• **Pushdown automaton** (PDA) $M = (K, \Sigma, \Gamma, \Delta, s, A)$ where

  - $K$ is a set of states
  - $\Sigma$ is an input alphabet
  - $\Gamma$ is a set of stack symbols
  - $s \in K$ is the start state
  - $A \subseteq K$ is a set of accepting states, and
  - $\Delta$ is a transition relation:
    \[ \Delta \subseteq (K \times (\Sigma \cup \{\epsilon\}) \times \Gamma^*) \times (K \times \Gamma^*) \]

• **Configuration** of a PDA $M \in K \times \Sigma^* \times \Gamma^*$

• **Yields-in-one-step** relation:
  \[ (q_1, cw, \gamma_1) \vdash_M (q_2, w, \gamma_2) \text{ iff } ((q_1, c, \gamma_1), (q_2, \gamma_2)) \in \Delta \]

• **Yields** relation:
  - Denoted $\vdash^*_M$

• **Computation** of $M$:
  - Finite sequence of configurations $C_0, C_1, \ldots, C_n, n \geq 0$, where
    - $C_0$ is initial configuration
    - $C_n$ is of the form $(q, \epsilon, \gamma)$ for some $q \in K$ and $\gamma \in \Gamma^*$
    - $C_0 \vdash_M C_1 \vdash_M \ldots \vdash_M C_n$
  - **Accepting computation**: One in which $(s, w, \epsilon) \vdash^* (q, \epsilon, \epsilon)$ for some $q \in A$
  - **Rejecting computation**: One in which $(s, w, \epsilon) \vdash^* (q, w', \alpha)$ where $M$ has no transitions from $q$ given $w', \alpha$

• Label arcs of transition diagram with $c / \gamma_1 / \gamma_2$ to denote $((q_1, c, \gamma_1), (q_2, \gamma_2))$

---

**Pushdown Automata: Deterministic PDAs**

• Two conditions must hold:
  1. $\Delta_M$ contains no pairs of transitions that compete
  2. If $q \in A_M$, there is no transition $((q, \epsilon, \epsilon), (p, a))$ for any $p, a$

---

**Theorem 12.1**

- Statement: Given a CFG $G = (V, \Sigma, R, S)$, there exists a PDA $M$ such that $L(M) = L(G)$
  1. Top-down parser
    * PDA $M$ has exactly 2 states:
      \[
      \begin{array}{c}
      \text{PDA} \\
      \text{CFGToPDA-TD (CFG G)}
      \end{array}
      \]
      
      * Algorithm: PDA CFGToPDA-TD (CFG G);
      - Create $M = (\{(p, q)\}, \Sigma, V, \Delta, p, \{q\})$, where $\Delta$ contains
        - $((p, \epsilon, \epsilon), (q, S))$
        - $((q, \epsilon, X), (q, \gamma_1 \gamma_2 \ldots \gamma_n))$ for each $X \rightarrow \gamma_1 \gamma_2 \ldots \gamma_n \in R_G$
        - $((q, \epsilon, X), (q, \epsilon))$ for each $X \rightarrow \epsilon \in R_G$
        - $((q, c, c), (q, \epsilon))$ for each $c \in \Sigma$
      \]
      
      \[
      \begin{array}{c}
      \text{all other transitions}
      \end{array}
      \]
2. Bottom-up parser

* PDA $M$ has exactly 2 states:

$$
\begin{array}{c}
\varepsilon/S/E \\
\end{array}
$$

* Algorithm: PDA $\text{CFGToPDA-BU (CFG G)}$

Create $M = ((p, q), \Sigma, V, \Delta, p, \{q\})$, where $\Delta$ contains

- $(p, \epsilon, S, (q, \epsilon))$
- $(p, \epsilon, (\gamma_1 \gamma_2 \ldots \gamma_n)^R, (p, X))$ for each $X \rightarrow \gamma_1 \gamma_2 \ldots \gamma_n \in R_G$
- $(q, c, \epsilon, (p, c))$ for each $c \in \Sigma$

**Pushdown Automata: Equivalence of CFGs and PDAs - Converting PDAs to CFGs**

- **Restricted Normal Form (RNF)**

  - Algorithm:

    ```
    \text{PDA convertPDAToRNF (PDA M)}
    \{
    M' = M;
    M'.K = M'.K + s';
    M'.Gamma = M'.Gamma + #;
    M'.Delta = M'.Delta + ((s', \epsilon, \epsilon), (s, \#));
    M'.s = s';
    M'.K = M'.K + a;
    for (each q in M'.A)
    M'.Delta = M'.Delta + ((q, \epsilon, \#), (a, \epsilon));
    M'.A = a;
    for (each t in M'.Delta)
    if (t == ((q, c, \epsilon, \epsilon), (p, \gamma)))
    M'.K = M'.K + qq_1;
    M'.Delta = M'.Delta + ((q, c, a_1), (qq_1, \epsilon));
    for (i = 1; i < n - 1; i++)
    M'.K = M'.K + qq_{i+1};
    M'.Delta = M'.Delta + ((qq_i, \epsilon, a_{i+1}), (qq_{i+1}, \epsilon));
    M'.Delta = M'.Delta + ((qq_{n-1}, c, a_n), (p, \gamma));
    M'.Delta = M'.Delta - t;
    \}
    for (each t in M'.Delta)
    if (t == ((q, c, \epsilon, \epsilon), (p, \gamma)))
    for (each a in M'.Gamma)
    M'.Delta = M'.Delta + ((q, c, a), (p, \gamma a));
    M'.Delta = M'.Delta - t;
    return M';
    \}
    ```
Theorem 12.2

Statement: Given PDA \( M = (K, \Sigma, \Gamma, \Delta, s, A) \), there exists a CFG \( G \) such that \( L(G) = L(M) \)

Proof: By construction

* Begin by converting PDA to RNF
* Nonterminals will have form \( <q_\gamma q_j> \)
  
  This nonterminal generates only those strings that drive \( M \) from state \( q_i \) to state \( q_j \) while popping all symbols on the stack from the top through the first occurrence of \( \gamma \)
  
  For PDA in RNF, \( <s\#a> \) generates all strings that drive \( M \) from the start state to \( a \), popping everything off the stack
  
  First rule in grammar is thus \( S \rightarrow <s\#a> \)

* To generate rest of rules, consider three kinds of transitions

1. Those that push no symbols
   
   Consider

   \[
   \begin{array}{c}
   q_i \\
   \rightarrow \\
   q_j \\
   \rightarrow \\
   w
   \end{array}
   \]

   and transition \( ((q, c, \gamma), (r, \epsilon)) \in M.\Delta \) where \( c \in \Sigma \cup \{\epsilon\} \)
   
   After \( M \) reaches \( r \), it could reach \( w \) via any states in \( M \)
   
   Nonterminal \( <q_\gamma w> \) generates strings that drive \( M \) from \( q \) to \( w \) while popping \( \gamma \)
   
   Some of these strings are those that start with \( c \) and that drive \( M \) from \( r \) to \( w \) without popping anything
   
   So need rule \( <q_\gamma w> \rightarrow c <rew> \)

2. Those that push one symbol
   
   Consider

   \[
   \begin{array}{c}
   q_i \\
   \rightarrow \\
   r \\
   \rightarrow \\
   w
   \end{array}
   \]

   \( M \) reads \( c \), pops \( \gamma \), pushes alpha, and transitions to \( r \)
   
   Nonterminal \( <q_\gamma w> \) generates strings that drive \( M \) from \( q \) to \( w \) while popping \( \gamma \)
   
   Some of these strings are those that drive \( M \) from \( q \) to \( r \) while popping \( \gamma \) and reading \( c \), and then driving \( M \) from \( r \) to \( w \) while popping \( \alpha \)
   
   So need rule \( <q_\gamma w> \rightarrow c <raw> \)

3. Those that push more than one symbol
   
   Consider

   \[
   \begin{array}{c}
   q_i \\
   \rightarrow \\
   r_1 \\
   \rightarrow \\
   \rightarrow \\
   r_{n-1} \\
   \rightarrow \\
   w
   \end{array}
   \]

   Since \( M \) is in RNF, can only pop one symbol per transition
   
   For every symbol pushed during one transition, will need an additional transition to pop each
   
   Some of the strings that drive \( M \) from \( q \) to \( w \) are those that drive \( M \) from \( q \) to \( r \) while popping \( \gamma \)
   
   They then drive \( M \) from \( r \) to \( r_1 \) while popping \( \alpha_1 \), from \( r_1 \) to \( r_2 \) while popping \( \alpha_2 \), etc.
   
   They finally transition from \( r_n \) to \( w \) while popping \( \alpha_n \)
   
   So need rule \( <q_\gamma w> \rightarrow c <ra_1 r_1> <r_1 a_2 r_2> ... <r_{n-1} a_n w> \)

4. Need to be able to terminate rule expansion
   
   So need rule \( <q\epsilon q> \rightarrow \epsilon \)
Algorithm: 

CFG convertPDAToCFG (PDA M) 
{
    \( M' = \text{convertPDAToRNF}(M); \)
    \( V = M'.S; \)
    \( R = \{S \rightarrow <s \# a>\}; \)
    for (each ((q, c, gamma), (r, epsilon)) in \( M'.\Delta \))
        for (each state w in \( M'.K \))
            if (w \neq M'.s)
                \( R = R + <q \gamma w> \rightarrow c<r \epsilon w>; \)
    for (each ((q, c, gamma), (r, alpha)) in \( M'.\Delta \))
        if (q \neq M'.s)
            for (each state w in \( M'.K \))
                if (w \neq M'.s)
                    \( R = R + <q \gamma w> \rightarrow c<r \alpha w>; \)
    for (each ((q, c, gamma), (r, alpha1 alpha2 ... alphan)) in \( M'.\Delta \))
        if (q \neq M'.s) 
            create rules \( \pi \) of form
            \( <q \gamma w> \rightarrow c<q \alpha1 r1><r1 \alpha2 r2>...<rn-1 alphan w>; \)
            \( R = R + \{\pi\}; \)
    for (each state q in \( M'.K \))
        if (w \neq M'.s)
            \( R = R + <q c q> \rightarrow \epsilon; \)
    \( G = (M'.V, M'.\Sigma, M'.S, R); \)
    return \( G; \)
}