Section 2.2 – Pushdown Automata

CSC 320

Pushdown Automata

- A *pushdown automata (PDA)* is like an NFA except it has a *stack* or *pushdown store* that can be used to record an unbounded amount of information.
- The input is read left to right as in an NFA.
- In each step, the machine is in some state. It can read an input symbol and it can read and pop the top of the stack (as an atomic operation) or it can transition without reading any input or without reading any stack symbol.

Pushown Automata

- Based on the top stack symbol, the input symbol it is currently reading, and its current state, it replaces the top symbol on the stack with a symbol or ε, goes to the next input symbol unless ε is used for the input, and enters a new state (which might be the same state), according to the transition relations of the machine.

- It can also make a move without reading/changing the stack

- The PDA accepts a string if after reading the string the machine is in an accept state.

Example: The Even-Length Palindromes

- Start by putting $\$$ on the stack.
- In state q_0 we're on the first half of the string.
- In state q_1 we've on the second half of the string.
- Pop $\$$ and go to accept state.

Formal Definition

A PDA is a 6-tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where

Q	=	finite set of states;
Σ	=	alphabet of input symbols;
Γ	=	alphabet of stack symbols;
q_0	=	initial state;
F	=	accept states
δ	=	transition function

Formal Definition

- $\delta(q, a, X)$ contains a set of pairs of the form (p_i, Y) where $q \in Q,\ a \in \Sigma \cup \{E\}$ and $X, Y \in \Gamma \cup \{\varepsilon\}$; $p \in Q$ is the new state; Y is the new top of the stack, after popping X.

- Note that ε appears as an option in three places, with the following meainings:

1. Not reading an input symbol
2. Not reading (popping) a stack symbol
3. Not pushing a stack symbol on the stack.

Example: Even Length Palindromes

$P = (\{q_0, q_1, q_2, q_3\}, \{1,0\}, \{1,0,\$\}, \delta, q_0, \{q_3\})$, where δ is described below:

$\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \$)\}$ put bottom of stack symbol on stack

$\delta(q_1, 1, \varepsilon) = \{(q_1, 1)\}$ push symbols on stack /transition

$\delta(q_1, 0, \varepsilon) = \{(q_1, 0)\}$

$\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, \varepsilon)\}$

$\delta(q_2, 0, 0) = \{(q_2, \varepsilon)\}$ pop symbols from stack

$\delta(q_2, 1, 1) = \{(q_2, \varepsilon)\}$

$\delta(q_2, \varepsilon, \$) = \{(q_3, \varepsilon)\}$

Graphical Representation of a PDA

Like an NFA, except an arc from state p to state q is labeled $a,\ X \rightarrow Y$ to indicate that $\delta(p, a, X)$ contains (q, Y).

Acceptance in a PDA

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a pushdown automaton. The string $w \in L(M)$ if $w = w_1 w_2 \cdots w_m$ where each $w_i \in \Sigma_\varepsilon$ and there exist sequences of states $r_0, r_1, \dots, r_m \in Q$ and strings $s_0, s_1, \dots, s_m \in \Gamma^*$ such that

1. $r_0 = q_0$ and $s_0 = \varepsilon$, (start state and empty stack to start)
2. For $i = 0, \dots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = at$ and $s_{i+1} = bt$ for $a, b \in \Gamma$ and $t \in \Gamma^*$, and
3. $r_m \in F$.

Example 2

$$L = \{0^n 1^n \mid n \geq 0\}$$

Example 3

$$L = \{a^i b^j c^k \mid i, j, k \geq 0 \; and \; i = j \text{ or } i = k\}$$

Pushdown Automata and Context-free Grammars

Theorem: The class of languages accepted by PDAs is exactly CFL.

We first show that every CFL is accepted by a PDA

Given a context-free grammar $G = (V, \Sigma, R, S)$, we construct a pushdown automaton $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$, where $\Gamma = V \cup \Sigma \cup \{\$\}$, which accepts the language.

Note: we can push a string $u = u_1 u_2 \ldots u_k$ onto a stack (from right to left) by having states $q_1, \ldots, q_{k-1}$ each of which pushes the next symbol on the stack. i.e.,
$\delta(q, a, s) = \{(q_1, u_k)\}$
$\delta(q_i, \varepsilon, \varepsilon) = \{(q_{i+1}, u_{k-i-1})\}$,$1 \leq i < k - 1$ and
$\delta(q_{k-1}, \varepsilon, \varepsilon) = \{(r, u_1)\}$.

We represent this transition in shorthand: $\delta(q, a, s) = \{(r, u)\}.$

The Translation

$Q = \{q_{start}, q_{loop}, q_{accept}\} \cup \{states\ to\ implement\ shorthand\}$

δ is defined as follows ($A \in V, a \in \Sigma, u \in (V \cup \Sigma)^*$):

1. $\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, S\$)\}$ {Place $\$$ and S on the stack}
2. $\delta(q_{loop}, \varepsilon, A) = \{(q_{loop}, u) \mid A \to u \text{ is a rule of } G\}$ {select a rule with A on LHS and push RHS onto stack}
3. $\delta(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\}$. {match terminal symbol in input to one in rule}
4. $\delta(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)\}$ {accept if stack empty and input read}

Simulating a Leftmost Derivation

Initially, $S\$$ are on the stack;

At each step, if the top is a nonterminal A, with rule $A \to u$ then A is popped and u is pushed.

If the top is a terminal matching the next input symbol, then the top is popped.

The computation mimics a leftmost derivation.

A more formal proof would prove this by induction on the length of the derivation and the length of the string.

Example 4

$$
\begin{aligned}
E &\rightarrow E + E \\
E &\rightarrow E * E \\
E &\rightarrow (E) \\
E &\rightarrow id \\
E &\rightarrow num
\end{aligned}
$$

PDA-recgonizable Languages are Context Free

Proof Idea: Make a CFG which generates exactly all the strings that are accepted by the PDA.

First we can preprocess any PDA so that it:

- has a single accept state q_{accept}
- empties its stack before accepting
- either pushes a symbol onto the stack or pops one off, but not both at the same time.

CFG Construction

For every pair of states p, q, in the PDA, create a variable A_{pq} which generates all strings which take p with empty stack to q with empty stack.

On any input x, the first move is a push: nothing to pop. The last move is a pop since the stack ends up empty.

Either the last symbol popped is the first one pushed, or not

- In the first case, the only time the stack is empty is at the beginning and the end. Add the rule R1:$A_{pq} \rightarrow aA_{rs}b$ where a is the symbol scanned on the first step, b is the symbol scanned on the last step, r is the state following p and s is the state preceding q
- In the second case, add the rule R2: $A_{pq} \rightarrow A_{pr}A_{rq}$ where r is some earlier state on the path from p to q when first symbol pushed is popped.

CFG Construction

Suppose $P = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}\}$. We construct CFG G.

1. The start symbol is $A_{q_0,\, q_{accept}}$. The variables are $\{A_{pq} \mid p, q \in Q\}$.

2. For each $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma \cup \{\varepsilon\}$,
   if $(r, u) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, u)$,
   then include $A_{pq} \rightarrow aA_{rs}b$ in G. (Match on symbol pushed/popped.)

3. For each $p, q, r \in Q$, put $A_{pq} \rightarrow A_{pr}A_{rq}$ in G.

4. For each $p \in Q$, put $A_{pp} \rightarrow \varepsilon$ in G.

We can now prove (by induction) that A_{pq} generates string x iff x can bring P from state p to state q, leaving the stack unchanged.