COMP 302: Programming Languages and Paradigms

Winter 2015: Assignment 4

This assignment is due on Wednesday, March 25 at midnight. It must be submitted using MyCourses**.** The cutoff time is automated. Assignments submitted within the next hour will be accepted but marked late. The system will not accept any attempts to submit after that time.

Problem 1: (30 marks)

We can define a reference based linked list using the following datatype.

```
datatype 'a rlist = Empty | RCons of 'a * (('a rlist) ref)
```

Implement an SML function `insert` which inserts a value into a sorted linked list. The type of insert should be:

```
 insert: ('a * 'a -> bool) * 'a * ('a list) ref -> unit
```

That is, insert takes three arguments: A comparison function of type `('a * 'a -> bool)`, an element of type `'a`, and a linked list `lst` of type `('a list) ref`. Your function should destructively update the list `lst`

Problem 2: (30 marks)

We defined a function make_account to generate a bank account using the following datatype.

```
datatype transactions =
         Withdraw of int | Deposit of int | Check_balance
```

Modify this code to implement a password protected bank account. Any transaction would only be possible if one provides the correct password.

To do this implement a function `make_protected_account` of type

```
   int * string -> (transaction * string -> int)
```

This function take two arguments, an opening balance and a password. It returns a function which, when given a transaction and the correct password, will perform the transaction. If the password is incorrect, the function will return:

```
        (print ("Wrong password.\n"); 0)
```

Problem 3 (15 marks)

Draw an environment diagram for the following expression and explain what the final result of its evaluation will be and how you obtained it.

```
let
  val y = 1
  val x = 2
in
  let
    val f = let val x = y
            in fn u => (u + x + y) end
  in
    let val y = 4
    in f(y) end
  end
end
```

Problem 4 (25 marks)
Consider the following two programs which reduce the elements of a list.

```
(* reduce: 'a list * 'a * ('a * 'a -> 'a) -> 'a *)
fun reduce (nil, base, op) = base
  | reduce (h::t, base, op) = op (h, reduce(t, base, op)

(* reduce_tr: 'a list * 'a * ('a * 'a -> 'a) -> 'a *)
fun reduce_tr (nil, base, op) = base
  | reduce_tr (h::t, base, op) = reduce_tr(t, op(h,base), op)
```

The second program reduce is the tail-recursive version of the first program. Prove that they return the same results, if applied to the same arguments.

You can assume the following properties about op. The value base is the identity for the op.

```
(a) op(base, m) = m
(b) op(n,m) = op(m,n)
(c) op(n, op(m,k))) = op(op(n,m), k)
```

Before you begin your proof, state carefully the statement you want to prove, show how your proof proceeds, and state carefully your induction hypothesis.

Hint: Proving the following lemma would be very useful in your overall proof.
For any h, l, n, op,
`op(h, reduce(l, n, op)) = reduce tr(l, op(h, n), op)`