CSC1048 Computability and Complexity

Functional Programming Lab 4: More Recursion and Lists

Aim

The aim of this week's exercise is to continue to learn how to use recursion with recursive data structures such as lists.

1. Develop your own list functions.

Haskell comes with a whole set of useful list functions. The goal of this exercise is to write your own version of the following standard Haskell list functions.

myAppend :: [a] -> [a] -> [a]
Append two lists.

myHead :: [a] -> a
Extract the first element of a list, which must be non-empty.

myLast :: [a] -> a
Extract the last element of a list, which must be finite and non-empty.

myTail :: [a] -> [a]
Extract the elements after the head of a list, which must be non-empty.

myInit :: [a] -> [a]
Return all the elements of a list except the last one. The list must be non-empty.

myLength :: [a] -> Int
returns the length of a finite list as an Int.

myReverse :: [a] -> [a]
returns the elements of xs in reverse order. xs must be finite.

myConcat :: [[a]] -> [a]
Concatenate a list of lists.

mySum :: Num a => [a] -> a
computes the sum of a finite list of numbers.

myProduct :: Num a => [a] -> a
computes the product of a finite list of numbers.

myMaximum :: Ord a => [a] -> a
returns the maximum value from a list, which must be non-empty, finite, and of an ordered type.

myMinimum :: Ord a => [a] -> a
returns the minimum value from a list, which must be non-empty, finite, and of an ordered type.

myElem :: Eq a => a -> [a] -> Bool
the list membership predicate.

myDelete :: Eq a => a -> [a] -> [a]
removes the first occurrence of x from its list argument.

2. Set functions for lists

This exercise is similar to the last in that we will develop our own version of 2 standard "set-like" list functions.

myUnion :: Eq a => [a] -> [a] -> [a]
returns the list union of the two lists. Duplicates, and elements of the first list, are removed from the the second list, but if the first list contains duplicates, so will the result. . For example,

```
myUnion [1,3,5,1] [2,2,3,4] == [1,3,5,1,2,4]
```

myIntersect :: Eq a => [a] -> [a] -> [a]
returns the list intersection of two lists. For example,

```
myIntersect [1,2,3,4] [2,4,6,8] == [2,4]
```

If the first list contains duplicates, so will the result.

```
myIntersect [1,2,2,3,4] [6,4,4,2] == [2,2,4]
```