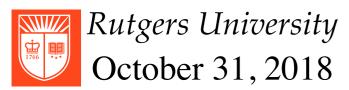
# CS 314 Principles of Programming Languages

Lecture 17: Lambda Calculus Exam Help

https://powcoder.com

Add WeChat powcoder

Prof. Zheng Zhang



### **Class Information**

- Midterm exam 11/7 Wednesday 10:20am 11:40am
- Extended hours are Posted
- No classes on 11/2 this Friday

Assignment Project Exam Help

https://powcoder.com

## Review: Lambda Calculus - Historical Origin

## • Church's model of computing is called the *lambda calculus*

It is based on the notion of parameterized expressions (with each parameter introduced by an occurrence of the letter  $\lambda$  — hence the notation's name). Lambda calculus was the inspiration for functional programming: one uses it to compute by *substituting* parameters into expressions, just as one computes in a high level functional program by passing arguments to functions.

https://powcoder.com

## **Review: Functional Programming**

• Functional languages such as Lisp, Scheme, FP, ML, Miranda, and Haskell are an attempt to realize Church's lambda calculus in practical form as a programming language

## • The key idea: do everything by composing functions

- No mutable state
- No side effects Assignment Project Exam Help
- Function as first-class values https://powcoder.com

### **Review: Lambda Calculus**

 $\lambda$ -terms are inductively defined.

#### A $\lambda$ -term is:

- a variable x
- $(\lambda x. M)$   $\Rightarrow$  where x is a variable and  $\lambda$  is a  $\lambda$ -term (abstraction)
- (M N)  $\Rightarrow$  where M and N are both  $\lambda$ -terms (application) Assignment Project Exam Help

https://powcoder.com

**Review:** λ-terms

The context-free grammar for  $\lambda$ -terms:

```
\lambda-term \rightarrow expr

expr \rightarrow name | number | \lambda name . expr | func arg

func \rightarrow name | (\lambda name . expr ) | func arg

arg \rightarrow name | number | (\lambda prame Expr | d | pfunc arg )
```

### Example 1:

https://powcoder.com

 $\lambda$  y Add WeChat powcoder name (as parameter) expr (another  $\lambda$ -term)

**Review:** λ-terms

The context-free grammar for  $\lambda$ -terms:

```
\lambda-term \rightarrow expr

expr \rightarrow name | number | \lambda name . expr | func arg

func \rightarrow name | (\lambda name . expr ) | func arg

arg \rightarrow name | number | \lambda prame Expr | \lambda func arg )
```

### Example 2:



### **Review: Lambda Calculus**

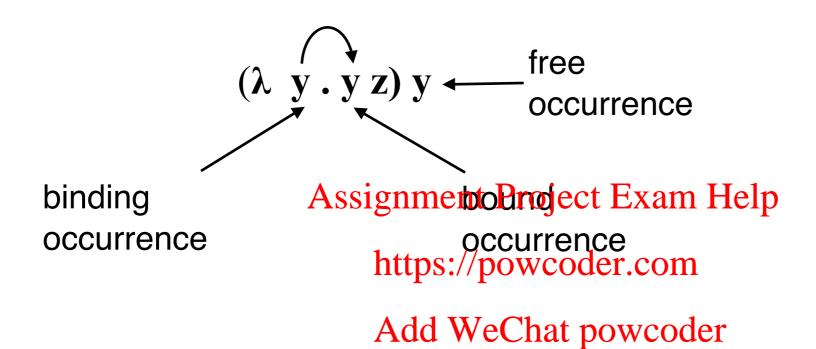
## Associativity and Precedence

- Function application is left associative: (f g z) is ((f g) z)
- Function application has precedence over function abstraction. "function body" extends as far to the right as possible:  $(\lambda x.yz)$  is  $(\lambda x.yz)$
- Multiple arguments: (λχγ,z) is (λχ(λγ,z)) Help

https://powcoder.com

### Review: Free and Bound Variables

Abstraction ( $\lambda x$ . M) "binds" variable x in "body" M. You can think of this as a declaration of variable x with scope M.

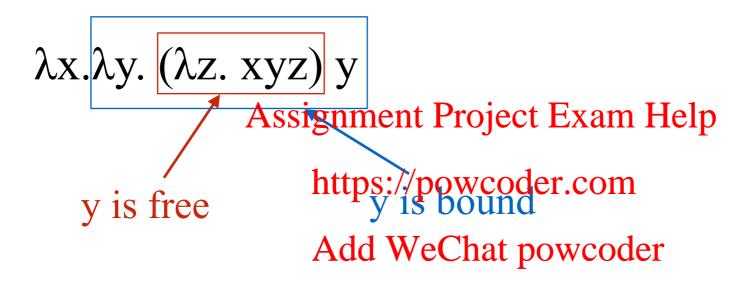


### **Review: Free and Bound Variables**

### Note:

A variable can occur **free** and **bound** in a  $\lambda$ -term.

## Example:



"free" is relative to a  $\lambda$ -sub-term.

### **Review: Free and Bound Variables**

Let M, N be  $\lambda$ -terms and x is a variable. The set of *free variable* of M, free(M), is defined inductively as follows:

- free(x) =  $\{x\}$
- free(M N) = free(M)  $\cup$  free(N)
- free  $(\lambda x.M)$  = free(M) free(x)

Assignment Project Exam Help

https://powcoder.com

## **Review: Function Application**

Computation in lambda calculus is based on the concept or reduction. Simplify an expression until it can no longer be simplified.

### **β**–reduction:

$$(\lambda x.\mathbf{E})y \rightarrow_{\beta} \mathbf{E}[y/x]$$

- 1. Return function body E
- 2. Replace every https://powredee.com/x in E with y

## **Review: Function Application**

Computation in lambda calculus is based on the concept or reduction. Simplify an expression until it can no longer be simplified.

### **β**–reduction:

$$(\lambda x.\mathbf{E})y \rightarrow_{\beta} \mathbf{E}[y/x]$$

- 1. Return function body E
- 2. Replace every https://pourredee.com/x in E with y

Add WeChat powcoder

## Example:

$$(\lambda a.\lambda b.a+b) 2 x \rightarrow_{\beta} (\lambda b.2+b) x$$

$$\rightarrow_{\beta} 2+x$$

# **Function Application**

Computation in lambda calculus is based on the concept or reduction. Simplify an expression until it can no longer be simplified.

### **β**–reduction:

$$(\lambda x.\mathbf{E})y \rightarrow_{\beta} \mathbf{E}[y/x]$$

### Assignment Project Exam Help

## We should not perform for education be a bound variable within E

### Add WeChat powcoder

Example:

$$(\lambda a.\lambda b.a+b) b 2 \rightarrow_{\beta} (\lambda b.b+b) 2 \rightarrow Incorrect$$

$$\rightarrow_{\beta} 2+2$$

b is a bound variable within λa.λb.a+b

This is called capturing

# **Review: Function Application**

Computation in lambda calculus is based on the concept or reduction. Simplify an expression until it can no longer be simplified.

### α-reduction:

$$(\lambda x.E) \rightarrow_{\alpha} \lambda y.E[y/x]$$

Assignment Project Exam Help

https://powcoder.com

## **Review: Function Application**

Computation in lambda calculus is based on the concept or reduction. Simplify an expression until it can no longer be simplified.

#### α-reduction:

$$(\lambda x. \mathbf{E}) \rightarrow_{\alpha} \lambda y. \mathbf{E}[\mathbf{y}/\mathbf{x}]$$

$$Assignment Project Exam Help$$

$$https://powcoder.com$$

$$(\lambda a. \lambda b. a + b) b 2 \rightarrow_{\alpha} (\lambda a. \lambda x. a + x) b 2$$

$$Add WeChat powcoder$$

$$\rightarrow_{\beta} \lambda x. b + x 2$$

$$\rightarrow_{\beta} b + 2$$

# Review: Programming in Lambda Calculus

Remember: Computation in the lambda calculus is a sequence of applications of reduction rules (mostly  $\beta$ –reductions).

Logical constants and operations (incomplete list):

true  $\equiv \lambda a$ .  $\lambda b$ . a select-first select-second

Assignment Project Exam Help

cond  $\equiv \lambda p$ . The properties  $\lambda p$  and  $\lambda p$  and  $\lambda p$  are  $\lambda p$  and  $\lambda p$  are  $\lambda p$ . The  $\lambda p$  are  $\lambda p$  and  $\lambda p$  are  $\lambda p$  are  $\lambda p$  and  $\lambda p$  are  $\lambda p$  are  $\lambda p$  and  $\lambda p$  are  $\lambda p$  are  $\lambda p$  and  $\lambda p$  are  $\lambda p$  and  $\lambda p$  are  $\lambda p$ 

not  $\equiv \lambda x$ . (Xafalse true) powcoder

and  $\equiv \lambda x. \lambda y. (x y false)$ 

 $or \equiv homework$ 

# Review: Programming in Lambda Calculus

What about data structures?

Data structures:

pairs can be represented as:

# Assignment Project Exam Help

```
first \equiv \lambda x. (x true) Add WeChat powcoder

second \equiv \lambda x. (x false) (cdr)

build \equiv \lambda x.\lambda y.\lambda z. (z x y) (cons)
```

# Programming in Lambda Calculus

What about the encoding of arithmetic constants?

### Church Numerals:

```
0 \equiv \lambda f x. \ x
1 \equiv \lambda f x. \ (f \ x)
2 \equiv \lambda f x. \ (f \ (f \ x))
\dots
Assignment Project Exam Help
\dots
n \equiv \lambda f x. \ (f \ (f \ (\dots \ (f \ x) \ .) https: \#proper (for x) om
```

The natural number n is represented as a function that applies a function f n-times to x.

```
succ \equiv \lambda m. (\lambda fx.(f(m f x)))
add \equiv \lambda mn. (\lambda fx.((m f) (n f x)))
mult \equiv \lambda mn. (\lambda fx.((m (n f)) x))
isZero? \equiv \lambda m. (m \lambda x.false true)
```

### Recursion in Lambda Calculus

Does this make sense?

$$f \equiv \dots f \dots$$

In lambda calculus, ≡ is "abbreviated as", but not an assignment.

Assignment Project Exam Help

https://powcoder.com

### Recursion in Lambda Calculus

Does this make sense?

F

$$f \equiv \dots f \dots$$

In lambda calculus, ≡ is "abbreviated as", not an assignment.

add = 
$$\lambda$$
mn. ( if (isZero?n) m (add (m+1) (n-1)) )

Incorrect Exameter Project Project Exameter Project Exameter Project Pro

 $\lambda mn.(if (isZero?n) m (add (m+1) (n-1))) = add$ 

F add = add

"add" is a fixed point to function F

# The fixed point of a function g is the set of values

$$\{ x \mid x = g(x) \}$$

## Examples:

function <b>g</b>	Assignment	t Projex(E) am Help
λx.6	https://j	powcoder.com {6}
$\lambda x.(6 - x)$		eChat powcoder
$\lambda x.((x * x) +$	(x - 4))	{-2, 2}
$\lambda x.x$		entire domain of function f
$\lambda x.(x+1)$		{ }

## Is there a way to "compute" the fixed point of any function F

$$x = F(x)$$

YES. x = YF, and Y is called the fixed point combinator.

Y ≠ Astig(nonef(txPx)) e(otxEx(anx H) elp

https://powcoder.com

 $YF = ((\lambda f.((\lambda x.f(xx))(\lambda x.f(xx))(\lambda x.f(xx)))))$ 

## Is there a way to "compute" the fixed point of any function F

$$x = F(x)$$

YES. x = YF, and Y is called the fixed point combinator.

$$YF = ((\lambda f.((\lambda x.\underline{f}(x x)) (\lambda x.\underline{f}(x x)))))$$

## Is there a way to "compute" the fixed point of any function F

$$x = F(x)$$

YES. x = YF, and Y is called the fixed point combinator.

$$YF = ((\lambda f.((\lambda x.f(x x)) (\lambda x.f(x x)) (\lambda x.f(x x)))) + F$$

$$= (\lambda x.F(x x)) (\lambda x.F(x x))$$

## Is there a way to "compute" the fixed point of any function F

$$x = F(x)$$

YES. x = YF, and Y is called the fixed point combinator.

$$YF = ((\lambda f.((\lambda x.f(x x)) (\lambda x.f(x x)) + (\lambda x.f(x x))))$$

$$= (\lambda x.F(x x)) (\lambda x.F(x x))$$

## Is there a way to "compute" the fixed point of any function F

$$x = F(x)$$

YES. x = YF, and Y is called the fixed point combinator.

$$YF = ((\lambda f.((\lambda x.f(x x)) (\lambda x.f(x x)) + f)))$$

$$= (\lambda x.F(x x)) (\lambda x.F(x x))$$

$$= F((\lambda x.F(x x)) (\lambda x.F(x x)))$$

## Is there a way to "compute" the fixed point of any function F

$$x = F(x)$$

YES. x = YF, and Y is called the fixed point combinator.

$$YF = ((\lambda f.((\lambda x.f(xx))(\lambda x.f(xx)))))$$

$$YF = (\lambda x.F(x x)) (\lambda x.F(x x))$$

$$YF = F((\lambda x.F(x x))(\lambda x.F(x x)))$$

## Is there a way to "compute" the fixed point of any function F

$$x = F(x)$$

YES. x = YF, and Y is called the fixed point combinator.

$$YF = ((\lambda f.((\lambda x.f(x x)) (\lambda x.f(x x)))))$$

$$YF = (\lambda x.F(x x)) (\lambda x.F(x x))$$

$$YF = F((\lambda x.F(x x))(\lambda x.F(x x)))$$

$$YF = F(YF)$$

## The Y - Combinator Example (Cont.)

• Informally, the Y-Combinator allows us to get as many copies of the recursive procedure body as we need. The computation "unrolls" recursive procedure calls one at a time

$$Y \equiv \lambda f.((\lambda x.f(x x)) (\lambda x.f(x x)))$$

Assignment Project Exam Help

https://powcoder.com

### The Y - Combinator

### Example:

```
\mathbf{F} = \lambda \mathbf{f}. (\lambda \mathbf{m}n. if (isZero? n) then m else (\mathbf{f} (succ m) (pred n)))
(YF 3 2) = (((\lambda f.((\lambda x.f(x x))(\lambda x.f(x x)))) F) 3 2)
            = ((F(YF))32)
            = ((\lambda mn.if (isZero? n) then m else YF (succ m) (pred n))) 3 2)
            = if (iszeignmentherpisceles and (pred 2))
            = (YF 4 https://powcoder.com
            = ((F (YF))dd WeChat powcoder
            = if (isZero? 1) then 4 else YF (succ 4) (pred 1))
            = (YF 5 0)
            = (F(YF) 5 0)
            = if (isZero? 0) then 5 else (YF (succ 5) (pred 0))
            = 5
```

### Lambda Calculus - Final Remarks

- We can express all computable functions in our  $\lambda$ -calculus.
- All computable functions can be expressed by the following two combinators, referred to as **S** and **K**.
  - $K \equiv \lambda xy.x$
  - $S = \lambda xyz.xz(yz)$

Combinatoric logic Assaigpower Public Texing Medphines.

https://powcoder.com

### **Next Lecture**

## Reading:

- Scott, Chapter 11.1 11.3
- Scott, Chapter 11.7
- ALSU, Chapter 11.1 11.3

Assignment Project Exam Help

https://powcoder.com