Midterm Examination, Cmput 325
Feb 15, 2001

LAST NAME _____

FIRST NAME _____

INSTRUCTIONS:

This is a closed book exam. The time for this exam is 80 minutes, and the mark total is also 80. There are 8 pages in this exam.

When writing a Lisp program, You may only use the builtin functions that have been allowed in the first two assignments. The correctness and the clarity of your code are both important. You don't need to write comments in this exam in general. But if you decompose a function, briefly comment on what each subfunction does.

Good luck!

Reminder: In the week right after the reading week, a demo on how to use Prolog will be given in the lab by a TA. Don't miss it!

STUDENT ID _____

QUESTION 1 (15 marks): _____

QUESTION 2 (20 marks): _____

QUESTION 3 (10 marks): _____

QUESTION 4 (20 marks): _____

QUESTION 5 (15 marks): _____

TOTAL:_____ OUT OT 80

1. (15 marks)

[4 marks] For the following pair of λ-expressions, indicate whether they can be reduced to a common expression. Show all of the reduction steps that lead to your conclusion. Indicate a α-reduction by $\rightarrow_\alpha$. Any reduction step without a subscript is considered a β-reduction. As a reminder of notation, for (sub)expressions N,M and Q, the expression NMQ means $((N\ M)\ Q)$ which is different from $(N\ (M\ Q))$.

$(\lambda x \mid x)\ \ (\lambda y \mid (\lambda x \mid xx)yw)$ $\qquad\qquad$ $(\lambda z \mid (\lambda x \mid z(xx)))\ (\lambda z \mid zw)$

[3 marks] Consider reducing the expression

$$(\lambda y \mid (\lambda z \mid yz))\ (\lambda x \mid xyz)$$

First, indicate any occurrence of a variable that is free. If we reduce it by direct substitution, we end up with $(\lambda z \mid (\lambda x \mid xyz)z)$ which reduces to $(\lambda z \mid zyz)$. This is incorrect. Your second task is to show a correct sequence of reductions leading to a normal form.

[4 marks] What is the Church-Rosser Theorem? That is, briefly, what does it say, and why it is useful?

[4 marks] Reduce the following λ-expression to a normal form twice, one by normal order reduction and the other by applicative order reduction

$$(\lambda x \mid x) \ \ ((\lambda x \mid xx)(\lambda y \mid y))$$

Normal Order Reduction:

Applicative Order Reduction:

2. (20 marks)

[4 marks] For each of the following s-expressions, show the Lisp code that generates it from the atoms appearing in it. E.g. `(a b)` is generated by `(cons 'a (cons 'b nil))`.

```
((a) (b))
```

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

[3 marks] Consider the following Lisp definition.

```
(defun f (L)
        (If (null L)
            nil
            (if (< (car L) 4)
                (f (cdr L))
                (cons (+ (car L) 1) (f (cdr L)))
            )
        )
    )
```

Show the result of evaluating the expression: `(f '(3 6 9 2 4 5))`

`your answer:`

[5 marks] Consider the following Lisp definition.

```
(defun r (E L)
      (cond ((null L) nil)
            ((null (cdr L)) L)
            ((equal E (car L)) (r E (cdr L)))
            (t (cons (car L) (r E (cddr L))))
      )
)
```

For each call below, what will be returned?

```
(r 'a '(b a d e a q))
```

your answer:

```
(r '(a b) '((a b) d e (a b)))
```

your answer:

[3 marks] What will be returned when the following expression is evaluated?

```
(mapcar
      '(lambda (x) (cons x (cons (* x x) nil)))
      '(1 2 3 4)
)
```

your answer:

[5 marks] Given a list of numbers, e.g. `(4 7 3)`, we want to get the sum of these numbers, each increased by one. For the above example, we get the sum of `(5 8 4)`, which is 17.

Write a Lisp expression that does this for a given list (the given list is part of your expression, just like the expression in the preceding question). That is, when the expression is evaluated, the sum of all the numbers, each increased by one, is returned.

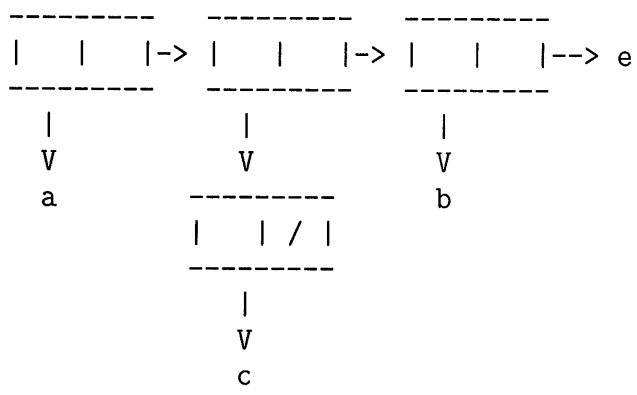
Hint: My solution uses *reduce*, *mapcar*, *lambda*, +, and *quote*. As a reminder, the syntax of *reduce* is `(reduce OP L)` where `OP` is a function/operator and `L` is a list.

3. (10 marks) Machine level representation.

[5 marks] Show the machine level representation of the following s-expression.

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

[5 marks] Show two different expressions that are represented internally by the following structure. One of the two expressions must be the the simplest.



4. (20 marks)

In this problem, we will use lists to represent sets and define some set operators. An element of a set may not necessarily be an atom; it may be represented by a list, too. For example, `(a b (a))` represents a set with three elements where `a` and `(a)` represent two different elements. Any list representing a set must not have repeated elements. You may assume that no given list has repeated elements.

For the three set operators described below, you only need to work out TWO for the full marks. If you have solutions for all three, I will just choose any two to mark, unless you indicate which two I should mark.

Define a function

```
(defun setIntersect (S1 S2) ...)
```

which returns the intersection of two sets `S1` and `S2`.

Define a function

```
(defun setUnion (S1 S2) ...)
```

which returns the union of two sets `S1` and `S2`.

Define a function

```
(defun setEqual (S1 S2) ...)
```

which returns true if S1 and S2 represent the same set, false otherwise.

5 (15 marks) Interpreters.

[5 marks] Consider the interpreter that we implemented in Assignment 2. Supppse we have the following program where we use setq to bind it to the atom P.

```
(setq P '(  ((sum X) = (if (eq X 0)
                           0
                           (+ X (sum (- X 1)))
                       )
             )
          )
)
```

Show all of the reduction steps in evaluating expression `(sum 5)` in the following call

```
(interp '(sum 5) P)
```

You need not write out an expression involving the if function; it can be reduced right away. For example, if a reduction step leads to an expression `(if (null nil) Exp1 Exp2)`, you can reduce this immediately and write out the expression `Exp1`.

your answer:

```
      (sum 5)
==>

==>

==>

...
```

In the remainder of this exam, we consider the context-based interpreter. We will use $\{x_1 \rightarrow v_1, ..., x_m \rightarrow v_m\}$ to denote a context, and $[Fn, CT]$ a closure where Fn is a lambda function and CT a context. We assume that the initial context is CT0.

[2 marks] Consider the following lambda expression.

```
((lambda (x y) (+ y (* x x))) 3 4)
```

Show the context when the subexpression `(+ y (* x x))` is being evaluated.

[8 marks] Consider evaluating the following lambda expression.

```
(((lambda (z) (lambda (x) (z x))) (lambda (x) (* x x)))  4)
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^    ^^
              function                                      argument
```

Be very careful when reading this expression. To help you parse the expression, we note that the function part itself is an application, which is depicted further by

```
((lambda (z) (lambda (x) (z x)))   (lambda (x) (* x x)))
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^   ^^^^^^^^^^^^^^^^^^^^
          function                 argument
```

(a) Show the result of evaluating the whole expression. (Result only, no need to show how you get it.)

(b) Show the context when the subexpression `(z x)` is evaluated.

(c) Show the context when the subexpression `(* x x)` is evaluated.