1. Prove that

$$reverse\ (xs +\!\!+\ ys) = reverse\ ys +\!\!+\ reverse\ xs$$

for all finite lists xs and ys. You may assume that $(+\!\!+)$ is associative.

2. Carry out an induction proof to find out if the following equation is true.

$$head \circ map\ f = f \circ head$$

3. Recall the catersian product function $cp :: [[a]] \to [[a]]$. Give a definition of the form $cp = foldr\ f\ e$ for suitable f and e.

The rest of this exercise concerns the proof of the identity

$$length \circ cp = product \circ map\ length$$

where *product* returns the result of multiplying a list of numbers.

(a) Using the fusion theorem, express $length \circ cp$ as an instance of *foldr*.

(b) Express *map length* as an instance of *foldr*.

(c) Using the fusion theorem again, express $product \circ map\ length$ as an instance of *foldr*.

(d) Check that the two results are identical. If they aren't you definition of *cp* was wrong.

4. The first two arguments of *foldr* are replacements for the constructors

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

of lists. A fold function can be defined for any data type: just give replacements for the constructors of the data type. For example, consider

$$\textbf{data}\ Either\ a\ b = Left\ a \mid Right\ b$$

to define a fold for *Either* we have to give replacements for

$$\begin{array}{l} Left :: a \to Either\ a\ b \\ Right :: b \to Either\ a\ b \end{array}$$

that leads to

$$\begin{array}{ll} foldE & :: (a \to c) \to (b \to c) \to Either\ a\ b \to c \\ foldE\ f\ g\ (Left\ x) & = f\ x \\ foldE\ f\ g\ (Right\ x) & = g\ x \end{array}$$

The type *Either* is not a recursive data type and *foldE* is not a recursive function. In fact *foldE* is a standard prelude function, except that it is called *either* not *foldE*.

Now define fold functions for

$$\begin{array}{l} \textbf{data}\ Nat = Zero \mid Succ\ Nat \\ \textbf{data}\ NEList\ a = One\ a \mid Cons\ a\ (NEList\ a) \end{array}$$

The second declaration introduces nonempty lists.

5. Prove that

$$foldl\ f\ e\ xs = foldr\ (flip\ f)\ e\ (reverse\ xs)$$

for all finite lists xs. Also prove that

$$foldl\ (\oplus)\ e\ xs = foldr\ (\otimes)\ e\ xs$$

for all finite lists, provided that

$$\begin{aligned}&(x \otimes y) \oplus z = x \otimes (y \oplus z)\\ &e \oplus x = x \otimes e\end{aligned}$$

6. Using

$$\begin{aligned}&foldl\ f\ e\ (xs +\!\!+\ ys) = foldl\ f\ (foldl\ f\ e\ xs)\ ys\\ &foldr\ f\ e\ (xs +\!\!+\ ys) = foldr\ f\ xs\ (foldr\ f\ e\ ys)\end{aligned}$$

prove that

$$\begin{aligned}&foldl\ f\ e \circ concat = foldl\ (foldl)\ e\\ &foldr\ f\ e \circ concat = foldr\ (flip\ (foldr\ f))\ e\end{aligned}$$

7. Mathematically speaking what is the value of

$$sum\ (scanl\ (/)\ 1\ [1\,..])$$

8. Calculate the efficient definition $scanr$ from the specification

$$scanr\ f\ e = map\ (foldr\ f\ e) \circ tails$$

9. Consider the problem of computing

$$\begin{aligned}&mss :: [Int] \to Int\\ &mss = maximum \circ map\ sum \circ subseqs\end{aligned}$$

where $subseqs$ returns all the subsequences if a finite list, including the list itself:

$$\begin{aligned}&subseqs && :: [a] \to [[a]]\\ &subseqs\ [\,] && = [[\,]]\\ &subseqs\ (x : xs) && = xss +\!\!+\ map\ (x:)\ xss\\ &\quad \textbf{where}\ xss = subseqs\ xs\end{aligned}$$

Find a more efficient alternative for mss.

10. The functions one and $none$ are defined by the equations

$$\begin{aligned}&one\ [x] = x\\ &none\ x = [\,]\end{aligned}$$

Complete the right-hand side of the following identities

$$\begin{aligned}&none \circ f = ...\\ &map\ f \circ none = ...\\ &map\ f \circ one = ...\end{aligned}$$