Fakultät für Informatik und Mathematik
Prof. Dr. Martin Griebl
Summer semester 2023

# Exercises for Lecture: Functional Programming
# Exercise Sheet 5 (Combinators, Trees, Divide & Conquer)

## Problem 1 (`foldr` and `unfoldr`)

The module `Data.List` offers a function

```
unfoldr :: (b -> Maybe (a, b)) -> b -> [a]
```

which can be used to construct lists. In `unfoldr f init`, the function `f` is called repeatedly (with `init` as the input of the first call) and as long as `f` returns `Just (x,next)`, the value `x` is put in the result list and `next` is used as the input for the following call to `f`. When `f` returns `Nothing`, the result list is finished.
Example:
`let f x = if x>5 then Nothing else Just (x,x+1) in unfoldr f 1 == [1,2,3,4,5]`

(a) Implement a function `fromTo x y` using `unfoldr` such that `fromTo x y == [x..y]` holds.

(b) Implement a function `mymap f xs` using `unfoldr` such that `mymap f xs == map f xs` holds.

(c) `unfoldr` constructs lists and `foldr` consumes lists, i.e., one can regard `foldr` as the inverse of `unfoldr`. Implement a (recursive) function `foldUnfoldr` for which the equation

```
foldUnfoldr f x g y  ==  foldr f x (unfoldr g y)
```

holds but which does not construct the intermediate list (returned by `unfoldr`).

## Problem 2 (Trees and Preoder)

Let the following data type for binary trees be given:

```
data Tree = E                    -- empty leaf
          | N Tree Char Tree     -- inner node
```

Example: `N (N E 'a' E) 'b' E`

(a) Implement a function `preorder :: Tree -> String` that prints a tree in preorder. The empty leaves of the tree shall be represented by the character `'.'` and inner node by the character carried by the inner node.

Example: `preorder (N (N E 'a' E) 'b' E) == "ba..."`

(b) Implement a function `fromPreorder :: String -> Tree` that reconstructs a tree from its preoder representation. (We assume that `'.'` does not occur as a character carried by an inner leaf.)

Example: `fromPreorder "ba..." == N ((N E 'a' E) 'b' E)`

Tip: The first character in a (sub)string tells whether the string/a prefix of the string represents an inner node or a leaf.

## Problem 3 (Data Type and Combinators for Binary Trees)

Let the following data type for binary trees with values at the inner nodes and leaves be given:

```
data BTree a = Leaf a | Node (BTree a) a (BTree a)  deriving Show
```

(a) Implement a function `mapBTree` which applies a function to all values in a tree (similar to `map` for lists).

(b) Define a function `foldBTree` which combines (i.e., reduces) the values in a tree to a single value, similar to `foldl1`/`foldr1`.

Note: Since `BTree` has two constructors, `foldBTree` should take two function parameters.

(c) Implement (using `foldBTree`) a function `postOrder :: BTree a -> [a]` which returs the values in a tree according to a postorder pass over the tree.

(d) Implement a function `isSearchTree :: (a -> a -> Bool) -> BTree a -> Bool`. This function shall check if a given tree is a binary search tree w.r.t. to the given order, e.g., `isSearchTree (<=) b` should check if `b` is a binary search tree with values ordered by `(<=)`.

Implement `isSearchTree` using `foldBTree`. Think about which value the reduction must compute such that the result tells whether the tree is a search tree.

**Note:** A binary tree is a search tree when the value at each inner node is greater than all the values in the left sub tree and less than the values in the right sub tree (w.r.t. the given order).

## Problem 4 (Divide and Conquer: Merge Sort)

The lecture introduced the divide and conquer skeleton `dc` (see also the file `DC.hs` in Stud.IP).

Implement, using the skeleton `dc`, a sorting function `mergesort :: Ord a => [a] -> [a]` that works according to the merge sort algorithm and sorts a given list in ascending order.

Note: The *divide* phase of merge sort corresponds to a partitioning of the list to be sorted in two lists whose lengths differ by at most 1. The *conquer* phase combines two (now sorted) sublists to a sorted list in linear time.

---

**Due date: Tuesday, May 23, 2023 at 16:00**

Upload your solutions as a PDF file (`*.pdf`) and your source codes to the `Submissions` folder for this exercise on Stud.IP.