Homework 7: Algebraic Datatypes (Binary tree)

Stephen Wagstaff
CS 431
October 26, 2018

The following questions use the following datatype definition.

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

---

1. A full 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 `isFull` of type `'a tree -> bool` that tests whether a tree is full or not. `Empty` tree is full.

   ```
   fun isFull Empty = true
     | isFull (Node(Empty, _, Node _)) = false
     | isFull (Node(Node _, _, Empty)) = false
     | isFull (Node(leftTree, _, rightTree)) = (isFull leftTree) andalso (isFull rightTree);
   ```

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.

   ```
   fun makeBST nil _ = Empty
     | makeBST (currentElement::list) comparisonFunction =
     let
       fun insert element Empty = Node (Empty, element, Empty)
         | insert element (Node(left, current, right)) =
         if comparisonFunction(element, current)
         then Node((insert element left), current, right)
         else Node(left, current, (insert element right))
     in
       insert currentElement (makeBST list comparisonFunction)
     end;
   ```

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 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
   ```

```
fun searchBST Empty _ _ = false
  | searchBST (Node(left, current, right)) comparisonFunction element =
  if current = element
  then true
  else
    if comparisonFunction(element, current)
    then searchBST left comparisonFunction element
    else searchBST right comparisonFunction element;
```