

Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages
Lambda calculus basics; Lambda calculus encodings and Recursion (Lectures 7–8)
Section and Practice Problems

Week 5: Tue Feb 26–Fri Mar 1, 2019


1 Lambda Calculus Basics

- (a) **Variable Bindings** Fully parenthesize each expression based on the standard parsing of λ -calculus expressions, i.e. you should parenthesize all applications and λ abstractions. Then, draw a box around all binding occurrences of variables, underline all usage occurrence of variables, and circle all free variables. For each bound usage occurrence, *neatly* draw an arrow to indicate its corresponding binding occurrence. (You may also use other methods to indicate binding occurrences of variables, usage occurrences of variables, free variables, and which uses correspond to which bindings.)

- $\lambda a. z \lambda z. a y$

Answer:

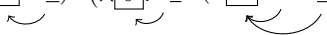
$(\lambda \boxed{a}. \textcircled{z} (\lambda \boxed{z}. \underline{a} \textcircled{y}))$



- $(\lambda z. z) \lambda b. b \lambda a. a a$

Answer:

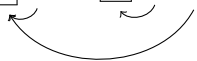
$(\lambda \boxed{z}. \underline{z}) (\lambda \boxed{b}. \underline{b} (\lambda \boxed{a}. \underline{a} \underline{a}))$



- $\lambda b. b \lambda a. a b$

Answer:

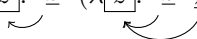
$(\lambda \boxed{b}. \underline{b} (\lambda \boxed{a}. \underline{a} \underline{b}))$



- $\lambda z. z \lambda z. z z$

Answer:

$(\lambda \boxed{z}. \underline{z} (\lambda \boxed{z}. \underline{z} \underline{z}))$



- $\lambda a. \lambda b. (\lambda a. a) \lambda b. a$

Answer:

$(\lambda \boxed{a}. (\lambda \boxed{b}. (\lambda \boxed{a}. a) (\lambda \boxed{b}. a)))$

- $x \lambda x. \lambda x. x (\lambda x. x)$

Answer:

$(\textcircled{x} (\lambda \boxed{x}. (\lambda \boxed{x}. \underline{x} (\lambda \boxed{x}. \underline{x}))))$

- $y (\lambda y. y)(\lambda y. z)$

Answer:

$(\textcircled{y} (\lambda \boxed{y}. \underline{y}) (\lambda \boxed{y}. \textcircled{z}))$

(b) **Alpha equivalence:** Which of these three lambda-calculus expressions are alpha equivalent?

- $\lambda x. y \lambda a. a x$
- $\lambda x. z \lambda b. b x$
- $\lambda a. y \lambda b. b a$

(Hint: to figure out whether two expressions are alpha equivalent, you need to know which variables are free and which variables are bound.)

Answer: Expressions i. and iii. are alpha-equivalent. Note that alpha equivalence applies only to bound variables; free variables cannot be renamed.

(c) **Evaluation** For each of the following terms, do the following: (a) write the result of one step of the *call-by-value* reduction of the term; (b) write the result of one step of the *call-by-name* reduction of the term; and (c) write *all* possible results of one step under *full β -reduction*. If the term cannot take a step, please note that instead.

- $(\lambda z. \lambda x. x x) (\lambda y. y)$

Answer: The semantics for CBV and CBN are the same, we have:

$$(\lambda z. \lambda x. x x) (\lambda y. y) \longrightarrow \lambda x. x x$$

There are no other possible reductions since only one application exists.

- $\lambda a. \lambda b. (\lambda c. c) (\lambda d. d)$

Answer: There are no possible steps under CBV and CBN because neither allows reductions inside a lambda term. For full β -reduction, we have the following:

$$\lambda a. \lambda b. (\lambda c. c) (\lambda d. d) \longrightarrow \lambda a. \lambda b. \lambda d. d$$

- $(\lambda x. x x x x) (\lambda x. \lambda y. x y)$

Answer: Under CBV and CBN we have:

$$(\lambda x. x x x x) (\lambda x. \lambda y. x y) \longrightarrow (\lambda x. \lambda y. x y) (\lambda x. \lambda y. x y) (\lambda x. \lambda y. x y) (\lambda x. \lambda y. x y)$$

and under full β -reduction we have no additional steps.

- $(\lambda x. \lambda y. x y) ((\lambda w. \lambda z. w z) (\lambda x. x))$

Answer: Under CBV we have:

$$(\lambda x. \lambda y. x y) ((\lambda w. \lambda z. w z) (\lambda x. x)) \longrightarrow (\lambda x. \lambda y. x y) (\lambda z. (\lambda x. x) z)$$

and under CBN:

$$(\lambda x. \lambda y. x y) ((\lambda w. \lambda z. w z) (\lambda x. x)) \longrightarrow \lambda y. ((\lambda w. \lambda z. w z) (\lambda x. x)) y$$

There are no additional steps that we can take under full β -reduction.

- $(\lambda a. (\lambda b. b a) a) (\lambda z. z) (\lambda w. w)$

Answer: First, note that the fully parenthesized expression is

$$((\lambda a. (\lambda b. b a) a) (\lambda z. z)) (\lambda w. w).$$

Under both CBV and CBN we have

$$(\lambda a. (\lambda b. b a) a) (\lambda z. z) (\lambda w. w) \longrightarrow ((\lambda b. b (\lambda z. z)) (\lambda z. z)) (\lambda w. w).$$

In addition, under full β -reduction we can also reduce the subexpression $(\lambda b. b a) a$, giving us

$$(\lambda a. (\lambda b. b a) a) (\lambda z. z) (\lambda w. w) \longrightarrow (\lambda a. a a) (\lambda z. z) (\lambda w. w).$$

- (d) Suppose we have an applied lambda calculus with integers and addition. Write the sequence of expressions that the following lambda calculus term evaluates to under call-by-value semantics. Then do the same under call-by-name semantics.

$$(\lambda f. f (f 8)) (\lambda x. x + 17)$$

Answer: The term under CBN semantics evaluates to:

$$\begin{aligned}
 (\lambda f. f (f 8)) (\lambda x. x + 17) &\longrightarrow (\lambda x. x + 17) ((\lambda x. x + 17) 8) \\
 &\longrightarrow ((\lambda x. x + 17) 8) + 17 \\
 &\longrightarrow (8 + 17) + 17 \\
 &\longrightarrow 25 + 17 \\
 &\longrightarrow 42
 \end{aligned}$$

and under CBV we have:

$$\begin{aligned}
 (\lambda f. f (f 8)) (\lambda x. x + 17) &\longrightarrow (\lambda x. x + 17) ((\lambda x. x + 17) 8) \\
 &\longrightarrow (\lambda x. x + 17) (8 + 17) \\
 &\longrightarrow (\lambda x. x + 17) (25) \\
 &\longrightarrow 25 + 17 \\
 &\longrightarrow 42
 \end{aligned}$$

2 Lambda calculus encodings

(a) Evaluate *AND FALSE TRUE* under CBV semantics.

Answer: Recall that:

$$\begin{aligned}
 TRUE &\triangleq \lambda x. \lambda y. x \\
 FALSE &\triangleq \lambda x. \lambda y. y \\
 AND &\triangleq \lambda b_1. \lambda b_2. b_1 b_2 FALSE
 \end{aligned}$$

We can substitute these definitions and evaluate:

$$\begin{aligned}
 AND \ FALSE \ TRUE &\triangleq (\lambda b_1. \lambda b_2. b_1 b_2 \ FALSE) (\lambda x. \lambda y. y) (\lambda x. \lambda y. x) \\
 &\longrightarrow (\lambda b_2. (\lambda x. \lambda y. y) b_2 \ FALSE) (\lambda x. \lambda y. x) \\
 &\longrightarrow (\lambda x. \lambda y. y) (\lambda x. \lambda y. x) \ FALSE \\
 &\longrightarrow (\lambda y. y) \ FALSE \\
 &\longrightarrow \ FALSE
 \end{aligned}$$

(b) Evaluate *IF FALSE Ω λx. x* under CBN semantics. What happens when you evaluated it under CBV semantics?

Answer: Recall that:

$$\begin{aligned}
 IF &\triangleq \lambda b. \lambda t. \lambda f. b \ t \ f \\
 \Omega &\triangleq (\lambda x. x \ x) (\lambda x. x \ x)
 \end{aligned}$$

We can substitute these definitions and evaluate under CBN semantics:

$$\begin{aligned}
 \text{IF FALSE } \Omega (\lambda x. x) &\triangleq (\lambda b. \lambda t. \lambda f. b \ t \ f) (\lambda x. \lambda y. y) ((\lambda x. x \ x) (\lambda x. x \ x)) (\lambda x. x) \\
 &\longrightarrow (\lambda t. \lambda f. (\lambda x. \lambda y. y) \ t \ f) ((\lambda x. x \ x) (\lambda x. x \ x)) (\lambda x. x) \\
 &\longrightarrow (\lambda f. (\lambda x. \lambda y. y) ((\lambda x. x \ x) (\lambda x. x \ x)) \ f) (\lambda x. x) \\
 &\longrightarrow (\lambda x. \lambda y. y) ((\lambda x. x \ x) (\lambda x. x \ x)) (\lambda x. x) \\
 &\longrightarrow (\lambda y. y) (\lambda x. x) \\
 &\longrightarrow (\lambda x. x)
 \end{aligned}$$

If we evaluate this expression under CBV semantics, we need to evaluate the Omega expression before we can apply the lambda abstraction. Since the Omega expression evaluates indefinitely, the overall expression never terminates.

- (c) Evaluate $\text{ADD } \bar{2} \ \bar{1}$ under CBV semantics. (Make sure you know what the Church encoding of 1 and 2 are, and check that the answer is equal to the Church encoding of 3.)

Answer: Recall that:

$$\begin{aligned}
 \bar{1} &\triangleq \lambda f. \lambda x. f \ x \\
 \bar{2} &\triangleq \lambda f. \lambda x. f \ (f \ x) \\
 \text{PLUS} &\triangleq \lambda n_1. \lambda n_2. n_1 \ \text{SUCC} \ n_2 \\
 \text{SUCC} &\triangleq \lambda n. \lambda f. \lambda x. f \ (n \ f \ x)
 \end{aligned}$$

We can substitute these definitions and evaluate under CBV semantics:

$$\begin{aligned}
 \text{ADD } \bar{2} \ \bar{1} &\triangleq (\lambda n_1. \lambda n_2. n_1 \ \text{SUCC} \ n_2) \ \bar{2} \ \bar{1} \\
 &\longrightarrow (\lambda n_2. \bar{2} \ \text{SUCC} \ n_2) \ \bar{1} \\
 &\longrightarrow (\bar{2} \ \text{SUCC} \ \bar{1}) \\
 &\longrightarrow (\lambda f. \lambda x. f \ (f \ x)) \ \text{SUCC} \ \bar{1} \\
 &\longrightarrow (\lambda x. \text{SUCC} \ (\text{SUCC} \ x)) \ \bar{1} \\
 &\longrightarrow \text{SUCC} \ (\text{SUCC} \ \bar{1}) \\
 &\longrightarrow \text{SUCC} \ ((\lambda n. \lambda f. \lambda x. f \ (n \ f \ x)) \ \bar{1}) \\
 &\longrightarrow \text{SUCC} \ (\lambda f. \lambda x. f \ (\bar{1} \ f \ x)) \\
 &\longrightarrow \lambda f. \lambda x. f \ ((\lambda f. \lambda x. f \ (\bar{1} \ f \ x)) \ f \ x)
 \end{aligned}$$

This is functionally equivalent to $\bar{3}$

- (d) In class we made use of a combinator *ISZERO*, which takes a Church encoding of a natural number n , and evaluates to *TRUE* if n is zero, and *FALSE* if n is not zero. (We don't care what *ISZERO* does if it is applied to a lambda term that is not a Church encoding of a natural number.)

Define *ISZERO*.

Answer: We define *ISZERO* to be the following:

$$\text{ISZERO} \triangleq \lambda n. n \ (\lambda x. \text{FALSE}) \ \text{TRUE}$$

3 Recursion

Assume we have an applied lambda calculus with integers, booleans, conditionals, etc. Consider the following higher-order function H .

$$H \triangleq \lambda f. \lambda n. \text{if } n = 1 \text{ then true else if } n = 0 \text{ then false else not } (f (n - 1))$$

(a) Suppose that g is the fixed point of H . What does g compute?

Answer: g computes whether a number is odd.

(b) Compute $Y H$ under CBN semantics. What has happened to the function call $f (n - 1)$?

Answer: Recall that:

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

We can substitute this definition and evaluate under CBN semantics:

$$\begin{aligned} ISODD &= Y H \triangleq (\lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))) H \\ &\rightarrow (\lambda x. H (x x)) (\lambda x. H (x x)) \\ &\rightarrow H ((\lambda x. H (x x)) (\lambda x. H (x x))) \\ &\rightarrow (\lambda f. \lambda n. \text{if } n = 1 \text{ then true else if } n = 0 \text{ then false else not } (f (n - 1))) ((\lambda x. H (x x)) (\lambda x. H (x x))) \\ &\rightarrow (\lambda n. \text{if } n = 1 \text{ then true else if } n = 0 \text{ then false else not } (((\lambda x. H (x x)) (\lambda x. H (x x))) (n - 1))) \\ &=_{\beta} (\lambda n. \text{if } n = 1 \text{ then true else if } n = 0 \text{ then false else not } (ISODD (n - 1))) \end{aligned}$$

Here, note that $((\lambda x. H (x x)) (\lambda x. H (x x)))$ was the result of beta-reducing $Y H$ and it has replaced f in the evaluation. Thus, f becomes our recursive call $ISODD$.

(c) Compute $(Y H) 2$ under CBN semantics.

Answer: From above:

$$\begin{aligned} &(\lambda n. \text{if } n = 1 \text{ then true else if } n = 0 \text{ then false else not } (ISODD (n - 1))) 2 \\ &\rightarrow \text{if } 2 = 1 \text{ then true else if } 2 = 0 \text{ then false else not } (ISODD (2 - 1)) \\ &\rightarrow \text{if false then true else if } 2 = 0 \text{ then false else not } (ISODD (2 - 1)) \\ &\rightarrow \text{if } 2 = 0 \text{ then false else not } (ISODD (2 - 1)) \\ &\rightarrow \text{if false then false else not } (ISODD (2 - 1)) \\ &\rightarrow \text{not } (ISODD (2 - 1)) \\ &\rightarrow \text{not } (ISODD 1) \\ &\rightarrow * \text{ not true} \\ &\rightarrow \text{false} \end{aligned}$$

(d) Use the “recursion removal trick” to write another function that behaves the same as the fixed point of H .

Answer: Define a function H'

$$H' \triangleq \lambda f. \lambda n. \text{if } n = 1 \text{ then true else if } n = 0 \text{ then false else not } (f \ f \ (n - 1))$$
$$g = H' \ H'$$