CS372
FORMAL LANGUAGES & THE THEORY OF COMPUTATION

Dr. Nguyen Thi Thu Huong,PhD.
Phone: +84 24 38696121, Mobi: +84 903253796
Email: huongnt@soict.hust.edu.vn,
huong.nguyenthithu@hust.edu.vn

Unit 5

Push Down Automata
Context Free Pumping Lemma

Push Down Automata (PDA)

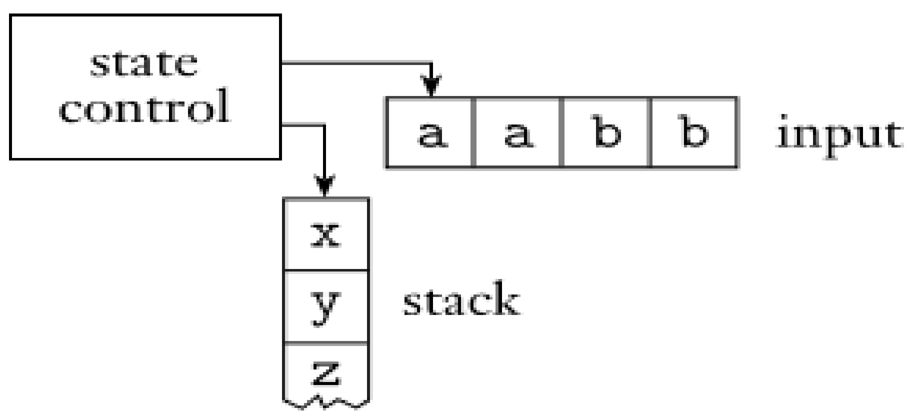
- Informal description
- Formal Definition
- Equivalence between PDA and CFG

Informal Description

- Push Down Automata (PDAs) are ε-NFAs with stack memory.
- Transitions are decided by an input symbol together with a pair of the form X/α
- The transition is possible only if the top of the stack contains the symbol X
- After the transition, the stack is changed by replacing the top symbol X with the string of symbols α. (Pop X, then push symbols of α)

Components of a PDA

- A control with a finite number of states
- A tape contains the input string
- The input head
- A stack



Formal Definition of a Nondeterministic PDA

- A *nondeterministic pushdown automaton* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where
  1. Q is the set of states,
  2. Σ is the input alphabet,
  3. Γ is the stack alphabet,
  4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P\,(Q \times \Gamma_\varepsilon)$ is the transition function,
  5. $q_0 \in Q$ is the start state,
  6. $F \subseteq Q$ is the set of accept states.

Example

- Here is the formal description of the PDA that recognizes the language $\{0^n1^n \mid n \geq 1\}$. Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where
  $Q = \{q_1, q_2, q_3, q_4\}$,
  $\Sigma = \{0,1\}$,
  $\Gamma = \{0, \$\}$,
  $F = \{q_4\}$, and δ is given table on the next slide, wherein blank entries signify $\emptyset$.

Transition function of M_1

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

$\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, \$)\}$

$\delta(q_2, 0, \varepsilon) = \{(q_2, 0)\}$

$\delta(q_2, 1, 0) = \{(q_3, \varepsilon)\}$

$\delta(q_3, 1, 0) = \{(q_3, \varepsilon)\}$

$\delta(q_3, \varepsilon, \$) = \{(q_4, \varepsilon)\}$

Details of transition function

$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$

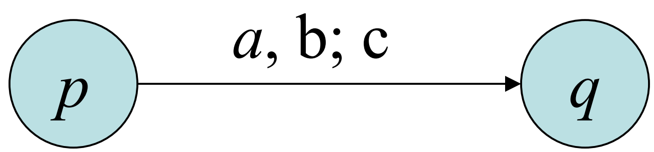
- $(q, \gamma) \in \delta(p, a, \varepsilon)$: make this transition without reading and popping any symbol from the stack.
- $(q, \gamma) \in \delta(p, \varepsilon, b)$: make this transition without reading any symbol from the input
- $(q, \varepsilon) \in \delta(p, a, b)$: does not write any symbol on the stack when going along this transition

Transitions

- Let ($(q, \gamma) \in \delta(p, a, \beta)$)be a transition.
- It means that we
  - Start in state p.
  - Read a from the tape,
  - Pop the string β from the stack,
  - Move to state q,
  - Push string γ onto the stack.
- The first three (p, a, β), are “input.”
- The last two (q, γ) are “output.”

Transitions

- We will draw it as



Pushing and Popping

- When we push string β, we push the symbols of β as we read them *right to left*.
- When we pop γ, we pop the symbols of γ as we read them from *left to right* (reverse order).
- For example,
  - When we push the string *abc*, we push *c*, then push *b*, then push *a*, i.e., *a* is on top.
  - When we pop the string *abc*, we pop *a*, then pop *b*, then pop *c*.
  - Thus, if we push the string *abc* and then pop it, we will get back *abc*, not *cba*.

Configurations

- A *configuration* fully describes the current "state" of the PDA.
  - The current state.
  - The remaining input.
  - The current stack contents.
- Thus, a configuration is a triple

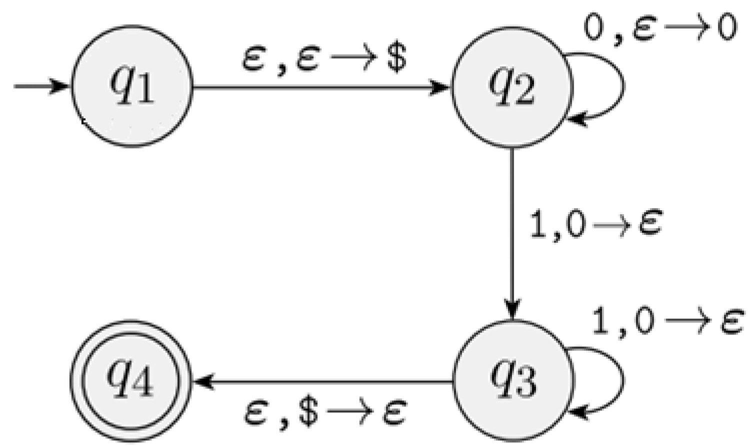
$$(p, w, \alpha) \in (Q, \Sigma^*, \Gamma^*).$$

State diagram of PDA

- State diagrams of PDAs are similar to the state diagrams of finite automata, with an addition of activities in the stack
- Write "a,b → c" to signify that when the machine is reading an a from the input, it may replace the symbol b on the top of the stack with a c.
- Any of a, b, and c may be ε.
- Example: State diagram of PDA

  P that recognize language

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



Computations

- A configuration (p, w, α) *yields* a configuration (p', w', α') *in one step*, denoted

$$(p, w, \alpha) \Rightarrow (p', w', \alpha'),$$

  if there is a transition $((p, a, \beta), (p', \gamma)) \in \delta$ such that $w = aw'$, $\alpha = \beta\eta$, and $\alpha' = \gamma\eta$ for some $\eta \in \Gamma^*$.
- The reflexive, transitive closure of $\Rightarrow$ is denoted $\Rightarrow^*$.

Accepting Strings

- After processing the string on the tape,
  - The PDA is in either a final or a nonfinal state, and
  - The stack is either empty or not empty.
- The input string is *accepted* if
  - The ending state is a final state, and
  - The stack is empty.
- That is, the string $w \in \Sigma^*$ is accepted if

$$(q_0, w, \varepsilon) \Rightarrow^* (p, \varepsilon, \varepsilon)$$

for some $p \in F$.

Example of PDA P's computation

$(q_1, 000111, \varepsilon) \Rightarrow (q_2, 000111, \$)$

$\Rightarrow (q_2, 00111, 0\$)$

$\Rightarrow (q_2, 0111, 00\$)$

$\Rightarrow (q_2, 111, 000\$)$

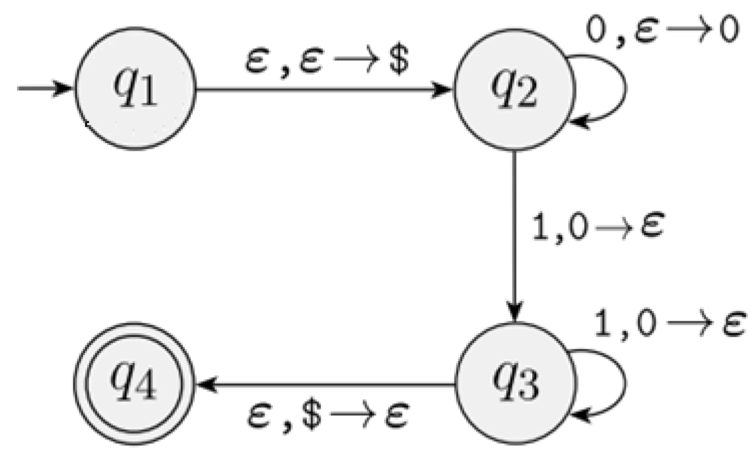
$\Rightarrow (q_3, 11, 00\$)$

$\Rightarrow (q_3, 1, 0\$)$

$\Rightarrow (q_3, \varepsilon, \$)$

$\Rightarrow (q_4, \varepsilon, \varepsilon)$

000111 is accepted by P



The Language of a PDA

- The *language* of a PDA *A* is

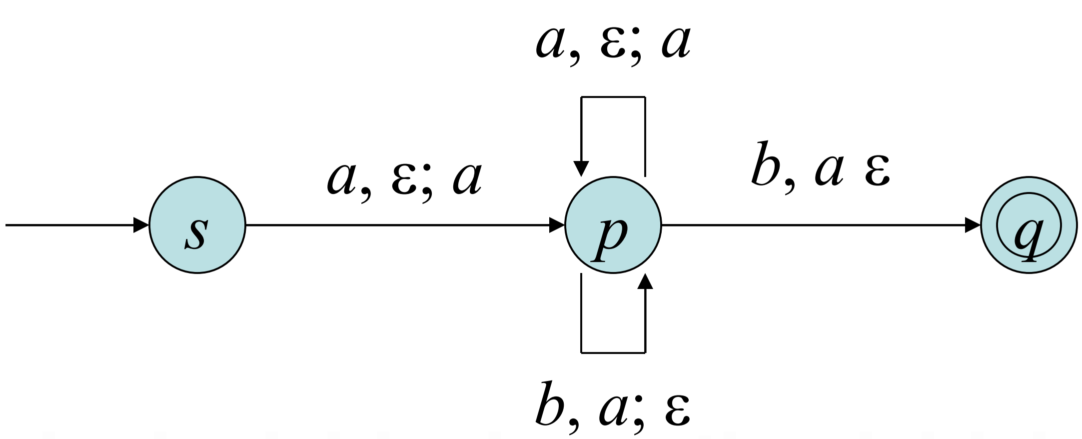
$$L(A) = \{w \in \Sigma^* \mid A \text{ accepts } w\}.$$

Or

$$\mathrm{L(A)} = \{w \in \Sigma^* \mid (q_0, w, \varepsilon,) \Rightarrow^* (p, \varepsilon, \varepsilon), \mathrm{p} \in \mathrm{F}\}$$

Example of a PDA

- Run the following PDA on the input string *aaabbb*.



Example of a PDA

- The steps in the processing are
  - $(s, aaabbb, \varepsilon,) \Rightarrow (p, aabbb, a)$

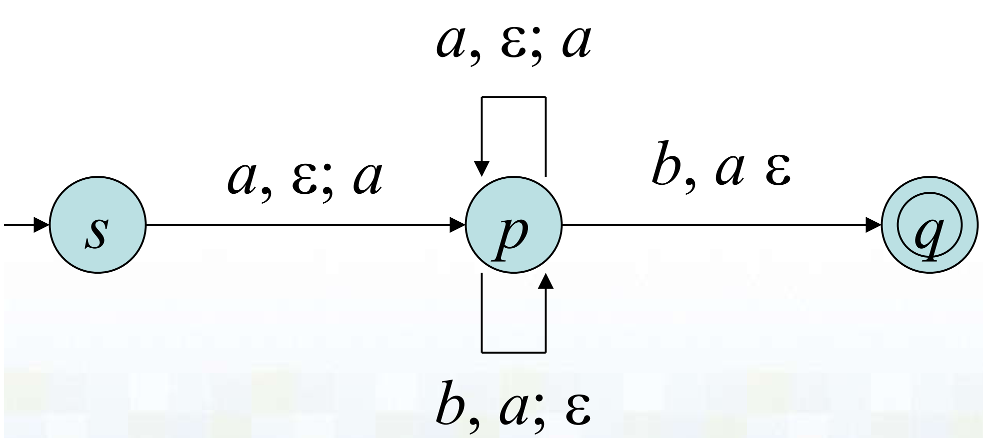
$\Rightarrow (p, abbb, aa)$

$\Rightarrow (p, bbb, aaa)$

$\Rightarrow (p, bb, aa)$

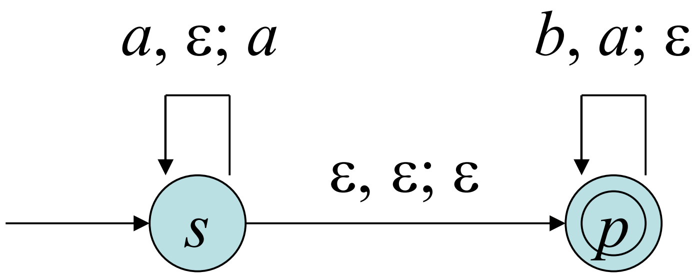
$\Rightarrow (p, b, \varepsilon,)$

$\Rightarrow (q, \varepsilon, \varepsilon,).$



Example of a PDA

- What is the language of the following PDA?



$$
\begin{aligned}
&(s,\ aaabbb,\ \varepsilon,) \\
\Rightarrow\ &(p,\ aabbb,\ a) \\
\Rightarrow\ &(p,\ abbb,\ aa) \\
\Rightarrow\ &(p,\ bbb,\ aaa) \\
\Rightarrow\ &(p,\ bb,\ aa) \\
\Rightarrow\ &(p,\ b,\ \varepsilon,) \\
\Rightarrow\ &(q,\ \varepsilon,\ \varepsilon,).
\end{aligned}
$$

CFG $\rightarrow$ PDA

Lemma

If a language is context free, then some pushdown automaton recognizes it.

Proof Idea:

- Let L be a CFL. From the definition we know that L has a CFG, G, generating it.
- We show how to convert G into an equivalent PDA, which we call P.

CFG $\rightarrow$ PDA

- The PDA P will accept string w, if G generates that input, by determining whether there is a derivation for w
- P begins by writing the start variable on its stack.
- It goes through a series of intermediate strings, making one substitution after another.
- Eventually it may arrive at a string that contains only terminal symbols, meaning that it has used the grammar to derive a string

Informal description of P

1. Place the marker symbol $ and the start variable on the stack.
2. Repeat the following steps forever.
   a. If the top of stack is a variable symbol A, select one of the rules for A and substitute A by the string on the RHS of the rule.

   b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a.
      - If they match, repeat.
      - If they do not match, reject on this branch of the nondeterminism

c. If the top of stack is the symbol $, enter the accept state. Doing so accepts the input if it has all been read.

Implement of PDA P

Assume L = L(G) , G = (Σ, N, R, S), we construct PDA

P = (Q, Σ, Γ, δ, q_{start}, F)

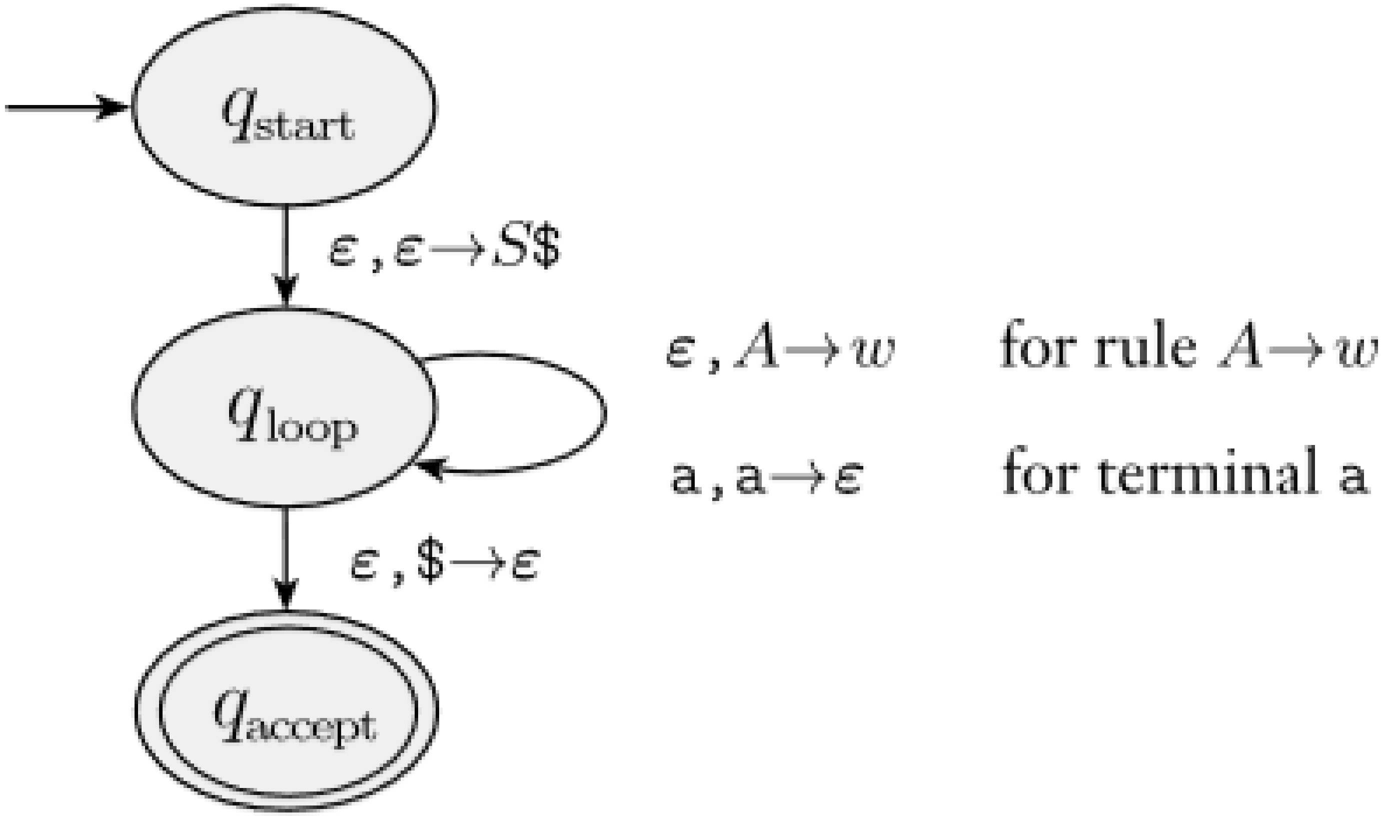
Q = {q_{start}, q_{loop}, q_{accept}} $\cup$ E , E is a set of states used to push the string w = $u_1 \cdots u_l$ on the stack

$\Gamma = \Sigma \cup \Delta \cup \{\$\}$, , F = {q_{accept}}.

δ is built by the following set of rules

- $\delta(q_{start},\varepsilon,\varepsilon) = \{(q_{loop},S\$)\}$.
- $\delta(q_{loop},\varepsilon,A) = \{(q_{loop},w) \mid$ where $A \rightarrow w$ is a rule in R}.
- $\delta(q_{loop},a,a) = \{(q_{loop},\varepsilon)\}$.
- $\delta(q_{loop},\varepsilon,\$) = \{(q_{accept},\varepsilon)\}$

State Diagram of P



To push a string on to a stack

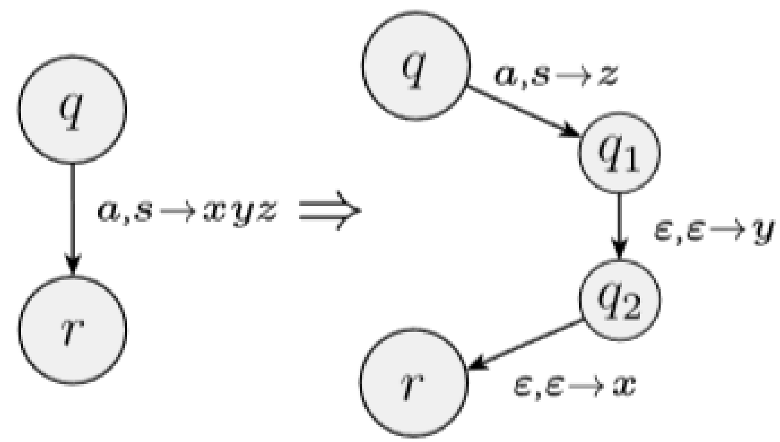
To push a string $w=u_1,\ldots u_l$, a RHS of a rule on stack, we use a set of states $q_1,\ldots,q_{l-1}$ and setting the transition function as follows:

$\delta(q,a,s)$ to contain (q_1,u_l),

$\delta(q_1,\varepsilon,\varepsilon) = \{(q_2,u_{l-1})\}$,

$\delta(q_2,\varepsilon,\varepsilon) = \{(q_3,u_{l-2})\}$, . . .

$\delta(q_{l-1},\varepsilon,\varepsilon) = \{(r,u_1)\}$.



Example:

- PDA accepts language $L = \{a^n b^n \mid n \geq 1\}$
- It is known that $L = L(G)$ with rule set:

$$S \to aSb \mid ab$$

- $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$
- $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$, $E = \{q_1, q_2, q_3\}$

$\delta(q_{start},\varepsilon,\varepsilon) = \{(q_{loop},S\$)\}$.

$\delta(q_{loop},\varepsilon,S)$ contains (q_1,b), $\delta(q_1,\varepsilon, \varepsilon) =\{(q_2, S)\}$,

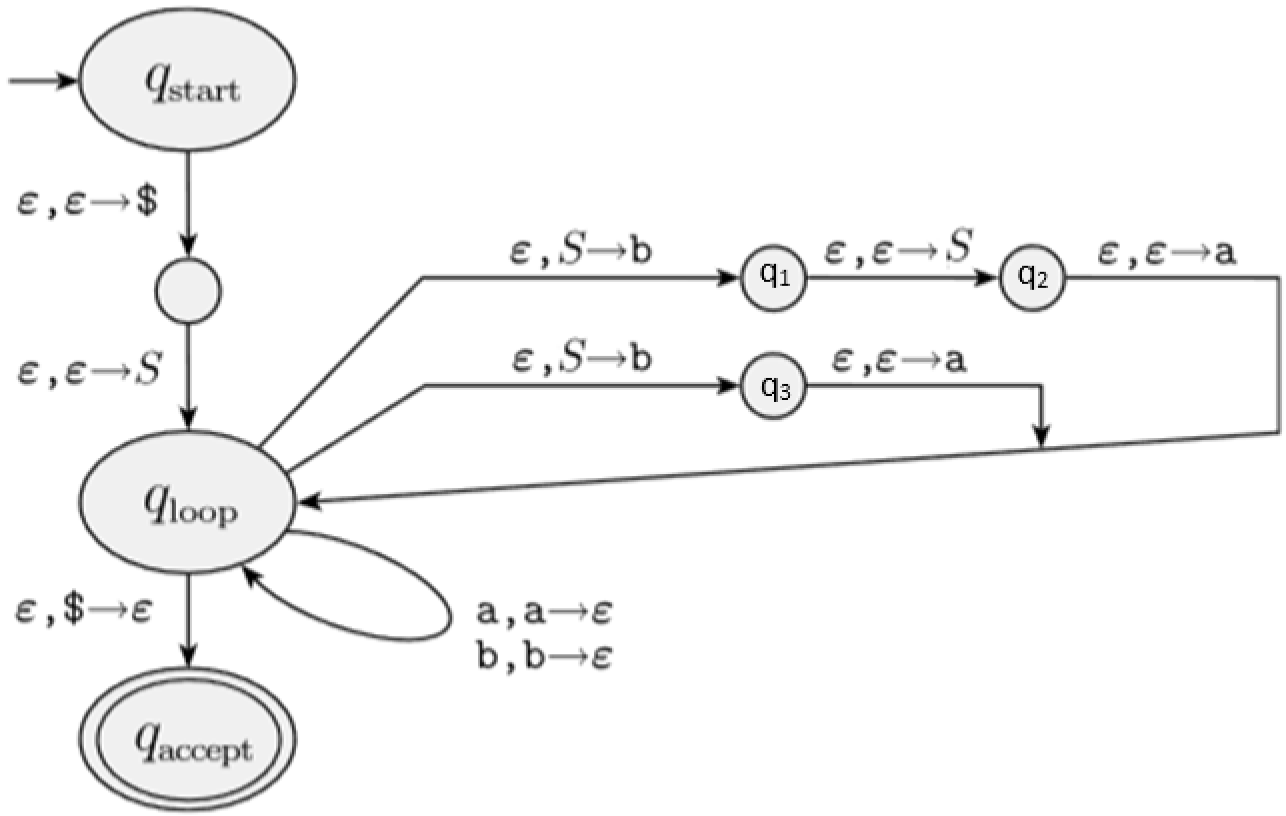
$\delta(q_2,\varepsilon, \varepsilon) =\{(q_{loop}, a)\}$,

$\delta(q_{loop},\varepsilon,S)$ contains (q_3,b), $\delta(q_3,\varepsilon, \varepsilon) =\{(q_{loop}, a)\}$,

$\delta(q_{loop},a,a) = \{(q_{loop},\varepsilon)\}$. $\delta(q_{loop},b,b) = \{(q_{loop},\varepsilon)\}$.

$\delta(q_{loop},\varepsilon,\$) = \{(q_{accept},\varepsilon)\}$

Example



q_{start}
$\varepsilon, \varepsilon \to \$$
$\varepsilon, \varepsilon \to S$
q_{loop}
$\varepsilon, \$ \to \varepsilon$
q_{accept}
a, a → ε
b, b → ε
$\varepsilon, S \to$ b
q_1
$\varepsilon, \varepsilon \to S$
q_2
$\varepsilon, \varepsilon \to$ a
$\varepsilon, S \to$ b
q_3
$\varepsilon, \varepsilon \to$ a

PDA TO CFG

- Lemma:

If a PDA recognizes some language, then it is context free.

- Proof Idea:

Create from P a CFG G that generates all strings that P accepts, i.e., G generates a string if that string takes PDA from the start state to some accepting state.

PDA To CFG–Preliminaries

Let us modify the PDA P

1. The PDA has a single accept state q_{accept}
    - Use additional $\varepsilon, \varepsilon \rightarrow \varepsilon$ transitions.
2. The PDA empties its stack before accepting.
    - Add an additional loop to flush the stack.

PDA To CFG–Preliminaries

- More modifications to the PDA P:
- Each transition either pushes a symbol to the stack or pops a symbol from the stack, but not both!.

1 Replace each transition with a pop-push, with a two-transition sequence.

For example replace a,b →c with a,b → followed by , →c, using an intermediate state.

2 Replace each transition with no pop-push, with a transition that pops and pushes a random symbol. For example, replace a, → with a, →x followed by ,x → , using an intermediate state.

PDA To CFG–Preliminaries

- For each pair of states p and q in P, the grammar with have a variable A_{pq}.
- A_{pq} generates all strings that take P from p with an empty stack, to q, leaving the stack empty.
- A_{pq} also takes P from p to q, leaving the stack as it was before p!

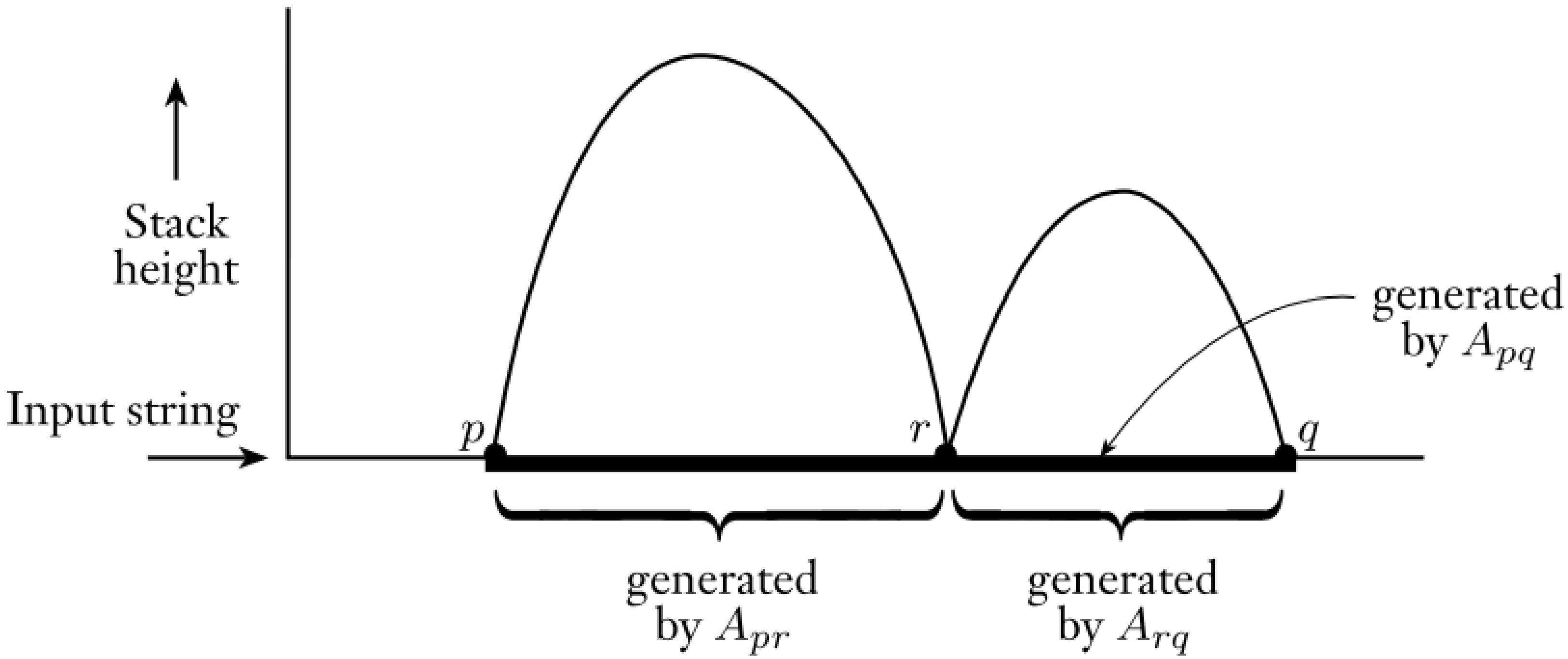
PDA To CFG–Preliminaries

- Let x be a string that takes P from p to q with an empty stack.
- There are two cases:
  - Symbol pushed after p, is the same symbol popped just before q
  - If not, that symbol should be popped at some point before!
  - First case can be simulated by rule $A_{pq} \rightarrow aA_{rs}b$ Read a, go to state r, then transit to state s somehow, and then read b.
  - Second case can be simulated by rule $A_{pq} \rightarrow A_{pr}A_{rq}$ r is the state the stack becomes empty on the way from p to q

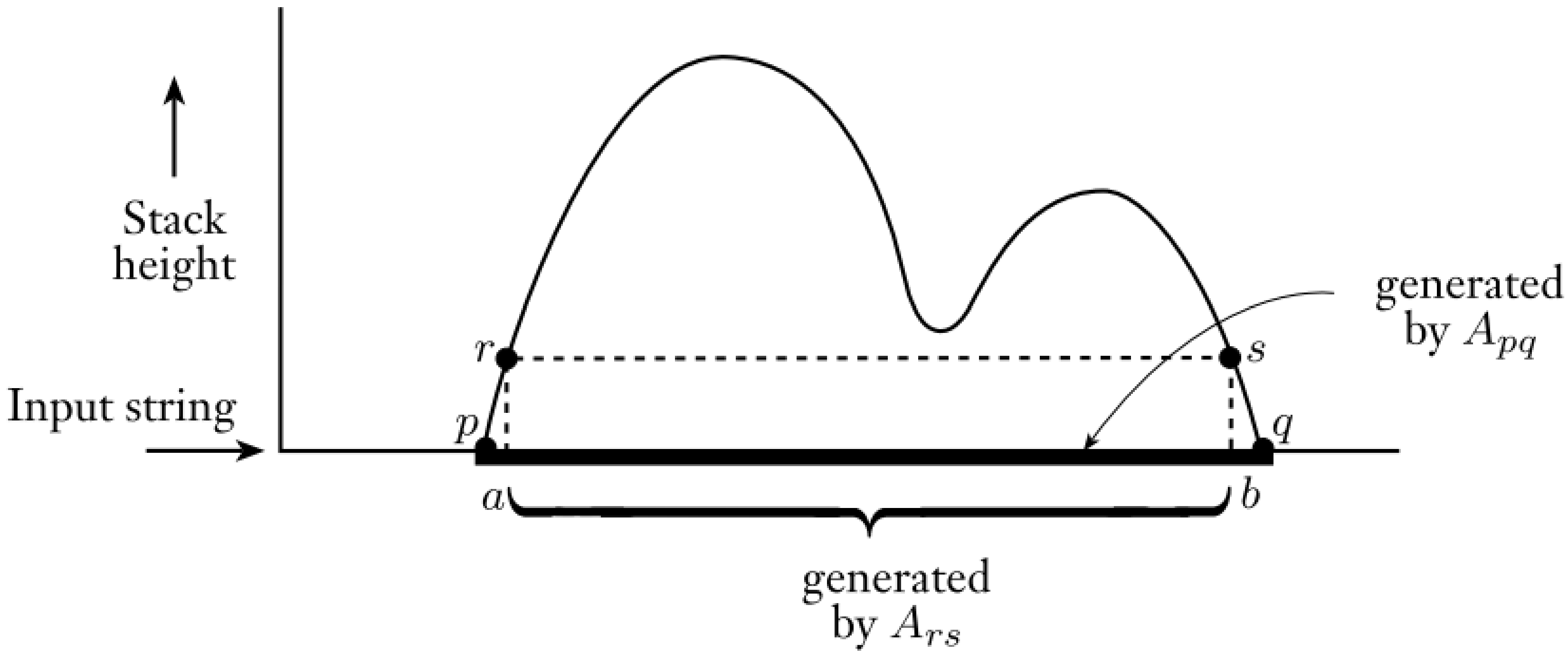
PDA to CFG: proof

- Assume $P = (Q,\Sigma,\Gamma,\delta,q_0,\{q_{accept}\})$. The variables of G are $\{A_{pq} \mid p,q \in Q\}$ The start variable is Aq_0,q_{accept}
- The rules of G are as follows:
- 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) Add rule $A_{pq} \to aA_{rs}b$ to G.
- For each $p,q,r \in Q$, add rule $A_{pq} \to A_{pr}A_{rq}$ to G. For each, $p \in Q$, add the rule $A_{pp} \to \varepsilon$ to G.

PDA computation for $A_{pq} \rightarrow aA_{rs}b$



PDA computation for $A_{pq} \rightarrow A_{pr}A_{rq}$



Example

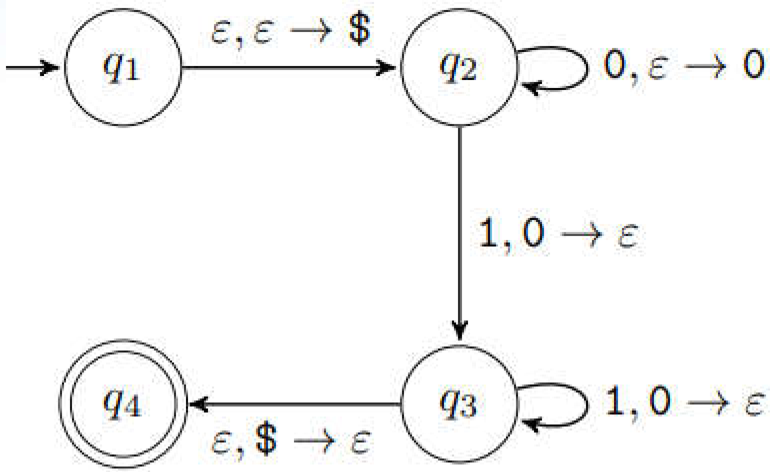
$A_{11} \rightarrow \varepsilon \qquad A_{22} \rightarrow \varepsilon$

$A_{33} \rightarrow \varepsilon \qquad A_{44} \rightarrow \varepsilon$

$A_{11} \rightarrow A_{11}A_{11} \mid A_{12}A_{21} \mid A_{13}A_{31} \mid A_{14}A_{41}$
$A_{12} \rightarrow A_{11}A_{12} \mid A_{12}A_{22} \mid A_{13}A_{32} \mid A_{14}A_{42}$
$A_{13} \rightarrow A_{11}A_{13} \mid A_{12}A_{23} \mid A_{13}A_{33} \mid A_{14}A_{43}$
. . .
$A_{42} \rightarrow A_{41}A_{12} \mid A_{42}A_{22} \mid A_{43}A_{32} \mid A_{44}A_{42}$
$A_{43} \rightarrow A_{41}A_{13} \mid A_{42}A_{23} \mid A_{43}A_{33} \mid A_{44}A_{43}$
$A_{44} \rightarrow A_{41}A_{14} \mid A_{42}A_{24} \mid A_{43}A_{34} \mid A_{44}A_{44}$
$A_{23} \rightarrow 0A_{22}1 \mid 0A_{23}1$

$A_{14} \rightarrow \varepsilon A_{23}\varepsilon$

$S \rightarrow A_{14}$



Pumping lemma for CFG

Lemma

If L is a CFL, then there is a number p such that if s is any string in L of length at least p, then s can be divided into 5 pieces s = uvxyz satisfying the conditions:

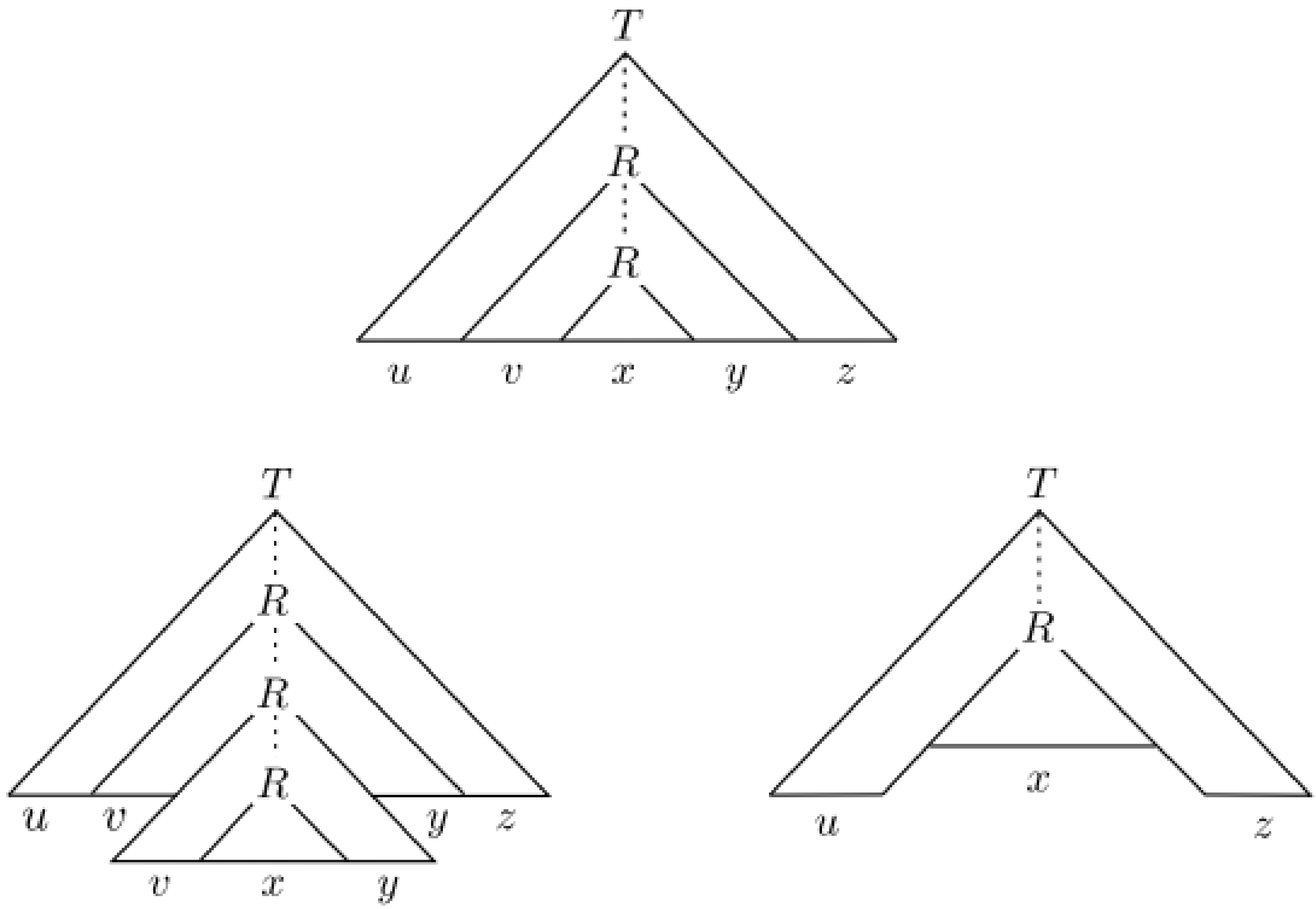
- $|vy| > 0$
- $|vxy| \leq p$
- for each $i \geq 0$, $uv^ixy^iz \in L$

p is called the pumping length. It is the number of variables of the grammar that generates L

Applications of the pumping lemma

- To prove that a language is not a CFL.
- Use proof by contradiction
- Assume L is CFL. Following the pumping lemma, we have the pumping length p
- We pick string s in L such that $|s| \geq p$. By the pumping lemma, s = uvxyz - the decomposition subject to $|vxy| \leq p$ and $|vy| \geq 1$.
- We try to pick an i such that $uv^ixy^iz \notin L$ for all possible decompositions. If we can find an i, it's a contradiction
- The contradiction proves that L is not context free.

Surgery on Parse Trees (Grammar in CNF)



Example

- Consider the language $L = \{a^n b^n c^n \mid n \geq 0\}$ Opponent picks p. We pick $s = a^p b^p c^p$. Clearly $|s| \geq p$. Opponent may pick the string partitioning in a number of ways. Let's look at each of these possibilities:

Cases 1,2 and 3:

vxy contains symbols of only one kind

1. Only *a*'s: $\underbrace{a\cdots a}_{u}\underbrace{a\cdots a}_{vxy}\underbrace{a\cdots ab\cdots bc\cdots c}_{z}$
2. Only *b*'s: $\underbrace{a\cdots ab}_{u}\underbrace{b\cdots b}_{vxy}\underbrace{b\cdots bc\cdots c}_{z}$
3. Only *c*'s: $\underbrace{a\cdots ab\cdots bc}_{u}\underbrace{c\cdots c}_{vxy}\underbrace{c\cdots c}_{z}$

Pumping v and y will introduce *more symbols of one type* into the string. The resulting strings will not be in the language.

Cases 4 and 5

vxy contains two symbols – crosses symbol boundaries.

1. Only *a*'s and *b*'s: $\underbrace{a\cdots a}_{u}\underbrace{a\cdots ab\cdots b}_{vxy}\underbrace{b\cdots bc\cdots c}_{z}$
2. Only *b*'s and *c*s: $\underbrace{a\cdots ab\cdots b}_{u}\underbrace{b\cdots c}_{vxy}\underbrace{c\cdots c}_{z}$

Note that vxy has length at most p so can not have 3 different symbols. Pumping v and y will both upset the *symbol counts* and the symbol patterns. The resulting strings will not be in the language.