CSc 372
Comparative Programming Languages

37: Haskell — Exercises

Christian Collberg

collberg@cs.arizona.edu

Department of Computer Science
University of Arizona

Copyright © 2004 Christian Collberg
Write a recursive function \texttt{begin} \( \text{xs} \ \text{ys} \) that returns 
true if \( \text{xs} \) is a prefix of \( \text{ys} \). Both lists are lists of integers. 
Include the type signature.

\begin{verbatim}
> 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]
\end{verbatim}
List Containment

Write a recursive function \( \text{subsequence} \ \text{xs} \ \text{ys} \) that returns true if \( \text{xs} \) occurs anywhere within \( \text{ys} \). Both lists are lists of integers. Include the type signature.

Hint: reuse \( \text{begin} \) from the previous exercise.

\[
\begin{align*}
> \ \text{subsequence} \ \text{[]} \ \text{[]} & \quad \text{True} \\
> \ \text{subsequence} \ \text{[1]} \ \text{[]} & \quad \text{False} \\
> \ \text{subsequence} \ \text{[1]} \ \text{[0,1,0]} & \quad \text{True} \\
> \ \text{subsequence} \ \text{[1,2,3]} \ \text{[0,1,0,1,2,3,5]} & \quad \text{True}
\end{align*}
\]
Consider the following function:

\[
mystery :: [a] \rightarrow [[a]]
mystery [] = [[]]
mystery (x:xs) = sets ++ (map (x:) sets) 
\]
\[
\text{where sets = mystery xs}
\]

What would \( \text{mystery [1,2]} \) return? \( \text{mystery [1,2,3]} \)?

What does the function compute?
foldr

Explain what the following expressions involving \texttt{foldr} do:

1. \texttt{foldr (:) [] xs}
2. \texttt{foldr (:) xs ys}
3. \texttt{foldr ( y ys \rightarrow ys ++ [y]) [] xs}
Define a function `shorter xs ys` that returns the shorter of two lists.

```haskell
> shorter [1,2] [1]
[1]
> shorter [1,2] [1,2,3]
[1,2]
```
Write function \texttt{stripEmpty} \texttt{xs} that removes all empty strings from \texttt{xs}, a list of strings.

\begin{verbatim}
> stripEmpty ['','Hello','','','World!']
['Hello','World!']
> stripEmpty ['']
[]
> stripEmpty []
[]
\end{verbatim}
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]` returns `[1,2,3,4]`
- `merge [1,2,3] [3,4]` returns `[1,2,3,4]`
- `merge [1,2] [1,2,4]` returns `[1,2,4]`
Consider the following type:

```haskell
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

\[ \text{map } f \ (\text{map } g \ x) \]

so that only a single call to \text{map} is used.
Let the Haskell function reduce be defined by

\[
\begin{align*}
\text{reduce } f \; \text{[]} & \; v = v \\
\text{reduce } f \; (x:xs) \; v & = f \; x \; (\text{reduce } f \; xs \; v)
\end{align*}
\]

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

\[
\begin{align*}
\text{mylength } xs & = \text{reduce } ___ \; xs \; ___ \\
\text{myappend } xs \; ys & = \text{reduce } ___ \; xs \; ___ \\
\text{myfilter } p \; xs & = \text{reduce } ___ \; xs \; ___ \\
\text{mymap } f \; xs & = \text{reduce } ___ \; xs \; ___
\end{align*}
\]