3. Context Free Grammar (CFG)

definition : CFG represented by 4 tuples

$G = (V, T, P, S)$

$A \to aBT$ (whenever A is there we substi aBT → cd)

$B \to cd$

V - is set of variable or non terminals upper case letters. (variables use for substitution purpose)

T - is set of terminals (or) symbols lower case letters. (terminals used for generating string ↳ id + id * id.)

P - is set of production Rules

production rule form

$A \to \alpha$, where $A \in V$, $\alpha \in (V \cup T)^*$

(we have Non T - Term only Terminal only Non-Terminals No problem because using (or) ∪ → operation)

S → is start symbol

(start symbol always a non-terminal)

ex: (A) → aB → first production

first Production Right side so A become the starting symbol.

B → ca

C → dE

This is the formal definition of the context free grammer.

Ex: $E \to E+E \mid E*E \mid (E) \mid id$

1st Prod, 2nd prod, 3rd, 4th production

in this example V mean non-terminal That is 'E' only

T mean Terminal +, *, ( ) id

P we have 4 Production rules

S starting symbol is first productio right hand sid E →

* our target we want to obtain id+id*id

→ from this we can get id

$E \rightarrow E+E$ ($\because E \rightarrow E+E$)

→ from this we can get id*id.

$E \rightarrow E*E$ $\quad E \rightarrow (E)$ $\quad E \rightarrow id$. $\quad E \rightarrow E+E$ → Production rules

$E \rightarrow id+E$ ($\because E \rightarrow id$)

$E \rightarrow id+E*E$ ($\because E \rightarrow E*E$)

$E \rightarrow id+id*E$ ($\because E \rightarrow id$)

$E \rightarrow id+id*id$

So, in this way we derive the grammer.

* collection of symbols called strings. collection of strings called language

* Here the language which is generated by context free grammer is know as context free language

Derivation ; left-most derivation

→ The process of derivation always starts from start symbol

Start symbol → left the grammer contain several Productions

The first production left hand side Non-Terminal is called as start symbol.

Derivation → means it is the process of applying a derivation sequence of production rules inorder to obtain a string

* To derive a string To apply sequence of production rules

Two Types of derivation

1. left-most derivation
2. Right most derivation

Leftmost - derivation

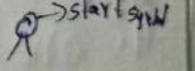
Left most derivation means in each step the left most non-terminal is to be expanded

Right most derivation

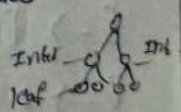
Right most derivation means in each step the right most non-terminal is to be expanded

Derivation tree / Parse tree

* Derivation tree also called as Pharse tree
* Derivation tree means the diagramatic representation of derivation. So here we uses leftmost derivation or right most derivation.
* So if we wan to represent leftmost derivation with the help of a diagram then we uses left most derivation tree
* Right most derivation representation with a diagram then uses Right most derivation tree.
* inorder to obtain derivation tree. we have to follow 3 rules



1. Root node → must be start symbol.
2. Internal node → internal nodes are Non-Terminal nodes
   internal nodes mean Parent nodes which are having childrens.



3. leaf node → leaf nodes are Terminal node (lower case letters)
   leaf nodes mean the node with no children

* by following these 3 rules by construct pharse tree.

$E \to E+E \mid E*E \mid (E) \mid id$ target id + id * id ?

Non terminal → E

Terminals → +, *, (,) → Rtg left Parantheis, id → variable

our target is id + id * id with the help of right most derivation left most derivation.

id + because after id we want +

L.M.D

Production rules	
$E \to E+E$	$E \to E*E$
$E \to (E)$	$E \to id$

$E \to E+E$ (E → E+E)

↳ out of the non terminal only the left most was expanded.

$E \to id+E$ → in each step we have to derive only 1 non-terminal.

→ Here only 1 non-terminal so expand it otherwise left most expand it.

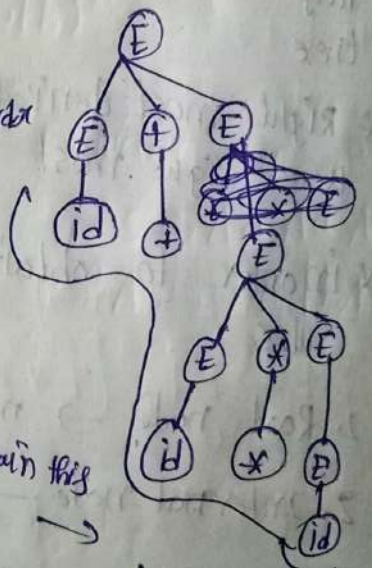
$E \to id+E*E$ $(E \to E*E)$

$E \to id+id*E$ $(E \to id)$

$E \to id+id*id$

left most Phrase tree

* Now we construct Phrase tree



* Now we have to obtain the string in order to obtain the string just we have to take the terminals from left to Right

Right most derivation and its Phrase tree

we obtain this →

$E \to E+E$ → Right Most means we have to expand Right Node only

$E \to E+E*E$ $(E \to E*E)$

$E \to E+E*id$ $(E \to id)$ → right most

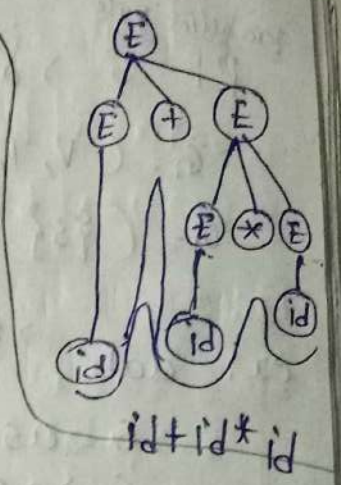
$E \to E+id*id$ $(E \to id)$

$E \to id+id*id$ $(E \to id)$

id + id * id

we have to obtain right most Phase tree.

construction of CFG for a language



1) Any number of a's over the set
$\Sigma = \{a\}$.

$L = \{\epsilon, a, aa, aaa \ldots\}$

$R.E = a^*$

production rules (minimum string)

1: $S \rightarrow aS$ → after a we have 2 a's recursively so placed 'S'.

$S \rightarrow \epsilon$ → any no. a's so 'o' no. of a's also There so ϵ' for ϵ $S \rightarrow \epsilon$ another production rule.

$S \rightarrow aS|\epsilon$ → simply we can write $S \rightarrow aS|\epsilon$ also

* Now let us write context free grammer

$G = (V, T, P, S)$ → production rule denoted by 'P'

$G = (\{S\}, \{a\}, P, S)$ → starting symbol.

ex: aaa

$S \rightarrow aS$

$S \rightarrow aaS$

$S \rightarrow aaaS$

$S \rightarrow aaa\epsilon$ → To complete the process substitute ebsilon

in this way we construct the CFG for any language.

2. Any no. of a's & b's?

$\Sigma = \{a, b\}$

$L = \{\epsilon, a, b, ab, ba, aa, bb, aab \ldots\}$

$R.E = (a+b)^*$

Now we construct CFG

Production rules

P: $S \to aS \mid bS \mid \epsilon$

$G = (V, T, P, S)$

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

ex: aab → suppose taking string aab

$S \to aS$

$S \to aaS$

$S \to aabS$

$S \to aab\epsilon$

3. containing strings of atleast 2 a's

$\Sigma = \{a, b\}$

$L = \{aa, aaa, aaaabb \ldots\}$

R.E = $(a+b)^* a (a+b)^* a (a+b)^*$

C.F.G = (V, T, P, S)

$= (\{S, A\}, \{a, b\}, P, S\}$

Production rules $(a+b)^* = A$

$S \to AaAaA$

$A \to aA \mid bA \mid \epsilon$

4. one occurence of 000.

$\Sigma = \{0, 1\}$

$L = \{000, 00111, 11000 \ldots\}$

R.E $(0+1)^* 000 (0+1)^*$

$S \to A000A$

$A \to 0A \mid 1A \mid \epsilon$

1. ~~one~~ set of strings with equal no. of a's & b's. 1a 1b 2a 2b's

L = {ab, ba, abba, baab..

P: S → aSbS | bSaS | ϵ

a: abba

S → aSbS
→ abSaSbS
→ abϵaϵb
→ abab

like we derive any string in easy way.

6. construction of CFG for (0+1)*

Σ = {0,1}

L = {ϵ, 0, 1, 00, 11, 01, 10 ...}

C.F.G = (V, T, P, S)

= ({S}, {0,1}, P, S)

production rules
P: S → 0S | 1S | ϵ

For example : 110

S → 1S
S → 11S
S → 110S
S → 110ϵ

2. construction of CFG for language which accepts Palindroms over {a,b}.

Σ = {a,b}

reverse of a is a
b is b

L = {ϵ, a, b, aa, bb, aba, baab...}

P: S → aSa | bSb | a | b | ϵ

ex: aba [odd length]
baab [even length]

$S \to aSa \mid bSb \mid \epsilon$ [even]
$S \to aSa \mid bSb \mid a \mid b$ [odd]

Ex: abc

$S \to aSa$
$S \to aba$ ($S \to a \mid b$)

Ex: abba

$S \to aSa$
$S \to abSba$
$S \to ab\epsilon ba$ → even
$S \to abba$

3. construct CFG for $L = \{ wcw^R \mid w \in \{a,b\}^* \}$ (string; w^R means Reverse of string)

ex: aba c aba (c b/w w & w^R)
(w, w^R) → length of string is 7, so odd leng

ex: aca
$S \to aSa$
$S \to aca$

$S \to aSa \mid bSb \mid a \mid b \mid \epsilon$
$S \to aSa \mid bSb \mid \epsilon$
$S \to aSa \mid bSb \mid c$

in question between no 'c' w

4. construct CFN exactly one 'a' over $\{a,b\}$ [ab b even

$\Sigma = \{a,b\}$
$L = \{a, bab, bbba\}$ (No restriction on b) → but 'a' is only 1
R.E: b^*ab^*

C.F.G = (V, T, P, S)
$= (\{S\}, \{ab\}, P, S)$

P: $S \to BaB$
$S \to \epsilon$
$S \to bB$

5. atleast one 'a'

$\Sigma = \{a, b\}$

$L = \{a, aabb, aab, baa \cdots\}$

R.E = $(a+b)^* a (a+b)^*$

$S \rightarrow AaA$

$A \rightarrow aA \mid bA \mid \epsilon$ → 'ε' no of 'a' & 'b's

'a' one 'a' after any b's

6. construct CFG for one lang $L = \{0^n 1^n \mid n \geq 0\}$

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

$\Sigma = \{0, 1\}$

$L = \{\epsilon, 01, 0^2 1^2 \ldots\}$ → '0' 'i'

Production rules

$S \rightarrow 0S1 \mid \epsilon$ → in b/w 0 & 1 we have any numbers so 'S' to substitute

$S \rightarrow 0S1$ → the 'ε' are used to terminate

$S \rightarrow 00S11$

$G = (V, T, P, S)$

$G = (\{S\}, \{0,1\}, P, S)$

7. $L = \{a^n b^n \mid n \geq 1\}$

$\Sigma = \{a, b\}$

$L = \{ab, a^2b^2 \ldots\}$ → aabb

P:

$S \rightarrow aSb \mid ab$

let's we derive $a^n b^n$ of a^2b^2 that is aabb check it.

aabb

$S \rightarrow asb \quad (S \rightarrow ab)$

$S \rightarrow aabb$

$G = (V, T, P, S)$

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

8. $L = \{0^n \mid n \geq 0\}$

$\Sigma = \{0\}$

$L = \{\epsilon, 0, 00, \ldots\}$ — now we derive this

P

$S \rightarrow 0S \mid \epsilon$

Now we derive 00

$S \rightarrow 0S$

$S \rightarrow 00S$

$S \rightarrow 00\epsilon$

$G = (\{S\}, \{0\}, P, S)$

9. $L = \{a^m . b^n \mid m \geq 0, n \geq 0\}$

P

$P: S \rightarrow AB \qquad L = \{\epsilon, ab, a^2b^2 \ldots\}$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

$G = (V, T, P, S)$

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

10. $L = \{a^m.b^n \mid m \geq 0, n \geq 1\}$

$L = \{\epsilon, ab, a^2b^2 \ldots\}$

P

$S \rightarrow AB$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid b$

11. $L = \{a^n b^{2n} \mid n \geq 0\}$

$L = \{0, abb, aabbbb \ldots\}$

$n=1$ $a^1 b^{2^1} = abb$

$n=2$ $a^2 b^{2^2} = aabbbb$

Production rules

$S \rightarrow aSbb \mid \epsilon$

* Now we derive the string aabbbb

$S \rightarrow asbb$ ($S \rightarrow asbb$)

$S \rightarrow a.asbbbb$ → we got this so we terminate it in place of S' substitute ϵ

$S \rightarrow aa\epsilon bbbb$

$S \rightarrow aabbbb$

$G = (V, T, P, S)$

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

12. $L = \{a^{2n}.b^m \mid n \geq 0 \; m \geq 0\}$

$L = \{\epsilon, aab, aaaabb \ldots\}$

$m=1$ $n=1$

$a^{2^1}.b^1$

aab

$m=2$ $n=2$

$a^{2^2}b^2 = aa$

P

$S \rightarrow aaSb \mid \epsilon$

Now we derive aaaabb

$S \rightarrow aaSb$

$S \rightarrow aaaaSbb$

$S \rightarrow aaaa\epsilon bb$

$G = (V, T, P, S)$

$G = (\{S\}$

$G = (V, T, P, S)$

$G = (\{S\}, \{ab\}, P, S)$

13. $0^* | (0+1)^*$

Let us assume $\underset{A^*}{0^*} \mid \underset{B^*}{(0+1)^*}$

P

$S \to A^* \mid B^*$

$A \to 0A \mid \epsilon$

$B \to 0B \mid 1B \mid \epsilon$

14. $L = \{ w \mid n_a(w) = n_b(w) \}$ → string, → no of 'a' are equal to no of 'b's

$L = \{ab, abba, baab \ldots\}$

P

$S \to aSbS \mid bSaS \mid \epsilon$ → minimum string

15. $L = \{a^n b^{n+2} \mid n \geq 0\}$

$L = \{abb, abbb, aabbbb \ldots\}$

P

$S \to aSb \mid bb$ → for termination

Now derive the string aaabbbbb

$S \to asb$

$S \to aasbb$

$S \to aabbbb$

$n=1$ $a^1 b^{1+2}$ $abbb$

$n=0$ $a^0 b^{0+2}$ $= bb$

$n=2$ $a^2 b^{2+2}$ $= aabbbb$

Construction of CFG for a language

16. Any number of a's over one set $\Sigma = \{a\}$

simplification of CFG

Removal of unit productions

$A \longrightarrow B$

n.t → non Terminal.

* The corresponded production treated as unit Product
* so left hand side should contain only one non-Terminal
* Right hand side should contain only one non-terminal

→ let us see how to remove unit productions ~~let me~~

let we have two productions

* Then that first production is produces B
* let the second production produces $B \rightarrow x_1, x_2 \ldots x_n$
* Here the first production is ending with B ($A \rightarrow B$) whereas the second production is starting with B ($B \rightarrow x_1, x_2 \ldots x_n$)
* so if there are two productions like this here x_1 x_2 so on x_n or combination of Non-terminals (or) combination of terminals (or) combinations of both there is No Problem.
* Then we can replace then we now that to eliminate a

A produces B → $A \rightarrow B$, $B \rightarrow x_1, x_2 \ldots x_n$

$A \rightarrow x_1, x_2 \ldots x_n$

* so in this way we can eliminate unit production.
* Now let us solve some examples

Ex2: (left, Right, No uppercase letters)

$S \to 0A \mid 1B \mid C$

$A \to 0S \mid 00$

$B \to 1 \mid A$ → Right hand side contains terminal

$C \to 01$

* first we identify unit productions
* unit productions means left hand side contains only 1 non-terminal
* Right hand side should also contain only 1 non-terminal

unit productions

$S \to C, B \to A$

our target is we have to eliminate these productions.

→ Eliminate $S \to C$ → if we want to eliminat $S \to C$ we have to take another production

$S \to C, C \to 01 \Rightarrow S \to 01$

after eliminating $S \to C$ we get $S \to 01$

→ Eliminate $B \to A$ → to elimat we have to another production starts with A

$B \to A, A \to 0S \mid 00 \Rightarrow B \Rightarrow 0S \mid 00$

$B \to 0S \mid 00$

* After eliminating unit productions the final grammer as is

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

$A \to 0S \mid 00$

$B \to 1 \mid 0S \mid 00$

$C \to 01$

Ex2: S → AB
A → a
B → c | b
C → D
D → E | bc
E → d | Ab

* identify unit productions

unit productions

B → C
C → D
D → E

* Now B → C → D Now replace B → D . but what is D D produce E we don't know what is E value
* we consider last two C → D, D → E replace C → E . But we don't know what is E value.
* so it is Better to go in a Reverse order.

Eliminate D → E, E → d | Ab (take production starts with E) so D produces D → d | Ab | bc

D → E | bc
D → d | Ab | bc
final grammer

Eliminate C → D, D → E (or) bc (it is our's choice) so C → E ⇒ C → d | Ab

Eliminal B → C, C → D so B → D ⇒ B → d | Ab | bc | b

Final grammer

S → AB
A → a
B → d | Ab | bc | b
C → d | Ab
D → d | Ab | bc
E → d | Ab

Simplification of CFG

Removal of Null Production (or) Ebsilon

* if a production in the form $A \to \epsilon$ This is called ebsilon production.

Ex:1 $S \to XYX$

$X \to 0X \mid \epsilon$ → it contain ϵ

$Y \to 1Y \mid \epsilon$ → ϵ

first we identify Null productions

Null Productions

$X \to \epsilon$

$Y \to \epsilon$

our target is These 2 productions are eliminate from the grammer

To Fliminal $X \to \epsilon$, Replace Each occurence of X with ϵ ebsilon

substitute ebsilon

$S \to XYX \mid YX \mid XY \mid 0Y$

both XX substitute ϵ

So These are the productions.

* go to Next 2nd productions.

$X \to 0X \mid 0$

$Y \to 1Y \mid \epsilon$

To eliminate $Y \to \epsilon$ Replace each occurance of y with ϵ

$S \to XYX \mid YX \mid XY \mid 0Y \mid 0$ → in Place of Y substitute ebsilon ϵ

$XX \mid X \mid \epsilon$ → one X already there

$X \to 0X \mid 0$

$Y \to 1Y \mid 1$ → In place of y subs ϵ

To remove ϵ its better

S → XY | XX | XY | Y | XX | X

X → 0X | 0

Y → 1Y | 1

Ex 2 : A → 0B1 | 1B1

B → 0B | 1B | ∈

first we identify ebsilon production

Null Production

B → ∈

in order to eliminate B → ∈ . we have to search where B is in the right hand side and replace each occurance of B with ∈ .

* Now write productions

A → 0B1 | 1B1 | 01 | 11

B → 0B | 1B | 0 | 1 → These are the productions after eliminate ebsilon.

Simplification of CFG

Removal of useless Symbols → Non-terminal

→ our target is remove useless (upper case) or (non-Term)

* if we call a symbol as useless symbol it satisfies two properties

1. Non-Terminal must not be derived from start symbol

ex: S → AB

A → a → B have no production so we called as B is useless symbol.

2. if a Non terminal can't produce a string of Terminals.

Ex1: $S \to XY \mid 0$
$X \to 1$

Y productions are Not available, so Y is useless symbol.
* So all the productions associated with Y are useless

So we Remove $S \to XY$

$S \to 0$ ✓ useful symbol

$X \to 1$ ✓ useful symbol

So after eliminating useful symbols grammer is there 2 productions.

Ex2: $S \to A11B \mid 11A$

$S \to 11B \mid 11$ A and B are useful

$A \to 0$

$B \to BB$

First let us check the Non-Terminals

* 1. step we find A and B are useful symbols.
* 2. step Non terminal can't produce a string of terminal useless
* if Non-terminal can produce a string of terminals so it is useful search it.

$A \to 0$ ✓

$B \to BB$ X useless production. because B Not Produces lower letters or Terminals.

* associated with B also useless

$S \to A11B$ X

$S \to 11B$ X

$S \to 11A$ A means zero

$S \to 110$ ✓

$S \to 11$ ✓

After eliminating useless symbols the final grammer are

$S \to 11A$

$S \to 11$

$A \to 0$

Simplification of CFG

We can simplify or reduces context Free grammar By using 3 steps

3) Elimination of unit productions
2) Elimination of Null productions
1) Elimination of useless symbols.

let us those 3 steps 1 by one.

step 1 : elimination of useless symbols.

$S \to AB \to$ B are useless

$A \to 01$

$S \to A \mid 0C1$

$A \to B \mid 01 \mid 10$

$C \to \epsilon \mid CD$

$S \to A \to$ A means 01 also

$S \to 01$ ✓

$S \to 0C1$

$S \to 0\epsilon 1$

$S \to 01$ ✓

$A \to BX$

$A \to 01$ ✓

$A \to 10$ ✓

$C \to \epsilon$ ✓

$C \to CDX$ → D Not available

After eliminating usless symbols

$S \to A \mid 0C1$

$A \to 01 \mid 10$

$C \to \epsilon$

step 2 : Elimination of Null Production (ϵ)

we have only one production

$C \to \epsilon$ ⟶ we have to search for C in right hand side and in place of C substitute ebsilon.

So we write orginal productions

$S \to A \mid 0C1 \mid 01$

$A \to 01 \mid 10$

Step3 : Elimination of unit productions

if we say that unit production it contains only one terminal in Right hand side and left hand side

ex: $A \to B$

$A \to \underline{B}$ $\underline{B} \to x_1, x_2 \cdots x_n$

The eliminate unit production we have $B \to x_1, x_2 \cdots x_n$

only 1 $A \to x_1, x_2 \cdots x_n$

$S \to A$, $A \to 01 \mid 10$

So we search start with A production

* we can eliminate This $S \to \underline{A}$, $\underline{A} \to 01 \mid 10$ replace A to values

$S \to 01 \mid 10$

Then the final grammer is

$S \to 01 \mid 10 \mid 0c1$

$A \to 01 \mid 10$

* So This is the simplifacation of CFG using the 3 steps.

Ambigious grammer

A grammer is said to be Ambigious Grammer if there exists more than 1 Pharse tree for the given Input string.

* we have some rules while constructing a tree

1. Root Node Must be start symbol
2. Internal Nodes are Non-Terminals
3. Leaft Nodes are terminals.

* if the pharse tree may be either only left most derivation tree (or) right Most derivation pharse tree or both combination of both.

We have 3 scenarios to grammer is Ambigious grammer

→ more than 1 left most Pharse tree
→ more than 1 right most Pharse tree
→ combination of both.

Ex: $E \to E+E \mid E*E \mid id$ P id+id*id

our target is to prove that grammer is ambigious grammer.

* we have 3 scenarios.
* we prove one of those. so we try left most Pharse tree

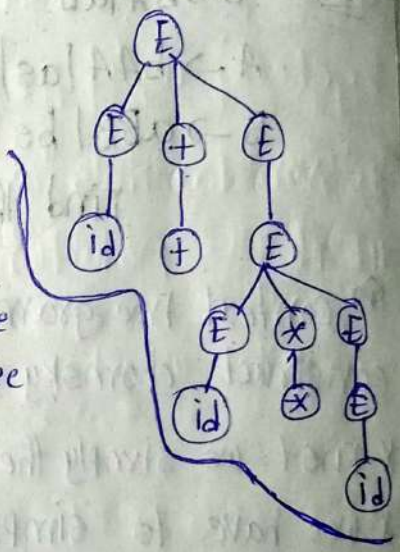
$E \to E+E$ ($E \to id$) < we need multiplication so

$E \to id+E$ ($E \to id*id$)

$E \to id+id*E$

$E \to id+id*id$

Now let us construct Pharse tree for This grammar



* This is the 1st construction tree

Now let us construct another Pharse tree

$E \to E*E$

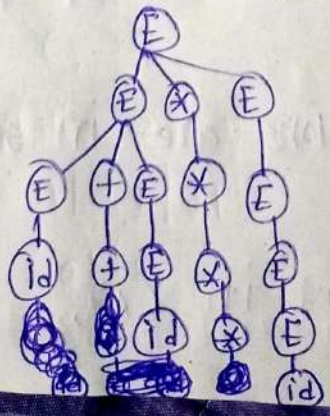
$E \to E+E*E$

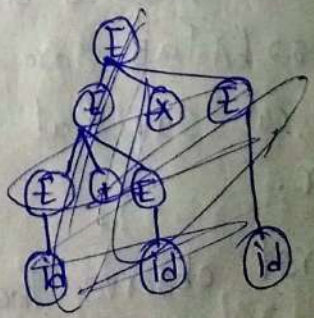
$E \to id+E*E$

$E \to id+id*E$

$E \to id+id*id$

Now we construct the Pharse tree





Elimination of left recursion → a function calling by itself

Elimination of left factoring

Chomsky Normal Form (CNF)

a CFG is said to be in CNF, if productions are of the form $A \to BC$ where A, B, C are variable & 'a' are terminal
$A \to a$

* if left hand side contain one Non-terminal and Right hand side contain a 2 Non-terminal then we can say that the production is chomsky Normal Form (CNF)

Ex: $S \to bA \mid aB$

$A \to bAA \mid as \mid a$

$B \to aBB \mid bs \mid b$

Find the equivalent Grammer in CNF.

Context Free grammer (CFG) is given our Aim is to construct chomsky Normal Form (CNF)

* First we simply the Grammer (or) Reduce the Grammer.
* we have to simply the Grammer by using 3 steps
  1. Eliminating useless symbols
  2. eliminating ebsilon productions
  3. eliminating unit productions
* observe These 3 are prsnt in Grammer or not.

  $S \to bA \mid aB$ are available
* we can say that all productions are in simplified form.
* first observe the productions in CNF.

$A \to a$ → Non-Terminal produces Terminal so it is in CNF.
$B \to b$ ✓

* Nxt what we have to do is, we have to replace each Terminal on right hand side with a Non-Terminal.

$S \to \underline{b}A \mid \underline{a}B$

$S \to bA$ → it is Non-Terminal c or any letter we take
$S \to C_bA$
$C_b \to b$

$S \to aB$ → replaced
$S \to C_aB$
$C_a \to a$

$A \to \underline{b}AA$
$A \to C_bAA$
$C_b \to b$

$A \to aS$
$A \to C_aS$
$C_a \to a$

$A \to a$ → already in CNF form

$B \to aBB$
$B \to C_aBB$
$C_a \to a$

$B \to bS$
$B \to C_bS$
$C_b \to b$

$B \to b$ → already in CNF

Now all are replaced. Next we observe wheather they are in CNF are not.

$S \to C_bA$ ✓ $S \to C_aB$ ✓ $A \to C_bAA$ → Not in CNF form → Right hand side 2 non-Terminals

→ $A \to C_bAA$ ✗
$A \to C_bD_1$ → we assume D_1
$D_1 \to AA$ → Now it is in CNF ✓

→ $B \to C_aBB$
$B \to C_aD_2$
$D_2 \to BB$ ✓

$S \to C_b A$ | $A \to C_b D_1$ | $B \to CaD_2$

$C_b \to b$ | $D_1 \to AA$ | $D_2 \to BB$

$S \to CaB$ | $A \to Cas$ | $B \to Cbs$

$Ca \to a$ | | $Cb \to b$ already written

Greibach Normal Form (GNF)

A context free grammer said to be in GNF

* if the productions or in the form of $A \to a\alpha$ (alpha)
  - A: Non-Terminal
  - a: Term
  - α: list of Non-Terminal

* let us see how to convert CFG to GNF

1. simplify the Grammer → to eliminat useless symbols, unit productions
2. Check if the Grammer is in CNF (chomsky) or not
3. Non terminals are replaced by $A_1, A_2, A_3 \ldots$
4. if the production is in the form

$$A_i \to A_j \beta$$

* Then we check 3 conditions
  - $i < j$ (leave the production as it is)
  - $i > j$ (we perform substitution)
  - $i = j$ (Then we have to diminate left recursion)

* if the grammer is not in ChomskyNF then we need to convert the grammer into CNF

* likewise we have to repeat the process untill all the productions are in GNF form. ($A \to a\alpha$)

$A \to a$ ✓ GNF Form

$A \to aB$ ✓

$A \to aBC$ ✓

Ex: $S \to AA \mid 0$

$A \to SS \mid 1$

* first we check There is a useless symbols or not.

* check whether it is chomsky NF or not.

x it is in CNF form already

x Next we have to replace S with A_1, A with A_2

$A_1 \rightarrow A_2 A_2 \mid 0$

$A_2 \rightarrow A_1 A_1 \mid 1$

x let us consider the 1st production.

$A_1 \rightarrow A_2 A_2 \mid 0$

$i \quad j$

$i=1 \quad j=2 \quad (i<j) \quad (1<2)$ → leave this production as it is

$A_2 \rightarrow A_1 A_1 \mid 1$

$i \quad j$

$i=2 \quad j=1 \quad (2>1)$ → we have to substitute A_1 in A_2

as it is, ex A_1

$A_2 \rightarrow A_2 A_2 A_1 \mid 0A_1 \mid 1$ → 1 as it is

$i=2 \quad j=2$

$(i=j)$ left recursion

* Now check this is in GNF form or not

* we observe this after terminal $A_2 A_2 A \mid 0A_1 \mid 1$ → we have non-terminal.

→ This is not in GNF because we must have some terminal.

How to eliminate recursion

we have a formula

$A \rightarrow A\alpha \mid \beta$ → constant, variables

1st β value = $0A_1$

2nd β value = 1

remember this formula

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \epsilon$

A' value = B_2 assume

$A_2 \rightarrow 0A_1 B_2 \mid 1B_2$

$B_2 \rightarrow A_2 A_1 B_2 \mid \epsilon$

→ we eliminate ϵ. So we write

$A_2 \rightarrow 0A_1 B_2 \mid 1B_2 \mid 0A_1 \mid 1$ → ②

(B_2 → ϵ)

→ after terminal we have 2 non-terminal so it is in GNF

$B_2 \rightarrow A_2 A_1 B_2 \mid A_2 A_1$ → ③

(B_2 → ϵ)

* Now let us substitute A_2 productions on A_1 ($A_1 \to A_2A_1$)

$A_1 \to A_2A_1 \mid 0$ → A_2 write as it is

$A_1 \to 0A_1B_2A_2 \mid 1B_2A_2 \mid$ [illegible] $0A_1A_2 \mid 1A_2 \mid 0$.

$B \to A_2A_1B_2 \mid A_2A_1$ — ③

$B_2 \to 0A_1B_2A_1B_2 \mid 0A_1B_2A_1 \mid$ in place of A_2 substitute all these

$1B_2A_1B_2 \mid 1B_2A_1 \mid 0A_1A_1B_2$

$\mid 0A_1A_2A_1 \mid 1A_1B_2 \mid 1A_2A_1$ → Now these are in GNF

Pumping Lemma for context Free language

Pumping lemma is used to prove a language as not context Free.

→ Let 'L' be context Free language. let 'n' be a integer constant select a string 'z' from 'L' such that $|z| \geq n$.

→ Divide the string into 5 parts uvwxy such that $|vwx| \leq n$ for $i \geq 0$, uv^iwx^iy is in L. $|vx| \geq 1$

Show that $L = \{a^n b^n c^n \mid n \geq 1\}$ is not a C.Free Language?

let 'L' is a CFL

we assume a integer value $n = 3$

$L = \{abc, aabbcc, aaabbbccc \ldots\}$

Here we select a string

$z = aaabbbccc$

$|z| \geq n$ → integer value

aaabbbccc

$9 \geq 3$ ✓

* divide string into 5 parts

aaa bbb ccc

u v wx y

$u = aa$, $v = a$

$w = b$, $x = b$,

$y = bccc$

1st condition

$|vwx| \leq n \Rightarrow |abb| \leq n$

$3 \leq 3$ ✓

2nd condition

$|vx| \geq 1$

$|ab| \geq 1$

$2 \geq 1$ ✓

next condition

$i \geq 0, uv^i wx^i y \in L$

$i = 0 \quad aa\,a^0 b.b^0.bccc$

$\Rightarrow aa.bbccc \notin L$

This string not present in Language $L = \{\}$. we got contradiction result

* so we can say that our language is not context free language.

* suppose if the string present in $L = \{\}$. increment the i

$i = 1$ $\quad a^5 a^5 a^7 \leftarrow$

2) Prove that $L = \{a^p | p$ is a prime number$\}$ is not context free language.

let 'L' be C.F Language

$L = \{aa, aaa, aaaaa, aaaaaaa \cdots\}$

let $n = 5$, $z = aaaaa$

$|z| \geq n \rightarrow |aaaaa| \geq 5$

$5 \geq 5$ ✓

$\underset{u}{a}\ \underset{v}{a}\ \underset{w}{a}\ \underset{x}{a}\ \underset{y}{a}$

$|vwx| \leq n$

$|aaaa| \leq 5$

$3 \leq 5$ ✓

$|vx| \geq 1$

$|aa| \geq 1$

$2 \geq 1$ ✓

$i=0 \Rightarrow uv^iwx^iy$

$aa^0aa^0a \Rightarrow aaa \in L$ ✓

$i=1 \Rightarrow uv^iwx^iy \Rightarrow aa^1aa^1a \Rightarrow aaaaa \in L$ ✓

$i=2 \Rightarrow uv^iwx^iy \Rightarrow aa^2aa^2a \Rightarrow aaaaaaa \in L$ ✓

$i=3 \Rightarrow uv^iwx^iy \Rightarrow aa^3aa^3a \Rightarrow aaaaaaaaaa \notin L$

* now we got a string which is not present in the langu
* so we can say that it is Not context free language

3) Prove that $L = \{a^{n^2} \mid n \geq 1\}$ is not a C.F language

$a^1 a^{2\cdot 2} a^{3^2=9}$

$L = \{aa, aaaa, aaaaaaaaaa \ldots\}$

let $n=2$, $z = aaaa$

$|z| \geq n \Rightarrow |aaaa| \geq n$

$4 \geq 2$ ✓

Divide the string into 5 parts

$\underline{a}\ \underline{a}\ \epsilon\ \underline{a}\ a$

$u\ v\ w\ x\ y$

→ we are taking epsilon

$|vwx| \leq n$

$|a\epsilon x|$

$|aa| \leq n$

$2 \leq 2$ ✓

$|vx| \geq 1$

$|aa| \geq 1$

$2 \geq 1$ ✓

for $i=0 \Rightarrow uv^i wx^i y \Rightarrow aa^0 \epsilon a^0 a \Rightarrow aa \notin L$

* we got a string which is Not present in the language
* so we can say that it is Not a context free language.

4) Prove that $L = \{a^i b^j \mid j = i^2\}$ is not a C.F language

let 'L' be the language

$L = \{ab, aabbbb, aaabbbbbb \ldots\}$

$a^1 b^{1^2} = ab$

$= 2$

$a^2 b^{2^2} = aabbbb$

$= 3$

$a^3 b^{3^3} = aaabbbbbb$

$n=4, \quad z = aabbbb$

$|z| \geq n \Rightarrow |aabbbb| \geq 4$

$6 \geq 4$ ✓

* Divide 'z' into 5 parts such as uvwxy.

aa bbbb

u v w x y

$|vwx| \leq n$

$|abbb| \leq n$

$4 \leq 4$ ✓

$|vx| \geq 1$

$|ab| \geq 1$

$2 \geq 1$ ✓

for $i=0 \Rightarrow uv^i wx^i y \Rightarrow aa^0 bbb^0 b \Rightarrow abbb \notin L$

* so we can say that it is not a context free language

Difference between Regular lang & context Free Language.

Regular Language	context Free language
1) one language which is generated by Regular grammer	1) one language which is generated by context Free gramm
2) Regular language is accepted by finite Automata such as DFA & NFA.	2) context Free language is accepted by Push down Automata (PDA).
3. Regular languages are closed under union, concatenation, Kleene closure, Intersection, complement.	3) CFL's are closed under union, concatenation, Klene closure
4) Regular languages are used in sequential circuit & text edition.	4) C.F.L's are used in compiler

closure Property of context Free language

* context Free closed under 3 properties
  a. union b. concatenation c. Kleene closure
* Not closed under
  a. Intersection → $L_1 \cap L_2$ not C.F.L b. complimentation → $\bar{L_1}$ $\bar{L_2}$ are not a C.F.L

union:- let L_1 & L_2 are 2 C.F.L's then $L_1 \cup L_2$ is also a context Free language.

ex: $L = \{a^n | n \geq 0\}$, $L_2 = \{b^n | n \geq 0\}$

$L_1 = \{\epsilon, a, aa \ldots\}$ $L_2 = \{\epsilon, b, bb \ldots\}$

production

$S_1 \rightarrow aS_1 | \epsilon$

$S_2 \rightarrow bS_2 | \epsilon$

$L_3 = L_1 \cup L_2$

$L_3 = \{\epsilon, a, aa, b, bb, \ldots\}$

$S \rightarrow S_1 | S_2$

Wher L_1 and L_2 are in L_3. So This is called union. The CF language closed under union.

concatenation :- if L_1 & L_2 are 2 context Free languages we can say that $L_1 . L_2$ is also a CFL.

$L_3 = L_1 \cdot L_2$ replacing U to .

$S \rightarrow S_1 \cdot S_2$

Kleene closure :- if L is a context Free lang then L^* is also a CF. Languages

ex: $L = \{a\}$

$L^* = \{\epsilon, a, aa \cdots\}$

Elimination of left recursion

recursion means a function calling by itself. Here Recursion is classified into 2 types.

1. Left Recursion
2. Right Recursion

* if the production rules are in the form

$A \rightarrow A\alpha | B$

* if a production rule is in this form then we can say that This grammer contain some left Recursion.

$A \rightarrow A\alpha | \beta$ → The left most symbol of Right hand side should be equal to the left hand side

* Then we can say that the Production is left Recursion.

```
A()
{
  A();
  α.....
}
```

calling itself ←

But we are not executing α, β statement
The problem is here

mainly There are 2 types of passing techniques available.

1. top door passing technique

* in top door passing technique the redimation strats from root node The process is continued till we get leaf node.

* the derivation process starts from the leaf node the production rules are applied till get the root symbol.

* in order to handle the top door passing. The problem is The top door can not handle the gramers which contain left recursive production

* so we have to eliminat left recursive production. Inorder to elimirate left recussive production we have 2 productions.

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' | \epsilon$

$A \rightarrow A\alpha | \beta$ — $\alpha \rightarrow$ only Terminals; $\beta \rightarrow$ only non-Ter or combination of or epsilon

non-Terminal

$\alpha, \beta \in (V \cup T)^*$

The problem of $A \rightarrow A\alpha | \beta$ only A will be executed the $\alpha\beta$ are not executed.

* Here the problem is top door passing technique can not handle The grammers which contains The left recursive, so we elimirate left Recursive

* we can elimirat left Recursive by Replacing 2 productions

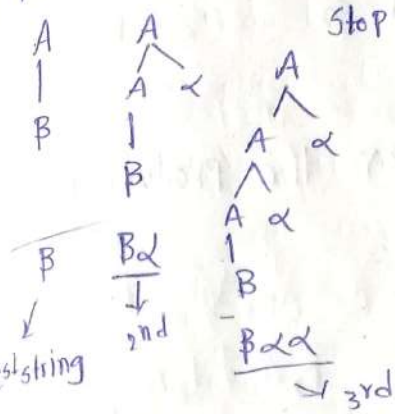
$A \rightarrow \beta A' |$

$A' \rightarrow \alpha A' | \epsilon$

$A \rightarrow \alpha A | \beta$

A()
{ α; ——→ now alpha α also executed
A(); }

Take minimum string as β



Stop the process substitute β

* The language is β^*. zero B Ntg but βϵ, 1 β = β 2 mean βα.

* we know the grammer should not contain * here. That's why

$A \to B\alpha^*$ → Not using * so

$A \to \beta A'$ → in place of A' we take any ϵ/α'

$A \to \epsilon \mid \alpha A'$

Ex1: Eliminate left recursion for the following grammer

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid id$

If the left production means $A \to A\alpha \mid \beta$ This form we replace

$A \to \beta A'$

$A' \to \alpha A' \mid \epsilon$

$E \to E+T \mid T$ → left recursion

$\bar{A}\quad A\ \alpha\ \bar{\beta}$

$E \to TE'$

$E' \to +T\,E' \mid \epsilon$

2nd production

$T \to T*F \mid F$

$\bar{A}\ \alpha\ \bar{\beta}$

$T \to FT'$

$T' \to *F\,T' \mid \epsilon$

3rd production

$F \to (E) / id$ → F and left side Not equal so No recursion.

after eliminating the left recursion the productions are

$E \to TE'$

$E' \to +TE' / \epsilon$

$T \to FT'$

$T' \to *FT' / \epsilon$

$F \to (E) / id$

after eliminating These are the grammer

Ex2 :- $S \to SOS1S / 01$ — left side rightside same symbol so this grammer contain left recursion.

$A \to A\alpha / \beta$

$A \to \beta A'$

$A' \to \alpha A' / \epsilon$

$S \to SOS1S / 01$

$S \to 01S'$

$S' \to 0S1SS' / \epsilon$

after eliminating left recursion. These 2 are Grammer

Ex3 :- $S \to (L) / x$

$L \to L,S / S$

$S \to (L) / x$ → S and right L. So Non recursion

$L \to L,S / S$ → left and Right same symbol. so left Recursion.

$A \quad A\alpha \quad \beta$

$A \to A\alpha \mid \beta$
$A \to \beta A'$
$A' \to \alpha A' \mid \epsilon$

$L \to SL'$
$L' \to ,SL' \mid \epsilon$

After eliminating left Recursion These two are the gramma

* we have one more concept
* if the production rule is in the form of $A \to A\alpha_1 \mid A\alpha_2 \ldots$
$\beta_1 \mid \beta_2 \ldots$
* previous we taken 1 production $A \to A\alpha \mid \beta$
$A \to \beta A'$
$A' \to \alpha A' \mid \epsilon$
* But we have taken multiple productions, if there are multiple production the grammer is replaced with
$A \to \beta_1 A' \mid \beta_2 A' \ldots$
$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \epsilon \ldots$

Ex4: expr → expr + expr + expr * expr | id
(left recursion)
A A α_1 A α_2 β_1

If a grammer like this, we have to replace These productions.
$A \to \beta_1 A' \mid \beta_2 A' \ldots$
$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \epsilon \ldots$
expr → id expr'
expr' → + expr expr' | * expr expr' | ϵ

* After eliminating left recursion. These 2 are grammer

Ex5:- S → Sx | ssb | xs | a
A A α_1 A α_2 β_1 β_2
S → xss' | as'
S' → xs' | ssbs' | ϵ

ex6: S → Aa | b ✗ → no left recursion
A → Ac | Sd | f
A → immediate left recur
Indirect left recursion
S → A → Sd → indirect
So in place S we subst S → Aa | b

$A \rightarrow Ac \mid Aad \mid f$

let order of non terminals first S, A

So substitute S productions is A. because first S' productions are given.

$A \rightarrow Ac \mid sd \mid f$

$A \rightarrow Ac \mid Aad \mid bd \mid f$

$A \quad A\alpha_1 \quad A\alpha_2 \quad \beta_1 \quad \beta_2$

$A \rightarrow bdA' \mid fA'$

$A' \rightarrow cA' \mid adA' \mid \epsilon$

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \beta\alpha_1 \mid \beta\alpha_2 \ldots$

$A \rightarrow \beta_1 A' + \beta_2 A'$

$A' \rightarrow \alpha_1 A' + \alpha_2 A' \mid \epsilon$

In this way we solve this problem.

* And also check first it is indirect left recursion (or) immediate
* after eliminate the grammar is

$S \rightarrow Aa \mid b$

$A \rightarrow bdA' \mid fA'$

$A' \rightarrow cA' \mid adA' \mid \epsilon$

Elimination of left factoring

* we can say that a grammer contain left factoring if it contain production rules in the form of ~~$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$~~

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \gamma_1 \mid \gamma_2 \ldots$

$A \rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2$

$A' \rightarrow \beta_1 \mid \beta_2$ } → To eliminate left factoring we use this 2 productions.

Ex1: $S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b$

→ common term so assume α

$S \rightarrow \underline{iEtS} \mid \underline{iEtSeS} \mid \underline{a}$

$A \quad \alpha \quad \alpha \quad \beta_2 \quad \gamma_1$

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow \epsilon \mid eS$ → we have No β_1

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \ldots \gamma_1 \mid \gamma_2$

$A \rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2$

$A' \rightarrow \beta_1 \mid \beta_2 \ldots$

After elimating the left factoring. The grammer are

$S \rightarrow aiEtsS's' | a$

$S' \rightarrow \epsilon | es$

$E \rightarrow b$

Ex2 :- $A \rightarrow aAB | aA | a$

$B \rightarrow bB | b$

$A \rightarrow \underset{\alpha}{a}\underset{\beta_1}{AB} | \underset{\alpha}{a}\underset{\beta_2}{A} | \underset{\alpha}{a}$ → common term 'a' so assume 'α'

$A \rightarrow aAB | aA | a$

$A \rightarrow aA' |$

$A' \rightarrow AB | A | \epsilon$

Second production

$B \rightarrow \underset{\alpha}{b}\underset{\beta_1}{B} | \underset{\alpha}{b}$ → common term 'b' of 'α'

$B \rightarrow bB'$

$B' \rightarrow B | \epsilon$

After eliminating. These 2 are resultad grammer. The grammer are

$A \rightarrow aA'$

$A' \rightarrow AB | A | \epsilon$

$B \rightarrow bB'$

$B' \rightarrow B | \epsilon$

Ex3 :- $X \rightarrow X+X | X*X | D$

$D \rightarrow 1 | 2 | 3$

$\underset{A}{X} \rightarrow \underset{\alpha}{X}\underset{\beta_1}{+X} | \underset{\alpha}{X}\underset{\beta_2}{*X} | \underset{\gamma_1}{D}$

$X \to Xx' \mid D$
$X' \to x \mid *x \mid$ }

$D \to 1 \mid 2 \mid 3$ → No left factoring

After eliminating the resultant grammar is,

$X \to Xx' \mid D$
$X' \to x \mid *x$

Exp: $E \to T+E \mid T$
$T \to int \mid int * T \mid (E)$

$\underset{A}{E} \to \underset{\alpha}{T} + \underset{\beta_1}{E} \mid \underset{\alpha}{T}$

$E \to TE'$
$E' \to E \mid \epsilon$ }

2nd product

$\underset{A}{T} \to \underset{\alpha}{int} \mid \underset{\alpha}{int} * \underset{\beta_2}{T} \mid \underset{\gamma_1}{(E)}$ → int as α

$T \to intT' \mid (E)$
$T' \to \epsilon \mid *T$ }

After eliminating the resultant gramer is.

$E \to TE'$
$E' \to E \mid \epsilon$
$T \to intT' \mid (E)$
$T' \to \epsilon \mid *T$

...

Push down Automata (PDA)

PDA is very useful inorder to overcome the drawback of finite Automata.

* we know that finite Automat recognizes Regular languages.
* inorder to recognize context free languages we use push down Automata.

F.A → reg lang

P.D.A → C.F.G

* push down Automata is a machine just like finite Automata it is mainly useful inorder to accept or recognizes context free language.

what is the problem with finite Automata?

* finite Automata has a finite amount of memory so limited Amount of memory
* if you want to store smaller languages then enough with Finite Automata
* But if you want to store some complex languages then it is not possible with finite automata.

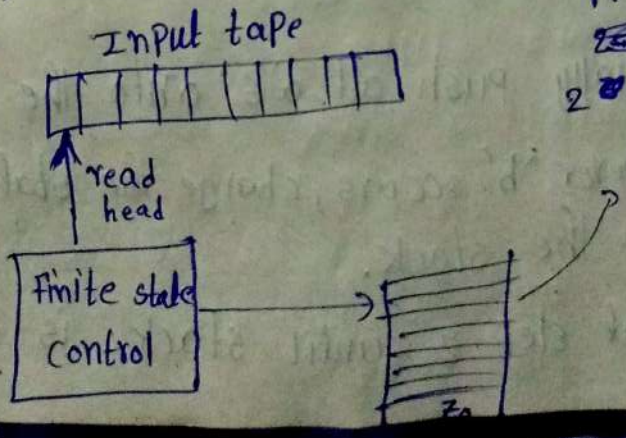
ex: $L = \{a^n b^n | n \geq 1\}$ → These type are not possible to store F.A

This example have comparisons no.of a's no.of b's so complex so we go to PDA.

* Model of push down Automata.

PDA contains 3 components

1. input tape
2. Finite state control.
3. stack



Input tape is mainly used for storing the input string

* where each cell can store 1 symbol at a time → a a b b
* in Finite state control all the states are available.

a b b b
↑ read
a b b b
↑ read head increments to read

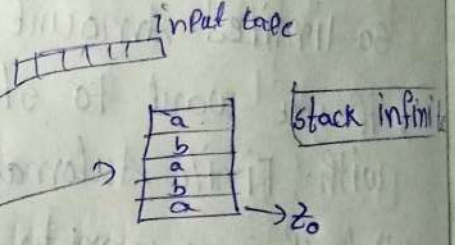
* by apply the input state ex: $\delta(q_0, a) \rightarrow q_1$ but here we have addition stack also
* stack is a data structure for storing the elements (or) symbols
* in stack push and pop operation are there to store and retrieve

Definition

* PDA is defined by 7 tuples

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

(δ → tow, delta)



where
- Q is Finite set of state
- Σ is an input alphabet
- Γ is an stack alphabet
- q_0 is initial state
- z_0 is starting or stop symbol of stack
- F is set of final state
- δ is transition function which requires from $Q \times (\Sigma \cup \{\epsilon\})$

Construct PDA for one language $L = \{a^n b^n / n \geq 1\}$

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

Step 1: Initially push all a's onto the stack

Step 2: Whenever 'b' occurs, change the state & pop 'a' from the stack.

Step 3: Repeat step 2 until stack is empty

$L = \{ab, aabb, aaabbb \ldots\}$

a	a	a	b	b	b	ϵ

$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, a, a) = (q_0, aaz_0)$

$\delta(q_0, a, a) = (q_0, aaaz_0)$

$\delta(q_0, b, a) = (q_1, \epsilon)$ → whenever b occurs, perform pop operation and state is changed to q_1 and ϵ represent pop

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$ → already there no need write but i' written

$\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$

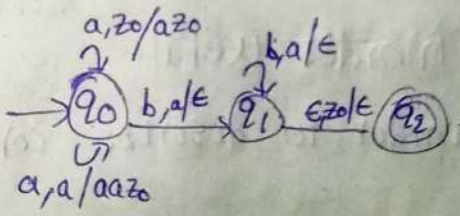
z_0 → remaining top of stack

input string is completly processed then we have to change the state

→ stack is empty

* when the stack is empty the string is accepted by the finite Automata

So, There are the transition function



So, This the diagram. first we write transition func then draw this

→ let the input string is aabb

$(q_0, aabb, z_0) = (q_0, abb, az_0)$

$\vdash (q_0, bb, aaz_0)$

$\vdash (q_1, b, az_0)$

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

$\vdash (q_2, \epsilon)$

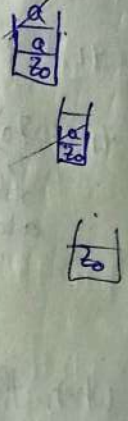
$m = (\{q_0, q_1, q_2\}, \{a, b\}, \{z_0, a\}, \delta, q_0, z_0, q_2)$

graphical Notation for PDA

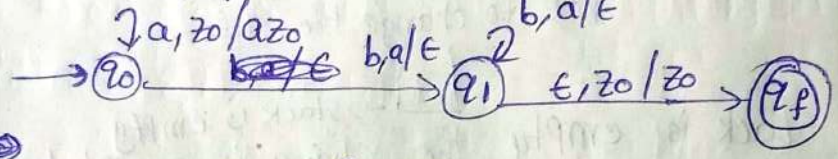
Draw graphical Notation for follows PDA transitions

1. $\delta(q_0, a, z_0) = (q_0, a z_0)$
2. $\delta(q_0, a, a) = (q_0, aa)$
3. $\delta(q_0, b, a) = (q_1, \epsilon)$
4. $\delta(q_1, b, a) = (q_1, \epsilon)$
5. $\delta(q_1, \epsilon, z_0) = (q_f, z_0)$

* graphical Notation is Ntg but transition diag only



PDA of $m = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$



In this we draw the diagram.

Instantaneous description of PDA

~~it is mainly describe inorder useful~~ useful

* it is mainly useful inorder to describe configuration of PDA.
* Instantaneous description (ID) is defined by a triple (Q, w, r)

where $Q \rightarrow$ means current state

$w \rightarrow$ means string

$r \rightarrow$ content of stack

Ex: $L = \{a^n b^n \mid n \geq 1\}$ ab aabb

$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, a, a) = (q_0, aaz_0)$

$\delta(q_0, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$? question given

* Now taking by these transition functions. let us work with ID now. n=1 ab, n=2 aabb

Here string ends with ϵ

$(q_0, aabb, z_0)$ → fallen T → reversing the string.

$\vdash (q_0, abb, az_0)$ → it specifig that we are performing single move

$\vdash (q_0, bb, aaz_0)$

$\vdash (q_1, b, az_0)$ (q_1, ϵ) → pop so aaz_0

$\vdash (q_1, \epsilon, z_0)$ → pop operation performed so z_0 also poped

$\vdash (q_2, \epsilon)$

$q_0 \; a \; z_0 = (q_0, az_0)$

$q_0 \; a \; a = (q_0, aaz_0)$

$q_0, b, a = (q_1, \epsilon)$

So we eliminate the stack we got (q_2, ϵ)

→ q_2 is ntg but final state.

→ So our string aabb is accepted by push down Automata. why because we got the final state

* So In this way we define ID. instantaneous description at a given moment so at this moment abb is id, bb is id

* So this is about Instaneous description (ID) of PDA.

ACCEP

Acceptance of PDA

We can say that a language is accepted by PDA in two ways

1. Acceptance by Final state.
2. Acceptance by empty stack

* After processing entire string if we get final state then we can say that the corresponding language is accepted by PDA.
* likewise after processing the entire input string if the stack is empty then we can say that the corresponding language is accepted by push down Automata
* we know that a PDA is represented by seven Tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Q represend set of states; Σ input alphabet; Γ represent stack alphabet; δ Transition function; q_0 initial state; Z_0 or T_{a1}; F topmost symbol is stack

Ex: $L = \{a^n b^n \mid n \geq 1\}$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$ (input symbol, stack topmost symbol, next state, stack containing)

$\delta(q_0, a, a) = (q_0, aaZ_0)$ (current state)

$\delta(q_0, b, a) = (q_1, \epsilon)$ → first b only we need to change state.

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$ (or) (q_2, ϵ) → Here Z_0 means not going to change stack; pop the stack we get ϵ

for the acceptance of Final state we need to use (q_2, Z_0)

for the acceptance of empty stack need use (q_2, ϵ)

take string aabb

$(q_0, aabb, Z_0) \vdash (q_0, abb, aZ_0)$

$\vdash (q_0, bb, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

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

$\vdash (q_2, z_0)$

* we process the entire string get the final string q_2,
* so we can say that this language is accepted by PDA.
* after processing entire string it reaches to the final state it is accepted by PDA.
* so we can write as $(q, w, z_0) \vdash^{*}$ (after many transitions) (P, ϵ, γ) → we getting this
  - P → final state, $P \in F$
  - $\gamma \in \Gamma^{*}$ → stack symbols

* Now let us see 2nd Approach acceptance by empty stack

Empty stack

$(q_0, aabb, z_0) \vdash (q_0, abb, az_0)$

$\vdash (q_0, bb, aaz_0)$

$\vdash (q_1, b, az_0)$

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

$\vdash (q_1, \epsilon)$ ⟶ we use this the stack is empty

Here after processing the entire string the stack is empty so we can say that this lang is accepted by empty stack.

* So now let us the formula

$(q, w, z_0) \vdash^{*}$ (many transition) (P, ϵ, γ)

γ → final state

where $P \in Q$

$\gamma \in \epsilon$ → γ should be empty

PDA ex:2

Construct PDA for the language $L = \{a^n b^{2n} \mid n \geq 1\}$

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

$L = \{abb, aabbbb, \ldots\}$

[a|a|b|b|b|b] [ϵ] → no. of strings

write the logic. There are 2 logics are there

logic 1

1. if we read 'a' Then push 2 a's on to the stack
2. if we read 'b' Then perform the POP operation (first POP operation change state q_0 to q_1)

Step 3 :- we have to repeat step 2 until the entire string is completely processed.

logic 2

Step 1 :- if we read a then push a onto the stack

step 2 : Here b's are double in first be don't perform POP and 2nd b perform P

where 3 'b' don't perform P. 4th perform POP

b b a a

* we can write any logic in the program now here I am implementing logic1 one let us take the string aabbb
* we know that input string ends with epsilon ϵ → it reads input string is complete
* based on the input string write the transition functions.

$\delta(q_0, a, z_0) = (q_0, aaz_0)$ → read 'a' pus 2 a's to stack

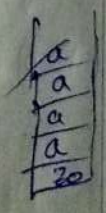
$\delta(q_0, a, a) = (q_0, aaa)$

$\delta(q_0, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

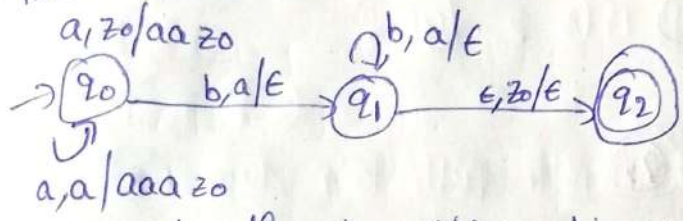


aa bbbb ε

so The input symbol is needed. we are at ebsilon

$\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$ → completly input string is processed so change q_1 to q_2.

* So These are transition function
* now draw the diagram



* So This is the transition diagram.
* Now write 7 PDA Tuples

$m = (\{q_0, q_1, q_2\}, \{a, b\}, \{z_0, a\}, \delta, q_0, z_0, q_2)$

construct PDA for the language $L = \{w c w^R \mid w \in (a+b)^*\}$

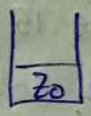
$m = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ → PDA Represent by 7 tuples

$L = \{aba\,C\,aba, abab\,C\,baba \ldots\ldots\}$ (w → aba, $abab$; c → C; w^R → aba, $baba$)

w-is combination of a & b so ab a, bab

$\delta(q_0, a, z_0) = \delta(q_0, a z_0)$

Logic



* we have to Push all the symbols into stack until we get c.
* if symbol 'c' don't perform any operation if symbol is c then keep the stack as it is
* remaining symbols after 'c' (aba C abb) if symbol is 'a' top of the stack is a then Perform Pop operation.
* likewise if symbol is 'b' top of the stack is 'B' then perform the pop operation. so we have to do matching.

$\delta(q_0, a, z_0) = (q_0, a z_0)$

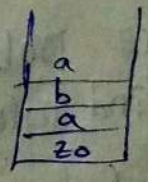
$\delta(q_0, b, z_0) = (q_0, b z_0)$ → Push b into stack

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, a, b) = (q_0, ab)$

$\delta(q_0, b, a) = (q_0, ba)$

$\delta(q_0, b, b) = (q_0, bb)$



$\delta(q_0, c, a) = (q_1, a)$

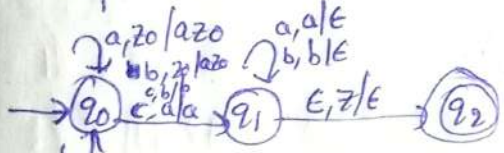
$\delta(q_0, c, b) = (q_1, b)$

$\delta(q_1, a, a) = (q_1, \epsilon)$ ← c=b a

$\delta(q_1, b, b) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$

q0 has 6 transitions



a, a/aa
a, b/ab
b, a/ba
b, b/bb

$m = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b\}, \delta, q_0, z_0, q_2)$

Design PDA for language $L = \{ww^R \mid w \in (a+b)^*\}$

let w is ab w^R is ba . if w is aba w^R is aba only.

if w is ab w^R is ba

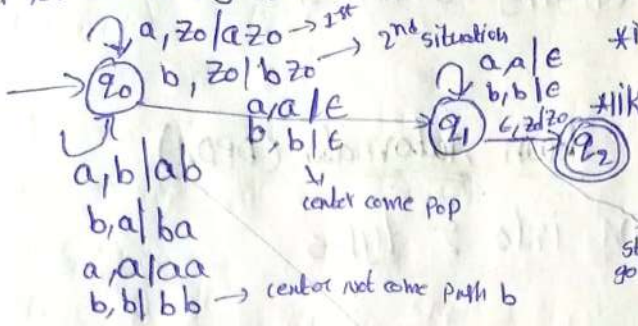
if w is abaaba w^R is (length is 5)

* So, this is an example of R length string.
* where aba is an example for even length string.
* odd length string is very very simple we have to push all the symbols onto the stack until we get c.
* so before_c we push to stack . ex: abac aba → reverse of string

every a we pop top most stack
every b we pop the top most b

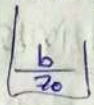
* Here centre marker. centre is specified so it is very easy to solve this problem
* Here problem is centre is not given. Here it is impossible to find centre of string
* for that purpose we have to make two moves. So let us start the problem

let us start the problem. we are initially at q_0. let us design PDA. let us assume.



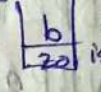
* input a Not problem Push to stack

* likewise input b. stack is empty. So No problem Push 'b' to stack



stack is empty so change go to Final state

* stack topmost b input symbol a middle part is not reached



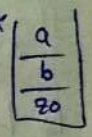
* like another scenario input 'b' stack top most 'a'. So Here also we assume middle part is not reached. so push 'b' on to stack.

* The main scenario input 'a' stack topmost 'a' there is a possibility that center may come may not come.

suppose ex: aaa|bbb → but 2 'as not center Her a after b center

w w^R
ab ba
End of w 'b' starting of w^R 'b'
ab aa ba
w w^R same

* Middle Part is matched only bb or aa. so Not midle Part push 'a' to stack

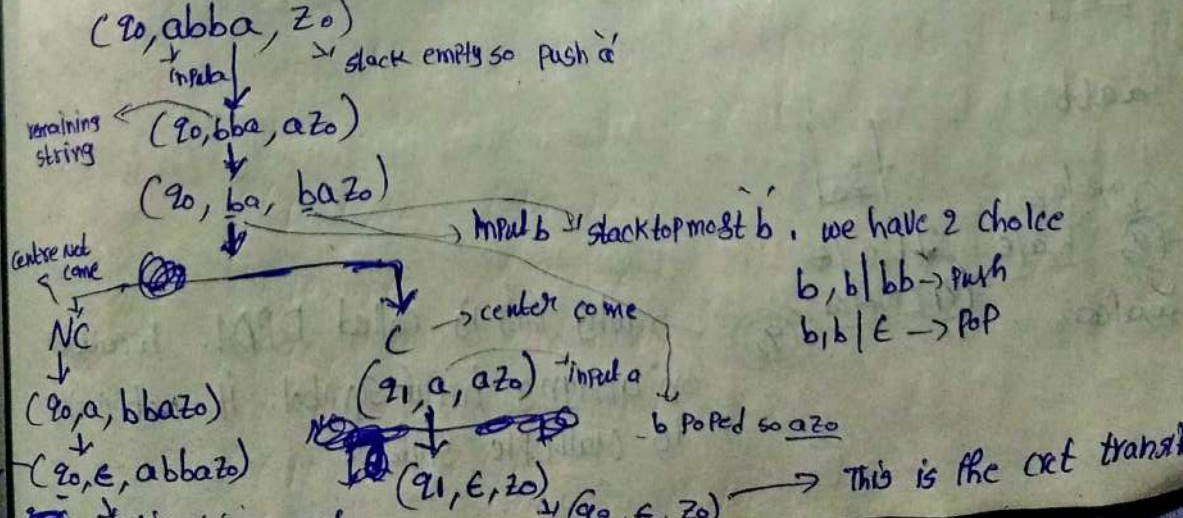


* suppose a a center has not come Then push a onto stack
ex: babab → if input is 'a' stack top most a then perform POP ε operation we need to change state q_0 to q_1. This situation is called Non-Deterministic PDA.

on applying a, a / aa (q0) same state and also. on applying a, a / ε (q1) to move another state is called Non-Deterministic PDA.

* Now check whether our PDA is crct or not?

let us assume the string abba



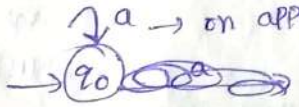
In this way we solve the problem w.r.t
It is NPDA. NPDA applying same input Moves to different
States is called NPDA.

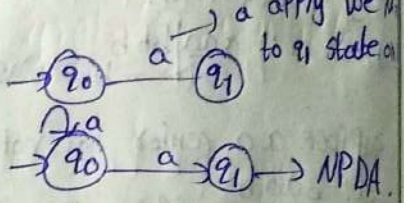
Deterministic Push down Automata (DPDA)

We can classify PDA into 2 types

1) DPDA
2) NDPDA (Non-Deterministic Push down Automata)

* DPDA means if we apply symbol on the current state
Then it produces only 1 state it is called as DPDA.
* let we have state called q_0





1. $L = \{a^n b^n / n \geq 1\}$

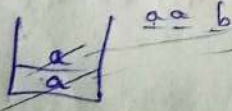
Here n. of a's followed by b's

$n=2 = aabb$

Logic

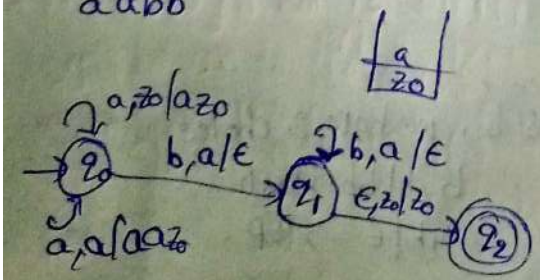
Push a's on to the stack
* if we read b' pop a from stack

ex: aabb



* let us design the PDA for this example

aabb



→ why this is called DPDA. because on applying input symbol. it not move to multiple states.

2. $L = \{a^n b^{2n} / n \geq 1\}$

3. $L = \{wcw^R / w \in (a+b)^*\}$
↓ DPDA

ww^R means NPDA

4. $L = \{w / n_a(w) = n_b(w)\}$

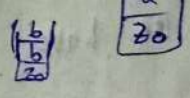
construct PDA for language $L = \{w / \overrightarrow{n_a(w)}^{\text{no. of 'a'}} = n_b(w)\}$
↓ string

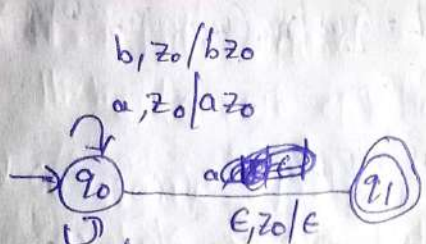
$L = \{\epsilon, ab, aabb, abab, abba, bbaa \ldots\}$ so no. of a's string = no. of b's string

Totally we follow 3 steps.

1. if the stack is empty as well as Input symbol a (or) b Then push it on to the stack.
2. if top of the stack is a and if the input is a so both symbols are same so push it on to the stack
3. if top of the stack is 'a' and iput is 'b'. both are different Then perform the pop Operation.
4. if top of the stack is 'b' if input is b. top of stack input same. so we perform push operation.
* if top of stack 'b' if input is a. Then different so perform POP Operation.

$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, b, z_0) = (q_0, bz_0)$
$\delta(q_0, a, a) = (q_0, aaz_0)$
$\delta(q_0, b, b) = (q_0, bbz_0)$
$\delta(q_0, a, b) = (q_0, \epsilon)$
$\delta(q_0, b, a) = (q_0, \epsilon)$
$\delta(q_0, \epsilon, z_0) = (q_1, \epsilon)$





a, a | aaz0
b, b | bbz0
a, b | ε
b, a | ε

* In this way we construct the pDA.

PDA = ({q0, q1}, {a, b}, {a, b, z0}, δ, q0, z0, q1)

* So this δ is PDA for this problem.

Design PDA for Balanced parenthesis

it mainly we have 3 types of symbols available

( )
{ }
[ ]

* An expression may contain these 3 symbols. Paranthesis, flower brackets, square brackets

* let we have a expression ( ). it is balanced paranthesis (opened last → closed first)

* let we have an expression { (}) — opened last, but we closed first flower brace not balanced if) closed first, balanced

* let we have [{ ( ) }] — we have to close this symbol first

* if the symbol is opening symbol. Then we have to push that symbol on to the stack

left paran ← (, z0 | (z0 — if it is opening symbol and stack empty push the symbol on to stack

ex: { }

{, z0 | {z0

Ex: ({ })

{, z0 | {z0
(, z0 | (z0
{, { | {{
(, { | ({

if ex: (( ))

(, (| ((
{, { | {{

{, { | ∈
{, z0 | {z0
(, z0 | (z0
(, (| ((
{, { | {{
{, (| {(
(, { | ({

q0 —∈, z0 | z0→ q1 (possibilities)

* if the input symbol is right parenthesis } then stack top most symbol is { left parenthesis. pop the stack

must be) right

* suppose input symbol is (left parenthesis. Then stack top most symbol is (left parenthesis Then we need to pop the stack.

we know that ∈ denotes end of string

{, { | ∈
(, (| ∈

Acceptance by empty stack,
Acceptance by final state is our wish

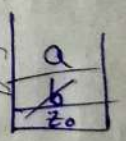
Design PDA for $L = \{ w \mid n_a(w) > n_b(w) \text{ where } w \in (a+b)^* \}$

* no. of a's strings > greater than string b's

$w = \{$ aab, baaba, abaaaa .... $\}$

a> a is greater a's are greater

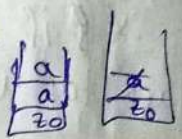
* let us consider the string baaba a



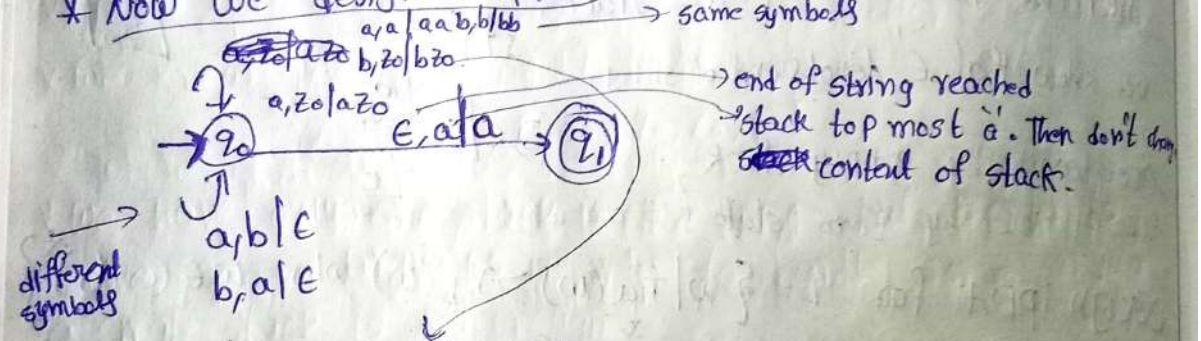
* if the stack is empty the input is a (or) b push them into stack
* if the input symbol is a stack topmost symbol is b. simply we are making the comparision. for every a we have comparision b and no. of a's must be greater.
* input symbol a stack top most b so pop the stack.
* if input symbol b stack top most a so pop the stack.
* if input symbols are same a stack topmost a. push operat

* if input symbol is b stack top most symbol b then also push operation.
* so same symbols means Push. where as different symbols means pop.
* because in our question $n_a > n_b$. so we are making some comparision blw a's and b's.
* Thats why if the symbols are different then pop the stack. if they are same then do the push
* Now we reached end of the string but the stack top most symbol is 'a'

baabaa



* so we can say that no. of as are > no. of b's
* Now we design a PDA



(q0, baaba, Z0)
(q0, aaba, bZ0) → a, b do the pop
(q0, aba, Z0)
(q0, ba, aZ0) → pop
(q0, a, Z0)
(q0, ϵ, aZ0) → we reached end of string
(q1, ϵ, aZ0)

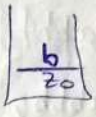
b over next 'a'

* So we say that This PDA accepts a string 'a' no. of accepted greater than b's.

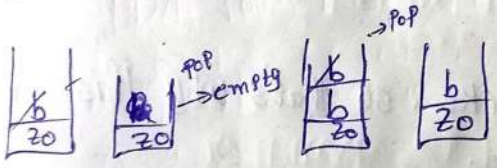
Design PDA for $L = \{w / n_a(w) < n_b(w) \text{ where } w \in (a+b)^*\}$ no. of a's less than no. of b's

$L = \{abb, babba, \ldots\}$

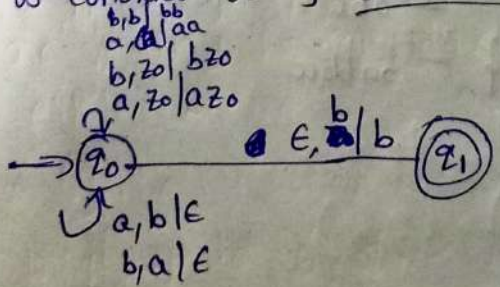
let us consider the string babba



* if the stack is empty. irrespective of symbol Push input a (or) b.
* next input 'a' stack top most 'b'. Pop Here we do the comparision on every a to have 1 b
* if input symbol 'b' stack top most 'a'. symbols are different for every b 1 'a'. Then also perform Pop
* If input 'a' stack top most 'a' both are same symbols Then we need to perform the Push operation.
* if input symbol 'b' stack top most 'b' so same symbols. Then we need to perform push opn.
* after processing entire string 'ε'. stack contains 'b' if the top most symbol prsnt in the 'b'. we can say no. of 'a's are < than no. of b's



* ending string processed 'b' is the top most, so we say that b's are more a's are less.
* Now lets draw the PDA for this
* let us consider string abaabbb a's 3 b's 4



whether our PDA is dcd or not by taking the example abaabbb

(example string)

$(q_0, aba\underline{a}abbb, z_0)$

$(q_0, b\underline{a}abbb, az_0)$

$(q_0, \underline{a}abbb, z_0)$

$(q_0, abbb, \underline{a}z_0)$

$(q_0, \underline{a}bbb, aaz_0)$

(q_0, bb, az_0)

$(q_0, \underline{b}, z_0)$

(q_0, ϵ, bz_0)

(q_1, ϵ, bz_0)

Two stack PDA

* we know PDA having 7 tuples

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

* we have (generated) only one stack in the PDA. but inorder to solve some languages in order to handle some languages a single stack is not enough

* we need to take two stacks. so that's why this concept is named as two stack PDA.

* so let us take the language $L = \{a^n b^n c^n / n \geq 1\}$

* so this example to handle this language one stack is not enough we need to take two stacks

ex: $L = \{a^n b^n c^n / n \geq 1\}$ $\xrightarrow{n=1}$ abc

$L = \{abc, aabbcc, \ldots\}$ $\xrightarrow{n=2}$ aabbcc

* Now let's see logic

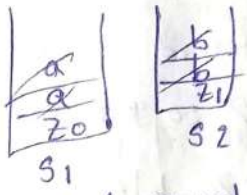
* let the string aabbcc

→ logic very simple

* we have to take two stacks so let the 1st stack is S1

let the second stack is S_2. topmost S_1 is Z_0 also S_2 Z_0.

a a b b c c ∈

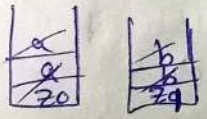


* if input symbol is 'a' Then simply we have to push that 'a' onto the stack.
* if input symbol 'b' stack top most 'a' The pop a on S1 and push to stack 2 S_2.
* if input symbol 'c' stack topMost 'b' The pop the S_2 elements
* Here string is processed 'ε'. The stack1 and stack2 is empty so the string is accepted by the PDA.
* Now let us see logic 2

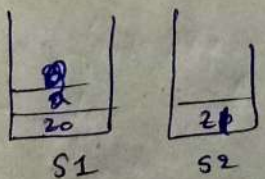
logic 2

* if input symbol 'a' the push 'a' on the stack1 (S_1)
* if input symbol 'b' then push 'b' into stack2 (S_1)
* Nxt if input symbol 'c' then pop 'a' from stack1 as well as b from stack 2.

a a b b c c
↑



* so, The string is processed stack is empty. so string is accepted
* Now let us work with logic 1 it is somewhat Easy.
* let us string aabbcc ∈

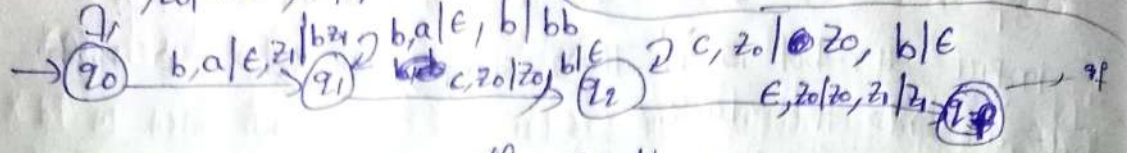


S_1 initial Z_0 $S_2 = Z_1$

* Now let us draw the PDA diagram.

a, a | aZ0 , Z1 | Z1 → No operation Performed

a, Z0 | aZ0 , Z1 | Z1 → moved to stack 2 so go to next state



* In this way we solve the problem
* The above are very suited to Two stack PDA. where A's followed by b's followed by c's / n n /.

convert of CFG to PDA

Here CFG is the input. so the corresponding CFG to find PDA.

* Here CFG mainly focus on the productions. So Here CFG means productions will be given.
* where PDA our focus mainly on transition functions(s) so The corresponding production rules we need to find out the delta (δ) → transition functions
* So, That is our target Here

ex: construct PDA for the grammer

$S \rightarrow AB$

$A \rightarrow 0S \mid 0$

$B \rightarrow 1S \mid 1$

where we construct PDA based on CFG, follow mainly two rules the first rule for non-terminal

1. For non terminal (capital), $A \rightarrow \alpha$

$$\delta(q, \epsilon, A) = (q, \alpha)$$

2. For Terminals → lower case, 0, 1 symbols +, -

$$\delta(q, a, a) = (q, \epsilon)$$

let we have a production in the form of $A \to \alpha$ (A: left, α: right)

Then we write $\delta(q, \epsilon, A) = (q, \alpha)$ ∈

(A: left hand side of production; α: right hand of production)

* for terminal input a

$$\delta(q, a, a) = (q, \epsilon)$$

suppose input t. Then write $\delta(q, t, t) = (q, \epsilon)$

Ex:

$S \to AB$
$A \to 0S \mid 0$
$B \to 1S \mid 1$

$S \to AB$, $\delta(q, \epsilon, S) = (q, AB)$
$A \to 0S$, $\delta(q, \epsilon, A) = (q, 0S)$
$A \to 0$, $\delta(q, \epsilon, A) = (q, 0)$
$B \to 1S$, $\delta(q, \epsilon, B) = (q, 1S)$
$B \to 1$, $\delta(q, \epsilon, B) = (q, 1)$

→ for all the productions writen the transition functions.

$\delta(q, 0, 0) = (q, \epsilon)$

↓ '0' is terminal symbol

$\delta(q, 1, 1) = (q, \epsilon)$

* So This is the Grammar for PDA.

Convert PDA to CFG

$M = (\{q_0, q_1\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$

$\delta(q_0, a, z_0) = (q_0, az_0)$ $\quad \delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, a) = (q_1, a)$ $\quad \delta(q_1, b, a) = (q_1, a)$

$\delta(q_1, a, a) = (q_1, \epsilon)$ $\quad \delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

Here PDA represented with 7 tuples

$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, q_f)$ → in question given it means final state

* Here 6 productions are given. Now we have to construct CFG for this PDA

* let us write the sol, CFG represented with the help of 4 tuples $G = (V, T, P, S)$ → set of variables; → Terminal = {a,b} only

$M = \{q_0, q_1\}$ $\Gamma = \{a, z_0\}$

→ for writing variables what we have to do is ① combine ② with Tow Γ

$V = \{S, [q_0, a, q_0], [q_0, a, q_1], [q_1, a, q_0], [q_1, a, q_1]$ → These are variables

$[q_0, z_0, q_0], [q_0, z_0, q_1], [q_1, z_0, q_0], [q_1, z_0, q_1]\}$

$T = \{a, b\}$

P = let us write productions later. after writing the start symbol

So, let us write the start symbol

$S \rightarrow [q_0 z_0 q_0]$ → start symbol gotted

$S \rightarrow [q_0 z_0 q_1]$ → next production

S → initial state is q_0, initial stack is z

P = 1st production

$\delta(q_0, a, z_0) = (q_0, a z_0)$ → in the stack contain 2 symbols so we have to write 4 symbols

suppose

[ ] → [ ] [ ]
[ ] → [ ] [ ]
[ ] → [ ] [ ]
[ ] → [ ] [ ]

4 symbols

$[q_0 z_0 q_0] \rightarrow a [z_0 a q_0]$ copy it $[q_0, z_0 q_0]$

$[q_0 z_0 q_0] \quad a [q_0 a q_1] \quad [q_1, z_0 q_0]$

$[q_0 z_0 q_1] \quad a [q_0 a z_0] \quad [q_0, z_0 q_1]$

$[q_0 z_0 q_1] \quad a [q_0 a q_1] \quad [q_1, z_0 q_1]$ → possibilities

q_0, q_1 possibilities

in this gets initial a symbol

In this initial z_0 symbol.

* in the later all these doubts will be clarify.

2nd transition

$\delta(q_1, a, a) = (q_0, aa)$ → Here also stack contains 2 symbols.

$[q_0\ a\ q_0] \rightarrow a[q_0\ a\ q_0]\ [q_0\ a\ z_0]$

$[q_0\ a\ q_0] \rightarrow a[q_0\ a\ q_1]\ [q_1\ a\ q_0]$

$[q_0\ a\ q_1] \rightarrow a[q_0\ a\ q_0]\ [q_0\ a\ q_1]$

$[q_0\ a\ q_1] \rightarrow a[q_0\ a\ q_1]\ [q_1\ a\ q_1]$

→ possibilities

3rd transition function

$\delta(q_0, b, a) = (q_1, a)$ → Here the stack contains only 1 symbol

$2^1 = 2$ productions.

$[q_0\ a\ q_0] \rightarrow b[q_1\ a\ q_0]$

$[q_0\ a\ q_1] \rightarrow b[q_1\ a\ q_1]$

4th transition function

$\delta(q_1, b, a) = (q_1, a)$ → stack contain 1 symbol

$[q_1\ a\ q_0] \rightarrow b[q_1\ a\ q_0]$

$[q_1\ a\ q_1] \rightarrow b[q_1\ a\ q_1]$

5th trionsition function

$\delta(q_1, a, a) = (q_1, \epsilon)$ → Here stack contain ϵ, it means we have only 1 Production.

$[q_1\ a\ q_1] \rightarrow a$ → if we have epsilon then we have to write this terminal symbol here

really written

The middle i's the terminal symbol

6th transition function

$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$ → 1 Production

$[q_1\ z_0\ q_1] \rightarrow \epsilon$ → we we have ϵ symbol. then what is Right hand side ϵ.