York University- Department of Computer Science and Engineering

SC/CSE 3401 3.00 – Functional and Logic Programming

Solutions to assignment 3

---

1) (5 marks) What is the order of evaluation of arguments in Common LISP? Are the arguments evaluated from left to right, or right to left? Under what circumstances does the order make a difference? Write two expressions (one for each of above methods) to test LISP's strategy. [refer to exercise 7 of Chapter 1- Wilensky]

Answer.

The order of evaluation is important when the value of an expression depends on the preceding expression being evaluated. The order of evaluation in Common LISP is left to right. The evaluation of the following expressions in Common LISP proves this strategy.

```
> (+ (setq y 3) (+ y 3))
9
> (+ (+ z 5) (setq z 2))
Error! Variable Z has no value!
```

---

2) (10 marks) In a certain application, we want to be able to call a function **dist** that can find the Euclidean distance between two points for which the coordinates are given as a list. In addition, we want to be able to call this function with one (or two) nil argument(s) indicating that we want to find the distance to a reference point, the coordinates of which are saved in a global variable **ref**. See the following for examples:

```
> (dist '(1  2) '(4  -2))
5
> (setq ref '(0 1))
(0 1)
> (dist '(0 2) nil)
1
> (dist nil nil)
0
```

Write the function definition for dist. The efficiency and readability of your code is important. Do not use setq in the function definition.

Answer.

```
; The dist function is defined as an application of a distance
; calculating lambda function to the correct arguments
; The if statements supply the correct arguments

(defun dist (p1 p2)
     ((lambda (x y)
          (sqrt (+ (expt (- (car x) (car y)) 2)
                   (expt (- (cadr x) (cadr y)) 2))))
       (if (null p1) ref p1) (if (null p2) ref p2)))
```

Or a simpler (more traditional way) would be:

```
(defun dist (p1 p2)
  (let*
      ((q1 (cond               ; q1 is the first point
            ((null p1) ref)    ; set it to ref if p1 is null
            (T p1)))           ; otherwise it is p1
       (q2 (cond               ; q2 is the second point
            ((null p2) ref)
            (T p2)))
       (x1 (car q1))           ; obtain coordinates from lists
       (y1 (car (cdr q1)))     ; and save in x1, y1, x2, y2
       (x2 (car q2))
       (y2 (car (cdr q2))) )
     (sqrt (+ (expt (- x1 x2) 2) (expt (- y1 y2) 2))))); calculate
```

---

3) (6 marks) Assume you have evaluated the following in LISP:

```
(setq v1 '((a b) (x d)))
(setq v2 '((a b) . (x d)))
(setq v3 (append (car v1) (car (cdr v1)) ))
```

Use car and cdr to return X when applied to

(a) V1
(b) V2
(c) V3

Answer.

```
(a) (car (car (cdr v1)))
(b) (car (cdr v2))
(c) (car (cdr (cdr v3)))
```

---

4) (8 marks) Do Exercise 3 on page 12 of Selinger's lecture notes (both parts a and b). In addition, for part b, (iii) show the term calculation.

Answer.

(a)

$(i)\ (\lambda x.(\lambda y.(\lambda z.((xz)(yz))))) \Rightarrow \lambda xyz.xz(yz)$

$(ii)\ (((ab)(cd))((ef)(gh))) \Rightarrow ab(cd)(ef(gh))$

$iii)\ (\lambda x.((\lambda y.(yx))(\lambda v.v)z)u)(\lambda w.w) \Rightarrow (\lambda x.(\lambda y.yx)(\lambda v.v)zu)\lambda w.w$

(b)

$(i)\ xxxx \Rightarrow (((xx)x)x)$

$(ii)\ \lambda x.x\lambda y.y \Rightarrow (\lambda x.(x(\lambda y.y)))$

$(iii)\ \lambda x.(x\lambda y.yxx)x \Rightarrow (\lambda x.((x(\lambda y.((yx)x)))x))$

term calculatio n:

$x,\quad y,\quad (yx),\quad ((yx)x), (\lambda y.((yx)x)),\quad (x(\lambda y.((yx)x))),$

$((x(\lambda y.((yx)x)))x),\quad (\lambda x.((x(\lambda y.((yx)x)))x))$

---

5) (8 marks)Assume that true (T) is defined as $\lambda xy.x$ and false (F) is defined as $\lambda xy.y$. Prove that the function $\lambda pq.ppq$ can implement OR(˅) in logic. (Hint: Show evaluation of OR for all four cases of the truth table of OR).

Answer.

$$(T \vee T) = (\lambda pq.ppq)TT \to_\beta (\lambda q.TTq)T \to_\beta TTT = (\lambda xy.x)TT \to_\beta (\lambda y.T)T \to_\beta T$$

$$(F \vee T) = (\lambda pq.ppq)FT \to_\beta (\lambda q.FFq)T \to_\beta FFT = (\lambda xy.y)FT \to_\beta (\lambda y.y)T \to_\beta T$$

$$(T \vee F) = (\lambda pq.ppq)TF \to_\beta (\lambda q.TTq)F \to_\beta TTF = (\lambda xy.x)TF \to_\beta (\lambda y.T)F \to_\beta T$$

$$(F \vee F) = (\lambda pq.ppq)FF \to_\beta (\lambda q.FFq)F \to_\beta FFF = (\lambda xy.y)FF \to_\beta (\lambda y.y)F \to_\beta F$$

As we can see above, applying the function above to all four possible states reduces to the value we expect to get from the OR function.

---

6) (8 marks) Do the following λ-terms have a β-normal form? If yes, how many? Show reduction steps.
Note: In λ-calculus, instead of the infix operators such as x + y, the prefix operator is used e.g. (+ x y).

(a) *(λx.* x 4)5*
(b) (λfx.f(fx))(λy.+ y 3)4
(c) (λx.xx)(λy.yy)
(d) (λxy.xy)y

Answer.

(a) $(\lambda x.{*}\ x\ 4)5 \to_\beta ({*}\ x\ 4)[x := 5] = ({*}\ 5\ 4) = 20$

Has one β-normal form.

(b)

$$(\lambda fx.f(fx))(\lambda y.+\ y\ 3)\ 4 \to_\beta (\lambda x.f(fx))[f := (\lambda y.+\ y\ 3)]\ 4 = (\lambda x.(\lambda y.+\ y\ 3)((\lambda y.+\ y\ 3)x))\ 4$$
$$\to_\beta (\lambda y.+\ y\ 3)((\lambda y.+\ y\ 3)x))\ [x := 4] = (\lambda y.+\ y\ 3)((\lambda y.+\ y\ 3)\ 4)) \to_\beta (\lambda y.+\ y\ 3)((+\ y\ 3)[y := 4]))$$
$$= (\lambda y.+\ y\ 3)7 \to_\beta (+\ y\ 3)[y := 7] = 10$$

We can also do the reduction this way:

$$(\lambda fx.f(fx))(\lambda y.+\ y\ 3)\ 4 \to_\beta (\lambda x.f(fx))[f := (\lambda y.+\ y\ 3)]\ 4 = (\lambda x.(\lambda y.+\ y\ 3)((\lambda y.+\ y\ 3)x))\ 4$$
$$\to_\beta (\lambda x.(\lambda y.+\ y\ 3)((+\ y\ 3)[y := x]))\ 4 = (\lambda x.(\lambda y.+\ y\ 3)(+\ x\ 3))\ 4 \to_\beta (\lambda x.(+\ y\ 3)[y := (+\ x\ 3)])\ 4$$
$$= (\lambda x.(+\ (+x\ 3)\ 3))\ 4 \to_\beta (+\ (+x\ 3)\ 3)[x := 4] = (+(+4\ 3)\ 3) = 10$$

It can be shown that regardless of the order of reductions a lambda term has at most one β-normal form.

(c)

$$(\lambda x.xx)(\lambda y.yy) \to_\beta (xx)[x := (\lambda y.yy)] = (\lambda y.yy)(\lambda y.yy) \to_\beta (yy)[y := (\lambda y.yy)] = (\lambda y.yy)(\lambda y.yy) \to_\beta \cdots$$

This term cannot be reduced to a β-normal form.

(d) $(\lambda xy.xy)y \to_\beta (\lambda y.xy)[x := y] \to_\alpha (\lambda y'.xy')[x := y] = (\lambda y'.yy')$

The substitution must be capture-preventing and therefore in above solution, renaming (or α-reduction) is done as well.

---

7) (10 marks) [ref: A short introduction to the Lambda Calculus- by A. Jung] Let S be the term $\lambda xyz.(xz)(yz)$ and K the term $\lambda xy.x$.

(a) Reduce SKK to β-normal form. (Hint: To keep things manageable, keep the abbreviations S and K as long as you can; replace them with their corresponding lambda terms only when needed.)
(b) Find the set of free variables of SKK. Show steps. Is your answer in accordance with part (a)?

Answer.

(a)

$$SKK = (\lambda xyz.(xz)(yz))KK \to_\beta (\lambda yz.(xz)(yz))[x := K]K = (\lambda yz.(Kz)(yz))K$$

$$\to_\beta (\lambda z.(Kz)(yz))[y := K] = (\lambda z.(Kz)(Kz)) = (\lambda z.((\lambda xy.x)z)((\lambda xy.x)z))$$

$$\to_\beta (\lambda z.((\lambda y.x)[x := z])((\lambda y.x)[x := z])) = (\lambda z.(\lambda y.z)(\lambda y.z)) \to_\beta (\lambda z.(z)[y := (\lambda y.z)]) = \lambda z.z$$

Note we avoided capture by keeping the abbreviation K. We need to rename variables otherwise.

(b)

$$FV(SKK) = FV(S) \cup FV(K) \cup FV(K) = FV(S) \cup FV(K)$$
$$= \left(FV((xz)(yz)) - \{x, y, z\}\right) \cup \left(FV(x) - \{x, y\}\right)$$
$$= \left((FV(xz) \cup FV(yz)) - \{x, y, z\}\right) \cup \left(\{x\} - \{x, y\}\right)$$
$$= \left((\{x, z\} \cup \{y, z\}) - \{x, y, z\}\right) \cup \left(\{\}\right)$$
$$= \left(\{x, y, z\} - \{x, y, z\}\right) = \{\}$$

The set of free variables is empty and this conforms to the reduction in part (a) which does not have a free variable either.