COMP0020 Functional Programming

Lecture 10

Higher Order Functions (2)

Contents

- Discussion of solutions to "challenge"
- Capturing common forms of recursion on lists
  - The Miranda functions foldr and foldl
    - (reduce and accumulate)
  - Examples : summing a list of numbers
- Combining HOFs

Challenge

- Can you write a function that takes a list of numbers (containing only the values 1, 2 and 0, where at least one 0 must occur) and returns a three-tuple containing :
  - The number of 1s before the first 0
  - The number of 2s before the first 0
  - The length of the longest sequence of 1s before the first 0
- Notes :
  - The value of this challenge is NOT in knowing the answer — the value is in the process of finding the answer ! So please don't "cheat" yourself by searching for the answer or looking at somebody else's answer.
  - Start by writing down the type of the function (always!)
  - Be prepared to write small "helper" functions, or look in the online manual (Section 28) for built-in operators.
  - If you find this easy, try designing the program a different way so that it makes use of higher order functions (e.g. filter, dropwhile, takewhile)

Solution 1

```
challenge :: [num]  -> (num, num, num)
challenge  any  =  xchallenge any  (0, 0, 0, 0)
```

```
xchallenge  (0 : xs)  (a, b, c, d)    =  (a, b, max [c, d])
xchallenge  (1 : xs)  (a, b, c, d)    =  xchallenge xs (a + 1, b, c, d + 1)
xchallenge  (2 : xs)  (a, b, c, d)    =  xchallenge xs (a, b + 1, max [c, d], 0)
xchallenge  any       (a, b, c, d)    =  error "bad input format"
```

Solution 2

```
challenge any = (a, b, c)
                where
                segment = takewhile (~= 0) any
                a = # (filter (= 1) segment)
                b = # (filter (= 2) segment)
                c = f segment 0 0

f []       longest current = max [longest, current]
f (2 : xs) longest current = f xs (max [current, longest]) 0
f (1 : xs) longest current = f xs longest (current + 1)
f any      longest current = error "bad input format"
```

Solution 3

```
challenge any = (a, b, c)
                where
                segment       = takewhile (~= 0) any
                a             = # (filter (= 1) segment)
                b             = # (filter (= 2) segment)
                c             = mymax (foldr g (0, 0) segment)
                g 1 (a, b)    = (a, b + 1)
                g 2 (a, b)    = (max [a, b], 0)
                g x (a, b)    = error "bad input format"
                mymax (a, b)  = a, if a > b
                              = b, otherwise
```

Foldr (sometimes known as “reduce")

Distributes a dyadic function over the elements of a list

```
foldr  (+)        0   anylist
        ↑         ↑  replaces “nil” in anylist
        ↑ replaces “cons” in anylist


Example :

foldr  (+)        0   (1 : (2 : (3 : [] )))
        →             (1 + (2 + (3 + 0 )))
        →             6
```

Foldl (sometimes known as “accumulate")

Distributes a dyadic function over the elements of a list; associates to the LEFT

foldl (+) 0 *anylist*

↑ ↑ *replaces “nil” in anylist (moved to LEFT end)*

↑ *replaces “cons” in anylist*

Exampl e :

foldl (+) 0 (1 : (2 : (3 : [])))

→ (((0 + 1) + 2) + 3)

→ 6

Definition of foldr

$$
\begin{aligned}
&\mathit{foldr} :: (* \,{-}{>}\, ** \,{-}{>}\, **) \,{-}{>}\, ** \,{-}{>}\, [*] \,{-}{>}\, ** \\
&\mathit{foldr}\ f\ \mathit{def}\ [\,] && = \mathit{def} \\
&\mathit{foldr}\ f\ \mathit{def}\ (x : xs) && = f\ x\ (\mathit{foldr}\ f\ \mathit{def}\ xs)
\end{aligned}
$$

- Note “f" is used in PREFIX position

$$
\begin{aligned}
& && \mathit{foldr}\ (+)\ 0\ (1 : (2 : (3 : [\,]))) \\
& \equiv && \mathit{foldr}\ (+)\ 0\ ((:)\ \ 1\ ((:)\ \ 2\ ((:)\ \ 3\ [\,]))) \\
& \rightarrow && ((+)\ 1\ ((+)\ 2\ ((+)\ 3\ \ 0))) \\
& \equiv && (1\ +\ (2\ +\ (3\ +\ 0)))
\end{aligned}
$$

Definition of foldl

$$\begin{aligned} &\mathit{foldl} :: (** -> * -> **) -> ** -> [*] -> ** \\ &\mathit{foldl}\ f\ \mathit{def}\ [] &&= \mathit{def} \\ &\mathit{foldl}\ f\ \mathit{def}\ (x : xs) &&= \mathit{foldl}\ f\ (f\ \mathit{def}\ x)\ xs \end{aligned}$$

- Note again “f" is used in PREFIX position

$$\begin{aligned} & \mathit{foldl}\ (+)\ 0\ \ (1 : (2 : (3 : []))) \\ \equiv\ & \mathit{foldl}\ (+)\ 0\ \ ((:)\ \ 1\ ((:)\ \ 2\ ((:)\ \ 3\ []))) \\ \rightarrow\ & \quad ((+)\ ((+)\ ((+)\ 0\ 1)\ 2)\ 3) \\ \equiv\ & \quad (((0\ +\ 1)\ +\ 2)\ +\ 3) \end{aligned}$$

Examples

- foldr (*) 1 [1,2,3,4] → 24
- foldl (+) 0 [1,2,3,4] → 10
- foldr (:) [] [1,2,3,4] → [1,2,3,4]
-

$$\begin{aligned} main &= foldr\ g\ 0\ [1,2,3,4] \\ &\quad where \\ &\quad g\ x\ y = x,\ if\ x > y \\ &\quad\qquad\ \ = y,\ otherwise \end{aligned}$$

→ 4

“Solution 3" revisited :

$$
\begin{array}{lcl}
\mathit{challenge}\ \mathit{any} & = & (a, b, c) \\
 & & \\
\mathit{where} & & \\
\mathit{segment} & = & \mathit{takewhile}\ (\ = 0)\ \mathit{any} \\
a & = & \#\ (\mathit{filter}\ (= 1)\ \mathit{segment}) \\
b & = & \#\ (\mathit{filter}\ (= 2)\ \mathit{segment}) \\
c & = & \mathit{mymax}\ (\mathit{foldr}\ g\ (0, 0)\ \mathit{segment}) \\
g\ 1\ (a, b) & = & (a,\ b + 1) \\
g\ 2\ (a, b) & = & (\mathit{max}\ [a, b],\ 0) \\
g\ x\ (a, b) & = & \mathit{error}\ \text{“}\mathit{bad\ input\ format}\text{”} \\
\mathit{mymax}\ (a, b) & = & a,\ \ \mathit{if}\ \ a > b \\
 & = & b,\ \ \mathit{otherwise}
\end{array}
$$

"Solution 3" revisited :

$$\begin{aligned} g\ 1\ (a, b) &= (a, b + 1) \\ g\ 2\ (a, b) &= (\mathit{max}\ [a, b],\ 0) \\ g\ x\ (a, b) &= \mathit{error}\ \text{``}\mathit{bad\ input\ format}\text{''} \end{aligned}$$

$\mathit{mymax}\ (\mathit{foldr}\ g\ (0, 0)\ [1, 2, 1, 1, 1, 2,])$

$\rightarrow\ \mathit{mymax}\ (g\ 1\ (\mathit{foldr}\ g\ (0, 0)\ [2, 1, 1, 1, 2,]))$

$\rightarrow\ \mathit{mymax}\ (g\ 1\ (g\ 2\ (\mathit{foldr}\ g\ (0, 0)\ [1, 1, 1, 2])))$

$\rightarrow\rightarrow\ \mathit{mymax}\ (g\ 1\ (g\ 2\ (g\ 1\ (g\ 1\ (g\ 1\ (g\ 2\ (0, 0)))))))$

$\rightarrow\ \mathit{mymax}\ (g\ 1\ (g\ 2\ (g\ 1\ (g\ 1\ (g\ 1\ (0, 0))))))$

$\rightarrow\ \mathit{mymax}\ (g\ 1\ (g\ 2\ (g\ 1\ (g\ 1\ (0, 1)))))$

$\rightarrow\ \mathit{mymax}\ (g\ 1\ (g\ 2\ (g\ 1\ (0, 2))))$

$\rightarrow\ \mathit{mymax}\ (g\ 1\ (g\ 2\ (0, 3)))$

$\rightarrow\ \mathit{mymax}\ (g\ 1\ (3, 0))$

$\rightarrow\ \mathit{mymax}\ (3, 1)$

$\rightarrow\ 3$

Equivalence of foldr and foldl ?

- Can foldr be replaced with foldl without changing the result ?
  - foldr (-) 0 [1,2,3]
  - foldl (-) 0 [1,2,3]
- Never ?
- Always ?
- Sometimes ? . . . if so, when ?

Equivalence of foldr and foldl

- They CAN be interchanged whenever the function “f" is :
  - Associative, AND
  - Commutative with the “def” value (often the identity value of the function “f”)
- Caveat – they don’t work the same on infinite lists !
  - hd (foldr (:) [] ones) → 1
  - hd (foldl (swap (:)) [] ones) → ?

Combining HOFs

$$
\begin{array}{lll}
results & = & [("fred", 45), ("sally", 79), ("chris", 65)] \\
f & = & (map\ fst)\ .\ (filter\ ((< 50).snd)) \\
main & = & f\ results \\
& \rightarrow & ((map\ fst)\ .\ (filter\ ((< 50).snd)))\ results \\
& \rightarrow & (map\ fst)\ \ ((filter\ ((< 50).snd))\ results) \\
& \rightarrow & map\ fst\ \ (filter\ ((< 50).snd)\ \ [("fred", 45), ("sally", 79), ("chris", 65)]) \\
& \rightarrow & map\ fst\ \ [("fred", 45)] \\
& \rightarrow & ["fred"]
\end{array}
$$

Combining HOFs

$$
\begin{array}{lll}
\mathit{myreverse} & = & \mathit{foldl}\ (\mathit{swap}\ (:))\ []\\
\mathit{main} & = & \mathit{myreverse}\ [1,2,3,4]\\
 & \rightarrow & (\mathit{foldl}\ (\mathit{swap}\ (:))\ [])\ [1,2,3,4]\\
 & \rightarrow & \mathit{foldl}\ (\mathit{swap}\ (:))\ []\ [1,2,3,4]\\
 & \rightarrow & \mathit{foldl}\ (\mathit{swap}\ (:))\ (\mathit{swap}\ (:)\ []\ 1)\ [2,3,4]\\
 & \rightarrow & \mathit{foldl}\ (\mathit{swap}\ (:))\ (\mathit{swap}\ (:)\ (\mathit{swap}\ (:)\ []\ 1)\ 2)\ [3,4]\\
 & \rightarrow & \mathit{foldl}\ (\mathit{swap}\ (:))\ (\mathit{swap}\ (:)\ (\mathit{swap}\ (:)(\mathit{swap}\ (:)\ []\ 1)\ 2)\ 3)\ [4]\\
 & \rightarrow & \mathit{foldl}\ (\mathit{swap}\ (:))\ (\mathit{swap}\ (:)\ (\mathit{swap}\ (:)\ (\mathit{swap}\ (:)(\mathit{swap}\ (:)\ []\ 1)\ 2)\ 3)\ 4)\ []\\
 & \rightarrow & (\mathit{swap}\ (:)\ (\mathit{swap}\ (:)\ (\mathit{swap}\ (:)(\mathit{swap}\ (:)\ []\ 1)\ 2)\ 3)\ 4)\\
 & \rightarrow & (:)\ 4\ (\mathit{swap}\ (:)\ (\mathit{swap}\ (:)(\mathit{swap}\ (:)\ []\ 1)\ 2)\ 3)\\
 & \rightarrow\rightarrow & (:)\ 4\ ((:)\ 3\ ((:)\ 2\ ((:)\ 1\ [])))\\
 & \rightarrow\rightarrow & (4\ :(3\ :\ (2\ :\ (1\ :\ [])))) \quad \equiv \quad [4,3,2,1]
\end{array}
$$

Combining HOFs

Don't go overboard : things can get silly !
What does this do ?

```
g = (foldr (+) 0) . (foldr ((:) . ((#) . (:[]))) [])
```

Summary

- Discussion of solutions to ‘challenge’
- Capturing common forms of recursion on lists
  - The Miranda functions foldr and foldl
    - ★ (reduce and accumulate)
  - Examples
- Combining HOFs

END OF LECTURE