CS 312 Data Structures and Functional Programming
Fall 2005

Prelim I Solutions

10/13/2005

NAME:____

CU ID:______________________________ Net ID:______________________________

Section instructor______________________________

You have one and a half hours to do this exam.
All programs in this exam must be written in SML.

SOLUTIONS

1. (Types, 10 points)

- `fun f x = x` __(a)__
- `fun f x y = let val x = y in SOME (x*y) end` ___ (i)____
- `fun f x y = if x then NONE else SOME (not x)` ___(c)____

-
  ```
  fun f(x,y) =
     case y of nil => x | u::t => (SOME u)::f(x,t)
  ```
  ___(g)____

-
  ```
  fun foo f a l =
     case l of nil => a | u::t => foo f (f(u,a)) t
  ```
  ___(h)____
- `fun K x y = x` ____(d)___
- `fun f (ref a) = a` ___(e)____

-
  ```
  fun wrap f = let val counter = ref 0          ___(b)_____
               in (fn x => (counter := !counter + 1; f x),
                   fn () => !counter
                   )
               end
  ```
- `fun f a b = (a := !a +1; !b)` ___(f)____

(a) `'a -> 'a`

(b) `('a -> 'b) -> ('a -> b)*(unit -> int)`

(c) `bool -> 'a -> bool option`

(d) `'a -> 'b -> 'a`

(e) `'a ref -> 'a`

(f) `int ref -> 'a ref -> 'a`

(g) `'a option list * 'a list -> 'a option list`

(h) `('a*'b ->'b) -> 'b -> 'a list ->'b`

(i) `'a -> int -> int option`

2. (Tree reduce, 15 points)

   In class, we defined a reduction operator as a commutative and associative function, and showed how to write a function that applies a reduction operator to the elements of a list. Write a curried function that takes a reduction operator, an initial value and a binary tree as input, and applies the reduction operator to the initial value and the elements of the binary tree. Your code must specify the type of this function. You may assume that the binary tree is built from the following datatype.

```
datatype 'a binaryTree =
      LEAF of 'a
    | NODE of 'a binaryTree * 'a binaryTree * 'a
```

   For example, `reduce (op+) 4 (NODE(LEAF(1),LEAF(2),3))` should evaluate to the value `10`.

   Solution:

```
fun reduce (f:'a*'b->'b) (z:'b) (t:'a tree) =
    case t of
       (LEAF(n))      => f (n, z)
      |(NODE(l,r,d)) => reduce f (f (d, reduce f z l)) r
```

   Alternative solution:

```
fun reduce f z t =
 let fun red1 t =
       case t of
       (LEAF(n)) => n
      |(NODE(l,r,d)) => f(d,f(red1 r,red1 l))
 in f(z,red1 t)
 end
```

3. (Properties of foldl, 20 points)

   Consider the following assertion about the `foldl` function.

   `foldl f f(x,a) l = foldl f a (x::l)`

   Answer the following questions.

   (a) Is the statement true for any function `f` (assume the type of the function is appropriate)?

   (b) If the statement is true, prove it. If not, give a counter-example.

   (c) Is your proof an induction? If so, explain clearly what the inductive hypothesis and inductive step are. If not, explain why your proof is not an induction.

   For your convenience, the definition of `foldl` is shown below.

```
fun foldl f a nil = a
  | foldl f a (h::t) = foldl  f  (f(h,a)) t
```

   Solution:

   (a) Statement is true.

   (b) There are two cases.

   Case 1: empty list.

   `foldl f f(x,a) [] = f(x,a)` (by definition of foldl)

   `foldl f a (x::[]) = foldl f f(x,a) [] = f(x,a)` (by def of foldl)

   Therefore, Case 1 is proved.

   Case 2: non-empty list (h::t)

   `foldl f f(x,a) (h::t)`

   `  = foldl f f(h,f(x,a)) t` (by definition of foldl)

   `foldl f a (x::h::l) = foldl f f(x,a) (h::l)`

   `  = foldl f f(h,f(x,a)) l` (by definition of foldl)

   Therefore case 2 is proved.

   Conclusion: theorem is true for all lists.

   (c) The proof looks inductive but it is not because the proof of Case 2 did not use any assumptions about other (smaller) lists.

4. (Induction on integers, 20 points)

   Use induction to prove the following results. Your answers must state clearly (i) the base case or cases, (ii) the inductive hypothesis, (iii) the inductive step, and (iv) the conclusion.

   (a) $(1-\frac{1}{4})(1-\frac{1}{9})\dots(1-\frac{1}{n^2}) = \frac{n+1}{2n}$ for $n \geq 2$

   (b) Show that

   $$\frac{m!}{0!} + \frac{(m+1)!}{1!} + ... + \frac{(m+n)!}{n!} = \frac{(m+n+1)!}{n!(m+1)}$$

   where m and n are non-negative integers. Hint: you can do an induction on either m or n, but the induction on n is easier.

   (a) (10 points)

   - (2 points) Base case: for n = 2, (1 - 1/4) = 3/4 = (2+1)/2*2 = 3/4.
   - (2 points) Inductive hypothesis: assume the result is true for some integer k $\geq$ 2.
   - (6 points) Inductive step:
     $(1-\frac{1}{4})(1-\frac{1}{9})...(1-\frac{1}{(k+1)^2})$
     $=$
     $(1-\frac{1}{4})(1-\frac{1}{9})...(1-\frac{1}{k^2})(1-\frac{1}{(k+1)^2})$
     $=$
     $\frac{(k+1)}{2k} * (1-\frac{1}{(k+1)^2})$
     $=$
     $\frac{(k+2)}{2(k+1)}$
     as desired.
   - Conclusion:
     $(1-\frac{1}{4})(1-\frac{1}{9})...(1-\frac{1}{n^2}) = \frac{n+1}{2n}$ for $n \geq 2$

   (b) (10 points)

   - (2 points) Base case:
     $\frac{m!}{0!} = \frac{(m+1)!}{0!(m+1)} = \frac{m!}{0!}$
   - (2 points) Inductive hypothesis: for some integer $k \geq 0$
     $\frac{m!}{0!} + \frac{(m+1)!}{1!} + ... + \frac{(m+k)!}{k!} = \frac{(m+k+1)!}{k!(m+1)}$

- (6 points) Inductive step:
  $\frac{m!}{0!} + \frac{(m+1)!}{1!} + ... + \frac{(m+k)!}{k!} + \frac{(m+k+1)!}{k+1!}$
  $= \frac{(m+k+1)!}{k!(m+1)} + \frac{(m+k+1)!}{(k+1)!}$
  $= \frac{(k+1)(m+k+1)!+(m+1)(m+k+1)!}{(k+1)k!(m+1)}$
  $= \frac{(k+1+m+1)(m+k+1)!}{(k+1)!(m+1)}$
  $= \frac{(m+k+2)!}{(k+1)!(m+1)}$
- Conclusion:
  $\frac{m!}{0!} + \frac{(m+1)!}{1!} + ... + \frac{(m+n)!}{n!} = \frac{(m+n+1)!}{n!(m+1)}$ for all $m, n \geq 0$.

5. (Using foldl, 20 points)

   For each of the following functions, show how to use foldl to re-implement the function in one or two lines, without using `case`, `hd`, `tl` or `nth`. Your answer must specify the type signature of the function used to perform the fold.

   - Given a list of integers, find the sum of all elements after the first 0 in the list. Here is a function that computes this without using foldl.

```
fun afterZero l =
   let
      fun summer l =
        case l of
            nil => 0
             |h::t => h + summer t
   in
     case l of
        nil  => 0
            |0::t => summer t
            |h::t => afterZero t
   end


fun afterZero (l:int list): int =
  #2 (foldl (fn (e: int,(b: bool,sum: int)) =>
                if b orelse (e = 0) then (true,sum+e)
                else (false,sum))
            (false,0) l)
```

- Given a list `l`, return a list containing every $\mathtt{n}^{th}$ element of the list. Here is a function that computes this without using foldl. You may use List.rev in your solution.

```
fun everynth(l,n) =
  let
    fun iter(l,counter) =
        case (l,counter) of
             ([],_) => []
             |(x::xs,1) => x::(iter(xs,n))
             |(x::xs,c) => iter(xs,c-1)
  in iter(l,n)
  end


fun everyNth(l: 'a list, n:int):'a list =
  List.rev (#2(foldl (fn (elt: 'a, (c: int, lst: 'a lsit)) =>
                         if (c = 1) then (n, elt::lst)
                         else (c - 1, lst))
                     (n, []) l))
```

6. (Bytecode interpreter, 15 points)

   In this problem, you have to implement a bytecode interpreter for an abstract stack machine called SaMueL. SaMueL operates only on integers and it has the following instructions:

   - PUSHIMM integer : the integer value is pushed on the stack
   - ADD: pop two values from the stack, add them, and push the result on the stack. Raise an exception tooFewOperands if there are fewer than two operands on the stack.
   - STOP : pop the topmost value on the stack and return it as the result of evaluating the program. Raise tooFewOperands if the stack is empty.

   Instructions are executed sequentially until the STOP instruction is encountered. For example, the program

```
PUSHIMM 2
PUSHIMM 3
ADD
STOP
```

   should return the value 5.

   Fill in the rest of the shell shown below to produce an interpreter for SaMueL. The function SaMueL will be called with a list of instructions and an empty stack. If the interpreter runs out of instructions before encountering a STOP instruction, it should raise the tooFewInstructions exception.

```
Solution:

datatype bytecode = PUSHIMM of int | ADD| STOP

type code = bytecode list

type evalStack = int list

exception tooFewOperands and tooFewInstructions

fun SaMueL (program:code) (st:evalStack) =
   let
     (*like hd but raises tooFewOperands exception if list is nil *)
     fun chkHd l = if (l = nil)
                   then raise tooFewOperands
                   else hd(l)
   in
     case program of
      nil => raise tooFewInstructions
    |(h::t) => case h of
                 STOP => chkHd(st)
                |PUSHIMM i => SaMueL t (i::st)
                |ADD => SaMueL t ((chkHd(st)+chkHd(tl(st)))::tl(tl(st)))
   end
```

Alternative solution

```
fun SaMueL (program:code) (st:evalStack) =
   case (program, st) of
       ([],_)             => raise tooFewInstructions
     | (STOP::t, e::_)    => e
     | (PUSHIMM(i)::t, _) => SaMueL t (i::st)
     | (ADD::t, a::b::s)  => SaMueL t ((a+b)::s)
     | _                  => raise tooFewOperands
```