BU CS 332 – Theory of Computation

Lecture 8:

- Equivalence between PDAs and CFGs
- Closure Properties

Reading:

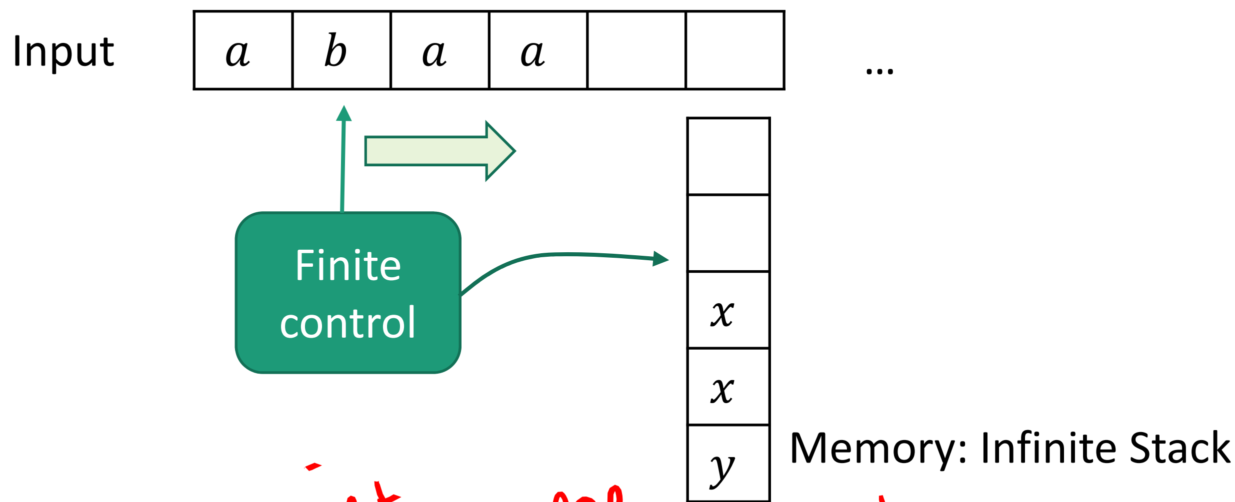
Sipser Ch 2.2

Mark Bun

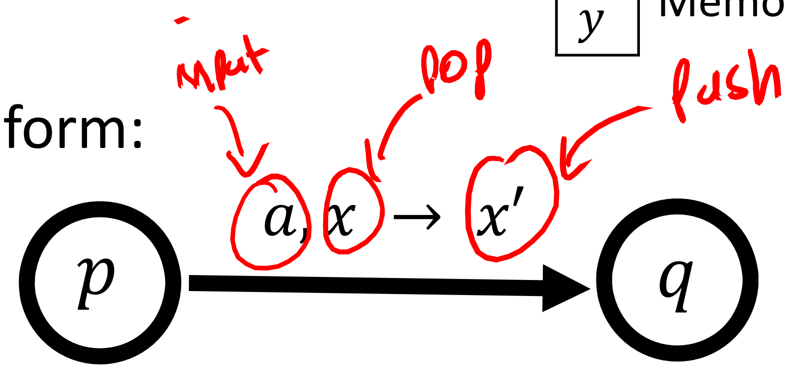
February 18, 2020

Pushdown Automaton (the idea)

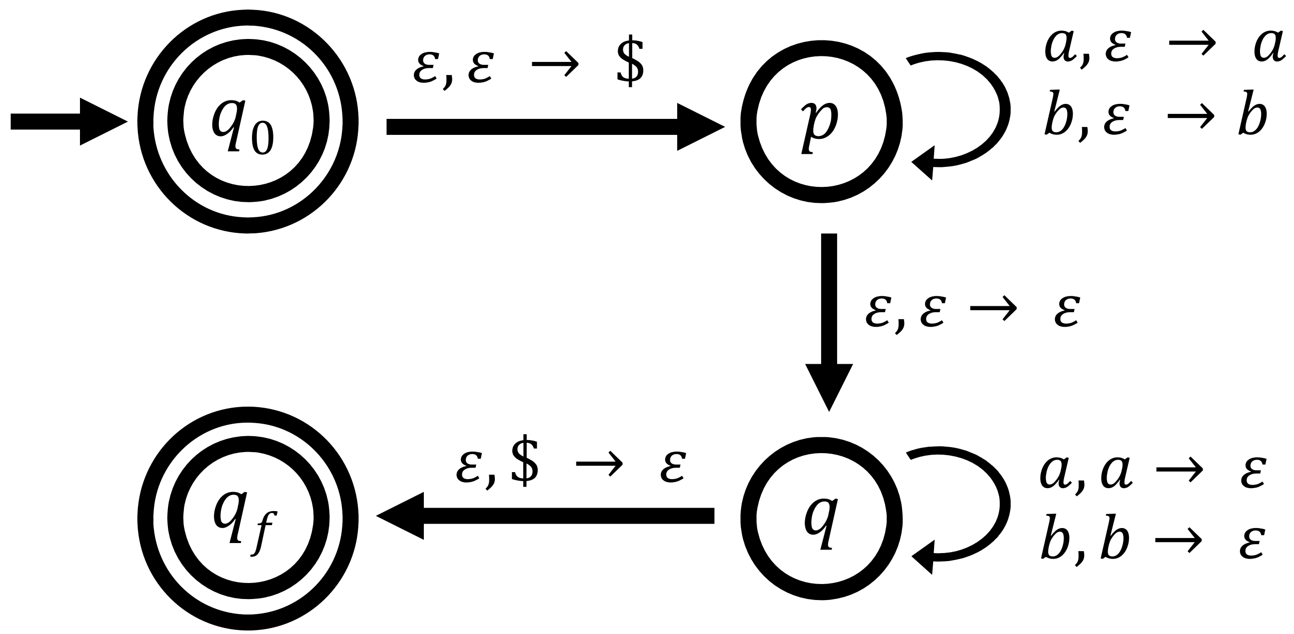
- Nondeterministic finite automaton + stack
- Stack has unlimited size, but machine can only manipulate (push, pop, read) symbol at the top



Transitions of the form:



Example: Even Palindromes



Pushdown Automaton (formal)

A PDA is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

- Q is a finite set of states
- Σ is the input alphabet
- Γ is the stack alphabet
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ is the transition function
- q_0 is the start state
- F is the set of final states

$p \xrightarrow{a, x \to x'} q$

M ***accepts*** a string w if, starting from q_0 and an empty stack, **there exists** a path to an accept state that can be followed by reading all of w.

Example

$L = \{w | w$ has an equal number of a's and b's$\}$

a bbb aa

push pop

Algorithmic description

1. Push $ to stack

2. Repeat indefinitely:

Do one of the following:

- Read character & push to the stack
- Match next character to stack & pop it

Next char = a, stack = b

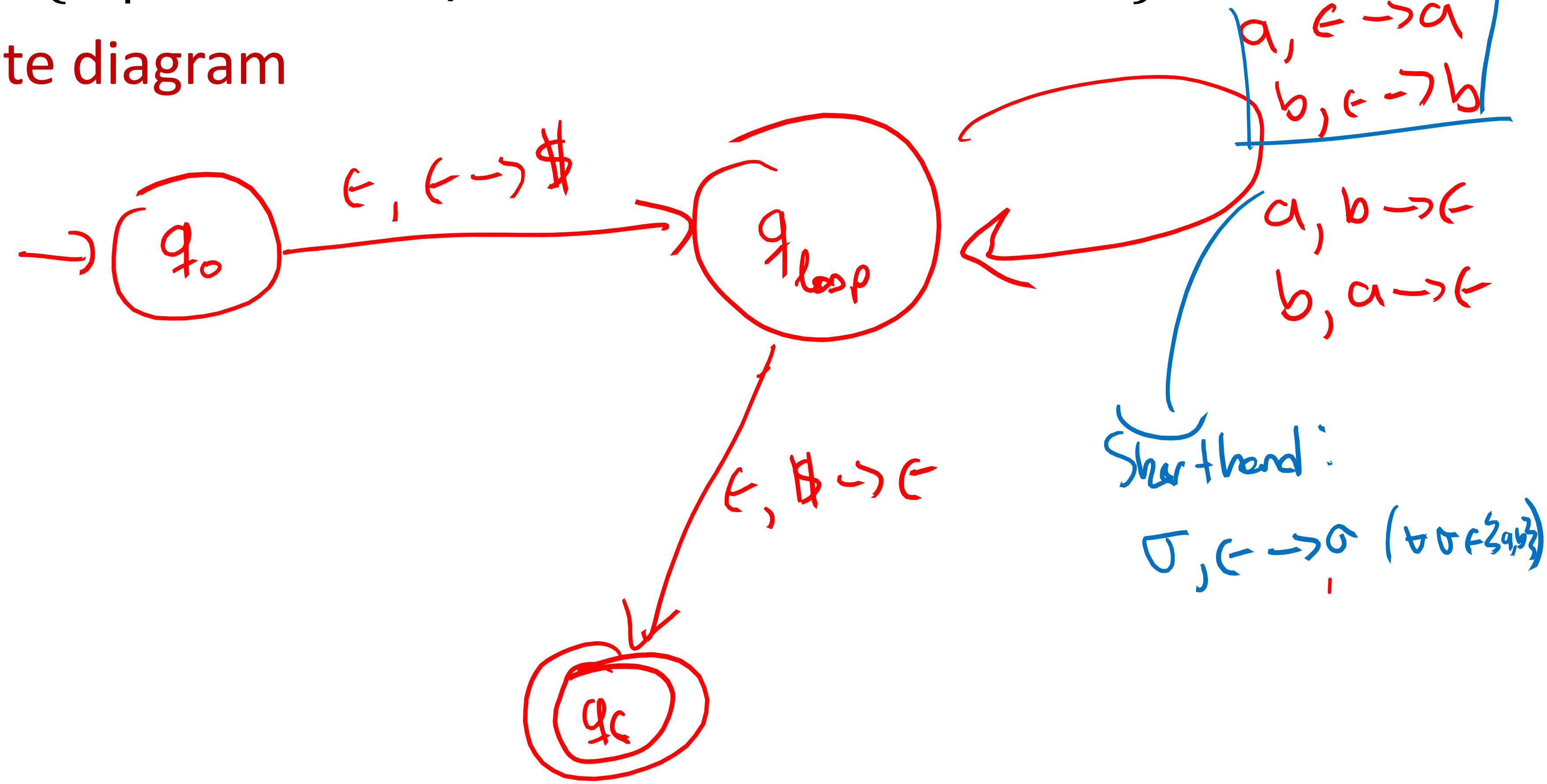
Next char = b, stack = a

- Accept if $ on the stack

Example

$L = \{w \mid w$ has an equal number of a's and b's$\}$

State diagram



CFGs vs. PDAs

The language $L(M)$ of a PDA M is the set of all strings it accepts.

Theorem: The class of languages recognized by PDAs is exactly the context-free languages.

Theorem 1: Every CFG has an equivalent PDA

Theorem 2: Every PDA has an equivalent CFG

CFG -> PDA

CFG -> PDA Conversion

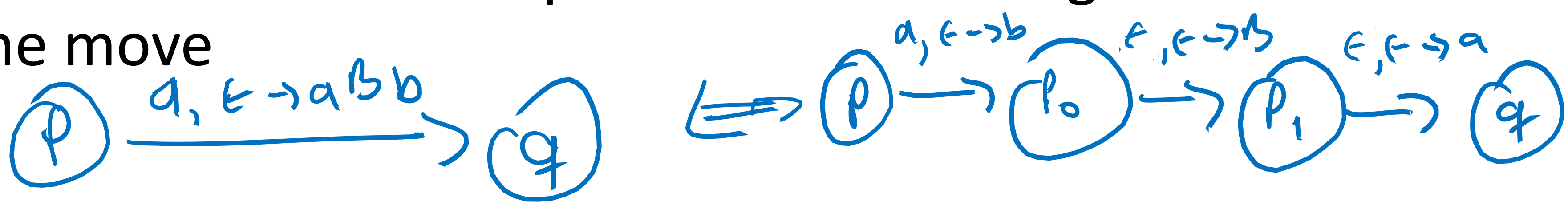
Suppose language L is generated by CFG $G = (V, \Sigma, R, S)$

Goal: Construct a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ recognizing L

Idea: M will guess the steps of the CFG derivation of its input w, and use its stack to check the derivation

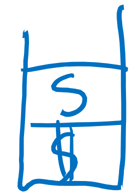
A helpful intermediate abstraction

Generalized PDA: Can push an entire string to the stack in one move



Algorithmic Description

1. Place $\$$ and the start variable S on the stack

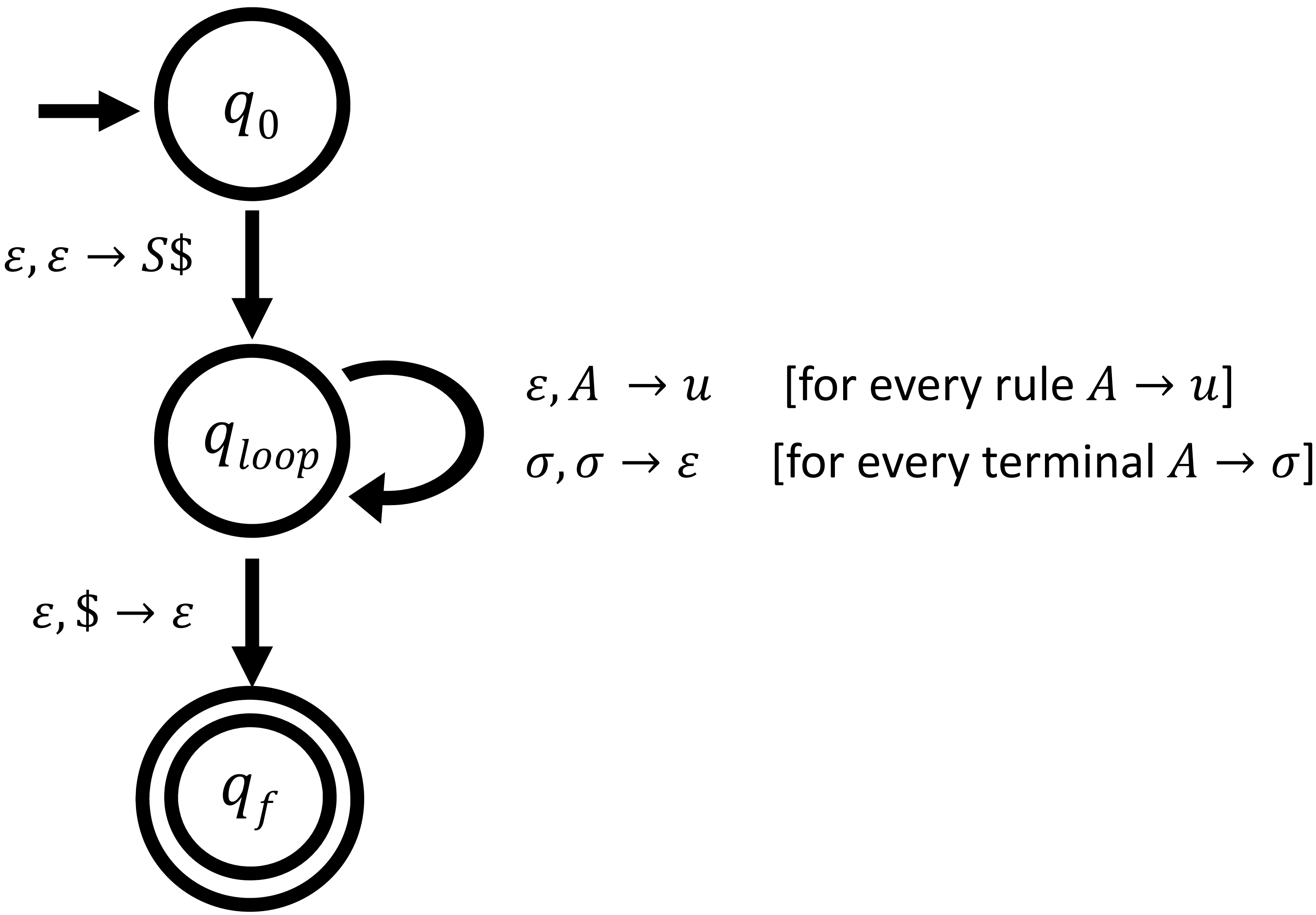


2. Repeat forever:
   a) If the top of the stack holds variable A:
      Nondeterministically guess a rule $(A \to u) \in R$
      Replace A with u on the stack

   b) If the top of the stack holds terminal σ:
      Pop σ and verify that it matches the next input char

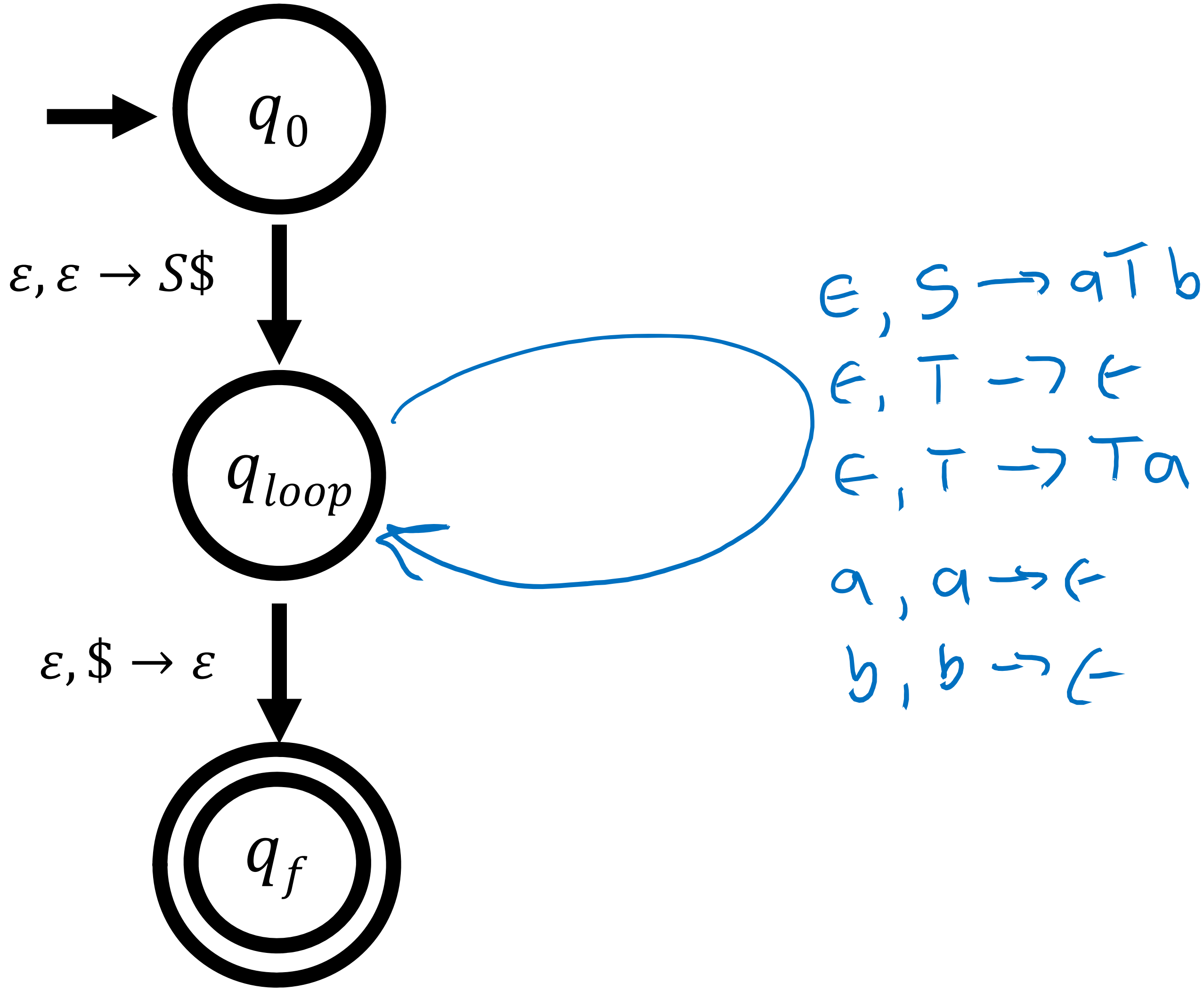
   c) If the top of the stack holds $\$$:
      Accept if the input is empty

State Diagram



Example

$$S \rightarrow aTb$$
$$T \rightarrow Ta \mid \varepsilon$$



$S \Rightarrow aTb$
$\Rightarrow aTab$
$\Rightarrow \underline{aab}$

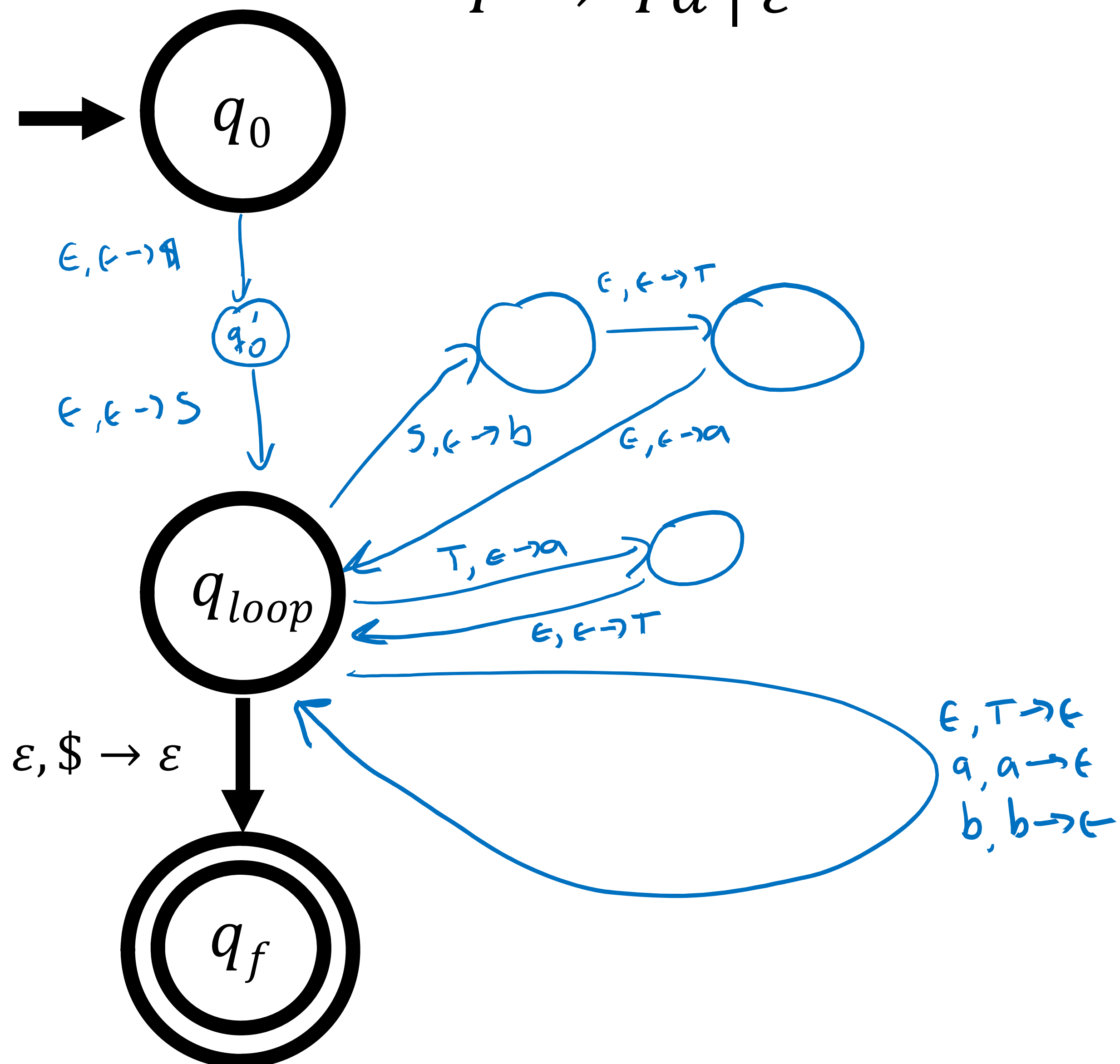
Input aab

Stack

$

Example

$$S \rightarrow aTb$$
$$T \rightarrow Ta \mid \varepsilon$$



PDA -> CFG

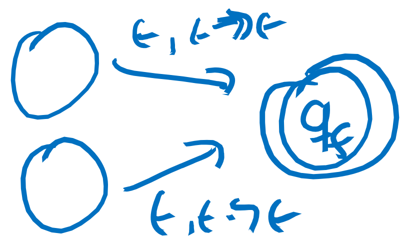
PDA -> CFG Conversion

Theorem 2: Every PDA has an equivalent CFG

Suppose L is recognized by PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

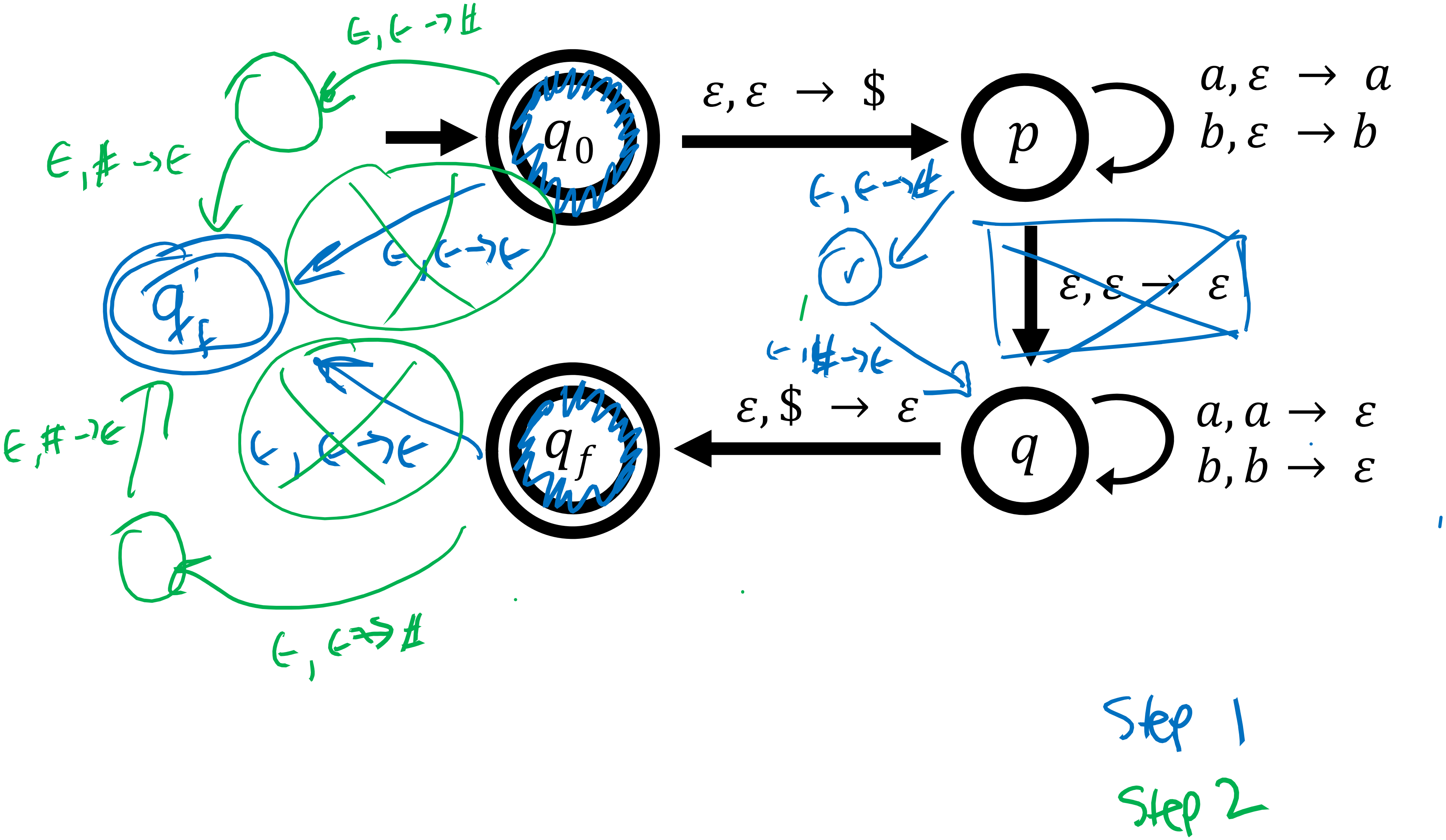
Goal: Construct a CFG $G = (V, \Sigma, R, S)$ generating L

First simplify M so that:



1. It has a single accept state q_f
2. It empties the stack before accepting
3. Every transition either pushes a symbol or pops a symbol (but not both)

Simplification Example



Conversion Idea

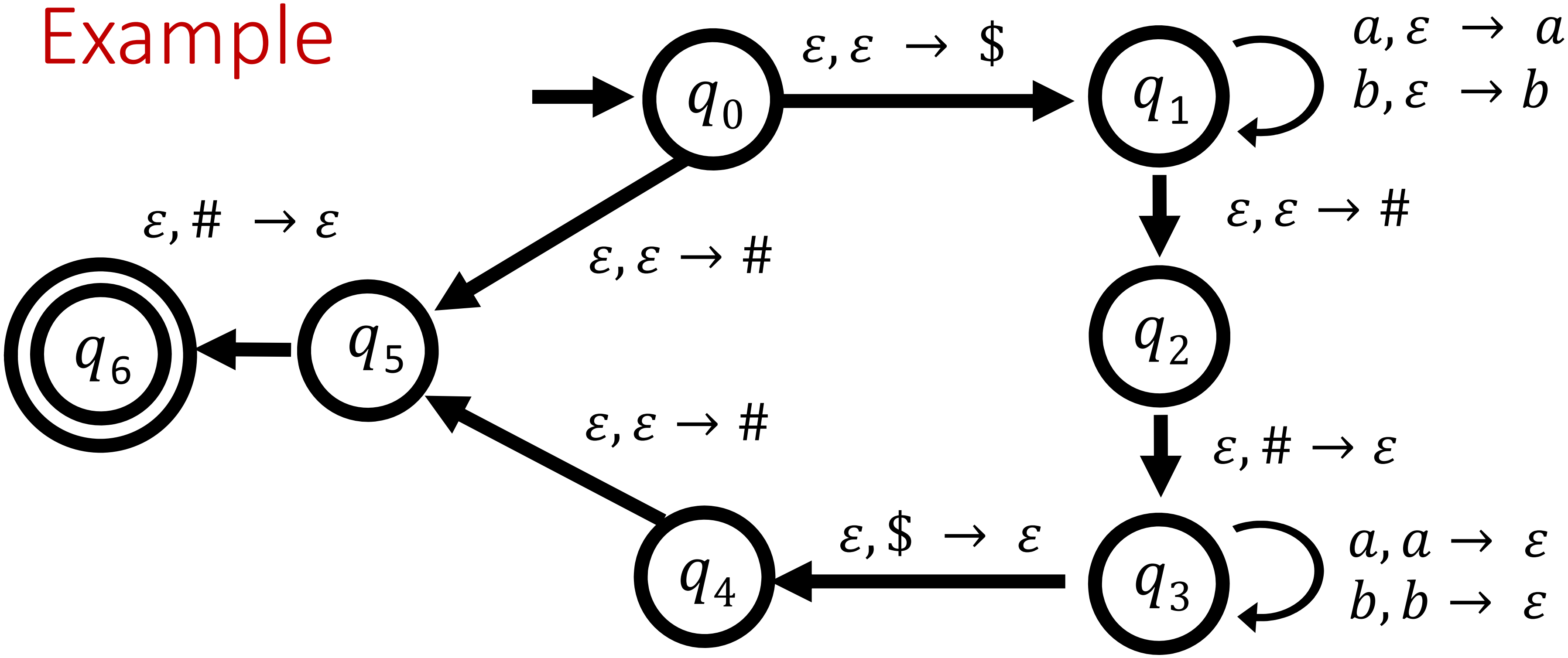
Variables: A_{pq} for every pair of states p, q in PDA M

Idea: A_{pq} generates all strings that can take M from p (with an empty stack) to q (with an empty stack)

$V = \{A_{pq} \mid p, q \in Q\}$

$S = A_{q_0 q_{final}}$

Example



What strings should $A_{q_0 q_1}$ generate? None

What strings should $A_{q_1 q_3}$ generate? $\{ ww^R \mid w \in \{a,b\}^* \}$

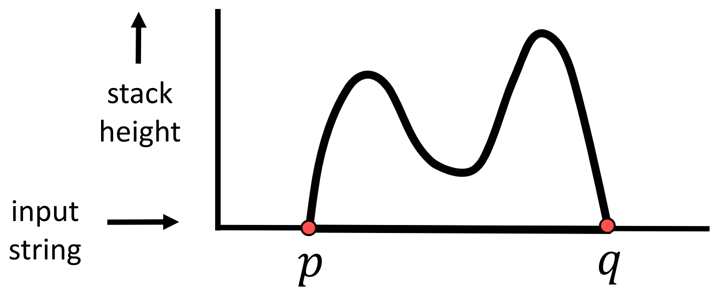
What strings should $A_{q_1 q_4}$ generate? None

What rules should define A_{pq}?

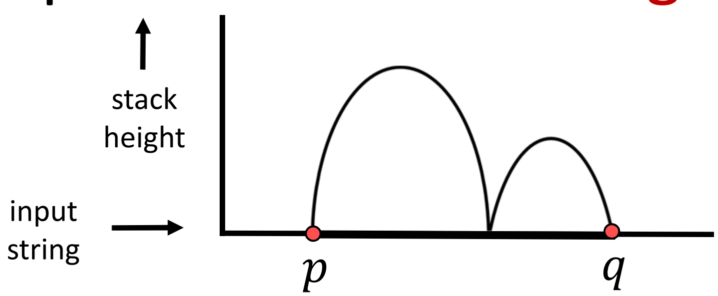
Let x be a string generated by A_{pq}

Two cases:

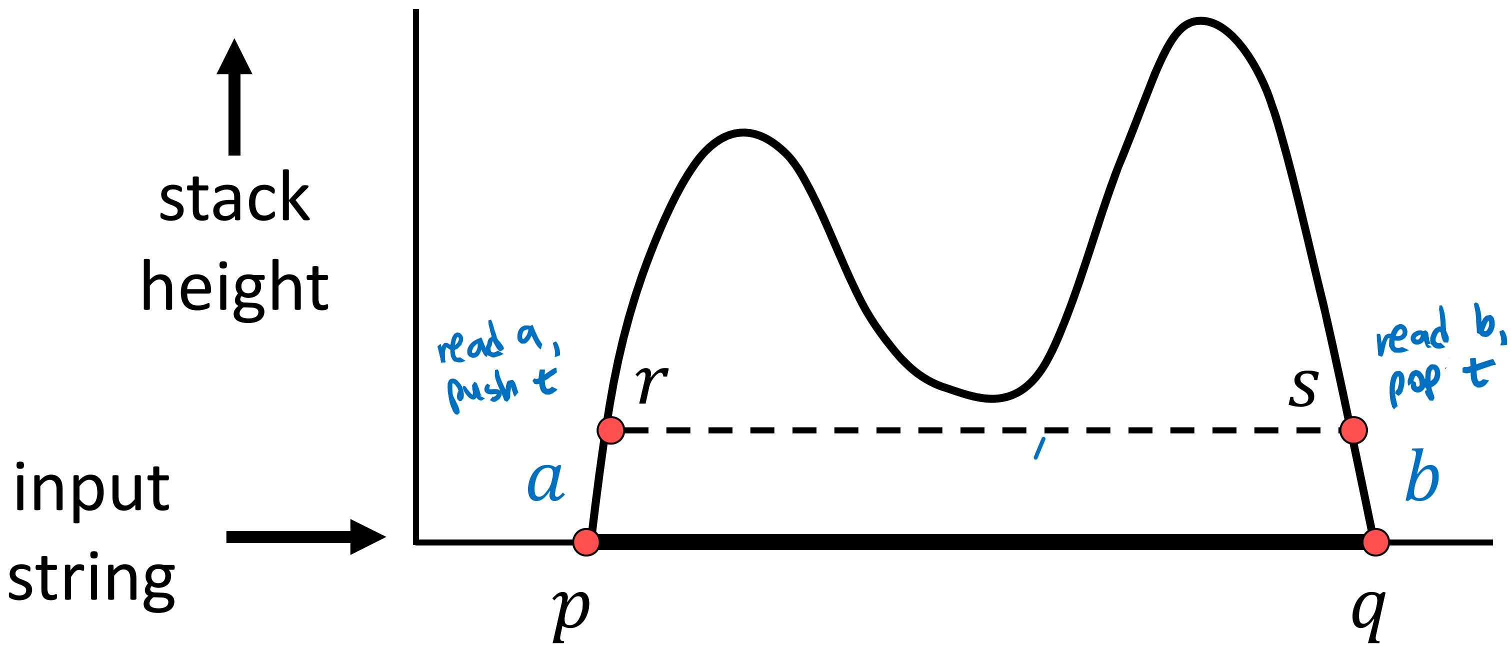
1) Stack first empties after reading all of x



2) Stack empties before reaching the end of x

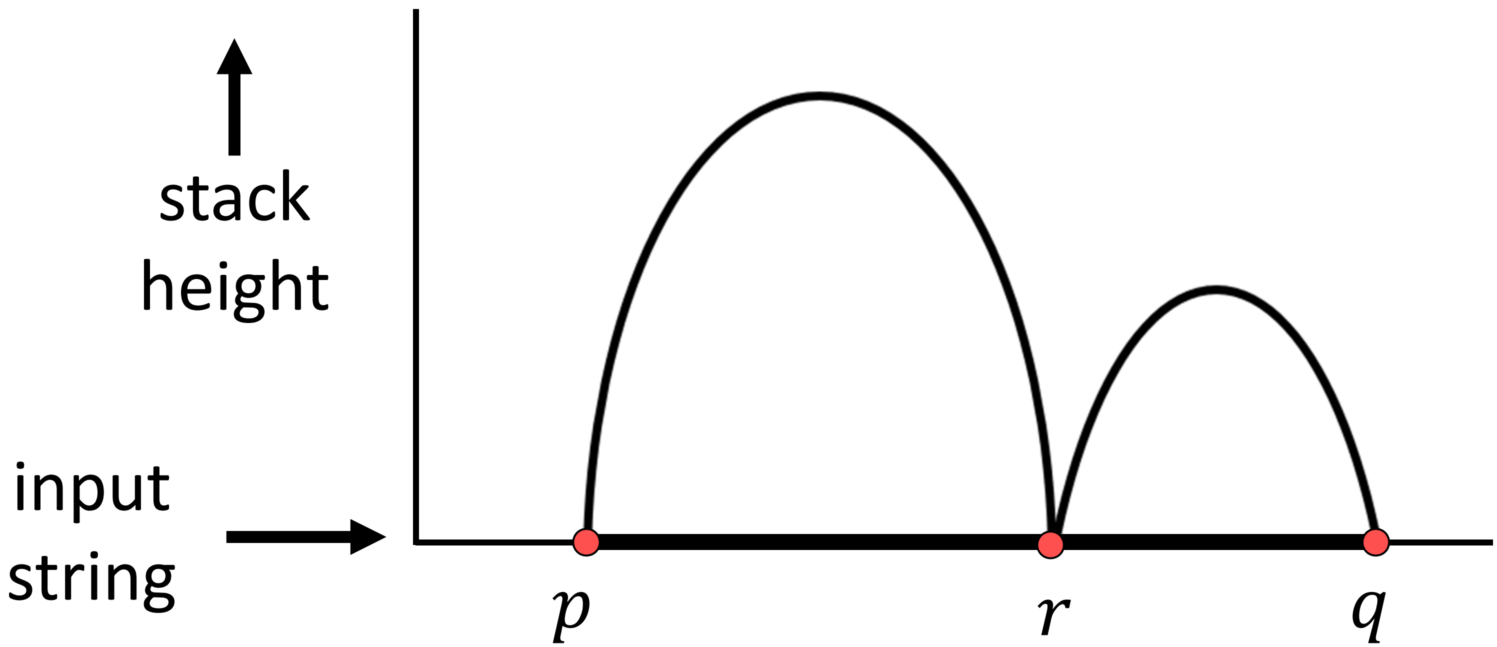


1. Stack first empties after reading all of x



Add rule $A_{pq} \to aA_{rs}b$

2. Stack empties before reaching the end of x



Add rule $A_{pq} \to A_{pr}A_{rq}$

Formal CFG Construction

$V = \{A_{pq} \mid p, q \in Q\}$

$S = A_{q_0 q_f}$

Three kinds of rules:

1. For every $p, q, r, s \in Q$, $t \in \Gamma$, $a, b \in \Sigma_\varepsilon$:

   current state   stack

   If $(r, t) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, t)$,

   new state   new stack   next char

   include the rule $A_{pq} \to aA_{rs}b$

2. For every $p, q, r \in Q$, include the rule $A_{pq} \to A_{pr}A_{rq}$
3. For every $p \in Q$, include the rule $A_{pp} \to \varepsilon$

Sketch of proof that CFG generates $L(M)$

Claim: $A_{pq} \Rightarrow^* x$ if and only if x takes M from p to q, beginning and ending with empty stack

Proof idea:

$\Longrightarrow$ Induction on number of steps of derivation of x from A_{pq}

$\Longleftarrow$ Induction on number of steps of computation taking M from p to q

Closure Properties

Closure Properties

- The class of CFLs is closed under

    Union

    Concatenation

    Star

    regular operations

    Intersection with *regular* languages

    A context-free, B regular ⇒ A ∩ B CFL

- Beware: It is not closed under intersection or complement

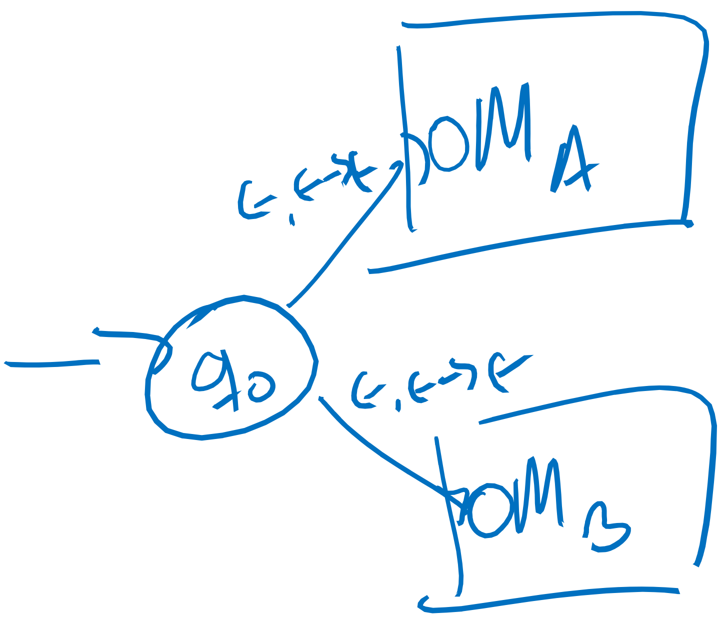
    (Find counterexamples!)

    A CFL, B CFL, A ∩ B not CFL

Closure under union (Proof 1)

Let A be a CFL recognized by PDA M_A and let B be a CFL recognized by PDA M_B

Goal: Construct a PDA recognizing $A \cup B$



Closure under union (Proof 2)

Let A be a CFL generated by CFG G_A and let B be a CFL recognized by CFG G_B

Goal: Construct a CFG G recognizing $A \cup B$

$G_A = (V_A, \Sigma_A, R_A, S_A)$

$G_B = (V_B, \Sigma_B, R_B, S_B)$

Relabel variables so V_A and V_B are disjoint

Let $G = (V, \Sigma, R, S)$

$R = R_A \cup R_B \cup \{S \to S_A, S \to S_B\}$