Introduction to Theoretical Computer Science

Tutorial Sheet 7

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

(2) Here is a recursive function that takes a \text{nat} and returns its Church numeral version:

\[
\text{letrec } \text{church} = \lambda n: \text{nat}. \text{if}(= n 0)(\lambda f. \lambda x. x)(\lambda f. \lambda x. f(\text{church}(\neg n 1)fx)) \text{ in church}(2)
\]

What types should we give to \( f \) and \( x \)?

Unsugar the declarations into \( \lambda \) and \text{fix}, and evaluate it using our usual call-by-name strategy. Does it evaluate all the way to the expected answer \( \lambda f. \lambda x. f(fx) \)?

(3) Languages like Haskell and ML allow the creation of ‘tagged union’ or ‘variant’ types, such as

\[
\text{data IntOrBool = MyInt int | MyBool bool}
\]

Such types can equally well be added to the simply-typed \( \lambda \)-calculus, using the syntax of your choice. How would you actually do this? What would you need to add?

Now throw type variables into the mix. Suppose we allow ourselves to write \textit{type equations} such as

\[
\alpha = \text{Empty}(\text{unit}) \mid \text{Cons}(\text{nat}, \alpha)
\]

What is the? a? solution to this equation?

What about

\[
\alpha = \text{Cons}(\text{nat}, \alpha)
\]  ?