Midterm Exam
CS6848

Maximum marks = 40, Time: 2hrs

16-Mar-2014

Read all the instructions and questions carefully. You can make any reasonable assumptions that you think are necessary; but state them clearly. There are total four questions totaling 40 marks + 5 marks (bonus). Each (non-bonus) five marks will approximately take 15 minutes. For questions that have sub-parts, the division for the sub-parts are mentioned in square brackets.

Leave the first page empty. Start each question on a new page. Think about the question before you start writing and write briefly. **The answer for any question (including all the sub-parts) should NOT cross more than two pages.** If the answer is spanning more than two pages, we will ignore the spill-over text. If you scratch/cross some part of the answer, you can get compensation space from the next page.

1. [10] **Scheme Programming**

   (a) [2] Write a program in Scheme to implement the `filter` function. It takes two arguments, a predicate and a list of elements and returns a sublist of elements from the given list, satisfying the given predicate:
   `(filter even? '(1 2 3 4 6))` returns `(2 4 6)`
   `(filter null? '((1 2) ()))` returns `(())`

   (b) [2] Write a function `mergeLists` to merge two given sorted integer lists to return a merged sorted list.
   `mergeList '(3 4 9 10) '(1 2 4 6)` returns `(1 2 3 4 6 9 10)`

   (c) [3] Given a list of integers, write a function to return a sorted list.

   (d) [3] Given a matrix represented as a list of lists, write a function to transpose it.

2. [10] **Interpreter** Write an interpreter for the subset of scheme that admits `let`, `lambda`, *application*. Assume, call by name semantics. The following code should evaluates to 9:

```
(let ((x (lambda (y z) y))
      (id (lambda (x) x))
      (double (lambda (x) (x x)))
      (z 9))
      (x (id z) (double double)))
```

3. [10] **Type rules and small step semantics** Write the type rules and small step semantics for the following subset of scheme that allows list operations: cons, car, cdr, and isnull. Assume that 1) all the elements in a list will be of the same type, 2) `isnull` is defined only on a list.

```
e :=  car e | cdr e | cons e1 e2 | isnull e
            | if (e0) e1 e2 | '() | false | true
```

Bonus [5] **Type soundness** Prove the type soundness theorem (well typed programs cannot go wrong). – *Attempt at the end.*
Definitions.

- An expression e is *stuck* if it is not a value and there is no expression e' such that $e \rightarrow_V e'$.
- An expression e *goes wrong* if $\exists e' : e \rightarrow_V^* e'$ and e' is stuck.
- An expression is *well typed* iff there exists a type t such that $\vdash e : t$.

4. **CPS** [10] Translate the following scheme code to scheme code in imperative-form in a step by step manner.

```
(define run-sudan
  (letrec
   ((f (lambda (n x y)
     (cond
      ((= n 0) (+ x y))
      ((= y 0) x)
      (else (f (- n 1) (f n x (- y 1)) (+ y (f n x (- y 1))))))))))
    (f 2 1 2)))
```