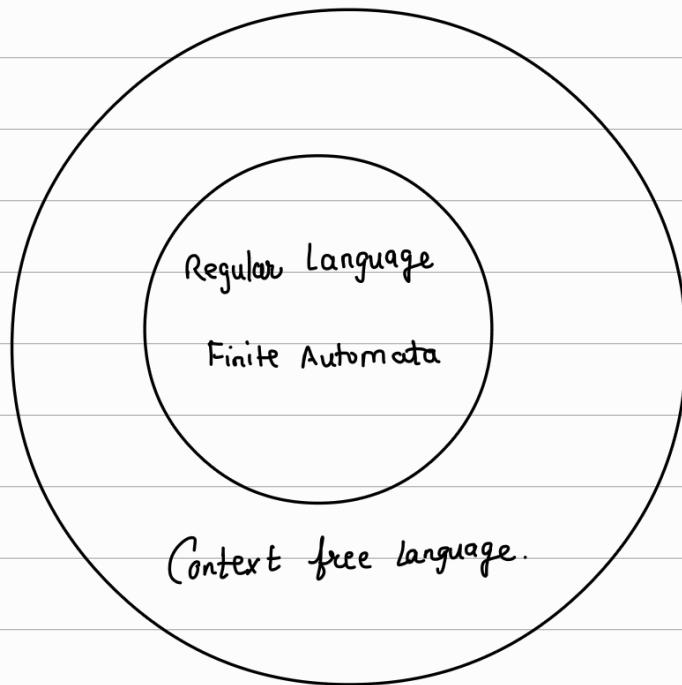


Context Free Language



Context Free Grammar \leftarrow Regular Expression

Push Down Automata.

- Context free Grammars are more expressive than finite automata.
- Language generated by context free Grammar is context free language.

→ gives
⇒ derives

Production is defining rule, always single line.

Derivation is deriving a rule,

Context free Grammar $G = (V, T, P, S)$ where

V is the finite set of variables,

T is the finite set of alphabets/ terminals

P is the finite set of productions

$S \in V$ is the start variable.

- Each production P is of the form $\frac{A \rightarrow S}{(A \text{ gives } S)}$ where $A \in V$ and $S \in (V \cup T)^*$

$\langle \text{sentence} \rangle \rightarrow \langle \text{noun_phrase} \rangle \langle \text{predicate} \rangle \rightarrow \text{The Cat runs.}$

$\langle \text{noun_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle$

$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle$

$\langle \text{article} \rangle \rightarrow \text{the}$

$\langle \text{noun} \rangle \rightarrow \text{Cat}$

$\langle \text{verb} \rangle \rightarrow \text{runs}$

IF you have productions:

$$A \rightarrow d_1$$

$$A \rightarrow d_2$$

$$A \rightarrow d_3$$

you can write them as,

$$A \rightarrow \alpha_1 | \alpha_2 | \alpha_3$$

Design CFG for language $a^n b^n$ where $n \geq 0$.

$$P \rightarrow S \rightarrow a^* b | \epsilon$$

$$S \rightarrow a s b | \epsilon$$

$$\Rightarrow a a s b b$$

$$\Rightarrow a^2 a s b b^2$$

$$\Rightarrow a^3 a s b b^3$$

$$\Rightarrow a^{n-1} a s b b^{n-1}$$

$$\Rightarrow a^n s b^n$$

to remove S , we

need s to become ϵ ,

∴ we write

$$S \rightarrow a s b | \epsilon$$

Design CFG means majorly designing productions

Final Answer:

$$G = (\{S\}, \{a, b\}, P, S)$$

Grammar

derived

V is one step derivable from U denoted by $U \xrightarrow{f} V$, if $U = x \alpha z$,

$$V = x \beta z$$

if $\alpha \xrightarrow{\beta}$
gives

Productions in Grammar

$$S \rightarrow AB$$

$$A \rightarrow aaA \mid \epsilon$$

$$B \rightarrow Bb \mid \epsilon$$

$$S \Rightarrow AB$$

$$\Rightarrow aaAB$$

$$\Rightarrow aa\epsilon B$$

$$\Rightarrow aaBb$$

$$\Rightarrow aa\epsilon b$$

$$\Rightarrow aab$$

Milam's
way



$$\text{OR } \Rightarrow aaABb$$

$$\Rightarrow aa\epsilon Bb$$

$$\Rightarrow aa\epsilon Bb$$

$$\Rightarrow aa\epsilon\epsilon b$$

$$\Rightarrow aab$$

In a step you have to apply only one production

Can we simultaneously remove ϵ and production B .

→ Yes

CFL:

given a Context Free Grammar $G = (V, T, P, S)$ the language generated from G is called context free grammar.

It is denoted from $L(G) = \{w \mid s \xrightarrow{*} w\}$

CFL for strings of balanced parentheses {, (,), }

Terminals $\rightarrow \{, (,), \}$

Production:

$$① \quad S \rightarrow SS \mid \epsilon \mid (S) \mid \{S\}$$

R-Rule

$$S \Rightarrow SS \dots R_1$$

$$\Rightarrow \{S\} S \dots R_4$$

$$\Rightarrow \{(S)\} S \dots R_3$$

$$\Rightarrow \{(\epsilon)\} S \dots R_2$$

$$\Rightarrow \{\epsilon\} S \dots \epsilon\text{-removing}$$

$$\Rightarrow \{\epsilon\}(S) \dots R_3$$

$$\Rightarrow \{\epsilon\}(\epsilon) \dots R_2$$

$$\Rightarrow \{\epsilon\}() \dots \epsilon\text{ Removal}$$

A language can have more than one grammar.

② Production:

$$S \rightarrow (S)S \mid \{S\}S \mid \epsilon$$

$$\{\epsilon\}()$$

$$S \Rightarrow \{S\}S$$

$$\Rightarrow \{(S)\}S$$

$$\Rightarrow \{(\epsilon)\}S$$

$$\Rightarrow \{\epsilon\}(S)S$$

$$\Rightarrow \{\epsilon\}(\epsilon)S$$

$$\Rightarrow \{\epsilon\}C \epsilon$$

$$\Rightarrow \{\epsilon\}()$$

Design grammar for ww^R where $w \in \{a,b\}^*$

$S \rightarrow aSa \mid bSb \mid \epsilon \mid ab$

$abba$

$S \Rightarrow aSa$

$S \Rightarrow abSba$

$S \Rightarrow ab \epsilon ba$

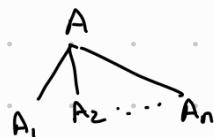
$S \Rightarrow abba$.

A parse tree of a derivation is a tree in which each internal node is labelled with a variable.

If you have rule,

$$A \rightarrow A_1 A_2 \dots A_n$$

then you can draw parse tree



Derivation or yield is the final string obtained by concatenating labels of the leaves of tree from left to right. ignoring ϵ

Suppose, you have rule:

$$E \rightarrow E + E \mid a$$

$$E \rightarrow E + E$$

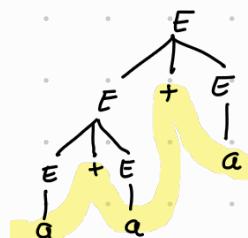
$$\Rightarrow E + E + E$$

$$\Rightarrow E + E + a$$

$$\Rightarrow E + a + a$$

$$\Rightarrow a + a + a$$

Parse Tree



→ Yield of parse tree

Q. $S \rightarrow AB$

$$A \rightarrow aaA \mid \epsilon$$

$$B \rightarrow Bb \mid \epsilon$$

Find parse tree for aab

$$S \Rightarrow AB$$

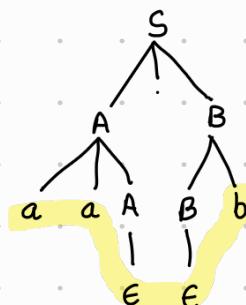
$$\Rightarrow aaAB$$

$$\Rightarrow aaA Bb$$

$$\Rightarrow aa\epsilon Bb$$

$$\Rightarrow aa\epsilon\epsilon b$$

$$\Rightarrow aab$$



$\text{Yield} = aa\epsilon\epsilon b$

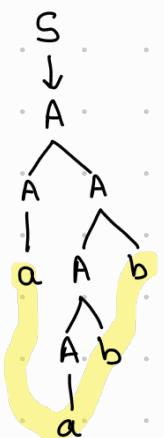
but we ignore ϵ in yield;

$\therefore \text{Yield} = aab$

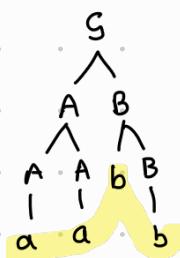
$$\begin{aligned} Q. \quad S &\rightarrow A|AB \\ A &\rightarrow \epsilon|Ab|AA|a \\ B &\rightarrow b|bc|Bc|bB \end{aligned}$$

for string aabb, can you draw more than one parse tree?

$$\begin{aligned} ① \quad S &\Rightarrow A \\ &\Rightarrow AA \\ &\Rightarrow AAb \\ &\Rightarrow AAabb \\ &\Rightarrow Aabb \\ &\Rightarrow aabb \end{aligned}$$



$$\begin{aligned} ② \quad S &\Rightarrow AB \\ &\Rightarrow AAB \\ &\Rightarrow AABB \end{aligned}$$



Thus we can have multiple parse trees, these multiple parse trees cause the compiler to get confused. This is Ambiguous Grammar.

A partial derivation tree is a subtree of a parse tree such that either all of its children in subtree or none in the subtree.

- If a partial derivation tree contains a variable is called a sentential form.
- Left-most derivation or LMD of a sentential form is one in which productions are applied always to the left most variables.

RMD - Replace Right Most

LMD

$$\begin{aligned} S &\Rightarrow A \underline{B} \\ &\Rightarrow AAB \\ &\Rightarrow a AB \\ &\Rightarrow aa B \\ &\Rightarrow aa bB \\ &\Rightarrow aabb \end{aligned}$$

$$\begin{aligned} S &\rightarrow A | AB \\ A &\rightarrow \epsilon | Ab | AA | a \\ B &\rightarrow b | bc | Bc | bB \end{aligned}$$

LMD

②

$$\begin{aligned} S &\Rightarrow A \\ &\Rightarrow AB \\ &\Rightarrow AAB \\ &\Rightarrow aAB \\ &\Rightarrow aaB \\ &\Rightarrow aa bB \\ &\Rightarrow aabb \end{aligned}$$

RMD:

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow A b \\ &\Rightarrow Abb \\ &\Rightarrow AA bb \\ &\Rightarrow Aabb \\ &\Rightarrow aabb \end{aligned}$$

aabb

$$\begin{aligned} S &\rightarrow A | AB \\ A &\rightarrow \epsilon | Ab | AA | a \\ B &\rightarrow b | bc | Bc | bB \end{aligned}$$

Ambiguous Grammar:

A Grammar G is ambiguous if a string belonging to grammar has at least two different:

- 1) Parse Trees
- 2) LMDs
- 3) RMDs

E.g. $S \rightarrow AS \mid E$
 $A \rightarrow A1 \mid 0A1 \mid 01$
String : 0011

To prove grammar is ambiguous,
take a string derive parse string.

$$\begin{aligned} \textcircled{1} \quad S &\Rightarrow AS \\ &\Rightarrow OA1S \\ &\Rightarrow 0011S \\ &\Rightarrow 0011E \\ &\Rightarrow 0011E \end{aligned}$$

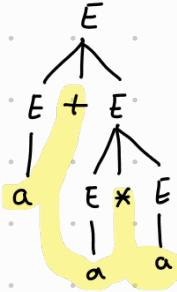
$$\begin{aligned} \textcircled{2} \quad S &\Rightarrow A\$ \\ &\Rightarrow \\ &= \end{aligned}$$

$$q) E \rightarrow E+E \mid E * E \mid (E) \mid a$$

$$\Sigma = \{a, +, *\}$$

Show that language is ambiguous

$a+a*a$



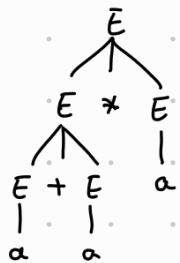
$$E \Rightarrow E+E$$

$$\Rightarrow a+E$$

$$\Rightarrow a+E*E$$

$$\Rightarrow a+a*E$$

$$\Rightarrow a+a*a$$



$$E \Rightarrow E * E$$

$$\Rightarrow E+E * E$$

$$\Rightarrow a+E * E$$

$$\Rightarrow a+a * E$$

$$\Rightarrow a+a+a$$

- Operator Precedence and left associativity:

$$F \rightarrow (E) \mid a$$

$$T \rightarrow T * F \mid F$$

$$\rightarrow E \rightarrow E + T \mid T$$

$$E \Rightarrow E + T$$

$$\Rightarrow E + T * F$$

$$\Rightarrow T + F * F$$

$$\Rightarrow F + F * F$$

$$\Rightarrow a + a * a$$



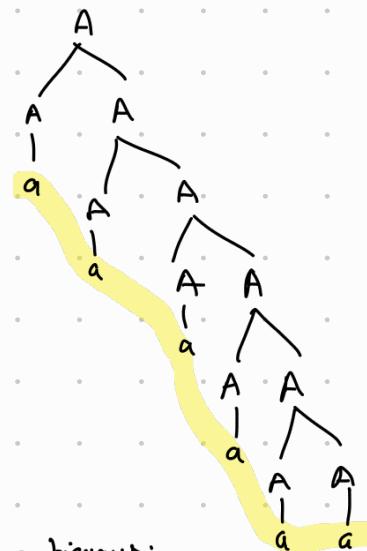
Q) $A \rightarrow AA | a$

String = aaaaaaa

$A \Rightarrow AA$
 $\Rightarrow AAA$
 $\Rightarrow AAAAA$
 $\Rightarrow AAAAAA$
 $\Rightarrow AAAAAAA$
 $\Rightarrow a AAAAA$
 $\Rightarrow aaAAAA$
 $\Rightarrow aaaAAA$
 $\Rightarrow aaaaAA$
 $\Rightarrow aaaaat$
 $\Rightarrow \text{aaaaaa}$
 \Rightarrow



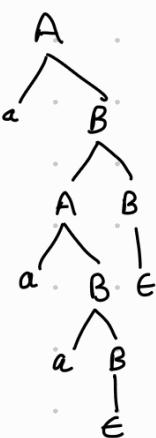
$A \Rightarrow AA$
 $\Rightarrow AAA$
 $\Rightarrow AAAAA$
 $\Rightarrow AAAAAA$
 $\Rightarrow AAAAAAA$
 $\Rightarrow a AAAAA$
 $\Rightarrow aaAAAA$
 $\Rightarrow aaaAAA$
 $\Rightarrow aaaaAA$
 $\Rightarrow aaaaat$
 $\Rightarrow \text{aaaaaa}$
 \Rightarrow



To make above language non ambiguous:

for string = aaa

$A \rightarrow aB$
 $B \rightarrow AB | E$



Search up! \rightarrow Right Recursion Removal) Left factoring / /

$$A \rightarrow \alpha B_1 \mid \alpha B_2$$

left factoring

$$A \rightarrow \alpha B$$

$$B \rightarrow B_1 \mid B_2$$

Inherently Ambiguous Grammar:

Let 'L' be a context free language, if every Context free Grammar 'G' with language 'L' is ambiguous the 'L' is said to be inherently ambiguous.

$$G = \{0^i 1^j 2^k \mid i=j \text{ or } j=k\}$$

G_1 :

$$S \rightarrow AB \mid CD$$

$$A \rightarrow 0A1 \mid 01$$

$$B \rightarrow 2B \mid 2$$

$$C \rightarrow 0C \mid 0$$

$$D \rightarrow 102 \mid 12$$

V, \underline{T}, P, S .

$\in \notin L$

- ① Useless Variables
- ② Unit Production
- ③ $A \rightarrow \epsilon$

① Useless Symbol:

- A symbol ' x ' is useful for a grammar $G = (V, T, P, S)$ if there is

$$S \xrightarrow{*} \alpha X \beta \xrightarrow{*} \omega \in T^*$$

if variable is not involved in any derivation of a language.

$$S \rightarrow aSb | \epsilon | A$$

$$A \rightarrow aA$$

here A is useless as once we go into A , there is no termination of string.

→ A symbol ' x ' is useful if

1. x is a generating variable.

2. x is reachable

} make sure to apply these rules in their order.

$$S \rightarrow AB | a$$

$$A \rightarrow a$$

Generating Variables: $\{S, A\}$

Non Generating $\{B\}$

① $\therefore S \rightarrow AB | a$

will become,

$$\begin{array}{l} S \rightarrow a \\ A \rightarrow a \end{array}$$

$\because AB$ has B which is non generating.

② $\therefore S \rightarrow a$

$\because A$ is unreachable.

Procedure for eliminating non generating symbols:

- 1) Every symbol of T is generating.
- 2) If $a \rightarrow \alpha$ and α is generating, then a is generating.

} calculate
Generating
Variables here.

Non Generating Variables = Variables - Generating Variables

NGV

G.V

Q. $S \rightarrow aS \mid A \mid C$

$$A \rightarrow a$$

$$B \rightarrow aa$$

$$C \rightarrow acb$$

① G.V. = { S, A, B }

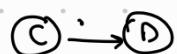
$$\text{NGV} = \{ C \}$$

② $S \rightarrow aS \mid A$

$$A \rightarrow a$$

$$B \rightarrow aa$$

step 2: For production of form $C \rightarrow x\alpha y$ create a dependency graph



Draw dependency graph for all production, if there is an edge not reaching variable x starting from start variable S , then x is not reachable.



$\therefore S \rightarrow aS \mid A$

$$A \rightarrow a$$

is the grammar.

$T \rightarrow aaB \mid abA \mid aaT$

$A \rightarrow aA$

$B \rightarrow ab \mid b$

$C \rightarrow ad$

Step 1:

① Non Generating symbols = {A}

Generating Symbols = {B, C}

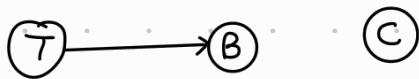
Production 1:

$T \rightarrow aaB \mid aaT$

$B \rightarrow ab \mid b$

$C \rightarrow ad$

Step 2) Dependency Graph



Production 2 :

$T \rightarrow aaB \mid aaT$

$B \rightarrow ab \mid b$

Production of type

' $S \rightarrow \epsilon$ ' are called ϵ production.

IF A in multiple derivations reaches ϵ , then A is nullable variable.

$\hookrightarrow A \xrightarrow{*} \epsilon$

① Find out nullable variables

② Add productions with nullable variables removed.

③ Remove ϵ productions and duplicates.

Algorithm to find nullable variable:

• $V \leftarrow$ set of variables

• $N^0 \leftarrow \{ A \mid A \rightarrow \epsilon \text{ in production and } A \in V \}$

Repeat

$N_i \leftarrow N_{i-1} \cup \{ A \mid A \in V, A \rightarrow \alpha, \alpha \text{ is in } N_i^0 \}$

until $N_i = N_{i-1}$

Q. $S \rightarrow XYX$

$X \rightarrow 0X|\epsilon$

$Y \rightarrow 1Y|\epsilon$

$NV = \{ X, Y, S \}$

$S \rightarrow XYX$

$\rightarrow X Y X | Y X | X Y | Y | X X | X$

$X \rightarrow 0X|0$

$Y \rightarrow 1Y|1$

$S \rightarrow AB$

$A \rightarrow aAA|\epsilon$

$B \rightarrow bBB|\epsilon$

$NV = \{ S, A, B \}$

$S \rightarrow AB|A|B$

$A \rightarrow aAA|aA|a$

$B \rightarrow bBB|bB|b$

Grammar w/o epsilon production and useless symbols then follow sequence

1. Eliminate ϵ -production
2. Eliminate useless symbols.

a)

$$S \rightarrow a | aA | b | c$$

$$A \rightarrow aB | \epsilon$$

$$B \rightarrow aA$$

$$C \rightarrow aCD$$

$$D \rightarrow ddd$$

Nullable variables = $\{A\}$

Production 1

P₁

$$S \rightarrow a | aA | b | c$$

$$A \rightarrow aB$$

$$B \rightarrow aA | a$$

$$C \rightarrow aCD$$

$$D \rightarrow ddd$$

Step 2: Eliminate useless symbols by finding generating variables.

Non-generating Variable $\Rightarrow \{C\}$

Generating Variable $\Rightarrow \{S, A, B, D\}$

P₂:

Remove Non generating

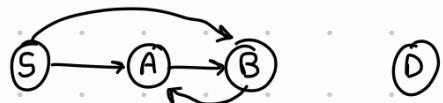
$$S \rightarrow a | aA | B$$

$$A \rightarrow aB$$

$$B \rightarrow aA | a$$

$$D \rightarrow ddd$$

Reachability



P₃

$$S \rightarrow a | aA | B$$

$$A \rightarrow aB$$

$$B \rightarrow aA | a$$

Productions of the form $A \rightarrow B$ where A and B are variables are known as unit productions:

Procedure to remove Unit

1. Add all non-unit productions of P to P_1 .
2. For each unit production of form $A \rightarrow B$ add to P_2 , $A \rightarrow \alpha$ if $B \rightarrow \alpha$ is a non-unit production in P.
3. Delete all unit productions.

If a grammar has E productions, useless symbols and unit productions the sequence of removal is

1. Remove E productions and get G_1 .
2. Remove unit productions from G_1 and get G_2 .
3. Remove useless from G_2 and get G_3 .

Q) $S \rightarrow 0A \mid 1B \mid C$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid A$

$C \rightarrow 01$

P_i:

$S \rightarrow 0A \mid 1B \mid 01$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid A$

Q. Simplify Grammar:

$S \rightarrow a \mid aA \mid B \mid C$

$A \rightarrow aB \mid \epsilon$

$B \rightarrow aA$

$C \rightarrow cCD$

$D \rightarrow dd$

1. Remove Nullable variables: $\{\epsilon\}$

P_i:

$S \rightarrow a \mid aA \mid B \mid C$

$A \rightarrow aB$

$B \rightarrow aA \mid a$

$C \rightarrow cCD$

$D \rightarrow dd$

2. Remove unit productions:

$S \rightarrow a \mid aA \mid \underline{aAa} \mid \{cCD\}$
already there

$A \rightarrow aB$

$B \rightarrow aA \mid a$

$C \rightarrow cCD$

$D \rightarrow dd$

3. Remove useless symbols:

Generating Variables = { A, B, D }

NGV = { C }

$S \rightarrow a | \alpha A$

$A \rightarrow aB$

$B \rightarrow aAa$

$D \rightarrow \alpha\alpha\alpha$

4. Reachability



P₄:

$S \rightarrow a | \alpha A$

$A \rightarrow aB$

$B \rightarrow aAa$

$D \rightarrow \alpha\alpha\alpha$

Chomsky Normal Form:

Every non empty CFL without ϵ has a grammar with productions in CNF, if all productions are of the form,

$S \rightarrow AB$

$A \rightarrow \alpha$

$S, A, B \in V$

$\alpha \in T^+$

Step 1: Simplify grammar by eliminating useless symbols, ϵ -production and unit production.

2: Sim - II- RHS of production

A. Add all productions of the form $A \rightarrow BC / A \rightarrow \alpha$

B. Consider a production $A \rightarrow x_1 x_2 \dots x_n$

e.g. if x_i is a terminal 'a' then you add a production CAI gives A and

Replace x_i with (A_i) in the A production.

e.g. $A \rightarrow aBC$
new variable 'D' is to be introduced,

$D \rightarrow a$
rewrite A as $A \rightarrow DBC$.

Step 3: Consider a production $A \rightarrow x_1 x_2 \dots x_n$ where $n \geq 3$, and all x_i 's are variables. Introduce new productions

$$\begin{aligned} A &\rightarrow x_1 C_1 \\ C_1 &\rightarrow x_2 C_2 \\ C_{n-2} &\rightarrow x_{n-1} x_n \end{aligned}$$

Suppose: $A \rightarrow BCDE$

 $\left. \begin{array}{l} C_1 \rightarrow BC \\ C_2 \rightarrow DE \\ A \rightarrow C_1 C_2 \end{array} \right\}$ Wrong Approach


Do this

$$\begin{aligned} A &\rightarrow BC_1 \\ C_1 &\rightarrow CC_2 \\ C_2 &\rightarrow DE \end{aligned}$$

E.g.

$$\begin{aligned} S &\rightarrow aAD \\ A &\rightarrow aB \mid bAB \\ B &\rightarrow b \\ D &\rightarrow d \end{aligned}$$

$\frac{P_i}{=}$

$$\begin{aligned} B &\rightarrow b \\ D &\rightarrow d \\ C_a &\rightarrow a \\ S &\rightarrow C_a AD \\ &\quad \downarrow \\ S &\rightarrow C_a C_1 \\ C_1 &\rightarrow AD \\ A &\rightarrow C_a B \end{aligned}$$

$A \rightarrow bAB$

$A \rightarrow BAB$

\downarrow

/

/

$A \rightarrow BC_2$

$C_2 \rightarrow AB$

Grammar:

{ A, B, C₁, C₂, D, S }

{ a, b, d }

P,

S }

- 1) To reach a sentential form of length 'n' you require 'n-1' steps. To reach a string of terminals of length 'n' you need 2n-1 derivation.
- 2) Grammar in CNF is not ambiguous.

Push Down Automata (PDA)

There are 2 types of PDA

- 1) NPDA - Non-deterministic PDA
- 2) DPDA - Deterministic PDA

By default all are NPDA.

An NPDA is a 7 tuple, $(\Phi, \Sigma, \Gamma, \delta, q_0, z_0, F)$

stack alphabet

Start stack symbol

stack

$$L = a^n b^n \mid n \geq 0$$

Moves:

- ① Consuming Input
- ② Epsilon Move.

Configuration:

$$(p, a, t) \rightarrow (q, u)$$

Cases:

1) Skip operation.

$$u = t$$

• Consume a terminal but without changing the stack.

$$(p, a, t) = (q, u)$$

2) Pop operation :

$$(p, a, t) = (q, \epsilon)$$

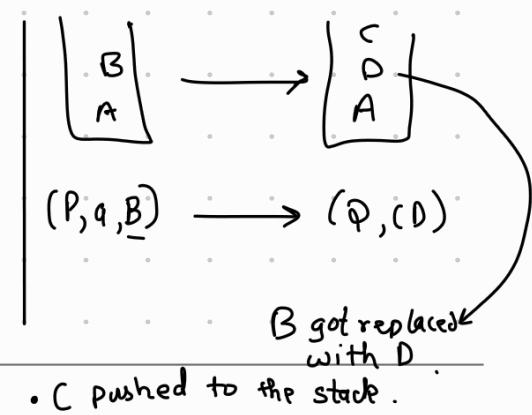
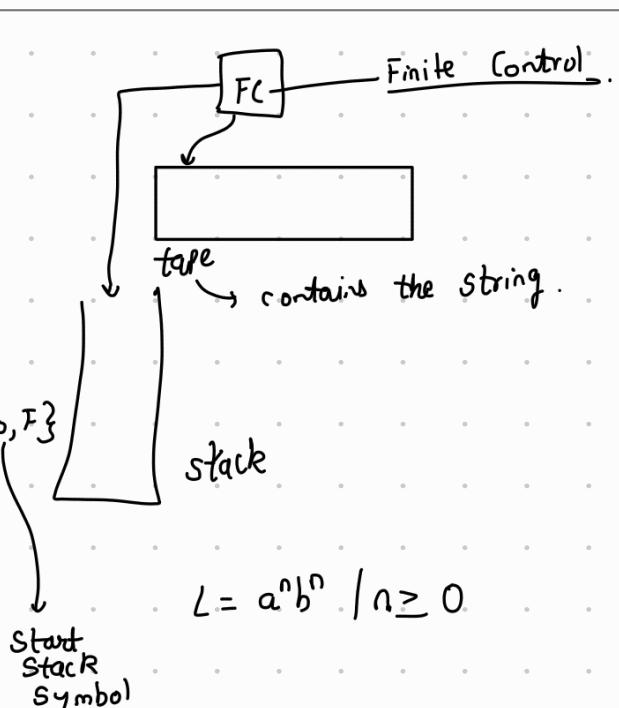
• Consuming input and popping.

3) Push operation

if $u = wx$,

$$(p, a, t) \xrightarrow{} (q, wx)$$

• t is replaced with x , w is pushed into stack.



$\Phi \left\{ a^n b^n \mid n \geq 0 \right\}$

$(q_0, a, z_0) \longrightarrow (q_0, Az_0)$

$(q_0, a, A) \longrightarrow (q_0, AA)$

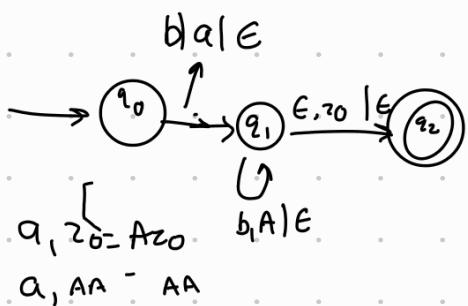
$(q_0, b, A) \longrightarrow (q_1, \epsilon)$

$(q_1, b, A) \longrightarrow (q_1, \epsilon)$

$(q_1, \epsilon, z_0) \longrightarrow (q_2, \epsilon)$

PDA M has 3 states $\{q_0, q_1, q_2\}$

Any other moves \rightarrow PDA stops.



/ /

Instantaneous Description (ID) or Configuration:

- Describes execution status of a PDA, anytime.
- Represent by 8-

q - current state of PDA

ω: Unread part of the input string.

ll - Stack content written as a string with left most symbol at top of stack.

* Id like \overline{s} but for PDA.

Q) Show moves of PDA for input aabb

Input string aabb

(aabb, z_0) $\xleftarrow{ }$

↑
Represents
Move of the
PDA.

$(q_0, aa, bb, z_0) \xleftarrow{} (q_0, abb, A z_0)$

$\vdash (q_0, bb, AA z_0)$

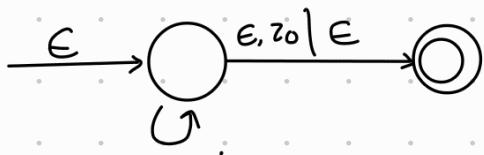
$\vdash (q_1, b, A z_0)$

$\vdash (q_1, \epsilon, z_0)$

$\vdash (q_1, \epsilon, z_0)$

$\vdash (q_2, \epsilon, \epsilon)$

Q) $L = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$
 $L = \{\epsilon, ab, ba, aabb, abab, babab, \dots\}$



$a, z_0 \mid A z_0$

$b, z_0 \mid B z_0$

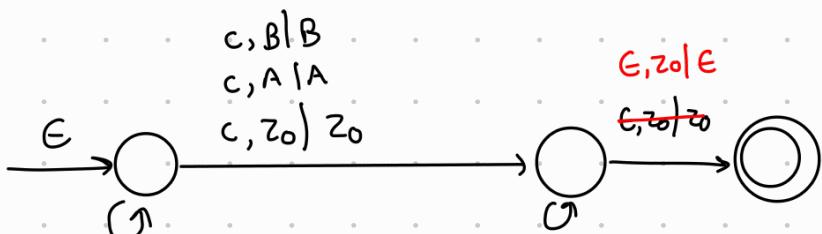
Mq $a, A \mid AA$

$b, B \mid BB$

$a, B \mid \epsilon$

$b, A \mid \epsilon$

Q) $L = \{w \in \{a, b\}^* \mid w c w^R\}$



$a, z_0 \mid A z_0$

$a, A \mid AA$

$b, z_0 \mid B z_0$

$b, B \mid BB$

$a, B \mid AB$

$b, A \mid BA$

} it may happen
that you get

$a \mid \text{top of } b$

$\epsilon, z_0 \mid \epsilon$

$a, A \mid \epsilon$

$b, z_0 \mid C$

not acceptable

Acceptance by
empty stack

Ma'am's Sol:



$a, z \mid Az$

$a, A \mid AA$

$a, B \mid AB$

$b, z \mid Bz$

$b, A \mid BA$

$b, B \mid BB$

$b, B \mid \epsilon$

$\epsilon, z \mid z$

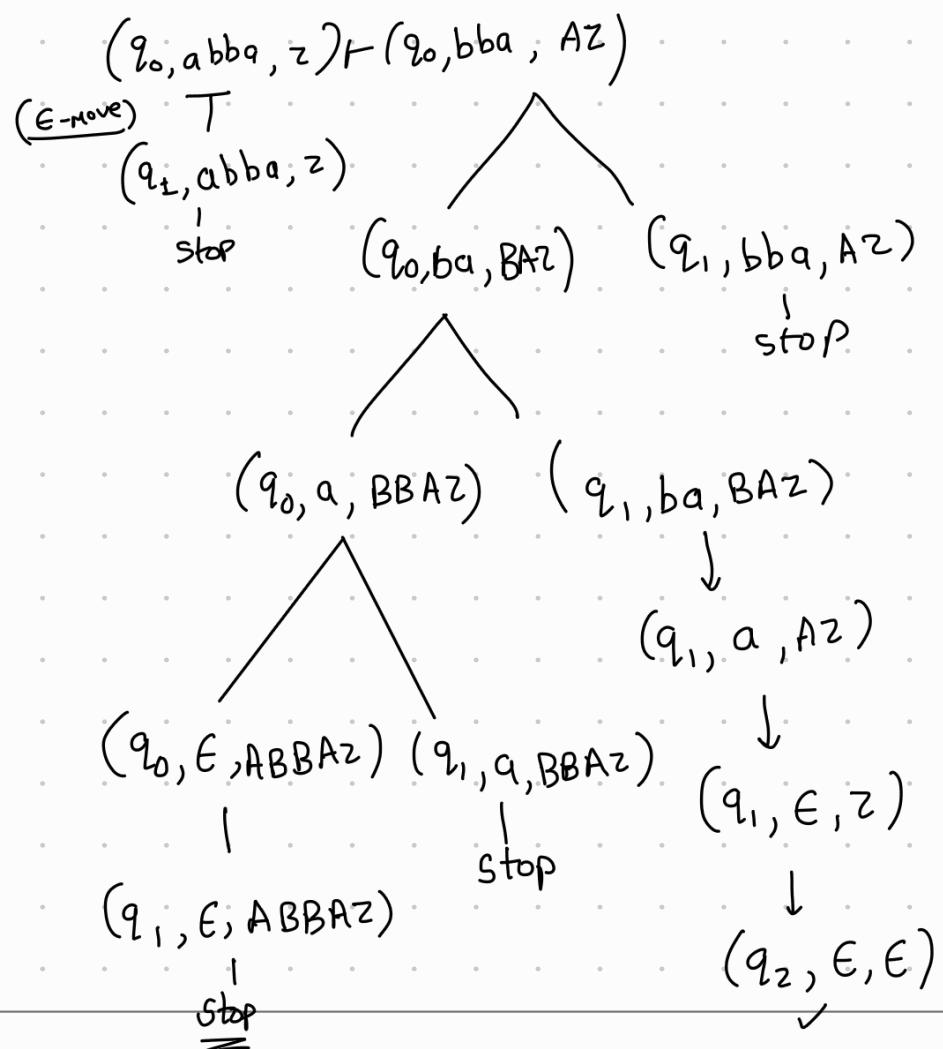
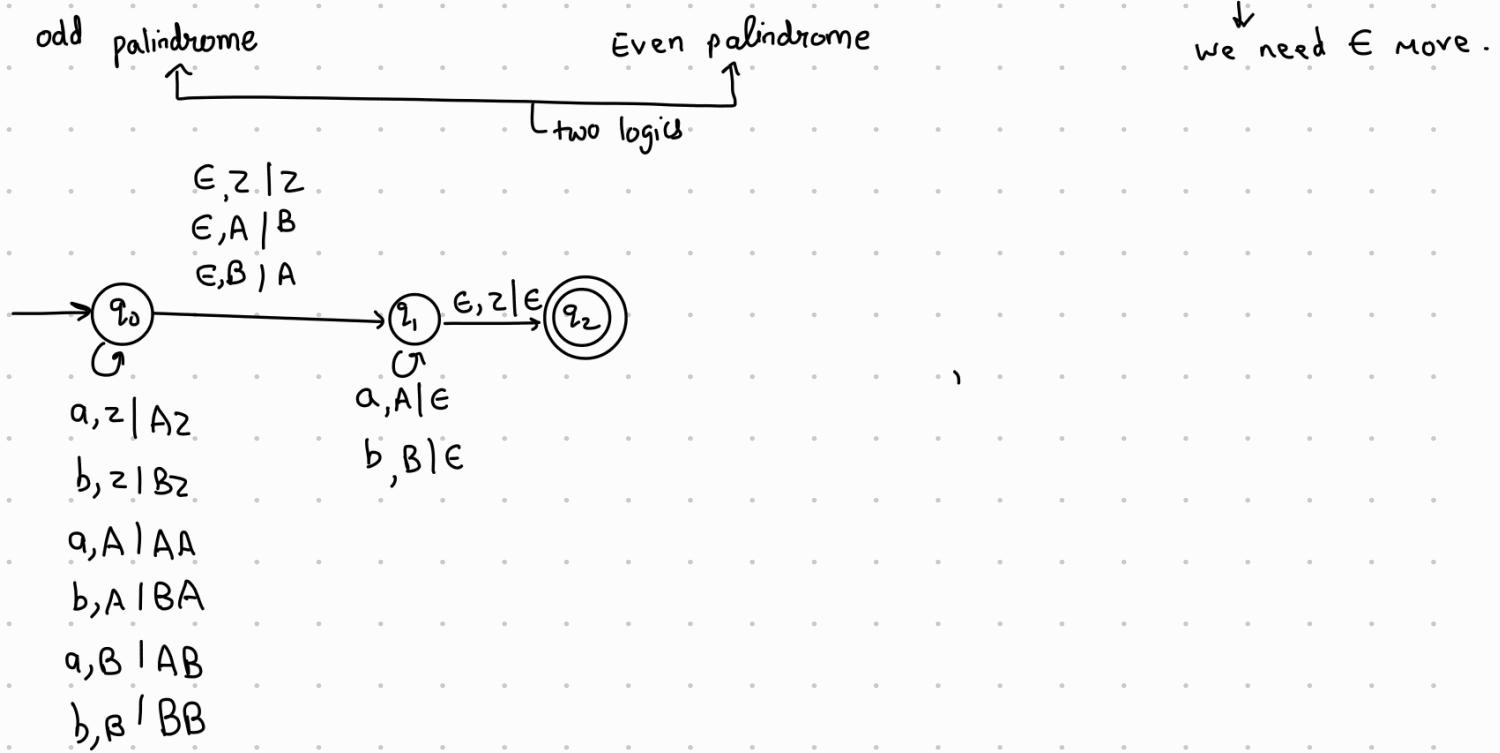
$b, B \mid \epsilon$

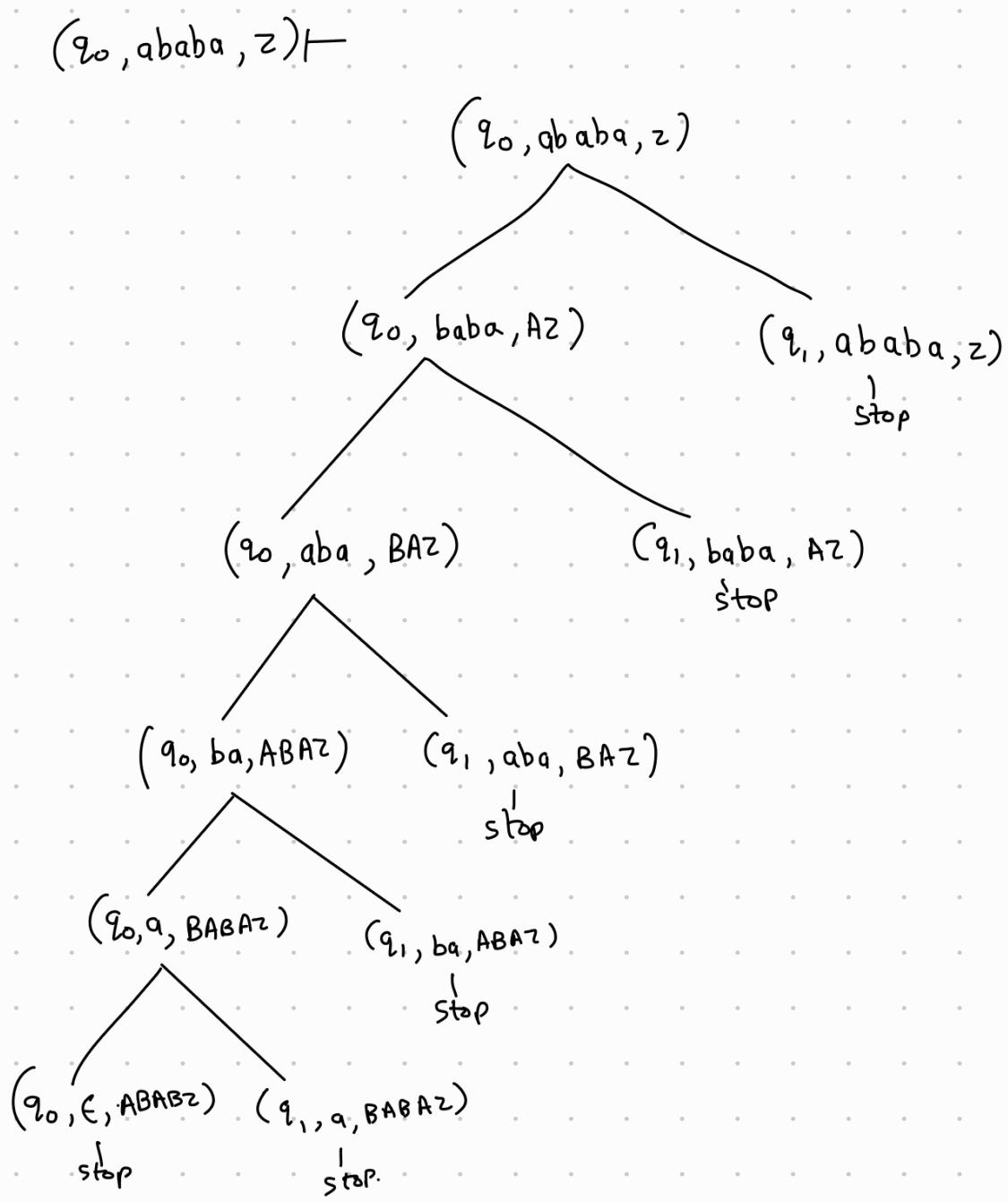
$a, A \mid \epsilon$

Acceptance by
final state.

$$L = \{ \omega\omega^R \mid \omega \in (\alpha+b)^*\}$$

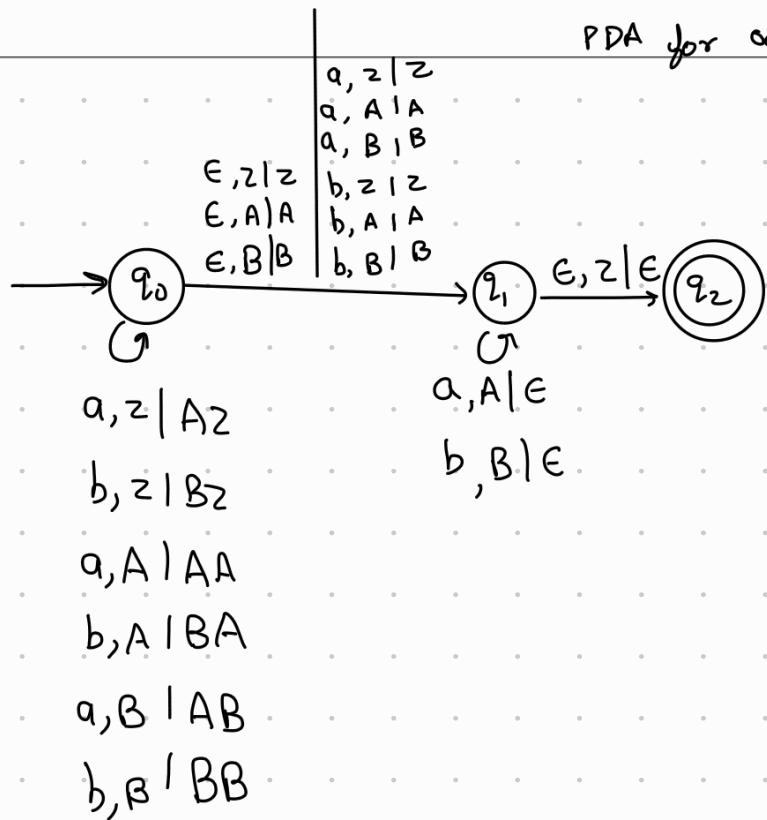
Non-Determinism





Won't work for odd Palindrome.

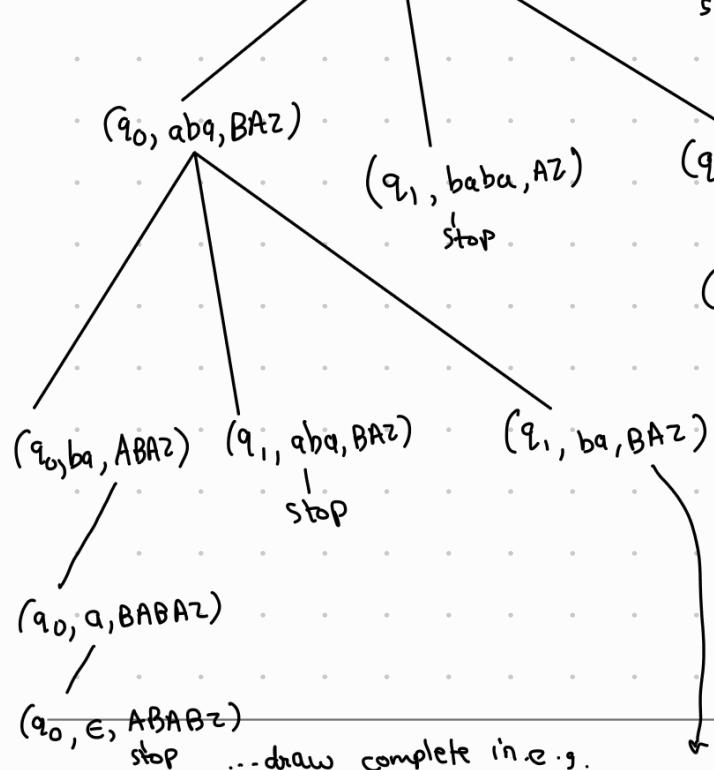
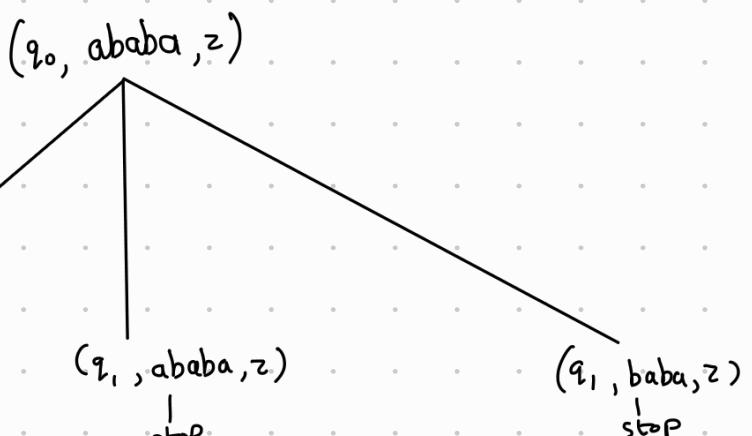
PDA for odd palindrome.



$a, z | A z$
 $b, z | B z$
 $a, A | A A$
 $b, A | B A$
 $a, B | A B$
 $b, B | B B$

$a, A | \epsilon$
 $b, B | \epsilon$

$a, z | z$
 $a, A | A$
 $a, B | B$
 $b, z | z$
 $b, A | A$
 $b, B | B$



\downarrow

$(q_1, ba, B\alpha^2)$

/ /

(q_1, a, α^2)

(q_1, e, z)

(q_2, e, e) accepted.