软件科学基础习题课3

总复习

黄柘铳 TA

本slides不是划重点，之中没有涉及的知识点也可能考察。slides中指出的可能出题方式也只是TA的个人猜测。

Coq基础

Coq包含两种基本的语言：

- Gallina: 用于描述数学对象和证明的语言(一定终止，静态类型分析)
  `Inductive ...`， `Fixpoint ...`， `Definition ...`
- Ltac: Coq的策略语言，用于构造证明的脚本(不一定终止，动态类型分析)
  `simpl ...`， `intros ...`， `apply ...`

实际之中需要掌握这两种语言**怎么写**：

1. 填空题挖一个Fixpoint定义的代码片段
2. 从一个goal开始，自己写一段tactic，变成另一个goal

推荐大家过一下tactics这章，了解一些常用的证明策略

Coq逻辑

在Coq之中，表述一阶谓词逻辑：

逻辑概念	Coq表示	证明策略	说明
合取 (∧)	P /\ Q	split，destruct	证明时用split分解，使用时用destruct
析取 (∨)	P \/ Q	left，right，destruct	证明时选择左右分支，使用时分情况讨论
蕴含 (→)	P -> Q	intros，apply	引入假设用intros，应用用apply
否定 (¬)	~ P 或 P -> False	contradiction	否定即蕴含假
真	True	trivial，exact I	平凡真命题
假	False	contradiction	矛盾命题
全称量词 (∀)	forall x : T, P x	intros，apply	引入变量和假设
存在量词 (∃)	exists x : T, P x	exists，destruct	证明时用exists构造，使用时用destruct
等价 (↔)	P <-> Q	split，destruct	双向蕴含
等式	x = y	reflexivity，rewrite，symmetry	等式推理

Coq逻辑的特点

- Coq的逻辑基础是CIC(Calculus of Inductive Constructions)包含：
  i. Stlc：函数式编程（值依赖值是函数）
     Eg. `Defenition id (x: A) : A := x`
  ii. Parametric Polymorphism：值依赖类型
     Eg. `Defenition id (A: Type) (x: A) : A := x`
  iii. Type–level Function: 类型依赖类型
  iv. Dependent Types: 类型依赖值
     Eg. `Inductive List(n: nat) : Type := | nil : List 0 | cons : forall m, List m –> List (S m)`
  v. Inductive Types: 归纳类型

Coq逻辑的特点

- 注意命题`True,False`和布尔值`true,false`的区别：
  - `True`是一个类型，`I`是它的一个值
  - `False`是一个类型，没有值
  - `true`和`false`是布尔值，属于`bool`类型，可以用于if,match等
- Coq是直觉主义逻辑，无排中律，也没有双重否定律
  - `P \/ ~P`不成立
  - `~~P -> P`不成立

Coq逻辑-其他

- ProofObjects:
  根据Curry–Howard同构，Coq的命题可以看作是类型，证明可以看作是值
  了解简单的proof object的构造

```
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.
```

Coq逻辑-其他

- IndProp: 命题可以被归纳地定义为若干个元素之间的“关系”

```
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).

Notation "n <= m" := (le n m).
```

- IndPrinciple: 每个Inductive类型都有一个归纳原则

```
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.
```

IMP语言与Hoare逻辑

- IMP的语法：aexp, bexp,以及5种语句（assign, seq, if, while, skip）
- IMP的语义（表达式级别）：简单情况下，函数更常见
  i. 定义为函数 `aeval: aexp –> nat`…
  ii. 定义为关系 `aevalR: aexp –> nat –> Prop`…
- IMP的语义（全语言级别）：
  i. 定义为关系（大步语义） `step: state –> com –> state –> Prop`
  ii. Hoare逻辑: `{P} C {Q}`表示在状态满足P的前提下，执行C后状态满足Q
  iii. 定义为关系（小步语义） `step: (state * com) –> (state * com) –> Prop`

IMP的操作语义（大步）

$$\frac{}{\texttt{st =[skip]=> st}}\ \text{(E\_Skip)}$$

$$\frac{\texttt{aeval st a = n}}{\texttt{st =[x := a]=> (x !-> n ; st)}}\ \text{(E\_Ass)}$$

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

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

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

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

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

IMP的操作语义（Hoare逻辑）

$$\text{SKIP}\quad \frac{\{P\}\ \text{skip}\ \{P\}}{\{P\}}$$

$$\text{ASSIGN}\quad \frac{}{\{P[a/x]\}x := a\{P\}}$$

$$\text{CONSEQUENCE}\quad \frac{\models (P \Rightarrow P') \quad \{P'\}c\{Q'\} \quad \models (Q' \Rightarrow Q)}{\{P\}c\{Q\}}$$

$$\text{SEQ}\quad \frac{\{P\}c_1\{R\} \quad \{R\}c_2\{Q\}}{\{P\}c_1; c_2\{Q\}}$$

$$\text{IF}\quad \frac{\{P \wedge b\}c_1\{Q\} \quad \{P \wedge \neg b\}c_2\{Q\}}{\{P\}\text{if } b \text{ then } c_1 \text{ else } c_2\{Q\}}$$

$$\text{WHILE}\quad \frac{\{P \wedge b\}c\{P\}}{\{P\}\ \text{while}\ b\ \text{do}\ c\{P \wedge \neg b\}}$$

Decorated Imp

```
  {{ X <= 3 }}
while X <= 2 do
    {{ X <= 3 /\ X <= 2 }} ->>
    {{ X + 1 <= 3 }}
  X := X + 1
    {{ X <= 3 }}
end
  {{ X <= 3 /\ ~(X <= 2) }} ->>
  {{ X = 3 }}
```

一个显然的考点是给定一个Imp程序，要求你写出/补全它的decoration

Stlc

```
t ::= x (variable)
    | \x:T,t (abstraction)
    | t t (application)
    | true (constant true)
    | false (constant false)
    | if t then t else t (conditional)

T ::= Bool
    | T → T
```

Stlc小步语义

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

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

$$\frac{\texttt{value } v_1 \qquad t_2 \to t_2'}{v_1\ t_2 \to v_1\ t_2'}\ \text{(ST\_App2)}$$

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

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

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

Stlc类型规则

$$\frac{\texttt{Gamma x = } T_1}{\texttt{Gamma} \vdash \texttt{x} \in T_1} \quad \text{(T\_Var)}$$

$$\frac{\texttt{x} \mapsto T_2 \texttt{ ; Gamma} \vdash t_1 \in T_1}{\texttt{Gamma} \vdash \backslash\texttt{x}{:}T_2,\, t_1 \in T_2\texttt{->}T_1} \quad \text{(T\_Abs)}$$

$$\frac{\begin{array}{c}\texttt{Gamma} \vdash t_1 \in T_2\texttt{->}T_1 \\ \texttt{Gamma} \vdash t_2 \in T_2\end{array}}{\texttt{Gamma} \vdash t_1\ t_2 \in T_1} \quad \text{(T\_App)}$$

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

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

$$\frac{\texttt{Gamma} \vdash t_1 \in \texttt{Bool} \qquad \texttt{Gamma} \vdash t_2 \in T_1 \qquad \texttt{Gamma} \vdash t_3 \in T_1}{\texttt{Gamma} \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \in T_1} \quad \text{(T\_If)}$$

Stlc性质

- Progress: 任何一个类型正确的表达式都可以被求值
- Preservation: 任何一个类型正确的表达式在求值过程中保持类型正确
- Stlc扩展: Nat, Let, Pairs, Unit, Sums, Lists, Fix, Records
- Progress, Preservation两个性质在扩展之后依然成立，然而此时求值的过程不一定会终止（Fix引入了任意的递归函数）
- 需要掌握Progress, preservation的推导，比如如果我改了/新加入某个规则，那么他们是否依然成立

Reference

- 基本操作：
  i. 取地址： `ref tm`
  ii. 解引用： `!tm`
  iii. 赋值： `tm := tm`
  配合上`let ... in ...`，可以写出类似命令式的程序（其实就是一种Monad）
- 经典例题:

```
let newcounter =  \_:Unit.
        let c = ref 0 in
        let incc = \_:Unit, (c := succ (!c); !c) in
        let decc = \_:Unit, (c := pred (!c); !c) in
        {i=incc, d=decc}
```

如下代码返回什么？

```
let c1 = newcounter unit in
let c2 = newcounter unit in
let r1 = c1.i unit in let r1 = c1.i unit in
let r2 = c2.i unit in let r2 = c2.i unit in
r2
```

去掉第一个_:Unit会怎样？去掉第二个会怎样？

- 考点，用reference实现某个递归函数

Subtyping

- 子类的引入：如果$A <: B$，那么在任意一个上下文中，A的值也可以被看作是B的值
- 子类关系
  - 反射性：$A <: A$
  - 传递性：$A <: B \land B <: C \Rightarrow A <: C$
  - Records:

$$\frac{\begin{array}{c}\forall j_k \in j_1 \ldots j_n, \\ \exists i_p \in i_1 .. i_m, \text{ such that} \\ j_k = i_p \,\&\&\, \mathrm{S_p} <: \mathrm{T_k}\end{array}}{\{i_1 : S_1 \ldots \mathrm{i_m} : \mathrm{S_m}\} <: \{j_1 : T_1 \ldots \mathrm{j_n} : \mathrm{T_n}\}}$$

- 逆变与协变：
  - 子类关系对于大部分类型都是协变的
  - 对于函数的输入参数是逆变的，对于函数的返回值是协变的: $\frac{\mathrm{T_1 <: S_1} \quad \mathrm{S_2 <: T_2}}{\mathrm{S_1 \to S_2 <: T_1 \to T_2}}$
  - 对于引用类型既然是协变的也是逆变的: $\frac{\mathrm{S_1 <: T_1} \quad \mathrm{T_1 <: S_1}}{\mathrm{Ref\ S_1 <: Ref\ T_1}}$

祝大家复习顺利～