1. Reading: D. Kozen *Automata and Computability*, lecture 36

2. The main message of this lecture:

*Like finite or push-down automata, Turing machines have equivalent generational counterparts: type 0 grammars, Post systems. A very different axiomatic approach is realized in $\mu$-recursivity. These (and all other) reasonable attempts to define computability led to the same class of computable functions.*

**Definition 33.1.** Type 0 grammars (or *unrestricted* grammars) are similar to context-free grammars but with productions of a more general form $\alpha \rightarrow \beta$, where $\alpha, \beta$ are arbitrary strings of terminals and nonterminals, $\alpha$ containing at least one nonterminal.

**Example 33.2.** A type 0 grammar generating the language $\{a^n b^n c^n | n \geq 1\}$. Productions:

$$S \rightarrow aSB,C \quad CB \rightarrow BC, \quad aB \rightarrow ab, \quad bB \rightarrow bb, \quad bC \rightarrow bc, \quad cC \rightarrow cc$$

A derivation of $aabbcc$: $S \xrightarrow{1} aSB,C \xrightarrow{1} aaBCBC \xrightarrow{1} aabBBCC \xrightarrow{*} aabbc.$

A derivation of $a^n b^n c^n$: $S \xrightarrow{*} a^{n-1} S(BC)^{n-1} \xrightarrow{1} a^n(BC)^n \xrightarrow{*} a^n B^n C^n \xrightarrow{*} a^n b^n c^n.$ (We leave this an exercise to show that all generated terminal strings are of the form $a^n b^n c^n$).

**Theorem 33.3.** Type 0 grammars generate exactly r.e. languages.

The proof is too long for our course. The main ideas are the following. By the Church Thesis, any type 0 grammar computation can be emulated by a Turing Machine, therefore, all type 0 languages are r.e. Conversely, type 0 grammars can encode configurations of Turing Machines.

In devising a grammar to generate a given language, we may have to be tricky. A more convenient (but not more general!) programming tool is provided by so-called *Post systems* which, like grammars, substitute strings by strings, but use *variables* to denote unspecified substrings. For example, in elementary algebra a common operation is to replace the string $(a - b)(a + b)$ whenever it occurs by the string $(a^2 - b^2)$. This string manipulation may be denoted by writing $X(a - b)(a + b)Y \rightarrow X(a^2 - b^2)Y$.

**Definition 33.4.** A Post system consists of disjoint finite sets of nonterminals $(N)$ terminals $(\Sigma)$ and variables $(V)$, a start symbol $S \in N$, and a finite set of *Post productions* of the form

$$u_0 X_1 u_1 X_2 u_2 \ldots X_n u_n \rightarrow w_0 X_i w_1 X_i w_2 \ldots X_i w_m,$$

where

a) $u_0, u_1, \ldots, u_n, w_0, w_1, \ldots, w_m \in (N \cup \Sigma)^*$,

b) $X_1, X_2, \ldots, X_n$ are variables ranging over $(N \cup \Sigma)^*$,

c) the subscripts $i_1, i_2, \ldots, i_m$ are all from $1, 2, \ldots, n$ and need not be distinct.

This production applied to $u_0 x_1 u_1 x_2 u_2 \ldots x_n u_n \in (N \cup \Sigma)^*$ produces $u_0 x_i w_1 x_i w_2 \ldots x_i w_m$. 
Example 33.5. A Post system generating \( \{a^nb^nc^n \mid n \geq 0\} \): \( \Sigma = \{a, b, c\}, N = \{S, \#\} \), \( V = \{X, Y, Z\} \), productions \( S \rightarrow \# \), \( X \# Y \# Z \rightarrow aX \# bY \# cZ \mid XYZ \). A derivation of \( a^nb^nc^n \): \( S \xrightarrow{1} \# \xrightarrow{1} a \# b \# c \xrightarrow{1} a^nb^nc^n \). It is also clear that all derived terminal strings are on the form \( a^nb^nc^n \). Indeed, an easy induction on the derivation length shows that in any derivation all strings other than the first \( S \) and the last one are of the form \( a^nb^nc^n \). The last production of the derivation strips \( \# \)'s.

Example 33.6. A Post system computing the function \( f(n) = n^2 \), represented by the set of strings 1\(^n\)-1\(^n^2\): \( \Sigma = \{1, \}, N = \{S\} \), variables \( X, Y \), productions \( S \rightarrow 1 \), \( X \cdot Y \rightarrow X \cdot Y \cdot X \cdot X \cdot 1 \). Deriving \( 3^2 = 9 \): \( S \xrightarrow{1} \cdot \xrightarrow{1} 1 \cdot \xrightarrow{1} 11 \cdot \xrightarrow{1} 1111 \). The correctness of the algorithm is justified by the formulas \( 0^2 = 0, (n+1)^2 = n^2 + 2n + 1 \).

The type 0 grammars may be regarded as special case of Post systems. Indeed, the result of applying a type 0 production \( \alpha \rightarrow \beta \) to a string \( u = x\alpha y \in (\Sigma \cup N)^* \) is \( x\beta y \) which is equal to the result of applying a Post production \( XuY \rightarrow X\beta Y \) to the same string \( u \).

Theorem 33.7. Post systems generate r.e. sets and only them.

Definition 33.8. Let \( \bar{u} = u_1 \ldots u_n \). Primitive recursion takes two functions \( h(\bar{u}), g(x, y, \bar{u}) \) and produces \( f(x, \bar{u}) \) such that \( f(0, \bar{u}) = h(\bar{u}), f(x+1, \bar{u}) = g(x, f(x, \bar{u}), \bar{u}) \). Minimization takes a (possibly partial) function \( g(y, \bar{u}) \) and produces \( f(\bar{u}) = \mu_y(g(y, \bar{u}) = 0) \) which equals to the least value \( y \) such that \( g(0, \bar{u}), g(1, \bar{u}), \ldots, g(y-1, \bar{u}) \) are all defined and \( g(y, \bar{u}) = 0 \) if such a \( y \) exists and undefined otherwise. \( \mu \)-recursive functions are obtained from the original set of functions \( s(x) = x + 1 \) (successor), \( z(x) = 0 \) (zero), \( \pi^i_1(x_1, \ldots, x_n) = x_i, 1 \leq i \leq n \) (projections) by compositions, primitive recursions and minimizations. Functions generated without minimization are called primitive recursive (p.r.). Note, that p.r. functions are total.

Example 33.9. Addition \( f(x, u) = u + x \) is primitive recursive. Indeed, take \( h(u) = \pi^1_1(u) = u, g(x, y, u) = s(y) = y + 1 \). Then \( u + 0 = u, u + (x + 1) = (u + x) + 1 \), which provides a classical definition of addition. Likewise, multiplication \( u \cdot x \) is defined by \( u \cdot 0 = 0, u \cdot (x + 1) = u \cdot x + u \). Here \( h(u) = z(u) = 0, g(x, y, u) = y + u \), therefore multiplication is also p.r. More examples of p.r. functions: the predecessor \( x - 1 \) defined by \( 0 - 1 = 0, (x + 1) - 1 = x \); proper subtraction \( u - x \) defined by \( u - 0 = u, u - (x + 1) = (u - x) - 1 \). Here is a \( \mu \)-recursive function which is not p.r. (why?): \( f(x) = \mu_y(x + y = 0) \); note, that \( f(0) = 0 \) and \( f(x) \) is undefined for all \( x \geq 1 \).

Theorem 33.10. \( \mu \)-recursive functions = Turing computable functions.

HW Problem 33.1. Build a type 0 grammar for \( \{ww \mid w \in \{a, b\}\} \)

HW Problem 33.2. Give a Post system for \( f(n) = 3^n \).

HW Problem 33.3. Prove that the function \( f \) such that \( f(n) = n \) for \( n \geq 3 \) and undefined for \( n = 0, 1, 2 \) is \( \mu \)-recursive.