PDA and CFG are equivalent in power

Pushdown automata are for context-free languages what finite automata are for regular languages.

Note: PDAs are nondeterministic.

PDAs versus CFL

Theorem 2.20: A language L is context-free if and only if there is a pushdown automata M that recognizes L.

Two step proof:

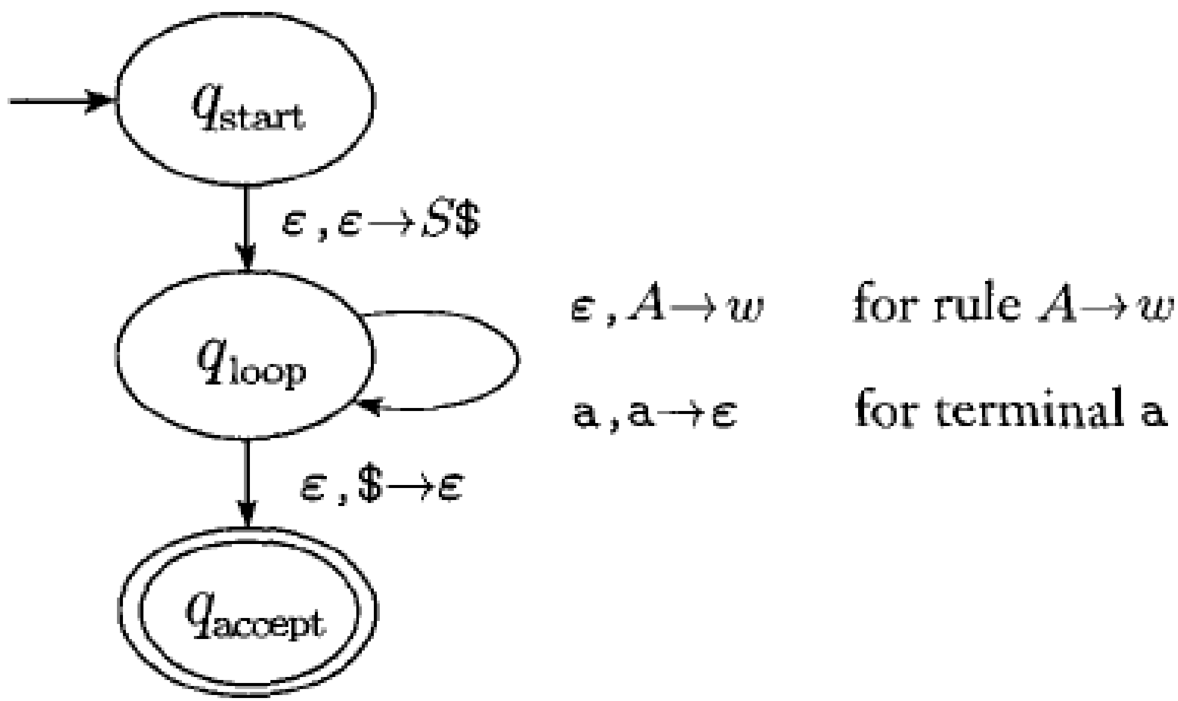
($\Longrightarrow$) Given a CFG G, construct a PDA M_G (proved last time)

($\Longleftarrow$) Given a PDA P, make a CFG G_P

Equivalence of PDA and CFG (review)

(⟹) : For every CFG, we can build an equivalent PDA.

General construction: each rule of CFG A → ω is included in the PDA's move.



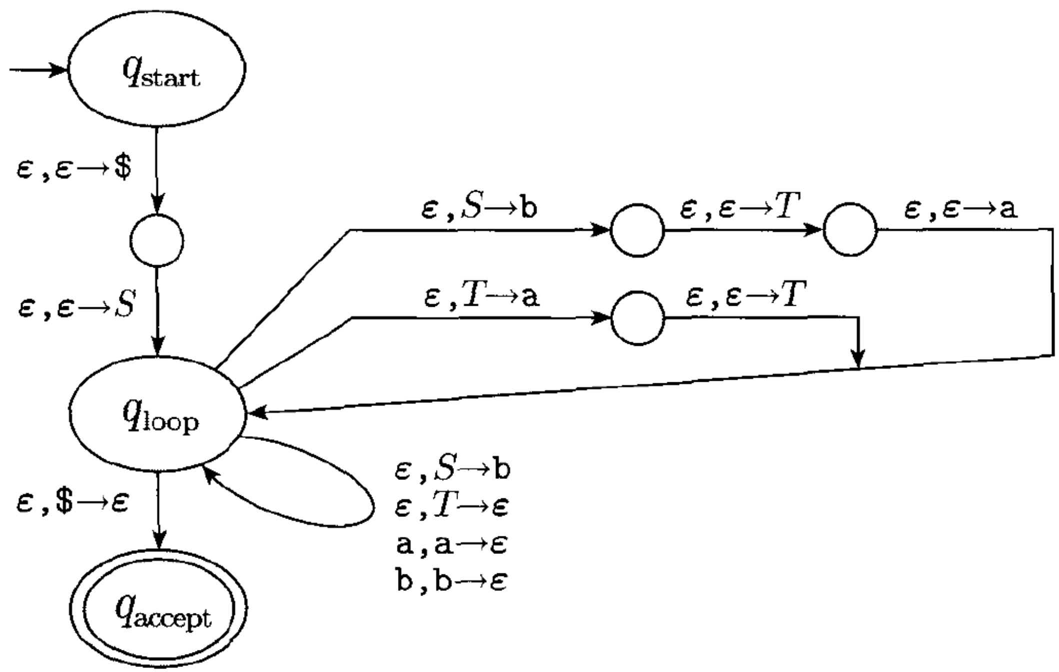
Equivalence of PDA and CFG (example)

<u>(⟹)</u> : For every CFG, we can build an equivalent PDA.

Example: (page 120 of text)

$$S \to \mathtt{a}T\mathtt{b} \mid \mathtt{b}$$
$$T \to T\mathtt{a} \mid \varepsilon$$

The transition function is shown in the following diagram.



PDA, CFG equivalence

Proof of ($\Longleftarrow$): If L is recognized by a PDA P implies that L is described by a CFG G.

Proof idea:

G should generate a string if the string causes P to go from its start sate to an accept state.

We design grammar that would do more:

For each pair of states p and q, the grammar will have a variable A_{pq}. A_{pq} generates all strings that can take P from p (w/ empty stack) to q (w/ empty stack).

PDA, CFG equivalence

- **main idea**: non-terminal $A_{p,q}$ generates exactly the strings that take the PDA from state p to state q, regardless of the stack contents at p, leaving the stack at the same condition as it was at p.

- then $A_{start,\ accept}$ generates all of the strings in the language recognized by the PDA.

PDA, CFG equivalence

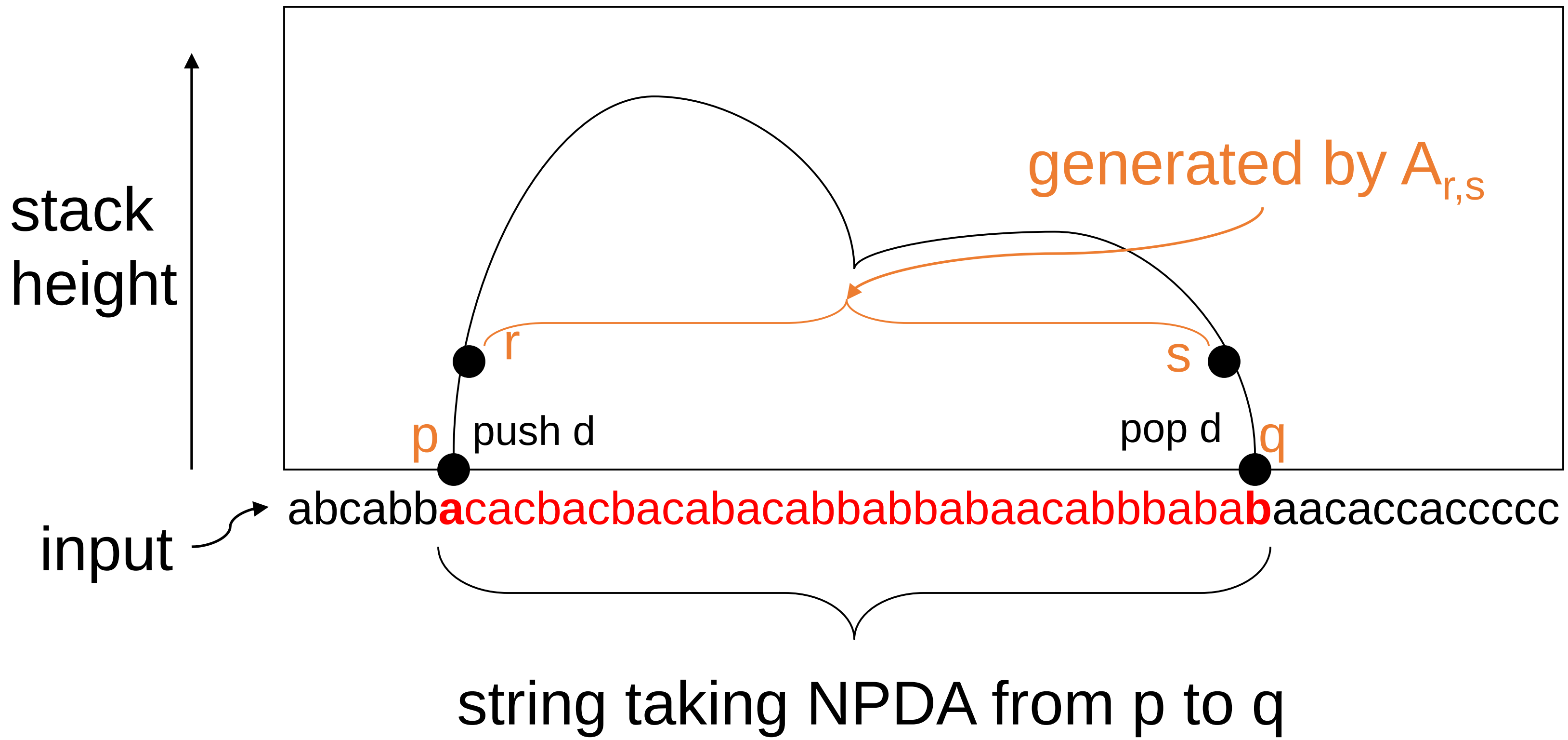
<u>**Proof of (⟸):**</u> If L is recognized by a PDA P implies that L is described by a CFG G.

First step: convert PDA into "normal form":

1. single accept state
2. empties stack before accepting
3. each transition *either* pushes *or* pops a symbol, but does not do both at the same time

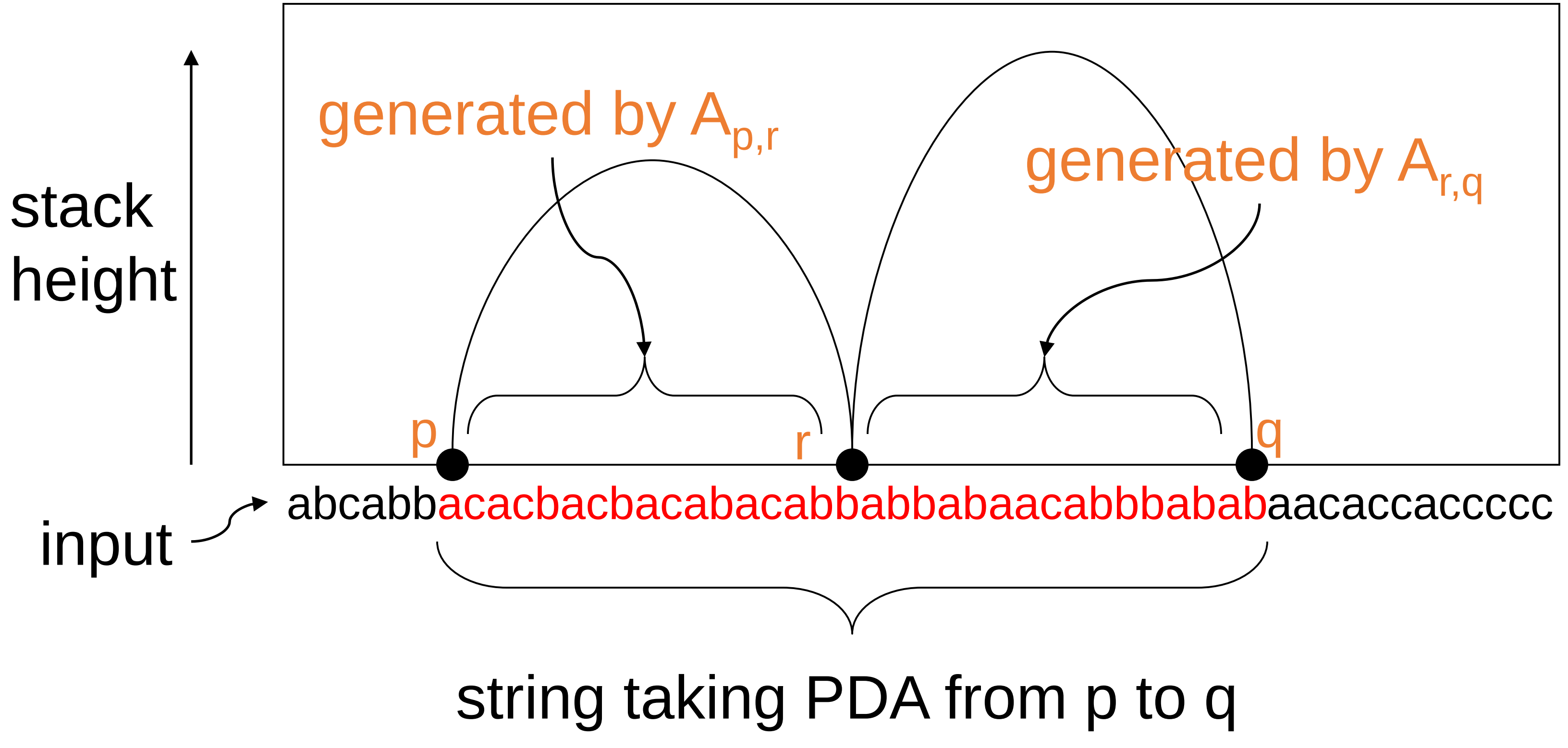
Two possibilities to get from state p to q:

Case 1: The symbol popped at the end is the same as pushed at the beginning. Then, stack is empty only at the beginning and at the end.



Two possibilities to get from state p to q:

Case 2: The initially pushed symbol is popped somewhere in the middle and stack becomes empty at that point :



NPDA, CFG equivalence

- PDA $P = (Q, \Sigma, \Gamma, \delta, start, \{accept\})$
- CFG G:
  - non-terminals $V = \{A_{p,q} : p, q \in Q\}$
  - start variable $A_{start,\ accept}$
  - productions:

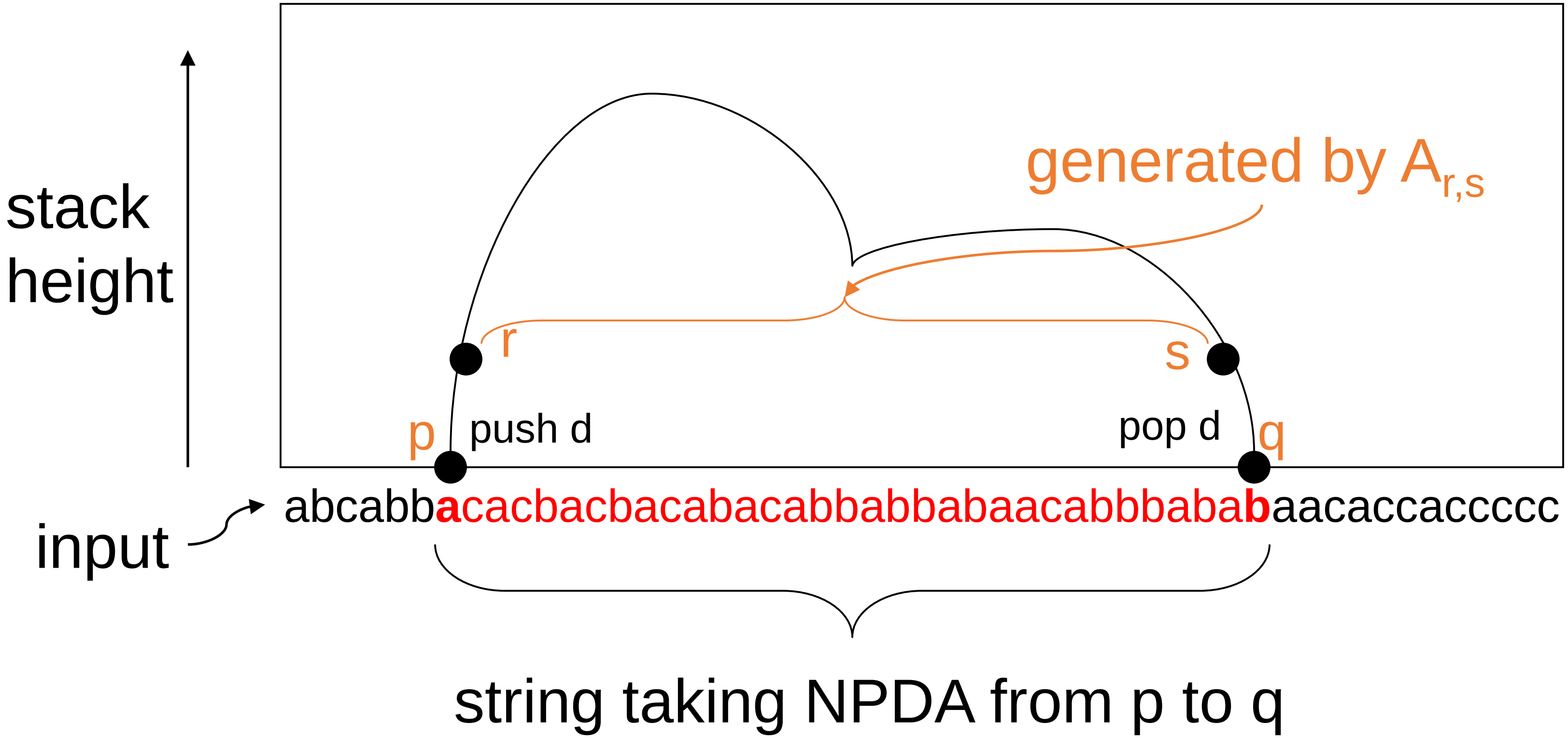
    for every $p, r, q \in Q$, add the rule

$$A_{p,q} \rightarrow A_{p,r}A_{r,q}$$

// simulates case 2

Two possibilities to get from state p to q:

Case 1: The symbol popped at the end is the same as pushed at the beginning. Then, stack is empty only at the beginning and at the end.



NPDA, CFG equivalence

- NPDA P = (Q, Σ, Γ, δ, start, {accept})
- CFG G:
  - non-terminals V = $\{A_{p,q} : p, q \in Q\}$
  - start variable $A_{start,\ accept}$
  - productions:

    for every p, r, s, q ∈ Q, d ∈ Γ, and a, b ∈ (Σ ∪ {ε})

    if (r, d) ∈ δ(p, a, ε), and

    (q, ε) ∈ δ(s, b, d), add the rule

from state p, read a, push d, move to state r

from state s, read b, pop d, move to state q

$$A_{p,q} \rightarrow aA_{r,s}b$$

// simulates case 1

NPDA, CFG equivalence

- NPDA $P = (Q, \Sigma, \Gamma, \delta, \text{start}, \{\text{accept}\})$
- CFG G:
  - non-terminals $V = \{A_{p,q} : p, q \in Q\}$
  - start variable $A_{\text{start, accept}}$
  - productions:

for every $p, r, s, q \in Q$, $d \in \Gamma$, and $a, b \in (\Sigma \cup \{\varepsilon\})$
if $(r, d) \in \delta(p, a, \varepsilon)$, and $(q, \varepsilon) \in \delta(s, b, d)$,
add the rule $A_{p,q} \rightarrow aA_{r,s}b$

for every $p, r, q \in Q$, add the rule $A_{p,q} \rightarrow A_{p,r}A_{r,q}$

for every $p \in Q$, add the rule $A_{p,p} \rightarrow \varepsilon$

NPDA, CFG equivalence

- two claims to verify correctness:

1. If $A_{p,q}$ generates string x, then x can take PDA P from state p (w/ empty stack) to q (w/ empty stack)
2. If x can take PDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x

NPDA, CFG equivalence

1. if $A_{p,q}$ generates string x, then x can take PDA P from state p (w/ empty stack) to q (w/ empty stack)
   - induction on length of derivation of x from $A_{p,q}$.
   - base case: 1 step derivation. must have only terminals on the right hand sise. In G, must be production of form $A_{p,p} \rightarrow \varepsilon$.

     Clearly, input ε takes P from p to p.

NPDA, CFG equivalence

1. if $A_{p,q}$ generates string x, then x can take PDA P from state p (w/ empty stack) to q (w/ empty stack)

Inductive step:

- assume true for derivations of length at most k, prove for length k+1.
- verify case: $A_{p,q} \rightarrow A_{p,r}A_{r,q} \quad \rightarrow^* \quad x = yz$
- verify case: $A_{p,q} \rightarrow aA_{r,s}b \quad \rightarrow^* \quad x = ayb$ (y is generated by $A_{r,s}$)

NPDA, CFG equivalence

2. If x can take PDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x
   - Induction on # of steps in P's computation
   - Base case: 0 steps. starts and ends at same state p. Only has time to read empty string ε.
   - G contains $A_{p,p} \to \varepsilon$.

NPDA, CFG equivalence

2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x

Induction step.

- Assume true for computations of length at most k, prove for length k+1.
- If stack becomes empty sometime in the middle of the computation (at state r)
  - y is read going from state p to r ($A_{p,r} \rightarrow^* y$)
  - z is read going from state r to q ($A_{r,q} \rightarrow^* z$)
  - conclude: $A_{p,q} \rightarrow A_{p,r}A_{r,q} \rightarrow^* yz = x$

NPDA, CFG equivalence

2. if x can take PDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x
   - if stack becomes empty only at beginning and end of computation.
     - first step: state p to r, read a, push d
     - go from state r to s, read string y ($A_{r,s} \rightarrow^* y$)
     - last step: state s to q, read b, pop d
     - conclude: $A_{p,q} \rightarrow aA_{r,s}b \rightarrow^* ayb = x$

PDA→CFG conversion

Summary of the construction:

Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$ and construct G. The variables of G are $\{A_{pq} \mid p, q \in Q\}$. The start variable is $A_{q_0, q_{\text{accept}}}$. Now we describe G's rules.

- For each $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ε), put the rule $A_{pq} \to aA_{rs}b$ in G.
- For each $p, q, r \in Q$, put the rule $A_{pq} \to A_{pr}A_{rq}$ in G.
- Finally, for each $p \in Q$, put the rule $A_{pp} \to \varepsilon$ in G.