Introduction to Theoretical Computer Science

Tutorial Sheet 6

The following questions/comments are intended as prompts for discussion. Of course, you can ask/discuss about anything. Some of these topics we’ve touched on in discussion in lectures – this is an opportunity to think about them a bit more.

(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) = \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 (or car for LISPers) function.

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

(2) The recursion combinator we used was

\[ Y \overset{\text{def}}{=} \lambda F.\left(\lambda X.F(XX)\right)\left(\lambda X.F(XX)\right) \]

What happens if you try to use \(Y\) in a call-by-value evaluation strategy?

Here is the version of \(Y\) that works for call-by-value:

\[ Y' \overset{\text{def}}{=} \lambda F.\left(\lambda X.F(\lambda Z.XXZ)\right)\left(\lambda X.F(\lambda Z.XXZ)\right) \]

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 if \(t' : \tau\) and \(t \overset{\beta}{\rightarrow} t'\), then \(t : \tau\)?