Advanced Progamming Languages	Lecture 15
CS 6110 Spring 2015	Wed. Feb. 25, 2015

Lecture 15

Midterm reminder - Friday, March 6

Topics

1. Comments on Abstract State Machine in Dr. Rahli's lectures
   – The heart of Theory B
   – with a Theory A analogue
2. Partial recursive numerical functions (Kleene 1936, Turing 1937) and General recursive functions, Herbrand-Gödel 1932
3. Kleene Normal Form theorem (Kleene T-predicate) using primitve recursion, only the elementary subset. Related to acceptable indexing, Rogers 1967
4. Textbook material

   Church-Rosser Theorem p.163
   Typed λ-calculus, section 2.6 p.42
   Strong normalization theorem p.45
5. Deep problems remain lurking here on the Theory B side in Gödel numbering versus meta language insights. Need to ask the right question.

---

1. **Abstract State Machines for the λ-calculus**

   These machines are not only conceptually interesting, they are practical and provide the basis for functional language compilers and bootstrapping. This is Theory B semantics in practice. You can study the Krevine and Landin machines in detail in the BRICS 2003 article along with other state machines.

2. **Partial recursive functions and General recursive functions**

   Please read the Kleene material included with the lecture. He describes Gödel's insight to simplify Herbrand's idea, which is this (p.274 Kleene):

   (i) Write down any mutually recursive equations that define a total function φ recursively
   (ii) Show that they define the intended function φ by finitary means, say intuitionistic methods.

Gödel's "bold generalization" - only require (i), since showing they define a function will likely require a proof by induction, but any fintary proof will suffice. Hilbert had suggested that meta-mathematics should use only finitary means. The intuitionistic methods would be adequate, and they are what Herbrand had in mind. We will study these methods later in the course.

3. **Kleene's normal form theorem for partial recursive functions on the natural numbers** $\mathbb{N}$

   Gödel, Herbrand, Kleene and others were studying functions on the natural numbers and using natural numbers to describe the recursion equations. This idea is due to Gödel and is called *gödel numbering* or *arithmetization of syntax*. So the numbers are both the "meta language" and the object language. We leave this idea for future reading and discussion. The numerical coding is infeasible, it uses factorization of numbers!

   Kleene Normal Form Theorem (1936)
   For each $n \geq 0$ and given any recursive function $\varphi(x_1, ..., x_n)$ a number e can be found such that:

   (i) $\forall x_1, ..., x_n. \exists\, y. T_n(e, x_1, ..., x_n, y)$ and

   (ii) $\varphi(x_1, ..., x_n) = U(\mu y. T_n(e, x_1, ..., x_n, y))$

   (iii) $\forall x_1, ..., x_n, y.\, T_n(e, x_1, ..., x_n, y) \Rightarrow U(y) = \varphi(x_1, ..., x_n)$

   Where T_n and U are specific elementary recursive functions (hence primitive recursive).

   The Kalmar elementary functions are also the level 3 of the Grzegorczyk hierarchy $\mathcal{E}_0 \subseteq \mathcal{E}_1 \subseteq \mathcal{E}_2 \subseteq \mathcal{E}_3 \subseteq ...$ all contained among the primitive recursive functions. $\mathcal{E}_3$ is characterized by having these exponential functions, a^x, a^{2^x}, $a^{2^{2^x}}$, ... The $\mathcal{E}_n$ classes are defined by bounded primitive recursion. Essentially $\mathcal{E}_1$ has addition, $\mathcal{E}_2$ multiplicatoin, and $\mathcal{E}_3$ exponentiation, then $\mathcal{E}_4$ hyperexponentiation, etc.

   This normal form theorem bounds the "structural" complexity of partial recursive functions in the same way that our Abstract State Machines for the λ-calculus bound the complexity of the evaluator. There is a basic "computing engine" whose behavior is given by algorithms that the computing engine can execute. In the case of the λ-calculus, the λ-terms are the programs. In the case of general recursive functions, the algorithms are the *numerical codes* for the recursion equations.

   The Abstract State Machine and the T-predicate code up a universal machine, similar to a universal Turing Machine, in which the program is a sequence of 0's and 1's.

   The standard approach to Basic Recursive Function Theory (BRFT) is to use numbers as codes for programs and write the function computed by code i as φ_i. This indexing of the functions is said to be *acceptable* if we can define the universal function and if we can effectively move inputs into the control of the universal machine (by Kleene's s-m-n theorem). We might discuss these acceptable indexings later when we cover the Blum Size Theorem.