Accumulator Passing Style: A Reminder

Introduction to Compilers

Dr. William Harrison

University of Missouri

February 20, 2019

More Basic Functions on Lists

`reverse` reverses a list.

```
ghci> reverse [5,4,3,2,1]
[1,2,3,4,5]
```

- What is the type of `reverse`?
- How do we write `reverse` in Haskell?

An Aside on Efficiency

Here's a simple way to write `reverse` and append (++).

```
reverse :: [a] -> [a]
reverse []     = []
reverse (x:xs) = reverse xs ++ [x]
(++) :: [a] -> [a] -> [a]
[] ++ ys      = ys
(x:xs) ++ ys = x : (xs ++ ys)
```

Why is this inefficient?

An Aside on Efficiency

Here's a simple way to write `reverse` and append (++).

```
reverse :: [a] -> [a]
reverse []     = []
reverse (x:xs) = reverse xs ++ [x]
(++) :: [a] -> [a] -> [a]
[] ++ ys      = ys
(x:xs) ++ ys = x : (xs ++ ys)
```

Why is this inefficient? Here's why:

```
reverse [x0,...,xn]
  = reverse [x0,...,xn−1] ++ [xn−1]  -- (n × reverse)
      ⋮
  = [x0] ++ ... ++ [xn−1]            -- (n−1 × ++)
```

Accumulator Passing Style

This is more efficient. Why?

```
rev :: [a] -> [a]
rev xs = rev' [] xs
   where rev' :: [a] -> [a] -> [a]
         rev' acc []     = acc
         rev' acc (x:xs) = rev' (x:acc) xs
```

Accumulator Passing Style

This is more efficient. Why?

```
rev :: [a] -> [a]
rev xs = rev' [] xs
   where rev' :: [a] -> [a] -> [a]
         rev' acc []     = acc
         rev' acc (x:xs) = rev' (x:acc) xs
```

```
rev [x0,...,xn]
  = rev' [] [x0,...,xn]
  = rev' [x0] [x1,...,xn]
    ⋮
  = rev' [xn,...,x0] []
  = [xn,...,x0]              -- (n+1 × rev')
```