Pushdown Automata

Machines to combat the robot uprising.
FSM vs PDA

PDA gets a stack!
Can be pushed and popped.
PDA Formal Definition

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, F), \] where \( Q, \Sigma, \Gamma, \) and \( F \) are finite sets, and

1. \( Q \) is the set of states,
2. \( \Sigma \) is the input alphabet,
3. \( \Gamma \) is the stack alphabet,
4. \( \delta: Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma) \) is the transition function,
5. \( q_0 \in Q \) is the start state, and
6. \( F \subseteq Q \) is the set of accept states.

Computation:
Accepts \( w \) if \( w \) can be written as \( w = w_1w_2 \cdots w_m \) where each \( w_i \in \Sigma^* \), and the sequences of states \( r_0, r_1, \ldots, r_m \in Q \) and strings \( s_0, s_1, \ldots, s_m \in \Gamma^* \) exist that satisfy the following conditions. The strings \( s_i \) represent the sequence of stack contents that \( M \) has on the accepting branch of the computation.

1. \( r_0 = q_0 \) and \( s_0 = \varepsilon \).
2. For \( i = 0, \ldots, m-1 \), \((r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)\), where \( s_i = at \) and \( s_{i+1} = bt \) for some \( a, b \in \Gamma \) and \( t \in \Gamma^* \). (i.e., \( M \) moves properly wrt the state, stack, and next input symbol.)
3. \( r_m \in F \).
Example: PDA definition for $L = \{0^n1^n \mid n \geq 0\}$

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{\emptyset, 1\}$,

$\Gamma = \{\emptyset, \$\}$,

$F = \{q_1, q_4\}$, and

$\delta$ is given by the table (blank is $\emptyset$)

<table>
<thead>
<tr>
<th>Input:</th>
<th>Stack:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack:</td>
<td>0</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_2, 0)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_3, \varepsilon)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \varepsilon)$</td>
</tr>
</tbody>
</table>

State diagram for the PDA.

Rules in the form of $A, B \rightarrow C$ signify that the machine reads $A$ from the input string, pops $B$ from the stack, and pushes $C$ onto the stack. $A, B,$ or $C$ may (either individually or altogether) be $\varepsilon$, indicating that no symbol is processed at that time.
Example: PDA for $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i=j \text{ or } i=k\}$
Exercise: PDA for $L = \{ww^R|w \in \{0,1\}^*\}$

Hints:

- Multiple “rules” may exist on the same edge, but they must be written separately.
- At the “midpoint”, your machine will nondeterministically choose to move from a state that is reading $w$, to a state that is reading $w^R$.
- How can you determine whether or not you have read everything in the stack?
Theorem: Equivalence of PDAs and CFGs

“A language is context free if and only if some pushdown automaton recognizes it.”

“If and only if” requires proof in two directions, hence 2 lemmas.
Lemma: CFL to PDA

“If a language is context free, then some pushdown automaton recognizes it.”

Let $A$ be a CFL. From the CFL definition, $A$ must have a CFG, $G$, generating it. We show how to convert $G$ into an equivalent PDA called $P$.

$P$ will work by accepting input $w$, if $G$ generates that input, by determining whether there is a derivation for $w$.

Remember: A derivation is simply a sequence of substitutions made as a grammar generates a string. Each step of the derivation yields an intermediate string of variables and terminals.

$P$ determines whether some series of substitutions using the rules of $G$ can lead from the start variable to $w$.

Intuition: Use the stack to store intermediate strings (well, sort of).
CFL to PDA: Additional Shorthand Notation

Shorthand notation to push multiple symbols onto the stack:

Let $q, r \in Q$ (the states), $a \in \Sigma$ (the terminals), and $s \in \Gamma$ (the variables).

For the rule $a, s \rightarrow u$, say the PDA goes from $q$ to $r$ when it reads $a$, pops $s$, and pushes a string $u = u_1u_2\ldots u_l$ onto the stack at the same time.

To summarize: $(r, u) \in \delta(q, a, s)$ means that $q$ is the state of the automaton, $a$ is the next input symbols, $s$ is the symbol at the top of the stack. The PDA may read $a$, pop $s$, push the string $u$ onto the stack and go to state $r$.

This multiple-push is accomplished by introducing new states $q_1, \ldots, q_{l-1}$ and setting the transition functions as:

$\delta(q, a, s)$ to contain $(q_1, u)$,
$\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, u_l)\}$,
$\delta(q_2, \varepsilon, \varepsilon) = \{(q_3, u_{l-2})\}$,
\vdots
$\delta(q_{l-1}, \varepsilon, \varepsilon) = \{(r, u_1)\}$.
Lemma: CFL to PDA

Reminder:
- CFG $G = (V, \Sigma, R, S)$
- PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$

$Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$, where $E$ is the set of states needed for implementing the shorthand just described.

$q_0 = q_{start}$

$q_{accept}$ is the only accept state.

$\delta(q_{start}, \epsilon, \epsilon) = \{(q_{loop}, S$)$\}$

3 possible states of the stack:

1. Top symbol is a variable.
   $\delta(q_{loop}, \epsilon, A) = \{(q_{loop}, w) \mid A \rightarrow w \text{ is in } R\}$

2. Top symbol is a terminal.
   $\delta(q_{loop}, a, a) = \{(q_{loop}, \epsilon)\}$

3. Top symbol is the empty stack marker $\$$. 
   $\delta(q_{loop}, \epsilon, \$) = \{(q_{accept}, \epsilon)\}$
Example: CFG to PDA

Construct a PDA from the following CFG:

\[ S \rightarrow aTb \mid b \]
\[ T \rightarrow Ta \mid \varepsilon \]
Lemma: PDA to CFL

“If a pushdown automaton recognizes some language, then it is context free.”

Given a PDA $P$, we will construct a CFG $G$ that generates all the strings that $P$ accepts (i.e., causes $P$ to go from its start state to an accept state).

For each pair of states $p$ and $q$ in $P$, the grammar will have a variable $A_{pq}$. This represents all the strings that can take $P$ from $p$ with an empty stack to $q$ with an empty stack.

Note: It is not required that the stack be empty, but rather that the stack, after reaching state $q$, is in the same condition as it was when it was in state $p$. 
Lemma: PDA to CFL (cont.)

Modify $P$ to give it 3 features:

1. Single accept state, $q_{\text{accept}}$.
2. It empties its stack before accepting.
3. Each transition must either push or pop a symbol, but not both at the same time.

#1 & #2 are easy.

#3 is done by replacing a rule that simultaneously pops and pushes with 2 rules in sequence. Also, if a rule neither pushes or pops, then convert to a 2-rule sequence that pushes and pops an arbitrary stack symbol.

Note:

$P$’s first move must be a push.
$P$’s last move must be a pop.
**P**’s computation on \( x \)

When \( P \) generates a string from \( p \) to \( q \), one of two things can happen to the stack.

1. The symbol popped at the end is the same as the symbol pushed in the beginning.
   
   \[ A_{pq} \rightarrow aA_{rs}b \]
   
   Where \( a \) is the input read at the first move, \( b \) is the input read at the last move, \( r \) is the state following \( p \), and \( s \) is the state preceding \( q \).

2. The symbol popped at the end is not the same as the symbol pushed in the beginning.
   
   \[ A_{pq} \rightarrow A_{pr}A_{rq} \]
   
   Where \( r \) is the state when the stack becomes empty.
PDA to CFL Proof

Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$ and construct $G$. The variables of $G$ are $\{A_{pq} \mid p, q \in Q\}$. The start variable is $A_{q_0, q_{\text{accept}}}$. Now $G$’s rules are in 3 parts:

1. For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma^*$, if $\delta(p, a, \varepsilon)$ contains $(r, u)$ and $\delta(s, b, u)$ contains $(q, \varepsilon)$, put the rule $A_{pq} \rightarrow aA_{rs}b$ in $G$.

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

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

Now, we must prove that this construction works.
PDA to CFL Proof, Claim # 1

If $A_{pq}$ generates $x$, then $x$ can bring $P$ from $p$ with empty stack to $q$ with empty stack.

**Base:** Derivation has 1 step. Single step is a rule whose RHS contains no variables. Only rule in this form is $A_{pp} \rightarrow \varepsilon$. Clearly, $\varepsilon$ takes $P$ from $p$ with empty stack to $p$ with empty stack, so basis is proved.

**Inductive:** Suppose $A_{pq}$ derives $x$ with $k + 1$ steps. 1st step is either $A_{pq} \rightarrow aA_{rs}b$ or $A_{pq} \rightarrow A_{pr}A_{rq}$.

In $A_{pq} \rightarrow aA_{rs}b$, consider $y$ to be the portion generated by $A_{rs}$, so $x = ayb$. Therefore $A_{rs}$ derives $y$ with $k$ steps, and induction hypothesis says $P$ can go from $r$ on empty stack to $s$ on empty stack.

Because $A_{pq} \rightarrow aA_{rs}b$ is a rule of $G$, $\delta(p, a, \varepsilon)$ contains $(r, u)$ and $\delta(s, b, u)$ contains $(q, \varepsilon)$ for some stack symbol $u$.

1. If $P$ starts at $p$ with empty stack it will read $a$, go to state $r$, and push $u$ on the stack.
2. Reading $y$ brings it from $r$ to $s$, while $u$ is still on the stack.
3. Lastly, reading $b$ takes it to state $q$ and pops $u$ from stack.

Therefore, $x$ can bring from $p$ with empty stack to $q$ with empty stack.
For second case, where $A_{pq} \rightarrow A_{pr} A_{rq}$, consider $x = yz$, where $y$ and $z$ are generated by $A_{pr}$ and $A_{rq}$, respectively.

Because $A_{pr}$ derives $y$ in at most $k$ steps and $A_{rq}$ derives $z$ in at most $k$ steps, the induction hypothesis tells us that $y$ can bring $P$ from $p$ to $r$, and $z$ can bring $P$ from $r$ to $q$, with empty stacks at the beginning and end.

Hence, $x$ can bring it from $p$ with empty stack to $q$ with empty stack.
PDA to CFL Proof, Claim #2

If $x$ can bring $P$ from $p$ with empty stack to $q$ with empty stack, $A_{pq}$ generates $x$.

**Base:** The computation has 0 steps. If a computation has 0 steps, it starts and ends at the same state—say $p$. So we must show that $A_{pp}$ derives $x$. In 0 steps, $P$ cannot read any characters, so $x = \varepsilon$. By construction, $G$ has the rule $A_{pp} \rightarrow \varepsilon$, so basis is proved.

**Inductive:** Assume true for $k$ steps, prove for $k + 1$.

Suppose computation on $P$ with $x$ brings $p$ to $q$ with empty stacks with $k + 1$ steps.

Either the stack is empty only at the beginning and end, or it is also empty somewhere in the middle.

In 1st case, symbol that is pushed at the first move must be same as popped at end. Call the symbol $u$. Let $a$ be input read in 1st move, $b$ be the input read in last move, $r$ be the state after the first move, and $s$ be the state before the last move.

$\delta(p, a, \varepsilon)$ contains $(r, u)$ and $\delta(s, b, u)$ contains $(q, \varepsilon)$, so rule $A_{pq} \rightarrow aA_{rs}b$ is in $G$. 
Let $y$ be the portion of $x$ without $a$ and $b$, so $x = ayb$. Input $y$ can bring $P$ from $r$ to $s$ without touching the symbol $u$ on the stack, and so $P$ can go from $r$ with an empty stack to $s$ with an empty stack on input $y$.

We have removed the first and last steps of the $k + 1$ steps, so the computation on $y$ has $k - 1$ steps.

Induction hypothesis tells us that $A_{rs}$ derives $y$, therefore $A_{pq}$ derives $x$.

In 2nd case, let $r$ be a state where the stack becomes empty, other than at the beginning or end of computation on $x$.

Computations from $p$ to $r$ and from $r$ to $q$ each contain at most $k$ steps. Say $y$ is input read from $p$ to $r$ and $z$ is the input read from $r$ to $q$.

Induction hypothesis tells us that $A_{pr}$ derives $y$ and $A_{rq}$ derives $z$.

Because rule $A_{pq} \rightarrow A_{pr}A_{rq}$ is in $G$, $A_{pq}$ derives $x$. 

PDA to CFL Proof, Claim #2 (cont)
Lemma 1:

“If a language is context free, then some pushdown automaton recognizes it.”

Lemma 2:

“If a pushdown automaton recognizes some language, then it is context free.”

Lemma 1: Build PDA from CFG

Lemma 2: Build CFG from PDA

1. All rules are $A_{pq} \rightarrow \epsilon$, $A_{pq} \rightarrow aA_{rs} b$, and $A_{pq} \rightarrow A_{pr} A_{rq}$

2. If $A_{pq}$ generates $x$, then $x$ can bring $P$ from $p$ with empty stack to $q$ with empty stack. (Prove true for all possible rule types.)

3. If $x$ can bring $P$ from $p$ with empty stack to $q$ with empty stack, $A_{pq}$ generates $x$.
   a. Case 1: Rule $A_{pq} \rightarrow aA_{rs} b$
   b. Case 2: Rule $A_{pq} \rightarrow A_{pr} A_{rq}$
Corollary

Every regular language is context free.

Every Regular Language is recognized by a Finite Automaton.

Every Finite Automaton is a Pushdown Automaton that ignores its stack.

Therefore, every Regular Language is also a Context Free Language.