CSE-3401A
Functional and Logic Programming
Summer 2008
Report 1 Specification
Due: Wednesday, June 4, in class

Shakil M. Khan

May 20, 2008

Be sure to read www page on "*On Reports*" from the home page for the course. Be sure the first page of the report you hand in is a standard *cover page* (see On Reports). Do not use functions described in chapters after the chapter mentioned in each exercise unless you are explicitly told to. For conditionals, **you must use** the function `cond`.

Question 1.

based on Chapter 15

A. Represent the following s-expressions in terms of cons cells and pointers (binary trees):

1. ((A).(((X).(nil.Y)).(Z)))
2. (((A (B) C)((D) E)) F)

B. Write the representations in Figure 1 and 2 in (1) fully dotted notation and (2) as Lisp would print them.

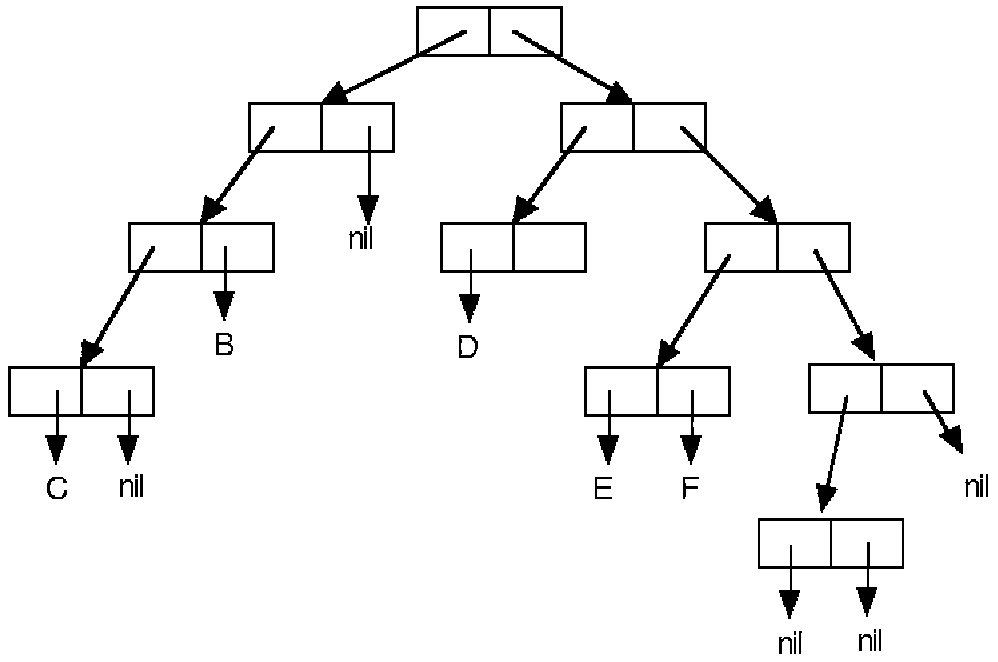

Figure 1:

Question 2.

based on Chapter 2

Evaluating **(caadadr '(a '(b (c))))** returns the value **b**. So does evaluating **(caadr (cadr '(a '(b (c))))**. If we evaluate the second part of this expression, i.e., **(cadr '(a '(b (c))))**, we get **'(b (c))**. But evaluating **(caadr '(b (c)))** returns **c**. Explain this.

Question 3.

based on Chapter 3

Consider the following functions, where *var1* is a free symbol:

1. (defun fun1 (a b) (+ a b (fun2 a b)))
2. (defun fun2 (b *var1*) (+ b (fun3 *var1*)))

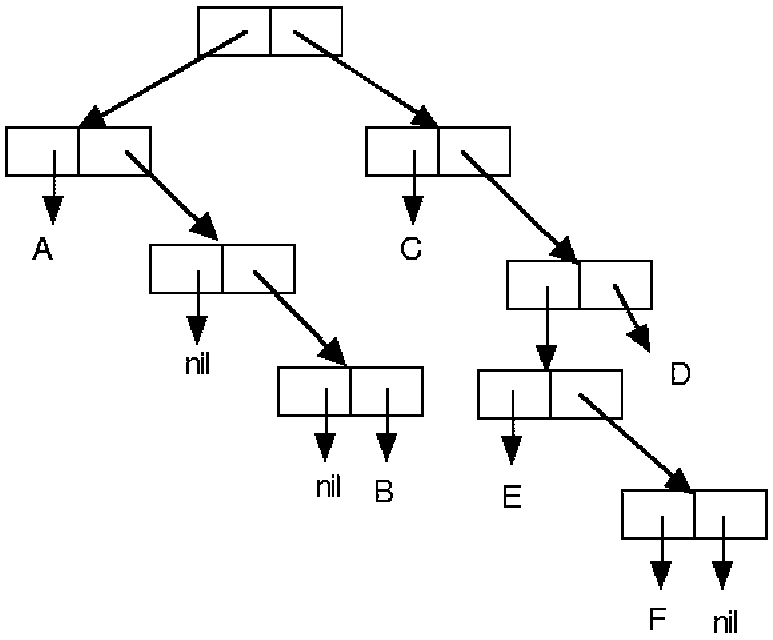

Figure 2:

3. (defun fun3 (c) (+ c *var1*))

Now consider the following code:

c1. (setq *var1* 2)
c2. (fun1 3 4)

What value does c2 returns when we are using a Lisp implementation with (1) static scoping, and (2) dynamic scoping. Explain by drawing the associated environments (see Lecture 2, slides on environments).

Question 4.

based on Chapter 6, 7, and 8

1. Write a function accepting an atom and an association list that returns the s-expression associated with the atom. Assume the atom appears in the association list.

2. Assume a *wff* (well-formed formula) is either:

   - a constant: **t** or **nil**;
   - a variable (denoted by any valid symbol) that is an atom;
   - **(null x1)**, **(and x1 x2)**, or **(or x1 x2)**, where x1 and x2 are wffs.

Write a function accepting a wff and an association list for the variables in the wff (each variable is either associated with t or nil) and determines whether the wff is true or not.

Question 5.

based on Chapter 8 and functionals-base.lsp and functional.lsp available on the course page

A two-dimensional matrix can be represented as a list of rows of the matrix where each row is a list of its column elements. For example, the following matrix with two rows and three columns

1 2 3
4 5 6

would be represented as ((1 2 3) (4 5 6)). The sum of the rows of the above yields the list (6 15). The sum of the columns of the matrix in part 2 yields the list (5 7 9).

Write functionals to do the following operations on two-dimensional matrices. They are examples of reduction from two to one dimension. Compare with the reduce operator that reduces from 1 dimension – a vector – to 0 dimensions – a scalar. In general, one can reduce a p-dimensional matrix along any of its p dimensions giving a p-1 dimensional matrix.

1. **(row-sum matrix)** – The result is the sum of each row of the matrix. The input is a PxQ matrix, while the output is a vector of length P (a Px1 matrix) – reduction along the second dimension. Use the reduce operator, do not rely on the +–reduction in Lisp.

2. **(column-sum matrix)** – The result is the sum of each column of the matrix. The input is a PxQ matrix, while the output is a vector of length Q (a 1xQ matrix) – reduction along the first dimension. Use the reduce operator, do not rely on the +–reduction in Lisp.

3. **(matrix-sum m1 m2)** – Input is two PxQ matrices. The result is a single PxQ matrix, where each element is the sum of the corresponding elements in the input matrices.

Question 6.

based on Chapter 8 and functionals-base.lsp and functional.lsp available on the course page

1. Define a functional **map2nd-level** that applies **f(x)** to every item in a list of lists.
   map-level2(f , ((1 2 3) (4 5) (6) ()))
   → ((f(1) f(2) f(3)) (f(4) f(5)) (f(6)) ())

2. The expression **(mapcar f (mapcar g list))** requires two traversals of the list. Define the functionals **map2op-v1(f g list)** and **map2op-v2(f-g-list)** that implement the expression with only one traversal of the list.

3. Define the functional **diagonals(sum)** that generates all pairs of natural numbers whose sum is at most **sum**. The pairs should be generated in the order shown by the example.
   Example: diagonals(3)
   → ((0 0) (0 1) (1 0) (0 2) (1 1) (2 0) (0 3) (1 2) (2 1) (3 0)).
   Notice that the result has all pairs that sum to 0, then all pairs that sum to 1, then 2, etc. You want to generate lists (use **genlist**) for each possible sum 0..n inclusive.