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 $P = (Q, \Sigma, \Gamma, \delta, q_1, F)$ which accepts the language.

Note: we can push a string $u = u_1 u_2 ... u_k$ onto a stack (from right to left) by having k states $q_1, ..., q_k$ each of which pushes the next symbol on the stack. I.e.,
$\delta(q_1, a, s) = (\{q_1, u_k)\}$
$\delta(q_i, \epsilon, \epsilon) = \{(q_{i+1}, u_{k+1-i}\}$, $2 \leq i < k$ and
$\delta(q_k, \epsilon, \epsilon) = \{(r, u_1)\}$.

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

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

If a language is recognized by a PDA, it's context free

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

- First we can change any PDA to one which:
- has a single accept state
- empties its stack before accepting
- either pushes a symbol onto the stack or pops one off, but not both at the same time.

To construct the CFG

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 or 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.

Preprocessing the PDA

In order for the construction to work, we only work with P satisfying the following properties

1. It has a single accept state q_{accept}
2. It empties its stack before accepting
3. Each transition either pushes a symbol on to the stack, or pops a symbol from the stack, but does not do both

It is easy to preprocess P so that it satisfies these properties (How?)

The 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 $p, q, r, s \in Q, t \in \Gamma$ and $a, b \in \Sigma \cup \{\epsilon\}$,
   if $(r, t) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, t)$,
   then include $A_{pq} \to aA_{rs}b$ in G. (Match on symbol pushed/popped.)

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

4. For each $p \in Q$, put $A_{pp} \to \epsilon$ 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.