Unit 3

Context free Languages

- Context free Grammar generate context free languages.
- Type 2 Grammar
- The productions must be of the form $A \longrightarrow Y$ where $A \in N$ (nonterminal) and $Y \in (T \cup N)^*$ (Strings of terminal and non-terminal)
- These languages generated by the grammars are to be recognized by Non-deterministic Pushdown automaton.
- Formal definition

A context free Grammar (CFG), consisting of a finite set of grammar rules, is a quadruple (N, T, P, S) where

N is a set of non-terminal symbols

T is a set of terminals where $N \cap T = NULL$

P is a set of rules, $P : N \longrightarrow (N \cup T)^*$, i,e the left hand side of the production rule P does have any right context or left context.

S is the Start symbol.

1) Construct a context free grammar over a, b that contain set of strings of length two.

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

$L = \{aa, ab, ba, bb\}$

. $S \rightarrow aa \mid ab \mid ba \mid bb$

To generate a string

$S \longrightarrow AA$

$A \longrightarrow a$

$A \longrightarrow b$

Thus the CFG for L is

$G = (V, \Sigma, S, P)$

$V = \{S, A\}$

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

$S = S$

$P : S \rightarrow AA$

$A \rightarrow a \mid b$

2) construct a CFG for the Language

$L = \{a^n , n \geq 1\}$

Soln $\Sigma = \{a\}$

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

$S \rightarrow aS$

$S \rightarrow a$

Thus the CFG for L is

$V = \{S\}$ $\quad P : S \rightarrow aS \mid a$

$\Sigma = \{a\}$

$S \rightarrow S$

3) Construct a CFG for the Language

$L = \{ a^m b^n \mid m, n > 0 \}$

Soln

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

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

To generate some number of a's

$A \longrightarrow aA$

$A \longrightarrow a$

To generate some number of b's

$B \longrightarrow bB$

$B \longrightarrow b$

To generate a complete string

$S \longrightarrow AB$

Thus the CFG for L is

$G = (V, \Sigma, S, P)$

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

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

$S = \{S\}$

$P : S \longrightarrow AB$

$A \longrightarrow aA \mid a$

$B \longrightarrow bB \mid b$

$w = aaabbb$

S	Expanded by $S \rightarrow AB$
AB	Expanded by $A \rightarrow aA$
aAB	Expanded by $A \rightarrow aA$
aaAB	Expanded by $A \rightarrow a$
aaaB	Expanded by $B \rightarrow bB$
aaabB	Expanded by $B \rightarrow bB$
aaabbB	Expanded by $B \rightarrow b$
aaabbb	

4) Construct a CFG over a and b that contain strings beginning with a ending with b.

Soln

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

$L = \{ab, aab, abb, aaab, \ldots\}$

$a(a+b)^* b$

$S \longrightarrow aAb \mid ab$

$A \longrightarrow aA$

$A \longrightarrow bA$

$A \longrightarrow \varepsilon$

Thus the CFG for L is

$G = (V, \Sigma, S, P)$

$V = \{S, A\}$

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

$S = \{S\}$

$P : S \longrightarrow aAb \mid ab$

$A \longrightarrow aA \mid bA \mid \varepsilon$

Derivations

- Grammar generate a string

1. Derivation is a technique to verify whether given string is generated by grammar or not.
2. Derivation begins from start symbol of the Grammar.
3. At each derivation step a variable on head of the production is replaced by the corresponding body of the production rule.

There are two types of derivations.

1. LMD - left most derivation
2. RMD - Right most derivation

1. LMD - Left Most Derivation

At each derivation step a leftmost variable in the sentential form is replaced by corresponding body of the production rule.

2. RMD - Right Most Derivations

At each derivation step a rightmost variable in the Sentential form is replaced by corresponding body of the production rule.

1) Check whether string generated by grammar is valid or not.

i) abbbb

G: $S \rightarrow aAB$
$A \rightarrow bBb$
$B \rightarrow A/\lambda$

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

$T = \{a, b\}$

$S = \{S\}$

P: $S \rightarrow aAB$
$A \rightarrow bBb$
$B \rightarrow A/\lambda$

LMD

$\Rightarrow S$ Expanded by $S \rightarrow aAB$
$\Rightarrow aAB$ Expanded by $A \rightarrow bBb$
$\Rightarrow abBbB$ Expanded by $B \rightarrow A$
$\Rightarrow abAbB$ Expanded by $A \rightarrow bBb$
$\Rightarrow abbBbbB$ Expanded by $B \rightarrow \lambda$
$\Rightarrow abbbbB$ Expanded by $B \rightarrow \lambda$
$\Rightarrow abbbb$

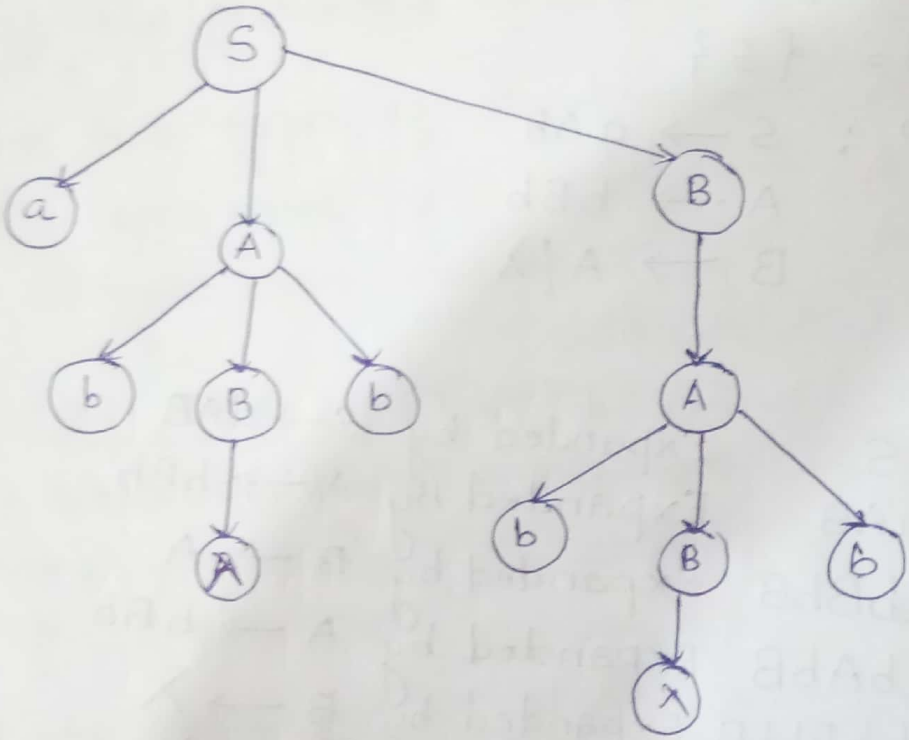
RMD

$\Rightarrow S$ Expanded by $S \rightarrow aAB$
$\Rightarrow aAB$ Expanded by $B \rightarrow \lambda$
$\Rightarrow aA$ Expanded by $A \rightarrow bBb$
$\Rightarrow abBb$ Expanded by $B \rightarrow A$
$\Rightarrow abAb$ Expanded by $A \rightarrow bBb$
$\Rightarrow abbBbb$ Expanded by $B \rightarrow \lambda$
$\Rightarrow abbbb$

Derivation or Yield of tree or Parse Tree

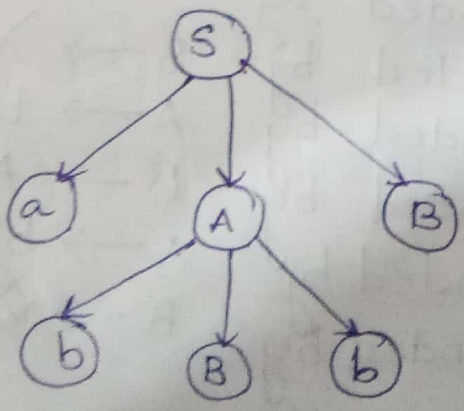
The derivation or the yield of a parse tree is the final string obtained by concatenating the labels of the leaves of the tree from left to right, ignoring the nulls. However, if all the leaves are null, derivation is null.

Derivation Tree (for Prblm NO. 1)



Partial derivation Tree

A partial derivation tree is a subtree of a derivation tree/Parse tree such that either all of its children are in the sub-tree or none of them are in the sub-tree.



If a Partial tree contains the root S, it is called sential form. The above sub-tree is also in Sentential form.

Left most and Rightmost Derivation of a string

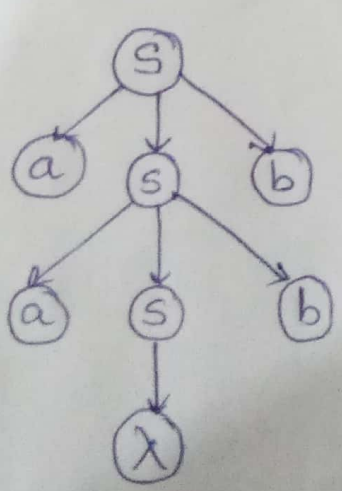
Leftmost derivation - A leftmost derivation is obtained by applying production to the leftmost variable in each step.

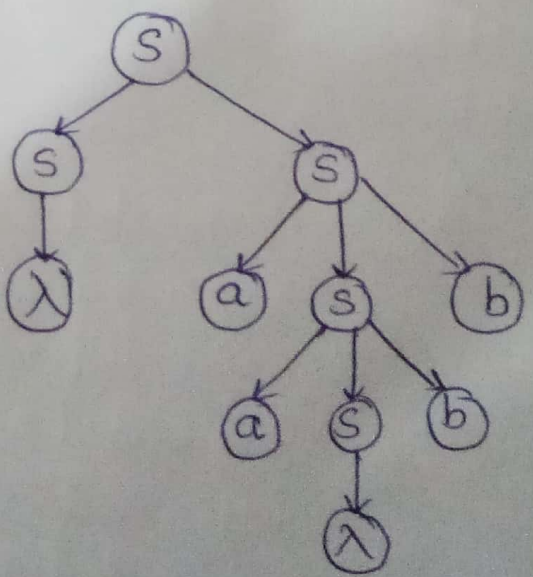
Rightmost derivation - A rightmost derivation is obtained by applying production to the rightmost variable in each step.

Ambiguity in Context free grammars

If a context free grammar G has more than one derivation tree for some string $w \in L(G)$, it is called ambiguous grammar. There exist multiple right most or left most derivations for some string generated from that grammar.

Ex. $S \rightarrow aSb \mid SS \mid \lambda$ is a ambigous grammar.
The string aabb has two derivation trees.





1) check whether string generated by grammar is valid or not.

(i) ((a , a) , a)

G : S ⟶ (L) | a

L ⟶ L, S | S

Soln V = {S, L}

T = {a, , , (,)}

S = {S}

P : S ⟶ (L) | a

L ⟶ L, S | S

LMD

⇒ S Expanded by S → (L)

⇒ (L) Expanded by L → L, S

⇒ (L, S) Expanded by L → S

⇒ (S, S) Expanded by S → (L)

⇒ ((L), S) Expanded by L → L, S

⇒ ((L, S), S) Expanded by L → S

⇒ ((S, S), S) Expanded by S → a

⇒ ((a, S), S) Expanded by S → a

⇒ ((a, a), S) Expanded by S → a

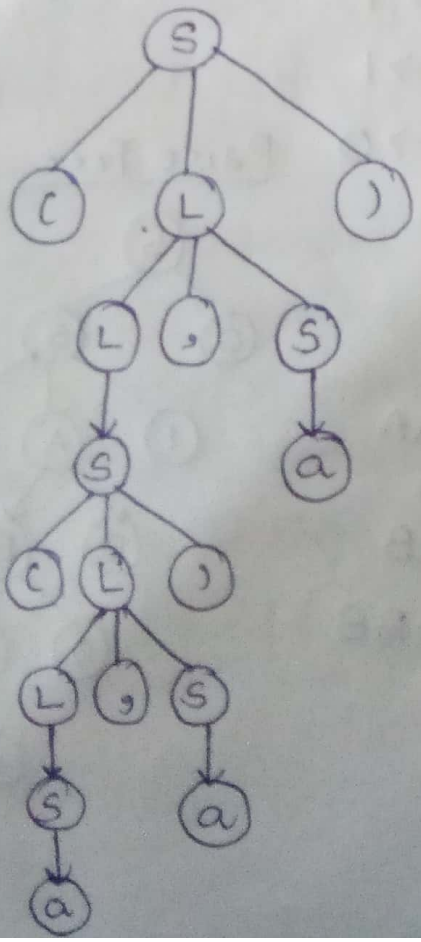
⇒ ((a, a), a)

RMD

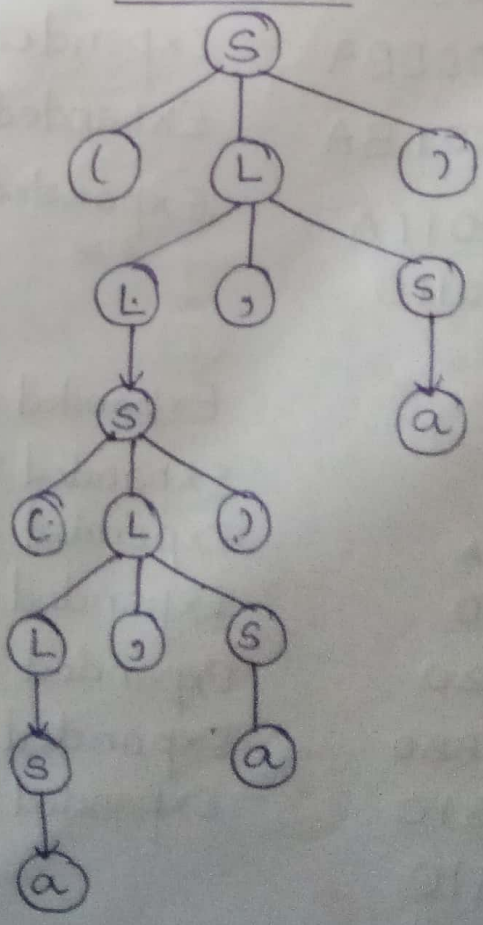
$\Rightarrow S$	
$\Rightarrow (L)$	Expanded by $S \rightarrow (L)$
$\Rightarrow (L,S)$	Expanded by $L \rightarrow L,S$
$\Rightarrow (L,a)$	Expanded by $S \rightarrow a$
$\Rightarrow (S,a)$	Expanded by $L \rightarrow \$$
$\Rightarrow ((L),a)$	Expanded by $S \rightarrow (L)$
$\Rightarrow ((L,S),a)$	Expanded by $L \rightarrow L,S$
$\Rightarrow ((L,a),a)$	Expanded by $S \rightarrow a$
$\Rightarrow ((\$,a),a)$	Expanded by $L \rightarrow S$
$\Rightarrow ((a,a),a)$	Expanded by $S \rightarrow a$

Parse Tree / Derivation Tree

LMD



RMD



2) Construct a string "0100110" from the CFG using LMD and RMD.

$S \rightarrow 0S \mid 1AA$

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

$B \rightarrow 0BB \mid 1$

Soln

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

$T = \{0, 1\}$

$S = S$

$P : S \rightarrow 0S \mid 1AA$

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

$B \rightarrow 0BB \mid 1$

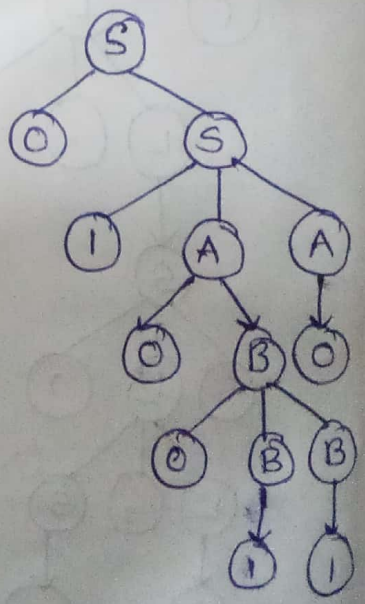
LMD

$\Rightarrow S$	Expanded by $S \rightarrow 0S$
$\Rightarrow 0S$	Expanded by $S \rightarrow 1AA$
$\Rightarrow 01AA$	Expanded by $A \rightarrow 0B$
$\Rightarrow 010BA$	Expanded by $B \rightarrow 0BB$
$\Rightarrow 0100BBA$	Expanded by $B \rightarrow 1$
$\Rightarrow 01001BA$	Expanded by $B \rightarrow 1$
$\Rightarrow 0100110A$	Expanded by $A \rightarrow 0$
$\Rightarrow 0100110$	

RMD

$\Rightarrow S$	Expanded by $S \rightarrow 0S$
$\Rightarrow 0S$	Expanded by $S \rightarrow 1AA$
$\Rightarrow 01AA$	Expanded by $A \rightarrow 0$
$\Rightarrow 01A0$	Expanded by $A \rightarrow 0B$
$\Rightarrow 010B0$	Expanded by $B \rightarrow 0BB$
$\Rightarrow 0100BB0$	Expanded by $B \rightarrow 1$
$\Rightarrow 0100B10$	Expanded by $B \rightarrow 1$
$\Rightarrow 0100110$	

Parse Tree



3) Construct 'id + id * id' from the CFG using LMD and RMD.

$E \longrightarrow E+E$

$E \longrightarrow E*E$

$E \longrightarrow id$

Soln $V = \{E\}$

$T = \{id, +, *\}$

$S = E$

P : $E \longrightarrow E+E$

$E \longrightarrow E*E$

$E \longrightarrow id$

LMD

$\Rightarrow E$ Expanded by $E \longrightarrow E*E$

$\Rightarrow E*E$ Expanded by $E \longrightarrow E+E$

$\Rightarrow E+E*E$ Expanded by $E \longrightarrow id$

$\Rightarrow id+E*E$ Expanded by $E \longrightarrow id$

$\Rightarrow id+id*E$ Expanded by $E \longrightarrow id$

$\Rightarrow id+id*id$

RMD

$\Rightarrow E$ Expanded by $E \longrightarrow E*E$

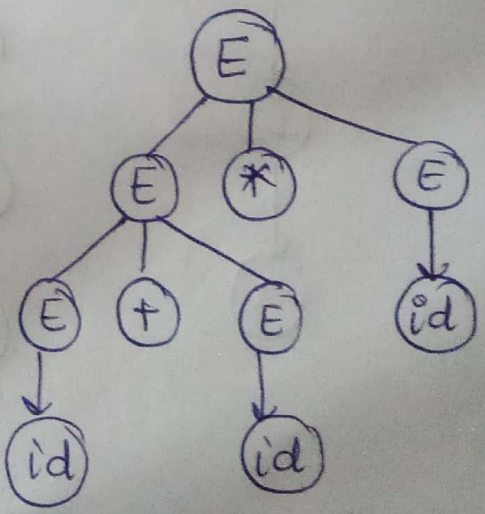
$\Rightarrow E*E$ Expanded by $E \longrightarrow id$

$\Rightarrow E*id$ Expanded by $E \longrightarrow E+E$

$\Rightarrow E+E*id$ Expanded by $E \longrightarrow id$

$\Rightarrow E+id*id$ Expanded by $E \longrightarrow id$

$\Rightarrow id+id*id$



Example of ambigous grammar

Consider the grammar $G = (V, T, E, P)$ with

$V = \{E, I\}$

$T = \{a, b, c, +, *, (,)\}$

and productions

$E \longrightarrow I$

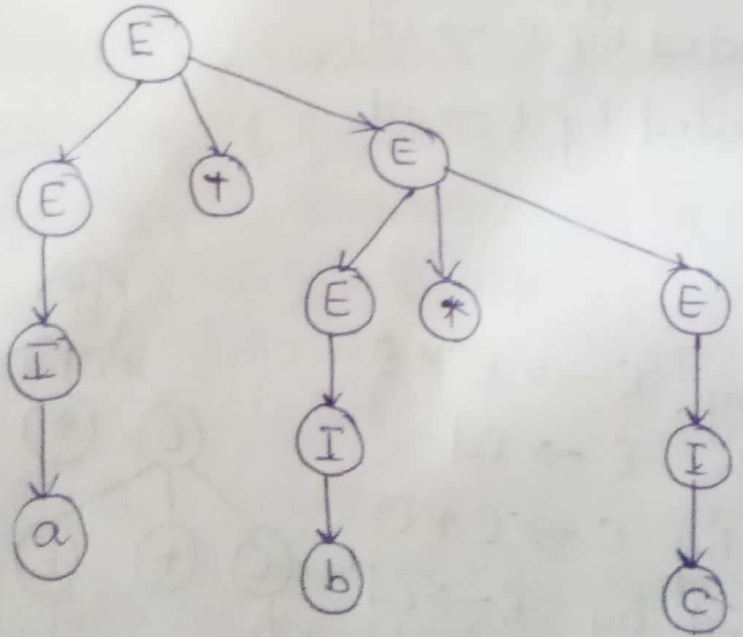
$E \longrightarrow E + E$

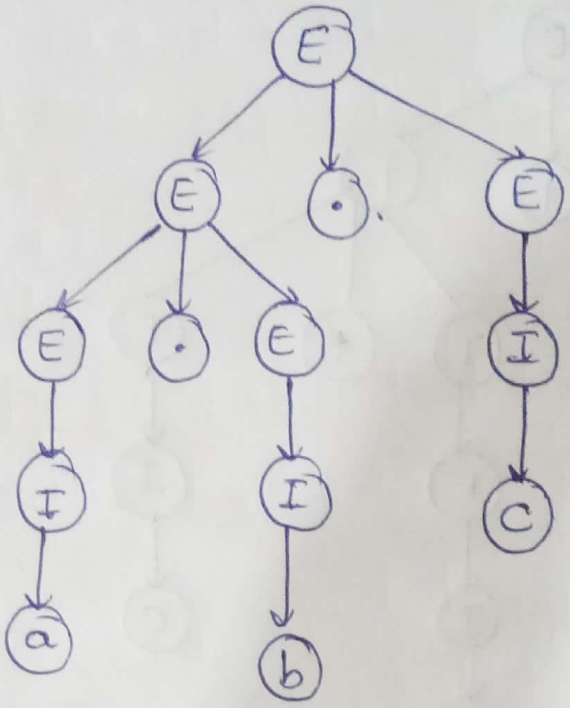
$E \rightarrow E * E$

$E \rightarrow (E)$

$I \longrightarrow a \mid b \mid c$

Derivation Tree for above grammar for string $a + b * c$





Example of unambigous grammar

Consider the grammar $G = (V, T, E, P)$ with

$V = \{E, T, F, I\}$

$T = \{ a, b, c, +, *, (,) \}$

and Production

$E \rightarrow T$

$T \rightarrow F$

$F \rightarrow I$

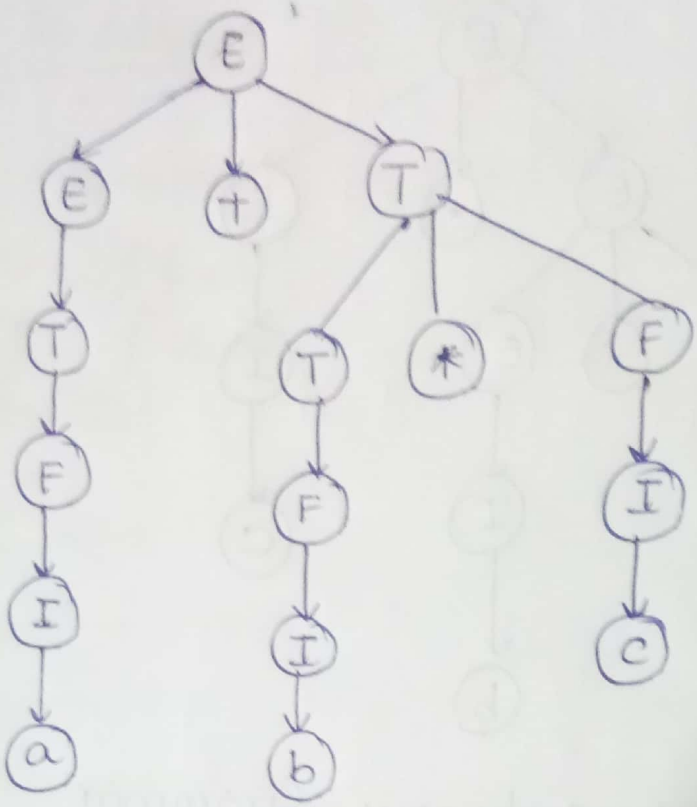
$E \rightarrow E + T$

$T \rightarrow T * F$

$F \rightarrow (E)$

$I \rightarrow a \mid b \mid c$

Derivation Tree for string $a + b * c$.



No other derivation tree is possible for string a + b * c, considering given production.

Definition

If L is a context free language for which there exists an unambigous grammar, than L is said to be unambigous. If every grammar that generates L is ambigous, then the language is called inherently ambigous

Simplification of CFG

1) Removal of null productions (λ production or ε-production)

In a CFG a non-terminal symbol 'A' is nullable variable if there is a production $A \longrightarrow \lambda$ or there is a derivation that starts at A and finally ends up with ε: $A \longrightarrow \ldots \longrightarrow \varepsilon$

Removal procedure

Step 1: find out nullable non-terminal variables which derive ε.

Step 2: For each production $A \longrightarrow a$, construct all productions $A \longrightarrow x$ where x is obtained from 'a' by removing one or multiple non-terminals from step 1.

Step 3: combine the original productions with the result of step 2 and remove ε-productions.

Finding Nullable variable

A nullable variable set V_N in G is defined as

Rule 1: Any variable $A \in V$, for which P contains a production $A \longrightarrow \varepsilon$ is nullable.
Add A into V_N.

Rule 2: If P contains a production rule $A \longrightarrow B_1 B_2 B_3 \ldots B_n$ and all $B_i \in V$ and $B_i \in V_N$ for $i = 1, \ldots, n$ then include A also into V_N.

Rule 3: No other variables are nullable.

1) Remove the null productions from given CFG.

G: $S \rightarrow ABaC$
$A \rightarrow BC$
$B \rightarrow b \mid \varepsilon$
$C \rightarrow D \mid \varepsilon$
$D \rightarrow d$

Soln

$V_N = \{S, A, B, C, D\}$
$T = \{a, b, d\}$
$S = S$

P: $S \rightarrow ABaC$
$A \rightarrow BC$
$B \rightarrow b \mid \varepsilon$
$C \rightarrow D \mid \varepsilon$
$D \rightarrow d$

To find Nullable variable

$V_N = \phi$
$V_N = \{B, C\}$ by Rule 1.
$V_N = \{A, B, C\}$ by Rule 2.

$S \rightarrow ABaC$, None of V_N is nullable
$S \rightarrow Bac$, when A is nullable
$S \rightarrow Aac$, when B is nullable
$S \rightarrow ABa$, when C is nullable
$S \rightarrow aC$, when A and B are nullable
$S \rightarrow Ba$, when A and C are nullable
$S \rightarrow Aa$, when B and C are nullable
$S \rightarrow a$, when A, B, C is nullable.
$S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Ba \mid Aa \mid a$

$A \rightarrow BC$, None of V_N is nullable
$A \rightarrow C$, when B is nullable
$A \rightarrow B$, when C is nullable
$A \rightarrow BC \mid B \mid C$

G: $S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Ba \mid Aa \mid a$

$A \rightarrow BC \mid B \mid C$

$B \rightarrow b$

$C \rightarrow D$

$D \rightarrow d$

2) Remove the null productions from given CFG.

$S \rightarrow AACD$

$A \rightarrow aAb \mid \varepsilon$

$C \rightarrow aC \mid a$

$D \rightarrow aDa \mid bDb \mid \varepsilon$

Soln $V_N = \{S, A, C, D\}$

$T = \{a, b\}$

$S = S$

$P = S \rightarrow AACD$

$A \rightarrow aAb \mid \varepsilon$

$C \rightarrow aC \mid a$

$D \rightarrow aDa \mid bDb \mid \varepsilon$

To find Nullable variable

$V_N = \phi$

$V_N = \{A, D\}$ by Rule 1

$V_N = \{A, D\}$

consider $S \rightarrow AACD$; when none of V_N is nullable

$S \rightarrow CD$, when A is nullable

$S \rightarrow AAC$, when D is nullable

$S \rightarrow C$, when A and D are nullable.

$S \rightarrow AACD \mid CD \mid AAC \mid C$

Consider $A \longrightarrow aAb$, when none of V_N is nullable

$A \longrightarrow ab$, when A is nullable.

$A \longrightarrow aAb \mid ab$

Consider $D \longrightarrow aDa$, when none of V_N is nullable

$D \longrightarrow aa$, when D is nullable

consider $D \longrightarrow bDb$, when none of V_N is nullable

$D \longrightarrow bb$, when D is nullable.

$D \longrightarrow aDa \mid aa \mid bDb \mid bb$

G : $S \longrightarrow AACD \mid ACD \mid AAC \mid C \mid AC \mid CD$

$A \longrightarrow aAb \mid ab$

$C \longrightarrow ac \mid a$

$D \longrightarrow aDa \mid aa \mid bDb \mid bb$

2. Elimination of unit production

Any Production rule in the form $A \longrightarrow B$ where $A, B \in$ Non-terminal is called unit production.

Removal procedure

Step 1 : Initialize P_1 to be P

Step 2 : for each variable A in V, find the set of A derivables.

Step 3 : for each pair (A,B) such that B is A-derivable and every non-unit production of $B \rightarrow \alpha$.
Add $A \longrightarrow \alpha$ into P_1, if it is not already exist.

Step 4 : Delete all unit production from P_1.

Finding A - derivables

Rule 1 : If $A \rightarrow B$ is a production in P, then B is A derivable.

Rule 2 : If C is A derivable and B is C- derivable then B is also A-derivable.

Rule 3 : No other variable are A-derivable.

1. Eliminate unit production from grammar.

$G : S \rightarrow S+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (S) \mid a \mid b$

Sol^n $P : P_1 \quad S \rightarrow S+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (S) \mid a \mid b$

$V = \{S, T, F\}$

Finding the derivables

S- derivable = $\{T, F\}$

T - derivable = $\{F\}$

F - derivable = ϕ

Applying all non-unit productions of F into T.

$T \rightarrow T*F \mid (S) \mid a \mid b$

Applying all non-unit productions of T into S.

$S \rightarrow S+T \mid T*F \mid (S) \mid a \mid b$

Therefore the productions are

$T \rightarrow T*F \mid (S) \mid a \mid b$

$S \rightarrow S+T \mid T*F \mid (S) \mid a \mid b$

$F \rightarrow (S) \mid a \mid b.$

2. Eliminate the unit production from the given grammar.

$G: S \rightarrow aT \mid bT \mid \varepsilon$

$T \rightarrow aS \mid bS$

Soln Removing Null productions

$P_1 : P : S \rightarrow aT \mid bT \mid \varepsilon$

$T \rightarrow aS \mid bS$

$V_N = \{S\}$

Consider $T \rightarrow as$, when S is not nullable.

$T \rightarrow a$, when S is nullable.

Consider $T \rightarrow bs$, when S is not nullable

$T \rightarrow b$, when S is nullable.

∴ The productions are

$P: S \rightarrow aT \mid bT$

$T \rightarrow aS \mid bS \mid a \mid b$

There are no unit productions in the above grammar.

Elimination of useless productions and variables

Def: Let $G = (V, T, S, P)$ be a context-free grammar. A variable $A \in V$ is said to be useful if and only if there is at least one, $w \in L(G)$ such that

$$S \overset{*}{\Rightarrow} xAy \overset{*}{\Rightarrow} w$$

with x, y in $(V \cup T)^*$.

A variable is useful if and only if it occurs in at least one derivation. A variable that is not useful is called useless. A production is useless if it involves any useless variables.

Theorem

Let $G = (V, T, S, P)$ be a context-free grammar. Then there exists an equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, \hat{S}, \hat{P})$ that does not contain any useless variables or Productions.

Proof:- The grammar $\hat{G}$ can be generated from G by an algorithm consisting of two parts. In the first part we construct an intermediate grammar $G_1 = (V_1, T_2, S, P_1)$ Such that V_1 contains only variables A for which

$$A \overset{*}{\Rightarrow} w \in T^*$$

is possible. The steps in the algorithm are

1. Set V_1 to ϕ
2. Repeat the following step until no more Variables are added to V_1. for every $A \in V$ for which P has a production of the form

$$A \longrightarrow x_1 x_2 \ldots x_n \text{ with all } x_i \text{ in } V_1 \cup T$$

add A to V_1

3. Take P_1 as all the productions in P whose symbols are all in $(V_1 \cup T)$

Dependency Graph (Unreachable variable)

A dependency graph of CFG G has its vertices labelled with variables with an edge between vertices x and y if and only if there is production of the form

$$X \longrightarrow pYq \quad | \quad p, q \in T^*$$

A variable is reachable only if there is a path from the vertex labelled X to the vertex labelled with variable.

1) Eliminate the useless symbol from the grammar.

$S \longrightarrow aS \mid A \mid C$
$A \longrightarrow a$
$B \longrightarrow aa$
$C \longrightarrow aCb$

Sol^n ① Elimination of NULL Transition

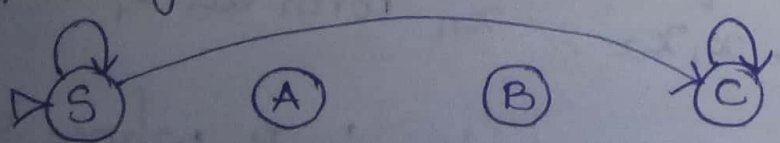
In the above grammar there are no null transitions.

② Elimination of UNIT transition

$G: S \longrightarrow aS \mid a \mid acb$
$A \longrightarrow a$
$B \longrightarrow aa$
$C \longrightarrow acb$

③ Elimination of useless symbols and Production

Dependency Graph



from the dependency graph, the variable A and B are found to be unreachable.
After Eliminating all the productions of A and B.

$S \rightarrow aS \mid aCb \mid a$

$C \rightarrow aCb$

Elimination of useless production

$V' = \phi$

$V' = \omega_1 = \{S\}$

$\omega_2 = \{S\}$

$\omega_3 = \{S\}$

A variable C is not transformed into a technical string.

After Eliminating C, we get

$S \rightarrow aS \mid a$

2. Eliminate useless symbol from the grammar

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

$A \rightarrow aB \mid \epsilon$

$B \rightarrow Aa$

$C \rightarrow cCD$

$D \rightarrow ddd \mid Cd$

Sol^n ① Elimination of null production

Find nullable variable V_N

$V_N = \phi$

$V_N = \{A\}$ by Rule 1.

Considering production containing V_N

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

$A \rightarrow aB$

$B \rightarrow Aa$

$C \rightarrow CCD$

$D \rightarrow ddd \mid Cd$

Consider $S \rightarrow aA$

$S \rightarrow aA$ none of V_N is nullable

$S \rightarrow a$ when A is nullable

Consider $B \rightarrow Aa$

$B \rightarrow Aa$ none of V_N is nullable

$B \rightarrow a$ when A is nullable.

After removing the null production,

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

$A \rightarrow aB$

$B \rightarrow Aa \mid a$

$C \rightarrow CCD$

$D \rightarrow ddd \mid cd$

② Elimination of unit production

S- derivable = {B, C}

A- derivable = ϕ

B- derivable = ϕ

C- derivable = ϕ

D- derivable = ϕ

Substituting all non unit production of B and C into S.

$S \rightarrow a \mid aA \mid Aa \mid CCD$

$A \rightarrow aB$

$B \rightarrow Aa \mid a$

$C \rightarrow CCD$

$D \rightarrow ddd \mid cd$

③ Elimination of useless productions and symbols variables transforming into a terminal string.

$V_1 = \phi$

for ω_1,

$V_1 = \{S, B, D\}$

for ω_2

$V_1 = \{S, B, D\} \cup \{A\}$

For ω_3

$V_1 = \{S, A, B, D\}$

A variable C is not transformed into terminal string

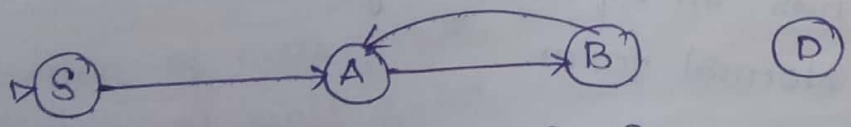
$S \longrightarrow a \mid aA \mid Aa$

$A \longrightarrow aB$

$B \longrightarrow Aa \mid a$

$D \longrightarrow ddd$

To remove useless symbols



Unreachable variable = {D}

∴ The productions are as follows.

$S \longrightarrow a \mid aA \mid Aa$

$A \longrightarrow aB$

$B \longrightarrow Aa \mid a$

Normal forms

1) Chomsky Normal form (CNF)
2) Greibach Normal form (GNF)

1) Chomsky Normal Form

A context free grammar is in chomsky Normal form if all productions are of the form

$$A \longrightarrow BC$$

or

$$A \longrightarrow a$$

where A, B, C are in V and a is in T.

Theorem

Any context-free grammar $G = (V, T, S, P)$ with $\lambda \notin L(G)$ has an equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ in chomsky Normal form.

Proof :-

Let the given grammar G has no null production and no unit productions the construction of $\hat{G}$ will be done in two steps.

Step 1 :-

Construct a grammar $G_1 = (V_1, T, S, P_1)$ from G by considering all the productions in P in the form

$$A \longrightarrow x_1 x_2 \ldots x_n,$$

where each x_i is a symbol either in V or in T.
In this case, put the production into P_1.
If $n \geq 2$, introduce new variables B_a for each $a \in T$.
Change this production of P in the form & put into P_1

$$A \longrightarrow C_1 C_2 \ldots C_n$$

where

$C_i = x_i$ if x_i is in V

and

$C_i = Ba$ if $x_i = a$

for every Ba we also put into P_1 the production

$Ba \longrightarrow a$

This part of the algorithm removes all terminals from production whose right side has length greater than one, replacing them with newly introduced variables. At the end of this step we have a grammar G_1 all of whose production have the form.

$A \longrightarrow a$

Step 2:

Introduce additional variables to reduce length of right side of production when necessary.

Convert CFG into CNF

1) Elimination of null production
2) Elimination of unit production
3) Restrict the body of the production rule
4) Represent in CNF form.

1) Convert the given CFG into CNF.

$S \longrightarrow AACD$

$A \longrightarrow aAb \mid \varepsilon$

$C \longrightarrow aC \mid a$

$D \longrightarrow aDa \mid bDb \mid \varepsilon$

Soln

Step 1:- Elimination of null production

Identify nullable variables $V_N = \{A, D\}$

∴ the production are often eliminating V_N

G_1: $S \longrightarrow AACD \mid ACD \mid AC \mid AAC \mid CD \mid C$

$A \longrightarrow aAb \mid ab$

$C \longrightarrow aC \mid a$

$D \longrightarrow aDa \mid bDb \mid aa \mid bb$

Step 2: Elimination of unit production from G_1

find S-derivable = {C}

A-derivable = ϕ

C-derivable = ϕ

D-derivable = ϕ

Substituting all non-unit production of C in S

$S \longrightarrow AACD \mid ACD \mid AAC \mid AC \mid CD \mid aC \mid a$

$A \longrightarrow aAb \mid ab$

$C \longrightarrow aC \mid a$

$D \longrightarrow aDa \mid bDb \mid aa \mid bb$

Step3 : Restricting the body of production Rule in P_1

A production in G, which are already in CNF form

$S \longrightarrow CD \mid AC \mid a$

$C \longrightarrow a$

Consider the Remaining grammar

For S, $S \rightarrow AACD \mid ACD \mid AAC$

$S \rightarrow aC$

$S \rightarrow B_aC$

$B_a \rightarrow a$

for A, $A \rightarrow aAb$

$A \rightarrow B_aABb$

$B_b \rightarrow b$

$A \rightarrow ab$

$A \rightarrow B_aB_b$

for C, $C \rightarrow aC$

$C \rightarrow B_aC$

For D, $D \rightarrow aDa$

$D \rightarrow B_aDBa$

$D \rightarrow bDb$

$D \rightarrow B_bDB_b$

$D \rightarrow aa$

$D \rightarrow B_aB_a$

$D \rightarrow bb$

$D \rightarrow B_bB_b$

Step 4: Represent G_1 in CNF

$S \rightarrow AACD$

$S \rightarrow AX_1$

$X_1 \rightarrow AX_2$

$X_2 \rightarrow CD$

$S \longrightarrow ACD$

$S \longrightarrow AX_2$

$X_2 \longrightarrow CD$

$S \longrightarrow AAC$

$S \longrightarrow AX_3$

$X_3 \longrightarrow AC$

$A \longrightarrow BaABb$

$A \longrightarrow BaX_4$

$X_4 \longrightarrow ABb$

$D \longrightarrow BaDBa$

$D \longrightarrow BaX_5$

$X_5 \longrightarrow DBa$

$D \longrightarrow BbDBb$

$D \longrightarrow BbX_6$

$X_6 \longrightarrow DBb$

Thus the CNF of G_1 is

$S \longrightarrow AX_1 \mid AX_2 \mid AX_3 \mid AC \mid CD \mid BaC \mid a$

$X_1 \longrightarrow AX_2$

$X_2 \longrightarrow CD$

$X_3 \longrightarrow AC$

$A \longrightarrow BaX_4 \mid BaBb$

$X_4 \longrightarrow ABb$

$C \longrightarrow BaC \mid a$

$D \longrightarrow BaX_5 \mid BbX_6 \mid BaBb \mid BbBb$

$X_5 \longrightarrow DBa$

$X_6 \longrightarrow DBb$

$Ba \longrightarrow a \qquad Bb \longrightarrow b$

2) Convert the CFG into CNF

$S \rightarrow AB \mid ABC$

$A \rightarrow BA \mid BC \mid a \mid \varepsilon$

$B \rightarrow AC \mid CB \mid b \mid \varepsilon$

$C \rightarrow BC \mid AB \mid A \mid C$

Sol^n

Step1: Elimination of null production

Identify nullable variable $V_N = \{A, B, C, S\}$

G: $S \rightarrow AB \mid ABC \mid A \mid B \mid C \mid AB \mid BC \mid AC$

$A \rightarrow BA \mid BC \mid a \mid A \mid B \mid C$

$B \rightarrow AC \mid CB \mid C \mid A \mid B \mid C$

$C \rightarrow BC \mid AB \mid A \mid C \mid CB \mid AC$

Step2: Elimination of unit production from G_1

finding derivatives

S- derivable = {A, B, C}

A - derivable = {B, A, C}

B - derivable = {C, A, B}

C - derivable = {B, C, A}

Substituting all non-unit production of A, B and C in S.

G: $S \rightarrow AB \mid ABC$

$A \rightarrow BA \mid BC$

$B \rightarrow AC \mid CB$

$C \rightarrow BC \mid AB$

Step3: Restrict the body of production rule in P_1 A production in G which are already in CNF.

$S \rightarrow AB$

$A \rightarrow BA \mid BC$

$B \rightarrow AC \mid CB$

$C \rightarrow BC \mid AB$

for $S \rightarrow ABC$

step 4 : Represent G in CNF

$S \rightarrow ABC$

$S_1 \rightarrow AX_1$

$X_1 \rightarrow BC$

Thus the CNF is

$S \rightarrow AB \mid BC \mid AC \mid BA \mid CB \mid AX_1 \mid a \mid b \mid c$

$X_1 \rightarrow BC$

$A \rightarrow BA \mid BC \mid AC \mid CB \mid AB \mid a \mid b \mid c$

$B \rightarrow AC \mid CB \mid BC \mid AB \mid BA \mid a \mid b \mid c$

$C \rightarrow BC \mid AB \mid AC \mid CB \mid BA \mid a \mid b \mid c$

Griebach Normal form (GNF)

A context free Grammar is in Griebach Normal form (GNF) if the productions are in the following forms.

$$A \longrightarrow a$$

or

$$A \longrightarrow b C_1 C_2 \ldots C_n$$

where $A, C_1, \ldots C_n$ are non-terminal and b is terminal

Steps to convert CFG to GNF

Step 1: check if the given CFG has unit production or null productions and remove if there are any.

Step 2: check whether the CFG is already in chomsky Normal form and convert it to CNF if it is not.

Step 3: change the names of the non-terminal symbols into Some A_i in ascending order of i.

Step 4: Alter the Rules so that the non-terminal are in ascending order, such that, if the production is of the form $A_i \longrightarrow A_j X$, then $i < j$ and should never be $i >= j$.

Ex: Convert the given CFG into GNF

$S \rightarrow CA \mid BB$

$B \rightarrow b \mid SB$

$C \rightarrow b$

$A \rightarrow a$

Soln

Step 1) There are no null production and no unit production

Step 2) The given CFG is already in CNF.

Step 3) Replace S with A_1
C with A_2
A with A_3
B with A_4

After Replacing we get:

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b \mid A_1A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step 4) Alter the rules so that the non-terminal are in ascending order, such that, if the production is of the form $A_i \rightarrow A_jX$, then $i < j$ and should never be $i >= j$.

$A_4 \rightarrow b \mid A_1A_4$

$A_4 \rightarrow b \mid A_2A_3A_4 \mid A_4A_4A_4$

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

Step 5) Removing left Recursion → left recursion

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

Introduce a new variable to remove left recursion.

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

$Z \rightarrow A_4A_4 \mid A_4A_4Z$

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

Now the grammar is

$A_1 \longrightarrow A_2A_3 \mid A_4A_4$

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

$Z \longrightarrow A_4A_4 \mid A_4A_4Z$

$A_2 \longrightarrow b$

$A_3 \longrightarrow a$

Considering $A_1 \longrightarrow A_2A_3 \mid A_4A_4$

$A_1 \longrightarrow bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$

Considering $Z \longrightarrow A_4A_4 \mid A_4A_4Z$

$Z \longrightarrow bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4 \mid bA_4Z \mid bA_3A_4A_4Z \mid$
$bZA_4Z \mid bA_3A_4ZA_4Z$

Thus the final grammar is

$A_1 \longrightarrow bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$

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

$Z \longrightarrow bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4 \mid$
$bA_4Z \mid bA_3A_4A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4Z$

$A_2 \longrightarrow b$

$A_3 \longrightarrow a$

2) Convert the given CFG into GNF

$S \rightarrow AB$

$A \rightarrow aA \mid bB \mid a$

$B \rightarrow b$

Soln

Step 1) no null production and no unit production

Step 2)

$S \rightarrow AB$

$A \rightarrow AA \mid BB \mid a$

$B \rightarrow b$

The above grammar is in CNF

Step 3)

S with A_1

A with A_2

B with A_3

After replacing the production we get

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_2A_2 \mid A_3A_3 \mid a$

$A_3 \rightarrow b$

After resolving A_2

$A_2 \rightarrow aA_2 \mid bA_3 \mid a$

Thus the final Grammar is

$A_1 \rightarrow aA_2A_3 \mid bA_3A_3 \mid aA_3$

$A_2 \rightarrow aA_2 \mid bA_3 \mid a$

$A_3 \rightarrow b$

3) convert the given CFG into GNF.

$S \rightarrow abSb \mid aa$.

Soln

Step1: no null production and no unit production

Step2: $S' \rightarrow S$

$S \rightarrow abSb \mid aa$

After removing

$S \rightarrow abSb$

$S \rightarrow aXb$

$X \rightarrow bS$

$Y \rightarrow Xb$

$S \rightarrow aY$

∴ the production will be

$S' \rightarrow S$

$S \rightarrow aY \mid AA$

$A \rightarrow a$

Step3: S' with A_1

S with A_2

Y with A_3

A with A_4

$A_1 \rightarrow A_2$

$A_2 \rightarrow aA_3 \mid A_4A_4$

$A_4 \rightarrow a$

After substituting the value of A_4 and A_2.

$A_2 \rightarrow aA_3 \mid aA_4$

$A_1 \rightarrow aA_3 \mid aA_4$

$A_4 \rightarrow a$.

Thus the final grammar is

$A_1 \rightarrow aA_3 \mid aA_4$

$A_2 \rightarrow aA_3 \mid aA_4$

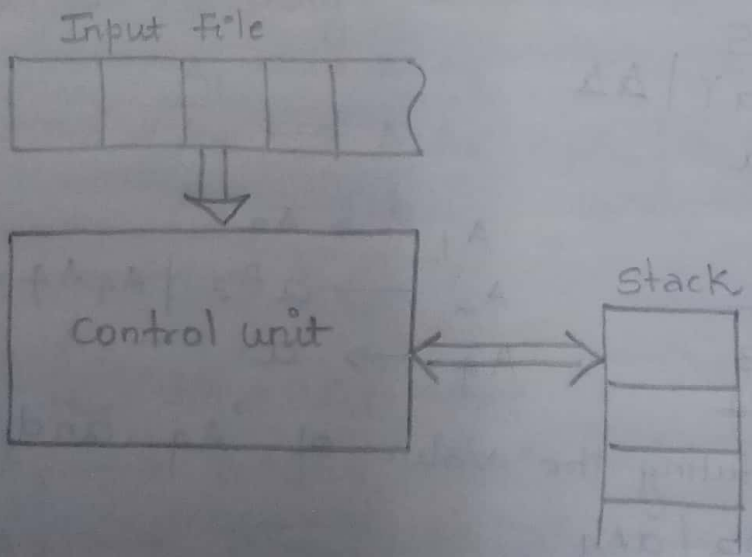
$A_4 \rightarrow a$

Pushdown Automata (PDA)

A pushdown automaton is a way to implement a context free grammar in a similar way we design DFA for a regular grammar. A DFA can remember a finite amount of information but a PDA can remember an infinite amount of information.

Basically a pushdown automaton is a "finite state machine" + "a stack".

A pushdown automaton has a three components
- A input tape
- A control unit and
- A stack with infinite size.



The stack head scans the top symbol of the stack.
A stack does two operations.
Push: a new symbol is added at the top.
Pop : the top symbol is read and removed.

A PDA may or may not read an input symbol but it have to read the top of the stack in every transition.

A non-deterministic pushdown automata (NPDA) is defined by 7-tuple.

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

where

Q is a finite set of internal states of the control unit.

Σ is the input alphabet.

Γ is a finite set of symbols called the stack alphabet.

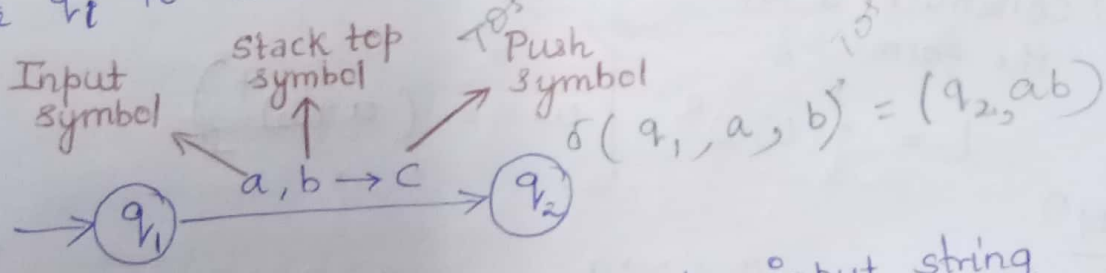
$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow$ set of finite subsets of $Q \times \Gamma^*$ is the transition function,

$q_0 \in Q$ is the initial state of the control unit,

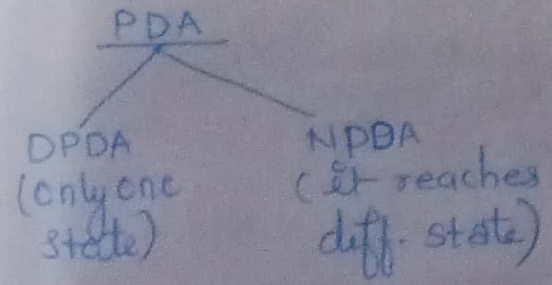
$Z \in \Gamma$ is the stack start symbol,

$F \subseteq Q$ is the set of final states.

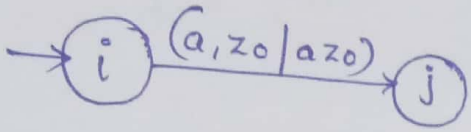
The following diagram shows a transition in a PDA from state q_1 to state q_2, labeled as $a, b \rightarrow c$:

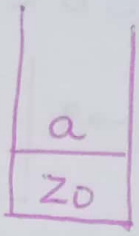


This means at state q_1, if we encounter input string 'a' and top symbol of the stack is 'b', then we pop 'b', push 'c' on top of the stack and move to state q_2.



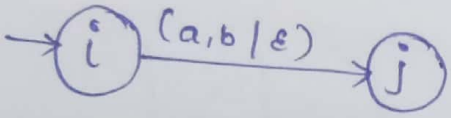
Push operation

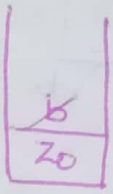




$$\delta(i, a, z_0) = (j, a z_0)$$

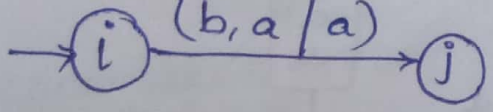
Pop operation

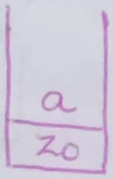




$$\delta(i, a, b) = (j, \varepsilon)$$

No operation





$$\delta(i, b, a) = (j, a)$$

Pushdown Automata

Deterministic Pushdown Automata

- ✓ In DPDA centre is known
  eg. $0^n 1^n$
- ✓ There is only one move in every situation

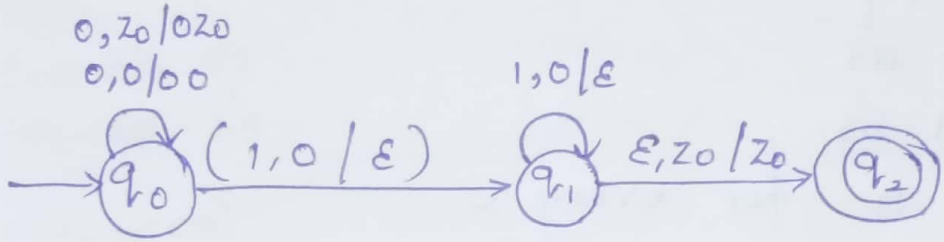
Non-deterministic Pushdown Automata.

- ✓ In NPDA centre is not known
  eg. WW^R
- ✓ There are multiple moves.

Ex1. Construct a NPDA that accepts

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

Soln



$$\delta(q_0, 0, Z_0) = (q_0, 0Z_0)$$
$$\delta(q_0, 0, 0) = (q_0, 00)$$
$$\delta(q_0, 1, 0) = (q_1, \varepsilon)$$
$$\delta(q_1, 1, 0) = (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, Z_0) = (q_0, Z_0)$$

2) Construct a PDA for the language

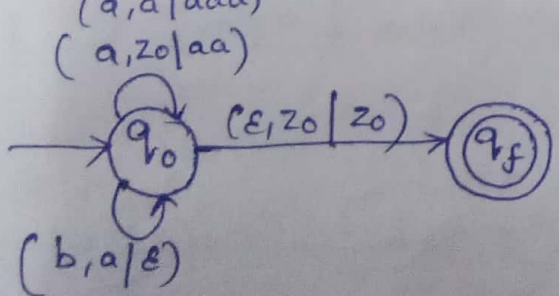
$$L = \{ a^n b^{2n} \mid n > 1 \}$$

Soln

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

Logic

1) for every one a, Push two a's.
2) Pop two b's.

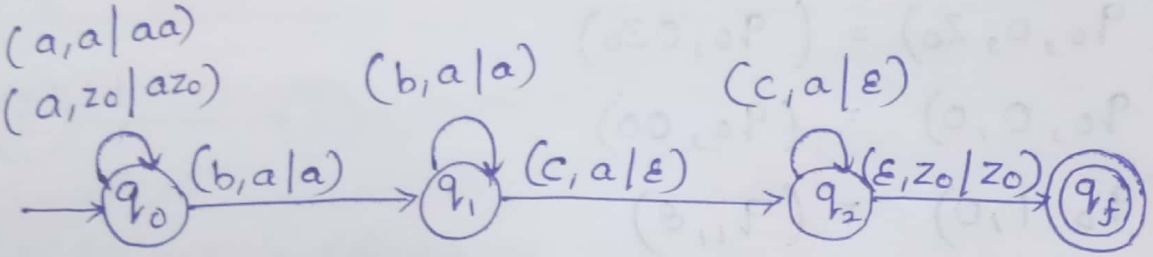


3) Construct a PDA $L = \{a^n b^m c^n \mid n, m \geq 1\}$

Soln

Logic

1) Push all a's
2) Drop all b's
3) Pop one a for every c.



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

$Q = \{q_0, q_1, q_2, q_f\}$

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

$\Gamma = \{z_0, a\}$

$Z_0 = \{Z\}$

$q_0 = \{q_0\}$

$F = \{q_f\}$

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

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

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

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

$\delta(q_1, c, a) = (q_2, \epsilon)$

$\delta(q_2, c, a) = (q_2, \varepsilon)$

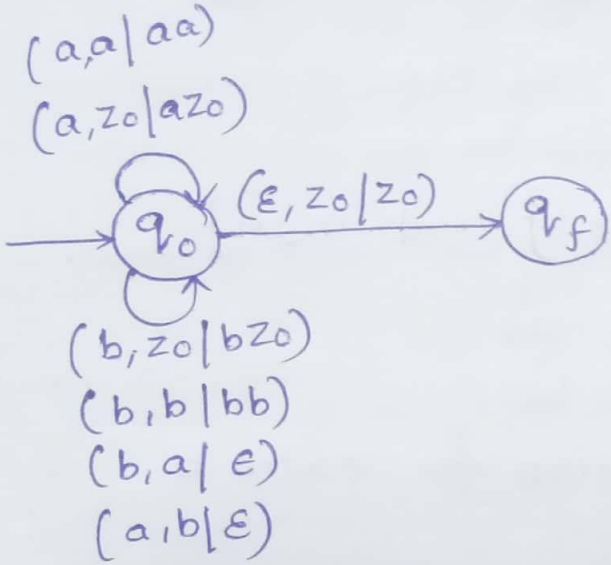
$\delta(q_2, \epsilon, z_0) = (q_f, z_0)$

4) $L = \{ w \in \{a,b\}^* \mid n_a(w) = n_b(w) \}$

soln $L = \{ aabb, bbaa, ab, ba, \ldots \}$

logic

1) Push all a's
2) Pop all b's



consider string aabbbab

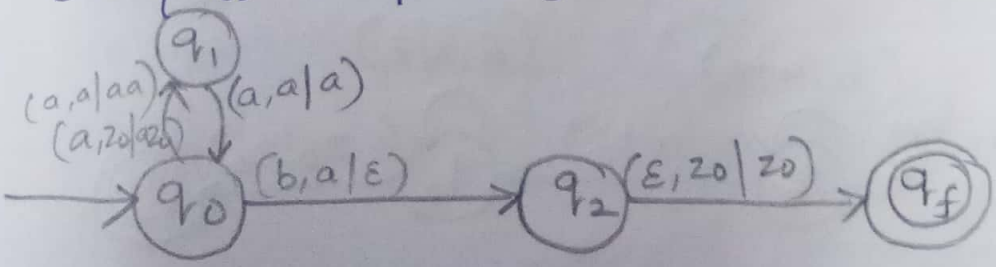
$\delta(q_0, aabbbab, z_0) \vdash \delta(q_0, abbbab, az_0)$

$\vdash \delta(q_0, bbbab, aaz_0) \vdash \delta(q_0, bbab, az_0)$

$\vdash \delta(q_0, bab, z_0) \vdash \delta(q_0, ab, z_0)$

$\vdash \delta(q_0, b, az_0) \vdash \delta(q_2, \varepsilon, z_0) \vdash q_f$

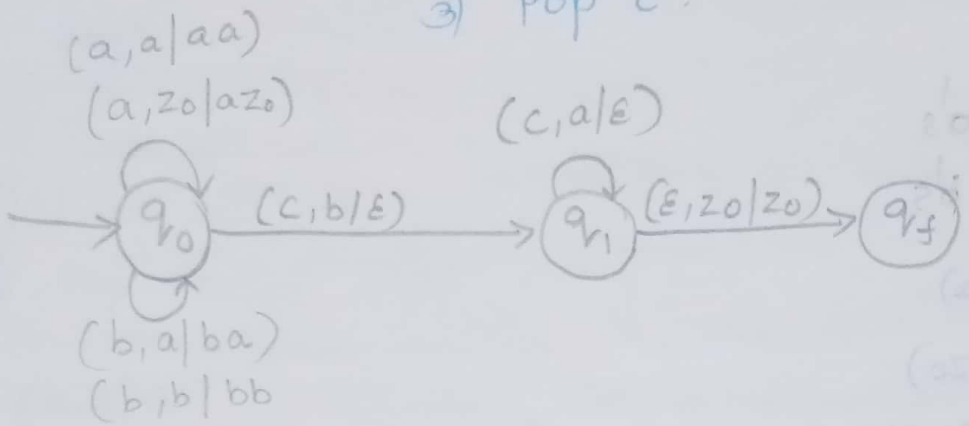
5) $L = \{ a^{2n} b^n \mid n \geq 1 \}$



6) $L = \{ a^i b^j c^k \mid k = i+j \;,\; i, j, k \geq 0 \}$

Soln

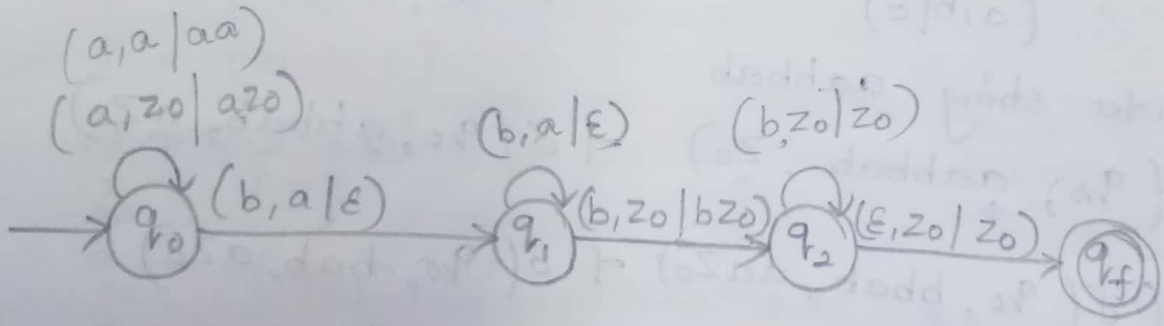
Logic
1) Push a's
2) Push b's
3) Pop c



7) $L = \{ a^m b^n \mid n, m \geq 1 \text{ and } m \leq n \}$

Soln

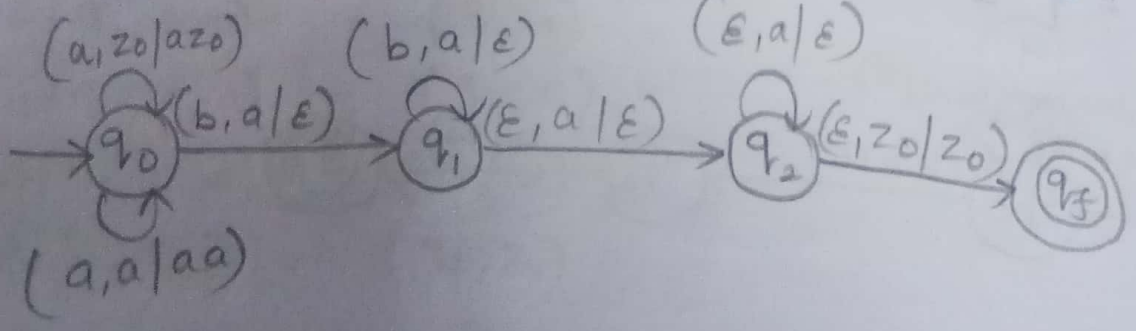
Logic
1) Push all a's
2) Pop all b's
3) no operation for extra b



8) $L = \{ a^m b^n \mid n, m \geq 1 \text{ and } n < m \}$

Soln

Logic
1) Push all a's
2) Pop all b
3) No matter what i/p symbol it just pop.



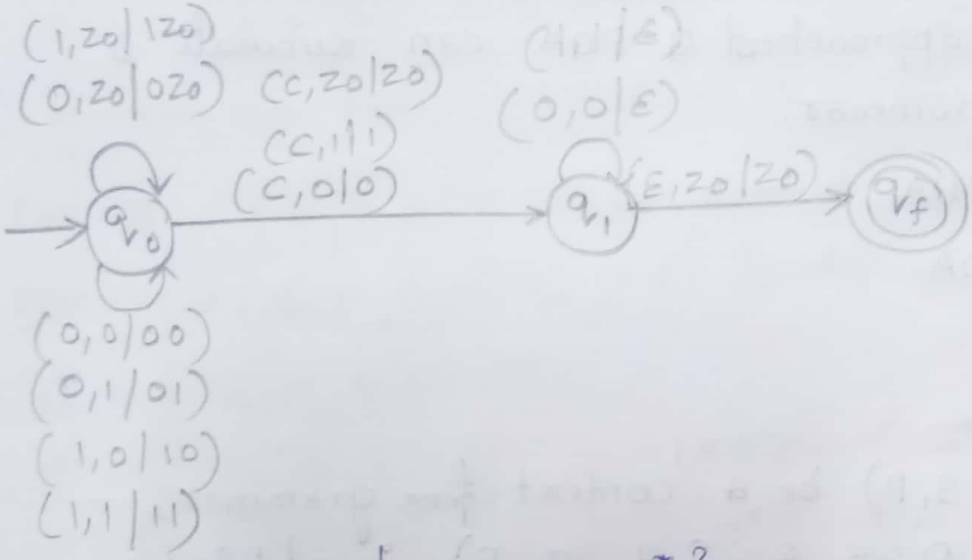
9) Construct a PDA for the language L.

$L = \{ \omega c \omega^R \mid \omega \in (0,1)^* \text{ and } \omega^R \text{ denotes the reverse of } \omega \}$

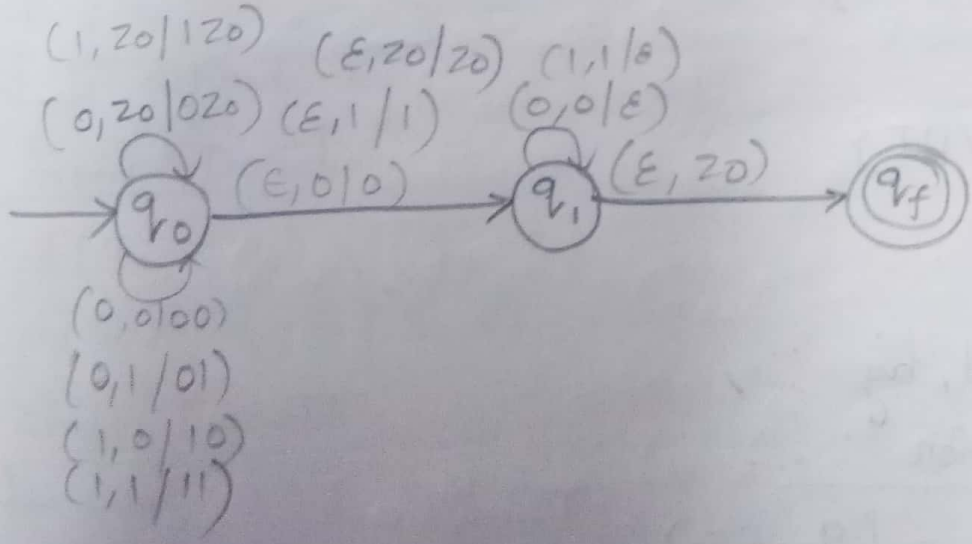
Soln.

Logic

1) Push 0 and 1
2) Do not perform any operation
3) Pop



10) $L = \{ \omega \omega^R \mid \omega \in \{0,1\}^* \}$ NPDA



Pushdown Automata And context free Language.

- With the given CFG to build a PDA that can test an arbitary string and determine whether it can deliver from string.
- The basic strategy is to simulate a derivation or reduction of string in the given grammar.
- It will require guesting the steps of reduction or derivation and the PDA will be non-deterministic

There are two approached a PDA can simulate a context free grammar.

1) Top Down PDA
2) Bottom Up PDA.

1) Top Down PDA

Let $G = (V, T, S, P)$ be a context free grammar, the PDA $M = (Q, \Sigma, S, \delta, q_0, Z, F)$ is defined as

$Q = \{q_0, q_1, q_2\}$

$\Sigma = T$

$S = \{Z_0 U V U T\}$

$q_0 = q_0$

$F = q_2$

δ is defined by

Fixed Transition

$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$

$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$

Production Transition / Pushing Transition

for every $A \in V$, a production $A \longrightarrow B$ in P, add transition,

$$\delta(q_1, \varepsilon, A) = (q_1, B)$$

Matching Transition / Popping Transition

for every $a \in \Sigma$, add a transition

$$\delta(q_1, a, a) = (q_1, \varepsilon)$$

2) Bottom up PDA

Let $G = (V, T, S, P)$ be a context free grammar, the PDA $M = (Q, \Sigma, S, \delta, q_0, Z, F)$ is defined as

$Q = \{q_0, q_1, q_2\}$

$\Sigma = T$

$S = \{Z_0 \cup V \cup T\}$

$q_0 = q_0$

$F = q_2$

δ is defined by

Fixed Transition

$$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$$

$$\delta(q_1, \varepsilon, S) = (q_2, \varepsilon)$$

Shift Transition

for every $a \in \Sigma$ and every $X \in S$, add the transition

$$\delta(q_1, a, X) = (q_1, aX)$$

Reduced Transition

for every $A \longrightarrow B$, add a transition

$$\delta(q_1, \varepsilon, A) = (q_1, B)$$

1) Construct a Pushdown Automata using top down PDA corresponding to given CFG. Also show the moves made by a PDA for string aaabc.

$S \rightarrow aA$

$A \rightarrow aABC \mid bB \mid a$

$B \rightarrow b$

$C \rightarrow c$

Sol^n

The PDA $M = (Q, \Sigma, S, \delta, q_0, Z, F)$ is defined by

$Q = \{q_0, q_1, q_2\}$

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

$S = \{Z_0, a, b, C, S, A, B, c\}$

$q_0 = q_0$

$F = q_2$

fixed Transition

$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$

$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$

Production Transition / Pushing Transition

for every $A \in V$, a production $A \rightarrow B$ in P, add transition.

$\delta(q_1, \varepsilon, S) = (q_1, aA)$

$\delta(q_1, \varepsilon, A) = \{(q_1, aABC), (q_1, bB), (q_1, a)\}$

$\delta(q_1, \varepsilon, B) = (q_1, b)$

$\delta(q_1, \varepsilon, c) = (q_1, c)$

Matching Transition / popping Transition

for every $a \in \Sigma$, add a transition

$\delta(q_1, a, a) = (q_1, \varepsilon)$

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

$\delta(q_1, c, c) = (q_1, \varepsilon)$

Therefore the PDA will be

$(\varepsilon, S | aA)$
$(\varepsilon, A | aABC)$
$(\varepsilon, A | bB)$
$(\varepsilon, A | a)$
$(\varepsilon, B | b)$
$(\varepsilon, C | c)$
$(a, a | \varepsilon)$
$(b, b | \varepsilon)$
$(c, c | \varepsilon)$

To check whether the string is accepted or not.

$(q_0, aaabc, z_0) \vdash (q_1, aaabc, Sz_0)$ Assume (q_1, ε, S)

$\vdash (q_1, aaabc, aAz_0)$ pop operation

$\vdash (q_1, aabc, Az_0)$ Assume (q_1, ε, A)

$\vdash (q_1, aabc, aABCz_0)$ pop operation

$\vdash (q_1, abc, ABCz_0)$ Assume (q_1, ε, A)

$\vdash (q_1, abc, aBCz_0)$ Pop operation

$\vdash (q_1, bc, BCz_0)$ Assume (q_1, ε, B)

$\vdash (q_1, bc, bCz_0)$ Pop operation

$\vdash (q_1, c, Cz_0)$ Assume (q_1, ε, C)

$\vdash (q_1, \varepsilon, z_0)$ pop operation.

q_2 & $q_2 \in F$

Therefore string aaabc is accepted.

2) Construct a PDA using Top down approach corresponding to given CFG. Also show the moves made by a PDA for a+a*a.

G: $S \longrightarrow S+T \mid T$

$T \longrightarrow T*a \mid a$

Soln

The PDA $M = (Q, \Sigma, S, \delta, q_0, Z, F)$ is defined by

$Q = \{q_0, q_1, q_2\}$

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

$S = \{Z_0, a, b, c, S, A, B, C\}$

$q_0 = q_0$

$F = q_2$

Fixed Transition

$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$

$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$

Production Transition / Pushing Transition

$\delta(q_1, \varepsilon, S) = \{(q_1, S+T), (q_1, T)\}$

$\delta(q_1, \varepsilon, T) = \{(q_1, T*a), (q_1, a)\}$

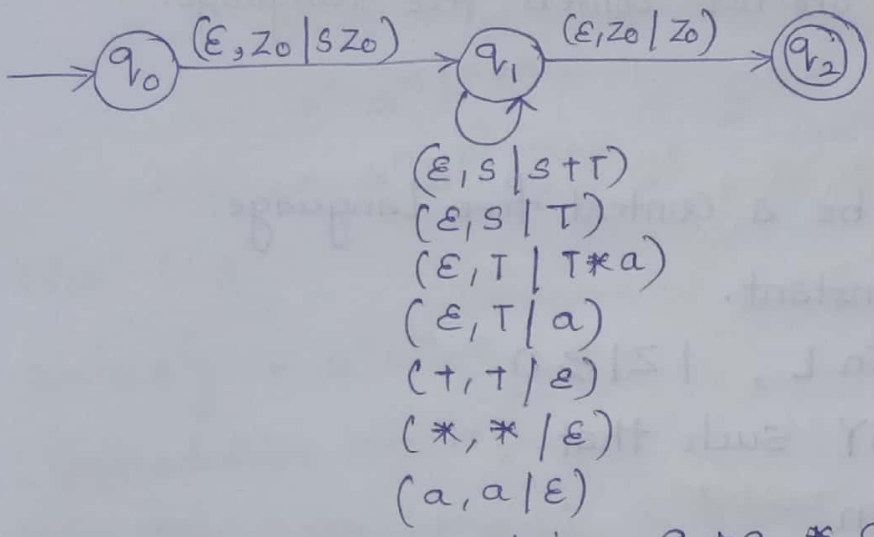
Matching Transition / Popping Transition.

$\delta(q_1, +, +) = (q_1, \varepsilon)$

$\delta(q_1, *, *) = (q_1, \varepsilon)$

$\delta(q_1, a, a) = (q_1, \varepsilon)$

Therefore the PDA will be



To check whether string $a+a*a$ is accepted or not.

$$
\begin{aligned}
(q_0, a+a*a, Z_0) &\vdash (q_1, a+a*a, SZ_0) \\
&\vdash (q_1, a+a*a, S+TZ_0) \\
&\vdash (q_1, a+a*a, T+TZ_0) \\
&\vdash (q_1, a+a*a, a+TZ_0) \\
&\vdash (q_1, +a*a, +TZ_0) \\
&\vdash (q_1, a*a, TZ_0) \\
&\vdash (q_1, a*a, T*aZ_0) \\
&\vdash (q_1, a*a, a*aZ_0) \\
&\vdash (q_1, *a, *aZ_0) \\
&\vdash (q_1, a, aZ_0) \\
&\vdash (q_1, \varepsilon, Z_0) \\
&\quad q_2 \text{ where } q_2 \in F.
\end{aligned}
$$

Therefore string $a+a*a$ is accepted.

Pumping Lemma for context free Languages.

Pumping lemma is a technique used to prove that certain languages are not context free language.

Steps:

1) Assume Let L be a context free Language.
2) Let n be a constant.
3) Any string z in L, $|z| \geq n$
4) Split $z = uvwxy$ such that
   (i) $|vwx| \leq n$
   (ii) $vx \neq \varepsilon$ or $|vx| \geq 1$
   (iii) for all $i \geq 0$, $uv^iwx^iy \in L$.

Ex1: Prove that the following language is not a CFL.

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

Sol^n

1) Let L be a CFL.
2) Let n be a constant
3) Let $z = a^n b^n c^n$, $|z| \geq n$
4) split $z = uvwxy$

$u = a^n$

$vwx = b^n$, $|vwx| \leq n$

$vx = b^{n-m}$, $|vx| \geq 1$, $m < n$

$y = c^n$

$$uv^iwx^iy = uvv^{i-1}wxx^{i-1}y$$
$$= uvwx(vx)^{i-1}y$$
$$= a^n b^n (b^{n-m})^{i-1} c^n$$
$$= a^n b^{n+ni-n-mi+m} c^n$$
$$= a^n b^{ni-mi+m} c^n$$

Pick $i=0$

$$uv^iwx^iy = a^n b^m c^n \notin L$$

Contradiction occurs!!

Hence the given Language L is not CFL.

Ex2: Prove that the following Language is not a CFL.

$$L = \{ a^k b^j c^k d^i \mid k \geq 1, j \geq 1 \}.$$

Soln

1) Let L be a CFL.
2) Let n be a constant
3) Let $z = a^n b^m c^n d^m$, $|z| \geq n$
4) Split $z = uvwxy$

$u = a^n b^m$

$vwx = c^n$, $|vwx| \leq n$

$y = d^m$

$vx = c^{n-p}$, $|vx| \geq 1$

$$uv^iwx^iy = uvv^{i-1}wxx^{i-1}y$$
$$= uvwx(vx)^{i-1}y$$
$$= a^n b^m c^n (c^{n-p})^{i-1} d^m$$
$$= a^n b^m c^{ni-pi+p} d^m$$

Pick $i=0$

$uv^iwx^iy = a^n b^m c^p d^m \notin L$

Contradiction occurs!!

Hence the given Language L is not a CFL.

3) Prove that the following language is not a CFL.

$$L = \{0^{2^i} \mid i \geq 1\}$$

Soln

1) Let L be a CFL
2) Let n be a constant
3) Let $z = 0^{2^p} = 0^n$, $|z| \geq n$, $n = 2^p$
4) split $z = uvwxy$

$u = 0^q$, $q < n$

$vwx = 0^r$, $r \leq (n-q)$

$vx = 0^s$, $|vx| \geq 1$, $s < r$

$y = 0^{n-(q+r)}$

$$\begin{aligned} uv^iwx^iy &= uvv^{i-1}wxx^{i-1}y \\ &= uvwx(vx)^{i-1}y \\ &= 0^q 0^r (0^s)^{i-1} 0^{n-q-r} \\ &= 0^q 0^r 0^{si-s} 0^{n-q-r} \\ &= 0^{q+r+si-s+n-q-r} \\ &= 0^{si-s+n} \end{aligned}$$

Pick $i=0$

$uv^iwx^iy = 0^{n-s} \notin L$

Contradiction occurs!

Hence the given L is not a CFL.

Closure Properties of context free Language (Refer Pdf).

- Context free Language are closed under
  - union
  - concatenation
  - Kleen star closure.
- Context free Language are not closed under
  - Intersection
  - Intersection with Regular Language
  - complement.

8.2 Closure Properties of CFL

In this section, we investigate the closure of CFLs under some operations like union, intersection, difference, substitution, homomorphism, and inverse homomorphism etc. The first result that we will prove is closure under substitution, using which we establish closure under union, catenation, catenation closure, catenation +, and homomorphism.

Theorem 8.2 *Let L be a CFL over T_Σ and σ be a substitution on T such that $\sigma(a)$ is a CFL for each a in T. Then $\sigma(L)$ is a CFL.*

Proof Let $G = (N, T, P, S)$ be a CFG generating L. Since $\sigma(a)$ is a CFL, let $G_a = (N_a, T_a, P_a, S_a)$ be a CFG generating $\sigma(a)$ for each $a \in T$. Without loss of generality, $N_a \cap N_b = \phi$ and $N_a \cap N = \phi$ for $a \neq b$, $a, b \in T$. We now construct a CFG $G' = (N', T', P', S')$ which generates $\sigma(L)$ as follows:

1. N' is the union of N_a's, $a \in T$ and N
2. $T' = \bigcup_{a \in T} T_a$
3. P' consists of:
   - all productions in P_a for $a \in T$
   - all productions in P, but for each terminal a occurring in any rule of P, is to be replaced by S_a. i.e., in $A \to \alpha$, every occurrence of a $(\in T)$ in α is replaced by S_a.

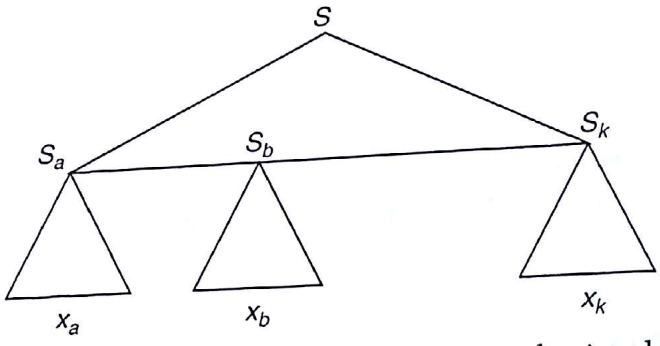


Figure 8.2. A derivation tree showing a string obtained by substitution

Any derivation tree of G' will typically look as in the following figure (Figure 8.2).

Here $ab\ldots k$ is a string of L and $x_a x_b \ldots x_k$ is a string of $\sigma(L)$. To understand the working of G' producing $\sigma(L)$, we have the following discussion:

A string w is in $L(G')$ if and only if w is in $\sigma(L)$. Suppose w is in $\sigma(L)$. Then, there is some string $x = a_1 \ldots a_k$ in L and strings x_i in $\sigma(a_i)$, $1 \leq i \leq k$, such that $w = x_1 \ldots x_k$. Clearly from the construction of G', $S_{a_1} \cdots S_{a_k}$ is generated (for $a_1 \ldots a_k \in L$). From each S_{a_i}, x_is are generated where $x_i \in \sigma(a_i)$. This becomes clear from the above picture of derivation tree. Since G' includes productions of G_{a_i}, $x_1 \ldots x_k$ belongs to $\sigma(L)$.

Conversely for $w \in \sigma(L)$, we have to understand the proof with the help of the parse tree constructed above. That is, the start symbol of G and G' are S. All the non-terminals of G, G_a's are disjoint. Starting from S, one can use the productions of G' and G and reach $w = S_{a_1} \cdots S_{a_k}$ and $w' = a_1 \ldots a_k$, respectively. Hence, whenever w has a parse tree T, one can identity a string $a_1 a_2 \ldots a_k$ in $L(G)$ and string in $\sigma(a_i)$ such that $x_1 \ldots x_k \in \sigma(L)$. Since $x_1 \ldots x_k$ is a string formed by substitution of strings x_i's for a_i's, we conclude $w \in \sigma(L)$. □

Remark One can use the substitution theorem of CFLs to prove the closure of CFLs under other operations.

Theorem 8.3 *CFLs are closed under union, catenation, catenation closure* $(*)$, *catenation* $+$, *and homomorphism.*

Proof

1. **Union:** Let L_1 and L_2 be two CFLs. If $L = \{1, 2\}$ and $\sigma(1) = L_1$ and $\sigma(2) = L_2$. Clearly, $\sigma(L) = \sigma(1) \cup \sigma(2) = L_1 \cup L_2$ is CFL by the above theorem.
2. **Catenation:** Let L_1 and L_2 be two CFLs. Let $L = \{12\}$. $\sigma(1) = L_1$ and $\sigma(2) = L_2$. Clearly, $\sigma(L) = \sigma(1).\ \sigma(2) = L_1 L_2$ is CFL as in the above case.

3. **Catenation closure** $(*)$: Let L_1 be a CFL. Let $L = \{1\}^*$ and $\sigma(1) = L_1$. Clearly, $L_1^* = \sigma(L)$ is a CFL.
4. **Catenation** $+$: Let L_1 be a CFL. Let $L = \{1\}^+$ and $\sigma(1) = L_1$. Clearly $L_1^+ = \sigma(L)$ is a CFL.
5. **Homomorphism:** This follows as homomorphism is a particular case of substitution. □

Theorem 8.4 *If L is a CFL then L^R is CFL.*

Proof Let $L = L(G)$ be a CFL where $G = (N, T, P, S)$ be a CFG generating L. Let $G^R = (N, T, P^R, S)$ be a new grammar constructed with $P^R = \{A \to \alpha^R | A \to \alpha \in P\}$. Clearly, one can show by induction on the length of the derivation that $L(G^R) = L^R$. □

Theorem 8.5 *CFLs are not closed under intersection and complementation.*

Proof Let $L_1 = \{a^n b^n c^m | n, m \geq 1\}$ and $L_2 = \{a^m b^n c^n | n, m \geq 1\}$.

Clearly, L_1 and L_2 are CFLs. (Exercise: Give CFG's for L_1 and L_2).

$L_1 \cap L_2 = \{a^n b^n c^n | n \geq 1\}$ which has been shown to be non-context-free. Hence, CFLs are not closed under $\cap$.

For non-closure under complementation, if CFL's are closed under complementation, then for any two CFLs L_1 and L_2, $L_1 \cap L_2 = (L_1^c \cup L_2^c)^c$ which is a CFL. Hence, we get CFLs are closed under intersection, which is a contradiction. □

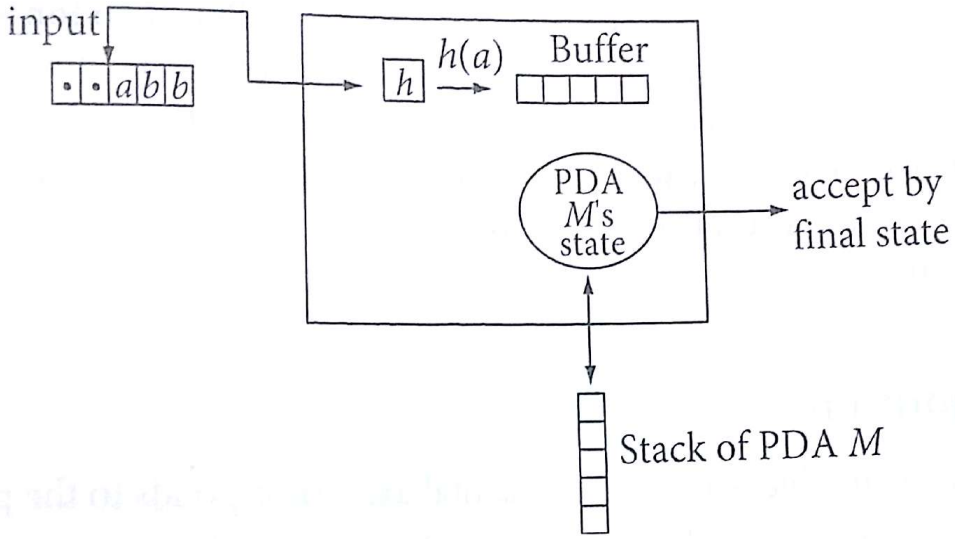
Remark CFLs are not closed under difference i.e., $L_1 - L_2$ need not be a CFL for all CFLs L_1 and L_2. If for all CFLs L_1 and L_2 if $L_1 - L_2$ where a CFL, then taking L_1 as Σ^* we get $\Sigma^* - L_2$ is a CFL for all CFL's L_2, which we know is not true.

Even though intersection of two CFLs need not be context-free, an attempt is made to look for the closure under intersection of a CFL with a regular language. This is a weaker claim.

Theorem 8.6 *If L is a CFL and R is a regular language, then $L \cap R$ is a CFL.*

Proof *Let* $M = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a pushdown automata (PDA) such that $T(M) = L$ and let $A = (\bar{K}, \Sigma, \bar{\delta}, \bar{q}_0, \bar{F})$ be a DFSA such that $T(A) = R$. A new PDA M' is constructed by combining M and A such that the new automaton simulates the action of M and A on an input parallely. Hence, the new PDA M' will be as follows: $M' = (K \times \bar{K}, \Sigma, \Gamma, \delta', [q_0, \bar{q}_0], Z_0, F \times \bar{F})$ where $\delta'([p, q], a, X)$ is defined as follows: $\delta'([p, q], a, X)$ contains $([r, s], \gamma)$ where $\bar{\delta}(q, a) = s$ and $\delta(p, a, X)$ contains (r, γ).

Clearly for each move of the PDA M', there exists a move by the PDA M and a move by A. The input a may be in Σ or $a = \varepsilon$. When a is in Σ, $\bar{\delta}(q, a) = s$ and when $a = \varepsilon, \bar{\delta}(q, a) = q$, i.e., A does not change its state while M' makes a transition on ε.



□

Hence the theorem.

Since the family of CFL is closed under the six basic operations of union, concatenation, Kleene closure, arbitrary homomorphism, intersection with regular sets, and inverse homomorphism, the family of CFL is a full abstract family of languages (AFL). It is also a full trio.