York University- Department of Computer Science and Engineering

SC/CSE 3401 3.00 – Functional and Logic Programming

Solutions to assignment 3

Question 1. (2 marks) Write the following terms with as few parentheses as possible, without changing the meaning or structure of the term:

a) $(\lambda x.(\lambda y.(((xy)(zw))(xy)))) = \lambda xy.xy(zw)(xy)$

b) $((\lambda x.(y(\lambda y.(xy))))z) = (\lambda x.y\lambda y.xy)z$

Question 2. (6 marks) For the following terms,

(i) Restore all the dropped parentheses, without changing the meaning or structure.

(ii) Show the term calculation for each.

(iii) Find the set of free variables, showing all steps.

(a) $(\lambda x.xx)yz =$

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

$x, y, z, (xx), (\lambda x.(xx)), ((\lambda x.(xx))y), (((\lambda x.(xx))y)z)$

$FV((\lambda x.xx)yz) = FV(\lambda x.xx) \cup FV(y) \cup FV(z) = (FV(xx) - \{x\}) \cup \{y, z\} = \{y, z\}$

(b) $\lambda zw.\lambda x.x\lambda y.y(zw) =$

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

$x, y, z, w, (zw), (y(zw)), (\lambda y.(y(zw))), (x(\lambda y.(y(zw)))), (\lambda x.(x(\lambda y.(y(zw))))),$

$(\lambda w.(\lambda x.(x(\lambda y.(y(zw)))))), (\lambda z.(\lambda w.(\lambda x.(x(\lambda y.(y(zw)))))))$

$FV(\lambda zw.\lambda x.x\lambda y.y(zw)) = FV(x(\lambda y.(y(zw)))) - \{z, w, x\} = (FV(x) \cup FV(\lambda y.(y(zw)))) - \{z, w, x\}$

$= (\{x\} \cup (FV(y(zw)) - \{y\})) - \{z, w, x\} = \{x, z, w\} - \{z, w, x\} = \{\}$

Question 3. (4 marks) Apply the following renaming operations. Are the results α-equivalent to the following terms? Explain.

(a) $(x(\lambda xz.xz))\{x \setminus y\} = y(\lambda yz.yz))$ They are not α - equivalent, since x is free.

(b) $((\lambda x.xy)z)\{x \setminus y\} = ((\lambda y.yy)z)$ They are not α - equivalent, since y is not fresh.

Question 4. (6 marks) Convert the following terms to β normal form:

(a) $((\lambda x.(\lambda x.xy)z)y) = ((\lambda x.xy)z)[x := y] = ((\lambda x.xy)[x := y])(z[x := y]) = ((\lambda x.xy)z) = zy$

(b) $(\lambda x.(\lambda x.xy))(\lambda y.xy) = (\lambda x.xy)[x := (\lambda y.xy)] = \lambda x.xy$

(c) $(\lambda y.(\lambda x.xy))(\lambda y.xy) = (\lambda x.xy)[y := (\lambda y.xy)] = (\lambda x'.x'\,y)[y := (\lambda y.xy)] = \lambda x'.x'(\lambda y.xy)$

Question 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 $\lambda pq.qpq$ can implement AND in logic. Show all steps. (Hint: Show evaluation of AND for all four cases of the truth table).

$(T \wedge T) = (\lambda pq.qpq)TT \rightarrow_\beta (\lambda q.qTq)T \rightarrow_\beta TTT = (\lambda xy.x)TT \rightarrow_\beta (\lambda y.T)T \rightarrow_\beta T$

$(F \wedge T) = (\lambda pq.qpq)FT \rightarrow_\beta (\lambda q.qFq)T \rightarrow_\beta TFT = (\lambda xy.x)FT \rightarrow_\beta (\lambda y.F)T \rightarrow_\beta F$

$(T \wedge F) = (\lambda pq.qpq)TF \rightarrow_\beta (\lambda q.qTq)F \rightarrow_\beta FTF = (\lambda xy.y)TF \rightarrow_\beta (\lambda y.y)F \rightarrow_\beta F$

$(F \wedge F) = (\lambda pq.qpq)FF \rightarrow_\beta (\lambda q.qFq)F \rightarrow_\beta FFF = (\lambda xy.y)FF \rightarrow_\beta (\lambda y.y)F \rightarrow_\beta F$

Question 6. (10 marks) Write a function dist that finds the Euclidean distance between two points given their coordinates as lists. Your code must work for coordinates in any number of dimensions. If the lists of coordinates for the two points are not the same size, your code must return nil. You can assume that the lists contain numbers only.

Hint: You can define a second function to calculate squared distance.

Answer.

```
; Dist function can calculate the Euclidean distance between two points
; given their coordinates in any number of dimensions
; The coordinates for each point is given to DIST as a list

(defun dist (p1 p2)
(cond
      ; If any of the inputs is not a list, return nil
      ((or (not (listp p1)) (not (listp p2))) nil)

      ; If the two coordinate lists are not the same size, return nil
     ((not (eq (llist p1) (llist p2))) nil)

      ; Otherwise return square root of value returned by DIST2
     (T (sqrt (dist2 p1 p2)))
))
```

```
; DIST2 calculates the squared distance between two points
(defun dist2 (p1 p2)
(cond
     ; Stopping condition: if first list is empty, return 0
     ; Note we already checked in DIST for the lists to be of same size
      ((null p1)  0)

     ; Otherwise add squared distance of the head of the lists to return
     ; value of recursive call for the tails of the lists
      (T (+ (expt (- (car p1) (car p2)) 2) (dist2 (cdr p1) (cdr p2) )))
))

; LLIST return the length of a list
(defun llist (lst)
(cond
     ; Stopping condition: if empty list, length is zero
     ((null   lst)   0)

     ; Otherwise length is 1 + length of tail (recursive call)
     (t   (1+  (llist (cdr lst))))
))
```