CSCI 361 Lecture 8: Push-down Automata

Shikha Singh

Announcements & Logistics

- Hand in **Exercise # 7**, no exercise for next class 🎉
- **HW 4** out, due tomorrow
  - Short homework to allow time for midterm prep
- Practice midterm will be released soon
- Thursday lecture we will spend some time on review/practice questions
- **Reminder:** Midterm 1 in-class on Oct 7
  - Closed book but can ask clarification on definitions
  - Several textbooks will be available for referencing
  - Everything up to HW 4 included
- Today's office hours slightly shifted **2.30-3.55 pm**

Last Time

- Introduced CFGs as the next model of computation
  - Recursion provide more power and state
- Practiced CFGs
- Any regular language has a regular CFG that generates it and a regular CFG can be recognized by a DFA
- CFGs are closed under union, concatenation and Kleene star

Closure Properties of CFLs

- CFLs are closed under
  - Union
  - Concatenation
  - Kleene star
- **Important**. Not closer under complement and intersection!

Closure Properties of CFLs

Given $G_1 = (V_1, \Sigma_1, R_1, S_1)$

$G_2 = (V_2, \Sigma_2, R_2, S_2)$

Union: $L(G_1) \cup L(G_2)$ is generated by

$R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$

Concatenation: $L(G_1)L(G_2)$ is generated by

$R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}$

Kleene $*$: $L(G_1)^*$ is generated by

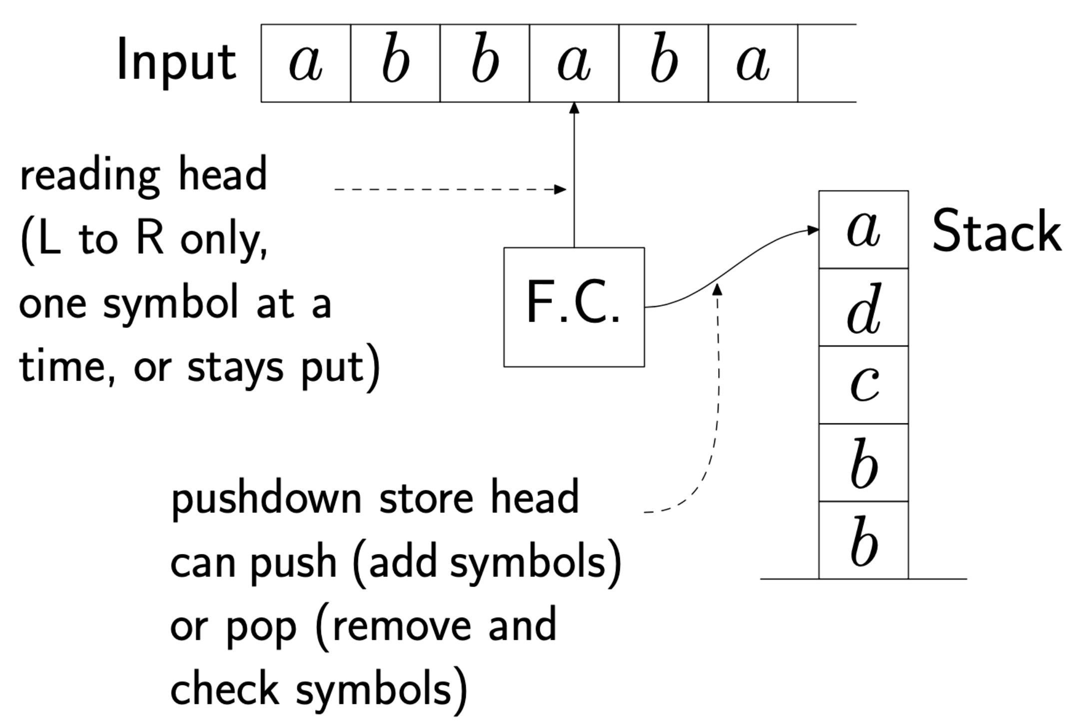
$R_1 \cup \{S \rightarrow e | S \rightarrow S_1 S\}$

Automata for CFGs

- Regular Languages : Finite Automata
- Context-free languages: ??

Pushdown Automata

- Basically an NFA with a stack (pushdown store)
- The stack can consist of unlimited number symbols but can only be read and altered at the top:
    - Can only pop symbol from top or push symbol to top



Pushdown Automata Transitions

- Transitions of a PDA have two parts:
  - **State transition** and **stack manipulation** (push/pop)
  - If in state p reading input symbol a and b on the stack, replace b with c on the stack and enter state q
    - $(p, a, b) \to (q, c)$
    - $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathscr{P}(Q \times \Gamma_\varepsilon)$
  - In state diagram arrow goes from $p \to q$ with label $a, b \to c$

Pushdown Automata Transitions

- If in state p reading input symbol a and b on the stack, replace b with c on the stack and enter state q, that is, $(p, a, b) \rightarrow (q, c)$
- In state diagram arrow goes from $p \rightarrow q$ with label $a, b \rightarrow c$
  - **(Non-determinism)** $\varepsilon, b \rightarrow c$ means without reading any input symbol, one branch jumps from p to q, popping b and pushing c
  - **(Push only)** $a, \varepsilon \rightarrow c$ means read a from the input, move from state p to q without popping anything from stack and pushing c on it
  - **(Pop only)** $a, b \rightarrow \varepsilon$ means read read a from the input, move from state p to q popping b off the stack, without pushing anything

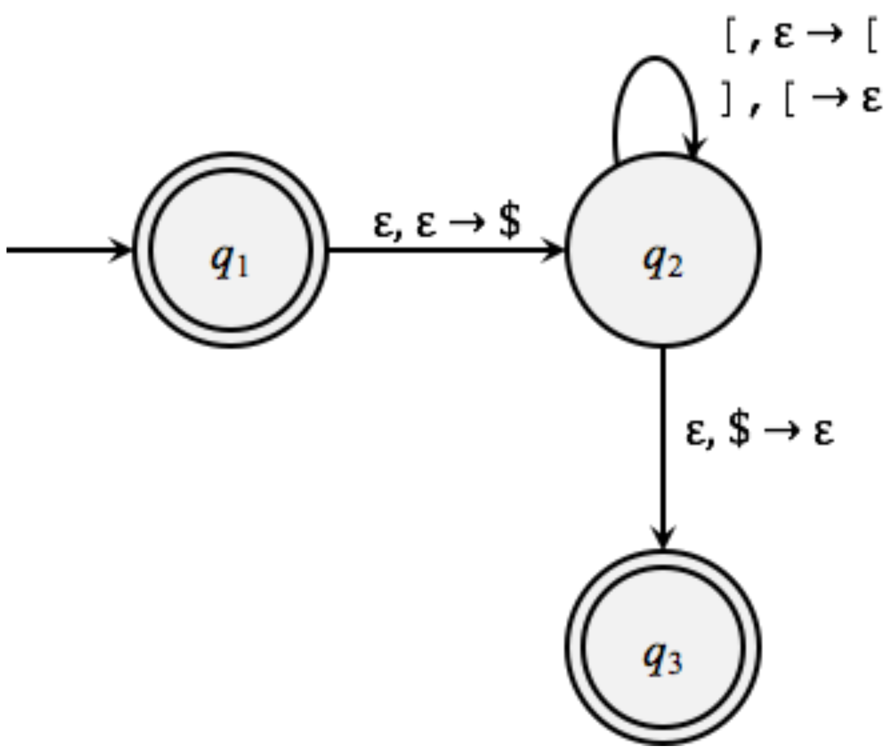
Formal Definition: PDA

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

Example PDA

- Consider the language over $\Sigma = \{[,]\}$ of all strings made up of correctly nested brackets
- CFG for this language: $S \rightarrow \varepsilon \mid [S] \mid SS$
- Now lets create a push-down automata for this language
- What to store on the stack?

Example PDA for Balanced Brackets



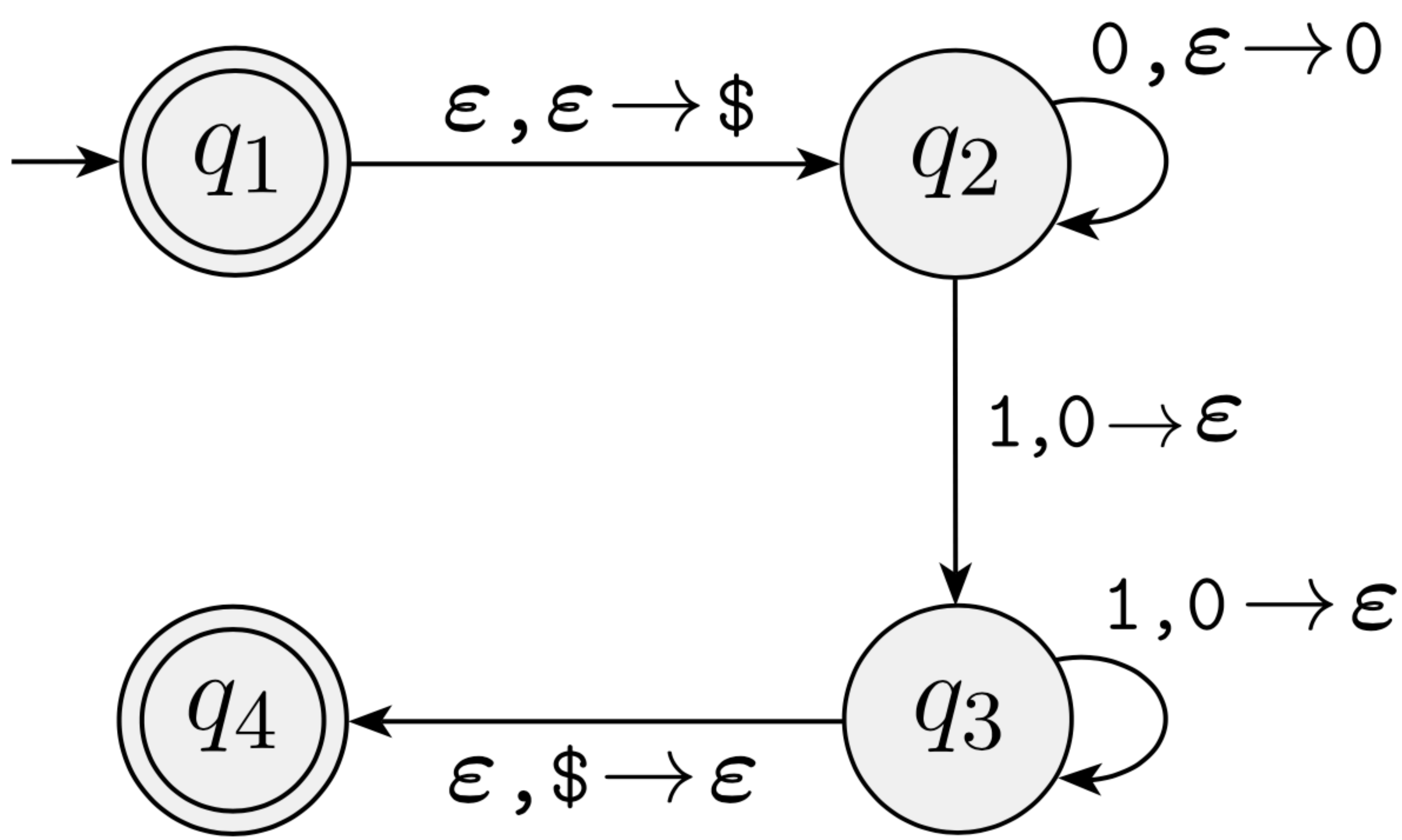
Recall: A transition of the form a, b → z means "if the current input symbol is a and the current stack symbol is b, then follow this transition, pop b, and push the string z"

PDA Acceptance: Informal

- A **PDA accepts an input string** ***w*** if there is a computation that:
  - starts in the start state and empty stack
  - has a sequence of valid transitions
  - at least one computation branch ends in an accept state with an empty stack
- A PDA computation branch "dies off" if
  - no transition matches the input (as in an NFA), or if
  - no rule matches the stack states
- **Language of a PDA**: set of all strings that are accepted by it

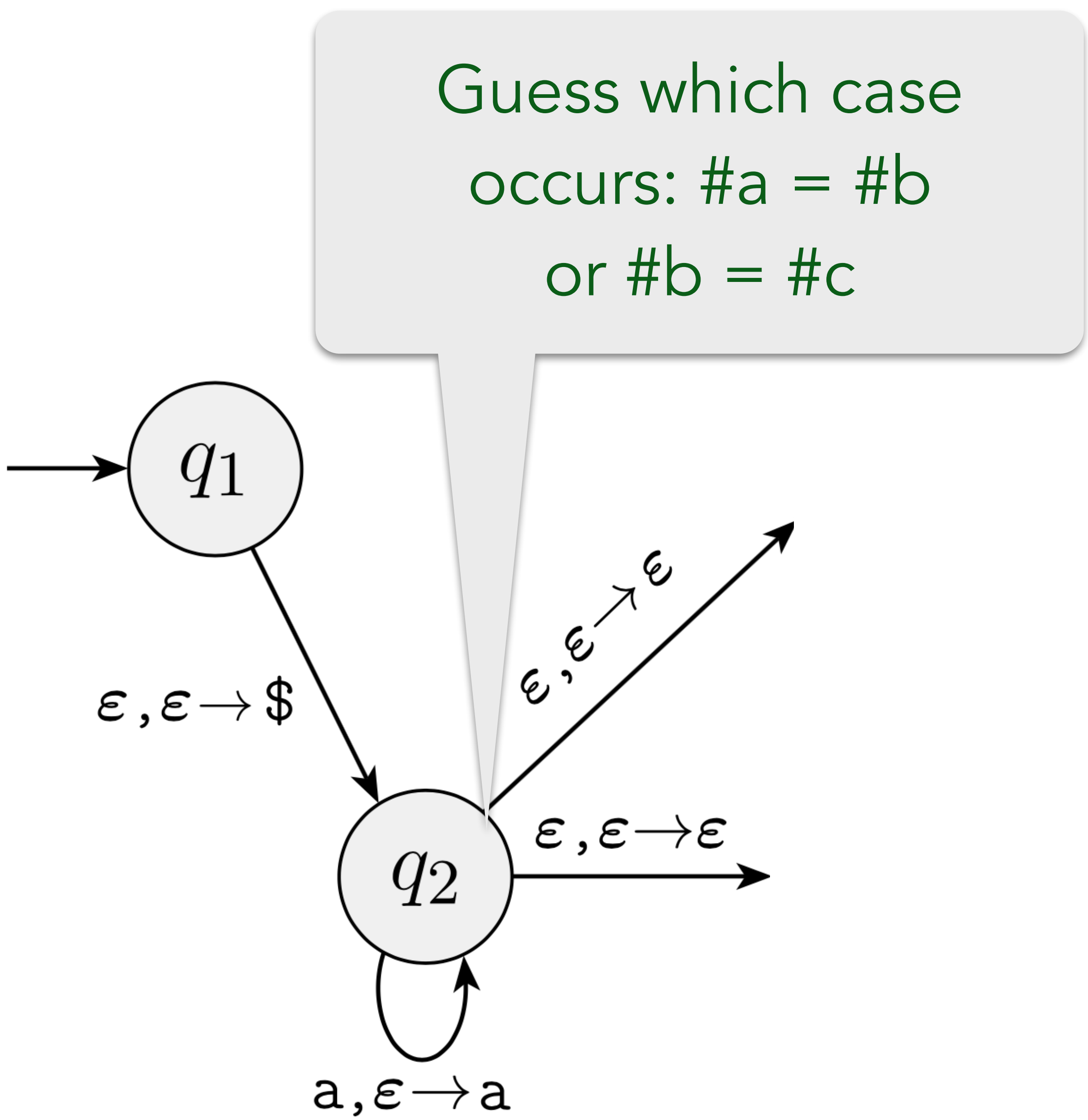
PDA More Examples

- $L = \{0^n 1^n \mid n \geq 0\}$



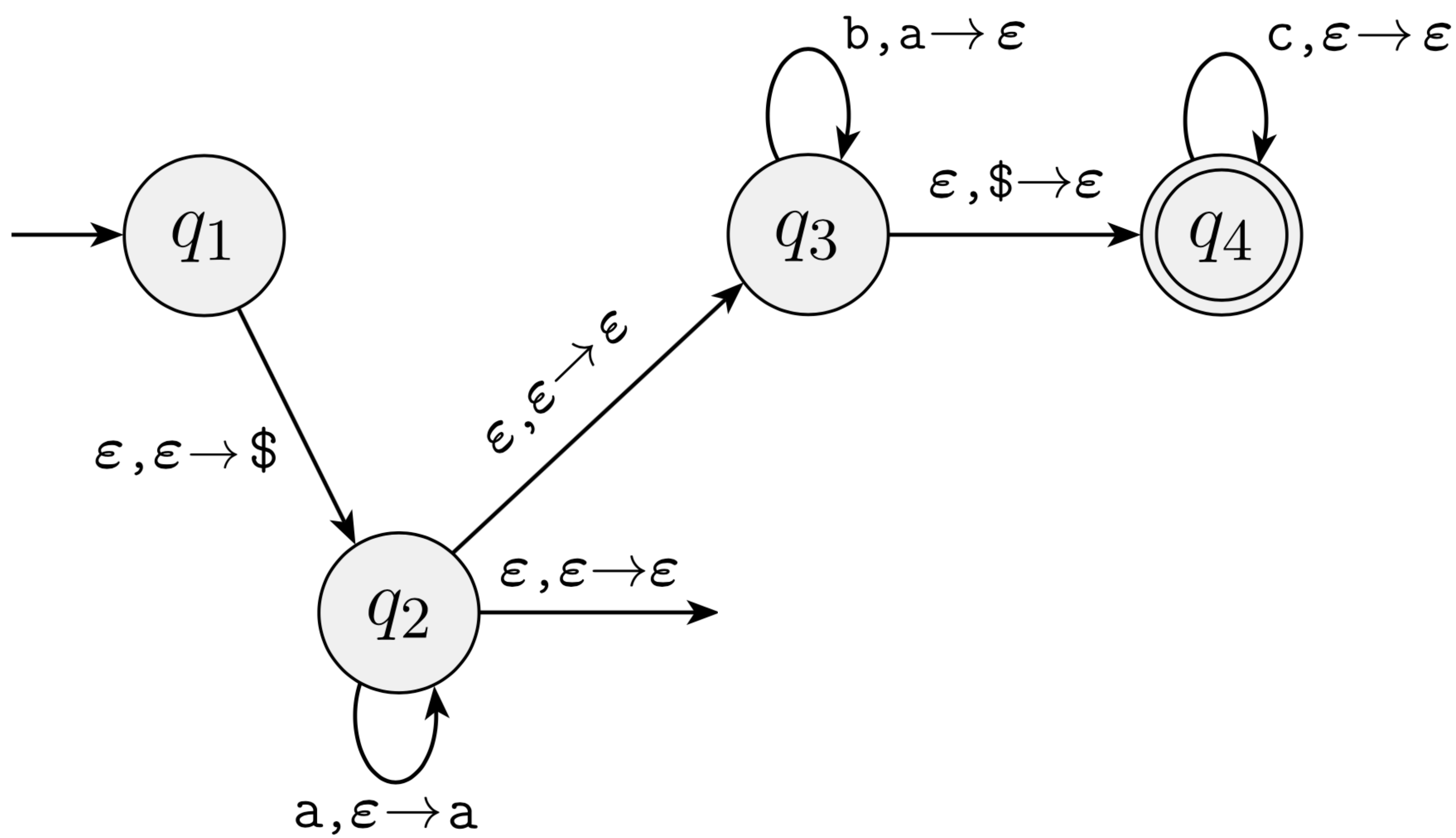
PDA More Examples

- PDA for $L = \{a^i b^j c^k \mid i = j \text{ or } i = k\}$



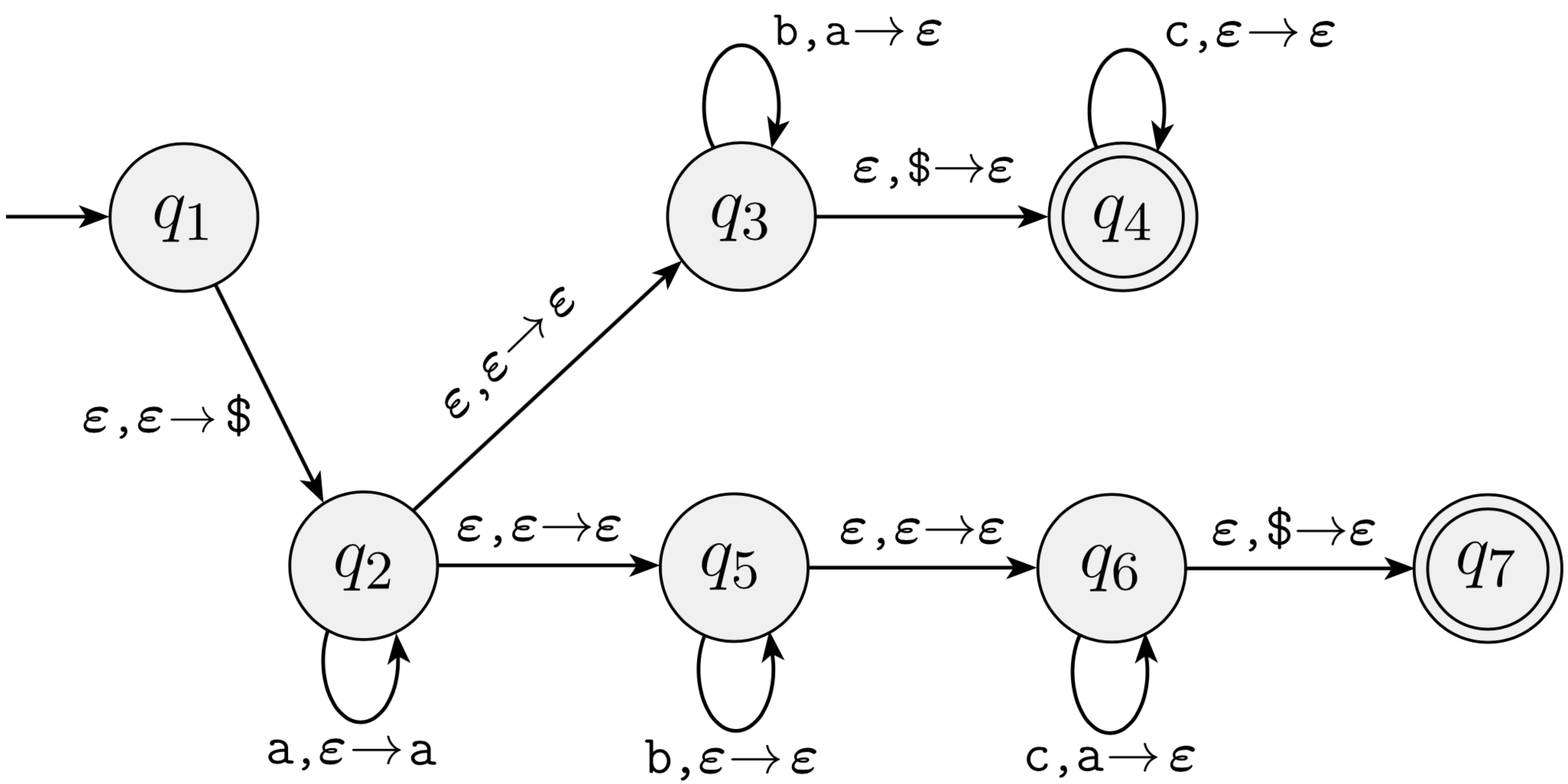
PDA More Examples

- PDA for $L = \{a^i b^j c^k \mid i = j \text{ or } i = k\}$



PDA More Examples

- PDA for $L = \{a^i b^j c^k \mid i = j \text{ or } i = k\}$



CFGs Not Closed under Intersection

- Consider $L_1 = \{a^i b^j c^k \mid i, j, k \geq 0$ and $i = j\}$ and $L_2 = \{a^i b^j c^k \mid i, j, k \geq 0$ and $i = k\}$
- Both are context-free languages
- However, their intersection $L_1 \cap L_2 = \{a^i b^i c^i \mid i \geq 0\}$ is not a CFL
  - We will prove this by pumping lemma soon
  - Intuition: Only one stack: can either match a's and b's or a's and c's but not both (once something is popped, gone forever)

Practice Problems

- Draw a PDA for the following languages:
  - $L = \{a^i b^j c^k \mid i, j, k \geq 0$ and $i + k = j\}$
    - Can you also give a CFG generating such strings?
  - $L = \{ww^R \mid w \in \{0,1\}^*\}$

Few Things to Note

- PDAs can be a little tricky to draw
  - Need to worry about non-determinism + stack at the same time
  - Don't confuse the ε which is a NFA "guess" from the ε in stack transition which indicates push only/pop only
  - Remember that whenever either the input symbol or top of stack doesn't match an available rule, that branch dies off
- Sometimes you may want to push more than one symbol at once
  - Abuse notation to write $a, \$ \rightarrow \$b$ (pop $\$$ then push $\$$ back followed by push b

Equivalence: CFG $\Longleftrightarrow$ PDA

Theorem. A language is context-free if and only it is recognized by some (non-deterministic) pushdown automaton.

Won't prove this formally but will discuss high-level intuition towards the end of lecture

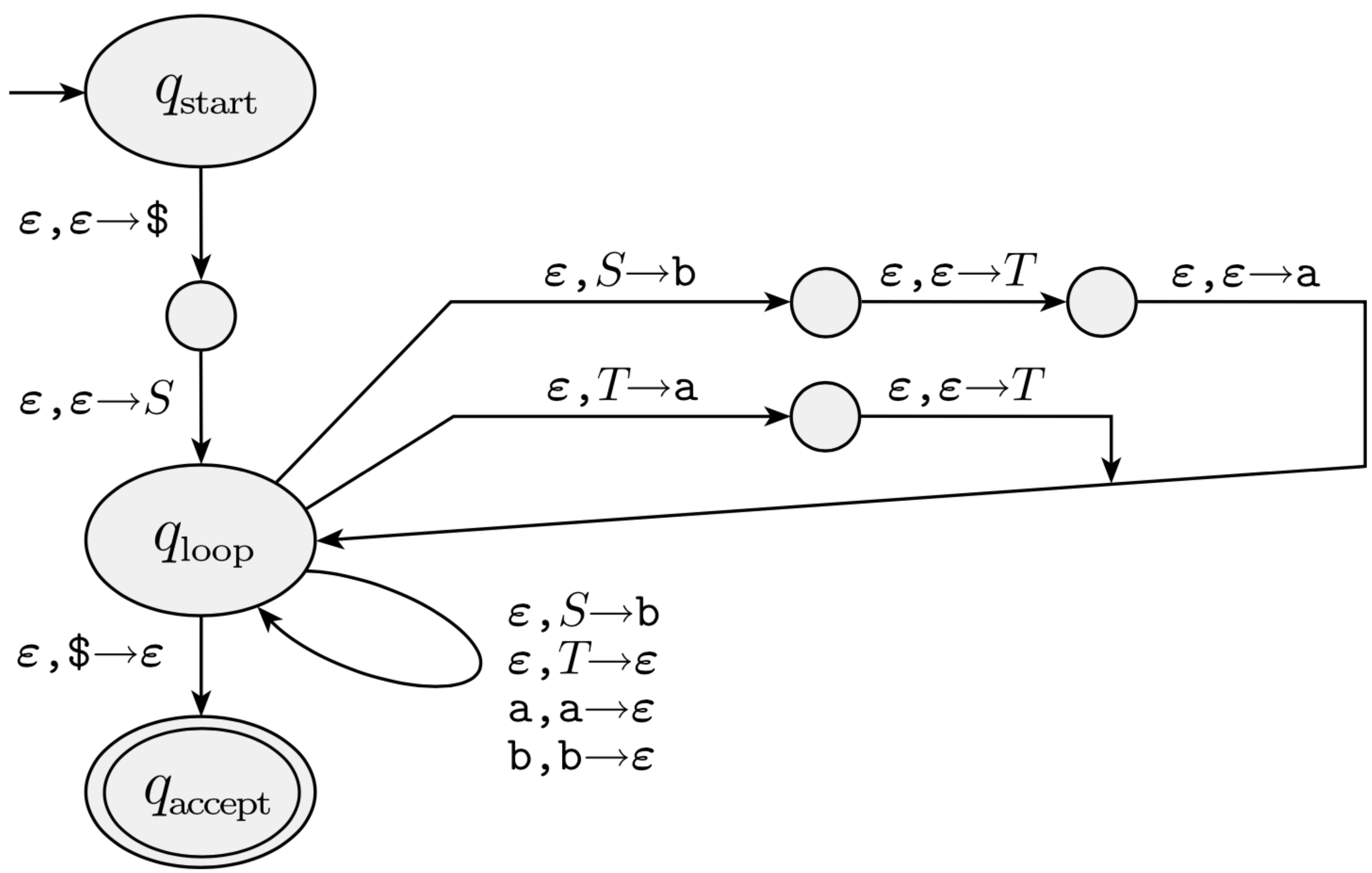
Note: Unlike DFA and NFA, non-deterministic PDAs are more powerful than deterministic PDAs.

Intuition: CFG $\Longrightarrow$ PDA

- Consider a CFG $G = (V, \Sigma, R, S)$
- Construct a PDA with three main states: **start**, **loop** and **accept** state (some extra states for bookkeeping)
  - Start by putting S on the stack
  - Each time top of stack is a variable A, guess a rule of the type $A \to u$ replace A with RHS of the rule
  - Each time top of stack is a terminal match it to the current input symbol (computation dies off it they don't match)
  - If you reach bottom of stack at any point in a branch, accept
    - All variables have been replaced and non-terminals matched

Example: CFG $\Longrightarrow$ PDA

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



Intuition: PDA $\Longrightarrow$ CFG

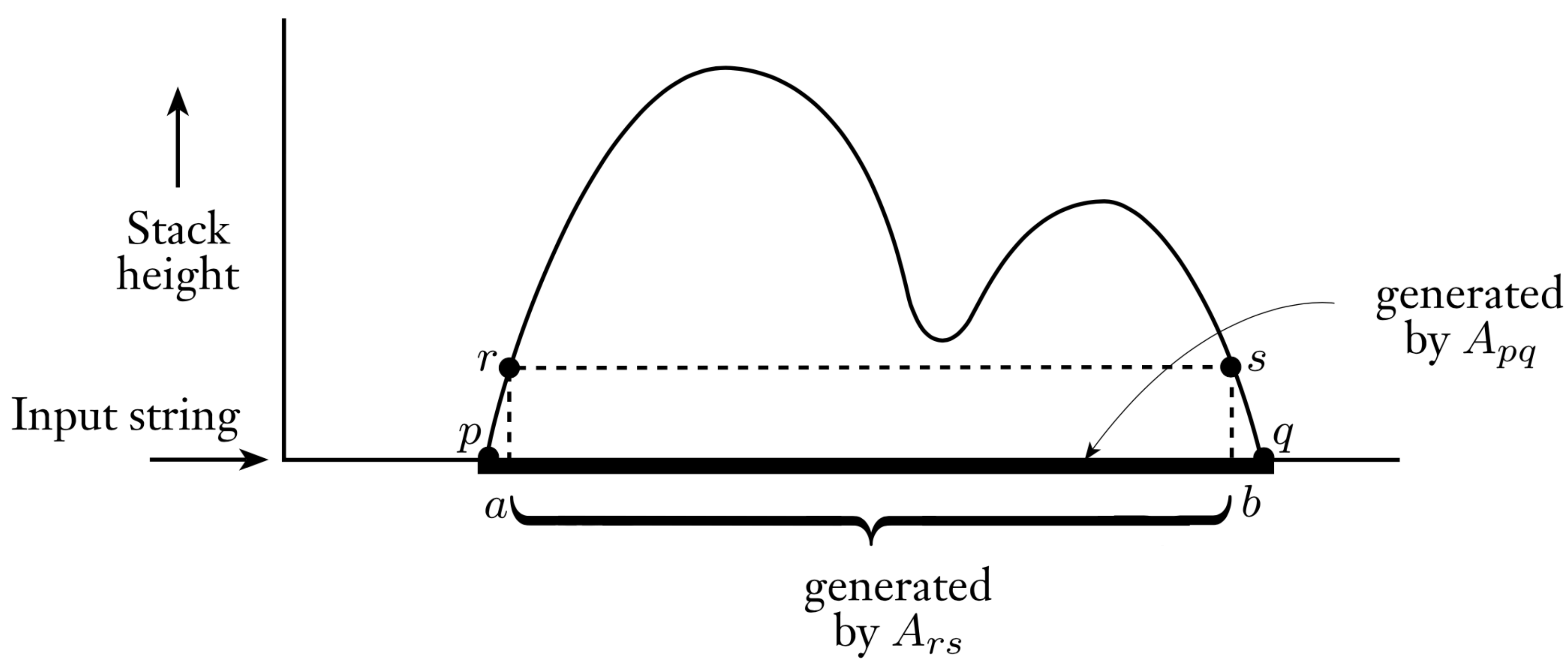
- Wlog assume the PDA has one accept state, empties stack before accepting and each move is a push or pop (but not both)
- Let Q be the states of the PDA
- Create variables for each pair of states: $\{A_{pq} \mid p, q \in Q\}$
- A_{pq} generates all strings that take the PDA from p to q starting from an empty stack and ending at an empty stack
  - Such strings can also take PDA from p to q from a non-empty stack returning to exactly the same stack contents
- **Start variable** is A_{q_0,q_f} where q_0 is start state and q_f is accept state

Intuition: PDA $\Longrightarrow$ CFG

- Consider the computation of the PDA on the input string that takes it from a state p (and empty stack) to a state q (and empty stack)
- Two possibilities:
  - Stack is only empty at the beginning and end: first symbol pushed first is the last symbol to be popped
  - Stack is empty in the middle of the computation (the first symbol pushed is popped off at some point)

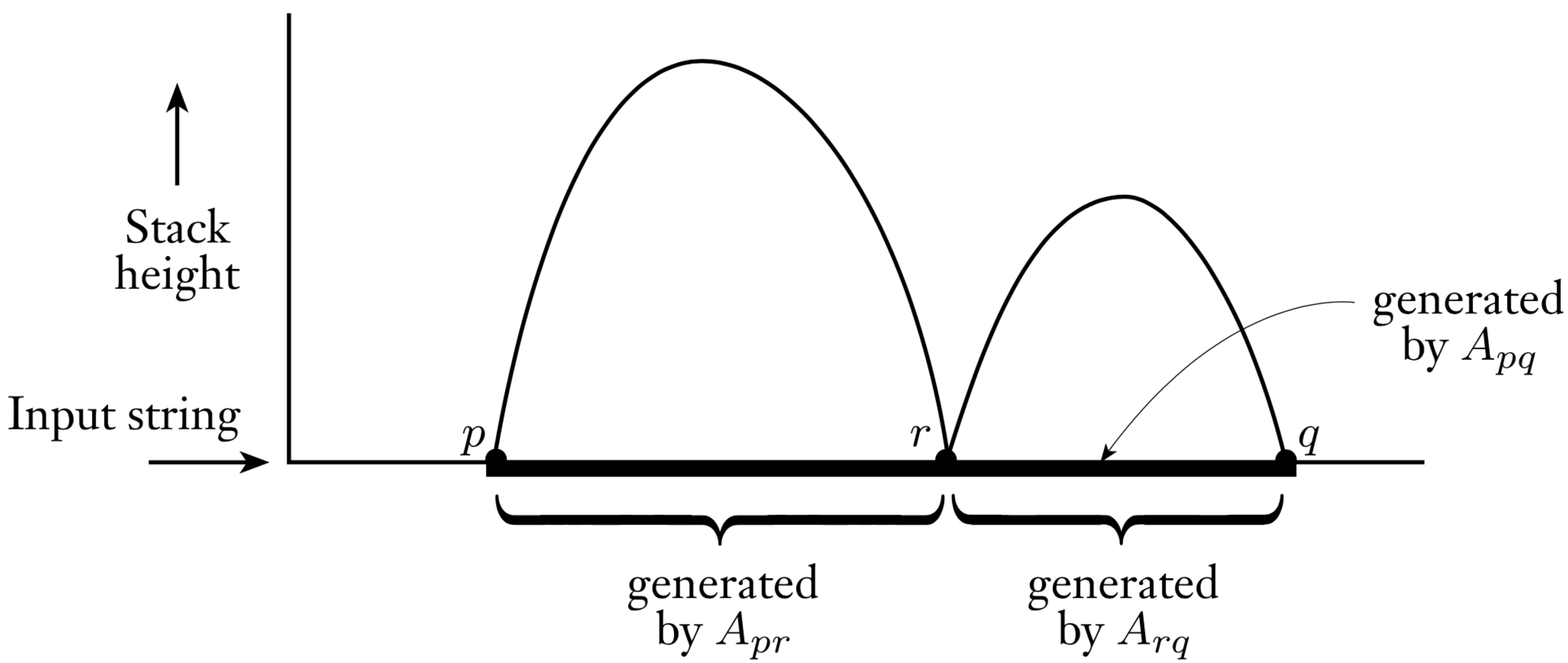
CFG Rule for Possibility 1

- Stack is only empty at the beginning and end: first symbol pushed first is the last symbol to be popped
- That is, $(p, a, \epsilon) \rightarrow (r, u)$ and $(b, s, u) \rightarrow (q, \epsilon)$ where PDA goes from p to q after pushing a and s to r after popping b
- Then, add the rule $A_{pq} \rightarrow aA_{rs}b$



CFG Rule for Possibility 2

- Stack is empty in the middle of the computation (the first symbol pushed is popped off at some point)
- Add the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ for every triple $p, q, r \in Q$



Base Case

- Finally, for each $p \in Q$, add the rule $A_{pp} \to \varepsilon$

All At Once

- Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$, construct CFG with variables $\{A_{pq} \mid p, q \in Q\}$ and start variable $A_{q_0 q_{\text{accept}}}$ and rules:

1. For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε), put the rule $A_{pq} \to aA_{rs}b$ in G.

2. For each $p, q, r \in Q$, put the rule $A_{pq} \to A_{pr}A_{rq}$ in G.

3. Finally, for each $p \in Q$, put the rule $A_{pp} \to \varepsilon$ in G.

Intuition: Why it Works?

- The proof of correctness relies on the following claim:
  - A_{pq} generates x if and only if string x can bring P from p with empty stack to q with empty stack
- Both directions are induction:
  - $(\Rightarrow)$ Induction on the derivation length
  - $(\Leftarrow)$ Induction on the computation length

Non-Context-Free Languages

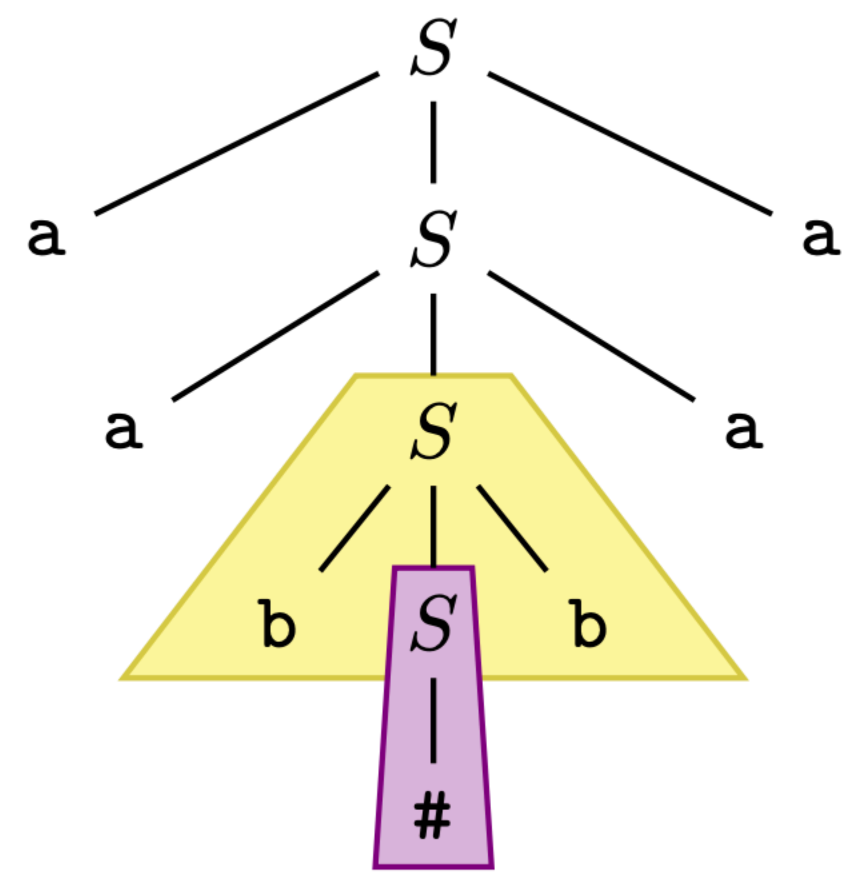
- Proved using a similar "pumping lemma" as regular languages
- With respect to regular languages:
    - pumping lemma exploits the fact that if a string is long enough, a state is repeated in the DFA for the language (loop)
- With respect to CFLs:
    - pumping lemma exploits the fact that if a string is long enough, deriving it requires recursion (repeated use of a variable)
- Lemma based length of parse trees for derivations

Parse Trees and CFGs

- Consider the CFG for $A = \{w\#w^R \mid w \in \{a, b\}^*\}$:

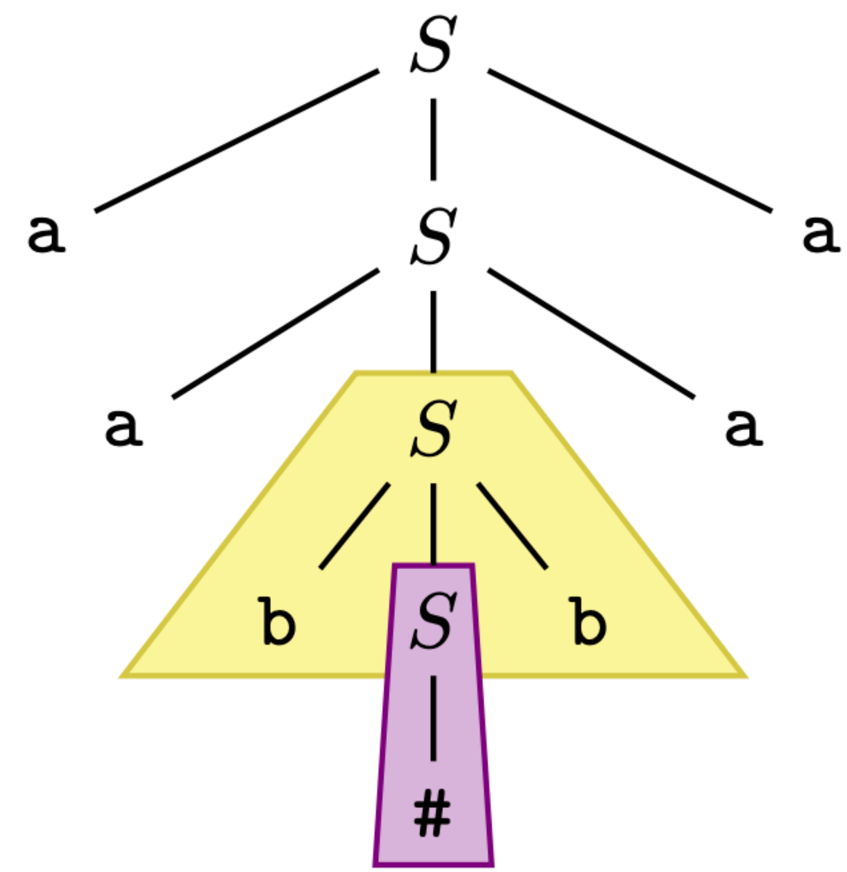
  $S \to aSa \mid bSb \mid \#$

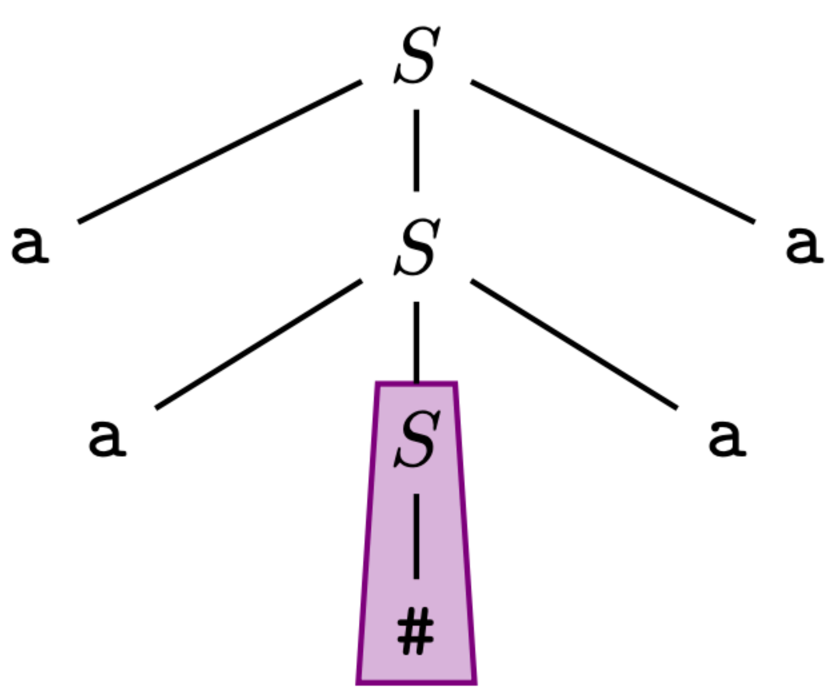
- Consider a parse tree for $w =$ aab#baa

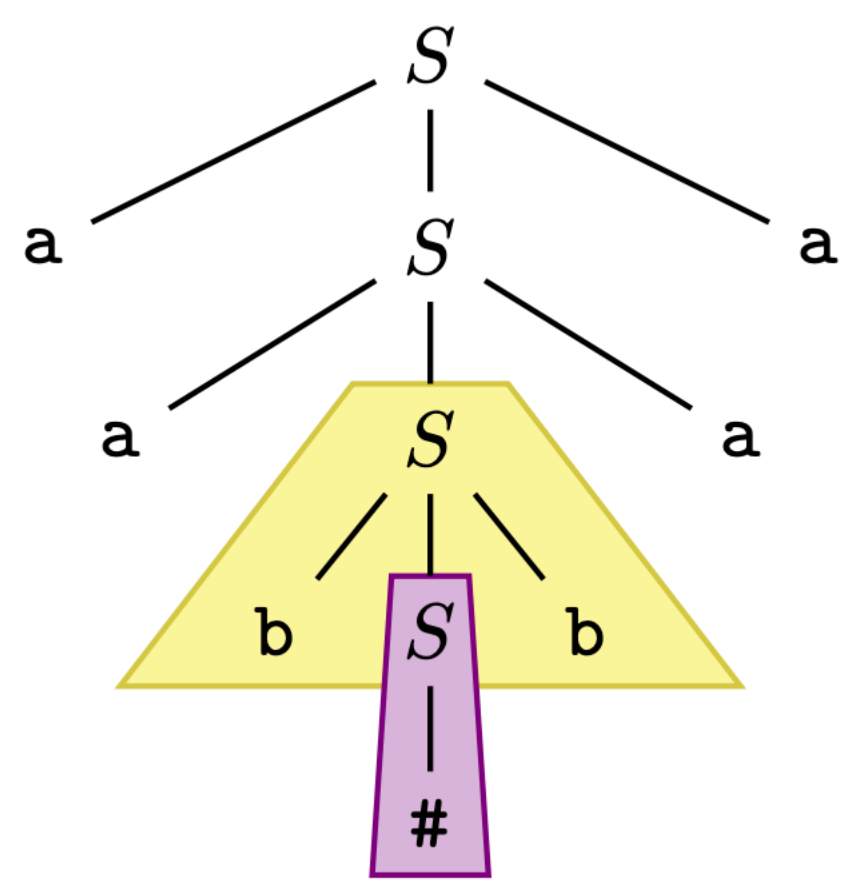


Parse Trees and CFGs

- Variable S is repeated
- Can "pump up" or "pump down" to create strings in the language
  - Replace yellow with violet: aa#aa
  - Replace violet with yellow: aabb#bba







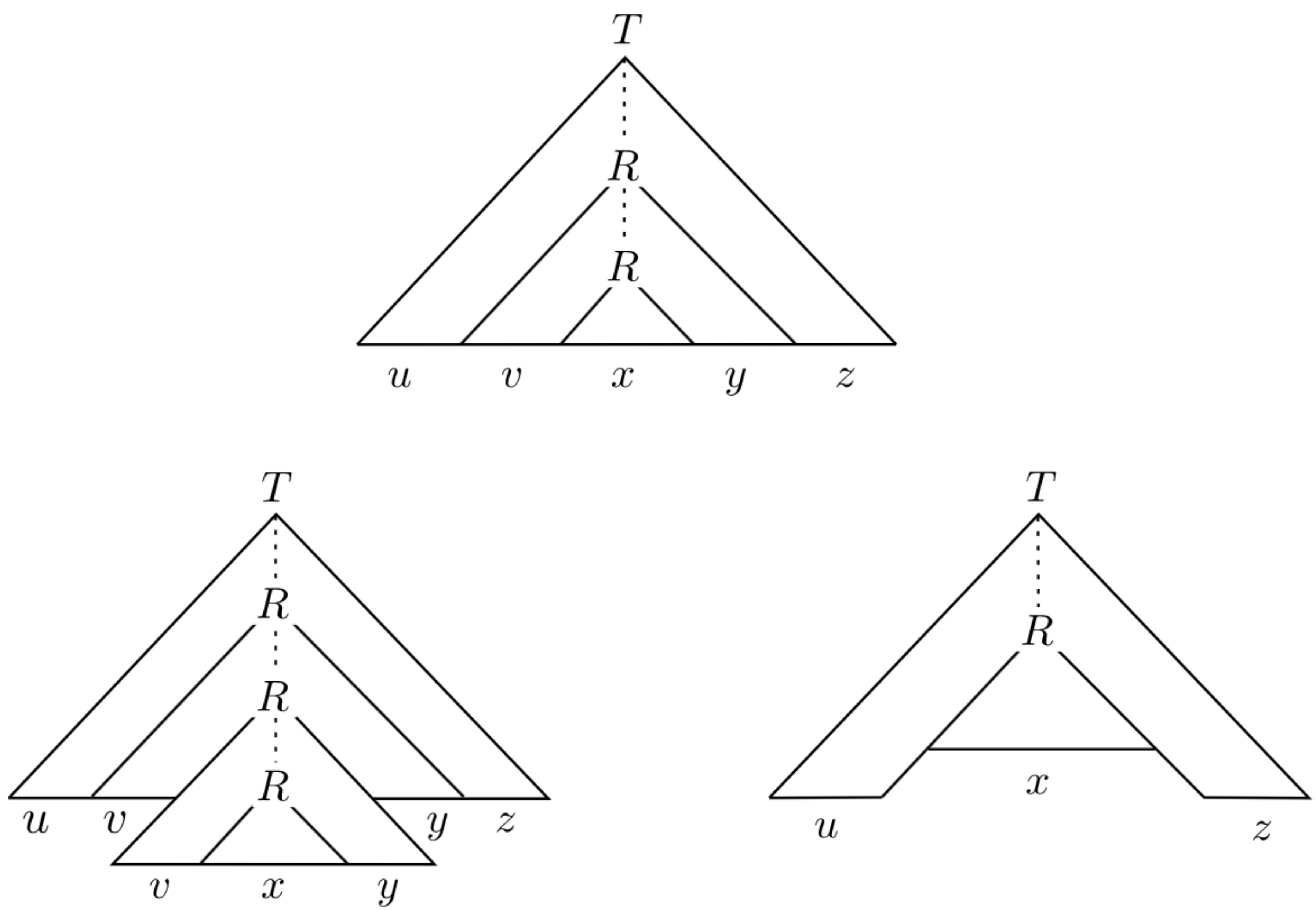
Pumping Lemma: CFLs

- **Statement**: If L is a CFL, then there is a number p (the pumping length) where for any $s \in L$ of length at least p, it is possible to divide s into five pieces $s = uvxyz$ satisfying the conditions

    1. $|vy| > 0$
    2. $|vxy| \leq p$
    3. For each $i \geq 0$, $uv^ixy^iz \in L$

- Note that vxy can appear anywhere in the string as long as they are no longer than p symbols long

Non-Context-Free Languages



Pumping Lemma (CFL): Intuition

- If the grammar generates a long enough string then the parse tree for that derivation must be "tall enough"
- If each node in a tree has at most b children and the tree has height h, what is the maximum number of leaves it can have?
    - b^h
- If a tree has at least b^{h+1} leaves and each node has degree at most b, what can we say about the height?
    - At least $h + 1$