HW 16 **Prof. Caldwell**
Due: 6 November 2008 **COSC 3015**

1 Substitution

Here is some of the code given in class.

```
data Term = V String
          | Ap Term Term
          | Abs String Term
          | Pair Term Term
          | Spread Term (String,String) Term
  deriving (Eq)

fv (V s) = [s]
fv (Ap m n) = fv m ++ fv n
fv (Abs x m) =   filter (/=x) (fv m)
fv (Pair m n) = fv m ++ fv n
fv (Spread m (x,y) n) = fv m ++ fv (Abs x (Abs y n))
fresh x m =
   if x `elem` m then
     fresh (x ++ x) m
   else
     x

subst (y, n) (V s) = if y == s then n else (V s)
subst (x, n) (Ap m k) = Ap (subst (x, n) m) (subst (x, n) k)
subst (x, n) (Abs y m) =
    if x == y  then
       Abs y m
    else
       if y `elem` (fv n) then
          Abs z (subst (x,n) (subst (y, V z) m))
       else
          Abs y (subst (x,n) m)
  where z = fresh "z" (x : y : ((fv n) ++ (fv m)))
subst (x,n) (Pair m1 m2) = Pair (subst (x,n) m1) (subst (x,n) m2)
subst (x,n) (Spread m (z,y) m1) =
     let m' = subst (x,n) m in
     let t = (Abs z (Abs y m1)) in
     let t' = subst (x,n) t in
        case t' of
           (Abs z' (Abs y' m1')) -> Spread m' (z,y) m1'
           _ -> error "subst: impossible case!"

test f t = show t ++ " ---> " ++ show (f t)
```

The rest of the code is in the file linked to from the homework page.
There is problem with substitution on spread terms. Consider the following evaluation.

```
Main> test (subst ("x", V "y")) (Spread (Pair (V "M") (V "N")) ("y","z") (V "x"))
"subst (x, y) spread(<M,N>;y,z.x)  -->  spread(<M,N>;y,z.y)"
```

In this substitution, when the x is replaced by y, y got captured. This is what should happen.

```
Main> test (subst ("x", V "y")) (Spread (Pair (V "M") (V "N")) ("y","z") (V "x"))
"subst (x, y) spread(<M,N>;y,z.x)  -->  spread(<M,N>;yy,z.y)"
```

In this case, the binding variable y is renamed to yy before the substitution is done and changing x to y is now safe.

Here's another case.

```
Main> test (subst ("x", V "y"))
           (Spread (Pair (V "M") (V "N")) ("y","z") (Ap (V "x")(V "y")))
"subst (x, y) spread(<M,N>;y,z.(x y))  -->  spread(<M,N>;y,z.(y yy))"
```

It should read as follows:

```
Main> test (subst ("x", V "y"))
           (Spread (Pair (V "M") (V "N")) ("y","z") (Ap (V "x")(V "y")))
"subst (x, y) spread(<M,N>;y,z.(x y))  -->  spread(<M,N>;yy,z.(y yy))"
```

Exercise 1.1. Fix the subst function to have the proper behavior[1]

2 Evaluating Spread

In class I suggested the following semantics for evaluating the spread operator:

```
spread'(⟨M1,M2⟩;(x,y).N)  ⟶  (N[x:=M1])[y:=M2]
```

But under these semantics, the following bad behavior occurs.

```
Main> test spread'  (Spread (Pair(V "y") (V "M")) ("x","y") (Ap (V "x") (V "y")))
"spread(<y,M>;x,y.(x y)) ---> (M M)"
```

The problem is shown in more detail with the following step-by-step evaluation:

```
spread'  (Spread (Pair(V "y") (V "M")) ("x","y") (Ap (V "x") (V "y")))
⟶ subst ("y",V "M") (subst ("x", V"y") (Ap (V "x") (V "y")))
⟶ subst ("y",V "M") (Ap (V "y") (V "y")))
⟶ Ap (V "M") (V "M")
```

After substituting y for x it is essentially captured. This can be fixed by writing the code that explicitly tests for the appropriate cases, but instead we can use the code we've already written for the abstraction case and selective application of beta reduction to take care of it for us.

Here are the alternative semantics:

[1]This is a really easy fix after you think about it for a few minutes, a very minor modification.

```
spread(⟨M1,M2⟩;(x,y).N) ⟶ beta (beta (((λx.(λy.N))) M1) M2)
```

Defining it this way avoids having a `y` that occurs in the free variables of `M1` getting substituted for when `y` is replaced by `M2`. So, for example,

```
Main> test spread  (Spread (Pair(V "y") (V "M")) ("x","y") (Ap (V "x") (V "y")))
"spread(<y,M>;x,y.(x y)) ---> (y M)"
```

Exercise 2.1. Define the `spread` function (as we did in class for `beta`) using the alternative semantics and test your code to show that it behaves properly in a number of situations.