

Computer Science 51 Final Exam

Spring 2009

Instructions: There are 4 pages and 8 problems in this exam. Please read over the entire document before starting. Clearly write your answers in the books provided. Make sure to write your name on all of your exam books. Good luck!

1. [24 points] For each of the following questions, define `greg` so that the given expression will evaluate to 42. If it is not possible to do so, explain why.

a. `(greg (greg))`

b. `(let* ([n 41]
 [m (greg)])
 n)`

c. `(let* ([x (list 3 (list 42))]
 [m (greg x)])
 (car (car (cdr x))))`

d. `(let* ([x (mcons 3 (mcons 10 empty))]
 [z (let ([x (mcdr x)]
 (greg x))])
 (mcar (mcdr x))`

e. `(let ([greg (lambda (x) (+ 1 x))])
 (greg 0))`

f. `(let* ([x (list 20 10 10 2)]
 [n 0]
 [add-n (lambda (m) (set! n (+ n m))]])
 (begin
 (map (greg add-n) x)
 n))`

2. [10 points] The cosine of x can be computed using the following infinite series:

$$\cos(x) = 1 - (x^2/2!) + (x^4/4!) - (x^6/6!) + (x^8/8!) \dots$$

a. Write a function `cos-terms` which when given x (with $-1 < x < 1$), generates the infinite stream (i.e., lazy-list) of terms that when added, yields the cosine of x . You can assume `lcons`, `lcar`, and `lcdr` have been defined, but do not assume other lazy list operations are provided.

b. Using your `cos-terms` function, write a function `cos-approx` which when given x (with $-1 < x < 1$) and given a decimal fraction `epsilon` (such as 0.0001), returns a value that is within `epsilon` of the actual value of cosine x . That is, your function should have the property that:

$$\cos(x) - \text{epsilon} \leq (\text{cos-approx } x \text{ epsilon}) \leq \cos(x) + \text{epsilon}.$$

3. [16 points] Recall the definitions of `map` and `filter` for lists:

```
(define (map f x)
  (if (empty? x) empty
      (cons (f (car x)) (map f (cdr x)))))

(define (filter p? x)
  (cond [(empty? x) empty]
        [(p? (car x)) (cons (car x) (filter p? (cdr x)))]
        [else (filter p? (cdr x))]))
```

a. Formally prove that for purely functional Scheme (i.e., no `set!`, `set-mcar!`, `set-mcdr!`, etc.) the following equation holds:

$$(\text{filter } p? (\text{map } f \ x)) = (\text{map } f (\text{filter } p? \ x))$$

b. Why might a compiler want to take advantage of this equation?

c. Show, by giving a counter-example, that for full Scheme, which includes side effects, the equation is *not* valid.

4. [12 points] For each term, write one or two sentences that explain what the term means in the context of object-oriented programming:

- a. sub-typing
 - b. classes
 - c. inheritance
-

5. [9 points] Your friend Joe Scheme says that he has written a procedure `super-eq`, which takes strings that encode two Scheme functions `f` and `g` as arguments, and returns a boolean. Joe claims that `super-eq` returns true if for all numbers `n`,

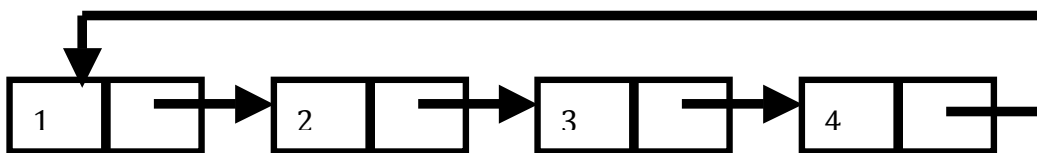
- I. $(f\ n)$ and $(g\ n)$ return the same number,
- II. $(f\ n)$ and $(g\ n)$ both run forever, or
- III. $(f\ n)$ and $(g\ n)$ both halt with a type error

Otherwise `super-eq` returns false. In short, `super-eq` is able to tell whether two Scheme functions compute the same result, given the same input.

Argue (formally) that Joe is an idiot.

6. [8 points] Write a tail-recursive version of `append` for lists. You may also define and use auxiliary functions, as long as they are also tail-recursive.

7. [8 points] Write a function `make-cycle` which takes a number `n` ($n \geq 1$) and produces a (mutable), circular list of length `n` with the values `1,2,3,...,n` as the elements of the list. For example, `(make-cycle 4)` should produce a list that looks like this:



8. [9 points] Suppose we have some paired data (x_i, y_i) and are interested in fitting it with one or more lines. The following are some properties the data might have. For each case, select which of the following algorithms would be best suited for the given task: least squares, total least squares, least median squares, segmented least squares.

- a. The data have no outliers, and the x values and y values are symmetric (for example, height and weight).
- b. The data have 45% bad values, which tend to "cluster" in a particular region.
- c. You require an exact closed-form solution.
- d. The data consists of a series of lines, one after another, but you do not know where one line begins and the other ends.

9. [5 points] Suppose we give you a system of constraints over a set of variables x_1, x_2, \dots, x_n where the constraints are of the form $x_i - x_j \leq c_k$ and c_k is a non-negative constant. For example, here is a sample set of constraints:

$$\begin{aligned}x_1 - x_2 &\leq 3 \\x_2 - x_4 &\leq 4 \\x_3 - x_1 &\leq 2 \\x_3 - x_4 &\leq 11\end{aligned}$$

A set of constraints is *consistent* if there is some substitution of non-negative integers for the variables that satisfies all of the inequations. For example, if we pick $x_1 = 7$, $x_2 = 4$, $x_3 = 11$, and $x_4 = 0$ then this substitution satisfies the inequations.

Describe an efficient algorithm for finding a solution to a consistent set of constraints.