Midterm Examination (Sample Solutions), Cmput 325

[15 marks] Consider the Lisp definitions below

```
(defun test (L S)
    (if (null L)
        S
        (let ((New (add (cdar L) S)))
           (test (cdr L) New))
     )
)

(defun add (A S)
     (if (null S)
         (cons (cons A 1) nil)
         (if (eq (caar S) A)
             (cons (cons A (+ 1 (cdar S))) (cdr S))
             (cons (car S) (add A (cdr S)))
          )
     )
)
```

For each of the following calls, show precisely what will be returned:

- `(add 'c nil)`
  answer: ((c . 1))
- `(add 'c '((a . 4) (b . 2) (c . 3)))`
  answer: ((a . 4) (b . 2) (c . 4))
- `(add 'd '((a . 4) (b . 2) (c . 3)))`
  answer: ((a . 4) (b . 2) (c . 3) (d . 1))
- `(test '((a . b) (a . c) (b . d)) nil)`
  answer: ((b . 1) (c . 1) (d . 1))
- `(test '((a . b) (a . c) (b . a) (b . b) (c . b)) nil)`
  answer: ((b . 3) (c . 1) (a . 1))

[6 marks] Show the machine level representation for each of the following S-expressions.

`(a (b) c)`

`Not drawn here`

`((a b) . (c d))`

`Not drawn here`

[4 marks] Show the simplest S-expression that is stored internally by the following structure:

```
---------    ---------    ---------               ---------
|   |   |-> |   |   |-> |   |   |-----------> |   |   | -> f
---------    ---------    ---------               ---------
  |            |            |                       |
  V            V            V                       V
  a            b          ---------                 e
                          |   |   | -> d
                          ---------
                            |
                            V
                            c
```

```
(a b (c . d) e . f)
```

[5 marks] The lambda expression for the successor function can be defined as follows:

$$SUC = (\lambda xsz \mid s(xsz))$$

Reduce the expression below to its normal form (this exercise intends to show that the successor of three is four).

$$\begin{aligned}&(\lambda xsz \mid s(xsz)) \;\; (\lambda sx \mid s(s(sx))) \\ &\to (\lambda sz \mid s((\lambda sx \mid s(s(sx)))\; sz)) \\ &\to (\lambda sz \mid s(s(s(sx))))\end{aligned}$$

[6 marks] Recall that boolean constants $true$ and $false$ are defined as: $T = (\lambda xy \mid x)$ and $F = (\lambda xy \mid y)$, respectively. Suppose that a boolean operator OP is defined as follows:

$$OP = (\lambda xy \mid (xTF)\ (yFT)\ x)$$

Draw a truth table for OP. That is, for every possible combination of boolean values X and Y, show the value of $(OP\ X\ Y)$.

```
X Y        OP X Y
-----------------
T T        F
T F        T
F T        F
F F        F
```

[2 marks] Suppose a lambda expression has a normal form. Is there any reduction strategy that guarantees the termination?

Answer: In this case, normal order reduction guarantees termination.

[10 marks] For each lambda expression below, evaluate it by the interpreter based on context and closure, and answer the corresponding questions. The initial context is assumed to be CT0. Clearly indicate your answer. You don't need to show how you get your answer.

(1) `(lambda (x) (g x)) ((lambda (x) (+ x x)) 3)`

Show the context when `(+ x x)` is being evaluated.
Answer: `{x -> 3} U CT0`

Show the context when `(g x)` is being evaluated.
Answer: `{x -> 6} U CT0`

(2) `((lambda (f z) (f 2 z)) (lambda (x y) (- y x)) 5)`

Show the context when `(f 2 z)` is being evaluated.
Answer: `{f -> [(lambda (x y) (- y x)), CT0], z -> 5} U CT0`

[17 marks] Define Lisp functions

```
(defun createPair (L)
    (if (null L)
        nil
        (cons (cons (car L) (cons (car L) nil)) (createPair (cdr L)))
    )
)
```

Write a *single* Lisp expression such that, given an L, the expression evaluates to the same result.

```
(mapcar '(lambda (x) (cons x (cons x nil))) L)
```

```
(defun member0 (A L)
    (cond ((null L) nil)
          ((not (atom L)) (or (member0 A (car L)) (member0 A (cdr L))))
          (t (eq A L))
    )
)
```

[15 marks]

```
; Define (insert A L) such that A is inserted into L in every possible
; position. The function returns a list of all these extended lists.
; E.g.  (insert 'a '(1 2 3)) ==> ((a 1 2 3) (1 a 2 3) (1 2 a 3) (1 2 3 a))

;-------------------------------------------------------------------
; Place A into L at the Nth position

(defun putAt (A L N)
   (if (eq N 1)
       (cons A L)
       (cons (car L) (putAt A (cdr L) (- N 1)))
   )
)

;calls insert0 with number of positions

(defun insert (A L)
    (let ((len (length L)))
       (insert0 A L (+ len 1))
    )
)

; Insert A into L at the Nth position, and recursively at N-1th position
; and so on.

(defun insert0 (A L N)
   (if (eq N 0)
       nil
       (cons (putAt A L N)
             (insert0 A L (- N 1)))
   )
)
```