FUNCTIONAL PROGRAMMING 1998

MOISE GABRIEL

QUESTION 3

(a)

```
> foldr :: (a -> b -> b) -> b -> [a] -> b
> foldr f e [] = e
> foldr f e (x: xs) = f x (foldr f e xs)
```

(b)

```
ffr op = foldr (foldr op)
```

```
foldr :: (a->b->b) -> b -> [a] -> b   |
(foldr op) => op :: (a -> b -> b)     | => (foldr op) :: b -> [a] -> b

foldr :: (c -> d -> d) -> d -> [c] -> d
  ↑
  this is the first foldr in the RHS

foldr (foldr op) =>   c = b
                      d = [a] |
                      d = b   | => [a] = b => op :: (a -> [a] -> [a])
                                          (foldr op) :: [a] -> [a] -> [a]
                                  foldr (foldr op) :: [a] -> [[a]] -> [a] =>

=> (ffr op) :: [a] -> [[a]] -> [a] =>

=> ffr :: (a -> [a] -> [a]) -> [a] -> [[a]] -> [a]
```

(c)

```
> map :: (a -> b) -> [a] -> [b]
> map = foldr (\x -> ((f x) :)) []

> (++) :: [a] -> [a] -> [a]
> (++) = flip (foldr (:))

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

(d)

```
> reverse'' :: [a] -> [a]
> reverse'' = (foldr combi id xs) []
>          where combi x nxt = nxt . (x:)
```

or (what I tried on my computer)

```
> rev = flip (foldr ((flip (.)) . (:)) id) []
```

(e) We want to express foldl as a foldr. For this, we will do the following:

- we will create a function

```
> foldl2 :: (b -> a -> b) -> [a] -> b -> b
> foldl2 f xs e = foldl f e xs
```

where we have just swapped the arguments.

Now, we'll make use of the fact that if

$$\left.\begin{array}{ll} & f\ [\,] = a \\ \text{and} & \qquad (*) \\ & f\ (x:xs) = g\ x\ (f\ xs) \end{array}\right| \Rightarrow f = foldr\ g\ a$$

, and we will do this for foldl2

- `foldl2 f [] e = e  =>  foldl2 f [] = id`
- `foldl2 f (x:xs) e = foldl2 f xs (f e x)`

```
foldl2 f (x:xs) e = (\z -> foldl2 f xs (f z x)) e
foldl2 f (x:xs) = \z -> foldl2 f xs (f z x)
foldl2 f (x:xs) = (\y z -> foldl2 f xs (f z y)) x
foldl2 f (x:xs) = (\y h z -> h (f z y)) x (foldl2 f xs)
```

From (*) we deduce that

```
foldl2 f = foldr (\y h z -> h (f z y)) id
foldl2 f xs e = foldr (\y h z -> h (f z y)) id xs e
```

$$foldl\ f\ e\ xs = (foldr\ \underbrace{(\backslash y\ h\ z \to h\ (f\ z\ y))}_{=\,g}\ \underbrace{id}_{=\,a}\ xs)\ e$$

```
foldl f e xs = (foldr g a xs) e
```

(f) From my definition, we get that

$$\text{reverse}'' \; [\,] = \text{id} \; [\,] = [\,]$$

$$\begin{aligned}
\text{reverse}'' \; (x:xs) &= (\text{foldr combi id } (x:xs)) \; [\,] \\
&= \text{combi } x \; (\text{foldr combi id } xs) \; [\,] \\
&= (\text{foldr combi id } xs) . (x:) \; [\,] \\
&= (\text{foldr combi id } xs) \; [x]
\end{aligned}$$

So, after each step, we cons the first element in the current list to the list we form, starting from [ ]. As (:) is done in constant time, my definition for reverse" is linear in the size of the list.