1. Reading: D. Kozen Automata and Computability, Lecture 25
J. Hopcroft and J. Ullman Introduction to Automata Theory, etc., section 5.3.

2. The main message of this lecture:

**Given a Nondeterministic Pushdown Automaton one can construct a Context Free Grammar generating the language accepted by this automaton.**

The proof will provide an algorithm of converting a NPDA to an equivalent CFG.

Greibach Normal Form for specific CFLs are usually not so difficult to build. A practical suggestion: forget about the general algorithm (Lecture 21 at Kozen’s), and to use a common sense along with some easy tricks.

**Example 23.1.** A Greibach Normal Form for PAREN−{ε}. We start with the usual CFG

\[ S \rightarrow [S] \mid SS \mid [ ] \]

Converting productions starting with terminals is straightforward. Introduce a new nonterminal \( R \), replace \( S \rightarrow [S] \) by \( S \rightarrow [SR, R \rightarrow ] \), and \( S \rightarrow [ ] \) by \( S \rightarrow [R, R \rightarrow ] \). A bit more creativity is needed to transform productions starting with nonterminal, here \( S \rightarrow SS \). Whenever this production is used and \( S \) is substituted by \( SS \), the left one of those new \( S \)'s will be eventually substituted by either \( [SR, or \ [R, or again \ SS \). The latter can be obviously replaced by some substitution for the right one of those \( S \)'s. Therefore, the resulting GNF for PAREN−{ε} is

\[ S \rightarrow [R] \mid [SR] \mid [RS] \mid [SRS], \quad R \rightarrow ] \]

By Lemma 22.2, leftmost derivations in a CFG with productions \( A \rightarrow cB_1B_2 \ldots B_n, n \geq 0, c \in \Sigma \cup \{\epsilon\} \) may be regarded as simulations of a single state automaton with acceptance by empty stack.

**Example 23.2.** Consider a single state automaton \( M \) accepting by empty stack:

\[ \delta(*)a, \perp) = (*, B), \quad \delta(*, a, B) = (*, BB), \quad \delta(*, b, B) = (*, \epsilon), \quad \delta(*, \epsilon, B) = (*, \epsilon). \]

(the latter is an \( \epsilon \)-transition). According to Lemma 22.2, we have to add to the set of productions \( A \rightarrow cB_1B_2 \ldots B_n \) for each \( \delta(*, c, A) = (*, B_1B_2 \ldots B_n) \). Here the automaton \( M \) is represented by a CFG \( G \) with the set of productions

\[ \perp \rightarrow aB, \quad B \rightarrow aBB \mid b \mid \epsilon \]

A computation \((*, aaabb, \perp) \rightarrow (*, aaabb, B) \rightarrow (*, abb, BB) \rightarrow (*, b, B) \rightarrow (*, *, B) \rightarrow (*, *, \epsilon, \) is then represented by a derivation

\[ \perp \rightarrow aB \rightarrow aaBB \rightarrow aabBB \rightarrow aaabbbB \rightarrow aaabbbB \rightarrow aaabbb \]

**Corollary 23.3 (of Lemma 22.2.)** Every NPDA with one state that accepts by empty stack has an equivalent CFG
Lemma 23.4. Every NPDA can be simulated by an NPDA with one state that accepts by empty stack.

Proof. First of all, using ε-transitions, if needed, we represent an arbitrary NPDA as $M = (Q, \Sigma, \Gamma, \delta, s, \bot, \{ t \})$ with a single final state $t$ and assume that $M$ empties its stack after reaching $t$ (therefore, $M$ accepts both by final state an by empty stack). The idea behind a one state automaton $M'$ simulating $M$ is that all state information will be conveniently placed on the stack. Let $p, q$ be states from $Q$ and $A$ a nonterminal from $\Gamma$. By $\langle pAq \rangle$ we denote a task of getting from $p$ to $q$ while reading $A$ from the stack. Look at an input string $x$ as a sequence of controls $\delta_i$ provided by individual input symbols $c_i$’s. Then a command $\delta(p, c, A) = (q, \epsilon)$ indicates that the control $\delta_c$ elementary solves the task $\langle pAq \rangle$. A command $\delta(p, c, A) = (q, B)$ then replaces any task $\langle pAr \rangle$ by the task $\langle qBr \rangle$. A command $\delta(p, c, A) = (q, B_1B_2 \ldots B_n)$ decomposes any task $\langle pAr \rangle$ into a sequence of tasks $\langle qB_1q_1 \rangle, \langle q_1B_2q_2 \rangle, \ldots, \langle q_{n-1}B_nr \rangle$. The ultimate task of reaching the final state $t$ from the start state $s$ with $\bot$ as the only symbol on the stack is then $\langle s \bot t \rangle$. We consider tasks as stack symbols in a new single state automaton $M' = (\{ \ast \}, \Sigma, \Gamma', \delta', \ast, \langle s \bot t \rangle, \emptyset)$. The whole process of computation of $M$ can be represented as a sequence of elementary solutions, replacements or decompositions of the topmost stack symbol of $M'$. To conclude a formal definition of $M'$ it suffices to specify $\Gamma' = Q \times \Gamma \times Q$, and $\delta'$. The latter is defined as follows: for each transition $\delta(p, c, A) = (q, B_1B_2 \ldots B_n)$ add $\delta'(*, c, < pAr >) = (*, < qB_1q_1 >, < q_1B_2q_2 >, \ldots, < q_{n-1}B_nr >)$ for all $q_1, q_2, \ldots, q_{n-1}, r \in Q$. For $\delta(p, c, A) = (q, \epsilon)$ this reduces to $\delta'(*, c, < pAq >) = (*, \epsilon)$. A formal proof that $M'$ simulates $M$ is given on pp. 174-175 of Kozen’s book. We will consider a simple example instead.

Example 23.5. Let $M$ have the transition function $\delta(s, a, \bot) = (s, B), \delta(s, a, B) = (s, BB), \delta(s, b, B) = (t, \epsilon), \delta(t, b, B) = (t, \epsilon), \delta(t, \epsilon, B) = (t, \epsilon)$. The corresponding $\delta'$ is

$\delta'(\ast, a, < s \bot s >) = (\ast, < s Bs >), \delta'(\ast, a, < s \bot t >) = (\ast, < s Bt >),$
$\delta'(\ast, a, < sBs >) = (\ast, < sBs < sBs >), \delta'(\ast, a, < sBs >) = (\ast, < sBs < tBs >),$
$\delta'(\ast, a, < sBt >) = (\ast, < sBs < sBt >), \delta'(\ast, a, < sBt >) = (\ast, < sBt < tBt >),$
$\delta'(\ast, b, < sBt >) = (\ast, \epsilon), \delta'(\ast, b, < tBt >) = (\ast, \epsilon), \delta'(\ast, \epsilon, < tBt >) = (\ast, \epsilon).$

Tracing for $M$ on the input $aaabb$:

$$(s, aaabb, \bot) \xrightarrow{1} (s, aaabb, B) \xrightarrow{1} (s, abb, BB) \xrightarrow{1} (s, bb, BBB) \xrightarrow{1} (s, b, BB) \xrightarrow{1} (t, \epsilon, B) \xrightarrow{1} (t, \epsilon, \epsilon).$$

The corresponding tracing for $M'$:

$$\langle *, aaabb, < s \bot t > \rangle \xrightarrow{1} (\ast, aaabb, < sBt >) \xrightarrow{1} (\ast, abb, < sBt < tBt >) \xrightarrow{1} (\ast, b, < tBt < tBt >) \xrightarrow{1} (\ast, \epsilon, < tBt >) \xrightarrow{1} (\ast, \epsilon, \epsilon).$$

Homework problem 23.1. Consider your NPDA $M$ accepting palindromes from HW problem 21.2. Convert it to a single state NPDA $M'$ and write down a CFG $G$ corresponding to $M'$ by the scheme above. Generate palindrome $ababa$ in $G$. 