CSc 520

Principles of Programming Languages

20: Haskell — Exercises

Christian Collberg

collberg@cs.arizona.edu

Department of Computer Science

University of Arizona

List Prefix

- Write a recursive function `begin xs ys` that returns true if `xs` is a prefix of `ys`. Both lists are lists of integers. Include the type signature.

```
> begin [] []
True
> begin [1] []
False
> begin [1,2] [1,2,3,4]
True
> begin [1,2] [1,1,2,3,4]
False
> begin [1,2,3,4] [1,2]
```


List Containment

- Write a recursive function `subsequence xs ys` that returns true if `xs` occurs anywhere within `ys`. Both lists are lists of integers. Include the type signature.
- Hint: reuse `begin` from the previous exercise.

```
> subsequence [] []
True
> subsequence [1] []
False
> subsequence [1] [0,1,0]
True
> subsequence [1,2,3] [0,1,0,1,2,3,5]
True
```


Mystery

- Consider the following function:

```
mystery :: [a] -> [[a]]
mystery [] = [[]]
mystery (x:xs) = sets ++ (map (x:) sets)
                 where sets = mystery xs
```

- What would `mystery [1,2]` return? `mystery [1,2,3]`?
- What does the funtion compute?

foldr

- Explain what the following expressions involving `foldr` do:
  1. `foldr (:)  [] xs`
  2. `foldr (:)  xs ys`
  3. `foldr ( y ys -> ys ++ [y]) [] xs`

shorter

- Define a function `shorter xs ys` that returns the shorter of two lists.

```
> shorter [1,2] [1]
[1]
> shorter [1,2] [1,2,3]
[1,2]
```


stripEmpty

- Write function `stripEmpty xs` that removes all empty strings from `xs`, a list of strings.

```
> stripEmpty ["", "Hello", "", "", "World!"]
["Hello","World!"]
> stripEmpty [""]
[]
> stripeEmpty []
[]
```


merge

- Write function `merge xs ys` that takes two ordered lists `xs` and `ys` and returns an ordered list containing the elements from xs and ys, without duplicates

```
> merge [1,2] [3,4]
[1,2,3,4]
> merge [1,2,3] [3,4]
[1,2,3,4]
> merge [1,2] [1,2,4]
[1,2,4]
```

Data Types

- Consider the following type:

```
data Shape = Circle Float |
             Rectangle Float Float
```

- Define a function `shapeLength` that computes the length of the perimeter of a shape.
- Add an extra constructor to `Shape` for triangles.
- Define a function which decides whether a shape is regular: a circle is regular, a square is a regular rectangular, and being equilateral makes a triangle regular.

Function Composition

- Rewrite the expression

```
map f (map g xs)
```

so that only a single call to map is used

Reduce

- Let the Haskell function reduce be defined by

```
reduce f []     v = v
reduce f (x:xs) v = f x (reduce f xs v)
```

- Reconstruct the Haskell functions length, append, filter, and map using reduce. More precisely, complete the following schemata (in the simplest possible way):

```
mylength xs    = reduce ___ xs ___
myappend xs ys = reduce ___ xs ___
myfilter p xs  = reduce ___ xs ___
mymap f xs     = reduce ___ xs ___
```