Lecture 9B: The Lambda Calculus

Michael Beeson

Lambda Calculus

There can't be a total App over $\mathbb{N}$. But over some other X?

- Just assume App is total. Amazingly, the axioms of a model of computation are not inconsistent with this assumption!
- We obtain a system equivalent to the λ-calculus introduced by Church in a 1932 publication (although discovered in 1928 when Church was 25 years old).
- Church use a binary $App(x, y)$ instead of different App functions for each number of arguments, and took λ as primitive, rather than assuming the existence of S^m_n functions (or Λ). He did not write App explicitly, but just wrote xy in place of $App(x, y)$.
- The main rule in λ-calculus is

$$(\lambda x\, t)u = t[x := u].$$

- Lower case λ is used in the λ-calculus, though technically, it is close to Kleene's Λ, in that it leads from indices to indices.

Currying

- In λ calculus, officially there is only one binary App, not one for each number of arguments.
- Functions of several arguments are handled like this: $x(y,z)$ is defined to be xyz. This is known as "currying", after Church's student Haskell Curry.
- Modulo this essentially trivial difference, the lambda calculus amounts to assuming the axioms for a model of computation, and also specifying that App is total.

Models of λ-calculus

- It is far from obvious that the λ-calculus has any models.
- If this course were longer, three or four lectures would be devoted to the lambda calculus.
- The point of those lectures would be that these axioms are consistent. That theorem is hard to prove, but very interesting.
- Its first proof was purely syntactic.
- Natural models for the lambda calculus were not discovered until half a century later.
- We won't have time to study these things.

Fixed-point theorem in λ-calculus

Theorem (Fixed-point theorem for λ-calculus)

In the lambda calculus, for every F there exists an e such that $e = Fe$.

Remark. This theorem is (of course) not true in any model of computation over $\mathbb{N}$, because the successor function has no fixed point.

Proof. Let $\omega := \lambda x\, F(xx)$. Let $e := \omega\omega$. Then e is the desired fixed point:

$$\begin{array}{rcl} e & = & \omega\omega \\ & = & (\lambda x\, F(xx))\omega \\ & = & F(\omega\omega) \\ & = & Fe \end{array}$$

That completes the proof.

Discussion of fixed-point theorem

It is probably this proof that inspired both the statement and proof of Rogers's fixed-point theorem for the Turing-computable functions. This proof is simpler and more memorable, and given this proof, it is believable that one might work out Rogers's theorem.

The fixed-point theorem for lambda calculus might well arouse the suspicion that lambda-calculus is inconsistent, because the fixed-point theorem implies that there is no term D in lambda-calculus such that $Dx \neq x$ is a theorem (for such a D has no fixed point). Hence there cannot be a way to construct definitions by cases in lambda calculus. Nevertheless, these are not the deal-breaking results they might seem at first.

λ-calculus and computability

- There is a way to define the natural numbers (the "Church numerals") in λ-calculus.
- Using the Church numerals, we can define the concept of λ-definable function from $\mathbb{N}$ to $\mathbb{N}$.
- The second important theorem about the lambda-calculus is that Kleene-Turing model of computability is embeddable in the lambda calculus, i.e., there is an App-preserving map from the Turing model into (but not onto) any model of the lambda-calculus.
- Every Turing-computable function is defined by a λ-term and, as it turns out, vice-versa.
- Thus the λ-definable functions turn out to be the same as the Turing computable functions, which as we have seen are the same as the partial recursive functions.
- The original "Church's thesis" was that every intuitively computable function is λ-definable.
- As a curious historical note, Gödel did not believe it until he learned about Turing machines.