(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 (& *pop*) 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, Σ, q_0, F as before
  - **Γ** ("gamma") is **stack alphabet**: symbols that may be on stack
    - Need not have any relation to ∑
  - $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$
    - $\Sigma_\varepsilon = \Sigma \cup \varepsilon$
    - $\Gamma_\varepsilon = \Gamma \cup \varepsilon$
  - A rule δ(q, x, s) is applicable only if
    - Machine is in state q
    - x from Σ matches next character on input
      - If x = ε, then we don't need a character on input
      - Like ε rules in NFA
    - s from Γ matches the current top of the stack
      - If s = ε, then we don't look at stack top
  - If a rule has a non-ε s and is chosen:
    - s is "popped" off stack before rhs is performed
  - Range of a δ rule is a (state, z) where z in Γ_ε
    - If z in Γ, push z onto stack
    - If z=ε, leave stack unchanged.

- Computation of PDA M
  - Assume
    - Input string w can be written as $w = w_1, \ldots w_m$, each character w_i either in Σ or an ε
      - I.e. whatever input is, we can assume εs 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 *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 **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 Γ_ε, t in Γ^* (i.e. a is top, t rest of stack)
          - If a != ε, we **pop** it off of stack before update
        - $s_{i+1} = bt$, a in Γ_ε, t stack after above step
          - If b != ε, we **push** it onto stack

- State diagrams similar to NFA but labels augmented
  - Instead of "a", write "a,b->c" where
    - a in Σ_ε is character on input that causes transition
      - a = ε says ignore input
    - b in Γ_ε must likewise match stack top
      - b = ε says ignore stack top
      - b != ε says we must match, AND pop after transition
      - **Shorthand** "a->c" for "a,ε -> c"
    - c in Γ_ε give stack top after transition
      - c = ε implies push nothing
      - c != ε implies push c
      - **Shorthand** "a,b" for "a,b->ε"
  - Summary of stack changes for a,b->c. Assume s_i = xt

b (match for stack)	**c (new stack top)**	**New stack s_{i+1}**
b = ε	c = ε	**NOP**: $s_{i+1} = s_i$ = xt
b = ε	c != ε	**Push**: s_{i+1} = cxt
b != ε i.e. x=b, s_i=bt	c = ε	**Pop**: s_{i+1} = t
b != ε i.e. x=b, s_i=bt	c != ε	**Change**: s_{i+1} = ct

- See pages 112-116 for examples