

Programming Languages: Functional Programming

Practicals 7. Folds, and Fold-Fusion

Shin-Cheng Mu

Autumn 2025

1. Express the following functions by *foldr*:

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

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

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

3. Express the following functions by *foldr*:

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

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

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

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

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

6. Prove that $\text{sum} \cdot \text{concat} = \text{sum} \cdot \text{map sum}$ by *foldr*-fusion, twice. Compare the proof with your previous proof in earlier parts of this course.
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.
 - (b) Express $\text{sum} \cdot \text{map}$ ($2 \times$) as a *foldr*.
 - (c) Show that $(2 \times) \cdot \text{sum}$ reduces to the same *foldr* as the one above.
8. Prove that $\text{map } f \ (xs \ \text{++} \ ys) = \text{map } f \ xs \ \text{++} \ \text{map } f \ ys$ by *foldr*-fusion. **Hint:** this is equivalent to $\text{map } f \cdot (\text{++ } ys) = (\text{++ map } f \ ys) \cdot \text{map } f$. You may need to do (any kinds of) fusion twice.
9. Prove that $\text{length} \cdot \text{concat} = \text{sum} \cdot \text{map length}$ by fusion.
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.
11. Recall the function *binary* :: Nat \rightarrow List Nat that returns the *reversed* binary representation of a natural number, for example $\text{binary } 4 = [0, 0, 1]$. Also, we talked about a function *decimal* :: List Nat \rightarrow Nat that converts the representation back to a natural number.
 - (a) This time, express *decimal* using a *foldr*.
 - (b) Recall the function $\text{exp } m \ n = m^n$. Use *foldr*-fusion to construct *step* and *base* such that

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

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

$$\text{fastexp } m = \text{foldr } step \ base \cdot \text{binary} \ .$$
12. Express *reverse* :: List *a* \rightarrow List *a* by a *foldr*. Let $\text{revcat} = (\text{++}) \cdot \text{reverse}$. Express *revcat* as a *foldr*.
13. Fold on natural numbers.
 - (a) The predicate *even* :: Nat \rightarrow Bool yields True iff. the input is an even number. Define *even* in terms of *foldN*.
 - (b) Express the identity function on natural numbers *id* $n = n$ in terms of *foldN*.
14. Fuse *even* into $(+n)$. This way we get a function that checks whether a natural number is even after adding *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 $id :: \text{Nat} \rightarrow \text{Nat}$.

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