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

Section 3

---

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.)

- $\lambda a.\, z\ \lambda z.\, a\ y$
- $(\lambda z.\, z)\ \lambda b.\, b\ \lambda a.\, a\ a$
- $\lambda b.\, b\ \lambda a.\, a\ b$
- $\lambda z.\, z\ \lambda z.\, z\ z$
- $\lambda a.\, \lambda b.\, (\lambda a.\, a)\ \lambda b.\, a$
- $x\ \lambda x.\, \lambda x.\, x\ (\lambda x.\, x)$
- $y\ (\lambda y.\, y)(\lambda y.\, z)$

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

i. $\lambda x.\, y\ \lambda a.\, a\ x$

ii. $\lambda x.\, z\ \lambda b.\, b\ x$

iii. $\lambda a.\, y\ \lambda b.\, b\ a$

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

(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.

- $(\lambda z.\, \lambda x.\, x\ x)\ (\lambda y.\, y)$
- $\lambda a.\, \lambda b.\, (\lambda c.\, c)\ (\lambda d.\, d)$
- $(\lambda x.\, x\ x\ x\ x)\ (\lambda x.\, \lambda y.\, x\ y)$
- $(\lambda x.\, \lambda y.\, x\ y)\ ((\lambda w.\, \lambda z.\, w\ z)\ (\lambda x.\, x))$
- $(\lambda a.\, (\lambda b.\, b\ a)\ a)\ (\lambda z.\, z)\ (\lambda w.\, w)$

(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.

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

2 Lambda calculus encodings

(a) Evaluate $AND\ FALSE\ TRUE$ under CBV semantics.

(b) Evaluate $IF\ FALSE\ \Omega\ \lambda x.\, x$ under CBN semantics. What happens when you evaluated it under CBV semantics?

(c) Evaluate $ADD\ \overline{2}\ \overline{1}$ 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.)

(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 $ISZERO$.

3 Recursion

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

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

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

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

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

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