

Programming in the λ -calculus: some exercises

November 16th 2007

Warm up: β -reduction

1. How many redexes are there in the term

$$(\lambda x.xxx)(\lambda x.xxx)?$$

Perform one step of β -reduction. How many redexes are there now? If there are any, reduce one. How many are there now? What can you say about the number of redexes in this term after n steps of reduction?

2. Now consider the term

$$(\lambda x.x(xx))(\lambda x.x(xx)).$$

Reduce it by a few steps. If there are choices of redex to be made, try various different options.

Programming

1. A common data structure in many programming languages is the *pair* (e.g. in Lisp, lists are really pairs consisting of a car and a cdr). Pairing should allow us to
 - put two terms M and N together to form the pair $\langle M, N \rangle$
 - retrieve the two terms from a pair, e.g. $L(\langle M, N \rangle)$ should give us M while $R(\langle M, N \rangle)$ should give us N .

Consider the λ -terms

$$\begin{aligned} P &= \lambda m.\lambda n.\lambda f.fmn \\ L &= \lambda x.\lambda y.x \\ R &= \lambda x.\lambda y.y \end{aligned}$$

Convince yourself that P , L and R can be used to simulate pairing in the λ -calculus; that is

- work out what that might mean
 - check that these terms do behave appropriately
2. A vital ingredient in programming is the *conditional* statement: lines of code like

`if B then M else N`

are essential if we want to do any worthwhile programming.

Define λ -terms C (conditional), T (true) and F (false) such that

$$\begin{aligned} C T M N &\rightarrow_{\beta}^* M \\ C F M N &\rightarrow_{\beta}^* N \end{aligned}$$

Hint This conditional operator is quite similar to a pairing operator: it picks one of M and N based on the value of the boolean. If we try using the L and R terms from above as the booleans, then we want

$$\begin{aligned} C L M N &\rightarrow_{\beta}^* L(\langle M, N \rangle) \\ C R M N &\rightarrow_{\beta}^* R(\langle M, N \rangle) \end{aligned}$$

You should be able to write a λ -term C which does that.

3. Another key ingredient are numbers. One way of encoding numbers is to use pairs like this:

$$\begin{aligned} 0 &= \lambda x.x \\ 1 &= \langle F, \lambda x.x \rangle \\ 2 &= \langle F, \langle F, \lambda x.x \rangle \rangle \\ 3 &= \langle F, \langle F, \langle F, \lambda x.x \rangle \rangle \rangle \\ &\vdots \end{aligned}$$

With this encoding, define terms S , D and Z for:

successor: Sn should reduce to $n + 1$

decrement: $D(n + 1)$ should reduce to n

zerop: $Z0$ should reduce to T while Zn should reduce to F for any other number.