COMP0020 Functional Programming

(2)

Contents

- Structural induct
- Passing data betw
- Modes of recursion : t
- Removing mutual recursion
- Lazy evaluation : infinite lists

Induction on Lists : "append"

- The "append" function takes two lists of anything and returns a single list consisting of all the elements of the first list, followed by all the element of the second list
- Type :
  append : : ([*], [*]) -> [*
- Possible Inductio

append (*xs*

OR : *append* ((*x*

OR : *append* (*xs*

to help define the general case :

$$append\ ((x : xs),\ (y : ys))\ = ????$$

Induction on Lists : “append”

- Think about what each possible induction hypothesis would give you
- For example, if we w

$ap\quad(\ ,(\ :\ ))$ $[\ ,\ ,\ ,\ ,\ ]$ *hat help?*

$append((x : xs), ys)$ *gives* *lp?*

$append(xs, ys)$ *gives* ?

Induction on Lists : "append"

- Answer : use append
  recursion. The gen

- Or, simply :

append ((x : xs), any) = x : (append (x

Induction on Lists : "append"

- Base case (for para

- We choose the answ

$$\text{append } ([], (y : ys)) = (y : y$$

- Or, simply :

$$\text{append } ([], \text{any}) = a$$

Induction on Lists : “append”

- Final solution :

$$
\begin{aligned}
([], \quad) &= \\
\mathit{append}\ ((x : xs), \mathit{any}) &= x
\end{aligned}
$$

Passing data between functions

- A functional progr
- Focus on how data pa
- Example : insertion sort "isort"

Insertion Sort (specification)

- Define "sorted list
  - An empty list is a
  - A singleton list i
  - The list (x :xs) is s
    - x is less than all items in xs, AND
    - xs is sorted
- NB only lists of numbers

Insertion Sort (strategy)

- Start with two lists A a
- A is the input list
- B is initially empty
- One at a time, move an element from A to B
- Ensure that at all times B is sorted
  - We will need a function that can insert a number into a sorted list and retu

Insertion Sort (design)

- The list B is an accumu
  - So use accumu
- Top-down appro
  (leap of faith !)
- Then design "insert"

Insertion sort

$||$*comments . . .*

$||$*comments . . .*

$xsort :: [num] -> [num] ->$

$xsort\ []\ sorted = sorted$

$xsort\ (x : xs)\ sorted = xsort\ xs\ (\qquad\qquad)$

Insertion sort

- Code for "insert"

$$
\begin{aligned}
&[] &&= [\ ] \\
&insert\ x\ (y : ys) &&= (x : (y \\
& &&= ???
\end{aligned}
$$

Insertion sort (code 3)

- Use induction hypothesis : assume that “insert x ys” correctly inserts x into the list ys and produces the correct sorted lis

$$
\begin{aligned}
[] &= [\ ] \\
insert\ x\ (y : ys) &= (x : (y : \\
&= (y : (ins
\end{aligned}
$$

Insertion sort — full code


```
isort :: [num] -> [num]
isort any = xsort any []
```



```
xsort (x : xs) sorted = xsort xs
```



```
insert :: num -> [num] -> [num]
insert x []        = [x]
insert x (y : ys)  = (x : (y : ys)), if (x < y)
                   = (y : (insert x ys)), otherwise
```

More modes of recursion

1 : tail recursion

$$mylast\ (x : []) = x$$
$$mylast\ (x : xs) = mylast\ xs$$

More modes of recursion

2 : mutual recursion

$(\; : \;) \qquad = (\; : (\qquad))$

$xnasty :: [char] -> [char]$

$xnasty\ [] = error\ “mi$

$xnasty\ (')' : rest) = nasty\ rest$

$xnasty\ (x : xs) = xnasty\ xs$

Removing mutual recursion

$$skip :: [char] \quad -> [char]$$

$$
\begin{aligned}
&doskip :: [char] && -> [char]\\
&doskip\ [\,] && = error\ ''m\\
&doskip\ (')' : rest) && = rest\\
&doskip\ (x : xs) && = doskip\ xs
\end{aligned}
$$

Lazy evaluation : infinite lists

- Lazy evaluation of function arguments
  - Evaluate fst (24, (37 / 0))
  - Remember de
    ```
    fst : : (*,**) -> *
    fst (x,y) = x
    ```


- Lazy evaluation of d
  - Some forms of "bad" recursion may NOT result in infinite execution because lazy evaluation of data constructors means that they are evaluated ONLY AS FAR AS NE
    ```
    f : : num -> [num]
    f x = (x : (f (x + 1)))
    main = hd (tl (f 34))
    ```


  - Another example :
    ```
    ones = (1 : ones)
    main = hd (tl (tl ones))
    ```

Summary

- Structural induc
- Passing data betw
- Modes of recursion : tail recursion and mutual recursion
- Removing mutual recursion
- Lazy evaluation : infinite lists