CS381 Fall 2000	Handout 34
November 22, 2000	

1. Reading: D. Kozen *Automata and Computability*, lecture 37

2. The main message of this lecture:

We have learned that moving pebbles around (abacus), performing symbol-by-symbol transformation of strings (Turing Machine), substituting substrings (type 0 grammars, Post systems), or allowing recursion and minimization (μ-recursivity) bring us universal models of computation. Today we will talk about yet another universal model of computability: untyped λ-calculus, based on a totally different idea of functional application/abstraction.

The usual mathematical slang uses variables in sometimes rather ambiguous way. Here is a typical example. We all remember from the calculus course that $(x^2)' = 2x$, where $'$ denotes the derivative with respect to variable x. Let us try to read this in a traditional mathematical manner as an identity that holds for any real number x, ant put $x = 1$. We get $(1^2)' = 2 \cdot 1$, i.e. $1' = 2$, and since $1'$ clearly equals to 0, we get a contradiction in mathematics $0 = 2$. The problem here occurs, in part, because of lack of clear distinction between a polynomial x^2 as a formal expression (say, a string xx), and the squaring function $x \mapsto x^2$. Such an ambiguity is tolerated in informal mathematics, but not in computer science. There is a special mechanism, called *λ-calculus*, which clarifies this matter.

The λ-calculus is based on two primitive notions. One is *application* of a function f to an argument a to get value $f(a)$. The other is *abstraction*, which forms from a term $t(x)$ the function $\lambda x.t(x)$, that assigns to argument a the value $t(a)$. In those λ -notations the above law of the derivative of squaring function becomes $(\lambda x.x^2)' = \lambda x.2x$. Note that the variable x in both parts of this equality is *bound* by λ and is not open for substitution. Moreover, bound variables can be *renamed* without changing of the meaning of a term, e.g. $(\lambda x.x^2)' = \lambda y.2y$.

There are two major classes of λ-calculi: *typed* and *untyped*. In the typed systems each variable or term has a preassigned type (class of types for polymorphic systems), for example, integers, booleans, functions integer $\mapsto$ integer, etc. Such computational systems as ML, Nuprl, Coq are based on the corresponding typed λ-calculi. From the theoretical point of view typed λ-calculus does not provide a universal computational model.

A more expressible untyped λ-calculus emulates every Turing Machine, and therefore provides a universal computational model. Such programming languages as LISP, DYLAN are based on the untyped λ-calculi. Below by λ-calculus we will mean the pure untyped λ-calculus.

Definition 34.1. The *pure λ-calculus* consists of λ-terms along with the substitution rule called *β-reduction*, and renaming bound variables rule called *α-reduction*. The λ-terms are built from the variables $\{f, g, h, u, v, w, x, y, z, \ldots\}$ according to the grammar

$M \longrightarrow x \mid (MM) \mid (\lambda x.M)$, where x stands for any variable.

Think of a λ-term fs as a functional application of the function f to input s. A step from t to $\lambda x.t(x)$ represents functional abstraction, where $\lambda x.t$ denotes the function that on input x returns $t(x)$. We say that x is bound in $\lambda x.t(x)$.

Application is not associative, and we assume some simplifying conventions about omitting '(,)'. In particular, multiple applications are conventionally associated to the left, e.g. *stuv* should be read as $(((st)u)v)$.

Example 34.2. Functions of several variables are represented by iterated abstractions. For example, a term $t(x,y)$ naturally generates a function of two variables $\lambda x.(\lambda y.t(x,y))$ which will written as $\lambda xy.t(x,y)$ for short. Here we assume the association to the right rule: $\lambda xyz.t$ means $\lambda x.(\lambda y.(\lambda z.t))$.

Example 34.3. The λ-term t defined as $\lambda fgx.f(gx)$ represents a function of x which is a composition of functions g (the inner one) and f (the outer one) in the most general setting. Now, we may use this construction outside the pure λ-calculus to generate compositions of any particular functions f and g of x. Let us take f to be a successor function $\lambda y.y+1$ and g a squaring function $\lambda z.z^2$. A composition of those particular f and g can be described as a consecutive application of t to $\lambda y.y+1$ and then to $\lambda z.z^2$:

$$(\lambda fgx.f(gx))(\lambda y.y+1)(\lambda z.z^2)$$

This term looks excessively long for just a composition of successor and squaring. Indeed, since abstraction and application naturally constitute an inverse pair of operations, we can perform some simplifications on the basis of substitution of the form $(\lambda x.h(x))u \longrightarrow h(u)$:

$(\lambda fgx.(f(gx)))(\lambda y.y+1)(\lambda z.z^2) \longrightarrow$ (substitute $(\lambda y.y+1)$ for f)

$(\lambda gx.((\lambda y.y+1)(gx)))(\lambda z.z^2) \longrightarrow$ (substitute $(\lambda z.z^2)$ for g)

$\lambda x.((\lambda y.y+1)((\lambda z.z^2)x)) \longrightarrow$ (substitute x for z in the inner rightmost subterm)

$\lambda x.((\lambda y.y+1)(x^2)) \longrightarrow$ (substitute x^2 for y in the inner subterm)

$\lambda x.(x^2+1)$ (no further reduction by substitution is possible).

Definition 34.4. The substitution rule used above in called *β-reduction*, and is formally defined as replacing a subterm of the form $(\lambda x.t(x))s$ by $t(s)$ (and renaming some bound variables in t to avoid a collision of variables, if needed, cf. Kozen, p. 264). The process of renaming bound variables is called *α-reduction*. A term is said to be in *normal form* if no β reductions apply. If a λ-term has a normal form, then such a form is unique (Kozen, p. 264-255). Not every term has a normal form. For example, $(\lambda x(xx))(\lambda x(xx))$ has no normal forms! Indeed, it is not a normal form itself, since a β-reduction of the left-hand $\lambda x(xx)$ for x is possible. However, after performing this reduction (i.e. substituting $\lambda x(xx)$ for x) we end up with the same term $(\lambda x(xx))(\lambda x(xx))$.

A fundamental property of the pure λ-calculus is that λ-terms can emulate all Turing machines, reduction sequences correspond to computation process, and normal forms correspond to halting configurations (outputs of computations). Terms without normal forms are configurations of programs which do not halt.

Definition 34.5. Representing integers in λ-calculus: Church Numerals (Kozen pp. 265-266).

Homework problem 34.1. Represent as a pure λ-term t a function of x which is a composition of a function f of two variables (the outer function) and functions g and h of x (left and right inner functions). Perform substitutions in t of $\lambda yz.(y+z^2)$, $\lambda x.\sin x$ and $\lambda x.(x+1)$ for f, g and h respectively. Find a normal form of the resulting λ-term.