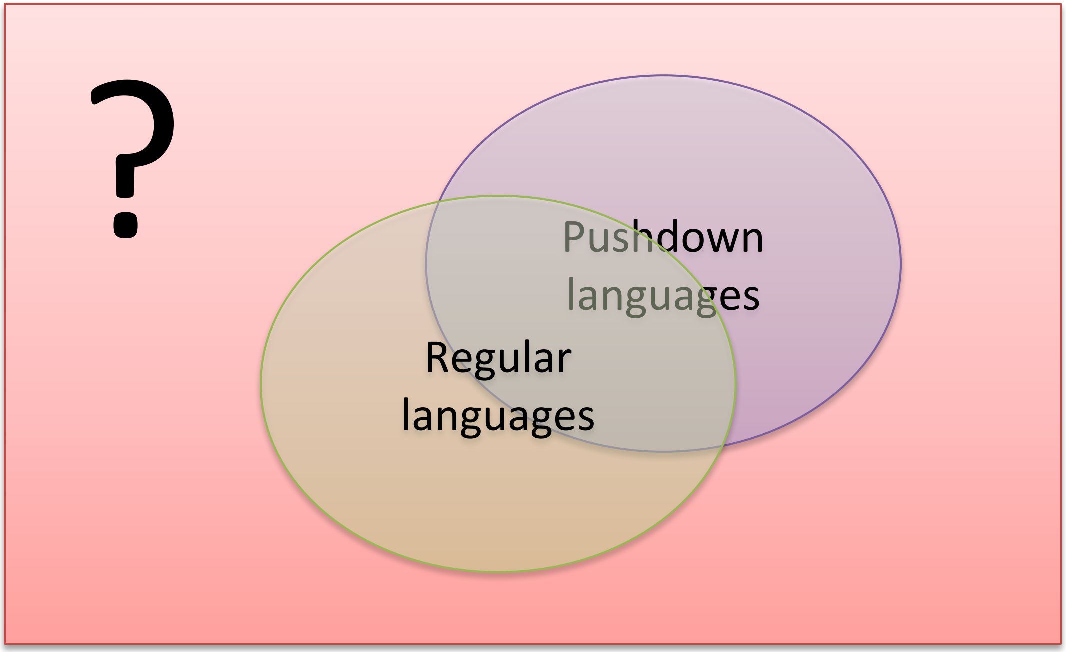
PDAs <=> CFGs

Sipser 2.2 (pages 119-122)

Finite automata and Pushdown automata



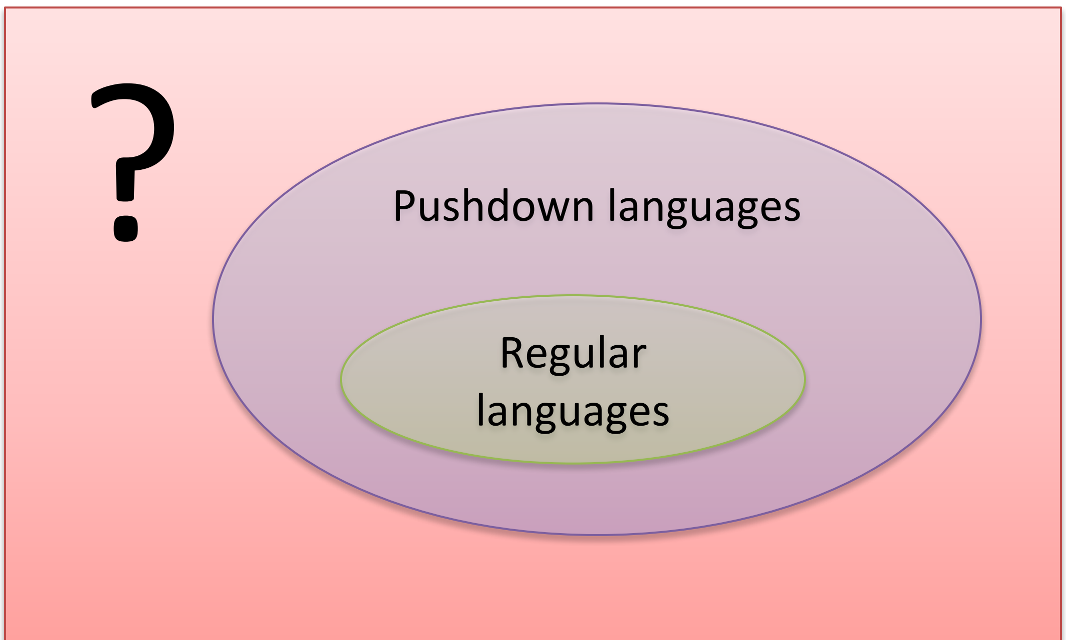
Regular => Pushdown

- Proposition: Every finite automaton can be viewed as a pushdown automaton that never operates on its stack.
- Proof:

  Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

  Define $M' = (Q, \Sigma, \Gamma, \delta', q_0, F)$, where...

Finite automata and Pushdown automata



CFGs and PDAs

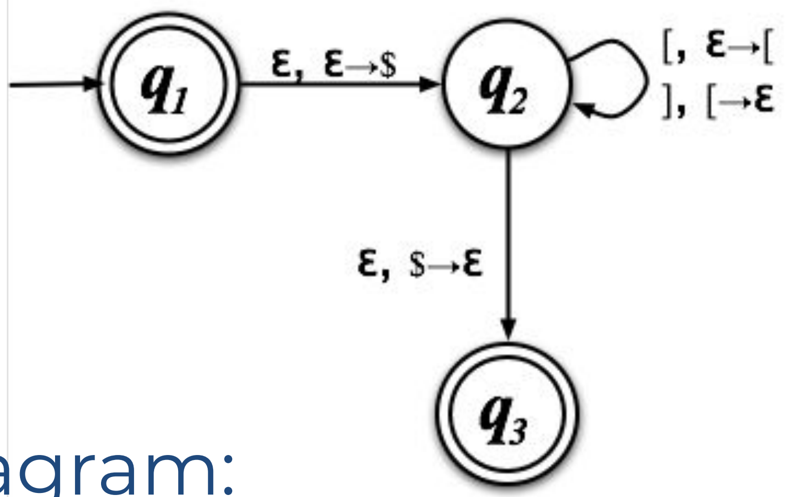
- Theorem 2.20: A language is context-free if and only if some pushdown automaton recognizes it.

Formally…

- A pushdown automaton is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where
  - Q is a finite set of states
  - Σ is a finite alphabet (the input symbols)
  - Γ is a finite alphabet (the stack symbols)
  - $\delta: (Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon) \rightarrow P(Q \times \Gamma_\varepsilon)$ is the transition function
  - $q_0 \in Q$ is the initial state, and
  - $F \subseteq Q$ is the set of accept states

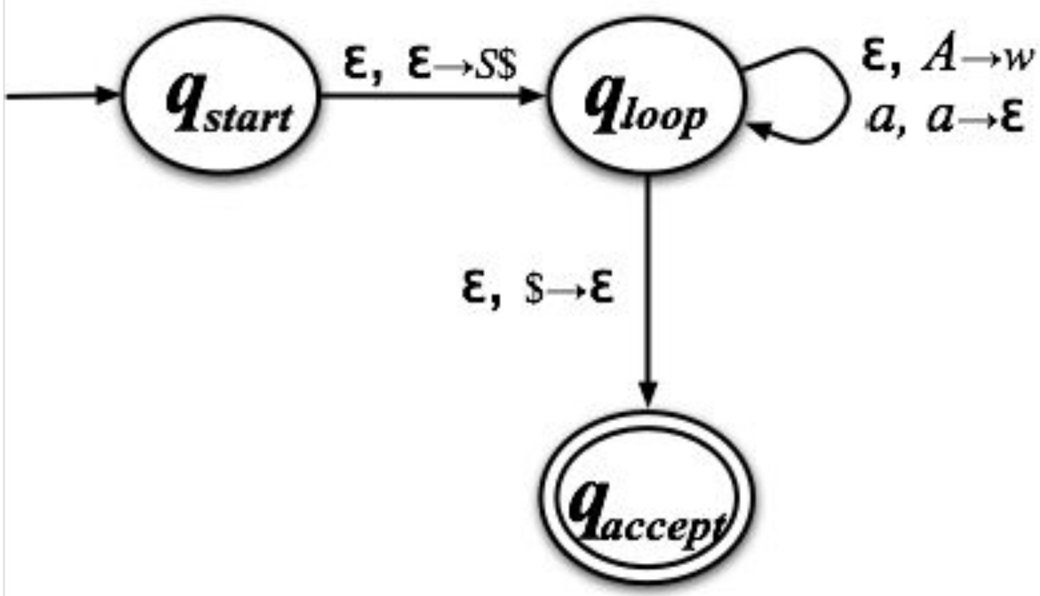
Balanced brackets

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where
  - $Q = \{q_1, q_2, q_3\}$
  - $\Sigma = \{[,]\}$
  - $\Gamma = \{[, \$\}$
  - $q_0 = q_1$
  - $F = \{q_1, q_3\}$
  - δ is given by state diagram:

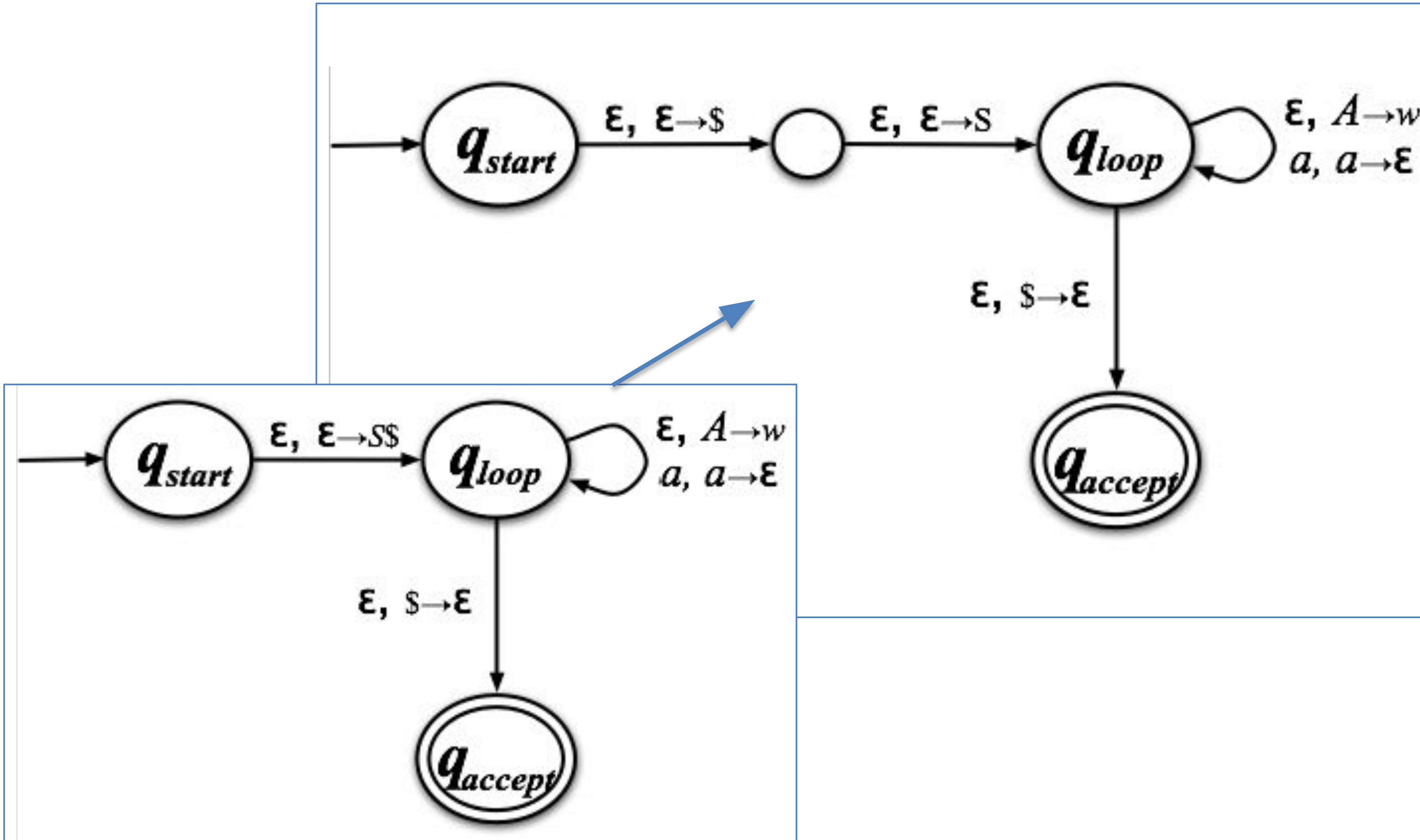


Recognizing context-free languages

- Lemma 2.21: If a language is context-free, then some pushdown automaton recognizes it.
- Proof:



Shorthand for...



For example…

- Let's use this construction on:
- $G = (V, \Sigma, R, S)$, where

$$V = \{S\}$$

$$\Sigma = \{[,]\}$$

$$R = \{ \quad S \to_G \varepsilon, \quad S \to_G SS, \quad S \to_G [S] \}$$

Go backwards!

The proof

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a pushdown automaton.
- Assume WLOG (Without Loss Of Generality)
  - P has exactly one accept state q_{accept}
  - P empties its stack before accepting
  - Each transition does either a *push* or a *pop* (but not both)

We build a grammar G…

- Given $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$
- Construct G = (V, Σ, R, S), where
  - $V = \{A_{pq} \mid p,q \in Q\}$
    - Idea: design the rules so that A_{pq} generates all strings that take P from p with empty stack to q with empty stack
  - $S = A_{q_0, q_{accept}}$

Designing the rules

P's operation on strings of A_{pq}

- Since P starts and ends with an empty stack:
  - The first move from p must be a *push*
  - The last move to q must be a *pop*
- Along the way, either:
  1. The stack never becomes empty
  2. There is some intermediate state where the stack is empty

Case 1: The stack never empties between p and q

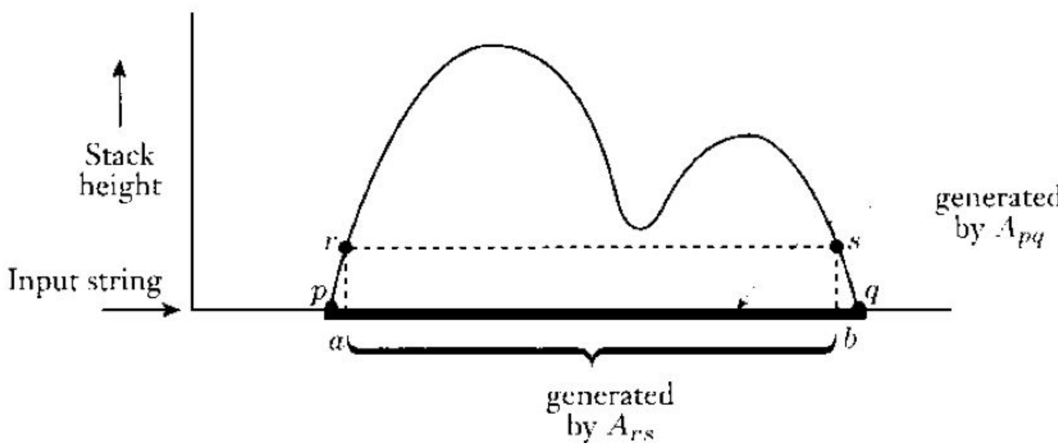
- On the first move from p
  - Let r be the state moving to
  - Let a be the input symbol read
  - Let t be the stack symbol pushed

- On the last move to q
  - Let s be the state moving from
  - Let b be the input symbol read
  - It must be the case that t is the stack symbol popped

Capturing this behavior

- Model with the rule

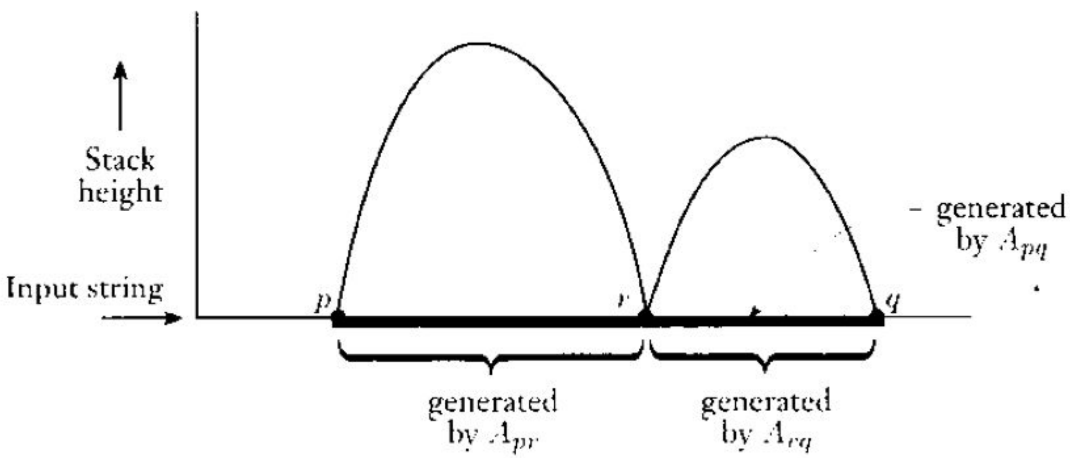
$$A_{pq} \rightarrow aA_{rs}b$$



Case 2: the stack empties along the way from p to q

- Let r be the state where the stack is empty
- Model with the rule

$$A_{pq} \rightarrow A_{pr}A_{rq}$$



Formally phrasing the rules

- If $(r, t) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, t)$ then add the rule $A_{pq} \rightarrow aA_{rs}b$
- For each $p, q, r, \in Q$, add the rule $A_{pq} \rightarrow A_{pr}A_{rq}$
- For each $p \in Q$, add the rule $A_{pp} \rightarrow \varepsilon$

Proving we were right

- A_{pq} generates x if and only if x can bring P from p with empty stack to q with empty stack

- => (Claim 2.30) If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack
- Proof
  – By induction on the number of steps in the derivation of x from A_{pq}

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack

Proof:

By (strong) induction on # steps in derivation.

<u>Base case</u>: *1 step.*

- What rule is it?

<u>IH</u>: *Assume for at most k steps.*

<u>Inductive step</u>: *k+1 steps.*

- What could first rule applied look like?

And now the other way!

- Claim 2.31: If x can bring P from p with empty stack to q with empty stack, then A_{pq} generates x
- Proof
  - By (strong) induction on the number of steps in the computation of P
  - Base case: *0 steps*.
    - starts and ends in same state
    - no time to read symbol, must read ε
  - IH: *Assume for at most k steps*
  - Inductive case: *k+1 steps*.
    - what could happen to the stack along the way?