Introduction to Theoretical Computer Science

Exercise Sheet: Week 9

(1) One way to code up list structures in λ-calculus is this. The list (x, y, z) is represented as a function that takes two arguments c and n, and gives back $cx(cy(czn))$; in other words, $(x, y, z) \stackrel{\mathsf{def}}{=} \lambda c.\lambda n.cx(cy(czn))$. Similarly for lists of other lengths.

Explain this construction. (*Hint:* the choice of 'c' and 'n' as letters is not random.)

Give definitions in this encoding of λ-terms representing the *nil* list, the *cons* function, and the *head* function for lists.

What happens if you call your *head* function on the *nil* list?

(2) The recursion combinator we used was

$$\mathsf{Y} \stackrel{\mathsf{def}}{=} \lambda F.(\lambda X.F(XX))(\lambda X.F(XX))$$

What happens if you try to use Y in a call-by-value evaluation strategy? Here is a different version of Y that works for call-by-value:

$$\mathsf{Y}' \stackrel{\mathsf{def}}{=} \lambda F.(\lambda X.F(\lambda Z.XXZ))(\lambda X.F(\lambda Z.XXZ))$$

This is very similar – study it, and describe what technique, mentioned in the slides, is being used to make Y' from Y. (*Hint:* a Greek letter is involved.)

(3) If t is a well-typed λ-term $t : \tau$, then it evaluates into a well-typed term $t' : \tau$. Is it true that for general terms s and s', if $s' : \tau$ and $s \xrightarrow{\beta} s'$, then $s : \tau$?