DM552 exercises

Department of Mathematics and Computer Science
University of Southern Denmark

September 27, 2017

1. In the *Data.List* module, one finds some useful functions on lists:

$$
\begin{aligned}
&\mathit{nub} :: \mathit{Eq}\ a \Rightarrow [a] \rightarrow [a]\\
&\mathit{nub}\ [\,] = [\,]\\
&\mathit{nub}\ (x : \mathit{xs}) = x : \mathit{nub}\ [y \mid y \leftarrow \mathit{xs}, x \not\equiv y]\\
&\mathit{delete} :: \mathit{Eq}\ a \Rightarrow a \rightarrow [a] \rightarrow [a]\\
&\mathit{delete}\ y\ [\,] = [\,]\\
&\mathit{delete}\ y\ (x : \mathit{xs}) = \mathbf{if}\ x \equiv y\ \mathbf{then}\ \mathit{xs}\ \mathbf{else}\ x : \mathit{delete}\ y\ \mathit{xs}\\
&(\backslash\backslash) :: \mathit{Eq}\ a \Rightarrow [a] \rightarrow [a] \rightarrow [a]\\
&(\backslash\backslash) = \mathit{foldl}\ (\mathit{flip}\ \mathit{delete})\\
&\mathit{union} :: \mathit{Eq}\ a \Rightarrow [a] \rightarrow [a] \rightarrow [a]\\
&\mathit{union}\ \mathit{xs}\ \mathit{ys} = \mathit{xs} +\!\!+ (\mathit{ys} \mathbin{\backslash\backslash} \mathit{xs})\\
&\mathit{intersect} :: \mathit{Eq}\ a \Rightarrow [a] \rightarrow [a] \rightarrow [a]\\
&\mathit{intersect}\ \mathit{xs}\ \mathit{ys} = [x \mid x \leftarrow \mathit{xs}, x\ \ \mathit{elem}\ \ \mathit{ys}]
\end{aligned}
$$

   (a) For each of the functions, describe what they do, and their time complexities.

   (b) Give an alternative definition of *nub*, using *filter* instead of the list comprehesion

   (c) Give an alternative definition of *delete*, using *foldr* instead of explicit recursion

   (d) Can *delete* be defined in terms of *filter* or a list comprehension?

   (e) Give an alternative definition of $(\backslash\backslash)$, using explicit recursion instead of *foldl*

   (f) Give an alternative definition of *intersect*, using *filter* instead of the list comprehension.

2. Corresponding to the fold right for lists,

$$\begin{aligned} &foldr\ \_\ z\ [\,] = && z \\ &foldr\ f\ z\ (x : xs) = && f\ x\ (foldr\ f\ z\ xs) \end{aligned}$$

we can define a *foldTree* that folds over tree structures:

$$\begin{aligned} &\mathbf{data}\ Tree\ a = Nil \mid Node\ a\ (Tree\ a)\ (Tree\ a) \\ &foldTree\ f\ z\ Nil = z \\ &foldTree\ f\ z\ (Node\ a\ l\ r) = f\ a\ (foldTree\ f\ z\ l)\ (foldTree\ f\ z\ r) \end{aligned}$$

- What is the type of *foldTree*?
- Give functions f such that
  - $height = foldTree\ f\ 0 :: Tree\ a \to Int$
  - $size = foldTree\ f\ 0 :: Tree\ a \to Int$
  - $treesum = foldTree\ f\ 0 :: Num\ a \Rightarrow Tree\ a \to a$
  - $flatten = foldTree\ f\ [\,] :: Tree\ a \to [a]$

3. For a type constructor F to be a functor, there should be defined a function *fmap*

$$fmap :: (a \to b) \to F\ a \to F\ b$$

such that

$$\begin{aligned} &fmap\ id && = id \\ &fmap\ (f \circ g) && = fmap\ f \circ fmap\ g \end{aligned}$$

A pointful version (an η-converted version) of the above statement is

$$\begin{aligned} &fmap\ id\ h && = id\ h && = h \\ &fmap\ (f \circ g)\ h && = (fmap\ f \circ fmap\ g)\ h && = fmap\ f\ (fmap\ g\ h) \end{aligned}$$

where we used the definitions of *id* and $(\circ)$:

$$\begin{aligned} &id\ x = x \\ &(f \circ g)\ x = f\ (g\ x) \end{aligned}$$

Show that $fmap = (\circ)$ makes the type constructor

$$F\ a = r \to a$$

a functor. (HINT: Use the pointful version of the *fmap*-conditions, and the definitions of *id* and $(\circ)$.)

4. Convert the following list comprehensions to combinatory style (version using $map/filter/concat$)

$$[(x, y) \mid x \leftarrow [1\,..\,n], odd\ x, y \leftarrow [1\,..\,n]]$$
$$[(x, y) \mid x \leftarrow [1\,..\,n], y \leftarrow [1\,..\,n], odd\ x]$$

Are they equal? Compare the costs of evaluating the two expressions

5. Recall the definition of *transpose*.

$$\begin{aligned}
&transpose :: [[a]] \rightarrow [[a]] \\
&transpose\ [\,] && = [\,] \\
&transpose\ ([\,] : xss) && = transpose\ xss \\
&transpose\ ((x : xs) : xss) && = (x : [h \mid (h : _) \leftarrow xss]) \\
& && \quad : transpose\ (xs : [t \mid (_ : t) \leftarrow xss])
\end{aligned}$$

Give an alternative definition, which does not use list comprehensions.

6. The function *remdups* removes adjacant duplicates from a list. For example, $remdups\ [1, 2, 2, 3, 3, 3, 1, 1] \equiv [1, 2, 3, 1]$. Define *remdups* using either *foldl* or *foldr*. What type does it have?