Introduction to Theoretical Computer Science

Tutorial Sheet for Week 9

(1) Recall that the *Church numerals* are a way of encoding natural numbers as λ-terms, thus:
- 0 is $\lambda f.\lambda x.\ x$
- 1 is $\lambda f.\lambda x.\ f\ x$
- 2 is $\lambda f.\lambda x.\ f\ (f\ x)$
- and so on

We also have *Church booleans* which encode boolean values by choosing between two options:
- False is $\lambda a.\lambda b.\ b$
- True is $\lambda a.\lambda b.\ a$

Write a λ-term which represents a function that determines if the given argument represents an even number. That is, given a term representing a number n, it reduces to (the Church encoding of) True if the number is even, and False otherwise.

(2) 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 $c\ x\ (c\ y\ (c\ z\ n))$; in other words, $[x, y, z] \stackrel{\mathsf{def}}{=} \lambda c.\lambda n.\ c\ x\ (c\ y\ (c\ z\ n))$. 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?

(3) The recursion combinator we used was

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

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:

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

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.)

(4) 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 terms s and s', if $s' : \tau$ and $s \xrightarrow{\beta} s'$, then $s : \tau$?

(5) What types make the expression $\lambda f.\lambda g.\lambda x.\ f\ x\ (g\ x)$ well typed? (This expression is known as the S combinator.)

(6) The term $\mathbf{fix}(\lambda x{:}\mathsf{nat}.\ x)$ is well-typable. What is its type? What is the value of the term?

(7) Here is a recursive function that takes a nat and returns its Church numeral version (assuming appropriate built-in equality and arithmetic functions):

$$\mathsf{church} \equiv \lambda n{:}\mathsf{nat}.\ \mathbf{if}(=\ n\ 0)(\lambda f.\lambda x.\ x)(\lambda f.\lambda x.\ f(\mathsf{church}\ (-\ n\ 1)\ f\ x))$$

What types should we give to f and x?

Unsugar the recursive definition above into λ and $\mathbf{fix}$, and evaluate $\mathsf{church}\ 2$ using our usual call-by-name strategy. Does it evaluate all the way to the expected answer $\lambda f.\lambda x.\ f\ (f\ x)$?