}$$

自然演绎系统中对应规则

逻辑与

- 不需要的参数也可以直接用下划线代替

```
Lemma proj1 : forall P Q : Prop,
  P /\ Q -> P.
Proof.
  intros P Q HPQ.
  destruct HPQ as [HP _].
  apply HP.  Qed.
```

$$\frac{A \wedge B \text{ true}}{A \text{ true}} \wedge_{E1} \qquad \frac{A \wedge B \text{ true}}{B \text{ true}} \wedge_{E2}$$

自然演绎系统中对应规则

关于逻辑与的定理

- 后续证明可直接使用

```
Theorem and_commut : forall P Q :
Prop,
  P /\ Q -> Q /\ P.

Theorem and_assoc : forall P Q R :
Prop,
  P /\ (Q /\ R) -> (P /\ Q) /\ R.
```

逻辑或

- 写作∨，也是一个函数
  - `Check or : Prop -> Prop -> Prop.`
- 目标中有逻辑或如何证明?

$$\frac{A \text{ true}}{A \vee B \text{ true}} \vee_{I1} \qquad \frac{B \text{ true}}{A \vee B \text{ true}} \vee_{I2}$$

```
Lemma or_intro_l : forall A B : Prop, A -> A \/ B.
Proof.
  intros A B HA.
  left.
  apply HA.
Qed.
```

逻辑或

- 写作∨，也是一个函数
  - `Check or : Prop -> Prop -> Prop.`
- 目标中有逻辑或如何证明?

$$\frac{A \text{ true}}{A \vee B \text{ true}} \vee_{I1} \qquad \frac{B \text{ true}}{A \vee B \text{ true}} \vee_{I2}$$

```
Lemma zero_or_succ :
  forall n : nat, n = 0 \/ n = S (pred n).
Proof.
  (* WORKED IN CLASS *)
  intros [|n'].
  - left. reflexivity.
  - right. reflexivity.
Qed.
```

逻辑或

- 前提中有逻辑或如何使用?
  - $A \vee B, A \rightarrow C, B \rightarrow C \Rightarrow C$

```
Lemma factor_is_O:
  forall n m : nat, n = 0 \/ m = 0 -> n * m = 0.
Proof.
  (* This pattern implicitly does case analysis on
     [n = 0 \/ m = 0] *)
  intros n m [Hn | Hm].
  - (* Here, [n = 0] *)
    rewrite Hn. reflexivity.
  - (* Here, [m = 0] *)
    rewrite Hm. rewrite <- mult_n_O.
    reflexivity.
Qed.
```

注意destruct的格式，和之前什么用法相同?

逻辑非

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

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

Check not : Prop -> Prop.
```

False：定义在标准库中用于表示假的命题

逻辑非

- 证明爆炸原理

```
Theorem ex_falso_quodlibet : forall (P:Prop),
  False -> P.
Proof.
  (* WORKED IN CLASS *)
  intros P contra.
  destruct contra.  Qed.
```

- ex_falso_quodlibet：爆炸原理的拉丁名称
- destruct类型为False的前提可自动证明定理
  - 之前什么策略也利用的爆炸原理?

逻辑非

- 基本证明方法——展开not的定义

```
Theorem zero_not_one : 0 <> 1.
Proof.
  unfold not.
  intros contra.
  discriminate contra.
Qed.
```

逻辑非

- 更复杂一点的证明

```
Theorem not_False :
  ~ False.
Proof.
  unfold not. intros H. destruct H. Qed.

Theorem contradiction_implies_anything : forall P Q : Prop,
  (P /\ ~P) -> Q.
Proof.
  intros P Q [HP HNA]. unfold not in HNA.
  apply HNA in HP. destruct HP.  Qed.

Theorem double_neg : forall P : Prop,
  P -> ~~P.
Proof.
  intros P H. unfold not. intros G. apply G. apply H.  Qed.
```

逻辑非

- 应用爆炸原理证明

```
Theorem not_true_is_false : forall
b : bool,
  b <> true -> b = false.
Proof.
  intros b H.
  destruct b eqn:HE.
  - (* b = true *)
    unfold not in H.
    apply ex_falso_quodlibet.
    apply H. reflexivity.
  - (* b = false *)
    reflexivity.
Qed.
```

可用策略
exfalso替代

真值

- True：标准库中定义的用于表示真的命题
- I：用来证明True为真的公理

```
Lemma True_is_true : True.
Proof. apply I. Qed.
```

真值

- 通常在证明中用不到，但特殊场合也可应用
- 如：替代discrimate证明

```
Definition disc_fn (n: nat) : Prop :=
  match n with
  | O => True
  | S _ => False
  end.

Theorem disc : forall n, ~ (O = S n).
Proof.
  intros n H1.
  assert (H2 : disc_fn O). { simpl. apply I. }
  rewrite H1 in H2. simpl in H2. destruct H2.
Qed.
```

蕴含关系

- 之前的证明中已经反复使用蕴含关系->
- 不同于其他逻辑连接词，蕴含关系是Coq内置的
- 蕴含关系出现在目标中
  - 推导规则：如果$\Gamma, \mathrm{P} \Rightarrow \mathrm{Q}$，则$\Gamma \Rightarrow \mathrm{P} \rightarrow \mathrm{Q}$
  - 对应策略：intro
- 蕴含关系出现在假设中
  - 推导规则：$P, P \rightarrow Q \Rightarrow Q$
  - 对应策略：apply

逻辑等价性

```
Definition iff (P Q : Prop) := (P -> Q) /\ (Q -> P).

Notation "P <-> Q" := (iff P Q)
                      (at level 95, no associativity)
                      : type_scope.
```

- 因为是用and连起来的两部分，所以证明策略和and相同

逻辑等价性

```
Theorem iff_sym : forall P Q : Prop,
  (P <-> Q) -> (Q <-> P).
Proof.
  intros P Q [HAB HBA].
  split.
  - (* -> *) apply HBA.
  - (* <- *) apply HAB.  Qed.

Lemma not_true_iff_false : forall b,
  b <> true <-> b = false.
Proof.
  intros b. split.
  - (* -> *) apply not_true_is_false.
  - (* <- *)
    intros H. rewrite H. intros H'. discriminate H'.
Qed.
```

rewrite和逻辑等价性

- 之前我们学过用rewrite来根据a=b进行重写
- 实际上rewrite可以用到任意等价关系上
  - setoid：带等价关系的集合
  - =是类型X上的等价关系
  - <->是Prop上的等价关系

rewrite和逻辑等价性

```
Lemma mul_eq_0 : forall n m, n * m = 0 <-> n = 0 \/ m = 0.

Theorem or_assoc :
  forall P Q R : Prop, P \/ (Q \/ R) <-> (P \/ Q) \/ R.


Lemma mul_eq_0_ternary :
  forall n m p, n * m * p = 0 <-> n = 0 \/ m = 0 \/ p = 0.
Proof.
  intros n m p.
  rewrite mul_eq_0. rewrite mul_eq_0. rewrite or_assoc.
  reflexivity.
Qed.
```

apply和逻辑等价性

- apply也可以直接采用逻辑等价性，会根据上下文猜测应用方向

```
Lemma apply_iff_example :
  forall n m : nat, n * m = 0 -> n = 0 \/ m = 0.
Proof.
  intros n m H. apply mul_eq_0. apply H.
Qed.
```

全称量词

- 写作forall，之前已经多次看过
- forall a:nat, p a等价于$\forall a\big(a \in Nat \rightarrow p(a)\big)$
- 全称量词出现在目标中
  - 推导规则：$P \Rightarrow \forall x\ P$
  - 对应策略：intro
- 全称量词出现在假设中
  - 推导规则：$\forall xP \Rightarrow P$[x\t]，其中t是P中的自由变量
  - 对应策略：apply

存在量词

- 写作exists，如
  - `Definition Even x := exists n : nat, x = double n.`
  - 在一阶逻辑中等价于$\exists n\big(n \in nat \wedge x = double(n)\big)$
- 存在量词出现在目标中
  - 推导规则：$P[x\backslash \mathrm{t}] \Rightarrow \exists x\ P$，
    - 其中x是P中的自由变量，t为任意项

```
Lemma four_is_even : Even 4.
Proof.
  unfold Even. exists 2. reflexivity.
Qed.
```

存在量词

- 存在量词出现在假设中
  - 推导规则：$P \to Q \Rightarrow (\exists x P) \to Q$，x不是Q中自由变量

```
Theorem exists_example_2 : forall n,
  (exists m, n = 4 + m) ->
  (exists o, n = 2 + o).
Proof.
  intros.
  (* n: nat
   * H: exists m : nat, n = 4 + m *)
  destruct H.
  (* n, x: nat
   * H: n = 4 + x *)
  exists (2 + m).
  apply Hm.  Qed.
```

递归定义的命题

递归定义的命题

- 定义列表中是否包括某个元素

```
Fixpoint In {A : Type} (x : A) (l : list A) : Prop :=
  match l with
  | [] => False
  | x' :: l' => x' = x \/ In x l'
  end.
```

递归定义的命题

- 证明方法和其他带递归结构的命题类似
- 1. 如果传的参数是具体值，展开成普通定理

```
Example In_example_1 : In 4 [1; 2; 3; 4; 5].
Proof.
(* In 4 [1; 2; 3; 4; 5] *)
  simpl.
(* 1 = 4 \/ 2 = 4 \/ 3 = 4 \/ 4 = 4 \/ 5 = 4 \/ False *)
  right. right. right. left. reflexivity.
Qed.
```

递归定义的命题

- 2. 如果传的参数不是具体值，利用induction递归证明

```
Theorem In_map : forall (A B : Type) (f : A -> B) (l : list A)
        (x : A), In x l -> In (f x) (map f l).
Proof. intros A B f l x.
(**  A : Type
 *   B : Type
 *   f : A -> B
 *   l : list A
 *   x : A
 *   ============================
 *   In x l -> In (f x) (map f l)
 *)
```

递归定义的命题

```
  induction l as [|x' l' IHl'].
(** [Coq Proof View]
 * 2 subgoals
 *
 *   A : Type
 *   B : Type
 *   f : A -> B
 *   x : A
 *   ============================================
 *   In x [ ] -> In (f x) (map f [ ])
 *
 * subgoal 2 is:
 *  In x (x' :: l') -> In (f x) (map f (x' :: l'))
 *)
```

递归定义的命题

```
- simpl.
(** [Coq Proof View]
 * 1 subgoal
 *
 *   A : Type
 *   B : Type
 *   f : A -> B
 *   x : A
 *   ==============
 *   False -> False
 *)
 intros [].
```

等价于intros contra. destruct contra.

递归定义的命题

```
- (** [Coq Proof View]
 * 1 subgoal
 *
 *   A : Type
 *   B : Type
 *   f : A -> B
 *   x' : A
 *   l' : list A
 *   x : A
 *   IHl' : In x l' -> In (f x) (map f l')
 *   ==========================================
 *   In x (x' :: l') -> In (f x) (map f (x' :: l'))
 *)
```

递归定义的命题

```
simpl.
(** [Coq Proof View]
 * 1 subgoal
 *
 *   A : Type
 *   B : Type
 *   f : A -> B
 *   x' : A
 *   l' : list A
 *   x : A
 *   IHl' : In x l' -> In (f x) (map f l')
 *   ==========================================================
 *   x' = x \/ In x l' -> f x' = f x \/ In (f x) (map f l')
 *)
intros [H | H].
    + rewrite H. left. reflexivity.
    + right. apply IHl'. apply H.
```

给定理传参数

给定理传参

- 在Poly一章，我们看到forall用于需要传递的类型参数
- 全称量词也是forall，是否同样也描述了参数?

```
Theorem in_not_nil :
  forall A (x : A) (l : list A), In x l -> l <> [].
Lemma in_not_nil_42 :
  forall l : list nat, In 42 l -> l <> [].
Proof.
  intros l H.
  Fail apply in_not_nil.
  (* Unable to find an instance for the variable x.*)
  apply (in_not_nil nat 42).
  apply H.
Qed.
```

给定理传参

- 蕴含和函数类型都用->描述，是否都可以传参?

```
Theorem in_not_nil :
  forall A (x : A) (l : list A), In x l -> l <> [].
Lemma in_not_nil_42 :
  forall l : list nat, In 42 l -> l <> [].
Proof.
  intros l H.
  Fail apply in_not_nil.
  (* Unable to find an instance for the variable x.*)
  apply (in_not_nil _ _ _ H).
Qed.
```

- 下一章介绍背后的原理

Coq的逻辑的特点

集合和类型

- 在集合论中，一个值可以属于多种集合
- Coq中最常用集合就是类型，但一个值只能属于一个类型
  - `Check nat.`
  - (* nat : Set *)
- 如果要在Coq里面定义集合，只能通过命题来间接定义
  - `Definition Even x := exists n : nat, x = double n.`

函数等价性

- 在集合论中，函数是二元组的集合
  - 所有的输入输出对都相同的话，两个函数相同
  - 称为Functional Extensionality
- 在Coq中，等价关系是在文本级别定义的
  - 导致有时函数等价性会无法证明
  - 下一章介绍原因

```
Example function_equality_ex1 :
  (fun x => 3 + x) = (fun x => (pred 4) + x).
Proof. reflexivity. Qed.
```

函数等价性

```
Example function_equality_ex2 :
  (fun x => plus x 1) = (fun x => plus 1 x).
Proof. (* 该定理在Coq中无法证明 *)
  Fail reflexivity.
  (* Unable to unify "1 + x" with "x + 1". *)
  Fail (rewrite add_comm).
  (* Found no subterm matching "?M60 + ?M61" in the current goal.*)
  Fail rewrite (add_comm x 1).
  (* The reference x was not found in the current environment. *)
Abort.
```

由于x是约束变量，无法为add_comm找到合适的参数。

函数等价性

- 添加公理来证明函数等价性

```
Axiom functional_extensionality : forall {X Y: Type} {f g : X -> Y},
  (forall (x:X), f x = g x) -> f = g.

Example function_equality_ex2 :
  (fun x => plus x 1) = (fun x => plus 1 x).
Proof. apply functional_extensionality. intros x.
(** [Coq Proof View]
 * 1 subgoal
 *
 *   x : nat
 *   =============
 *   x + 1 = 1 + x
 *)
  apply add_comm. Qed.
```

添加公理必须非常小心，不能导致逻辑不一致

命题和布尔的区别

- 逻辑式可以写作命题也可以写作布尔表达式
- 布尔表达式:
  - 逻辑式必须可判定
    - 返回bool的表达式必须在有限步终止
  - 可以用在match, if等语句中
  - 即使是等式，也不能用于rewrite
- 命题
  - 逻辑式无需可判定
  - 不可以用在match, if等语句中
  - 等式可以用于rewrite

用布尔表达式帮助证明

- 由于布尔表达式可判定，证明可能比命题容易

```
Example not_even_1001 :
  even 1001 = false.
Proof.
  reflexivity.
Qed.
```

```
Example not_even_1001'' :
  ~(Even 1001).
Proof.
  unfold Even.
  unfold not.
  intros.
  destruct H.
  destruct x.
  - simpl in H. discriminate.
  - simpl in H. destruct x.
    simpl in H. discriminate.
    simpl in H. destruct x.
    …(*重复500次*)
    simpl in H. discriminate.
    simpl in H. discriminate.
Qed.
```

用布尔表达式帮助证明

- 可以通过证明布尔表达式来帮助证明，称为Proof by reflection
- 在四色定理的证明中，用这个技巧减少了几千次分类讨论

```
Theorem even_bool_prop : forall n,
  even n = true <-> Even n.

Example not_even_1001' : ~(Even 1001).
Proof.
  rewrite <- even_bool_prop.
  unfold not. simpl. intro H. discriminate H.
Qed.
```

用命题帮助证明布尔表达式

- 这个技巧也可以反过来用

```
Theorem eqb_eq : forall n1 n2 : nat,
  n1 =? n2 = true <-> n1 = n2.

Lemma plus_eqb_example : forall n m p : nat,
  n =? m = true -> n + p =? m + p = true.
Proof.
  intros n m p H.
  rewrite eqb_eq in H.
  rewrite H.
  rewrite eqb_eq.
  reflexivity.
Qed.
```

经典逻辑vs直觉主义逻辑

- 经典逻辑Classical Logic：
  - 日常使用的逻辑系统通常是经典逻辑
- 直觉主义逻辑Intuitionistic Logic：
  - 相比经典逻辑，缺少排中律和其他一些等价公理（如：双重否定消除，见classical_axioms练习）
    - 排中律：$P \vee \neg P$
  - 好处：所有$\exists x. P$的定理证明都能提供具体的x值
    - 理解起来符合人的直觉
    - 又叫constructive logic，因为我们构造了x的值
  - 坏处：部分定理无法证明，部分证明方法无法使用
    - 反证法无法使用
    - 反证法使用规则$\neg P \rightarrow False \Rightarrow P$，等价于双重否定消除
- Coq采用的是直觉主义逻辑

例：排中律允许非构造证明

- 证明存在无理数a, b，使得a^b是有理数
- 证明:
  - $\sqrt{2}$是无理数
  - 如果$\sqrt{2}^{\sqrt{2}}$是有理数，那么令$a = b = \sqrt{2}$，结论成立
  - 如果$\sqrt{2}^{\sqrt{2}}$是无理数，那么
    - 令$a = \sqrt{2}^{\sqrt{2}}$，$b = \sqrt{2}$
    - 那么$a^b = \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = 2$，结论成立

- 以上证明过程中我们并没有给出一组满足条件的a和b的值

采用排中律证明1

- 对于可判定的式子，利用布尔表达式等价性证明

```
Theorem restricted_excluded_middle : forall P b,
  (P <-> b = true) -> P \/ ~ P.
Proof.
  intros P [] H.
  - left. rewrite H. reflexivity.
  - right. rewrite H. intros contra. discriminate contra.
Qed.

Theorem restricted_excluded_middle_eq : forall (n m : nat),
  n = m \/ n <> m.
Proof.
  intros n m.
  apply (restricted_excluded_middle (n = m) (n =? m)).
  symmetry. apply eqb_eq.
Qed.
```

采用排中律证明2

- 也可以将排中律加为公理
- 加入排中律后，Coq的逻辑仍然是一致的

作业

- 完成Logic.v中standard非optional且不在下面列表中的14道习题
  - tr_rev_correct
  - even_double_conv
  - eqb_list
  - 请使用最新英文版教材