Midterm examination, CMPUT 325

Oct 24, 2002

Last name: _____

First name: _____

Instructions:
- Please **do not open** this exam until I give the starting sign.
- Then, first write your student ID number on the top of page 2.
- Keep your student ID ready for checking.
- In the end, stop immediately when the time is up.

Time: 75 minutes.

This exam has 6 questions and 8 pages including this cover page.
Total 80 marks for all questions: (+ 21 14 8 8 19 10)

You can use one prepared sheet with your notes, but no other forms of help.

This exam is printed on the right side pages only. You can use the space on the empty left pages for temporary notes.

Write your final answers in the *space directly after each question* (preferred), or make it VERY clear what the final answer for each question is, if you write it on the left side page.
On the exam pages (right pages), cross out everything that you wrote that should not be part of your answer.
You can get additional empty pages from me but you must return them with the exam.

For writing Lisp functions, use only those builtin functions that we discussed in class. The correctness and clarity of your code are both important.

Student ID _____

Question 1 (21 marks) _____
Question 2 (14 marks) _____
Question 3 (8 marks) _____
Question 4 (8 marks) _____
Question 5 (19 marks) _____
Question 6 (10 marks) _____

Total (80 marks) _____

1. Understanding Lisp code (21 marks total)

1.1 Consider the following Lisp function **f**:

```
(defun f (L)
(if (null L)
        0
        (cons  (car L)
                (cons (f (cdr L)) nil)
        )
))
```

1.1.1 (6x1 mark)
What is the result of evaluating the following expressions?
Use ‘pure’ Lisp, where it is an error to call the functions car or cdr for nil, or for something that is not a list.

1.1.1.1 (f nil) 0

1.1.1.2 (f 'a) error - should be a list

1.1.1.3 (f '(a)) (a 0)

1.1.1.4 (f '(a b c)) (a (b (c 0)))

1.1.1.5 (f '(((a) b) c)) (((a) b) (c 0))

1.1.1.6 (f (f '(a b))) (a ((b 0) 0))

1.2 (3x2 marks) In each of the following examples, give a LISP expression that returns the element *a* from the given list *L*.
Example: for *L = (a b)*, a correct answer would be *(car L).*

1.2.1 L = (x a b y)
(cadr L)

1.2.2 L = (((x)) ((a) b))
(caaadr L)

1.2.3 L = (x y (b a))
(cadaddr L)

1.3 (9 marks) Consider the following Lisp function **g**:

```
(defun g (X Y Z)
(cond
        ((null Y) nil)
        ((equal X (car Y)) (cons (car Z) (g X (cdr Y) (cdr Z))))
        (t (g X (cdr Y) (cdr Z)))
))
```

1.3.1 (3x2 marks) What is the result of evaluating the following expressions?

1.3.1.1 (g 'a '(a b c a a b) '(6 5 4 3 2 1)) (6 3 2)

1.3.1.2 (g '(a) '((a b) c a (a) b) '(6 5 4 3 2 1)) (3)

1.3.1.3 (g nil '(a b nil nil a ()) '(6 5 4 3 2 1)) (4 3 1)

1.3.2 (3x1 mark) What happens if in the definition of **g**, *equal* is changed to *eq* as follows?

```
(defun g (X Y Z)
(cond
        ((null Y) nil)
        ((eq X (car Y)) (cons (car Z) (g X (cdr Y) (cdr Z))))
        (t (g X (cdr Y) (cdr Z)))
))
```

Now, what is the result of evaluating these expressions?

1.3.2.1 (g 'a '(a b c a a b) '(6 5 4 3 2 1)) (6 3 2)

2. Writing Lisp code (14 marks total)

2.1 (6 marks)
Write a Lisp function **remove** as follows:
Given a list of atoms X, and a list Y, remove all elements of X from Y. In other words, return a list that contains only those elements inY that do *not* appear in X. The elements in the result should be in the same order as they are in Y. Elements of X that appear in nested sublists of Y should not be removed.

Examples: (remove '(a b c) '(d e a b a c e f)) → (d e e f)
(remove '(a b c) '(d e (a b) a (c) e f)) → (d e (a b) (c) e f)

```
(defun remove (X Y)
(if (null Y) nil
        (if (member (car Y) X) (remove X (cdr Y))
        (cons (car Y) (remove X (cdr Y)))
)))
```

This solutions recurses on the Y list, decides for each element whether it should e removed or not.
Another solution is to recurse on the X list, and use a helper function such as removeone the removes a single element from all of Y.

2.2 (8 marks)
Write a function **rr** (remove recursively) similar to **remove** above, which also removes elements of X from nested sublists of Y. If all elements of a sublist are removed, the result should contain an empty list in that place. See the examples below.

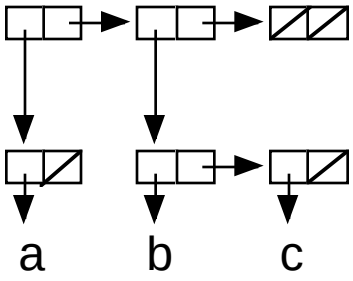
Examples: (rr '(a b c) ‘(d e a b a c e f)) → (d e e f)
(rr ‘(a b c) ‘(d e (a b) a (c) e f)) → (d e nil nil e f)
(rr ‘(a b c) ‘(d e (a (a d a b) (b)) a (c) e f)) → (d e ((d) nil) nil e f)

```
(defun rr (X Y)
(cond
        ((null Y) nil)
        ((member (car Y) X) (rr X (cdr Y)))
        ((atom (car Y)) (cons (car Y) (rr X (cdr Y))))
        (t (cons (rr X (car Y)) (rr X (cdr Y))))
))
```

Again, this recurses on Y, and could also be solved by recursion on X.
Doing the (atom (car Y)) test first, and then the member test inside, would be more

3. S-expressions, Machine Representation (8 marks total)

3.1 (2x2 marks) For the given diagram showing a machine representation, write the corresponding S-expression both in full dotted-pair form and in the simplest form.
Remark: in the diagram, nil is represented by a crossed-out box ⊠ .



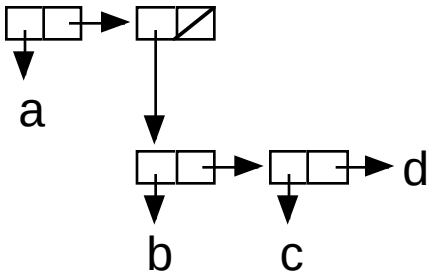
3.1.1 full dotted-pair form:

((a.nil).((b.(c.nil)).(nil.nil)))

3.1.2 simplest form:

((a) (b c) nil)

3.2 (2 marks)
Draw the diagram showing the machine representation of the S-expression (a (b.(c.d)))



3.3 (2 marks) Which of the s-expressions a) to d) below are equivalent?

a) (w (x.y).z)

b) (w.((x.y).z))

4. Higher-order functions in Lisp (8 marks total)
Solve the following questions by using the higher-order functions *mapcar* and *reduce*.

You are given a list L with employee information. The information for each employee is a list of three elements of the form *(name hours rate)*, where *name* is the name of the employee, *hours* is the number of hours that the employee worked, and *rate* is the salary per hour.

4.1 (4 marks) Use *mapcar* to write a Lisp function **salary** that computes a list of pairs (name payment), where each payment is computed by multiplying *hours* and *rate* for that person. Employees should appear in the same order in the output as in the input.

Example:
(salary '((lee 25 10) (sam 40 8) (ann 10 20)))
→ ((lee 250) (sam 320) (ann 200))

(defun f (T) (list (car T) (* (cadr T) (caddr T))))
(defun salary (L) (mapcar 'f L))

4.2 (4 marks) Use *reduce* to write a Lisp function **totalpay** that computes the sum of all payments for the employer for a list of employees. Use the form of reduce without an identity element. You can assume that the employee list is not empty.

Example:
(totalpay '((lee 25 10) (sam 40 8) (ann 10 20)))
→ 770

(defun totalpay (L) (reduce '+ (mapcar 'g L)))
(defun g (Tr) (* (cadr Tr) (caddr Tr)))

5. Lambda Calculus (19 marks total)

5.1 (2x3 marks)

Let functions **f** and **g** be defined by f(x,y) = 3*x + y, g(x,y) = x + y.

Reduce the following expression and show *each* reduction step for f and g:

g(f(2, f(1,1)),1)

5.1.1 use normal order reduction:

g(f(2, f(1,1)),1) = f(2, f(1,1)) + 1 = 3*2 + f(1,1) + 1 = 6+3*1+1+1 = 11

5.1.2 use applicative order reduction:

g(f(2, f(1,1)),1) = g(f(2, 4),1) = g(10,1) = 11

5.2 (2x3 marks)

Given the following lambda expressions a) to d):

a) ((λy | y+1) ((λx | x+1) 5))

b) ((λx | x+2) 5)

c) ((λy | y+1) 6)

d) ((λx | x+1) ((λx | x+1) 5))

Which of the expressions a) to d) can be reduced to other expressions among a) to d)? List all possible reductions. For example, an answer e→f, g→h, h→g would mean that there are reductions from e) to f), from g) to h), and from h) to g), and no other reductions.

5.2.1 (3 marks) List all possible alpha reductions between a), b), c) and d).

a) to d), d) to a)

5.2.2 (3 marks) List all possible beta reductions between a), b), c) and d). Also include thosebeta reductions that can be achieved by doing an alpha reduction as well.

a) to c), d) to c)

5.3 (2x2 marks)

Evaluate the expression below by using context and closure. Assume that evaluation takes place in an initial context CT0.

5.4 (3 marks) State the Church-Rosser theorem.

See lecture notes.

6 SECD machine (10 marks total)

6.1 (5 marks)
Show the execution of the following code on a SECD machine. Show the state of the stacks after each operation.

start state: s e c d, with c = (LDC 2 LDC 3 + LDC 2 LDC 6 + *)

(2.s) e (LDC 3 + LDC 2 LDC 6 + *) d

(3 2.s) e (+ LDC 2 LDC 6 + *) d

(5.s) e (LDC 2 LDC 6 + *) d

(2 5.s) e (LDC 6 + *) d

(6 2 5.s) e (+ *) d

(8 5.s) e (*) d

(40.s) e nil d

6.2 (5 marks)
Consider extending the SECD machine with an operation *GZ* that tests whether the top of the s stack is a number greater than zero. *GZ* replaces the top of the stack by *T* if the test outcome is true andby *F* otherwise. *GZ* can be described as follows:

(x.s) e (GZ.c) d $\rightarrow$ (r.s) e c d
where r = T if x>0, and r=F if x≤0.

Use the *GZ* command to compile the following Lisp code into machine code. Remarks: The SEL command can be used to test whether T or F is on top of the s stack. Assume that a binary function '-' for subtraction is available: it subtracts the second element on the stack from the top element and puts the result on the top of the stack.

Lisp code: (if (< 3 5) 10 20)

(LDC 3 LDC 5 - GZ SEL (LDC 10 JOIN) (LDC 20 JOIN))