MP8 – Higher-order Functions

CS 421 – Summer 2009
Revision 1.0

Assigned July 22, 2009
Due July 29, 2009, 1:00 PM
Extension 48 hours (penalty 20% of total points possible)
Total points 50 (+ 10 Extra Credit)

1 Change Log

1.0 Initial release.

2 Overview

This MP you will help you master programming with higher-order functions.

3 Collaboration

Collaboration in two-person groups is allowed.

4 What to submit

You will submit your `mp8.ml` and `mp8ec.java` files via Compass. Rename `mp8-skeleton.ml` to `mp8.ml`, `mp8ec-skeleton.java` to `mp8ec.java`, and start working from there.

5 Instructions

Here are the instructions for this MP.

- Download `mp8grader.tar.gz`. This tarball contains all the files you need.
- As always, extract the tarball, rename `mp8-skeleton.ml` to `mp8.ml`, and start modifying the file. If you choose to do the extra credit, similarly rename `mp8ec-skeleton.java` to `mp8ec.java`.
- Compile your solution with `make`. Run the `./grader` to see how well you do for the required problems, and `java ECGrader` for the extra credit problem.
- Note that you may not be able to run the grader until you implement several of the functions in the MP.
- Make sure to add several more test cases to the `tests` file.

The problems below have sample executions that suggest how to write your solutions. You have to use the same function name, but the name of the parameters that follow the function name need not be duplicated. That is, you are free to choose different names for the arguments to the functions from the ones given in the example execution. We also will use let rec to begin the definition of a function that is allowed to use rec. You are not required to start your code with let rec.

For all these problems, you are allowed to write your own auxiliary functions, either internally to the function being defined or externally as separate functions. In fact, you may find it helpful to do so on several problems. All helper functions must satisfy any coding restrictions (such as not using explicit recursion) as the main function being defined for the problem must satisfy. Here is the summary of restrictions for the assignment:

- The function name must be the same as the one provided in the skeleton.
- The type of parameters must be the same as the parameters shown in sample execution.
- You must comply with any special restrictions for each problem. Some of the problems require that the solution should use higher-order functions in place of explicit recursion.

Included Helper Functions

Here are some functions used in this assignement; they are included in `mp8common.ml` so you do not have to write them yourself; you can use them as desired. You may want to look at `mp8common.ml` to get a feel for what is available.

In some cases, they are used just for examples; in others, they are used in our solutions, so you might find them helpful in yours.

```
let incr x = x + 1;;
let decr x = x - 1;;
let funmod f x y = fun z -> if x = z then y else f z;;
let compose f g = fun x -> f (g x);;
```

6 Problems

1. (6 pts) Define `split` such that `split f lis` returns a pair of lists `(lis1, lis2)` where `lis1` contains all the elements of `lis` for which `f` is true, and `lis2` contains all the others (in the same order as they appear in `lis`).

```
# let rec split f lis =
val split : ('a -> bool) -> 'a list -> 'a list * 'a list = <fun>
# split (fun x -> x < 0) [1; 5; 0; -4; -3; 2];;
- : int list * int list = ([-4; -3], [1; 5; 0; 2])
```

2. (6 pts) Define `filter` such that `filter f lis` returns a list containing the elements of `lis` for which `f` is true. Then use `filter` to define `filterlt` such that `filterlt n lis` returns all the elements of `lis` that are less than or equal to `n`.

```
# let rec filter f lis = ...
val filter : ('a -> bool) -> 'a list -> 'a list = <fun>
# let filterlt n = filter ...
val filterlt : 'a -> 'a list -> 'a list = <fun>
# filterlt 10 [4; 25; 12; 8; 10; 7; 11];;
- : int list = [4; 8; 10; 7]
```

3. (6 pts) For this problem, you must use `fold_right` from the `List` module, and **no explicit recursion**. Define a function `length`, which gives the length of a list, by defining value `length_base` and function `length_recur`, and calling `fold_right length_recur lis length_base`. The function `length_recur` takes as arguments the head of the list and the result of the recursive call on the tail of the list, and gives the result for this list.

```
# let length_base = ...;;
val length_base : int = ...
# let length_recur h x = ...;;
val length_recur : 'a -> int -> int = <fun>
# let length lis = fold_right length_recur lis length_base;;
val length : 'a list -> int = <fun>
# length [1;2;3;4;5];;
- : int = 5
```

4. (6 pts) Using `map` from the `List` module, and **no explicit recursion**, define the function `case_map` such that `case_map f g h lis` returns a list with the following values: wherever `lis` has a value `x` for which `f` is true, it has `g x`, and everywhere else it has `h x`.

```
# let case_map f g h = map ...;;
val case_map : ('a -> bool) -> ('a -> 'b) -> ('a -> 'b) -> 'a list
  -> 'b list = <fun>
# case_map (fun x -> x > 2) incr decr [0;1;2;3];;
- : int list = [-1; 0; 1; 4]
```

5. (6 pts) Write a function `app_all : ('a -> 'b) list -> 'a list -> 'b list list` that takes a list of functions, and a list of arguments for those functions, and returns the list of list of results from consecutively applying the functions to all arguments, in the order in which the functions occur in the list and in the order in which the arguments occur in the list. Each list in the result list corresponds to a list of applications of each function to the given arguments. The definition of `app_all` may use the library function `List.map : ('a -> 'b) -> 'a list -> 'b list` but **no explicit recursion, and no other library functions**.

```
# let app_all fs lis = ... ;;
val app_all : ('a -> 'b) list -> 'a list -> 'b list list = <fun>
# app_all [(fun x -> x > 0); (fun y -> y mod 2 = 0);
  (fun x -> x * x = x)] [1; 3; 6];;
- : bool list list =
[[true; true; true]; [false; false; true]; [true; false; false]]
```

6. (20 pts) In this problem you will implement several multiset operations. A *multiset*, or a bag, is a set that can contain multiple copies of an element. Just like sets, multisets are not ordered. In this assignment, we represent multisets with functions. A multiset function returns the number of occurrences of the given element in the multiset.

```
type 'a multiset = 'a -> int

let emptymultiset : 'a multiset = fun x -> 0
```

The definitions above are given in the `mp8common.ml` file. Some examples for possible implementations of multisets:

```
{1,1,1,2,2} = fun n -> match n with
                          1 -> 3
                        | 2 -> 2
                        | _ -> 0
{4,2,4,2,3} = fun n -> if n = 4 || n = 2 then 2
                       else if n = 3 then 1
                       else 0
```

Implement the following multiset operations:

(a) `msAdd n s :  int -> 'a multiset -> 'a multiset.`

(b) `msMember n s :  'a -> 'a multiset -> bool.`

(c) `msUnion s1 s2 :  'a multiset -> 'a multiset -> 'a multiset.`
E.g., $\{1;1;1;2;2;3\} \cup \{1;1;2;3;3;4\} = \{1;1;1;2;2;3;3;4\}$.

(d) `msDisjointUnion s1 s2 :  'a multiset -> 'a multiset -> 'a multiset.`
E.g., $\{1;1;1;2;2;3\} \uplus \{1;1;2;3;3;4\} = \{1;1;1;1;1;2;2;2;3;3;3;4\}$.

(e) `msIntersection s1 s2 :  'a multiset -> 'a multiset -> 'a multiset.`
E.g., $\{1;1;1;2;2;3\} \cap \{1;1;2;3;3;4\} = \{1;1;2;3\}$.

(f) `msRemove n s :  'a -> 'a multiset -> 'a multiset.`
Remove an occurrence of `n` from `s`.

(g) `msFilter f s :  ('a -> bool) -> 'a multiset -> 'a multiset.`
Remove the elements of `s` that do not satisfy the prediate `f`.

(h) `msFromList lst :  'a list -> 'a multiset.`
Construct a multiset from the given list.

When you implement all the operations, any equality check below should evaluate to true. The code below is available in `msExamples.ml`.

```
let set1 = msFromList [1;1;1;2;2;3;4;7];;
let set2 = msFromList [1;1;2;3;3;5;6;6];;
let un = (msUnion set1 set2);;
let dun = (msDisjointUnion set1 set2);;
let fil = msFilter (fun x -> x>2) set2;;
let inter = (msIntersection set1 set2);;


set1 1 = 3;;
set1 2 = 2;;
set1 3 = 1;;
set1 4 = 1;;
set1 5 = 0;;
set1 6 = 0;;
set1 7 = 1;;
set1 8 = 0;;

msMember 3 set1 = true;;
msMember 5 set1 = false;;
```

```
(msAdd 5 set1) 5 = 1;;
(msAdd 1 set1) 1 = 4;;
(msRemove 1 set1) 1 = 2;;
(msRemove 2 set1) 2 = 1;;
(msRemove 3 set1) 3 = 0;;
(msRemove 3 set1) 1 = 3;;

fil 1 = 0;;
fil 2 = 0;;
fil 3 = 2;;
fil 4 = 0;;
fil 5 = 1;;
fil 6 = 2;;

un 1 = 3;;
un 2 = 2;;
un 3 = 2;;
un 4 = 1;;
un 5 = 1;;
un 6 = 2;;
un 7 = 1;;
un 8 = 0;;

dun 1 = 5;;
dun 2 = 3;;
dun 3 = 3;;
dun 4 = 1;;
dun 5 = 1;;
dun 6 = 2;;
dun 7 = 1;;
dun 8 = 0;;

inter 1 = 2;;
inter 2 = 1;;
inter 3 = 1;;
inter 4 = 0;;
inter 5 = 0;;
inter 6 = 0;;
inter 7 = 0;;
inter 8 = 0;;
```

7. (10 pts EC) The extra-credit problem is a bit different from the rest of the MP. Your task is to implement multisets, same as in Problem 6, but this time as function objects in Java. The definition of a multiset is given in `MultiSet.java`:

```
interface MultiSet{
    int apply(int n);
}
```

Some examples of possible implementations of multisets:
$\{1, 1, 1, 2, 2\}$ is

```
new MultiSet(){
    public int apply(int n){
        switch(n){
        case 1: return 3;
        case 2: return 2;
        default: return 0;
        }
    }
}
```

$\{4, 2, 4, 2, 3\}$ is

```
new MultiSet(){
    public int apply(int n){
        if(n == 4 || n == 2)
            return 2;
        else if(n == 3)
            return 1;
        else
            return 0;
    }
}
```

We ask you to implement the multiset operations `add`, `member`, `union`, `disjointUnion`, `intersection`, `remove`, `filter`, and `fromList`, as well as `emptymultiset`.

The definition of the `Predicate` type that is used in the filter function is given in the `Predicate.java` file:

```
interface Predicate{
    boolean apply(int n);
}
```

You should do your implementation in a file named `mp8ec.java`, and submit this file together with `mp8.ml` in the zip file. Use the usual handin process to submit your file. You are provided with a skeleton file to start from: `mp8ec-skeleton.java`.

Typing "`make`" will compile the files. Execute "`java ECGrader`" to test your implementation. We do not provide you with a compiled solution for this problem; instead, test cases are hand-coded in the ECGrader.java file. This is the exact grader we will use for this problem, so you do not need to add any additional test cases.