

Functional Programming Practicals 1

Shin-Cheng Mu

FLOLAC 2026

Folds and Fold-Fusion

1. Express the following functions by *foldr*:

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

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

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

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

3. Express the following functions by *foldr*:

1. $inits :: List\ a \rightarrow List\ (List\ a)$.
2. $tails :: List\ a \rightarrow List\ (List\ a)$.
3. $perms :: List\ a \rightarrow List\ (List\ a)$.
4. $sublists :: List\ a \rightarrow List\ (List\ a)$.
5. $splits :: List\ a \rightarrow List\ (List\ a, List\ a)$.

Solution:

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

$$splits = foldr\ spl\ [([], [])] ,$$

$$\text{where } spl\ x\ ((xs, ys) : zss) =$$

$$([], x : xs \# \# ys) : map\ ((x:) \times id)\ ((xs, ys) : zss) .$$

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

4. Prove the *foldr*-fusion theorem. To recite the theorem: given $f :: a \rightarrow b \rightarrow b$, $e :: b$, $h :: b \rightarrow c$ and $g :: a \rightarrow c \rightarrow c$, 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 $xs := []$:

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

Case $xs := x : xs$:

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

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

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

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

The fusion condition is proved below:

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

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

Solution:

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

Fusion conditions for the *foldr*-fusion is

$$\text{sum } (xs \ ++ \ ys) = \text{sum } xs + \text{sum } ys ,$$

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 $(+) \cdot \text{sum} = (\lambda xs n \rightarrow \text{sum } xs + n)$. Instead of *foldr-map* fusion we can also use *foldr* fusion alone. The fusion condition is $\text{sum } (\text{sum } xs : xss) = \text{sum } xs + \text{sum } xss$.

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: $\text{foldr } f \ e \cdot \text{map } g = \text{foldr } (f \cdot g) \ e$.

(a) Prove the theorem.

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

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

The fusion condition is proved below:

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

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

Solution:

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

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

Solution:

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

The fusion condition is

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

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

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

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

The fusion condition of the last step is:

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

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

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

Solution: We calculate

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

The fusion condition is proved below:

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

10. Let $\text{scanr } f\ e = \text{map } (\text{foldr } f\ e) \cdot \text{tails}$. 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*:

$$\text{tails} = \text{foldr } (\lambda x\ xss \rightarrow (x : \text{head } xss) : xss)\ [[]] ,$$

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

$$\text{map } (\text{foldr } f\ e)\ [[]] = [e] .$$

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

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

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

We have therefore constructed:

$$\textit{scanr}\ f\ e = \textit{foldr}\ (\lambda x\ ys \rightarrow f\ x\ (\textit{head}\ ys) : ys)\ [e] .$$

You may find the inductive definition easier to comprehend:

$$\begin{aligned}
\textit{scanr}\ f\ e\ [] &= [e] \\
\textit{scanr}\ f\ e\ (x : xs) &= f\ x\ (\textit{head}\ ys) : ys , \\
&\text{where } ys = \textit{scanr}\ f\ e\ xs .
\end{aligned}$$

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

(a) This time, express $\textit{decimal}$ using a \textit{foldr} .

Solution:

$$\textit{decimal} = \textit{foldr}\ (\lambda d\ n \rightarrow d + 2 \times n)\ 0 .$$

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

$$\textit{exp}\ m \cdot \textit{decimal} = \textit{foldr}\ \textit{step}\ \textit{base} .$$

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

$$\textit{fastexp}\ m = \textit{foldr}\ \textit{step}\ \textit{base} \cdot \textit{binary} .$$

Solution: For the base value, we have $\textit{base} = \textit{exp}\ m\ 0 = 1$.

For the step function, we calculate

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

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

Therefore we conclude

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

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

Solution: $\textit{reverse} = \textit{foldr } (\lambda x \ xs \rightarrow xs \ ++ \ [x]) \ []$.

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

$$(++) \ ((\lambda x \ xs \rightarrow xs \ ++ \ [x]) \ x \ xs) = \textit{step } x \ ((++) \ xs) .$$

If we calculate:

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

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:

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

We conclude that

$$\textit{revcat} = \textit{foldr } (\lambda x \ f \rightarrow f \cdot (x:)) \ \textit{id} .$$

13. Fold on natural numbers.

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

Solution:

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

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

Solution:

$$id = foldN \ 1_+ \ 0 .$$

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

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

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 $id :: \text{Nat} \rightarrow \text{Nat}$.

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

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

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

Therefore we conclude that

$$fib2 = foldN (\lambda(x, y) \rightarrow (x + y, x)) (1, 0) .$$

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

Solution:

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