Regular Grammer

Grammer

it can be described using 4 tuples $G = (N, T, S, P)$ where

N = Set of variables or Non-Terminal symbols (Non V)
↑ indices.

T = Set of Terminal Symbols.

S = Start Symbol

P = Production rules for terminals & Non-Terminals.

A production rules has the form $\alpha \to \beta$ where α & β are strings on $N \cup T$ & at least one symbol of α belongs to N

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

N T S P

S → AB
→ aB
→ ab

All Type 2 are Type 3, itself
Type 0 = Type1 ∪ Type2 ∪ Type3

CHOMSKY HIERARCHY

Grammer Type	Grammer Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammer	Recursively Enumerable language	Turing Machine
Type 1	Content Sensitive grammer	Content sensitive language	Linear Bounded automaton
Type 2	Content free Grammer	Content free language	Pushdown Automaton
Type 3	Regular grammer	Regular language	Finite State Automaton

Regular Grammer

1) Right linear grammer if production rule are of form

$A \to xB$

$A \to x$

where $A, B \in N$ (non terminals)

$x \in T$ (terminals)

2) Left Linear Grammer

$A \to Bx$

$A \to x$

$A, B \in N$

$x \in T$

eg $S \to abS \mid b$ Right linear

→ Set of all strings that can be derived from grammer is said to be Language generated from that grammer.

eg $G_1 = (\{S, A\}, \{a, b\}, S, \{S \to aAb, aA \to aaAb, A \to \varepsilon\})$

$S \to aAb$

$\to aaAbb$

$\to aaaAbbb$

$\to aaabbb$ ($A \to \varepsilon$)

$aA \to aaAb$

$aA \to aaAb$

(not regular) $a^n b^n$ form

↑ Language $L(G_1)$

eg 2 $G_2 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

$S \to AB$

$\to ab$

$L(G_2) = \{ab\}$

eg 3 $G_3 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA \mid a, B \to bB \mid b\})$

$S \to AB$

$\to ab$

$S \to AB$

$\to aAbB$

$\to aabb$

$S \to AB$

$\to aAb$

$\to aaab$

$L(G_3) = \{a^m b^n \mid m \geq 0$ and $n \geq 0\}$

CONTEXT FREE LANGUAGE

Set of all CFL is identical to set of languages accepted by Pushdown automata.

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

V - Set of Variables or Non-Terminal Symbols
Σ - Set of terminal symbols
S - Start Symbol
P - Production rule

here $P \Rightarrow \quad A \rightarrow a$

where $a = \{V \cup \Sigma\}^* \quad \{A \in V$

eg for generating language that generates equal no. of a's & b's in form $a^n b^n$, Context free grammer will be defined as

$G = \{(S, A), (a, b), (S \rightarrow aAb, A \rightarrow aAb \mid \epsilon)\}$

$S \rightarrow aAb$
$\rightarrow aaAbb = aabb \quad a^2b^2$

Method to find whether a String belongs to a Gra. Or Not

1) Start with start symbol & choose closest production that matches to given string.
2) Replace variables with its most appropriate production. Repeat the process until the string is generated or until no other production are left.

eg :- Verify whether the grammer $S \to 0B \mid 1A$, $A \to 0 \mid 0S \mid 1AA \mid \wedge$, $B \to 1 \mid 1S \mid 0BB$ generate string 00110101

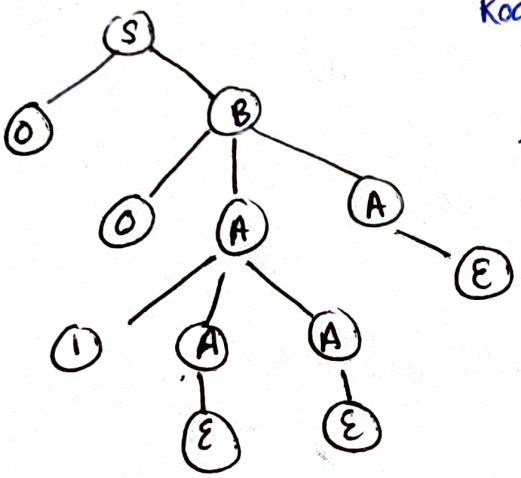
Sol

$S \to 0B$
$S \to 00BB$
$S \to 001B$
$S \to 0011S$
$S \to 00110B$
$S \to 00110 1S$
$S \to 00110 10B$
$S \to 00110101$ Hence string is generated.

DERIVATION TREE

A Derivation tree or Parse tree is an ordered rooted tree that graphically represents the semantic information of strings derived from context free grammer.

eg for grammer $G = \{V, T, P, S\}$ where $S \to 0B$, $A \to 1AA \mid \varepsilon$.
$B \to 0AA$



Root vertex → must be labelled by start symbol

Vertex → Labelled by non terminal symbol

Leaves → labelled by terminal symbols or ε

Left derivation tree

→ obtained by applying production to left most variable in each step.

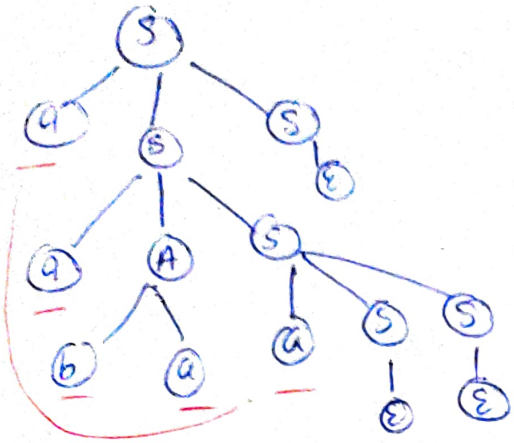
right derivation tree

→ obtained by applying production to right most variable in each step.

eg for generating aabaa from $S \to aAS \mid aSS \mid \varepsilon$

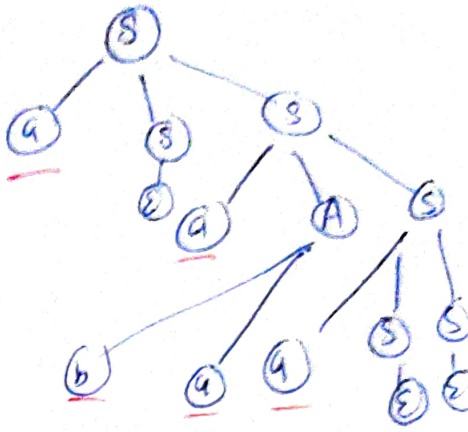
$A \to SbA \mid ba$

Left Derivation tree



aabaa

Right Derivation tree



Selecting right most

Ambiguous Grammer

A grammer is said to be Ambiguous if there are two or more derivation tree for a string ω (that means 2 or more left derivation tree)

eg $G=(\{S\}, \{a+b, +, *\}, P, S\}$ where P consist of $S \to S+S \mid S*S \mid a \mid b$

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

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

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

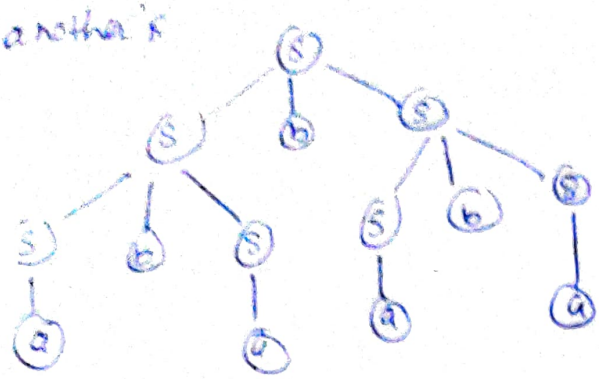
So is ambiguous.

Q for G: $S \to SbS$

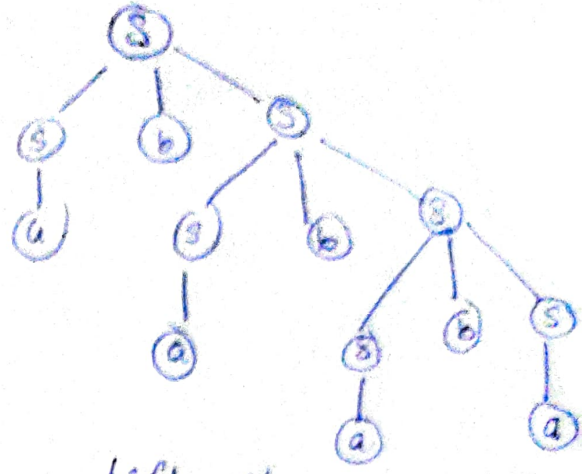
$S \to a$

for string abababa.

another's



not left most.



Leftmost

$S \Rightarrow SbS$ → solved leftmost part first

$S \Rightarrow abS$

$S \Rightarrow abSbS$

$S \Rightarrow ababS$

$S \Rightarrow ababSbS$

$S \Rightarrow abababS$

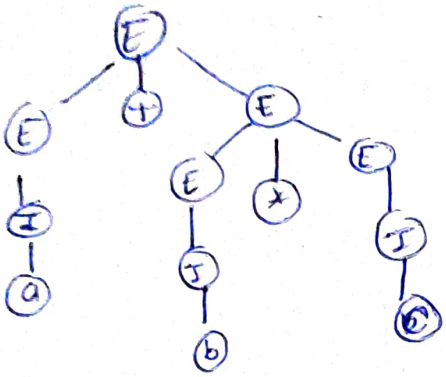
$S \Rightarrow abababa$

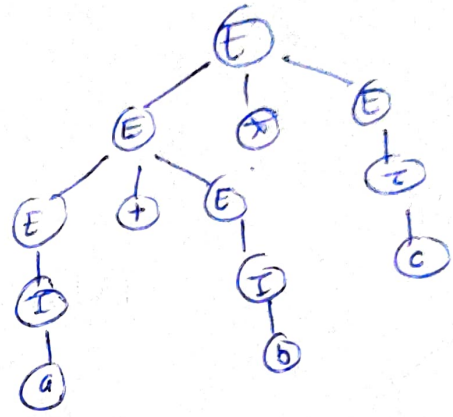
- For one derivation tree, there will be only one leftmost derivation.

Q Consider grammer $G = (N, T, E, P)$ with $N = \{E, I\}$,

$T = \{a, b, c, +, *\}$ & P : $E \to E+E$,
$E \to E*E$,
$E \to I$
$I \to a \mid b \mid c$

String $a+b*c$ is in $L(G)$



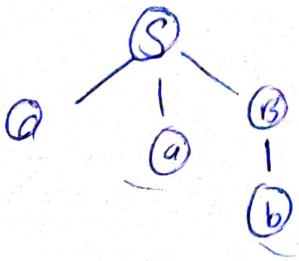


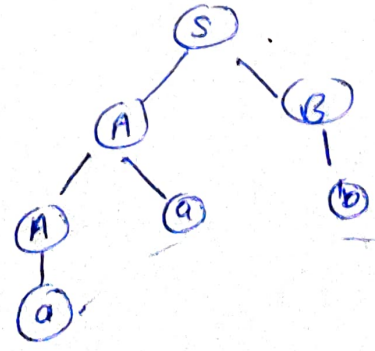
is ambiguous as more than one tree

Q Show that following grammer is ambiguous :-

$S \to AB \mid aaB$
$A \to a \mid Aa$
$B \to b$

To prove G is ambiguous, we have to find string $\omega \in L(G)$ which is ambiguous. Consider $\omega = aab \in L(G)$, then we get 2 derivation trees thus G is ambiguous





Simplification of CFG (Reduction of CFG)

In CFG, sometimes all the production rules & symbols are not needed for derivation of strings. Besides this, there may also be some NULL productions and UNIT productions. Elimination of this productions & symbols is called simplification of CFG.

Simplification consists of following steps :-

1) Reduction of CFG
2) Removal of Unit productions
3) Removal of NULL productions

1) Reduction of CFG.

CFG are reduced in 2 phases

- **Phase 1** : Derivation of equivalent grammer G', from CFG, G such that each variable derives some terminal string.

Procedure

Step 1 : Include all symbols W_1, that derives some terminal & initialize $i = 1$.

Step 2 : Include symbols W_{i+1}, that derives W_i

Step 3 : Increment i & repeat step 2, until $W_{i+1} = W_i$

Step 4 : Include all production rules that have W_i in it.

Phase 2 : Derivation of an equivalent grammer G'', from CFG, G' such that each symbol appears in sentential form.

Procedure

Step 1 : Include start symbol in Y_1 & initialize $i = 1$

Step 2 : Include all symbols Y_{i+1} that can be derived from Y_i & include all production rules that have been applied.

Step 3 : Increment i & repeat step 2, until $Y_{i+1} = Y_i$

eg find reduced grammer for

P: S → AC | B , A → a , C → c | Bc , E → aA | e

Phase 1 : T = {a, c, e}

W_1 = {A, C, E} — W_1 → that can derive terminal symbols

W_2 = {A, C, E, S} — W_2 → that can be derived from W_1

Same W_3 = {A, C, E, S} — W_3 = that can derive W_2

So

G' = {(A, C, E, S), {a, c, e}, P, (S)} (S → root)

P : S → AC , A → a , C → c , E → aA | e

Phase 2 : Y_1 = {S}

Y_2 = {S, A, C} — Y_2 → can be derived from Y_1, S → AC

Y_3 = {S, A, C, a, c} — Y_3 ⇒ can be derived from Y_2

Same Y_4 = {S, A, C, a, c}

G'' = {(A, C, S) (a, c), P, {S}} → Reduced grammer

P ⇒ S → AC , A → a , C → c

• Removal of Unit Productions

Any Production Rule of form A → B where A, B ∈ Non Terminals is called unit production.

Procedure for Removal

Step 1 : To remove A → B, add production A → x to grammer rule wherever B → x occurs in grammer [x ∈ T].

Step 2 : Delete A → B from grammer

Step 3 : Repeat from Step 1 until all Unit Productions are removed.

eg: Remove unit productions from

P: $S \to Xy$, $X \to a$, $y \to z \mid b$, $Z \to M$, $M \to N$, $N \to a$

unit productions

$y \to z$, $Z \to M$, $M \to N$

1) Since $N \to a$, we add $M \to a$

$S \to Xy$, $X \to a$ $y \to z \mid b$ $Z \to M$ $M \to a$ $N \to a$

2) Since $M \to a$ remove $Z \to a$

$S \to Xy$, $X \to a$, $y \to z \mid b$ $Z \to a$, $M \to a$ $N \to a$

3) Since $Z \to a$ remove $y \to z$

$S \to Xy$, $X \to a$, $y \to a \mid b$ $Z \to a$ $M \to a$ $N \to a$

unreachable

4) Remove unreachable

so rule is

P: $S \to Xy$, $X \to a$, $y \to a \mid b$

Removal of NULL Productions

In a CFG, a Non-terminal Symbol 'A' is a nullable varea if there is a production $A \to \varepsilon$ or there is a derivation that starts at 'A' & leads to ε.

Procedure

Step 1: To remove $A \to \varepsilon$, look for all productions whose right side contains A.

Step 2: Replace each occurance of 'A' in each of these productions with ε.

Step 3: Add resultant productions to Grammer.

eg Remove Null productions from following grammer.

$S \to ABAC$ $A \to aA \mid \varepsilon$ $B \to bB \mid \varepsilon$ $C \to c$

null productions $A \to \varepsilon$ $B \to \varepsilon$

1) To eliminate $A \to \varepsilon$

$S \to ABAC$

- $S \to ABC \mid BAC \mid BC$

$A \to aA$

- $A \to a$

Now P: $S \to ABAC \mid ABC \mid BAC \mid BC$

$A \to aA \mid a$ $B \to bB \mid \varepsilon$ $C \to c$

2) eliminate $B \to \varepsilon$

$S \to AAC \mid AC \mid C$ $B \to b$

Now P: $S \to ABAC \mid ABC \mid BAC \mid BC \mid AAC \mid AC \mid C$

$A \to aA \mid a$

$B \to bB \mid b$

$C \to C$

CHOMSKY NORMAL FORM

In chomsky normal form (CNF) we have a restriction on length of RHS, which is elements in RHS should either be two variables or a Terminal.
A CFG is a chomsky Normal form if productions are in following form:

$$A \rightarrow a$$
$$A \rightarrow BC$$

where A, B & C are non-terminals & a is a terminal.

Steps to convert a given CFG to Chomsky normal form

Step 1: If start symbol S occurs on some right side, create a new start symbol S' & new production $S' \rightarrow S$.

Step 2: Remove Null productions

Step 3: Remove Unit products.

Step 4: Replace each production $A \rightarrow B_1 \ldots B_n$ where $n > 2$, with $A \rightarrow B_1 C$ where $C \rightarrow B_2 \ldots B_n$. Repeat this step for all productions having two or more symbols on right side.

Step 5: If right side of any production is in the form $A \rightarrow aB$, where 'a' is a terminal & A &B are non-terminals, then the production is replaced by $A \rightarrow XB$ and $X \rightarrow a$. Repeat this step for every production of form $A \rightarrow aB$.

eg CFG to CNF

$S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \varepsilon$

1) $S' \to S$, $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \varepsilon$

2) remove null productions $B \to \varepsilon$ which result in $A \to \varepsilon$

After removing $B \to \varepsilon$: $S' \to S$, $S \to ASA \mid aB \mid a$, $A \to B \mid S \mid \varepsilon$, $B \to b$

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

3) Removal of unit productions : $S' \to S$, $S \to S$, $A \to B$ & $A \to S$:

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

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

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

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

4) Now find productions that have more than TWO variables in RHS

$S' \to ASA$, $S \to ASA$ and $A \to ASA$

After removing this we get : $S' \to AX \mid AB \mid a \mid AS \mid SA$
$S \to AX \mid aB \mid a \mid AS \mid SA$
$A \to b \mid AX \mid aB \mid a \mid AS \mid SA$
$B \to b$
$X \to SA$

5) Now change productions $S' \to aB$, $S \to aB$ & $A \to aB$

finally we get : $S' \to AX \mid YB \mid a \mid AS \mid SA$
$S \to AX \mid YB \mid a \mid AS \mid SA$
$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$
$B \to b$
$X \to SA$, $Y \to a$

Greibach Normal form

A CFG is in Greibach Normal form if productions are in following form:

$$A \to b$$
$$A \to bC_1C_2 \dots C_n$$

where $A, C_1, \dots C_n$ are non-terminals & b is a terminal

Steps to convert CFG to GNF

1) Check if given CFG has unit productions or Null productions and Remove if there are any.

2) Check if CFG is already in Chomsky normal form (CNF) and convert it to CNF if it is not.

Change names of non-terminal symbols into some A_i in ascending order of i.

eg $S \to CA \mid BB$, $B \to b \mid SB$, $C \to b$, $A \to a$.

Replace S with A_1 A with A_3 (acc. to first occurance)
C with A_2 B with A_4

we get

$$A_1 \to A_2A_3 \mid A_4A_4$$
$$A_4 \to b \mid A_1A_4$$
$$A_2 \to b$$
$$A_3 \to a$$

Step 4 Alter rules so that Non-terminals are in ascending order, such that If production is of form $A_i \to A_j x$, then $i < j$ & should never be $i \geq j$.

here problem is

$A_4 \rightarrow b \mid A_1 A_4$

$A_4 \rightarrow b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$

→ as first Terminal then nonterminals

again problem

$A_4 \rightarrow b \mid bA_3A_4 \mid A_4A_4A_4$

left recursion

Step 5 : Remove left recursion

Introduce new variable

$Z \rightarrow A_4A_4Z \mid A_4A_4$ (A_4A_4 with with new variable, one without)

$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

So now

$A_1 \rightarrow A_2A_3 \mid A_4A_4$ ← problem as N→Ternial, (Non terminal)

$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \rightarrow A_4A_4 \mid A_4A_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

$A_1 \rightarrow bA_3 \mid b \mid bA_3A_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$

$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \rightarrow bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4 \mid$

$bA_4Z \mid bA_3A_4A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4Z$

$A_2 \rightarrow b$

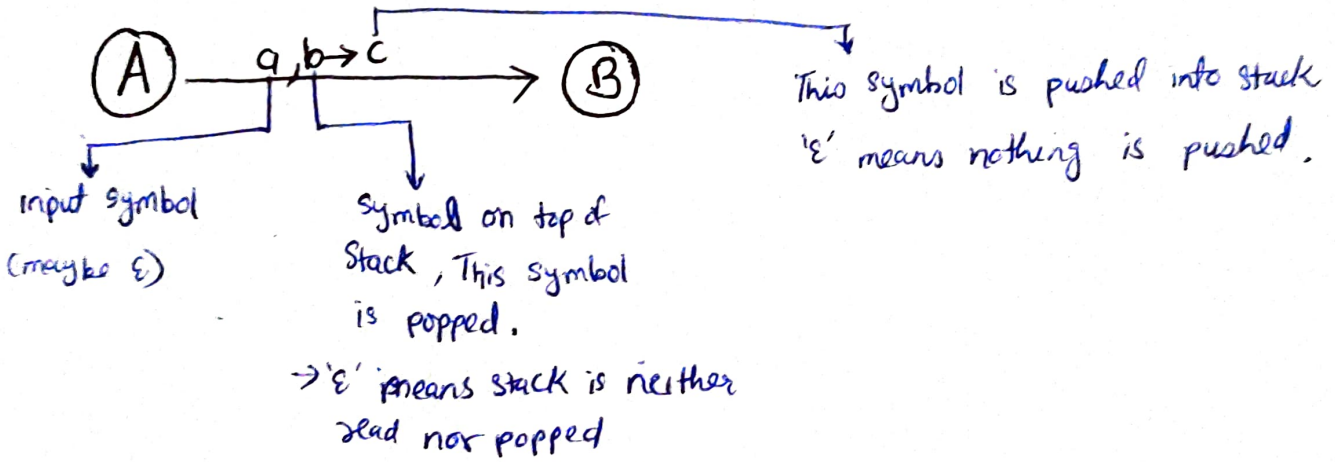
$A_3 \rightarrow a$

PUSHDOWN AUTOMATA (PDA)

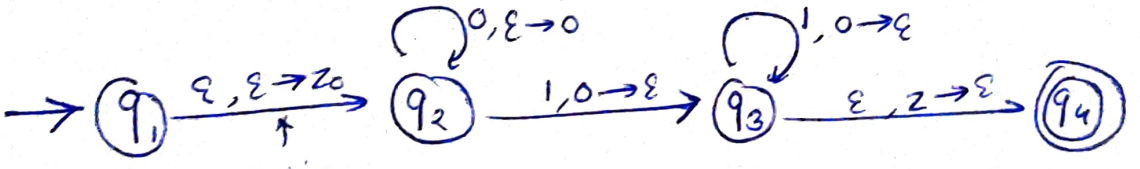
PDA → Finite State Machine + Stack ← has memory
↑ infinite memory

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

Q = A finite set of states
Σ = A finite set of input symbols
Γ = A finite stack Alphabet
δ = Transition fn → $\delta(q, a, X)$ X → stack symbol.
q_0 = Start state z_0 = Start stack symbol
F = Set of final / Accepting states.



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



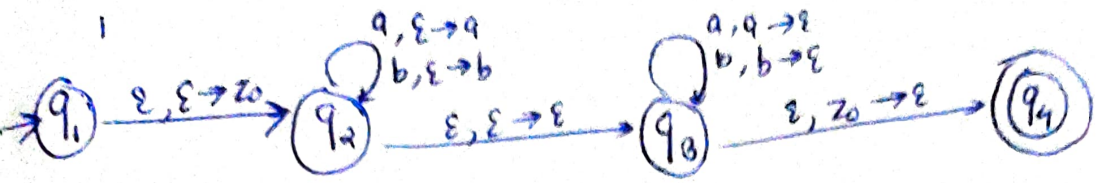
pushing z_0 at bottom.

String is accepted if 1) reached final state
2) stack gets empty

Sometimes $ is used in place of z_0

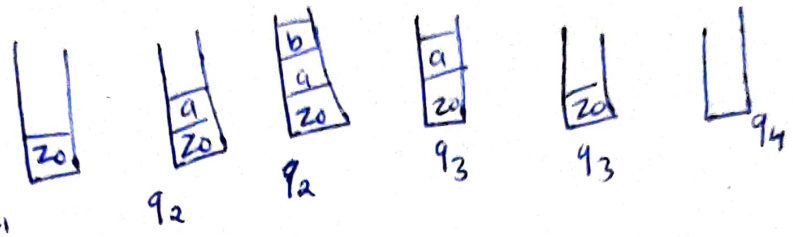
eg2 Even pallindrom (palindrom of even length)

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



a b | b a

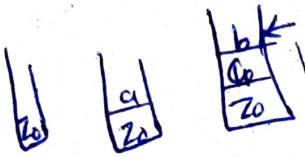
eg for abba →



at mid
a, ε → ε

but if abab

X



now $a, a \to \varepsilon$ } in order
$b, b \to \varepsilon$

bottom most but here a → abab

So problem

So not reach q_3.

Theorem

A language is content free iff some pushdown Automata recognizes it.

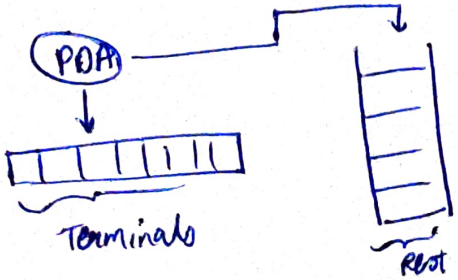
eg give CFG $S \to BB \mid A$, $A \to 0A \mid \varepsilon$, $B \to BB1 \mid 2$

construct PDA

General form :

aaaa, BaBC,

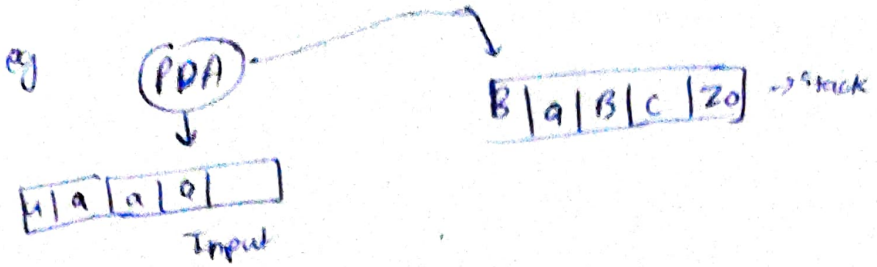
terminal Rest

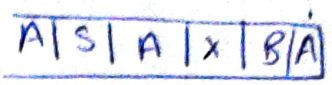


At each step expand Left most derivation

Left most derivation

→ S
→ BS
→ BB1S
→ 2B1S
→ 221S
→ 221A
→ 221ε
→ 221

eg 

eg Rule $B \rightarrow ASAxBA$ → ---aaa 

- Match Stack Top to a Rule
- Pop Stack
- Push Right Hand Side of Rule onto Stack,

for $A \rightarrow BCD$