Pushdown Automata

- like NFA-\(\Lambda\) but also has a stack
- transition takes the current state, the current input symbol, and the top-of-the-stack symbol (which is removed from the stack) and returns a state and a string to place in the stack
- the automaton starts in the initial state, reading the first symbol on the tape, and having a special initial symbol on the stack
- automaton accepts a string \(x\) if, after reading all of \(x\), it can get to an accepting state (the stack content does not matter)
- if the stack is empty, the automaton is stuck
Example: \( L = \{ a^k b^k \mid k \geq 0 \} \)
Example:  \( L = \{ w \in \{a,b\}^* \mid w \text{ is an even-length palindrome} \} \)
Def: A **pushdown automaton** (PDA) is a 7-tuple $(Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ where

- $Q$ is a finite set of states
- $\Sigma$ is a finite input alphabet
- $\Gamma$ is a finite stack alphabet
- $q_0 \in Q$ is the initial state
- $Z_0 \in \Gamma$ is the initial stack symbol
- $A \subseteq Q$ is the set of accepting states
- $\delta$ is the transition function from $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$
Def: Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ be a PDA. A configuration is a triple $(p, x, \alpha)$ where $p$ is the current state, $x$ is the rest of the input (the yet-unread part), and $\alpha$ is the content of the stack (the top is on the left).

We write $(p, x, \alpha) \vdash_M (q, y, \beta)$ if we can get from the configuration $(p, x, \alpha)$ to the configuration $(q, y, \beta)$ using one transition of $M$.

Similarly we write $(p, x, \alpha) \vdash_M^* (q, y, \beta)$ if we can get from $(p, x, \alpha)$ to $(q, y, \beta)$ through a finite sequence of transitions of $M$.

The language accepted by $M$:

$$L(M) = \{ x \in \Sigma^* | ___________ \}$$
Deterministic Pushdown Automata

Def: Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ be a PDA. $M$ is deterministic (DPDA) if

- $\forall q \in Q, \forall \sigma \in \Sigma \cup \{\Lambda\}, \forall \gamma \in \Gamma : \left| \delta(q, \sigma, \gamma) \right| \leq 1$

- if we have a $\Lambda$-transition from a state $q \in Q$ w. top-of-stack $\gamma \in \Gamma$, then no $\Sigma$-transition from $q$ w. $\gamma$ is allowed

A language $L$ is a deterministic context-free language (DCFL) if there exists a DPDA accepting $L$. 
Example: Give a DPDA for \( L = \{ a^k b^k \mid k \geq 1 \} \).

Can you give a DPDA for \( L_2 = \{ a^k b^k \mid k \geq 0 \} \)?
Determinism vs. Nondeterminism

For finite automata:

\[ \text{FA, NFA, NFA-} \Delta \text{ accept the same class of languages (regular)} \]

For PDA:

\[ \text{PDA & DPDA accept different classes of lang.} \]

There exists a language which can be accepted by a PDA but not by a DPDA!

- The language of palindromes can be accepted by a PDA \(\leftarrow\) done
- Cannot be accepted by any DPDA

(in book, won't do in class)
Example:
Is the language given by $S \rightarrow SS \mid (S) \mid \Lambda$ a DCFL?
Determinism vs. Nondeterminism

Example: $L = \{ x \in \{a,b\}^* \mid \text{number of } a's \text{ in } x = \text{number of } b's \text{ in } x \}$

$L_2 = \{ x \in \{a,b\}^* \mid \#a's \text{ in } x = 2 \left( \#b's \text{ in } x \right) \}$

Try to draw on your own.
CFG vs. PDA

It is possible to show that every language generated by a CFG can be accepted by a PDA and vice versa.

**Thm:** Let $G = (V, \Sigma_G, S, P)$ be a CFG. Then there exists a PDA $M = (Q, \Sigma_M, \Gamma, q_0, Z_0, A, \delta)$ such that $L(M) = L(G)$.

**Thm:** Let $M = (Q, \Sigma_M, \Gamma, q_0, Z_0, A, \delta)$ be a PDA. Then there exists a CFG $G = (V, \Sigma_G, S, P)$ such that $L(G) = L(M)$.

Which one is easier to prove?
**CFG vs. PDA**

**Thm:** Let $G = (V, \Sigma_G, S, P)$ be a CFG. Then there exists a PDA $M = (Q, \Sigma_M, \Gamma, q_0, Z_0, A, \delta)$ such that $L(M) = L(G)$.

**Example:**

\[
\begin{align*}
S &\rightarrow S + T | T \\
T &\rightarrow T * R | R \\
R &\rightarrow (S) | x
\end{align*}
\]

\[
S \Rightarrow S + T \Rightarrow T + T \Rightarrow R + T \Rightarrow x + T \Rightarrow x + T * R \Rightarrow x + R * R \Rightarrow x + x * R \Rightarrow x + x * x \Rightarrow x + x * x
\]