课程内容

Lambda演算部分ppt

中文讲义

Software Foundations

2022年10月8日线上课程录制

主题: 函数语言程序设计

日期: 2022-10-08 12:07:56

录制文件：https://meeting.tencent.com/v2/cloud-record/share?id=8321413f-7b24-4ead-aba2-0f117e0f4e20&from=3&is-single=true

访问密码：bSuj

函数语言程序设计20221010课程录像-destruct-induction.mp4

本视频任课教师设置为不允许下载，未经许可，请不要对该教学视频进行下载、录音、录像和传播。

函数语言程序设计20221017-Lists.mp4

本视频任课教师设置为不允许下载，未经许可，请不要对该教学视频进行下载、录音、录像和传播。

复习题

函数语言程序设计20221024Polymorphism.mp4

本视频任课教师设置为不允许下载，未经许可，请不要对该教学视频进行下载、录音、录像和传播。

函数语言程序设计-OCaml 20221212.mp4

本视频任课教师设置为不允许下载，未经许可，请不要对该教学视频进行下载、录音、录像和传播。

OCaml.pdf

函数语言程序设计 20221219 OCaml.mp4

本视频任课教师设置为不允许下载，未经许可，请不要对该教学视频进行下载、录音、录像和传播。

作业1（9.5）

1. **练习 1.4.** 令 $ST(M)$ 为 λ-项 M 的所有子项的集合，并假设 M 是其自身的一个子项。利用结构归纳的方式定义函数 $ST(\cdot)$。例如，

$$ST(\lambda x.xy) = \{\lambda x.xy,\ xy,\ x,\ y\}.$$

2. **练习 1.6.** 比较如下四个表达式的异同，说明重命名和替换操作的不同之处。

$$(\lambda x.xy)\{y/x\}, \qquad (\lambda x.xy)[y/x], \qquad (\lambda y.xy)\{y/x\}, \qquad (\lambda y.xy)[y/x]\ .$$

作业2（9.20）

练习 1.6. 下面四个项是否都能归约到范式：

(1) $(\lambda xy.x)(\lambda x.xx)(\lambda x.xx)$

(2) $(\lambda xy.x)((\lambda x.xx)(\lambda x.xx))$

(3) $(\lambda x.x)(\lambda y.yyy)(\lambda xy.x)$

(4) $(\lambda fx.fxx)(\lambda xy.y)y$

练习 1.18. 斯科特数（*Scott numerals*）是对自然数的另一种编码。

- 数字零 $\mathbf{Z} := \lambda z.\lambda s.\, z$
- 后继函数 $\mathbf{S} := \lambda n.\lambda z.\lambda s.\, s\, n$
- 模式匹配 $\mathbf{caseN} := \lambda n.\lambda u.\lambda f.\, n\, u\, f$

斯科特编码的自然数有下列形式的范式：

$$\begin{aligned} \mathbf{one} &:= \lambda z.\lambda s.\, s\,\mathbf{Z} \\ \mathbf{two} &:= \lambda z.\lambda s.\, s\,\mathbf{one} \\ \mathbf{three} &:= \lambda z.\lambda s.\, s\,\mathbf{two} \\ &\ \ \vdots \end{aligned}$$

为得到斯科特编码的加法，我们注意到

$$\mathbf{add}\, n\, m = \mathbf{caseN}\, n\, m\, (\lambda n'.\, \mathbf{S}\, (\mathbf{add}\, n'\, m))$$

验证这个等式成立，并借助一个不动点写出用 λ-项表示的加法函数。

作业3（9.26）

1. 为下面这个类型判断构造一棵推导树。

$$\vdash \lambda f^{A\to A} x^{A}.f(f(fx)) : (A \to A) \to A \to A$$

2. **练习 2.3.** 根据下面定义的 food 类型，定义一个函数 vegetarian，使得 vegetarian f 返回 true 当且仅当 f 是除了 steak 之外的食物。

```
Inductive fruit : Type :=
  | apple
  | pear.

Inductive food : Type :=
  | rice
  | steak
  | dessert (f : fruit).
```

作业4（10.8）

1. 定义函数gtb，使得gtb n m 返回布尔值 true当且仅当 n > m。至少用两种方法定义这个函数。

2. 证明如下性质。

Theorem plus_1_l' : forall n m o: nat, n = m -> m = o -> 1 + n = S o.

作业5（10.10）

完成下面四个性质的证明。提示：建议按照给定的次序完成。

1.
```
Theorem plus_n_Sm : ∀ n m : nat,
  S (n + m) = n + (S m).
Proof.
  (* FILL IN HERE *) Admitted.
```

2.
```
Theorem add_shuffle3 : ∀ n m p : nat,
  n + (m + p) = m + (n + p).
Proof.
  (* FILL IN HERE *) Admitted.
```

3.
```
Theorem mul_n_Sm : forall n m : nat,  n + n * m = n * S m.

Proof.
  (* FILL IN HERE *) Admitted.
```

4.
```
Theorem mul_comm : ∀ m n : nat,
  m × n = n × m.
Proof.
  (* FILL IN HERE *) Admitted.
```

作业6（10.18）

1. Complete the following definition of `alternate`, which interleaves two lists into one, alternating between elements taken from the first list and elements from the second.

```
Fixpoint alternate (l1 l2 : natlist) : natlist
  (* REPLACE THIS LINE WITH ":= _your_definition_ ." *). Admitted.

Example test_alternate1:
  alternate [1;2;3] [4;5;6] = [1;4;2;5;3;6].
  (* FILL IN HERE *) Admitted.

Example test_alternate2:
  alternate [1] [4;5;6] = [1;4;5;6].
  (* FILL IN HERE *) Admitted.

Example test_alternate3:
  alternate [1;2;3] [4] = [1;4;2;3].
  (* FILL IN HERE *) Admitted.
```

2. 假设我们用列表来表示集合，定义函数 inter使得（inter l1 l2)的结果为l1和l2这两个集合的交集。

作业7（10.24）

1. 定义函数max使得(max L)返回类型为natoption：当自然数列表L为空时返回None, 否责返回Some n，其中n为L中最大元素。

2. **练习 2.29.** 定义函数 maxPair，把输入的一个自然数列表中最大的奇数和偶数找出来，组成一个二元组作为返回值。如果列表中没有奇数或偶数，则用 0 替代。例如，

```
Example test_maxPair1: maxPair [1;2;5;4;8;10;3] = (5, 10).
Proof. reflexivity. Qed.
Example test_maxPair2: maxPair [2;4] = (0, 4).
Proof. reflexivity. Qed.
```

3.

```
Theorem rev_app_distr: ∀ X (l1 l2 : list X),
  rev (l1 ++ l2) = rev l2 ++ rev l1.
Proof.
  (* FILL IN HERE *) Admitted.
```

4.

```
Theorem rev_involutive : ∀ X : Type, ∀ l : list X,
  rev (rev l) = l.
Proof.
  (* FILL IN HERE *) Admitted.
```

作业8（10.31）

1. The function `map` maps a `list` X to a `list` Y using a function of type X → Y. We can define a similar function, `flat_map`, which maps a `list` X to a `list` Y using a function `f` of type X → `list` Y. Your definition should work by 'flattening' the results of `f`, like so:

```
    flat_map (fun n ⇒ [n;n+1;n+2]) [1;5;10]       = [1; 2; 3; 5; 6; 7; 10; 11; 12].
  Fixpoint  flat_map { X  Y:  Type} ( f:  X  →  list  Y) ( l:  list  X)
               :  list  Y
      (* REPLACE THIS LINE WITH ":= _your_definition_ ." *).  Admitted.

  Example  test_flat_map1:
     flat_map ( fun  n  ⇒  [ n ; n ; n ])  [1 ;5 ;4 ]
     =  [1 ;  1 ;  1 ;  5 ;  5 ;  5 ;  4 ;  4 ;  4 ].
    (* FILL IN HERE *)   Admitted.
```

2. 定义函数changelist使得（changelist L）返回一个新列表，把自然数列表L中所有的奇数扩大3倍，偶数扩大2倍。

`Example test: changelist [1;2;3;4;5;6] = [3; 4; 9; 8; 15; 12].`

3. 定义函数sumPair使得(sumPair L)返回一对元素，前一个为自然数列表L中所有奇数的和，后一个为L中所有偶数的和。

`Example test_sumPair : sumPair [1;2;3;4;5] = (9,6).`

4. 假设我们用列表代表集合，L为一个集合的集合，且所有集合中的元素在自然数0和n之间，定义函数bigInter使得(bigInter L n)返回L中所有元素的交集。

`Example test_bigInter: bigInter [[1;3;5]; [2;3;7;6;5]; [3;9;8;5]] 10 = [3;5].`

作业9（11.7）

1. Theorem or_commut : forall P Q : Prop, P ∨ Q -> Q ∨ P.

Proof. (* FILL IN HERE *) Admitted.

2.
```
Theorem not_both_true_and_false : ∀ P : Prop,
  ¬ (P ∧ ¬P).
Proof.
  (* FILL IN HERE *) Admitted.
```

3.
```
Theorem or_distributes_over_and : ∀ P Q R : Prop,
  P ∨ (Q ∧ R) -> (P ∨ Q) ∧ (P ∨ R).
Proof.
  (* FILL IN HERE *) Admitted.
```

4.
```
Theorem or_distributes_over_and' : ∀ P Q R : Prop,
  (P ∨ Q) ∧ (P ∨ R) -> P ∨ (Q ∧ R).
Proof.
  (* FILL IN HERE *) Admitted.
```

作业10（11.14）

1.
```
Theorem dist_exists_or : ∀ (X:Type) (P Q : X → Prop),
  (∃ x, P x ∨ Q x) ↔ (∃ x, P x) ∨ (∃ x, Q x).
Proof.
   (* FILL IN HERE *) Admitted.
```

2. Inductively define a relation CE such that (CE m n) holds iff m and n are two consecutive even numbers with m smaller than n.

Example test_CE : CE 4 6.

Proof. (* Fill in here *) Admitted.

3. Theorem CE_SS : forall n m, CE (S (S n)) (S (S m)) -> CE n m.

Proof. (* Fill in here *) Admitted.

4.
```
Theorem In_app_iff : ∀ A l l' (a:A),
  In a (l++l') ↔ In a l ∨ In a l'.
Proof.
  intros A l. induction l as [|a' l' IH].
  (* FILL IN HERE *) Admitted.
```

5. Drawing inspiration from `In`, write a recursive function `All` stating that some property `P` holds of all elements of a list `l`. To make sure your definition is correct, prove the `All_In` lemma below. (Of course, your definition should *not* just restate the left-hand side of `All_In`.)

```
  Fixpoint  All  { T  :  Type} ( P  :  T  →  Prop) ( l  :  list  T) :  Prop
      (* REPLACE THIS LINE WITH ":= _your_definition_ ." *).  Admitted.

  Theorem  All_In  :
     ∀  T  ( P  :  T  →  Prop) ( l  :  list  T),
       ( ∀  x,  In  x  l  →  P  x )  ↔
       All  P  l.
  Proof.
      (* FILL IN HERE *)   Admitted.
```

作业11（11.21）

1.
```
Theorem ev_sum : ∀ n m, ev n → ev m → ev (n + m).
Proof.
  (* FILL IN HERE *) Admitted.
```

2.
```
Lemma le_trans : ∀ m n o, m ≤ n → n ≤ o → m ≤ o.
Proof.
  (* FILL IN HERE *) Admitted.
```

3. `Example reg_exp_ex3 : [0;1;0;1] =~ Star (App (Char 0) (Char 1)).`

`Proof. (* FILL IN HERE *) Admitted.`

4.
```
Lemma MUnion' : ∀ T (s : list T) (re1 re2 : reg_exp T),
  s =~ re1 ∨ s =~ re2 →
  s =~ Union re1 re2.
Proof.
  (* FILL IN HERE *) Admitted.
```

作业12（11.30）

1. Construct a proof object for the following proposition.

`Definition conj_disj : forall P Q R,  (P /\ Q) \/ (P /\ R) -> P /\ (Q \/ R)`

(* REPLACE THIS LINE WITH ":= _your_definition_ ." *) . Admitted.

2. Complete this proof without using the `induction` tactic.

```
  Theorem      plus_one_r'    :      ∀      n:   nat,
       n      +    1      =      S      n.
  Proof.
       (* FILL IN HERE *)       Admitted.
```

3.
```
Theorem lt_trans' :
  transitive lt.
Proof.
  (* Prove this by induction on evidence that m is less than o. *)
  unfold lt. unfold transitive.
  intros n m o Hnm Hmo.
  induction Hmo as [| m' Hm'o].
    (* FILL IN HERE *) Admitted.
```

4.
```
Lemma rsc_trans :
  ∀ (X:Type) (R: relation X) (x y z : X),
      clos_refl_trans_1n R x y  →
      clos_refl_trans_1n R y z →
      clos_refl_trans_1n R x z.
Proof.
  (* FILL IN HERE *) Admitted.
```

作业13（12.5）

Since the `optimize_0plus` transformation doesn't change the value of `aexp`s, we should be able to apply it to all the `aexp`s that appear in a `bexp` without changing the `bexp`'s value. Write a function that performs this transformation on `bexp`s and prove it is sound. Use the tacticals we've learned to make the proof as elegant as possible.

```
  Fixpoint      optimize_0plus_b    (   b    :      bexp) :      bexp
       (* REPLACE THIS LINE WITH ":= _your_definition_ ." *).      Admitted.

  Theorem      optimize_0plus_b_sound    :      ∀      b,
       beval    (   optimize_0plus_b      b)      =      beval       b.
  Proof.
       (* FILL IN HERE *)       Admitted.
```

作业14（12.12）

先定义一个类型person，记录一个人的姓名和年龄。定义函数maxage，求一列人中年龄最大的那位。例如，

maxage [{name = "John"; age = 19}; {name = "Bob"; age = 20}; {name = "David"; age = 18}];;

(* - : person = {name = "Bob"; age = 20} *)

作业15（12.19）

1. 定义函数split使得（split l x）返回一对列表，其中一个包含l中所有小于x的元素，另一包含l中其它元素。例如，

```
partition [2; 5; 1; 7; 3; 9; 3; 0; 10] 5 ;;
- : int list * int list = ([2; 1; 3; 3; 0], [5; 7; 9; 10])
```

2. 利用上面的split函数实现对一个列表进行快速排序。例如，

```
quicksort [2; 5; 1; 7; 3; 9; 3; 0; 10];;
- : int list = [0; 1; 2; 3; 3; 5; 7; 9; 10]
```