Funtional and Logic Programming
Exercise Set 3

Tom Schrijvers   Steven Keuchel
{tom.schrijvers,steven.keuchel}@ugent.be

Deadline: Tuesday, October 16th, 2012, 11:59pm

Pipelines

1. An inverse dictionary lists words sorted from the end rather than from the beginning. Write a function $idict :: String \to String$ that takes an arbitrary text as input and produces an inverse dictionary with each word separated by a space. Write *idict* as a pipeline by using Haskell's function composition or the F# pipeline operator

$$\begin{aligned}&\textbf{infixl}\ 0\ |\!>\\&(|\!>) :: b \to (b \to c) \to c\\&(|\!>) = \mathit{flip}\ (\$)\end{aligned}$$

Example

```
idict $
"The soul, in many mythological, religious, philosophical, and" ++
" psychological traditions, is the incorporeal and, in many" ++
" conceptions, immortal essence of a person, living thing, or object."
  ≡
"a and essence the of thing living psychological mythological" ++
" philosophical incorporeal immortal soul in person or is traditions" ++
" conceptions religious object many"
```

You may find some functions defined in the modules *Data.Char* [1] and *Data.List* [2] useful for this exercise. You can include these modules by writing

```
import Data.Char
import Data.List
```

at the beginning of your file.

[1] http://hackage.haskell.org/packages/archive/base/latest/doc/html/Data-Char.html
[2] http://hackage.haskell.org/packages/archive/base/latest/doc/html/Data-List.html

Folds

The *foldr* function is defined by the structure of lists. Given the pseudo definition of lists

$$
\begin{array}{l}
\textbf{data}\ [\,a\,] \\
\quad = a : [\,a\,] \\
\quad \mid\ [\,]
\end{array}
$$

the *foldr* function is derived in the following way:

- Look at the type signatures of the constructor

  $$
  \begin{array}{l}
  a \to [\,a\,] \to [\,a\,] \\
  [\,a\,]
  \end{array}
  $$

- Replace the list type with a fresh type variable

  $$
  \begin{array}{l}
  a \to b \to b \\
  b
  \end{array}
  $$

- Take values of these types as arguments and recursively replace each constructor in a list with the corresponding value

  $$
  \begin{array}{l}
  \mathit{foldr} :: (a \to b \to b) \to b \to [\,a\,] \to b \\
  \mathit{foldr}\ \mathit{cons}\ \mathit{nil} = \mathit{go} \\
  \quad \textbf{where} \\
  \qquad \mathit{go}\ (a : \mathit{as}) = a\ `\mathit{cons}`\ \mathit{go}\ \mathit{as} \\
  \qquad \mathit{go}\ [\,] \qquad\quad\ = \mathit{nil}
  \end{array}
  $$

2. Derive a fold *foldExp* for the arithmetic expressions from exercise set 1

$$
\begin{array}{l}
\textbf{data}\ \mathit{Exp} = \mathit{Const}\ \mathit{Int} \\
\qquad\qquad\ \mid\ \mathit{Add}\ \mathit{Exp}\ \mathit{Exp} \\
\qquad\qquad\ \mid\ \mathit{Sub}\ \ \mathit{Exp}\ \mathit{Exp} \\
\qquad\qquad\ \mid\ \mathit{Mul}\ \mathit{Exp}\ \mathit{Exp} \\
\quad \textbf{deriving}\ \mathit{Show}
\end{array}
$$

and reimplement the interpreter *eval* and compiler *compile* in terms of *foldExp*. You may base your code on the example solutions for exercise set 1 you find on Minerva to do this exercise.

Equational reasoning

3. Prove the following statement using equational reasoning

$$\forall (\mathit{xs} :: [\,a\,])\ ((\mathit{ys} :: [\,a\,])).\quad \mathit{length}\ (\mathit{xs} +\!\!+\ \mathit{ys})\ \equiv\ \mathit{length}\ \mathit{xs} + \mathit{length}\ \mathit{ys}$$

You may use the fact that $(+)$ is associative and that 0 is the neutral element of $(+)$.

Functions as data

A language with first-class functions can not only abstract from recursion patterns with *map*s or *fold*s, but also represent data structures as functions and operate on them. In this exercise we will implement associative maps using functions of the following type

$$\textbf{type}\ \mathit{Map}\ k\ v = k \rightarrow \mathit{Maybe}\ v$$
$$\textbf{data}\ \mathit{Maybe}\ a = \mathit{Nothing} \mid \mathit{Just}\ a$$

where the *Maybe* datatype from the *Prelude* represents the existence of an association (*Just*) or the absence (*Nothing*). The empty associative map is the constant *Nothing* function

$$\mathit{empty} :: \mathit{Map}\ k\ v$$
$$\mathit{empty}\ k = \mathit{Nothing}$$

Looking up a value in the associative map is done using function application

$$\mathit{lookupMap} :: k \rightarrow \mathit{Map}\ k\ v \rightarrow \mathit{Maybe}\ v$$
$$\mathit{lookupMap}\ k\ m = m\ k$$

4. Implement the following operations on the functional representation of associative maps. You may have to perform case splitting with **case** ... **of**.

(a) $\mathit{member} :: k \rightarrow \mathit{Map}\ k\ v \rightarrow \mathit{Bool}$

(b) $\mathit{insertMap} :: \mathit{Eq}\ k \Rightarrow k \rightarrow v \rightarrow \mathit{Map}\ k\ v \rightarrow \mathit{Map}\ k\ v$

(c) $\mathit{fromList} :: \mathit{Eq}\ k \Rightarrow [(k, v)] \rightarrow \mathit{Map}\ k\ v$

(d) $\mathit{delete} :: \mathit{Eq}\ k \Rightarrow k \rightarrow \mathit{Map}\ k\ v \rightarrow \mathit{Map}\ k\ v$

(e) $\mathit{filterMap} :: (v \rightarrow \mathit{Bool}) \rightarrow \mathit{Map}\ k\ v \rightarrow \mathit{Map}\ k\ v$

(f) $\mathit{union} :: \mathit{Map}\ k\ v \rightarrow \mathit{Map}\ k\ v \rightarrow \mathit{Map}\ k\ v$

The union should be left-biased.

Optional

5. As in exercise 2 derive a function *foldTree* that folds the following datatype

$$\textbf{data}\ \mathit{Tree}\ a = \mathit{Leaf}\ a \mid \mathit{Node}\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)$$
$$\quad \textbf{deriving}\ \mathit{Show}$$

and implement the following functions in terms of *foldTree*

$$\mathit{size} :: \mathit{Tree}\ a \rightarrow \mathit{Int}$$
$$\mathit{size}\ (\mathit{Leaf}\ a) \quad = 1$$
$$\mathit{size}\ (\mathit{Node}\ l\ r) = \mathit{size}\ l + \mathit{size}\ r$$

$$
\begin{aligned}
&\mathit{flatten} :: \mathit{Tree}\ a \to [\,a\,]\\
&\mathit{flatten}\ (\mathit{Leaf}\ a) \quad\ = [\,a\,]\\
&\mathit{flatten}\ (\mathit{Node}\ l\ r)\ = \mathit{flatten}\ l +\!\!+ \mathit{flatten}\ r
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{mapTree} :: (a \to b) \to \mathit{Tree}\ a \to \mathit{Tree}\ b\\
&\mathit{mapTree}\ f\ (\mathit{Leaf}\ a) \quad = \mathit{Leaf}\ (f\ a)\\
&\mathit{mapTree}\ f\ (\mathit{Node}\ l\ r) = \mathit{Node}\ (\mathit{mapTree}\ f\ l)\ (\mathit{mapTree}\ f\ r)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{join} :: \mathit{Tree}\ (\mathit{Tree}\ a) \to \mathit{Tree}\ a\\
&\mathit{join}\ (\mathit{Leaf}\ t) \quad = t\\
&\mathit{join}\ (\mathit{Node}\ l\ r) = \mathit{Node}\ (\mathit{join}\ l)\ (\mathit{join}\ r)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{subst} :: (a \to \mathit{Tree}\ b) \to \mathit{Tree}\ a \to \mathit{Tree}\ b\\
&\mathit{subst}\ f\ (\mathit{Leaf}\ a) \quad = f\ a\\
&\mathit{subst}\ f\ (\mathit{Node}\ l\ r) = \mathit{Node}\ (\mathit{subst}\ f\ l)\ (\mathit{subst}\ f\ r)
\end{aligned}
$$

6. Prove the following theorem using equational reasoning:

$$\forall (t :: \mathit{Tree}\ a).\quad \mathit{length}\ (\mathit{flatten}\ t)\ \equiv\ \mathit{size}\ t$$