

Theory of Computation

Chapter 2

Equivalence of PDAs and CFGs



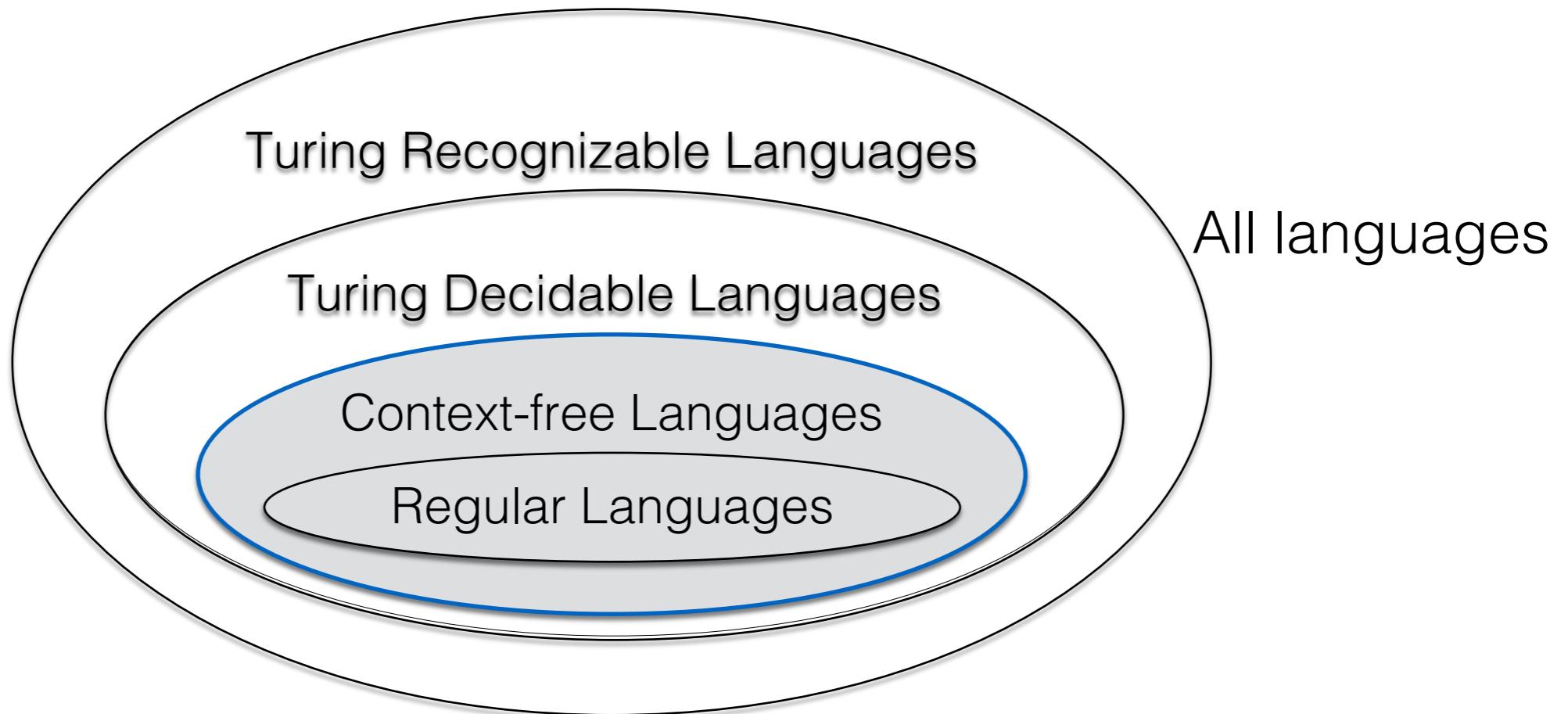
School of Engineering | Computer Science
1

Donald Knuth



- Author of multi-volume “The Art of Computer Programming” - *The books of algorithms*
 - Impact akin to the Bible’s impact on Christianity, or Pride and Prejudice’s impact on romance novels
- “Father” of the analysis of algorithms
- Turing Award, 1974
- Creator of TeX computer typesetting language
- Knuth used to pay a finder's fee of \$2.56 for any typographical errors or mistakes discovered in his books, because "256 pennies is one hexadecimal dollar"

Context-Free Languages



Context-Free Languages
PDA = CFG = CNF

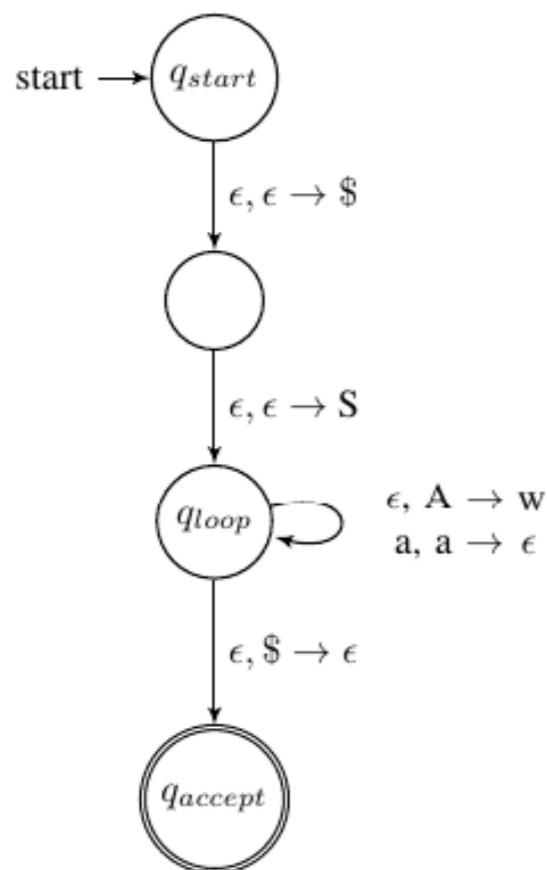
Closed under union, U, concatenation, \circ , and star, $*$.

Equivalence of PDAs and CFGs

- Theorem 2.20: A language is context-free if and only if a PDA recognizes it (iff, so two directions)
- Forward: Lemma 2.21
 - Every context-free language has a pushdown automaton that recognizes it
 - Proof: Let L be a CFL with a CFG, $G = (V, \Sigma, R, S)$. We construct a PDA $M = (Q, \Sigma', \Gamma, \delta, q_0, F)$ to recognize it.
 - In words:
 1. Design M to non-deterministically pick a derivation in G to follow to see if it derives the given input x
 2. Use a stack to store variables and terminals during the derivation. When a variable is popped, non-deterministically choose a substitution rule to push onto the stack.

Equivalence of PDAs and CFGs

- Theorem 2.20: A language is context-free if and only if a PDA recognizes it (iff, so two directions)
- Forward: Lemma 2.21- Construction of the PDA
 - Start with 3 states: q_{start} , q_{loop} , q_{accept}



$\epsilon, \epsilon \rightarrow \$$ is the starting transitions:
 $\epsilon, \epsilon \rightarrow S$ pushes on the start variable
 $\epsilon, A \rightarrow w$ - is a transition inserted for the rules: $A \rightarrow w$ in G, where w is a string of terminals and variables
 $a, a \rightarrow \epsilon$ - is a transition inserted for each terminal a

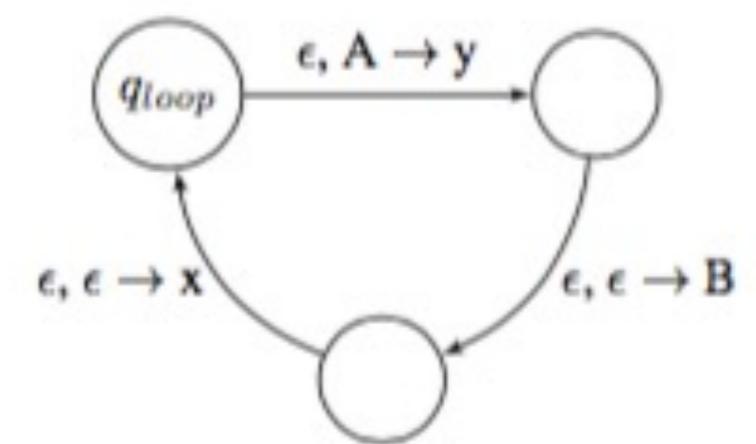
$\epsilon, \$ \rightarrow \epsilon$ is the last transition to the accept state

Equivalence of PDAs and CFGs

- Theorem 2.20: A language is context-free if and only if a PDA recognizes it (iff, so two directions)
- Forward: Lemma 2.21- Construction of the PDA
 - Two types of transitions are added to the loop state
 - $\epsilon, A \rightarrow w$ - is a transition inserted for the rules: $A \rightarrow w$ in G, where w is a string of terminals and variables
 - $a, a \rightarrow \epsilon$ - is a transition inserted for each terminal a
 - Ex: If there was a rule in the format of $A \rightarrow w$: such as:
 - $A \rightarrow xBy$
 - You would add the transition rules:
 - $\epsilon, A \rightarrow y$
 - $\epsilon, \epsilon \rightarrow B$
 - $\epsilon, \epsilon \rightarrow x$

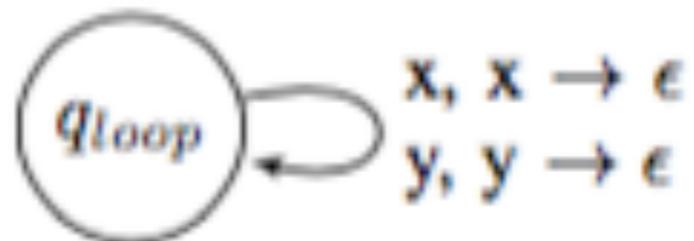
Stack	
	x
	B
A	y

- $A \rightarrow xBy$
- You would add the transition rules:
 - $\epsilon, A \rightarrow y$
 - $\epsilon, \epsilon \rightarrow B$
 - $\epsilon, \epsilon \rightarrow x$



Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA
 - Two types of transitions are added to the loop state
 - $\epsilon, A \rightarrow w$ - is a transition inserted for the rules: $A \rightarrow w$ in G, where w is a string of terminals and variables
 - $a, a \rightarrow \epsilon$ - is a transition inserted for each terminal a
 - Ex: If there was a rule in the format of $A \rightarrow w$: such as:
 - $A \rightarrow xBy$
 - You would add the transitions: $\epsilon, A \rightarrow y; \epsilon, \epsilon \rightarrow B; \epsilon, \epsilon \rightarrow x$ in that order
 - You would also add the transitions: $x, x \rightarrow \epsilon$ and $y, y \rightarrow \epsilon$ for the terminals x and y



Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA
 - $Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$ (E is the set of states needed to implement the $\varepsilon, A \rightarrow w$ transitions shown in the slides above)
 - $q_0 = q_{\text{start}}$
 - $F = q_{\text{accept}}$
 - $\Sigma' = \Sigma$
 - $\Gamma = \Sigma \cup V \cup \{\$\}$ (V = variables)
 - δ = the transitions from the rules as shown in the previous three slides

Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA
 - Ex 2.25:
 - CFG G: $S \rightarrow aTb \mid b$
 $T \rightarrow Ta \mid \epsilon$
 - PDA:

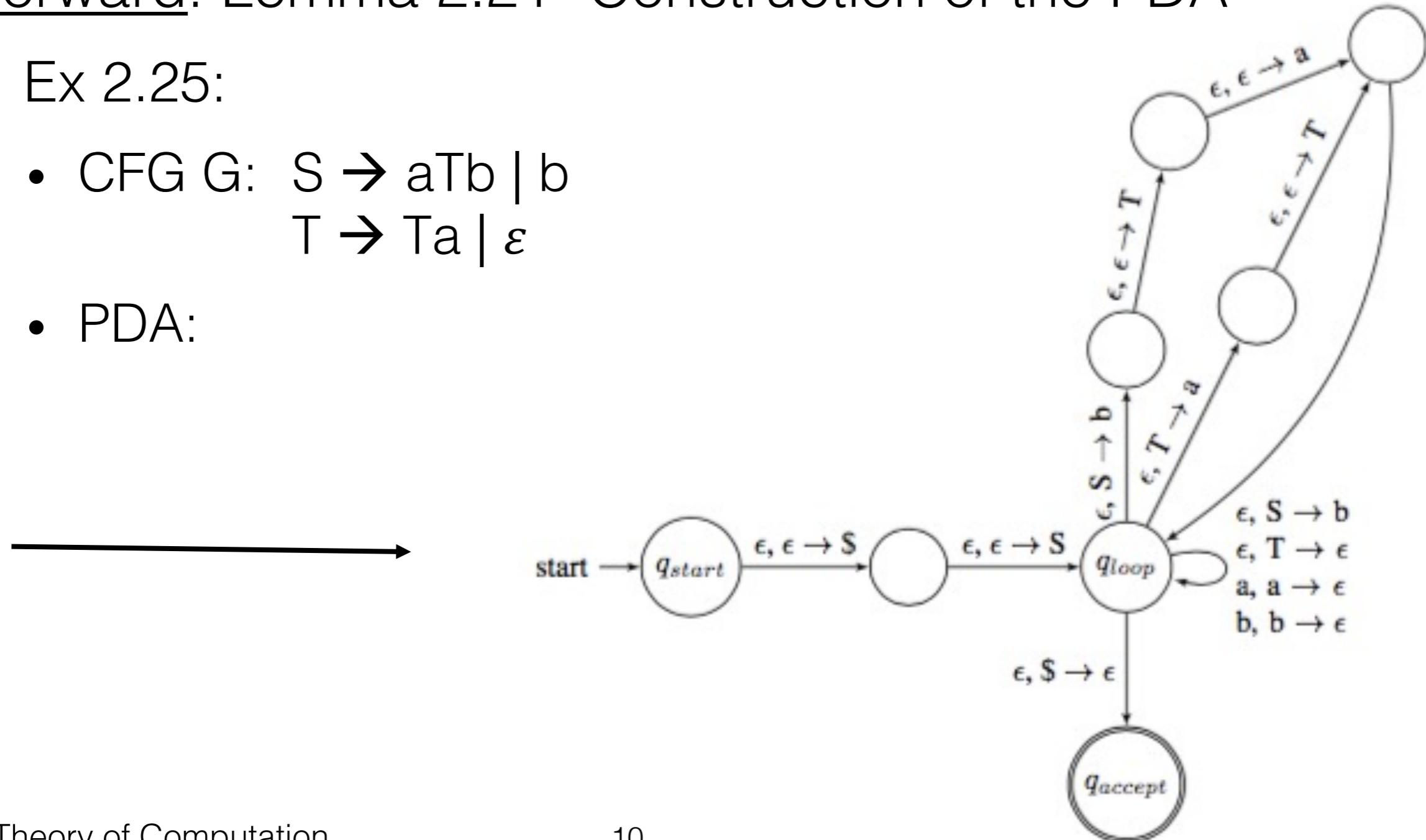
Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA

- Ex 2.25:

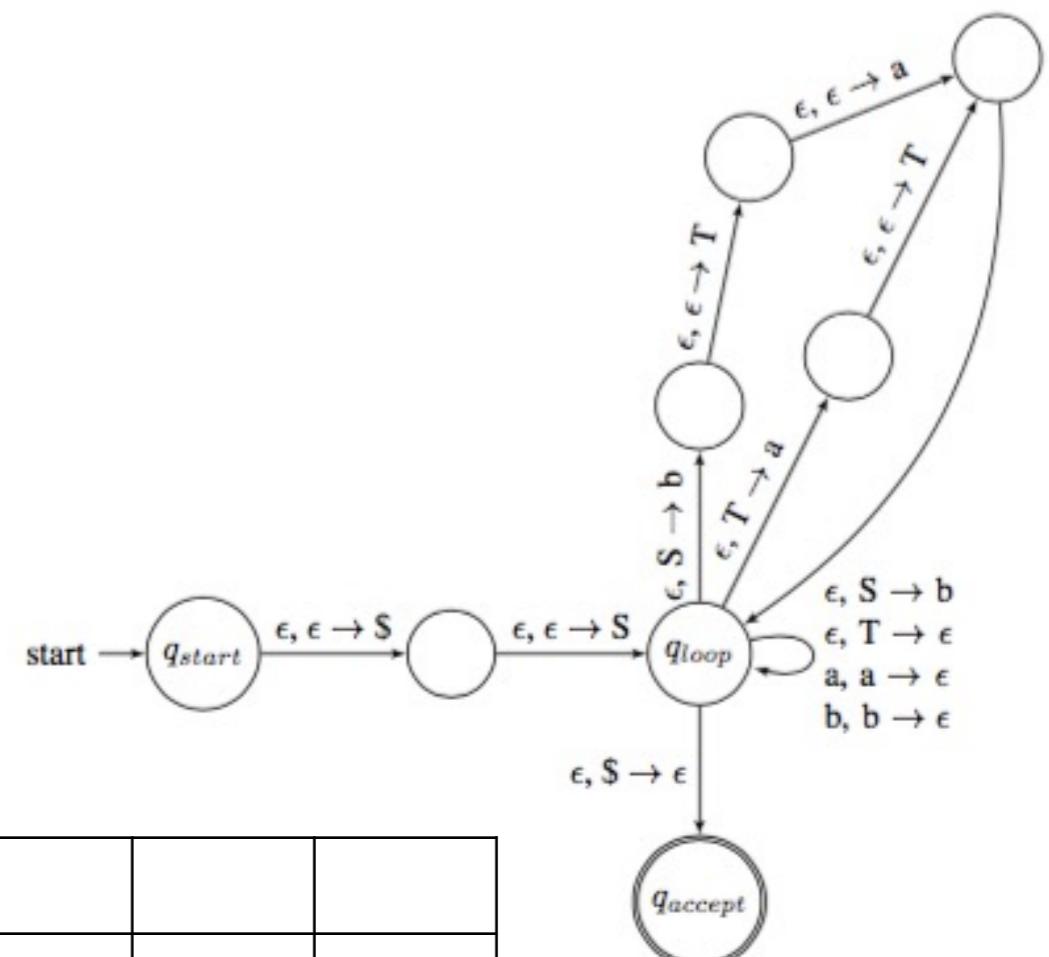
- CFG G: $S \rightarrow aTb \mid b$
 $T \rightarrow Ta \mid \epsilon$

- PDA:



Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA
 - Ex 2.25:
 - CFG G: $S \rightarrow aTb \mid b$
 $T \rightarrow Ta \mid \epsilon$
 - PDA: (stack example below)
 - $S \rightarrow aTb \rightarrow ab$



Equivalence of PDAs and CFGs

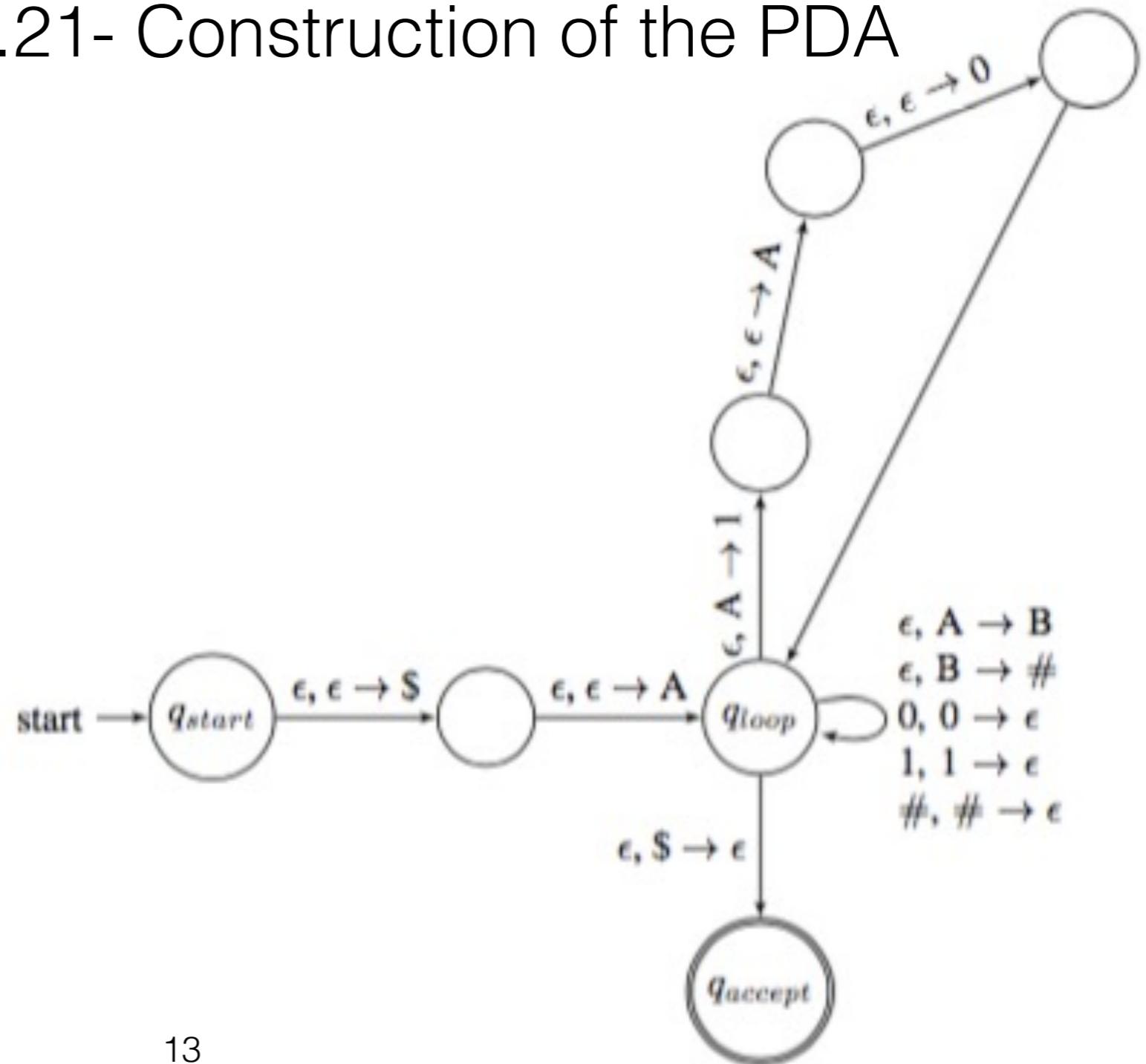
- Forward: Lemma 2.21- Construction of the PDA
 - Ex 2:
 - CFG G: $A \rightarrow 0A1$
 $A \rightarrow B$
 $B \rightarrow \#$
 - PDA:

Equivalence of PDAs and CFGs

- Forward: Lemma 2.21- Construction of the PDA

- Ex 2.25:
 - CFG $G: A \rightarrow 0A1$
 - $A \rightarrow B$
 - $B \rightarrow \#$

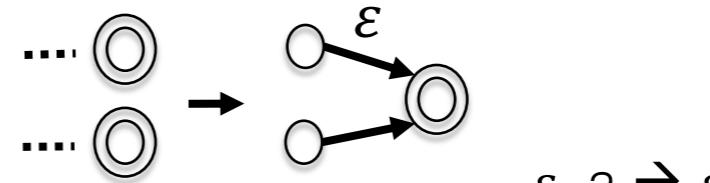
- PDA:



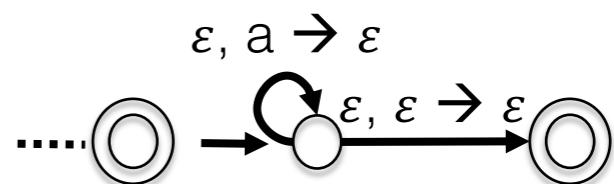
Equivalence of PDAs and CFGs

- Theorem 2.20: A language is context-free if and only if a PDA recognizes it (iff, so two directions)
- Backwards: Lemma 2.27 Every PDA has an equivalent CFG
 - Proof Idea: Let PDA P recognize L , construct a CFG G to generate L
 - Assume P Satisfies:

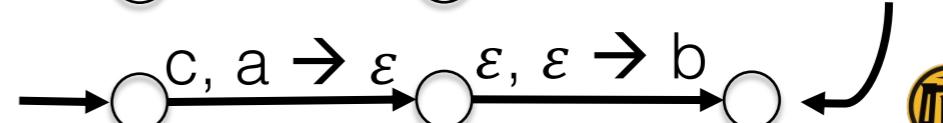
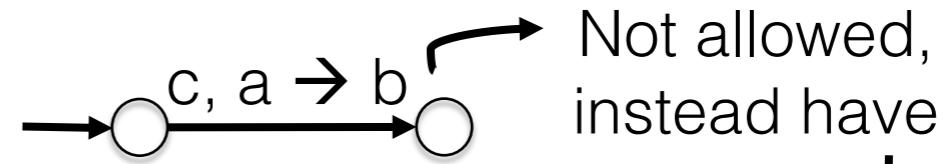
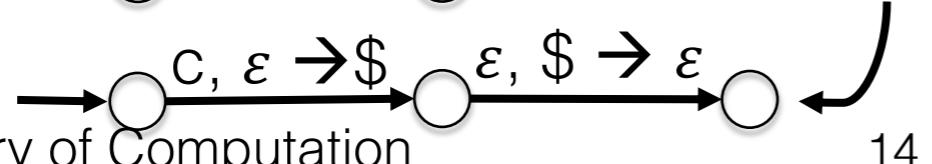
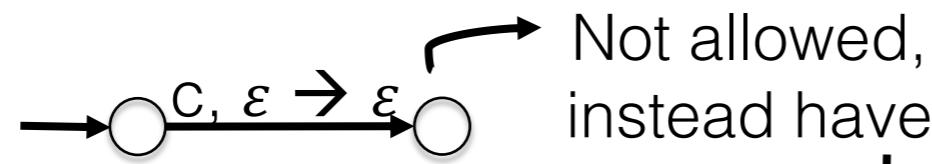
1. P has a single accept state



2. P empties its stack before accepting

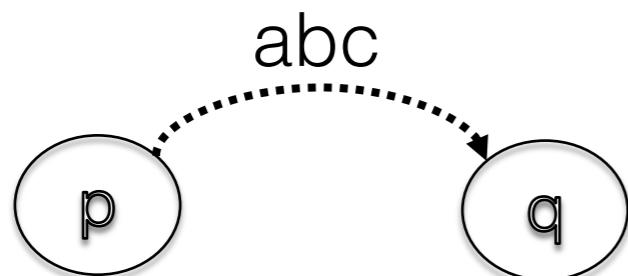


3. Each transition of P either pushes a symbol on the stack or pops one off, but not both or neither



Equivalence of PDAs and CFGs

- Backwards: Lemma 2.27
 - Start with an empty stack on p, and end with an empty stack on q. Between these two states, design G so that A_{pq} generates all strings that take P from p to q.



- In the CFG, “add rule” $R_p \rightarrow abc R_q$

Equivalence of PDAs and CFGs

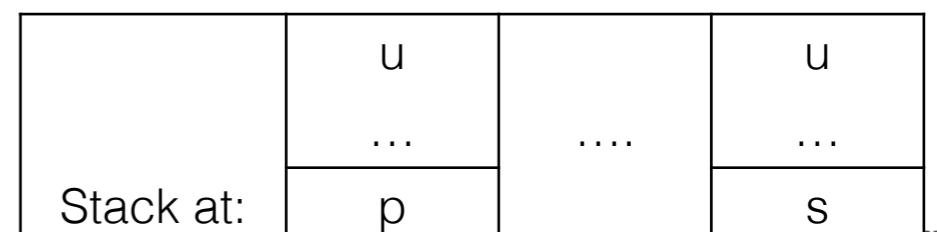
- Backwards: Lemma 2.27
 - Let PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$, construct CFG $G = (V, \Sigma', R, S)$ as follows:
 - $V = \{A_{pq} \mid p, q \in Q\}$
 - $\Sigma' = \Sigma$
 - S (start variable) = Aq_0q_{accept}
 - $R = \text{Cont. } \rightarrow$

Equivalence of PDAs and CFGs

- Backwards: Lemma 2.27
 - Let PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$, construct CFG $G = (V, \Sigma', R, S)$ as follows:
 - R (substitution rules) as follows:
 1. For each $p \in Q$, add rule $A_{pp} \rightarrow \varepsilon$ to R
 2. For each $p, q, r \in Q$, add rule $A_{pq} \rightarrow A_{pr}A_{rq}$
 3. If P contains a path:



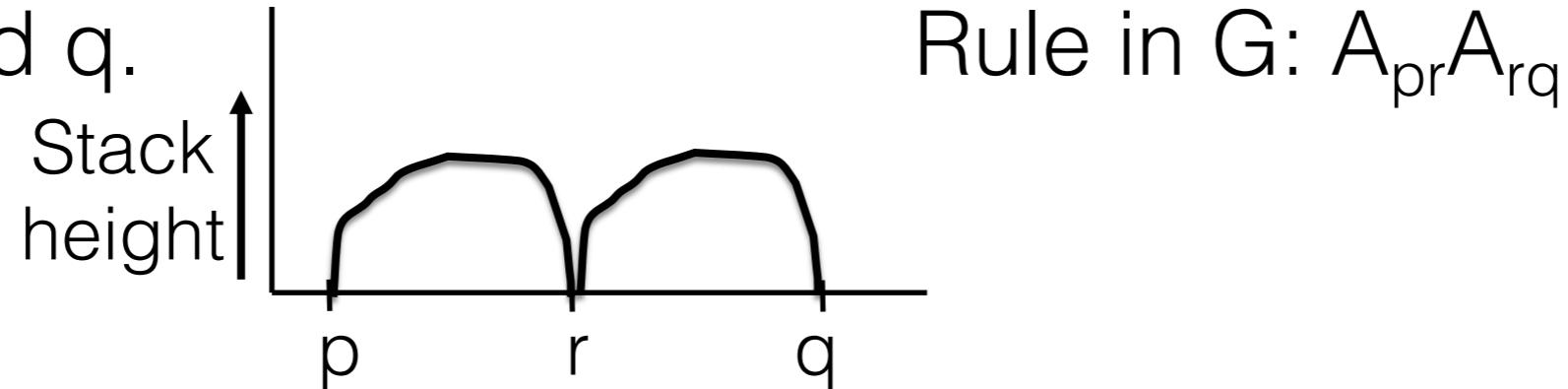
then add $A_{pq} \rightarrow aA_{rs}b$ to R



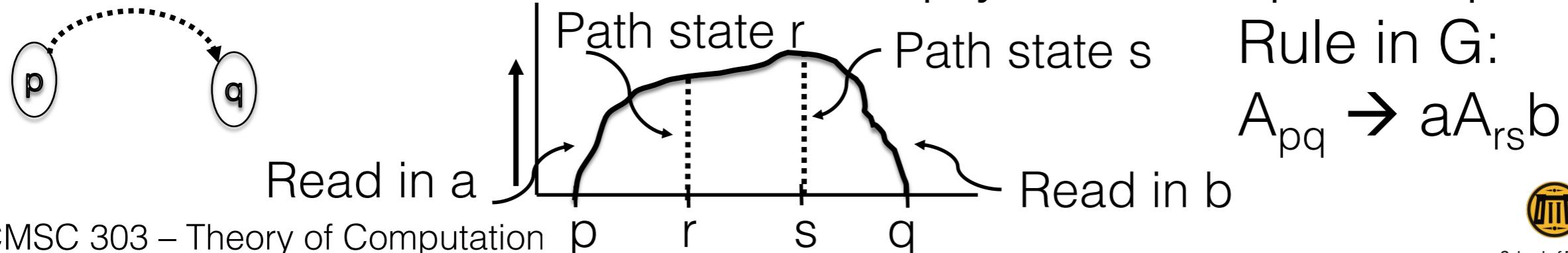
Equivalence of PDAs and CFGs

- Backwards: Lemma 2.27

- Each A_{pq} corresponds to transitions in P from p to q which start and end with the empty stack.
- Case 1: Stack is empty at some point r between p and q.



- Case 2: Stack is never empty between p and q.

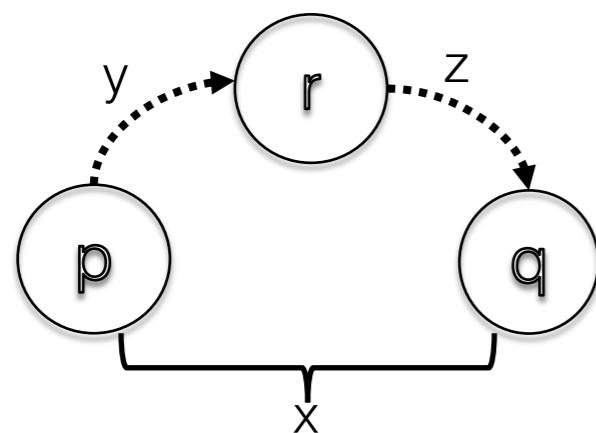


Equivalence of PDAs and CFGs

- Claim 2.31: If x can bring the PDA P from state p with an empty stack to q , then A_{pq} generates x
$$A_{pq} \Rightarrow^* x$$
- Proof by Induction: (based on the number of steps in P from p to q on input x)
 - Base Case: Path takes 0 transitions, so $p = q \Rightarrow x = \epsilon$
 - $A_{pp} \rightarrow \epsilon$, so this case works ($A_{pp} \rightarrow \epsilon$ is in CFG G)
 - Inductive Hypothesis: Assume true for paths of length at most k
 - Inductive Step: Prove for path of length $k+1$, cont.

Equivalence of PDAs and CFGs

- Claim 2.31:
- Proof by Induction: (based on the number of steps in P from p to q on input x)
 - Inductive Step: Prove for path of length $k+1$, cont.
 - Case 1: Stack becomes empty at some state $r \notin \{p, q\}$ during computation



Paths $p \rightarrow r$ and $r \rightarrow q$ have $\leq k$ steps.
This implies $A_{pr} \Rightarrow^* y$ and $A_{rq} \Rightarrow^* z$

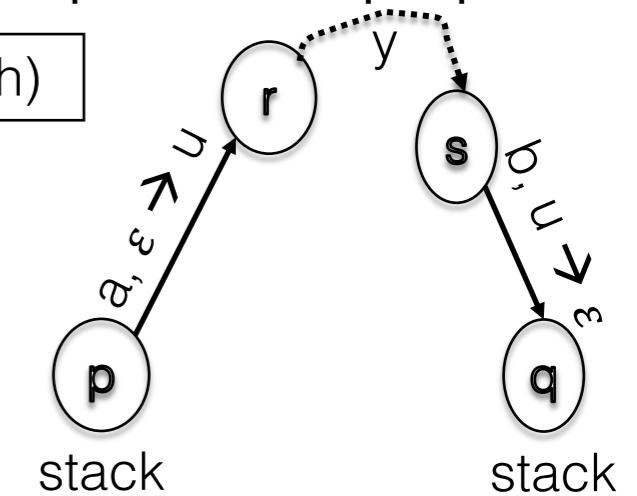
$$A_{pq} \rightarrow A_{pr} A_{rq} \Rightarrow yz = x$$

($A_{pq} \rightarrow A_{pr} A_{rq}$ is in CFG G)

Equivalence of PDAs and CFGs

- Claim 2.31:
- Proof by Induction: (based on the number of steps in P from p to q on input x)
 - Inductive Step: Prove for path of length k+1, cont.
 - Case 2: Stack empty only at p and q, since push or pop at each step:

$\delta(\text{state input pop}) \rightarrow (\text{state push})$
 - 1. First step has form: $\delta(p, a, \varepsilon) \rightarrow (r, u)$
 - 2. Last step has form: $\delta(s, b, u) \rightarrow (q, \varepsilon)$
 - Add rules: $A_{pq} \rightarrow aA_{rs}b$ to k
 - Suppose $x = ayb$, since P goes from p to q, y takes P from r to s in k-1 steps ($A_{rs} \Rightarrow^* y$, so $A_{pq} \Rightarrow^* ayb = x$)
 - ($A_{pq} \rightarrow aA_{rs}b$ is in CFG G)



Try It

- Construct the PDA from the given Grammar

CFG G: $S \rightarrow TS\#V \mid VST \mid \epsilon$

$T \rightarrow 0$

$V \rightarrow 1$

Try It

- Construct the PDA from the given Grammar

CFG G: $S \rightarrow TS\#V \mid VST \mid \epsilon$

$$T \rightarrow 0$$

V → 1

