Contents

Stack recursion answer

threes is a function that takes a list of

$$threes\ [] = 0$$
$$threes\ (3 : rest) = 1 + (thr$$
$$threes\ (x : rest) = threes\ re$$

Accumulative recursion answer

Alternative definition :

$$
\begin{aligned}
\mathit{xfours}\ ([], a) &= \\
\mathit{xfours}\ (4 : \mathit{rest}, a) &= \\
\mathit{xfours}\ ((x : \mathit{rest}), a) &=
\end{aligned}
$$

Currying

- Functions that take more than one argument
  - Either collect arguments into a tuple
  - Or use "curried" definition (named after Haskell B. Curry)
- A curried definition of a function that takes two arguments does **not** use a tuple. Compare curried function f and u

$$g\ (x, y)\ =\ x\ +\ y$$

  - compare f with the lambda expression $\lambda x.(\lambda y.(x +$
- The types of these functions are also different :

$$f :: num -> num -> num$$

$$g :: (num, num) -> num$$

- Application is also different : $(f\ 3\ 4)$ compared with $g\ (3, 4)$

Curried accumulative version


```
|| Alternative definition
fiv
fiv
```



```
xfives      []
xfives      (x :
xfives      (x :
```

Partial Applications

- Only for curried functions

$$f :: num -> num -> num$$

- We can *partiall* argument :

e

$$Miranda\ (f\ num ->$$

- We can give a name to a partial application :

$$fred = (f\ 3)$$
$$main = fred\ 4$$

Partial Applications of operators

- Operators have curried definitions and can also be partially applied.

$$-\ >$$

- Operator sections :
  - Operator pre-sections : $(3\ +)\ x \equiv (3\ +\ x)$, $(2\ *\ 5\ -)\ x \equiv (5\ -\ x)$, $(9\ /\ *\ )\ \equiv (\ *\ )\ (/\ 9)\ x \equiv (x\ /\ 9)$
  - Operator post-sections : $(+\ 3)\ x \equiv (x\ +\ 3)$,

- There is no post-section for the subtraction operator, because (-3) applies the unary minus operator to 3 to give negative 3.

Approaches to design

- Case Analysis (see t
  - consider wha
  - use PATTE
- Structural Induction (see Section 3.7.2)
  - Helps you to write the "looping" part of a recursive function

Case Analysis

- consider the function "myreverse", which takes a list of anything and returns the list with all elements reversed

$$
\begin{aligned}
&\mathit{myreverse}\ (x : []) && = (\\
&\mathit{myreverse}\ (x : (y : [])) && = (\\
&\mathit{myreverse}\ (x : (y : (z : []))) && \\
&\mathit{myreverse}\ (x : \mathit{rest}) && = ????
\end{aligned}
$$

Case Analysis : “reverse”

- To design the looping part of the function requires some thinking
- Look at the cases and t
- In this case “reverse t

(:) = ( ) ++ [ ]

- NB in the above code it is essential to put the item ecause the function ++ takes two lists as arguments. Compare this with the operator : akes an element and a list of elements.

Case Analysis : "reverse"

- Final version :

$$\begin{aligned} [] &= [] \\ myreverse\ (x : rest) &= (\end{aligned}$$

Structural Induction

- Induction versus D
- Induction versus S
  - Induction on Li
- Base Case
- Induction hypothesis
- Inductive step - you still need to do some thinking!

Induction on Lists : "startswith"

- The function "star[boilerplate] he second list starts wit
- E.g.
  - startswith ([1,2], [1,2,3,4]) returns True
  - startswith ([1,2], [2,3,4]) returns False

Induction on Lists : "startswith"

- Design Steps :
- Specify the TYPE
- Consider the Gener
  - This helps to identify the parameter of recursion !
- Consider the base case(s)

Induction on Lists : "startswith"

- Type

$$startswith \;::\; ([\,],\,[\,]) \;>\; bool$$

- General case

  - Possible induction hypotheses :

$$startswit$$
$$OR : startswit$$
$$OR : \; startswit \quad ,$$

  - Which one of the above helps us to define $startswith\;((x : xs),\;(y : ys))$?

Induction on Lists : "startswith"

- General case

$$startswith\ ((x : xs),\ (y : ys)) = True,\ if\ (x = y$$

$$= False,\ otherwis$$

Induction on Lists : "startswith"

- General case

$$startswith\ ((x : xs),\ (y :$$

$$= (x = y) \&$$

Induction on Lists : “startswith”

- Base cases(s)
  Because there are 2
  startswith ([], any)
  and
  startswith (any, [])

Induction on Lists : "startswith"

- Base case(s)
- For the first base case

$([], \quad) =$

- For the second base case the result is obvious :

$startswith\ (any, [$

Induction on Lists : "startswith"

- The final solution is :

$$\begin{aligned} \mathit{startswith}\ (\mathit{any},\ []) &= F \\ \mathit{startswith}\ (x : xs)\ (y : ys) &= (\qquad) \end{aligned}$$

Summary

- Answer to exercise
- Approaches to design
- Case analysis : exam
- Structural induction :
  - Induction hypothesis
  - Inductive step
  - Base case(s)
- Example “startswith”