Madhuri Pyreddy
17 October 2019

CSCI 117 Lab 8

Part One:

a1.
```
local X Y Generate Display DisplayH in
   fun {Generate N}
    fun {$} (N#{Generate (N+1)}) end
   end

    proc {Display X N}
      fun {DisplayH Z Num}
       if (Num == 0) then nil
       else
       (V#F) = {Z} in
       (V|{DisplayH F (Num-1)})
       end
      end
    local L in
      L = {DisplayH X N}
      skip Browse L
    end
   end

  //Testing
  X = {Generate 3}
  {Display X 5}

  end
```

a2.
```
local X Y Generate Display DisplayH Times in
    fun {Generate N}
     fun {$} (N#{Generate (N+1)}) end
    end
    fun {Times X Y}
    fun {$}
      (V#F) = {X} in
      ((V*Y)#{Times F Y})
     end
    end
```

```
   proc {Display X N}
     fun {DisplayH Z Num}
      if (Num == 0) then nil
      else
      (V#F) = {Z} in
      (V|{DisplayH F (Num-1)})
      end
     end
    local L in
     L = {DisplayH X N}
     skip Browse L
    end
   end

  //Testing
  X = {Generate 3}
  Y = {Times X 3}
  {Display Y 5}
   end
```

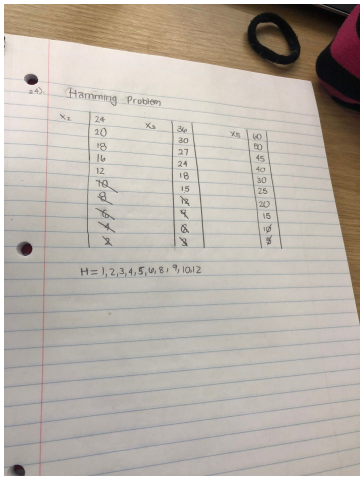
a3.
```
local X Y Generate Display DisplayH Merge in
   fun {Generate N}
    fun {$} (N#{Generate (N+1)}) end
   end
    Merge = fun {$ X Y}
    fun {$}
       (V#F) = {X}
       (U#H) = {Y} in
         if (V < U) then (V#{Merge F Y})
         else
            if (V > U) then (U#{Merge X H})
         else (V#{Merge F H})
             end
         end
       end
    end
   proc {Display X N}
     fun {DisplayH Z Num}
      if (Num == 0) then nil
```

```
            else
            (V#F) = {Z} in
            (V|{DisplayH F (Num-1)})
            end
           end
          local L in
           L = {DisplayH X N}
           skip Browse L
          end
         end

        //Testing
        X = {Generate 3}
        Y = {Generate 5}
        {Display {Merge X Y} 8}
         end
    a4. local X Y Generate Display DisplayH Merge Times H in
fun {Generate N}
 fun {$} (N#{Generate (N+1)}) end
end
 fun {Times X Y}
 fun {$}
  (V#F) = {X} in
  ((V*Y)#{Times F Y})
 end
end
 Merge = fun {$ X Y}
 fun {$}
    (V#F) = {X}
    (U#H) = {Y} in
      if (V < U) then (V#{Merge F Y})
      else
        if (V > U) then (U#{Merge X H})
      else (V#{Merge F H})
          end
      end
    end
  end
 H = fun {$}
```

```
  (1#{Merge {Times H 2} {Merge {Times H 3} {Times H 5}}})
 end
  proc {Display X N}
    fun {DisplayH Z Num}
     if (Num == 0) then nil
     else
     (V#F) = {Z} in
     (V|{DisplayH F (Num-1)})
     end
    end
  local L in
    L = {DisplayH X N}
    skip Browse L
  end
 end

//Testing
{Display H 10}
 end
```




```
      b. b1.
         b. modG :: Gen Int -> Int -> Gen Int
         modG x n = let G(v,f)= x in
         G ((v mod n), (modG f n))

         *Main> G (v,f) = modG (gen 0) 2
         *Main>   G (v1,f1) = f
```

```
*Main> G (v2, f2) = f1
*Main> v
0
*Main> v1
1
*Main> v2
0
```

```
b2. interleave :: [Gen Int] -> Gen Int
interleave (x:xs) = let G(v1,f1) = x in
                    G(v1, (interleave (xs ++ [f1])))
```

```
*Main> G (v1,f) = interleave [gen 3, gen 7, gen 13]
*Main> G (v1,f1) = f
*Main> G (v2,f2) = f1
*Main> G (v3,f3) = f2
*Main> v
0
*Main> v1
7
*Main> v2
13
*Main> v3
4
```

Part 2:

```
local GateMaker AndG OrG NotG A B S IntToNeed Out MulPlex in

GateMaker = fun {$ F}
        fun {$ Xs Ys} T
                GateLoop = fun {$ Xs Ys}
                        case Xs of nil then nil
                        [] '|'(1:X 2:Xr) then
                                case Ys of nil then nil
                          [] '|'(1:Y 2:Yr) then
                                        ({F X Y}|{GateLoop Xr Yr})
                                end
                        end
```

```
                        end
                        in
                        T = thread X = {GateLoop Xs Ys} in X end // thread wasn't
added to expressions
                        T
                end
        end

        NotG = fun {$ Xs} T
                Loop = fun {$ Xs}
                  case Xs of nil then nil
                  [] '|'(1:X 2:Xr) then ((1+(X*-1))|{Loop Xr})
                  end
                end
                in T = thread X = {Loop Xs} in X end T
        end

        AndG = {GateMaker fun {$ X Y} if (X == 0) then 0 else (X*Y) end end}
        OrG =  {GateMaker fun {$ X Y} if (X == 1) then 1 else (X+Y) end end}

        fun {IntToNeed L}
           case L of nil then nil
       [] '|'(1:X 2:Xr) then T W in
        byNeed fun {$} X end W
        T = {IntToNeed Xr}
        (W|T)
        end
      end

 fun {MulPlex A B S} R Z T W in
   R = {NotG S}
   Z = {AndG R A}
   T = {AndG S B}
   W = {OrG Z T}
   W
   end

 A = {IntToNeed [0 1 1 0 0 1]}
        B = {IntToNeed [1 1 1 0 1 0]}
```

```
        S = [1 0 1 0 1 1]
        Out = {MulPlex A B S}

    // run a loop so the MulPlex threads can finish before displaying Out
    local Loop in
    proc {Loop X}
      if (X == 0) then skip Basic
      else {Loop (X-1)} end
    end
    {Loop 1000}
    end
          skip Browse Out
     end
```

a.
```
fun {IntToNeed L}
          case L of nil then nil
        [] '|'(1:X 2:Xr) then T W in
         byNeed fun {$} X end W
         T = {IntToNeed Xr}
         (W|T)
         end
       end
```

b.
```
AndG = {GateMaker fun {$ X Y} if (X == 0) then 0 else (X*Y) end end}
OrG =  {GateMaker fun {$ X Y} if (X == 1) then 1 else (X+Y) end end}
```

c.
```
 fun {MulPlex A B S} R Z T W in
    R = {NotG S}
    Z = {AndG R A}
    T = {AndG S B}
    W = {OrG Z T}
    W
    end
```

d.

d1. It depends on the value of S. If S is zero, it does not need the values of A and B. If S is 1, then it needs the value of the other variables. For instance, if A = 0, B=1, and S=1, then S will take the values of other variables like A= 0 and B=1. If A= 0, B=0, and S=0, then S will not take the values of other variables because it doesn't need it.

d2. Yes, they match up with what I got in (d.1).