Report of Assignment 6

Haodong Liao

July 10, 2019

1 Preface

This is a LaTeXreport written by Haodong Liao, a student of UESTC who participated in the summer school of UCPH. More codes of this summer school are at MyRepository.

2 Introduction

A list in F# is an ordered, immutable series of elements of the same type[1], and some of the list functions like map, fold and filter are very powerful for list processing.

In this assignment, I replaced the library calls in *myFold* and *myFilter* with my own recursive implementations of these functions according to the assignment description. In detail, in both functions, I used *match..with* to achieved pattern-matching and the idea of recursion to implemented the functions.

3 Analysis and Design

```
List.fold:
  f:('State -> 'T -> 'State) -> elm:'State -> lst: 'T list -> 'State
```

According to [1], List.fold function *updates an accumulator iteratively by applying f to each element in lst*.

So there were two branches needed to be handled, one is an empty list, the other is a list with element(s). For empty list, the return value couldn't be constrained to list and should be as same as accumulator. For a list with element(s), fold function should be called recursively, the accumulator should always be the result of argument function which had parameters of the accumulator and the current element of the list. The key part of the pseudocode is shown as following:

```
Fold function accumulator list =
  match list with
    | emptylist -> return accumulator
    | list with element(s) ->
        Fold function (result of function with
        current accumulator and element) restList
```

```
List.filter: f:('T -> bool) -> lst:'T list -> 'T list
```

According to [1], List.filter function *returns a new list with all the elements of lst for which f evaluates to true*.

Same as fold function, there were two branches needed to be handled. For an empty list, the return value should be the empty list itself. For a list with element(s), filter function should be called recursively according to the bool expression, if the value was true, not only should we call the filter function recursively, the current element should be a part of the result list. If the value of bool expression is false, there was no need to care about the current element. The key part of the pseudocode is shown as following:

```
Filter boolexpression list =
  match list with
    | emptylist -> return emptylist
    | list with element(s) ->
        if boolexpression is true
        then
          concat current element
          call Filter function with boolexpression and rest list
        else
```

[1] https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/lists

```
10          call Filter function with boolexpression and rest list
```

4 Program description

My implementation of fold function was as follows:

```
let rec myFold (f: 'b -> 'a -> 'b) (acc: 'b) (lst: 'a list) : 'b =
  match lst with
    | [] -> acc
    // [] -> []          // second time wrong
    | elm::rest ->
        //let result = myFold f acc rest @ (f acc elm) // first time wrong
        let result = myFold f (f acc elm) rest
        result
```

As it shows, to my point of view, the place I made mistake was the key of this function, for an empty list or a list with element(s), a problem I met was that I constrained the type of return value. It was a subtle but vital problem.

My implementation of filter function was as follows:

```
let rec myFilter (p: 'a -> bool) (lst: 'a list) : 'a list =
  //List.filter p lst
  match lst with
  | [] -> []
  | elm::rest ->
    if (p elm) then [elm] @ (myFilter p rest)
    else myFilter p rest
```

I struggled in the code of a list with element(s) when I started to solve this problem, and the point confused me was that I got bogged down with the detail of recursion and ignored the branches of boolean expressions. Things went smoothly when I rearranged my thoughts.

5 Evaluation

The testing environment was macOS Mojave 10.14.5 system with iTerm and Microsoft (R) F# Compiler version 4.1.

I complied and tested the fold and filter function with parameter *fold 9* and *filter 9*, and the result were correct and shows in 1 and 2 separately:

```
λ MedillEast [Functional Programming/SummerSchool/SummerSchool] at  master !?
→ fsharpc 6\ Individual\ hand-in\ assigment:\ Map,\ Fold,\ and\ Filter.fsx && mono 6\ Indi
vidual\ hand-in\ assigment:\ Map,\ Fold,\ and\ Filter.exe fold 9
Microsoft (R) F# Compiler version 4.1
Copyright (c) Microsoft Corporation. All Rights Reserved.
Random list is: [5; 7; 8; 6; 4; 6; 7; 4; 7]
Result is CORRECT  : [14; 8; 14; 12; 8; 12; 16; 14; 10]
```

Figure 1: Testing of fold function

```
λ MedillEast [Functional Programming/SummerSchool/SummerSchool] at  master !?
→ fsharpc 6\ Individual\ hand-in\ assigment:\ Map,\ Fold,\ and\ Filter.fsx && mono 6\ Indi
vidual\ hand-in\ assigment:\ Map,\ Fold,\ and\ Filter.exe filter 9
Microsoft (R) F# Compiler version 4.1
Copyright (c) Microsoft Corporation. All Rights Reserved.
Random list is: [9; 0; 8; 8; 4; 0; 1; 5; 8]
Result is CORRECT  : [9; 8; 8; 5; 8]
```

Figure 2: Testing of filter function

6 Conclusion

In this assignment, I implemented the *myFold* and *myFilter* functions with my own recursive method. I stuck at the beginning of my writing, but things went smoothly when I rearranged my thoughts and wrote down the pseudocode. It's easy to get bogged down in the details of a program, but we should take a top-down functional programming approach and thinking more about what to do than how to do it.

References

[1] Jon Sporring. *Learning to Program with F#.* 2019.