Homework 6

February 28, 2014

1 GPA

The following definitions are for calculating grade point averages.

1. Define a ML data type `grade` for representing the letter grades `A, B, C, D, F`.
2. Define a function `percent2grade: real -> grade` that converts percentage grades x to letter grades

percentage x	letter grade
$x \geq 90.0$	A
$80.0 \leq x < 90.0$	B
$70.0 \leq x < 80.0$	C
$60.0 \leq x < 70.0$	D
$x \leq 60.0$	F

   For example, `percent2grade 85.0` should return B.
3. Define a function `grade2point: grade -> real` that converts letter grades to grade points on 4-point scale.

   For example, `grade2point A` should return 4.0.
4. Define a function `gpa: grade list -> real` that takes a list of letter grades and returns the grade point average of the grades.

   For example, `gpa [A, B, C, D, A, B, A]` returns 3.0.
5. Define a function `gpaFromPercent: real list -> real` that takes a list of percentage grades and returns the grade point average.

   For example, `gpaFromPercent [67.0, 77.0, 84.0, 99.0]` returns 2.5.

2 Binary tree

The following questions use the following datatype definition.

```
datatype 'data tree =
   Empty |
   Node of 'data tree * 'data * 'data tree;
```

1. A complete binary tree is one in which every `Node` has either two `Empty` children or two `Node` children but not one of each. Write a function `isComplete` of type `'a tree -> bool` that tests whether a tree is complete or not. Empty tree is complete.

2. Write a function `makeBST` of type

   `'a list -> ('a * 'a -> bool) -> 'a tree`

   that organizes the items in a list into a binary search tree. The tree needs not to be balanced and you may assume that no items in the list is repeated.

   The 2nd parameter of `makeBST` is a comparison function that compares two items and determine whether the first one is less than the second one or not.

   A binary search tree is either empty or it has two subtrees and a data item x, where the items in the left subtree are all smaller than x, the items in the right subtree are greater than x, and the two subtrees are binary search tree as well.

   For example,

   ```
   - makeBST [1,3,2] (op <);
   val it = Node (Node (Empty,1,Empty),2,Node (Empty,3,Empty)) : int tree
   ```

   Note that depending on your implementation, the shape of the tree may look different though it should contain the same elements.

3. Write a function `searchBST` of type

   `''a tree -> (''a * ''a -> bool) -> ''a -> bool`

   that searches a binary search tree for a given data element and returns true if it is found and false otherwise. Your solution should only search subtrees that may contain the element you are looking for.

   If we apply the function in the following way `searchBST tree f e`, then `searchBST` should first compare `e` with the tree data `d` using `=` to see if `e` and `d` are equal. If they are equal, then return true. Otherwises, `searchBST` should check if `f(e, d)` is true or false, if true, then search the left subtree and if false, it should search the right subtree.

   For example, in the following program, the variable `isFound` should be true.

   ```
   val t = Node( Node( Empty, 4, Empty ),
                 5,
                 Node( Empty, 6, Empty ));

   - searchBST t (op <) 4;
   ```

```
val it = true : bool

val t2 = makeBST [3, 6, 2, 1, 4] (op <);

- searchBST t2 (op <) 2;
val it = true : bool

- searchBST t2 (op <) 5;
val it = false : bool
```