(pp. 111-125) **Push Down Automata** (Sec. 2.2)

- **Push Down Automata (PDA)** = DFA + Stack
  - Capable of recognizing CFLs
- Difference from NFA: at each transition
  - Can read (& push) current top value on stack in δ arguments
  - Each δ rule specifies not just new state but optional value to push onto a stack
- Stack depth may become infinite – allows recognizing languages with arbitrary components
  - Notional execution for \( \{0^n1^n\} \) – non-regular language
    - At start, for each 0 input, push a 0 to stack
    - At first 1, for each 1 input, pop a 0 off stack
    - If stack & input run out at same time, accept
    - Else reject
- See Fig. 2.12 on p. 110
• **Formal Definition**: PDA $M = 6$ tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$
  • Same kind of nondeterminism as in NFA
  • $Q, \Sigma, q_0, F$ as before
  • $\Gamma$ ("gamma") is **stack alphabet**: symbols that may be on stack
    • Need not have any relation to $\Sigma$
  • $\delta: Q \times \Sigma^* \times \Gamma^* \rightarrow P(Q \times \Gamma^*)$
    • $\Sigma^* = \Sigma \cup \epsilon$
    • $\Gamma^* = \Gamma \cup \epsilon$
  • A rule $\delta(q, x, s)$ is applicable only if
    • Machine is in state $q$
    • $x$ from $\Sigma$ matches next character on input
      • If $x = \epsilon$, then we don’t need a character on input
      • Like $\epsilon$ rules in NFA
    • $s$ from $\Gamma$ matches the current top of the stack
      • If $s = \epsilon$, then we don’t look at stack top
  • If a rule has a non-$\epsilon$ $s$ and is chosen:
    • $s$ is “popped” off stack before rhs is performed
  • Range of a $\delta$ rule is a (state, $z$) where $z$ in $\Gamma^*$
    • If $z$ in $\Gamma$, push $z$ onto stack
    • If $z=\epsilon$, leave stack unchanged.
• Computation of PDA M
  • Assume
    • Input string \( w \) can be written as \( w = w_1, \ldots w_m \), each
c character \( w_i \) either in \( \Sigma \) or an \( \varepsilon \)
    • I.e. whatever input is, we can assume \( \varepsilon \)s can be
       assumed present between any 2 characters
  • Sequence of states \( r_0, r_1, \ldots r_m \) (i.e. \( |w|+1 \) states)
  • Sequence of stack \textit{strings} \( s_0, s_1, \ldots s_m \)
    • Each string is the stack at some time
    • Where leftmost symbol of each string is the “top”
  • A valid \textbf{computation} is when
    • \( r_0 = q_0 \) and \( r_m \) is in \( F \)
    • \( s_0 = \varepsilon \) (stack is initially empty)
    • For \( i = 0 \) to \( m-1 \)
      • \((r_{i+1}, b)\) is in \( \delta(r_i, w_i, a) \) where
        • \( s_i = at, a \in \Gamma_\varepsilon, t \in \Gamma^* \) (i.e. \( a \) is top, \( t \) rest of stack)
          • If \( a \neq \varepsilon \), we \textbf{pop} it off of stack before update
        • \( s_{i+1} = bt, a \in \Gamma_\varepsilon, t \) stack after above step
          • If \( b \neq \varepsilon \), we \textbf{push} it onto stack
• State diagrams similar to NFA but labels augmented
  • Instead of “a”, write “a,b->c” where
    • a in $\Sigma_\epsilon$ is character on input that causes transition
      • a = $\epsilon$ says ignore input
    • b in $\Gamma_\epsilon$ must likewise match stack top
      • b = $\epsilon$ says ignore stack top
      • b $\neq$ $\epsilon$ says we must match, AND pop after transition
    • **Shorthand** “a->c” for “a, $\epsilon$ -> c”
    • c in $\Gamma_\epsilon$ give stack top after transition
      • c = $\epsilon$ implies push nothing
      • c $\neq$ $\epsilon$ implies push c
    • **Shorthand** “a,b” for “a,b->$\epsilon$”
  • Summary of stack changes for a,b->c. Assume $s_i = xt$

<table>
<thead>
<tr>
<th>b (match for stack)</th>
<th>c (new stack top)</th>
<th>New stack $s_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>b = $\epsilon$</td>
<td>c = $\epsilon$</td>
<td><strong>NOP</strong>: $s_{i+1} = s_i = xt$</td>
</tr>
<tr>
<td>b = $\epsilon$</td>
<td>c $\neq$ $\epsilon$</td>
<td><strong>Push</strong>: $s_{i+1} = cxt$</td>
</tr>
<tr>
<td>b $\neq$ $\epsilon$ i.e. x=b, $s_i=bt$</td>
<td>c = $\epsilon$</td>
<td><strong>Pop</strong>: $s_{i+1} = t$</td>
</tr>
<tr>
<td>b $\neq$ $\epsilon$ i.e. x=b, $s_i=bt$</td>
<td>c $\neq$ $\epsilon$</td>
<td><strong>Change</strong>: $s_{i+1} = ct$</td>
</tr>
</tbody>
</table>

• See pages 112-116 for examples