COMP0020 Functional Programming

Lecture 7

Recursion and the Lambda Calculus

Contents

Introduction

Recursive function f

$\downarrow$

Intermediate function h

$\downarrow$ $\longleftarrow$ Fixed points and the fixpoint operator Y

Definition of f in terms of Y

$\downarrow$ $\longleftarrow$ How to define Y in the lambda calculus

Final Lambda-calculus definition of f

Recursion and the lambda calculus

- Consider :

$$\begin{aligned} f\ x &= 3,\ \text{if } (x = 0) \\ &= 11,\ \text{otherwise} \end{aligned}$$

- In the lambda calculus this is :

$$\lambda\ x.(\text{if } (x = 0)\ 3\ 11)$$

Recursion and the lambda calculus

- However, now consider :

$$\begin{aligned} \text{f x} &= 3, \text{ if } (x = 0) \\ &= 1 + f(x - 1), \text{ otherwise} \end{aligned}$$

- What's wrong with this ? :

$$\lambda\ x.(\text{if } (x = 0)\ 3\ (1 + (f(x - 1))))$$

Recursion and the lambda calculus

- Consider it again in the context of the whole program :

$$\begin{aligned} \text{f x} &= 3, \text{ if } (\text{x} = 0) \\ &= 1 + \text{f(x - 1), otherwise} \\ \text{main} &= \text{f } 7 \end{aligned}$$

- This would translate to the following, which still has a problem :

$$\lambda \text{f.(f} \;\; 7) \; (\lambda \text{ x.(if (x} = 0) \; 3 \; (1 + (\text{f(x - 1)))))}$$

$$\rightarrow^{\beta} \;\; \lambda \text{ x.(if (x} = 0) \; 3 \; (1 + (\text{f(x - 1)))) } \; 7$$

Recursion and the lambda calculus

- SO how can we represent a recursive function in the lambda calculus ?

1. First define a new NON-RECURSIVE function (e.g. call it "h"), whose body is identical to that of "f" but which takes "f" as an argument,[1] as follows :

$$
\begin{aligned}
h\ f\ x &= 3, \text{ if } (x = 0) \\
&= 1 + f(x - 1), \text{ otherwise}
\end{aligned}
$$

1. NB here we are using a *curried* style of function definition.

Recursion and the lambda calculus

1. Now the following lambda expression for "h" is fine, because "f" is bound :

   λ f.(λ x.(if (x=0) 3 (1+(f(x-1)))))

   BUT "h" is not "f", so we haven't solved the problem yet !

2. However, notice that the partial application (h f) gives the same result as f, so (h f) $\equiv$ f (this is an identity, not a definition)

Recursion and the lambda calculus

- A “fixed point” (or “fixpoint”) of any function g is a value x from the input domain of g such that $(g\ x) \equiv x$
- Example 1 :

$$id\ x = x$$

  - id is called the ‘identity function’
  - Every value in the input domain of id is a fixed point of id
- Example 2 :

$$\begin{aligned} three\ x &= 3,\ if\ (x = 3) \\ &= (x + 1),\ otherwise \end{aligned}$$

  - The input value 3 is the only fixed point of the function $three$
- Note that (from the previous slide) the function f is a fixpoint of the function h because $(h\ f) \equiv f$

Recursion and the lambda calculus

- There is a special operator (called the "fixpoint operator") that we can incorporate into the λ-calculus, which will return the fixed-point of any function.
- The fixpoint operator is often given the name Y
- It takes a function as its argument (call it g) and returns a fixed point of the function g
  - Thus, by definition, $g\ (Y\ g) = (Y\ g)$
  - If the identified fixed-point of g is itself a function [2] then Y returns the "least" (i.e. the definition with the least amount of arbitrary additional information) version of that function. [3]
- So now (Y h) gives the least fixpoint of h, which we know is f (because $(h\ f) \equiv f$)

  f = Y h

2. We alredy know that in the λ calculus we can easily pass functions as arguments, and return them as results.
3. We skate over some interesting problems : what if g doesn't have a fixpoint ? can g have more than one "least" fixpoint ? This is outside the scope of this module, but further explantions are found in Stoy's excellent book *Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory* by J.E.Stoy, 1979.

Recursion and the lambda calculus

- The reduction rule for operator Y is trivial : $Y\ g\ \rightarrow\ g\ (Y\ g)$
- Now, for example (because f is the least fixpoint of h), a Normal Order reduction[4] of f 1 gives :

f 1	= (Y h) 1	because f = Y h
	= (h (Y h)) 1	because Y g $\rightarrow$ g (Y g)
	= ((λ f.(λ x.(if (x = 0) 3 (1 + (f (x - 1)))))) (Y h)) 1	from the definition of h
	= (λ x.(if (x = 0) 3 (1 + ((Y h)(x - 1))))) 1	after one β reduction
	= (if (1 = 0) 3 (1 + ((Y h)(1 - 1))))	after another β reduction
	= (if *false* 3 (1 + ((Y h)(1 - 1))))	after δ reduction of =
	= 1 + ((Y h) (1 - 1))	after δ reduction of *if* loop !
	= 1 + ((h (Y h)) (1 - 1))	because Y g $\rightarrow$ g (Y g)
	= 1 + (((λ f.(λ x.(if (x=0) 3 (1 + (f (x-1)))))) (Y h)) (1-1))	from the definition of h
	= 1 + ((λ x.(if (x = 0) 3 (1 + ((Y h)(x - 1))))) (1 - 1))	after one β reduction
	= 1 + ((if ((1 - 1) = 0) 3 (1 + ((Y h)((1 - 1) - 1)))))	after another β reduction
	= 1 + ((if (0 = 0) 3 (1 + ((Y h)((1 - 1) - 1)))))	after δ reduction of =
	= 1 + 3	after δ reduction of *if*

4. Other reduction orders may not terminate.

Recursion and the lambda calculus

- Now we have a lambda expression that defines "f"

Y (λ f.(λ x.(if (x = 0) 3 (1 + (f (x - 1))))))

- But haven't we really just shifted the problem ? — how do we define Y in the lambda calculus ?

Recursion and the lambda calculus

The real magic : self-application

$$Y \equiv \lambda \mathrm{q}.(\ (\lambda \mathrm{x}.(\mathrm{q}\ (\mathrm{x}\ \mathrm{x})))\ (\lambda \mathrm{x}.(\mathrm{q}\ (\mathrm{x}\ \mathrm{x})))\ )$$

Recursion and the lambda calculus

Example :

$$
\begin{array}{lll}
\mathrm{Y\ h} & = \lambda q.(\ (\lambda x.(q\ (x\ x)))\ (\lambda x.(q\ (x\ x)))\ )\ h & \mathit{line}_1 \\
 & = (\lambda x.(h\ (x\ x)))\ (\lambda x.(h\ (x\ x))) & \mathit{line}_2 \\
 & = h\ ((\lambda x.(h\ (x\ x)))\ (\lambda x.(h\ (x\ x)))) & \mathit{line}_3 \\
 & = h\ (Y\ h) & \text{because } \mathit{line}_3 \equiv (h\ \mathit{line}_2) \text{ and } \mathit{line}_2 \equiv Y\ h
\end{array}
$$

Recursion and the lambda calculus

Deriving a λ-calculus definition of the recursive function f without using Y :

$$
\begin{aligned}
\text{f} \quad &= (\text{Y h}) \\
&= (\text{Y } (\lambda \text{ f.}(\lambda \text{ x.(if (x}=0)\text{ 3 }(1+(\text{f(x-1)}))))))) \\
&= ((\lambda \text{q.}((\lambda \text{x.(q (x x)))} (\lambda \text{x.(q (x x)))))} (\lambda \text{ f.}(\lambda \text{ x.(if (x}=0)\text{ 3 }(1+(\text{f(x-1)})))))))
\end{aligned}
$$

Summary

- Recursive functions in Miranda
- Recursive functions in the lambda calculus
  - Binding for function name
  - Fixed point
  - Fixed point operator
  - Self-application

END OF LECTURE