Homework 6

Please email your solutions to Rongdi Huang (rh1424@nyu.edu). Solutions to programming exercises **must** be submitted electronically as plain text files. No exotic formats, please!

The deadline for Homework 6 is October 31.

For the following problems, make sure your code runs under SML/NJ. SML/NJ is available on the CIMS machines and can be downloaded from `http://www.smlnj.org` for various platforms. Also, many Linux distributions provide appropriate packages.

Problem 1 Polymorphic Types in ML (4 Points)

Declare polymorphic functions that satisfy the following type signatures:

(a) a function `id: 'a -> 'a`

(b) a function `com: ('a -> 'b) -> ('b -> 'c) -> 'a -> 'c`

(c) a function `cas: ('a * 'b -> 'c) -> 'a -> 'b -> 'c`

(d) a function `car: ('a * 'b -> 'c) -> 'a * 'b -> 'c`

Your functions should not be recursive. Note that the given type signatures uniquely determine the behavior of each function.

Problem 2 ML Lists (6 Points)

(a) Declare a function `unzip: ('a * 'b) list -> 'a list * b' list` that takes a list of pairs and splits it component-wise into two lists. Some examples:

```
- unzip [(1, true), (3, false)];
val it = ([1, 3], [true, false]) : int list * bool list
- unzip [("a", 3), ("c", 2), ("b", 1)];
val it = (["a", "c", "b"], [3, 2, 1]) : sting list * int list
```

Try to use the function `foldr` in your implementation.

(b) Declare the fold function `foldr` using the fold function `foldl`. Do not use any auxiliary recursive functions.

(c) Declare the fold function `foldl` using the fold function `foldr`. Proceed as follows:

1. Declare `append` using `foldr`.
2. Declare `rev` in terms of `foldr` and `append`.
3. Declare `foldl` in terms of `foldr` and `rev`.

Problem 3 ML Datatypes (10 Points)

Your goal is to write a function that differentiates polynomials with respect to a variable x. Here is an example:

$$(x^3 + 3x^2 + x + 2)' = 3x^2 + 6x + 1$$

We represent polynomials using the following type:

```
datatype exp = Const of int
             | X
             | Add of exp * exp
             | Mult of exp * exp
             | Power of exp * int
```

For instance, the expression

```
Add (Add (Mult (Const 3, Power (X, 2)), Mult (Const 6, X)), Const 1)
```

represents the polynomial $3x^2 + 6x + 1$.

(a) Write a **`val`** declaration that binds the identifier `u` to the polynomial expression $x^3 + 3x^2 + x + 2$. Consider $+$ to be left-associative. **(1 Point)**

(b) Write a function `derive: exp -> exp` that computes the derivative of a polynomial expression according to the following rules:

$$\begin{aligned} c' &= 0 \\ x' &= 1 \\ (u+v)' &= u' + v' \\ (u \cdot v)' &= u' \cdot v + u \cdot v' \\ (u^n)' &= n \cdot u^{n-1} \cdot u' \end{aligned}$$

The expression representing the derivative is allowed to contain subexpressions that can be further simplified (e.g., $0 \cdot u$). **(3 Points)**

(c) Write a function `simplifyTop: exp -> exp` that tries to simplify an expression on the top-level by applying one of the following simplification rules:

$$\begin{array}{rcl@{\qquad\qquad}rcl} 0+u & \to & u & u+0 & \to & u \\ 0 \cdot u & \to & 0 & u \cdot 0 & \to & 0 \\ 1 \cdot u & \to & u & u \cdot 1 & \to & u \\ u^0 & \to & 1 & u^1 & \to & u \end{array}$$

If none of these rules can be applied on the top-level of the expression, then the expression should be returned unchanged. **(3 Points)**

(d) Write a function `simplify: exp -> exp` that simplifies an expression using the above rules until none of the rules can be applied. Proceed as follows:

1. First, simplify all components of an expression.
2. Then simplify the expression with the simplified components using the function `simplifyTop`. **(3 Points)**