Programming Languages: Functional Programming Practicals 7. Folds, and Fold-Fusion

Shin-Cheng Mu

Autumn 2025

1. Express the following functions by $foldr$:

   1. $all\ p$ :: List $a \to$ Bool, where $p :: a \to$ Bool.
   2. $elem\ z$ :: List $a \to$ Bool, where $z :: a$.
   3. $concat$ :: List (List a) $\to$ List a.
   4. $filter\ p$ :: List $a \to$ List a, where $p :: a \to$ Bool.
   5. $takeWhile\ p$ :: List $a \to$ List a, where $p :: a \to$ Bool.
   6. id :: List $a \to$ List a.

   In case you haven't seen them, $all\ p\ xs$ is True iff. all elements in xs satisfy p, and $elem\ z\ xs$ is True iff. x is a member of xs.

   Solution:

   1. $all\ p = foldr\ (\lambda x\ b \to p\ x \wedge b)$ True .
   2. $elem\ x = foldr\ (\lambda y\ b \to x == y \vee b)$ False ,
   3. $concat = foldr\ (+\!\!+)\ [\,]$.
   4. $filter\ p = foldr\ (\lambda x\ xs \to \mathbf{if}\ p\ x\ \mathbf{then}\ x : xs\ \mathbf{else}\ xs)\ [\,]$,
   5. $takeWhile\ p = foldr\ (\lambda x\ xs \to \mathbf{if}\ p\ x\ \mathbf{then}\ x : xs\ \mathbf{else}\ [\,])\ [\,]$,
   6. $id = foldr\ (:)\ [\,]$.

2. Given $p :: a \to$ Bool, can $dropWhile\ p$ :: List $a \to$ List a be written as a $foldr$?

Solution: No. Consider $dropWhile\ even\ [5,4,2,1]$, which ought to be $[5,4,2,1]$. Meanwhile, $dropWhile\ even\ [4,2,1] = [1]$, and the lost elements cannot be recovered.

3. Express the following functions by $foldr$:

   1. $inits :: \mathsf{List}\ a \to \mathsf{List}\ (\mathsf{List}\ a)$.
   2. $tails :: \mathsf{List}\ a \to \mathsf{List}\ (\mathsf{List}\ a)$.
   3. $perms :: \mathsf{List}\ a \to \mathsf{List}\ (\mathsf{List}\ a)$.
   4. $sublists :: \mathsf{List}\ a \to \mathsf{List}\ (\mathsf{List}\ a)$.
   5. $splits :: \mathsf{List}\ a \to \mathsf{List}\ (\mathsf{List}\ a, \mathsf{List}\ a)$.

Solution:

1. $inits = foldr\ (\lambda x\ xss \to [\,] : map\ (x:)\ xss)\ [[\,]]$.
2. $tails = foldr\ (\lambda x\ xss \to (x : head\ xss) : xss)\ [[\,]]$,
3. $perms = foldr\ (\lambda x\ xss \to concat\ (map\ (fan\ x)\ xss))\ [[\,]]$
4. $sublists = foldr\ (\lambda x\ xss \to xss +\!\!+ map\ (x:)\ xss)\ [[\,]]$
5. $splits$ can be defined by:

$$
\begin{array}{l}
splits = foldr\ spl\ [([\,],[\,])]\ , \\
\quad \textbf{where}\ spl\ x\ ((xs,ys) : zss) = \\
\qquad\qquad ([\,], x : xs +\!\!+ ys) : map\ ((x:) \times id)\ ((xs,ys) : zss)\ .
\end{array}
$$

   where $(f \times g)\ (x,y) = (f\ x, g\ y)$.

4. Prove the $foldr$-fusion theorem. To recite the theorem: given $f :: a \to b \to b$, $e :: b$, $h :: b \to c$ and $g :: a \to c \to c$, we have

$$h \cdot foldr\ f\ e = foldr\ g\ (h\ e)\ ,$$

   if $h\ (f\ x\ y) = g\ x\ (h\ y)$ for all x and y.

Solution: The aim is to prove that $h\ (foldr\ f\ e\ xs) = foldr\ g\ (h\ e)\ xs$ for all xs, assuming that $h\ (f\ x\ y) = g\ x\ (h\ y)$.

Case $xs := [\,]$:

$$
\begin{aligned}
& h\ (foldr\ f\ \ e\ [\,]) \\
= {} & h\ e \\
= {} & foldr\ g\ (h\ e)\ [\,]\ \ .
\end{aligned}
$$

Case $xs := x : xs$:

$$
\begin{aligned}
& h\ (foldr\ f\ \ e\ (x : xs)) \\
= {} & \quad \{\ \text{definition of } foldr\ \} \\
& h\ (f\ x\ (foldr\ f\ \ e\ xs)) \\
= {} & \quad \{\ \text{fusion condition: } h\ (f\ x\ y) = g\ x\ (h\ y)\ \} \\
& g\ x\ (h\ (foldr\ f\ \ e\ xs)) \\
= {} & \quad \{\ \text{induction}\ \} \\
& g\ x\ (foldr\ g\ (h\ e)\ xs) \\
= {} & \quad \{\ \text{definition of } foldr\ \} \\
& foldr\ g\ (h\ e)\ (x : xs)\ \ .
\end{aligned}
$$

5. Prove the map-fusion rule $map\ f \cdot map\ g = map\ (f \cdot g)$ by $foldr$-fusion.

Solution: Since $map\ g$ is a $foldr$, we proceed as follows:

$$
\begin{aligned}
& map\ f \cdot map\ g \\
= {} & \quad \{\ map\ g \text{ is a } foldr\ \} \\
& map\ f \cdot foldr\ (\lambda x\ ys \to g\ x : ys)\ [\,] \\
= {} & \quad \{\ foldr\text{-fusion}\ \} \\
& foldr\ (\lambda x\ ys \to f\ (g\ x) : ys)\ [\,] \\
= {} & \quad \{\ \text{definition of } map \text{ as a } foldr\ \} \\
& map\ (f \cdot g)\ \ .
\end{aligned}
$$

The fusion condition is proved below:

$$
\begin{aligned}
& map\ f\ (g\ x : ys) \\
= {} & \quad \{\ \text{definition } map\ \} \\
& f\ (g\ x) : map\ f\ \ ys\ \ .
\end{aligned}
$$

6. Prove that $sum \cdot concat = sum \cdot map\ \ sum$ by $foldr$-fusion, twice. Compare the proof with you previous proof in earlier parts of this course.

Solution:

$$
\begin{aligned}
& sum \cdot concat \\
= \; & sum \cdot foldr\ (+\!\!+)\ [\,] \\
= \; & \quad \{\ foldr\text{-fusion}\ \} \\
& foldr\ (\lambda xs\ n \to sum\ xs + n)\ 0 \\
= \; & \quad \{\ foldr\text{-}map\ \text{fusion, see Exercise 7}\ \} \\
& foldr\ (+)\ 0 \cdot map\ sum \\
= \; & sum \cdot map\ sum\ \ .
\end{aligned}
$$

Fusion conditions for the $foldr$-fusion is

$$sum\ (xs +\!\!+ ys) = sum\ xs + sum\ ys\ \ ,$$

which is the key property we needed in the early part of this term to prove the same property. We have proved the property before, by induction on xs. We omit the proof here. (Note that we can also prove it by two more $foldr$-fusion, noting that $(+\!\!+\ ys)$ is a $foldr$, and so is sum.)

See Exercise 7 for $foldr$-map fusion. The penultimate equality holds because $(+) \cdot sum = (\lambda xs\ n \to sum\ xs + n)$. Instead of $foldr$-map fusion we cal also use $foldr$ fusion alone. The fusion condition is $sum\ (sum\ xs : xss) = sum\ xs + sum\ xss$.

The $foldr$-fusion theorem captures the common pattern in these proofs. We only need to fill in the problem-dependent proofs.

7. The map fusion theorem is an instance of the $foldr$-map fusion theorem: $foldr\ f\ \ e \cdot map\ g = foldr\ (f \cdot g)\ e$.

   (a) Prove the theorem.

Solution: Since $map\ g$ is a $foldr$, we proceed as follows:

$$
\begin{aligned}
& foldr\ f\ \ e \cdot map\ g \\
= \; & \quad \{\ map\ g\ \text{is a}\ foldr\ \} \\
& foldr\ f\ \ e \cdot foldr\ (\lambda x\ ys \to g\ x : ys)\ [\,] \\
= \; & \quad \{\ foldr\text{-fusion}\ \} \\
& foldr\ (f \cdot g)\ (foldr\ f\ \ e\ [\,]) \\
= \; & \quad \{\ \text{definition of}\ foldr\ \} \\
& foldr\ (f \cdot g)\ e\ \ .
\end{aligned}
$$

The fusion condition is proved below:

$$
\begin{aligned}
& foldr\ f\ \ e\ (g\ x : ys) \\
= \; & \quad \{\ \text{definition}\ foldr\ \} \\
& f\ (g\ x)\ (foldr\ f\ \ e\ ys)\ \ .
\end{aligned}
$$

(b) Express $sum \cdot map\ (2\times)$ as a $foldr$.

Solution:

$$
\begin{aligned}
& sum \cdot map\ (2\times) \\
= {} & foldr\ (+)\ 0 \cdot map\ (2\times) \\
= {} & \quad \{\ foldr\text{-}map \text{ fusion } \} \\
& foldr\ ((+) \cdot (2\times))\ 0\ .
\end{aligned}
$$

(c) Show that $(2\times) \cdot sum$ reduces to the same $foldr$ as the one above.

Solution:

$$
\begin{aligned}
& (2\times) \cdot sum \\
= {} & (2\times) \cdot foldr\ (+)\ 0 \\
= {} & \quad \{\ foldr \text{ fusion } \} \\
& foldr\ ((+) \cdot (2\times))\ 0\ .
\end{aligned}
$$

The fusion condition is

$$
\begin{aligned}
& 2 \times (x + y) \\
= {} & \quad \{\ \text{distributivity } \} \\
& 2 \times x + 2 \times y \\
= {} & \quad \{\ \text{definition of } (\cdot) \ \} \\
& ((+) \cdot (2\times))\ x\ (2 \times y)\ .
\end{aligned}
$$

8. Prove that $map\ f\ (xs +\!\!+ ys) = map\ f\ xs +\!\!+ map\ f\ ys$ by $foldr$-fusion. **Hint**: this is equivalent to $map\ f \cdot (+\!\!+\ ys) = (+\!\!+\ map\ f\ ys) \cdot map\ f$. You may need to do (any kinds of) fusion twice.

Solution: Recall that $(+\!\!+\ ys)$ is a $foldr$. Use $foldr$ fusion and $foldr$-map fusion:

$$
\begin{aligned}
& (+\!\!+\ map\ f\ ys) \cdot map\ f \\
= {} & \quad \{\ foldr\text{-}map \text{ fusion } \} \\
& foldr\ ((:) \cdot f)\ (map\ f\ ys) \\
= {} & \quad \{\ foldr \text{ fusion } \} \\
& map\ f \cdot (+\!\!+\ ys)\ .
\end{aligned}
$$

The fusion condition of the last step is:

$$
\begin{aligned}
& map\ f\ (x : zs) \\
= {} & \quad \{\ \text{definition of } map \ \}
\end{aligned}
$$

$$
\begin{aligned}
& f\ x : map\ f\ zs \\
= \quad & \{ \text{ definition of } (\cdot) \ \} \\
& ((:) \cdot f)\ x\ (map\ f\ zs) \ .
\end{aligned}
$$

9. Prove that $length \cdot concat = sum \cdot map\ length$ by fusion.

Solution: We caculate

$$
\begin{aligned}
& length \cdot concat \\
= \ & length \cdot foldr\ (+\!\!+)\ [\,] \\
= \quad & \{ \ foldr\text{-fusion} \ \} \\
& foldr\ ((+) \cdot length)\ 0 \\
= \ & \{\!|\ sum = foldr\ (+)\ 0\ |, |\ foldr\ |-|\ map\ |\ fusion\ \!\} \\
& sum \cdot map\ length \ .
\end{aligned}
$$

The fusion condition is proved below:

$$
\begin{aligned}
& length\ (xs +\!\!+ ys) \\
= \quad & \{ \ (+\!\!+) \text{ and } (+) \text{ homorphic} \ \} \\
& length\ xs + length\ ys \\
= \quad & \{ \text{ definition of } (\cdot) \ \} \\
& ((+) \cdot length)\ xs\ (length\ ys) \ .
\end{aligned}
$$

10. Let $scanr\ f\ e = map\ (foldr\ f\ e) \cdot tails$. Construct, by $foldr$-fusion, an implementation of $scanr$ whose number of calls to f is proportional to the length of the input list.

Solution: Recall that $tails$ is a $foldr$:

$$tails = foldr\ (\lambda x\ xss \to (x : head\ xss) : xss)\ [[\,]] \ ,$$

We try to fuse $map\ (foldr\ f\ e)$ into $tails$. For the base value, notice that

$$map\ (foldr\ f\ e)\ [[\,]] = [e] \ .$$

To construct the step function, we work on the fusion condition:

$$
\begin{aligned}
& map\ (foldr\ f\ e)\ ((x : head\ xss) : xss) \\
= \quad & \{ \text{ definition of } map \ \} \\
& foldr\ f\ e\ (x : head\ xss) : map\ (foldr\ f\ e)\ xss
\end{aligned}
$$

$$
\begin{array}{ll}
= & \{ \text{ definition of } foldr \ \} \\
& f\ x\ (foldr\ f\ e\ (head\ xss)) : map\ (foldr\ f\ e)\ xss \\
= & \{\ foldr\ f\ e\ (head\ xss) = head\ (map\ (foldr\ f\ e)\ xss)\ \} \\
& \textbf{let}\ ys = map\ (foldr\ f\ e)\ xss \\
& \textbf{in}\ f\ x\ (head\ ys) : ys\ \ .
\end{array}
$$

We have therefore constructed:

$$scanr\ f\ e = foldr\ (\lambda x\ ys \to f\ x\ (head\ ys) : ys)\ [e]\ \ .$$

You may find the inductive definition easier to comprehend:

$$
\begin{array}{l}
scanr\ f\ e\ [\,] \qquad\quad = [e] \\
scanr\ f\ e\ (x : xs) = f\ x\ (head\ ys) : ys\ \ , \\
\quad \textbf{where}\ ys = scanr\ f\ e\ xs\ \ .
\end{array}
$$

11. Recall the function $binary$ :: Nat $\to$ List Nat that returns the *reversed* binary representation of a natural number, for example $binary\ 4 = [0, 0, 1]$. Also, we talked about a function $decimal$ :: List Nat $\to$ Nat that converts the representation back to a natural number.

    (a) This time, express $decimal$ using a $foldr$.

    Solution:

    $$decimal = foldr\ (\lambda d\ n \to d + 2 \times n)\ 0\ \ .$$

    (b) Recall the function $exp\ \ m\ \ n = m^n$. Use $foldr$-fusion to construct $step$ and $base$ such that

    $$exp\ m \cdot decimal = foldr\ step\ base\ \ .$$

    If the fusion succeeds, we have derived a hylomorphism computing m^n:

    $$fastexp\ m = foldr\ step\ base \cdot binary\ \ .$$

    Solution: For the base value, we have $base = exp\ m\ 0 = 1$.
    For the step function, we calculate

    $$
    \begin{array}{ll}
    & exp\ m\ (d + 2 \times n) \\
    = & \{\ \text{since } m^{x+y} = m^x \times m^y\ \}
    \end{array}
    $$

$$
\begin{aligned}
& exp\ m\ d \times exp\ m\ (2 \times n) \\
= \quad & \{\ \text{since } m^{2n} = (m^n)^2\text{, let } square\ x = x \times x\ \} \\
& exp\ m\ d \times square\ (exp\ m\ n) \\
= \quad & \{\ d \text{ is either } 0 \text{ or } 1\text{. Expand the definition}\ \} \\
& \textbf{if } d == 0 \textbf{ then } square\ (exp\ m\ n) \textbf{ else } m \times square\ (exp\ m\ n)\ .
\end{aligned}
$$

Therefore we conclude

$$
exp\ m \cdot decimal = foldr\ (\lambda d\ x \to \textbf{if } d == 0 \textbf{ then } square\ x \textbf{ else } m \times square\ x)\ 1\ .
$$

12. Express $reverse :: \text{List } a \to \text{List } a$ by a $foldr$. Let $revcat = (+\!\!+) \cdot reverse$. Express $revcat$ as a $foldr$.

Solution: $reverse = foldr\ (\lambda x\ xs \to xs +\!\!+ [x])\ [\,]$.

To fuse $(+\!\!+)$ into $reverse$, the base value is $(+\!\!+)\ [\,] = id$. To construct the step function, we try to meet the fusion condition:

$$
(+\!\!+)\ ((\lambda x\ xs \to xs +\!\!+ [x])\ x\ xs) = step\ x\ ((+\!\!+)\ xs)\ .
$$

If we calculate:

$$
\begin{aligned}
& (+\!\!+)\ ((\lambda x\ xs \to xs +\!\!+ [x])\ x\ xs) \\
& = (+\!\!+)\ (xs +\!\!+ [x])\ ,
\end{aligned}
$$

it is hard to figure out how to proceed, since $(+\!\!+)$ expects another argument. It is easier to calculate if we supply it another argument ys. We restart and calculate:

$$
\begin{aligned}
& (+\!\!+)\ ((\lambda x\ xs \to xs +\!\!+ [x])\ x\ xs)\ ys \\
& = (+\!\!+)\ (xs +\!\!+ [x])\ ys \\
& = (xs +\!\!+ [x]) +\!\!+ ys \\
= \quad & \{\ (+\!\!+) \text{ associative}\ \} \\
& xs +\!\!+ ([x] +\!\!+ ys) \\
= \quad & \{\ \text{definition of } (\cdot)\ \} \\
& (((+\!\!+)\ xs) \cdot (x:))\ ys \\
= \quad & \{\ \text{factor out } x, ((+\!\!+)\ xs), \text{ and } ys\ \} \\
& (\lambda x\ f \to f \cdot (x:))\ x\ ((+\!\!+)\ xs)\ ys\ .
\end{aligned}
$$

We conclude that

$$
revcat = foldr\ (\lambda x\ f \to f \cdot (x:))\ id\ .
$$

13. Fold on natural numbers.

    (a) The predicate $even :: \mathsf{Nat} \to \mathsf{Bool}$ yields True iff. the input is an even number. Define $even$ in terms of $foldN$.

    Solution:

    $$even = foldN\ not\ \mathsf{True}\ .$$

    (b) Express the identity function on natural numbers $id\ n = n$ in terms of $foldN$.

    Solution:

    $$id = foldN\ \mathbf{1}_+\ 0\ .$$

14. Fuse $even$ into $(+n)$. This way we get a function that checks whether a natural number is even after adding n.

    Solution: Recall that $(+n) = foldN\ \mathbf{1}_+\ n$. To fuse $even \cdot (+n)$ into one $foldN$, the base value is $even\ n$. To find out the step function, recall that $even\ (\mathbf{1}_+\ n) = not\ (even\ n)$. We may then conclude:

    $$even \cdot (+n) = foldN\ not\ (even\ n)\ .$$

15. The famous Fibonacci number is defined by:

    $$\begin{aligned} fib\ 0 &= 0 \\ fib\ 1 &= 1 \\ fib\ (2+n) &= fib\ (1+n) + fib\ n\ . \end{aligned}$$

    The definition above, when taken directly as an algorithm, is rather slow. Define $fib2\ n = (fib\ (1+n), fib\ n)$. Derive an $O(n)$ implementation of $fib2$ by fusing it with $id :: \mathsf{Nat} \to \mathsf{Nat}$.

    Solution: Recall that $id = foldN\ (\mathbf{1}_+)\ 0$. Fusing $fib2$ into id, the base value is $fib2\ 0 = (1, 0)$. To construct the step function we calculate

    $$\begin{aligned} & fib2\ (\mathbf{1}_+\ n) \\ = & (fib\ (\mathbf{1}_+\ (\mathbf{1}_+\ n)), fib\ (\mathbf{1}_+\ n)) \end{aligned}$$

$$
\begin{aligned}
&= \quad \{ \text{ definition of } \mathit{fib} \ \} \\
&\quad (\mathit{fib}\ (\mathbf{1}_+\ n) + \mathit{fib}\ n, \mathit{fib}\ (\mathbf{1}_+\ n)) \\
&= (\lambda(x,y) \to (x+y,x))\ (\mathit{fib2}\ n)\ .
\end{aligned}
$$

Therefore we conclude that

$$\mathit{fib2} = \mathit{foldN}\ (\lambda(x,y) \to (x+y,x))\ (1,0)\ .$$

16. What are the fold fusion theorems for ETree and ITree?

Solution:

$$
\begin{aligned}
h \cdot \mathit{foldIT}\ f\ e = \mathit{foldIT}\ g\ (h\ e) &\ \Leftarrow\ h\ (f\ x\ y\ z) = g\ x\ (h\ y)\ (h\ z)\ , \\
h \cdot \mathit{foldET}\ f\ k = \mathit{foldET}\ g\ (h \cdot k) &\ \Leftarrow\ h\ (f\ x\ y) = g\ (h\ x)\ (h\ y)\ .
\end{aligned}
$$