A stack based machine (Push-Down Automaton)

In addition to a finite state transition system, we have an (infinite capacity) stack.

\[(Q, \Sigma, \delta, \emptyset, \Phi, q_0, \Gamma, Z_0)\]

Two separate terminating conditions:
1. The stack is empty when input string is exhausted.
2. We are in a final state when input is exhausted.
\[ L = \{ w \in \{0,1\}^* \mid w \in \text{GR} \} \]

\[ M = (\{q_1, q_2\}, \{0,1,c\}, \{R,B,G\}, \delta, q_1, R, \text{\phi}) \]

1. \( \delta(q_1,0,R) = \{(q_1, BR), \_\} \) \( \delta(q_1,1,R) = (q_1, GR) \)
2. \( \delta(q_1,0,B) = (q_1, BB) \) \( \delta(q_1,1,B) = (q_1, GB) \)
3. \( \delta(q_1,0,G) = (q_1, BG) \) \( \delta(q_1,1,G) = (q_1, GG) \)
4. \( \delta(q_2,0,B) = (q_2, \epsilon) \) \( \delta(q_2,1,G) = (q_2, \epsilon) \)
5. \( \delta(q_2, \epsilon, R) = (q_2, \epsilon) \)
6. \( \delta(q_1, c, R) = (q_2, R) \) \( \delta(q_1, c, G) = (q_2, G) \)
7. \( \delta(q_1, c, B) = (q_2, B) \)
Instantaneous Description (1D) of a PDA

The complete information about a PDA can be obtained from:
1. Current state
2. The current symbol it is scanning
3. The stack contents

1D: \( (q, a \cdot w, \alpha) \) \( q \in Q \)
\( a \in \Sigma, w \in \Sigma^* \)
\( \alpha \in \Gamma^* \)

\( I_0 : (q_0, \varepsilon, z_0) \)

\( I_0 \rightarrow I_1 \rightarrow I_2 \rightarrow \ldots \rightarrow I_f \)

\( I_0 \xrightarrow{\gamma} I_f \)

\( I_j \xrightarrow{\delta} I_{j+1} \)

\( (p, a \cdot w, \alpha\varepsilon) \rightarrow (q, w, \beta\varepsilon) \)
\( \delta (p, a, \lambda) \) must contain \((q, \beta)\)
PDA's that accept by empty stack
$w$ is accepted by the PDA iff
$(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \quad q \in Q$

PDA's that accept by final state
$(q_0, w, Z_0) \vdash^* (q_f, \varepsilon, \alpha) \quad q_f \in F \quad \alpha \in \Gamma^*$

Thus let $L$ be a language accepted by a PDA using empty stack. Then $L$ is also accepted by some PDA that accepts using final state.

and vice versa
**Theorem**

Suppose \( L \) is a CFL generated by a CFG \( G = (V, T, S, P) \). Then we can design a PDA \( M \) (accepting using empty stack) such that \( L(M) = L \).

The proof uses Greibach Normal Form.

**Theorem**

Given a PDA \( M \) that accepts a language \( L \), then we can design a CFG \( G \) such that \( L(G) = L \).

We will provide a construction of a PDA \( M \) given CFG \( G = (S, V, T, P) \) given in Greibach Normal Form:

- \( Q = \{ q \} \) (only one state)
- \( q_0 = q \)
- \( \Gamma = V \)
- \( Z_0 = S \) (bottom stack)
- \( \delta(q, a, A) \) contains \( (q, a) \) if \( A \rightarrow \alpha \epsilon \) and \( \alpha \in V^* \) (including \( \epsilon \))
The idea is to simulate a leftmost derivation of the grammar given in GNF

\[ S \rightarrow x \lambda \text{ iff } (q, x, S) \xrightarrow{*} (q, \lambda, \lambda) \]
when \( x \in T^+ \) and \( \lambda \in V^* \)

The formal proof will be using induction on \( \xrightarrow{*} \)

Let us consider a running example

\[ L = \{ w = w^R \mid w \in (a+b)^* \}, \text{ i.e. palindromes over } \{a,b\} \]

**CFG**

\[
\begin{align*}
S & \rightarrow aSSa \mid bSSb \mid aS \mid bS \mid \lambda \\
S_a & \rightarrow a \\
S_b & \rightarrow b
\end{align*}
\]

in GNF

(It may be easier to compare with a more intuitive non GNF grammar

\[
S \rightarrow aSa \mid bSb \mid aa \mid bb \mid a \mid b
\]
So the PDA would have transition function as follows

1. \( \delta(q, a, s) \) contains \( \{(q, s, s_a), (q, s_a, (q, s_b), (q, s_b, (q, \varepsilon)\}

2. \( \delta(q, b, s) \) contains \( \{(q, s, s_b), (q, s_b, (q, \varepsilon)\}

3. \( \delta(q, a, s_a) \) contains \( (q, \varepsilon) \)

4. \( \delta(q, b, s_b) \) contains \( (q, \varepsilon) \)

Consider the derivation of \( bbaaabbb \) which is a palindrome

\[
S \rightarrow b SS_b \rightarrow b b SS_b S_b \rightarrow b b a S_a S_b S_b
\]

\[
\rightarrow b b a a S_1 S_2 S_2 \rightarrow b b a a b b \rightarrow b b a a b b
\]

Here is how the machine accepts by mimicking the left most derivation

\[S \quad \rightarrow \quad b \quad \rightarrow \quad b b S_1 \quad \rightarrow \quad b b a S_a \quad \rightarrow \quad b b a a b b \]
Verify if the string can be accepted using any alternate moves of the machine. The machine crashes (doesn’t accept) if there is no well-defined next move.