Pumping Lemma

Regular Language

designed by → Finite State Machine (FSM)

proved by → pumping lemma

– proves that language is **not** regular

A → regular language

pumping length → p

string → s, length of s ≥ p

s ∈ A

$|s| \geq p$

s is divided into 3 parts

$s = xyz$

Conditions to be regular language:

1) $xy^i z \in A$ $(i \geq 0)$
2) $|y| > 0$
3) $|xy| \leq p$

} pumping conditions

Proof by contradiction:

Steps: each case → 3 conditions check

Example: Using Pumping Lemma prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is not regular

Proof: Let A be regular language.

↳ pumping length → $p = 7$

Any string, $S = a^p b^p = a^7 b^7$

→ S = aaaaaaabbbbbbb

Case 1: The (y) is in the 'a' part

aaaaaaabbbbbbb

x y z

Here, $|y| > 0$

$|xy| = 7 \quad p = 7$

$\therefore |xy| \leq p$

② no

③ no

$xy^i z \rightarrow x y^2 z$

→ aa aaaaaaaaa abbbbbbb

x y^2 z

Here, length of 'a' part = 11

length of 'b' part = 7

$\therefore 11 \neq 7$

① no condition:

Case 2: The y is in the 'b' part.

aaaaaaaabbbbbbbb
x y z

Here, $|y| > 0$
$|xy| = 9$, $p = 7$
$\therefore |xy| \not\leq p$

$xy^i z \rightarrow xy^2 z$

$\rightarrow$ aaaaaaaabb bbbbbbbbb
x y^2 z

Here, length of 'a' part = 7
length of 'b' part = 11 $\therefore 7 \neq 11$

Case 3: The y is in the 'a' and 'b' part

aaaaaaabbbbbbbb
x y z

$|y| > 0$
$|xy| = 9$, $p = 7$
$\therefore |xy| \not\leq p$

$xy^i z \rightarrow xy^2 z$

$\rightarrow$ aaaaa aabb aabb bbbbb
y^2

It doesn't follow the pattern $a^n b^n$,

So, none of these can satisfy all the 3 pumping conditions at the same time. So, S is not regular.

Example: Using pumping lemma prove that language $A = \{yy \mid y \in \{0,1\}^*\}$ is not Regular.

Proof: Let A be a regular language with pumping length p

Let $S = \underbrace{0^p 1}_{y} \underbrace{0^p 1}_{y}$

For $p = 7$, $S = \underbrace{00}_{x}\underbrace{0000}_{y}\underbrace{10000001}_{z}$

① $xy^i z \rightarrow xy^2 z$

$\rightarrow \underbrace{00}_{x}\underbrace{00000000}_{y^2}\underbrace{0100000001}_{z} \notin A$ ✗

② $|y| = 4 > 0$ ✓

③ $|xy| = \boxed{6 \leq p} = 7$ ✓

Hence, ① no. condition isn't fulfilled, A is not a Regular language.

Regular Grammar

Context free grammar accepts **context free language**

accepted by → Automata used: **Pushdown** automata

Regular grammar → regular language

automata : **finite state automata**

Grammar 'G' → described formally using 4 tuples

∴ $G = (V, T, S, P)$

V → variables / non-terminal symbols' set

T → terminal symbols

(S) → start symbol

(P) → **production rules** of terminal and non-terminals

form: $\alpha \rightarrow \beta$ [α, β are strings on $V \cup T$]

at least a symbol of $\alpha \in (V)$

↳ non terminal ← variables

Ex: $G = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

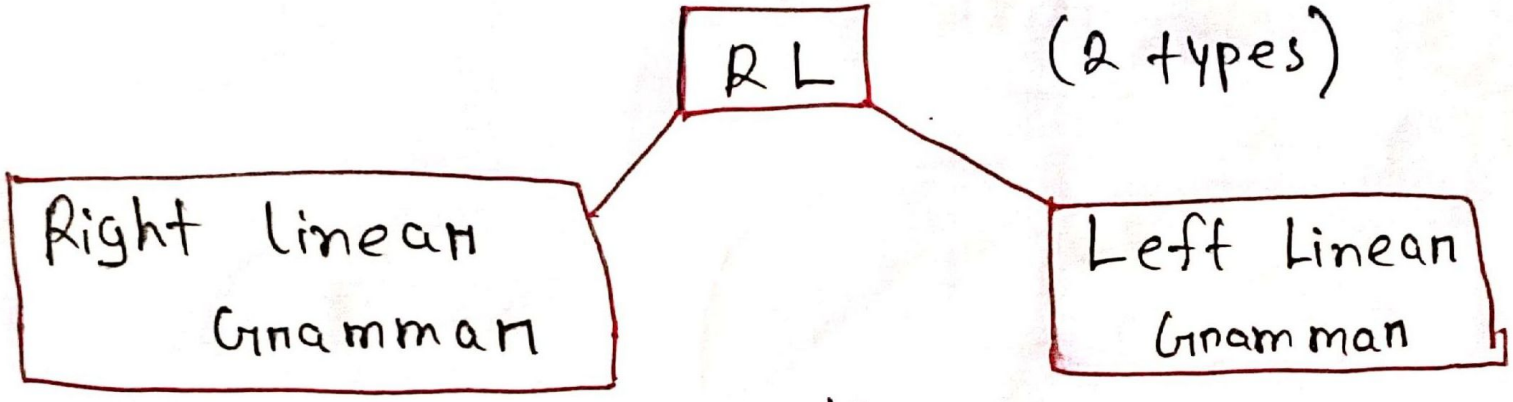
Here, $V = \{S, A, B\}$

$T = \{a, b\}$

$S = S$

$P = S \to AB, A \to a, B \to b$

Eg: $S \to AB \to aB \to ab$



N.B:
at least a variable (A) in the LHS

Eg: $A \to x(B)$ → variable in the right side

$A \to x$

Eg: $A \to (B)x$

$A \to x$

Example:

$S \to ab\underline{S} \mid b$ → Right linear

$S \to \underline{S}bb \mid b$ → left linear

Context Free Grammar

CFG forms

NT → T+NT
T
NT

described using 4 tuples

$G = \{ V, \Sigma, S, P \}$

(V) → set of variables / non-terminals

(Σ) → set of terminals

(S) → start symbol

(P) → production rule

$A \rightarrow a$

here, $A \in V$ → In the left, variable / NT

$a = \{ V \cup \Sigma \}^*$ → both non-term and terms

In the night

Ex: For generating a language that generates equal number of a's and b's in the form $a^n b^n$, the CFG will be defined as $G = \{ (S, A), (a, b), (S \rightarrow aAb, A \rightarrow aAb \mid \epsilon) \}$

Proof:

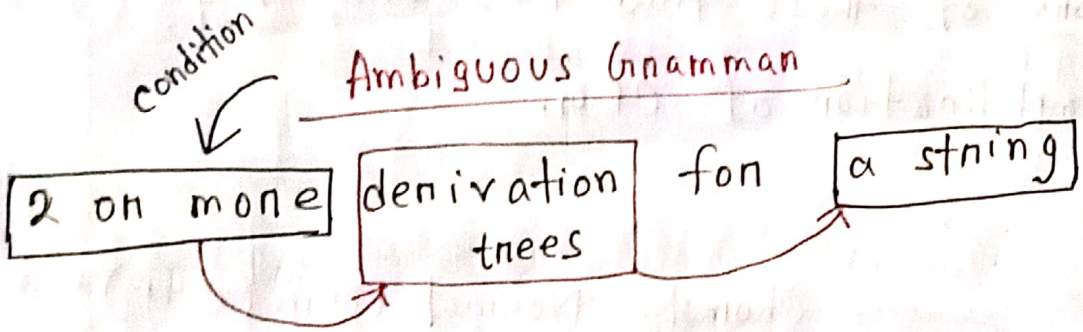
→ $S \rightarrow aAb$

$\rightarrow a\,aAb\,b$ ($A \rightarrow aAb$)

$= a\,a\,\underline{a\,A\,b}\,b\,b$ $(A \to a\,A\,b$ again$)$

$= a\,a\,a\,b\,b\,b$ $(A \to \epsilon)$

$= a^3\,b^3 \Rightarrow a^n\,b^n$



Example: $G = (\{S\}, \{a+b, +, *\}, \{S \to S+S \mid S*S \mid a \mid b\})$

The string $a+a*b$ can be derived as:

$S \to S+S$
$\to a+S$
$\to a+S*S$
$\to a+a*S$
$\to a+a*b$

(2 or more ways)

$S \to S*S$
$\to S+S*S$
$\to a+S*S$
$\to a+a*S$
$\to a+a*b$

Thus this grammar G is ambiguous

Simplification of CFG

* In CFG, sometimes all the production rules and symbols are not needed for the derivation of strings. Besides this, there may also be some null productions and unit productions. Eliminations of these productions and symbols is called simplification of CFG.

Chomsky Nonmal Fonm (CNF)

productions are in the following fonms;

$A \rightarrow a$

$A \rightarrow BC$

Here, A, B, C → non tenminals

a → tenminals

Conditions on RHS

1) either be two vaniables / NT
2) on a Tenminal

NT → NT (2 NT)

→ T (1 NT)

CFG → CNF

Steps:

① Let stant symbol be (S)
new stant symbol be (S')
new production $S' \rightarrow S$

② remove null productions

③ remove unit productions

④ Replace each production $A \rightarrow B_1 \cdots B_n$ $(n > 2)$
with $A \rightarrow B_1 C$ $(C \rightarrow B_2 \cdots B_n)$
Repeat this step for all productions having 2 or more symbols on right side.

more than 2 NT on RHS

goal: To make 2 NT on the RHS

⑤ Replace production $A \rightarrow aB$ (a - Terminal, B, A, X - NT)
with $A \rightarrow XB$ and $X \rightarrow a$
Repeat this step for all productions having terminals and non-terminals on right side.

when both NT and T in RHS

Goal:

1) simplifying grammar
2) Reducing structural ambiguity

Example: Convert from CFG to CNF:

P: $S \to ASA \mid aB,\ A \to B \mid S,\ B \to b \mid \epsilon$

→ (1) Let new start symbol be S'

and new production $S' \to S$

P: $S' \to S,\ S \to ASA \mid aB,\ A \to B \mid S,\ B \to b \mid \epsilon$

(2) Removing null productions: $B \to \epsilon$, $A \to \epsilon$

(a) Removing $B \to \epsilon$:

P: $S' \to S,\ S \to ASA \mid aB \mid a,$

$A \to B \mid S \mid \epsilon,\ B \to b$

(b) Removing $A \to \epsilon$:

P: $S' \to S,\ S \to ASA \mid aB \mid a \mid AS \mid SA \mid S,$

$A \to B \mid S,\ B \to b$

(3) Removing unit productions:

$S \to S$

$S' \to S$

$A \to B$

$A \to S$

(a) Removing $S \to S$:

P : $S' \to S$, $S \to ASA \mid aB \mid a \mid AS \mid SA$, $A \to B \mid S$, $B \to b$

(b) Removing $S' \to S$:

P : $S \to ASA \mid aB \mid a \mid AS \mid SA$,

$S' \to ASA \mid aB \mid a \mid AS \mid SA$,

$A \to B \mid S$, $B \to b$

(c) Removing $A \to B$:

P : $S \to ASA \mid aB \mid a \mid AS \mid SA$,

$S' \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to b \mid S$, $B \to b$

(d) Removing $A \to S$:

P : $S \to ASA \mid aB \mid a \mid AS \mid SA$,

$S' \to ASA \mid aB \mid a \mid AS \mid SA$,

$A \to b \mid ASA \mid aB \mid a \mid AS \mid SA$,

$B \to b$

④ Finding out the productions that has more than 2 variables in RHS

$$\begin{cases} S \to ASA \\ S' \to ASA \\ A \to ASA \end{cases}$$

After removing these, we get:

P: $S \to AX \mid aB \mid a \mid AS \mid SA$

$S' \to AX \mid aB \mid a \mid AS \mid SA$

$A \to b \mid AX \mid aB \mid a \mid AS \mid SA$

$B \to b$

$X \to SA$

⑤ Changing the productions $S \to aB$, $S' \to aB$, $A \to aB$

finally we get:

P: $S \to AX \mid YB \mid a \mid AS \mid SA$

$S' \to AX \mid YB \mid a \mid AS \mid SA$

$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$

$B \to b$

$X \to SA$, $Y \to a$

which is the required CNF for the given CFG

Pushdown Automata

PDA is a way to implement a context-free grammar.

So, {context-free grammar is a language which is recognized by PDA}

* Diff. between FSM and PDA:

FSM	PDA
① less powerful	① more powerful
② limited memory	② more memory
③ don't have stack	③ has a stack with infinite memory
④ implements regular language.	④ implements context-free grammar.

push — new element added

pop — top element read and removed

PDA components : (3)

1) input tape — takes input
2) finite control unit — accept / reject
3) stack — push / pop

* PDA is formally defined by (7) tuples :

$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

Q — finite set of states

Σ — finite set of input symbols

Γ — finite stack of alphabets

δ — transition function

q_0 — stant state

z_0 — stant stack symbol ($)

F — set of final / accepting states

* δ (transition function)

takes triple $\delta(q, a, X)$ → input

Here, (q) - state $\in Q$

(a) - input $\in \Sigma$

or empty symbol

(X) - stack symbol $\in \Gamma$

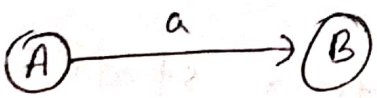
→ output pairs (p, γ)

(p) - new state

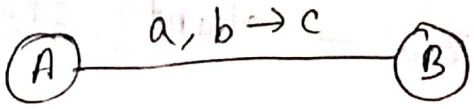
(γ) - string of stack symbols

Graphical notation

FSM



PDA

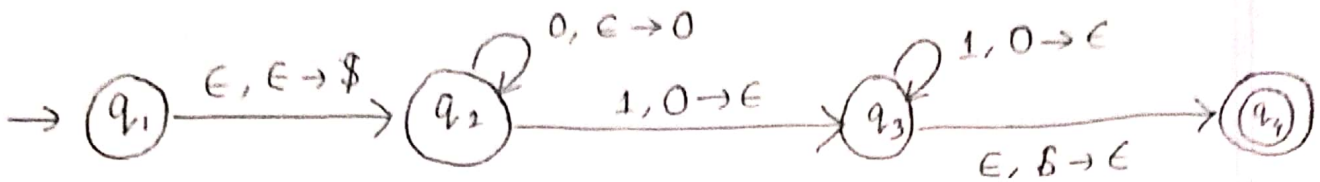


Here, a - input

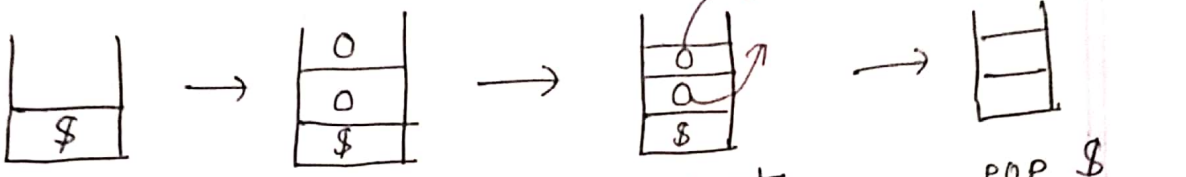
b - if top element, then pop

c - push

Ex: Construct a PDA that accepts $L = \{ 0^n 1^n \mid n \geq 0 \}$



Suppose, 0 0 1 1



(push dollar)

(0 input, q_2 state nothing popped push 0)

(1 input, q_3 state 0 popped, push nothing)

POP $ q_4 state ↓ (final)

Ex: PDA that accepts even palindromes of the form

$$L = \{ w w^R \mid w = (a+b)^+ \}$$