COMP 302: Midterm Exam.
5 June 2008

Please answer all questions in the space provided.

Question 1.a. (25 points) Write a function `transform` that has the following behavior. The input should be a generic function `f:'a -> 'b`, and the output should be a function which takes an `'a list` and applies `f` to all of the elements of the list. For example, the output of:

```
transform (fn x => 2 * x) [1,2,3,4,5]
```

will be

```
[2,4,6,8,10]
```

```
fun transform p =
            let
               fun t' nil = nil
                  |t' (x::xt) = p(x)::t'(xt);
            in
               t'
            end
```

```
fun transform' p nil = nil
   |transform' p (x::xt) = p(x)::(transform' p xt);
```

Question 1. b. (5 points) What is the type of `transform`?

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

Question 2.a. (25 points) Suppose that we use the following data structure to represent full binary trees:

```
datatype 'a bintree = Leaf of 'a |
                  Node of 'a bintree * 'a * 'a bintree
```

Write a function `predcount` which takes as input a predicate function `p` of type `'a -> bool` as well as an `'a bintree`, and returns the sum of all of the nodes and leaves which satisfy `p`.

For instance, we may call `predcount` with a function `fn x => x > 0` and an `int bintree`, and the result will be the total number of positive nonzero elements in the tree.

```
fun predcount p (Leaf(x)) = if p(x) then 1 else 0
  |  predcount p (Node(Ltree, x, Rtree)) =
                      (predcount p Ltree) +
                       (predcount p Rtree) +
                      (if p(x) then 1 else 0);
```

Question 2. b. (5 points) What is the type of `predcount`?

```
('a -> bool) -> 'a bintree -> int

(depending on your answer for 2.a, the answer might be
 ('a -> bool) * 'a bintree -> int )
```

Question 3: (20 points) Consider the following function:

```
fun fiter (f, 1: int) = f
  | fiter (f, n: int) =  let
                            val frest = fiter (f,(n-1))
                         in
                            (fn x => f(frest(x)))
                         end;
```

What is the output of `fiter (f,1)`? `fiter (f,2)`? `fiter (f,3)`? Be careful, and be precise.

```
fiter(f,1) => f

fiter(f,2) => (fn x => f(f(x)))

fiter(f,3) => (fn x => f((fn x' =>f(f(x')))x))
```

Question 4: In this question, you need to design an SML data type for logical expressions. Here is the definition of a logical expression that you should base your data type:

- *true* and $false$ are logical expressions,
- variables x, y, and z are logical expressions,
- if ϕ is a logical expression, then $\neg\phi$ is a logical expression,
- if ϕ_1 and ϕ_2 are logical expressions, then $\phi_1 \wedge \phi_2$ (the *and* operator) and $\phi_1 \vee \phi_2$ (the *or* operator) are logical expressions.
- if ϕ is a logical expression and v is a variable, then $\exists v\phi$ is a logical expression.

Question 4.a (10 points) Design a data type for logical expressions. Note: it may be helpful to define a separate data type for variables.

```
datatype var = X | Y | Z;

datatype LExp = True | False | Var of var |
            Not of LExp | And of LExp * LExp |
            Or of LExp * LExp | Exists of var * LExp;
```

Question 4.b (10 points) A variable is bound if it is enclosed within an $\exists$ quantifier. Otherwise it is free. for instance, in the expression $y \wedge (\exists x(x \wedge y))$, the occurrence of x is bound but both occurrences of y are free.

Write a function which takes as input an expression e, a variable v, and an assignment a (being either $true$ or $false$), and produces an expression e' which is the result of replacing all the free occurrences of v in e with a.

```
fun lebind (e, v, a) =
      case e of
          True => True
        | False => False
        | And(e1, e2) => And(lebind(e1,v,a), lebind(e2,v,a))
        | Or(e1,e2) => Or(lebind(e1,v,a),lebind(e2,v,a))
        | Not(e1) => Not(lebind(e1, v, a))
        | Exists(v1, e1) =>
            Exists(v1, if v1=v then e1 else lebind(e1,v,a))
        | Var(v1) => if v=v1 then
                        if a then True else False
                     else
                        Var(v1);
```