Handout 6: The Context-free Languages

6.1 Push-down automata. We have seen that the language $\{0^n1^n \mid n \geq 0\}$ is not regular, because finite automata do not have the ability to count. We now equip finite automata with an unbounded data structure —a stack— which will prove useful for counting. A *push-down automaton* $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ consists of

Q ... a finite set of states,
Σ ... a finite set of input symbols,
Γ ... a finite set of stack symbols,
δ: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ ... a transition relation,
$q_0 \in Q$ ... an initial state,
$F \subseteq Q$ ... a set of final states.

A *run* of M is a sequence

$$(p_0, s_0) \overset{a_0}{\to} (p_1, s_1) \overset{a_1}{\to} \cdots \overset{a_{n-1}}{\to} (p_n, s_n)$$

with $p_0, \ldots, p_n \in Q$ and $s_0, \ldots, s_n \in \Gamma^*$ and $a_0, \ldots, a_{n-1} \in \Sigma$ such that

(1) $p_0 = q_0$ and $s_0 = \varepsilon$ (initially the stack is empty),
(2) for all $i \in [0..n-1]$, we have $(p_{i+1}, \gamma_{i+1}) \in \delta(p_i, a_i, \gamma_i)$ and $s_i = \gamma_i s'$ and $s_{i+1} = \gamma_{i+1} s'$ for some $s' \in \Gamma^*$ (the letter γ, if different from ε, represents the top stack symbol, and s' represents the rest of the stack).

The run *accepts* the input word $a_0 \ldots a_{n-1}$ if

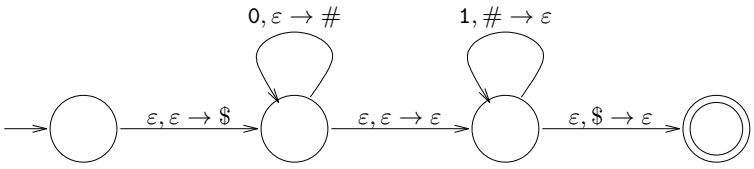
(3) $p_n \in F$.

The *language* of M is

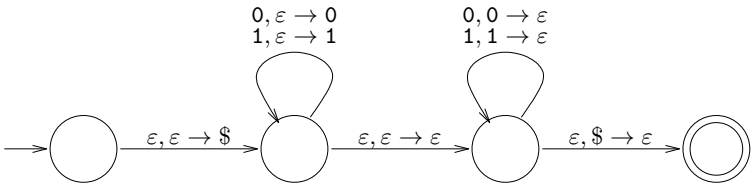
$$L(M) = \{w \in \Sigma^* \mid w \text{ is accepted by some run of } M\}.$$

Note that, according to our definition, PDAs are nondeterministic and may have ε-transitions.

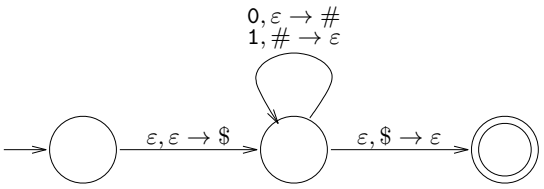
6.2 Examples. The following four languages are not regular but accepted by PDAs:
$L_1 = \{0^n1^n \mid n \geq 0\}$



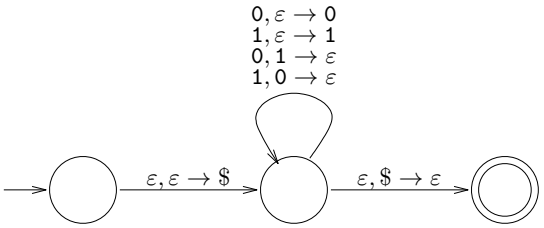
$L_2 = \{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$



$L_3 = \{w \in \{0,1\}^* \mid w \text{ is a balanced string of parentheses}\}$
(where 0 represents an open parenthesis, and 1 represents a closed parenthesis)



$L_4 = \{w \in \{0,1\}^* \mid \#(w,0) = \#(w,1)\}$
(where $\#(w,\sigma)$ denotes the number of occurrences of the letter σ in the word w)



6.3 Context-free grammars. A *context-free grammar* $G = (V, \Sigma, R, S)$ consists of

V ... a finite set of nonterminals (variables),
Σ ... a finite set of terminals (constants),
R ... a finite set of rules (productions) of the form $A \to w$, where $A \in V$ and $w \in (V \cup \Sigma)^*$,
$S \in V$ ... a start symbol.

If $A \to w$, then $xAy \Rightarrow xwy$ for all $x, y \in (V \cup \Sigma)^*$. A *derivation* of G is a sequence $S \Rightarrow w_1 \Rightarrow \cdots \Rightarrow w_n$ with $w_0, \ldots, w_n \in (V \cup \Sigma)^*$. The derivation generates the word w_n. The *language* of G is

$$L(G) = \{w \in \Sigma^* \mid w \text{ is generated by some derivation of } G\}.$$

We shall see that the language of every PDA can be defined by a CFG, and that the language of every CFG can be defined by a PDA. The languages of PDAs and CFGs are called *context-free*.

6.4 Examples. Recall the languages from Example 6.2:
$L_1 = \{0^n 1^n \mid n \geq 0\}$

$S \to \varepsilon \mid 0S1$

$L_2 = \{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$

$S \to \varepsilon \mid 0S0 \mid 1S1$

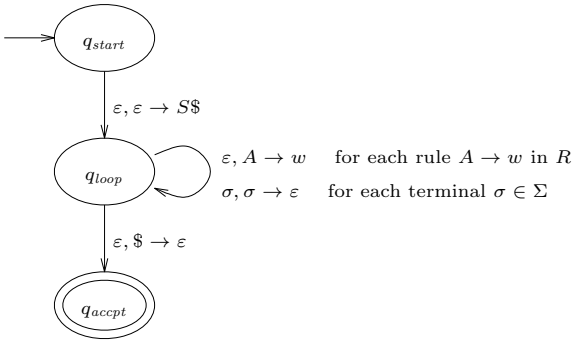
$L_3 = \{w \in \{0,1\}^* \mid w \text{ is a balanced string of parentheses}\}$

$S \to \varepsilon \mid 0S1 \mid SS$

$L_4 = \{w \in \{0,1\}^* \mid \#(w,0) = \#(w,1)\}$

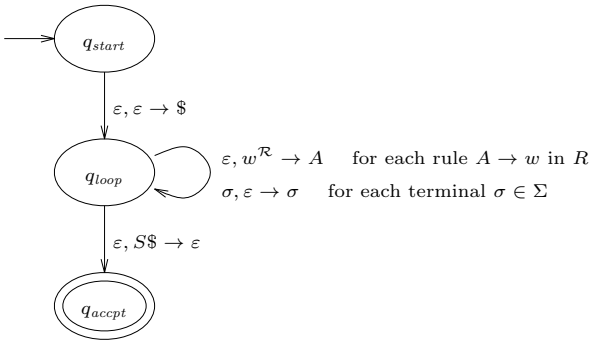
$S \to \varepsilon \mid 0A \mid 1B$
$A \to 1S \mid 0AA$
$B \to 0S \mid 1BB$

6.5 Top-down parsing. Let $G = (V, \Sigma, R, S)$ be a context-free grammar. Let T_G be the PDA with the following state-transition diagram:



If w contains more than one letter, then an edge labeled with $\varepsilon, A \to w$ is really shorthand for several consecutive edges, each pushing one letter of w. Similarly, the edge labeled with $\varepsilon, \varepsilon \to S\$$ is shorthand for two consecutive edges, one labeled with $\varepsilon, \varepsilon \to \$$ and the second one with $\varepsilon, \varepsilon \to S$. It can be seen that each run of T_G corresponds to a derivation of G, and vice versa; that is, $L(T_G) = L(G)$.

6.7 Bottom-up parsing. Let B_G be the PDA with the following state-transition diagram:



Again, each run of T_G corresponds to a derivation of G, and vice versa; that is, $L(B_G) = L(G)$.

6.8 Ambiguity. On the same input word, the runs of the top-down and bottom-up parsers may correspond to different derivations. The top-down parser produces a *left-most* derivation, because at each intermediate stage during the derivation, always the left-most nonterminal is replaced by the right-hand side of a production. By contrast, the bottom-up parser produces a *right-most* derivation. An input word may even have several different left-most (or right-most) derivations; in this case, the grammar is called *ambiguous*. A simple example of an ambiguous grammar is $E \to E + E \mid \mathtt{1}$: the two left-most derivations

$$E \Rightarrow E + E \Rightarrow \mathtt{1} + E \Rightarrow \mathtt{1} + E + E \Rightarrow \mathtt{1} + \mathtt{1} + E \Rightarrow \mathtt{1} + \mathtt{1} + \mathtt{1}$$
$$E \Rightarrow E + E \Rightarrow E + E + E \Rightarrow \mathtt{1} + E + E \Rightarrow \mathtt{1} + \mathtt{1} + E \Rightarrow \mathtt{1} + \mathtt{1} + \mathtt{1}$$

derive the same word (they correspond to different parse trees). Some context-free languages are *inherently ambiguous*, that is, they cannot be generated by unambiguous CFGs. An example of such a language is $\{\mathtt{0}^i\mathtt{1}^j\mathtt{2}^k \mid i = j \text{ or } j = k\}$.

6.9 From push-down automata to context-free grammars. We have seen how given a CFG, we can construct an equivalent PDA. The converse is also true: given a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, we can construct a CFG $G = (V, \Sigma, R, S)$ such that $L(G) = L(M)$. We first modify M so that it satisfies the following three conditions:

—M has a single accept state; i.e., $F = \{q_A\}$.
—M empties its stack before accepting.
—Each transition either pops or pushes a stack symbol, but does not do both.

Let $V = \{A_{p,q} \mid p, q \in Q\}$ and $S = A_{q_0,q_A}$. The intention is to have each nonterminal $A_{p,q}$ derive a string w of terminals iff M, when started in p with the empty stack, on reading input w can end up in q, again with the stack empty. This is achieved by adding the following rules to R[1]:

—For each $p \in Q$, $A_{p,p} \to \varepsilon$.
—For each $p, q, r \in Q$, $A_{p,r} \to A_{p,q}A_{q,r}$.
—For each $p, p', q', q \in Q$ and $a, b \in \Sigma_\varepsilon$ and $\gamma \in \Gamma$, if $(p', \gamma) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(q', b, \gamma)$, then $A_{p,q} \to aA_{p',q'}b$.

Note that the size of G is $O(|M|^3)$.

[1]The construction of G is proved correct in Claims 2.30 and 2.31 (pages 123–124) of M. Sipser's book.