Exercise 2.38.

The `accumulate` procedure is also known as `fold-right`, because it combines the first element of the sequence with the result of combining all the elements to the right. There is also a `fold-left`, which is similar to `fold-right`, except that it combines elements working in the opposite direction:

```
(define (fold-left op initial sequence)
  (define (iter result rest)
    (if (null? rest)
        result
        (iter (op result (car rest))
              (cdr rest))))
  (iter initial sequence))
```

What are the values of

```
(fold-right / 1 (list 1 2 3))

(fold-left / 1 (list 1 2 3))

(fold-right list nil (list 1 2 3))

(fold-left list nil (list 1 2 3))
```

Give a property that `op` should satisfy to guarantee that `fold-right` and `fold-left` will produce the same values for any sequence.

Answer.

We can evaluate these four expressions using substitution model:

```
(fold-right / 1 (list 1 2 3))
(/ 1 (fold-right / 1 (list 2 3)))
(/ 1 (/ 2 (fold-right / 1 (list 3))))
(/ 1 (/ 2 (/ 3 (fold-right / 1 nil))))
(/ 1 (/ 2 (/ 3 1)))
(/ 1 (/ 2 3))
(/ 1 2/3)
3/2

(fold-left / 1 (list 1 2 3))
(iter 1 (list 1 2 3))
(iter 1 (list 2 3))
(iter 0.5 (list 3))
(iter 1/6 nil)
1/6

(fold-right list nil (list 1 2 3))
(list 1 (fold-right list nil (list 2 3)))
(list 1 (list 2 (fold-right list nil (list 3))))
(list 1 (list 2 (list 3 (fold-right list nil nil))))
(list 1 (list 2 (list 3 nil)))
(1 (2 (3)))

(fold-left list nil (list 1 2 3))
(iter nil (list 1 2 3))
(iter (list nil 1) (list 2 3))
(iter (list (list nil 1) 2) (list 3))
(iter (list (list (list nil 1) 2) 3) nil)
(list (list (list nil 1) 2) 3)
```

```
(((1) 2) 3)
```

A property that `op` should satisfy to guarantee that `fold-right` and `fold-left` will will produce the same values for any sequence would be commutativity.