COMP0020 Functional Programming

Combin ons

Contents

- Combinators
  - Definition
  - Examples : S, K, I
- Higher Order Functions
  - Definition
  - Example : composition
- Capturing common forms of recursion on lists
  - Examples : map and filter

Combinators

- Definition
  - A combinator is a function that contains no free variables
- Examples :

$$
\begin{aligned}
&id\ x &&= [illegible] && \| [illegible]' \\
&cancel\ x\ y &&= [illegible] && \| [illegible]'' \\
&swap\ f\ x\ y &&= f\ y\ x && \| ``C'' \\
&distribute\ f\ g\ x &&= f\ x\ (g\ x) && \| ``S''
\end{aligned}
$$

Combinators (2)

- Basis for implementation
- S & K computationally complete
- All required data available via arguments (passed on the stack) so stant bindings

Higher Order Functions

- Definition :
  - A Higher Order Function is a function that either (i) takes a function as an argument, or (ii) returns a function as a result, or (iii) both.

$$
\begin{aligned}
h\ x &= (+\ x) \\
j\ f\ p\ g &= (+\ (f \text{ [illegible]} \\
&= (+\ (g\ p)),\ \textit{otherwise}
\end{aligned}
$$

Higher Order Functions : types

$$
\begin{aligned}
&f \;::\; (* \;\; -> **) -> * -> ** \\
&f\ g\ x \qquad = (g\ x) \\[1em]
&h :: num \;\; -> (nu \\
&h\ x \qquad\quad = (+\ x) \\[1em]
&j :: (bool -> num) -> bool -> (bool-> num) -> (num-> num) \\
&j\ f\ \ p\ g \qquad = (+\ (f\ \ p)),\ \ if\ \ p \\
&\qquad\qquad\quad = (+\ (g\ \ p)),\ \ otherwise
\end{aligned}
$$

Function composition

$$compose :: (** -> ***) -> (* -> **) -> * -> ***$$
$$compose\ f\ g\ x\ =\ f\ (g\ x)$$

- Can partially apply "compose":

$$fred = (comp$$

$$main = fred\ ($$

- Built-in operator "."

$$fred\ =\ ((+1)\ .\ abs)$$

Function composition (2)

- twice x = x * 2
- many x = twice (twice (twice (twice x)))

- mymany : : num -> num
- mymany = (twice . twice . twice . twice)

Example HOF

- "myiterate" repeatedly applies its second parameter to its final parameter ; the final parameter is an accumulator for the result. The function loops n times, where n is the first parameter :

$$
\begin{array}{l}
\mathit{myiter}\ [illegible]\ > *\\
\mathit{myiterate}\ 0\ f\ \mathit{state} = \mathit{sta}[illegible]\\
\mathit{myiterate}\ n\ f\ \mathit{state} = \mathit{my}[illegible]\ ([illegible] - [illegible])\ f\ (f\ \mathit{state})
\end{array}
$$

Example HOF

- *printdots n* = *myiterate n* ((++) ".") ""

printdots 3

$\rightarrow$ *myiterate* 3 ((++) ".") ""

$\rightarrow$ *myiterate* 2 ((++) ".") ((++) "." "")

$\rightarrow$ *myiterate* 1 ((++) ".") ((++) "." ((++) "." ""))

$\rightarrow$ *myiterate* 0 ((++) ".") ((++) "."((++) "." ((++) "." "")))

$\rightarrow$ ((++) "." ((++) "." ((++) "." "")))

$\rightarrow$ "..."

Recursion on lists : capturing common forms

- Mapping a function across the values of a list :

$notlist\ [\,] = [\,]$

$\ldots t\ xs))$

$i \ldots [\,] = [\,]$

$inclist\ (x : xs) = (\ldots))$

$abslist\ [\,] = [\,]$

$abslist\ (x : xs) = ((abs\ x) : (abslist\ xs))$

Recursion on lists : map

- Built-in function “map" makes life easier :

$$\begin{aligned} notlist\ any &= map\ (\sim)\ any \\ inclist\ any &= map\ (+1)\ any \end{aligned}$$

- Or, simply :

$$\begin{aligned} notlist &= map\ (\sim) \\ inclist &= map\ (+1) \\ abslist &= map\ abs \end{aligned}$$

Definition of map

$$\begin{aligned} map\ f\ [\,] &= [\,] \\ map\ f\ (x : xs) &= ((f\ x) : (map\ f\ xs)) \end{aligned}$$

- So, what is the type of map ?
- Write it here (don't "cheat" yourself by asking Miranda — work it out for yourself !) :

Recursion on lists : capturing common forms

- Filtering some elements out of a list :

$$
\begin{aligned}
\textit{firsts}\ [] &= []\\
\textit{firsts}\ (& \qquad >= 70)
\end{aligned}
$$

$$
\begin{aligned}
\textit{not34}\ [] &= []\\
\textit{not34}\ (x : xs) &= (x : (\textit{not34}\ xs)),\ \textit{if}\ (x \sim= 34)\\
&= \textit{not34}\ xs,\ \textit{otherwise}
\end{aligned}
$$

Recursion on lists : filter

- Recursion on lists : filter

$$\begin{aligned} \mathit{firsts}\ \mathit{any} &= \mathit{filter}\ (>= 70)\ \mathit{any} \\ \mathit{not34}\ \mathit{any} &= \mathit{filter}\ (\quad = 34)\ \mathit{any} \end{aligned}$$

- Or, simply :

$$\begin{aligned} \mathit{firsts} &= \mathit{filter}\ (>= 70) \\ \mathit{not34} &= \mathit{filter}\ (\sim= 34) \end{aligned}$$

Definition of filter

$$
\begin{aligned}
\mathit{filter}\ p\ [\,] &= [\,]\\
\mathit{filter}\ p\ (x : xs) &= (x : (\mathit{filter}\ p\ xs)),\ \mathit{if}\ (p\ x)\\
&\ \mathit{e}
\end{aligned}
$$

- So, what is the type of filter ?
- Write it here (don't "cheat" yourself by asking Miranda — work it out for yourself !) :

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
- Notes :
  - The value of this challenge is NOT in knowing the answer — the val ing the answer ! So please don't "cheat" yourself by searching for the mebody 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)

Summary

- Combinators
  - Definition
  - Example : S, K, I
- Higher Order Functions
  - Definition
  - Example : composition
- Use of example HOF
- Capturing common forms of recursion on lists
  - Examples : map and filter