CS381 Fall 2000	**Handout 23**
October 23, 2000	

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$-\{\epsilon\}$. We start with the usual CFG $S \longrightarrow [S] \mid SS \mid [\ ]$. Converting productions starting with terminals is straightforward. Introduce a new nonterminal R, replace $S \longrightarrow [S]$ by $S \longrightarrow [SR,\ R \longrightarrow\ ]$, and $S \longrightarrow [\ ]$ by $S \longrightarrow [R,\ R \longrightarrow\ ]$. A bit more creativity is needed to transform productions starting with nonterminal, here $S \longrightarrow SS$. 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$-\{\epsilon\}$ is

$$S \longrightarrow [R \mid [SR \mid [RS \mid [SRS, \qquad R \longrightarrow\ ].$$

By Lemma 22.2, leftmost derivations in a CFG with productions $A \longrightarrow cB_1B_2 \dots B_n$, $n \geq 0$, $c \in \Sigma \cup \{\epsilon\}$ 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 ϵ-transition). According to Lemma 22.2, we have to add to the set of productions $A \longrightarrow cB_1B_2 \dots B_n$ for each $\delta(*, c, A) = (*, B_1B_2 \dots B_n)$. Here the automaton M is represented by a CFG G with the set of productions

$$\perp \longrightarrow aB, \quad B \longrightarrow aBB \mid b \mid \epsilon$$

A computation $(*, aaabb, \perp) \xrightarrow{1} (*, aabb, B) \xrightarrow{1} (*, abb, BB) \xrightarrow{1} (*, bb, BBB) \xrightarrow{1} (*, b, BB) \xrightarrow{1} (*, \epsilon, B) \xrightarrow{1} (*, \epsilon, \epsilon)$ is then represented by a derivation

$$\perp \xrightarrow{1} aB \xrightarrow{1} aaBB \xrightarrow{1} aaaBBB \xrightarrow{1} aaabBB \xrightarrow{1} aaabbB \xrightarrow{1} aaabb.$$

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 $M = (Q, \Sigma, \Gamma, \delta, s, \perp, \{t\})$ 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 Γ. By $<pAq>$ 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 δ_c provided by individual input symbols c's. Then a command $\delta(p, c, A) = (q, \epsilon)$ indicates that the control δ_c **elementary solves** the task $<pAq>$. A command $\delta(p, c, A) = (q, B)$ then **replaces** any task $< pAr >$ by the task $< qBr >$. A command $\delta(p, c, A) = (q, B_1 B_2 \ldots B_n)$ **decomposes** any task $<pAr>$ into a sequence of tasks $<qB_1q_1>, <q_1B_2q_2>, \ldots, <q_{n-1}B_nr>$. The ultimate task of reaching the final state t from the start state s with $\perp$ as the only symbol on the stack is then $< s\perp t >$. We consider tasks as stack symbols in a new single state automaton $M' = (\{*\}, \Sigma, \Gamma', \delta', *, < s\perp t >, \emptyset)$. 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 δ'. The latter is defined as follows: for each transition $\delta(p, c, A) = (q, B_1 B_2 \ldots B_n)$ 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 $\delta(s, a, \perp) = (s, B)$, $\delta(s, a, B) = (s, BB)$, $\delta(s, b, B) = (t, \epsilon)$, $\delta(t, b, B) = (t, \epsilon)$, $\delta(t, \epsilon, B) = (t, \epsilon)$. The corresponding δ' is

$$\begin{aligned}
&\delta'(*, a, <s\perp s>) = (*, <sBs>), \qquad \delta'(*, a, <s\perp t>) = (*, <sBt>), \\
&\delta'(*, a, <sBs>) = (*, <sBs><sBs>), \qquad \delta'(*, a, <sBs>) = (*, <sBt><tBs>), \\
&\delta'(*, a, <sBt>) = (*, <sBs><sBt>), \qquad \delta'(*, a, <sBt>) = (*, <sBt><tBt>), \\
&\delta'(*, b, <sBt>) = (*, \epsilon), \qquad \delta'(*, b, <tBt>) = (*, \epsilon), \qquad \delta'(*, \epsilon, <tBt>) = (*, \epsilon).
\end{aligned}$$

Tracing for M on the input $aaabb$:

$$\begin{aligned}
&(s, aaabb, \perp) \xrightarrow{1} (s, aabb, B) \xrightarrow{1} (s, abb, BB) \xrightarrow{1} (s, bb, BBB) \xrightarrow{1} \\
&\xrightarrow{1} (t, b, BB) \xrightarrow{1} (t, \epsilon, B) \xrightarrow{1} (t, \epsilon, \epsilon)
\end{aligned}$$

The corresponding tracing for M':

$$\begin{aligned}
&(*, aaabb, <s\perp t> \xrightarrow{1} (*, aabb, <sBt>) \xrightarrow{1} (*, abb, <sBt><tBt>) \xrightarrow{1} \\
&\xrightarrow{1} (*, bb, <sBt><tBt><tBt>) \xrightarrow{1} (*, b, <tBt><tBt>) \xrightarrow{1} (*, \epsilon, <tBt>) \xrightarrow{1} (*, \epsilon, \epsilon)
\end{aligned}$$

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.