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

3/2

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

1/6

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

(1 (2 (3 ())))

```
(fold-left list 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)

`op` needs to be commutative to guarantee that `fold-right` and `fold-left` will produce the same values for any sequence. Commutative is defined as `(op a b)` = `(op b a)` for all `a` and `b` that we are operating on. In the above examples, `/` and `list` are not commutative, because for example `(/ 1 2)` is not equal to `(/ 2 1)` and `(list 1 2)` is not equal to `(list 2 1)`. This is why `fold-right` and `fold-left` produce different results above for the same sequences, because they apply `op` in opposite directions when `op` is not commutative.

An example of an operator that is commutative is `+` because $a + b = b + a, \forall a, b \in \mathbb{R}$. As seen below, $0 + 3 + 2 + 1 = 0 + 1 + 2 + 3$. When the operator is commutative and the operands are the same, the order does not affect the result.

```
(fold-right + 0 (list 1 2 3))
(+ 1 (+ 2 (+ 3 0)))
```

6

```
(fold-left + 0 (list 1 2 3))
(iter (+ 0 1) (list 2 3))
(iter (+ 1 2) (list 3))
(iter (+ 3 3) nil)
(+ 3 3)
```

6