MP 9A – More Fun with Higher-order Functions

CS 421 – Spring 2010

Revision 1.0

Assigned Tuesday, March 30, 2010
Due Monday, April 5, at 10:00pm
Extension 48 hours (20% penalty)
Total points 15

1 Change Log

1.0 Initial Release.

2 Overview

After completing this MP, you should have a better understanding of higher-order functions.

3 Collaboration

Collaboration is NOT allowed on this assignment.

4 Problems (15 pts)

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.

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

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

The definitions above are given in the `mp9Acommon.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
```

In this MP we ask you to implement the following multiset operations.

1. `add n s : int -> 'a multiset -> 'a multiset.`
2. `member n s : 'a -> 'a multiset -> bool.`
3. `union s1 s2 : 'a multiset -> 'a multiset -> 'a multiset.`
   E.g. $\{1,1,1,2,2,3\} \cup \{1,1,2,3,3,4\} = \{1,1,1,2,2,3,3,4\}$
4. `disjointUnion s1 s2 : 'a multiset -> 'a multiset -> 'a multiset.`
   E.g. $\{1,1,1,2,2,3\} \uplus \{1,1,2,3,3,4\} = \{1,1,1,1,1,2,2,2,3,3,3,4\}$
5. `intersection s1 s2 : 'a multiset -> 'a multiset -> 'a multiset.`
   E.g. $\{1,1,1,2,2,3\} \cap \{1,1,2,3,3,4\} = \{1,1,2,3\}$
6. `remove n s : 'a -> 'a multiset -> 'a multiset.`
   Remove an occurrence of `n` from `s`.
7. `filter f s : ('a -> bool) -> 'a multiset -> 'a multiset.`
   Remove the elements of `s` that do not satisfy the predicate `f`.
8. `fromList 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 `examples.ml`.

```
let set1 = fromList [1;1;1;2;2;3;4;7];;
let set2 = fromList [1;1;2;3;3;5;6;6];;
let un = (union set1 set2);;
let dun = (disjointUnion set1 set2);;
let fil = filter (fun x -> x>2) set2;;
let inter = (intersection 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;;

member 3 set1 = true;;
member 5 set1 = false;;
(add 5 set1) 5 = 1;;
(add 1 set1) 1 = 4;;
(remove 1 set1) 1 = 2;;
(remove 2 set1) 2 = 1;;
(remove 3 set1) 3 = 0;;
(remove 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;;
```