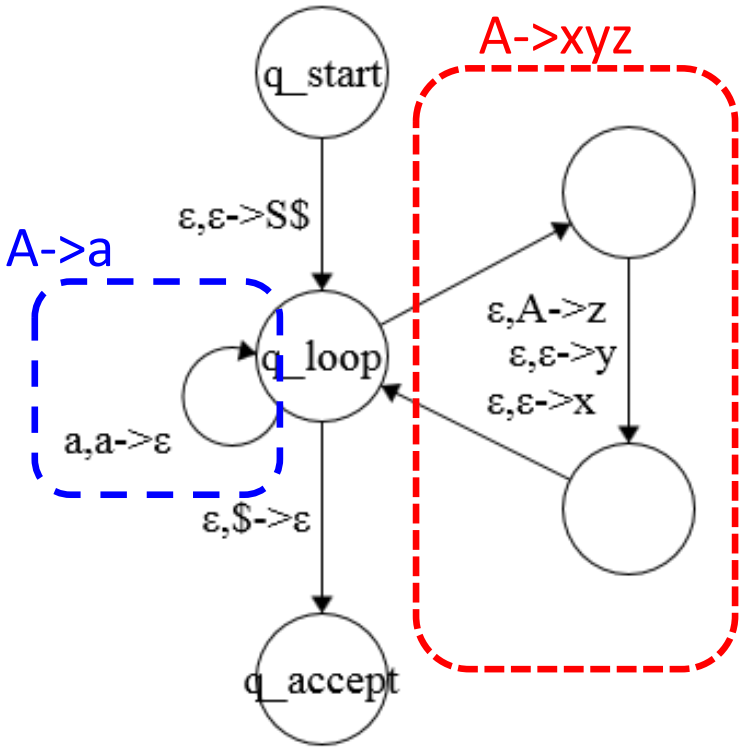
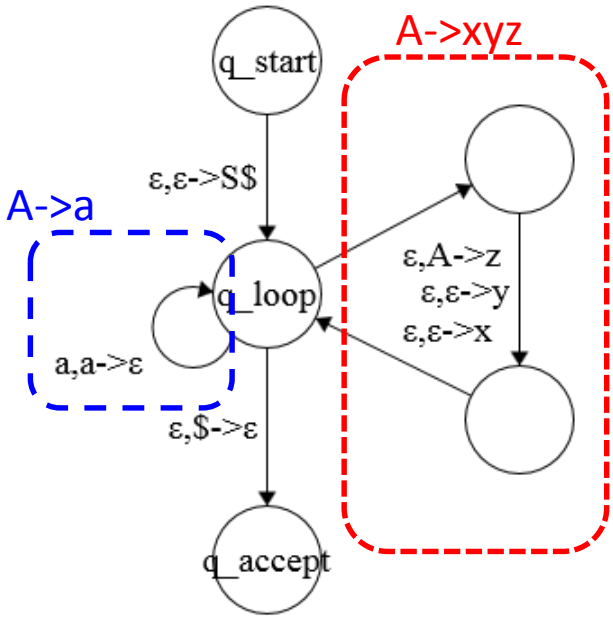
(pp. 117-124) **PDAs and CFGs** (Sec. 2.2)

- A language is context free iff all strings in L can be generated by some context free grammar
- Theorem 2.20: **L is Context Free iff a PDA accepts it**
  - I.e. if L is context free than some PDA accepts it
  - AND if a PDA accepts L, then it is context free
- Outline of proof: must prove in both directions
  - If language A is CF, then we show construction of a PDA P
    - Use stack to keep the right hand of the intermediate string that includes the leftmost variable on top
    - Create transition rules from grammar rules
    - Use nondeterministic choice of rules to match terminals
  - If a PDA P recognizes L, then L is CF
    - Proof by constructing a CFG from P that matches

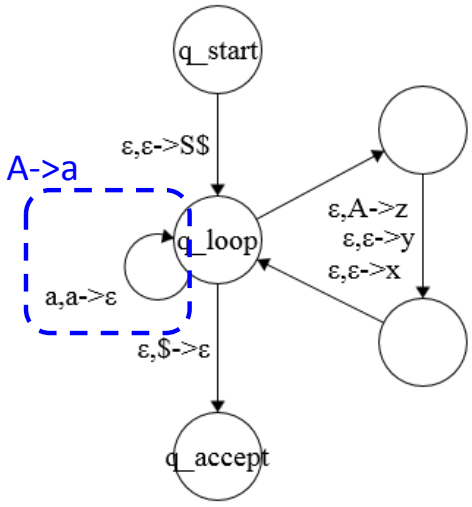
- (p117) 1st part: **if G=(V,∑,R,S) is CFG for L, then some PDA P=(Q,∑,Γ,δ,q_{start},F) accepts it**
  - **Proof: Build the PDA from the Grammar**
  - Overview of proof by construction
  - Assume G=(V,∑,R,S)
    - Γ_ε = V U ∑ U {$, ε}
      - = Alphabet + non-terminals + special characters $, ε
    - 3 common states: q_{start}, q_{loop}, q_{accept}
      - F = {q_{accept}}
      - Additional states added for each grammar rule
        - Extra states for rules like A->xyz
        - Self-loop on q_{loop} for rules like A->a



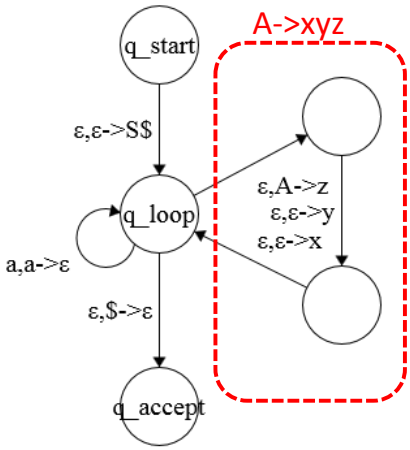


- **Overview of PDA P in operation (see Fig. 2.26)**:
  - Start with pushing "**S**$" onto stack (with **S** on top)
    - Used $ to mark bottom of stack
  - Repeatedly loop around state q_{loop}
    - **If stack top is $**, enter accept state
    - **If stack top is non-terminal A**, select an edge (non-deterministically) based on one of rules for A
      - Pop the variable
      - Push the RHS (in reverse order – leftmost on top)
    - **If stack top is terminal "a",** next symbol on input must be "a" to be accepted. Pop a.

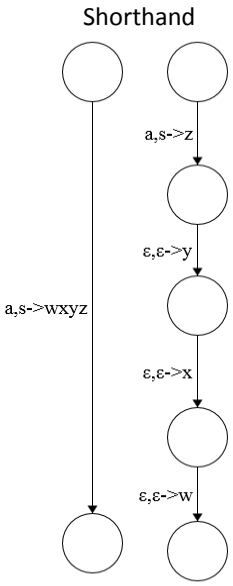
- Formal Construction of P from Grammar
  - Remember PDA transition rule specifies pair (q, x)
    - q is next state
    - x is character to push on stack
  - Q = $\{q_{start}, q_{loop}, q_{accept}\}$ U E
    - q_{loop} is special state where all grammar rules start & end
    - E = all states generated by grammar rules as discussed below
  - F = $\{ q_{accept}\}$
  - Add startup transitions to push S$ on start
    - $\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_1, \$)\}$, q_1 a new state in E
    - $\delta(q_1, \varepsilon, \varepsilon) = \{(q_{loop}, S)\}$
    - Note **shorthand** "single edge" ε, ε->S$
  - For each terminal a in ∑, add the following self-loop
    - $\delta(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\}$ (We match the a and pop from stack)
  - To detect acceptance, add rule
    - $\delta(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)\}$



- (p.118) For kth rule S->$u_1u_2…u_L$, u_i from ∑ U V
  - $\delta(q_{loop}, \varepsilon, S)$ includes $(q_{k,1}, u_L)$, $q_{k,1}$ a new state
  - Add L-1 transitions to push $u_1u_2…u_{L-1}$ onto stack, with u_1 on top as follows
    - $\delta(q_{k,1}, \varepsilon, \varepsilon) = \{(q_{k,2}, u_{L-1})\}$, $q_{k,2}$ a new state in E
    - $\delta(q_{k,2}, \varepsilon, \varepsilon) = \{(q_{k,3}, u_{L-2})\}$, $q_{k,3}$ a new state in E
    - …
    - $\delta(q_{k,L-2}, \varepsilon, \varepsilon) = \{(q_{k,L-1}, u_2)\}$, q_l a new state in E
    - $\delta(q_{k,L-1}, \varepsilon, \varepsilon) = \{(q_{loop}, u_1)\}$



- (p. 119, Fig. 2.23) Book uses shorthand a,s->w (w a string) on edge for sequence of steps:
  - a,s->w_n
  - ε,ε->w_{n-1}
  - …
  - ε,ε->w_1



- (p. 120) Final machine looks like Fig.2.24
- (p.120) Example problem Fig.2.25
- (p. 155) See also problems 2.5, 2.7, 2.9 esp. 2.11, and create PDAs from CFGs 2.13, 2.14. 2.46

(p. 121) **Now prove if some PDA accepts L, L must be CF**

- Outline:
  - Generate variable A_{pq} in G for pair of states p & q in P
  - Generate rules for A_{pq} that correspond to strings that cause transitions between the 2 states p & q
    - When stack starts as empty at p
    - And ends as empty at q
    - I.e. nothing needed or left on stack
  - Make start variable that has "or" of all variables A_{pq} where p is start state and q is a final state
    - This will collect all strings from start to finish

- Again by construction of a CFG
  - Modify P slightly
    - Ensure a single accept state q_{accept}
      - From any prior accept state, add set of transitions that ensure stack is empty before final accept state
    - Ensure each transition *either* pushes *or* pops *but not both* or neither
      - If a transition does <u>both</u> (δ(q,a,x)->{(q',y)},…),
        - add new intermediate state
        - Transition from original state does pop: a,x->ε
        - Transition from new state does push: ε,ε->y
      - If a transition does <u>neither</u> (δ(q,ε,ε)->{(q',ε)},
        - add new state and select any terminal x
        - Transition from original state pushes x: a,ε->x
        - Transition from new state pops that x: ε,x->ε

- Now construct G = (V,∑,R,S)
  - ∑ the same
  - V = {A_{pq} | p, q in Q} – 1 symbol for each pair of states
  - S = $A_{q0,qaccept}$
  - Construct grammar rules R as follows
    - For each p,q,r,s in Q, u in Γ, and a,b in ∑
      - If δ(p, a, ε) contains (r,u) (we are pushing u)
      - And δ(s, b, u) contains (q, ε) (we are popping u)
      - Then add grammar rule A_{pq} -> aA_{rs}b
      - Notes on what this does:
        - From state p we push a u and go to state r
        - From state r we can get to state s via transitions that leave nothing new on stack above u
        - Transition from s to u P pops the original u
        - See P. 122 Fig. 2.29 for stack changes
    - For each p,q,r in Q
      - Then add grammar rule A_{pq} -> $A_{pr}A_{rs}$
      - Again nothing added from p to q
      - See p. 122 Fig. 2.28
    - For each p in Q
      - Then add grammar rule A_{pp} -> ε

- (p. 123) Claim 2.30. If variable A_{pq} generates string x, then x can bring P from state p with an empty stack to state q with empty stack
  - Proof by induction on # of steps in derivation of x
  - Basis step: it took 1 step
    - Only grammar rules with no RHS variables are A_{pp}->ε
    - i.e. ε must take P from p to p without pushing anything onto empty stack, or attempting a pop
  - Induction Hypothesis: assume true for derivations of length at most k, k≥1.
  - Induction step: prove true for derivations of length k+1
    - Suppose A_{pq}=*>x (x a string) with k+1 steps
    - Two possibilities: A_{pq} -> aA_{rs}b or A_{pq} =>$A_{pr}A_{rq}$
      - First case: A_{pq} -> aA_{rs}b for some a,b, r,s
        - A_{rs} must have generated y where x = ayb
          - But this must have happened in k steps, so P can go from r to s on empty stack
          - Because A_{pq} -> aA_{rs}b is a rule in G
            - δ(p, a, ε) contains (r, u) for some u
              - i.e. it pushes u
            - and δ(s, b, u) contains (q, ε)
              - i.e. it pops u

        - Thus if P starts at p with empty stack
            - After reading a it goes to state r with u on stack
            - Then reading y brings P to s and leaves u on stack
    - Second case: $A_{pq} \Rightarrow A_{pr}A_{rq}$
        - Assume x=yz where
            - $A_{pr} \Rightarrow y$ in at most k steps
            - $A_{rs} \Rightarrow z$ in at most k steps
        - Then induction hypothesis says y can bring P from p to r, and z can bring P from r to q, with empty stacks on both ends
- (p.123) Claim 2.31: If we can bring P from p to q with empty stacks on both sides then A_{pq} generates x
- (p. 124) Corollary 2.32. Every regular language is CF.
    - Every regular language recognized by some DFA
    - Every DFA is a PDA
    - All languages accepted by a PDA are CFL
- Sample: convert Fig. 2.15 (p. 115) back to CFG
- Also attempt: (p. 116) Fig. 2.17 and 2.18, (p.120) 2.26
- Also try Probs. 2.11, 2.12,
    - and convert grammars of problems 2.13, 2.14, 2.19, 2.46