Folding

A função 'aplicar' abaixo recebe uma função 'f' cuja aridade é 2, um inteiro 'a', uma lista de inteiros e retorna um inteiro. 'Aplicar' aplica 'f' em 'a' e no primeiro elemento da lista. O resultado disso é armazenado em a e a fução aplicar repete o processo até que a lista esteja vazia.

```
aplicar :: (Integer -> Integer -> Integer) -> Integer -> [Integer] -> Integer
aplicar f a [] = a
aplicar f a (b:bs) = aplicar f (f a b) bs
```

De modo que é possível dizer que 'aplicar' aplica uma função a um elemento e a toda uma lista. De modo que, caso tenhamos `aplicar (+) 1 [2,3,4]`, seria o mesmo que efetuar a soma de todos os elementos: `1 + 2 + 3 + 4`. De forma análoga, caso f fosse "^" (elevado a), teríamos `(((1 ^ 2) ^ 3) ^ 4)`. E, desse último exemplo, podemos afirmar também que "aplicar", aplica a esquerda.

```
aplicarEsquerda :: (Integer -> Integer -> Integer) -> Integer -> [Integer] -> Integer
aplicarEsquerda f a [] = a
aplicarEsquerda f a (b:bs) = aplicarEsquerda f (f a b) bs
```

Mas se quiséssemos que a mesma função começasse a aplicar à direita, seria simples. Pouca coisa seria mudada:

```
aplicarDireita :: (Integer -> Integer -> Integer) -> Integer -> [Integer] -> Integer
aplicarDireita f a [] = a
aplicarDireita f a (b:bs) = f b (aplicarDireita f a bs)
```

Assim, `aplicarDireita (^) 1 [2,3,4]` seria o mesmo que `(2 ^ (3 ^ (4 ^ 1)))`. E é para isso que servem foldl e foldr. A diferença é que são mais poderosas, podendo ser aplicadas para tipos além dos Inteiros.