COMP0020 Functional Programming

Lecture 14

Lists as Functions

2 : reversed lists

Motivation

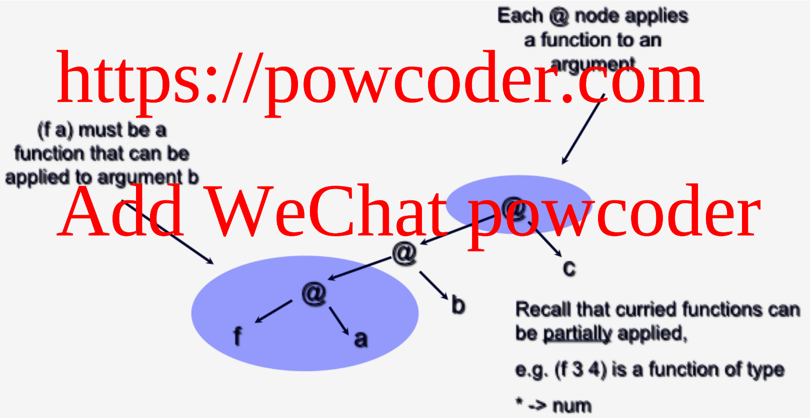
- Primarily to give a better understanding of, and facility with, higher order functions

Contents

- Preamble (repeated from last lecture)
  - Curried functions
  - Higher order functions (functions as args)
  - Higher order functions (function as result)
- Example :
  - reminder : normal lists as functions
  - new : adding elements to a list in reverse order

Preamble (1)

- Recall CURRIED functions :
  - f a b c = (a + b)
  - Can help to think of binary tree giving syntax of the function applied to its arguments :



Each @ node applies a function to an argument

(f a) must be a function that can be applied to argument b

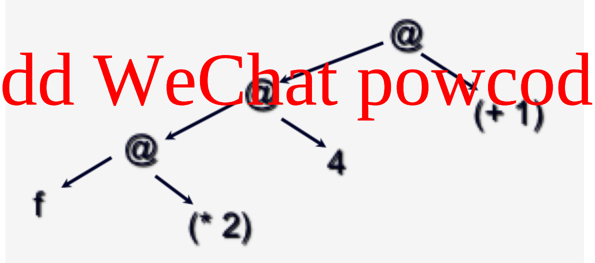
Recall that curried functions can be partially applied,

e.g. (f 3 4) is a function of type

* -> num

Preamble (2)

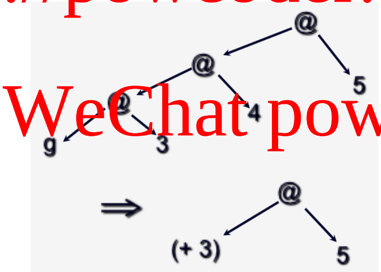
- Recall HIGHER ORDER functions :
  - f a b c = c (a b)
  - c must be a function (of at least one argument)
  - a must be a function (of at least one argument)
  - e.g. f (*2) 4 (+1)
    = (+1) ((*2) 4)
    = 1 + (2 * 4)



Preamble (3)

- Recall HIGHER ORDER functions can return functions :

  -
    ```
    g a b = (+ a)
    main = g 3 4 5
    ```
  - Too many args ?
  - No !



$$
\begin{array}{llll}
 & head & tail & isnil \\
 & \downarrow & \downarrow & \downarrow \\
cons\ a\ b\ f = f & a & b & False \\
nil\ \ f = f & (error\ "head\ of\ nil") & (error\ "tail\ of\ nil") & True
\end{array}
$$

$$
\begin{array}{lllll}
head\ x & = & x\ h & || & Example: \\
 & & where & || & z = cons\ \ 'A'\ \ nil \\
 & & h\ a\ b\ c = a & || & y = cons\ \ 'B'\ z \\
tail\ x & = & x\ t & || & \\
 & & where & || & head\ y \\
 & & t\ a\ b\ c = b & || & \rightarrow (cons\ \ 'B'\ z)\ h \\
isnil\ x & = & x\ g & || & \rightarrow cons\ \ 'B'\ z\ h \\
 & & where & || & \rightarrow h\ \ 'B'\ \ z\ False \\
 & & g\ a\ b\ c = c & || & \rightarrow 'B'
\end{array}
$$

- Consider :

  head (tail (tail (cons a (cons b nil))))
  → (tail (tail (cons a (cons b nil)))) h
  → ((tail (cons a (cons b nil)) t) h
  → (((cons a (cons b nil)) t) t) h
  → ((cons a (cons b nil) t) t) h
  → ((t a (cons b nil) False) t) h
  → ((cons b nil) t) h
  → (cons b nil t) h
  → (t b nil False) h
  → nil h
  → h (error "head of nil") (error "tail of nil") True
  → error "head of nil"

- Consider :

  isnil nil
  → nil g
  → g (error “head of nil") (error “tail of nil") True
  → True

  isnil (cons a nil)
  → (cons a nil) g
  → g a nil False
  → False

Reversed lists as functions !

- Sometimes you may find that you need to add elements to a list, but that the elements arrive in the reverse order to the order in which you wish to process them.
  - Either save in a list and then reverse the list (slow)
  - Or use a special data type that allows you to add in reverse order but creates the list the right way around !

Reversed lists as functions !

```
|| Type definition of a reversed list (called an "rlist") — it's a function!
rlist * ==            [*] -> [*]

|| Definition of rcons which adds an element to a list (c.f. (:) :: * -> [*] -> [*])
rcons ::       *      -> rlist * -> rlist *
rcons v g =           f
                      where
                      f x = g (v : x)

|| Definition of rnil    which is the empty rlist
rnil :: rlist *
rnil x = x
```

rlist (alternative definition for rcons)

$$rcons\ ::\ * -> rlist\ * -> rlist\ * \qquad ||\ \ * -> ([*] -> [*]) -> ([*] -> [*])$$

$$||\ \equiv\ * -> ([*] -> [*]) -> [*] -> [*]$$

$$rcons\ v\ g\ x\ =\ g\ (v\ :\ x)$$

NB : for function types, type expressions bracket from the right. Thus :

$$num -> num -> num -> num$$

$$\equiv num -> (num -> (num -> num))$$

rlist

```
|| Defined previously :
rlist * == [*] -> [*]
rcons v g x = g (v : x)
rnil x = x

|| Definition of rhd which gives the head of the rlist (in correct sequence!)
rhd :: rlist * -> *
rhd g = hd (g [])
```

rlist

```
|| Defined previously :

rlist * == [*] -> [*]

rcons v g x = g (v : x)

rnil x = x

rhd g = hd (g [])


|| Example :

x = (rcons 2 (rcons 3 rnil))

main = rhd x

→ hd ((rcons 2 (rcons 3 rnil)) [])

→ hd ((rcons 3 rnil) (2 : []))

→ hd (rnil (3 : (2 : [])))

→ hd (3 : (2 : []))

→ 3
```

rlist

```
|| Defined previously :
rlist * == [*] -> [*]
rcons v g x = g (v : x)
rnil x = x
rhd g = hd (g [])

|| Turning an rlist into a list
rlist2list :: rlist * -> [*]
rlist2list r = r []
```

rlist

```
|| Defined previously :
rlist * == [*] -> [*]
rcons v g x = g (v : x)
rnil x = x
rhd g = hd (g [])
rlist2list r = r []

|| Example :
main = rlist2list (rcons 'A' (rcons 'B' rnil))
→ rcons 'A' (rcons 'B' rnil) []
→ (rcons 'B' rnil) ('A' : [])
→ rnil ('B' : ('A' : []))
→ ('B' : ('A' : []))  ≡  ['B', 'A']
```

Summary

- Two Examples :
  - reminder : normal lists represented by functions
  - new : adding elements to a list in reverse order
- Motivation :
  - Primarily, to give a better understanding of, and facility with, higher order functions

END OF LECTURE