2 : reversed lists

Motivation

- Primarily to give a be

Contents

- Preamble (repeat
  - Curried func
  - Higher order f
  - Higher order f
- 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.

Preamble (2)

- Recall HIGHER ORDER functions :
  - f a b = 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 arg
  - No!

```
                                  head          tail          isnil
                                   ↓             ↓             ↓
cons a b f = f                    a             b             False
nil      f = f                    (error "head of nil")       (error "tail of nil")  True

head x = x a
tail x = x t
  where
    t a b c = b
isnil x = x g
  where
    g a b c = c
```

- 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
  → ((cons a (cons b nil) t
  → ((t a (cons b nil) False
  → ((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 n
  → True

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

Reversed lists as functions !

- Sometimes you ma
the reverse order to t
  - Either save in a lis
  - Or use a special data type that allows you to add in reverse order but creat
around !

Reversed lists as functions !


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

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

rcons v g x

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

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

rlist


```
|| Defined previously :
rlist *
rcons x g x*
rnil x

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

rlist

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

x = (rcons 2 (rcons
rnil = rhd x
→ hd ((rcons 2 (rcon
→ hd ((rcons 3 rnil) (2 : []))
→ hd (rnil (3 : (2 : [])))
→ hd (3 : (2 : []))
→ 3
```

rlist


```
|| Defined previously :
=     ( [])

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

rlist

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

|| Example :
main = rlist2list (rcons 'A
-> 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, high-