Homework 3

James Zhang zhany111

September 30, 2019

1 Q1

1) Filter Function
Computed using Ruby.

```
def Filter( condition, list)
  a = []
  for i in list do
    if condition.call(i)
      a += i
    end
  end
  return a
end

def f(n)
  if (n == 2)
    return true
  else
    return false
  end
end

l = [1, 2, 3]
puts Filter( f(:int), l)
```

2) Map Function
Computed using Oz

```
declare
%%Map Function
fun {Map Rule L}
   % return [nil] when the input is nil.
   if L == [nil] then
      [nil]
   else
      [{Rule A} suchthat A in L]
   end
end

%%Two functions that can be use to test
fun {Twice A}
   A + A
end
fun {Addone A}
   A + 1
end

declare
L0 = [nil]
L1 = [0]
L2 = [2 0 1 9]
L3 = [99 9 19 29]
L4 = [1 2 3 4]
{Browse {Map Twice L0}}
{Browse {Map Twice L1}}
{Browse {Map Twice L2}}
{Browse {Map Addone L3}}
{Browse {Map Addone L4}}
```

3) Reduce Function
Computed using F-sharp. I will test 3) and 4) together.

```
let rec Reduce (rule, result, li : List<'T>) =
    let mutable l = li
    let mutable a = result
    ///when there are enough elements, produce
    /// the next output by recursion.
    if l.Length >= 1 then
        a <- rule(a, l.Head)
        l <- l.Tail
        Reduce (rule, a, l)
    //if there are no more element, return the result
    else
        a
```

4) Accumulate Function
Also using F=sharp

```
///The same idea as the last function, just inverse
/// the input list
let rec Accumulate (rule, result, li : List<'T>) =
    let mutable l = li |> List.rev
    let mutable a = result
    if l.Length >= 1 then
        a <- rule(a, l.Head)
        l <- l.Tail
        Accumulate (rule, a, l)
    else
        a
```

Test cases for 3) and 4):

```
///Functions to test
let Sum (a, b) =
    a + b
let Power (a, b) =
    let mutable p = 1
```

```
    for i = 1 to b do
        p <- p * a
    p
let Add (a, b) =
    a + b

//empty list
let l0 = []
let t01 = Reduce(Sum, 0, l0)
let t02 = Accumulate(Sum, 0, l0)
printfn "%O" t01
printfn "%O" t02

//Sum and Power on non-empty int list
let l1 = [1; 2; 3; 4]
let t11 = Reduce(Sum, 0, l1)
let t12 = Accumulate(Sum, 0, l1)
printfn "%O" t11
printfn "%O" t12


let l2 = [1; 2]
let t21 = Reduce(Power, 2, l2)
let t22 = Accumulate(Power, 2, l2)
printfn "%O" t21
printfn "%O" t22

let l3 = [1; 2; 3]
let t31 = Reduce(Power, 3, l3)
let t32 = Accumulate(Power, 3, l3)
printfn "%O" t31
printfn "%O" t32

//test Accumulate does reverse the list
let l4 = ["1"; "2"; "3"]
let t41 = Reduce(Add, "", l4)
let t42 = Accumulate(Add, "", l4)
printfn "%O" t41
printfn "%O" t42
```

2 Q2

1.