Unit-4: Context Free Grammar

-Ankit Pangeni

DATE

The term grammar applied to a language refers to the mechanism for constructing phrases & sentences that belong to a language. It consists a set of rules, by which strings in a language can be generated.

Context free Grammars (CFGs)

CFGs are used to describe context-free language. It is a set of recursive rules used to generate patterns of strings.

Formally a context free grammar is defined by 4-tuples (V, T, P, S) where,

Components of CFG

- V: finite set of variables (Non Terminal). (In capital letters)
- T: finite set of Terminals ($V \cap T = \phi$) (small letters)
- P: Production Rules (substitution Rules) [$\alpha \to \beta$ where $\alpha \in V$ which is only one variable, $\beta = \{V \cup T\}^*$]
- S: Start variable.

Here,

P is the set of production rules which is used in replacing non-terminal symbols 'α' on the left side of production in a string with other terminal or non-terminal symbols ie. 'β' on the right side of production

S is the start symbol used to derive the string.

DATE

Example:
Rule 1: $S \to \epsilon$
Rule 2: $S \to 0S1$

This is a CFG defining the grammar of all the strings with equal no of 0's followed by equal no of 1's.

Here, two rules define the production P.
$\epsilon, 0, 1$ are the terminals defining T, S is a variable symbol defining V, And S is start symbol where production starts

Formally, grammar is defined as $\{ V=\{S\}, T=\{\epsilon, 0, 1\}, P=\{S \to \epsilon, S \to 0S1\}, S=\{S\} \}$

Or

$P=\{S \to \epsilon | 0S1\}$

Q1: CFG for language $0^n 1^n$ where $n \geq 0$

$\Rightarrow$ $S \to \epsilon$
$S \to 0S1$

$S \overset{(2)}{\Rightarrow} 0S1$
$\overset{(2)}{\Rightarrow} 00S11$
$\Rightarrow 0011$

Q2: Construct CFG for the language having any no. of a's over the set $\Sigma=\{a\}$

$\Rightarrow$ $\Sigma=\{a\}$
$L=\{\epsilon, a, aa, aaa, \ldots\}$
$R.E = a^*$

So, production rules are

$S \to \epsilon$
$S \to Sa$

PAGE

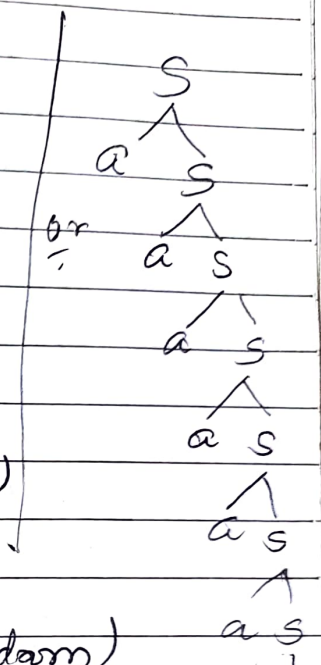
DATE

Rule 1: $S \to \epsilon$
Rule 2: $S \to aS$

$\therefore V = \{S\}$
$T = \{a, \epsilon\}$
$S = \{S\}$

Let's derive a string "aaaaaaa"

$S \Rightarrow aS$
$\Rightarrow aaS$ $(S \to aS)$
$\Rightarrow aaaS$ $(S \to aS)$
$\Rightarrow aaaaS$ $(S \to aS)$
$\Rightarrow aaaaaS$ $(S \to aS)$
$\Rightarrow aaaaaaS$ $(S \to aS)$
$\Rightarrow aaaaaaa$ $(S \to \epsilon)$



Ex. Construct a CFG for language (Pavidam)
$L = \{ wcw^R \mid \text{where } w \in (a+b)^* \}$
string, reverse string.

or

$L = \{ wcw^T \mid \text{---} \}$

$\Rightarrow L = \{ aca, aacaa, bcb, abcba, abbcbba, \text{------} \}$

The Production rules are:
Rule 1: $S \to asa$
Rule 2: $S \to bsb$
Rule 3: $S \to c$

Lets derive: abbcbba

$S \xrightarrow{(1)} asa$
$(2) \Rightarrow abSba$
$(2) \Rightarrow abbSbba$
$(3) \Rightarrow abbcbba$

DATE

Ex:- Construct a CFG for the language $L = a^n b^{2n}$ where $n \geq 1$

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

The production rules are:

Rule 1: $S \rightarrow a\,S\,bb$

rule 2: $S \rightarrow abb$

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

Ex: any no. of a's and b's

$\Rightarrow$ $L = \{\epsilon, a, b, aa, ab, \ldots\}$

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

The production rules (P) are:

$S \rightarrow aS$

$S \rightarrow bS$

$S \rightarrow \epsilon$

Or $S \rightarrow aS \mid bS \mid \epsilon$

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

Ex: at least 2 a's

$\Rightarrow$ $L = \{aa, aba, aaab, aaaabb, \ldots\}$

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

The production rules are:

$S \rightarrow AaAaA$

$A \rightarrow aA \mid bA \mid \epsilon$

DATE

Ex: One occurance of 000 over $\Sigma = \{0,1\}$

$\Rightarrow L = \{000, 0001, 00001, 11000, \ldots\}$

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

The production rules are.

$S \Rightarrow A\,000\,A$

$A \rightarrow 0A \mid 1A \mid \epsilon$

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

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

$S \rightarrow aSb$

$S \rightarrow ab$

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

$\Rightarrow$

$S \rightarrow aSc \mid aAc$

$A \rightarrow bA \mid b$

Ex: $L = \{a^n b^m c^n d^n \mid n \geq 1, m \geq 1\}$

$\Rightarrow S = aSd \mid aAd$

$A \rightarrow bAc \mid bc$

Ex: $L = \{a^m b^m c^n \mid m, n \geq 0\}$

Ex: $L = \{a^n b^m c^m d^n \mid n, m \geq 1\}$

$\Rightarrow$ ~~$S \rightarrow aSb \mid a$~~

$S \rightarrow AB$

$A \rightarrow aAb \mid ab$

$B \rightarrow cBd \mid cd$

* $L = \{a^n b^{n+2}, n \geq 0\}$

$\Rightarrow S \rightarrow aSb \mid bb$

* $L = \{a^{2n+3} b^n, n \geq 0\}$

$\Rightarrow S \rightarrow aaSb \mid aaa$

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

$\Rightarrow S \rightarrow AS_1$

$S_1 \rightarrow aS_1b \mid \epsilon$

$A \rightarrow aA \mid \epsilon$

* $L = \{a^{m+n} b^m c^n \mid n, m \geq 1\}$

$\Rightarrow L = \{a^m a^n b^m c^n \ldots$

$S \rightarrow aSc \mid aAc$

$A \rightarrow aAb \mid ab$

classmate

PAGE

Ex: CFG for string having length atleast two

→ R.E = $(0+1)(0+1)(0+1)^*$

$S \rightarrow AAB$

$A \rightarrow 0 \mid 1$

$B \rightarrow 1S \mid 0S \mid \epsilon$

DATE

Uses of CFG

→ Context-free grammars are used to describe context-free languages.

→ CFGs are studied in the fields of theoretical computer science, compiler design, and linguistics.

→ CFGs are used to describe programming languages and parser programs in compilers can be generated automatically from context-free grammars.

→ Document Type Definition in XML is a context free Grammars which describes the HTML tags & the rules to use the tags in a nested fashion.

Context Free Language (CFL) (Language of Grammar)

A language L is a context free language if there exists a context free Grammer G such that $L = L(G)$.

where $L(G) = \{ x \in T^* \mid S \rightarrow^* x \}$

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

The set of all context Free Language is identical to the set of languages accepted by pushdown automata, & the set of regular languages is a subset of context-free languages.

CFL & CFGs have applications in Computer science and linguistics such as natural language processing and computer language design.

PAGE

DATE

* Derivation Using Grammar rule.

The process of producing strings using the production rules of grammar is called derivation.

Formally, a derivation of a context free grammar is a finite sequence of strings $\beta_0, \beta_1, \beta_2 \ldots \beta_n$ such that:

- For $0 \leq i \leq n$, the string $\beta_i \in (V \cup T)^*$
- $\beta_0 = S$
- For $0 \leq i \leq n$, there is a production of P that applied to β_i yields β_{i+1}.
- $\beta_n \in T^*$

There are two possible approaches of derivation.

i) Body to head (Bottom up) approach.
ii) Head to body (Top down) approach. [leftmost, rightmost]

* Body to head (Bottom up) approach.

Here, we take strings known to be in the language of each of the variables of the body, concatenate them in the proper order, with any terminals appearing in the body and infer that the resulting string is the language of the variables in the head.

DATE

Ex: Consider grammar $S \to S+S \mid S/S \mid (S) \mid S-S \mid S*S \mid a$

Here to generate $a+(a*a) \mid a-a$

S.N.	String Inferred	Variable	production	String Used
1.	a	S	S→a	a; String 1
2.	a*a	S	S→S*S	a*a; String 2
3.	(a*a)	S	S→(S)	String 1 & 2; String 3
4.	(a*a)/a	S	S→S/S	String 1 & 3; String 4
5.	a+(a*a)/a	S	S→S+S	String 1 & 4; String 5
6.	a+(a*a)/a-a	S	S→S-S	String 1 & 5; String 6

Thus, in this process we start with any terminal appearing in the body and use the available rules from body to head.

* Head to body (Top down) approach

Here, we use production from head to body. We expand the start symbol using a production whose head is the start symbol. Here we expand the resulting string until all strings of terminal are obtained. Here we have two approaches:

1. Leftmost derivation.
2. Right most derivation.

DATE

1. Leftmost derivation.

A derivation in CFG is a leftmost derivation (LMD) if, at each step, a production is applied to the leftmost variable occured in the current string. It is denoted by $\xrightarrow{lm}$

Ex. Derive the string "aabbabba" for leftmost derivatn. using a CFG.

$S \rightarrow aB \mid bA$
$A \rightarrow a \mid aS \mid bAA$
$B \rightarrow b \mid bS \mid aBB$

⇒ Leftmost derivation of the string aabbabba is shown as:
Here, 'S' is start variable, so:

$S \xrightarrow{lm} aB$ (Rule: $S \rightarrow aB$)
$S \xrightarrow{lm} aaBB$ (Rule: $B \rightarrow aBB$)
$S \xrightarrow{lm} aabB$ (Rule: $B \rightarrow b$)
$S \xrightarrow{lm} aabbS$ (Rule: $B \rightarrow bS$)
$S \xrightarrow{lm} aabbaB$ (Rule: $S \rightarrow aB$)
$S \xrightarrow{lm} aabbabS$ (Rule: $B \rightarrow bS$)
$S \xrightarrow{lm} aabbabbA$ (Rule: $S \rightarrow bA$)
$S \xrightarrow{lm} aabbabba$ (Rule: $A \rightarrow a$)

Ex: Consider a grammar defined as
$S \rightarrow S*S \mid S+S \mid S-S \mid S/S \mid (S) \mid a$
Derive the string $a+(a*a)$ using leftmost derivatn

⇒ Leftmost derivation of string $a+(a*a)$ is shown as

DATE

Here, 'S' is start variable. so,

$S \xrightarrow{lm} S+S$ (Rule: $S \to S+S$)
$S \xrightarrow{lm} a+S$ (Rule: $S \to a$)
$S \xrightarrow{lm} a+(S)$ (Rule: $S \to (S)$)
$S \xrightarrow{lm} a+(S*S)$ (Rule: $S \to S*S$)
$S \xrightarrow{lm} a+(a*S)$ (Rule: $S \to a$)
$S \xrightarrow{lm} a+(a*a)$ (Rule: $S \to a$)

2. Rightmost derivation

A derivation in context-free grammar is a RMD if, at each step, a production is applied to the rightmost variable occured in the current string. It is denoted by $\xrightarrow{rm}$

Ex: Derive the string "aabbabba" for ~~left~~ right most derivation using a CFG

$S \to aB \mid bA$
$A \to a \mid aS \mid bAA$
$B \to b \mid bS \mid aBB$

⇒

$S \xrightarrow{rm} aB$ (Rule: $S \to aB$)
$S \to aaBB$ (Rule: $B \to aBB$)
$S \xrightarrow{rm} aaBbS$ (Rule: $B \to bS$)
$S \xrightarrow{rm} aaBbbA$ (Rule: $B \to bA$)
$S \xrightarrow{rm} aaBbba$ (Rule: $A \to a$)
$S \xrightarrow{rm} aabSbba$ (Rule: $B \to bS$)
$S \xrightarrow{rm} aabbAbba$ (Rule: $S \to bA$)
$S \xrightarrow{rm} aabbabba$ (Rule: $A \to a$)

DATE

Ex: Consider a grammar defined as

$S \to S*S \mid S+S \mid S-S \mid S/S \mid (S) \mid a$

Now the rightmost derivation of the string a+(a*a) is shown as:

$S \xrightarrow{rm} S+S$ (Rule: $S \to S+S$)
$S \xrightarrow{rm} S+(S)$ (Rule: $S \to (S)$)
$S \xrightarrow{rm} S+(S*S)$ (Rule: $S \to S*S$)
$S \xrightarrow{rm} S+(S*a)$ (Rule: $S \to a$)
$S \xrightarrow{rm} S+(a*a)$ (Rule: $S \to a$)
$S \xrightarrow{rm} a+(a*a)$ (Rule: $S \to a$)

Ex: Consider a grammar: $S \to aAS \mid a$
$A \to SbA \mid ss \mid ba$

Given a string aaabaaa, give left most & right most derivation

⇒

Leftmost derivation

$S \xrightarrow{lm} aAs$ (1)
$S \xrightarrow{lm} asss$ (1)
$S \xrightarrow{lm} aass$ (1)
$S \xrightarrow{lm} aaaAss$ (1)
$S \xrightarrow{lm} aaabass$ (4)
$S \xrightarrow{lm} aaabaas$ (1)
$S \xrightarrow{lm} aaabaaa$ (4)

Rightmost derivation

$S \xrightarrow{rm} aAs$ (1)
$S \xrightarrow{rm} aAa$ (1)
$S \xrightarrow{rm} assa$ (4)
$S \xrightarrow{rm} asaAsa$ (4)
$S \xrightarrow{rm} asaAaa$ (1)
$S \xrightarrow{rm} asabaaa$ (1)
$S \xrightarrow{rm} aaabaaa$ (4)

classmate

PAGE

DATE

* Sentential forms

- Derivations from the start symbol are called sentential forms.
- That is, given a CFG $G=(V,T,P,S)$, if $S \overset{*}{\Rightarrow} \alpha$ where $\alpha \in (V \cup T)^*$, then α is a sentential form
- If $S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$ where $\alpha \in (V \cup T)^*$, then α is a left-sentential form
- if $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$ where $\alpha \in (V \cup T)^*$, then α is a right-sequential form.

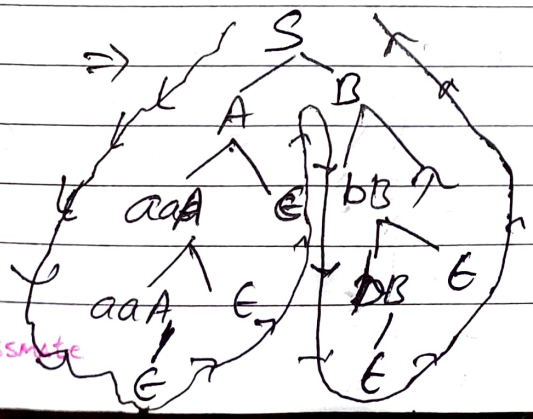
Note:- we already used sentential form in previous topic of leftmost derivation & rightmost derivation.

There are basically two methods: Sentential method, another is tree method.

an example of tree method is:

$S \to AB$
$A \to aaA \mid \epsilon$
$B \to bB \mid \epsilon$

Check whether $w = aaaabb \in L(G)$.



$\therefore aaaabb \in L(G)$

PAGE

DATE

* Parse Tree / Derivation Tree:

A parse tree or parsing tree or derivation tree is an ordered, rooted tree that represents the syntactic structure of an input string according to some context-free grammar.

It is a hierarchical representation of terminals or non terminals. The leaves of parse tree represent terminals and each interior node represents productions of grammar. The in-order traversal gives original input string.

Note: The root is labeled by the start symbol.

Formally, given a context free Grammar (CFG) $G = \{V, T, P, S\}$, a parse tree is a n-ary tree having the following properties:

Ex: Consider the grammar:

$S \rightarrow S*S \mid S+S \mid S-S \mid S/S \mid (S) \mid a$

The parse tree for the string $a+(a*a)$ is shown as follows:

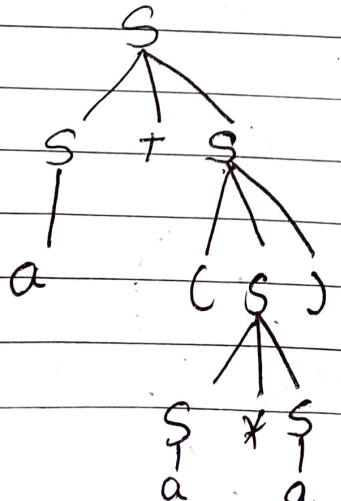


fig: Derivation tree for $a+(a*a)$

DATE

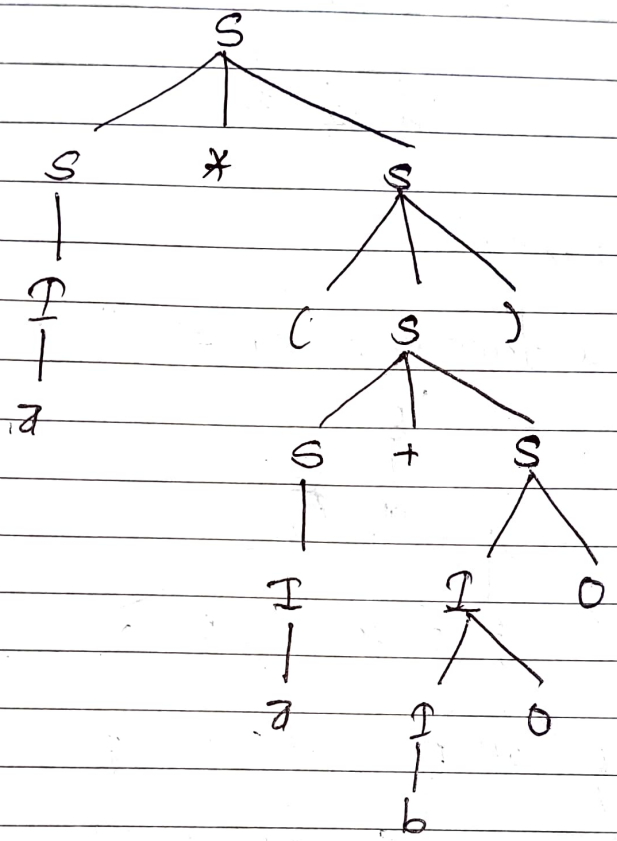
Ex: Consider the Grammar

$S \rightarrow I \mid S+S \mid S*S \mid (S)$

$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Construct the parse tree for $a*(a+b00)$

=>



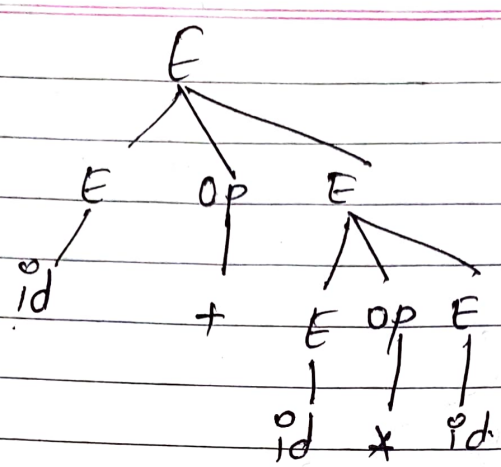
Ex. Construct a grammar defining arithmetic expression and generate parse tree for id+id*id and (id+id)*(id+id)

=> $E \rightarrow E \ OP \ E \mid (E) \mid id$

$OP \rightarrow + \mid - \mid * \mid / \mid \uparrow$

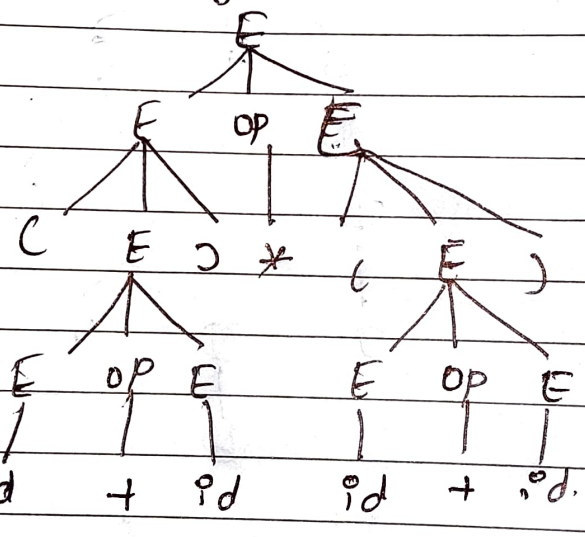
Now, the parse tree for id+id*id is

DATE



id + id * id

Now, parse tree for (id+id)* (id+id)



(id + id) * (id + id

DATE

* Ambiguous grammar

A grammar $G = (V, T, P, S)$ is said to be ambiguous if there is a string $w \in L(G)$ for which we can derive two or more distinct derivation tree rooted at S and yielding the string w

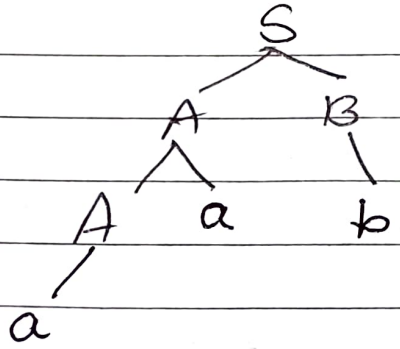
In other words, a grammar is ambiguous if it can produce more than one leftmost and more than one rightmost derivation for the same string in the language of the grammar.

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

For any string aab, we have two leftmost derivations

$S \to AB$
$S \to AaB$ (Rule: $S \to aaB$)
$S \to aaB$ (Rule: $A \to a$)
$S \to aab$ (Rule: $B \to b$)

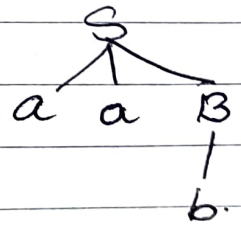
The parse tree for this derivation is:



Also,

$S \to aaB$
$S \to aab$ (Rule $B \to b$)

The parse tree for this derivation is.



Hence, the above grammer is ambiguous grammer

DATE

* Inherently ambiguity

A context free language L is said to be inherently ambiguous if all its grammars are ambiguous. e.g. $L = \{0^i 1^j 2^k \mid i=j \text{ or } j=k\}$

* Regular Grammar

A regular grammar represents a language which is represented by regular expressions. The regular grammar is accepted by NFA and DFA and the language of the grammar is called regular language. A regular grammar is a subset of CFG. The regular grammar may be either left or right linear.

1) Right Linear Regular Grammar

A regular grammar in which all of the productions are of the form $A \to wB$ or $A \to w$ where $A, B \in V$ and $w \in T$ is called right linear.

Ex. $S \to 00B \mid 11C$

$B \to 11C \mid \epsilon \mid 1$

$C \to 00B \mid 00$

2) Left Linear Regular Grammar.

A regular in which all of the productions are of the form $A \to Bw$ or $A \to w$

classmate

PAGE

DATE

Where $A, B \in V$ and $w \in T^*$ is called left linear.

Ex: $S \to B00 \mid C11$
$B \to C11 \mid S \mid 1$
$C \to B00 \mid 00$

* Equivalence Of Regular Grammar and Finite Automata

Let $G = (V, T, P, S)$ be a right linear grammar of a language $L(G)$, we can construct a finite automata M accepting $L(G)$ as:

$M = (Q, T, \delta, [S], \{[\in]\})$

where.

Q = Consists of symbols $\{\alpha\}$ such that α is either S or a suffix from right hand side of a production in P.

T = Set of input symbols Σ which are terminal symbols of G.

$[S]$ = Start Symbol of grammar G. which is start state of FA

$\{[\in]\}$ = Set of final states in finite automata.

δ = transition function defined as

If A is a variable then,

$\delta([A], \in) = [\alpha]$ such that $A \to \alpha$ is a production.

If 'a' is a terminal & α is in $(T \cup V)^*$ then

$\delta([a\alpha], a) = [\alpha]$

Note: No. of states in automata will be equal to no. of non terminals plus one. Each state in automata represents each non-terminal in regular grammar. Additional State will be Final state

PAGE

DATE

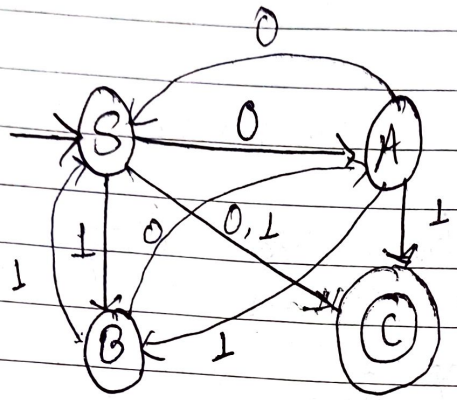
Ex: Convert the following CFG into FA

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

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

$B \to 0A \mid 1S$

$\Rightarrow$



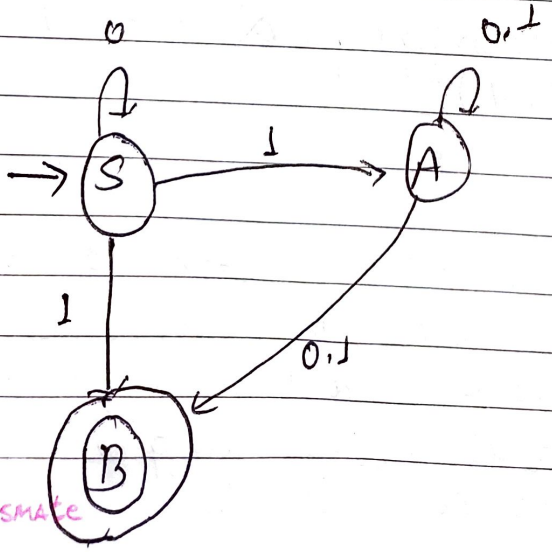
Here, we can see no. of non terminals in CFG = 3 so, no. of states in FA = 3 + 1 = 4.

Note: (imp)

1. For every production $A \to aB$, make $\delta(A, a) = B$ (Make an edge labelled 'a' from A to B)
2. For every production $A \to a$, make $\delta(A, a) =$ Final state
3. For every production $A \to \epsilon$, mark $\delta(A, \epsilon) = A$ and A will be Final state

Ex: $S \to 0S \mid 1A \mid 1$

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



PAGE

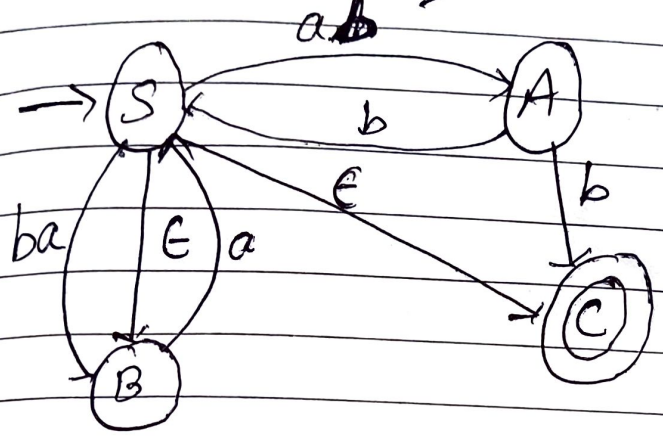
DATE

Ex: Convert the following CFG into FA. Also, trace the transition to accept the string abba

$S \to abA \mid B \mid baB \mid \epsilon$
$A \to bS \mid b$
$B \to aS$

Note: dont write a,b. write ab

=>



Now, to trace the transition to accept string abba.

$$S \xrightarrow{ab} A \xrightarrow{b} S \xrightarrow{\epsilon} B \xrightarrow{a} S \xrightarrow{\epsilon} C$$

∴ The transition required to traverse string abba is
$S \to A \to S \to B \to S \to C$.

Ex: Convert the following CFG into FA.

$S \to 00B \mid 11C \mid \epsilon$
$B \to 11C \mid 11$
$C \to 00B \mid 00$

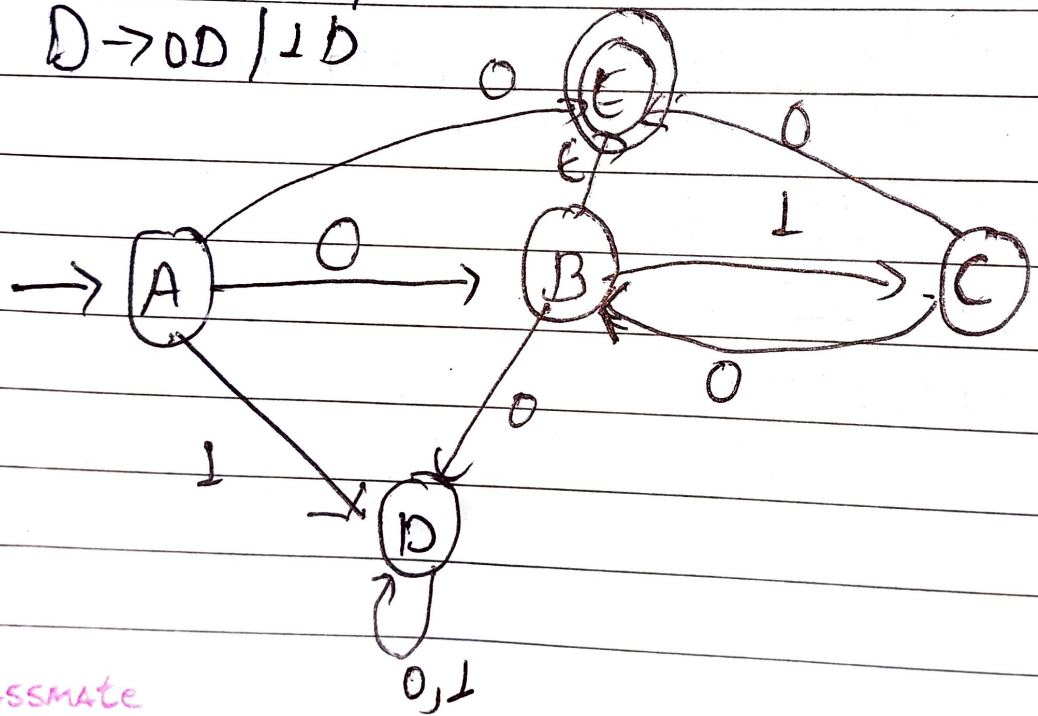
PAGE

Ex: $A \to 0B \mid 1D \mid 0$

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

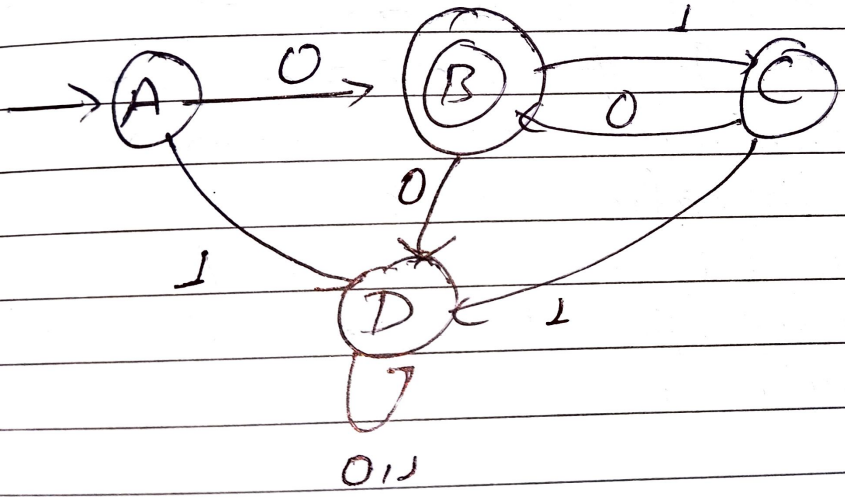
$C \to 0B \mid 1D \mid 0$

$D \to 0D \mid 1D$



DATE

Or



Simplification of CFG

DATE

In a CFG, it may not be necessary to use all the symbols in V and T, or all the productions in P for deriving strings from the grammar. Thus simplification of CFG refers to elimination of those symbols and productions in G which are not useful or are unnecessary for the derivation of strings. It involves following:

i) Removal of useless symbols
ii) Removal of Nullable symbol (ϵ)
iii) Removal of unit production.

Eliminating ϵ-productions / removal of nullable symbols

To eliminate ϵ-productions we have to find nullable variables. Nullable variables are the variables that produce ϵ. ie. Every variable A for which there is a production $A \to \epsilon$ is nullable. If $A_1, A_2, A_3 \ldots, A_k$ are nullable variables and $B \to A_1, A_2, A_3, \ldots A_k$, then B is also nullable variable.

Recursive defn

Algorithm / procedure:

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

Step-2: Replace each occurances of 'A' in each of these productions with B.

Step-3: Add the resultant productions to the Grammar

Note: ① If $A \to \epsilon$, then A is nullable variable.

② Given $X \to Y_1 Y_2 Y_3 \dots Y_n$ If $Y_1, Y_2, Y_3, \dots, Y_n$ are nullable variables then X is also nullable variable.

DATE

Ex: Consider the Grammar:- Remove all null prod.

$S \to AB$

$A \to aAA \mid \epsilon$

$B \to bBB \mid \epsilon$

=> Here

$A \to \epsilon$ A is nullable

$B \to \epsilon$ B is nullable.

$S \to AB$ Since A & B are both nullable S is too

Now, removing ϵ-production, we get.

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

$A \to aAA \mid aA \mid a$

$B \to bBB \mid bB \mid b$

To remove $A \to \epsilon$

$S \to AB \Rightarrow S \to B$

$A \to aAA \mid \epsilon \Rightarrow A \to aA \mid a$

To remove $B \to \epsilon$

$S \to AB \mid B$, $A \to aA \mid a$, $b \to bBB \mid \epsilon$

$aAA \mid$

Ex. Remove Null productions from the following g-

$S \to ABAC$

$A \to aA \mid \epsilon$

$B \to bB \mid \epsilon$

$C \to c$

=> Here

$A \to \epsilon$

$B \to \epsilon$

1) To eliminate $A \to \epsilon$

$S \to ABAC$

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

$A \to aA$

$A \to a$

New productn: $S \to ABAC \mid ABC \mid BAC \mid BC$

$A \to aA \mid a$, $B \to bB \mid \epsilon$, $C \to c$

classmate PAGE

DATE

2) To eliminate $B \to \epsilon$

$S \to AAc \mid Ac \mid (Ac) \mid c$ (no need to write)

$B \to b$

New production:

$S \to AAAc \mid AAc \mid BAc \mid Bc \mid AAc \mid Ac \mid c$

$A \to aA \mid a$

$B \to bB \mid b$

$C \to c$

Ex: Remove null productions from following gram

$S \to AB$

$A \to aAA \mid \epsilon$

$B \to bBB \mid \epsilon$

⇒ Here nullable variables are

$A \to \epsilon$

$B \to \epsilon$

1) To eliminate $A \to \epsilon$

$S \to AB$

$S \to B$

$A \to aAA$

$A \to aA \mid a$

New production.

$S \to AB \mid B$

$A \to aAA \mid aA \mid a$

$B \to bBB \mid \epsilon$

DATE

2) To eliminate $B \to \epsilon$

$S \to AB/B$, $S \to A$

$B \to bBB$

$B \to bB/b$

New production:

$S \to AB/A/B$
$A \to aAA/aA/a$
$B \to bBB/bB/b$

Elimination (removal) of unit productions.

Any production rule of the form $A \to B$ where $A, B \in$ Non Terminals is called unit Productions.

The procedure for Removal

Step-1: To remove $A \to B$, add production $A \to x$ to the grammar rule whenever $B \to x$ occurs in the grammar [$x \in$ Terminal, x can be Null]

Step-2: Delete $A \to B$ from the grammar

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

PAGE

DATE

Example:
Remove unit productions from the Grammar whose production rule is given by
P: S → XY, X → a, Y → Z | b, Z → M, M → N, N → a

⇒ The unit productions are:
Y → Z, Z → M, M → N

1) To remove M → N
Here, M → N and N → a
So, we add M → a

New production rule:
P: S → XY, X → a, Y → Z | b, Z → M, M → a, N → a

2) To remove Z → M
Since, M → a So, we add Z → a

New production rule:
P: S → XY, X → a, Y → Z | b, Z → a, M → a, N → a

3) To remove Y → Z
Since, Z → a So, we add Y → a

New production rule is:
P: S → XY, X → a, Y → a | b, Z → a, M → a, N → a

Since, we can't reach the symbols Z, M, N.

DATE

So, we remove the unreachable Symbol
Final production rule is:-
P: $S \to XY,\ X \to a,\ Y \to a/b$.

Ex. Remove the unit production from following grammar
$S \to S+T \mid T$
$T \to T*F \mid F$
$F \to (S) \mid a$

⇒ The unit productions are: $S \to T,\ T \to F$
Also, $S \to T$ & $T \to F$ so, (S, F) is a unit pair.

Now, after removing unit production, new production rule is:

$S \to S+T \mid T*f \mid (S) \mid a$
$T \to T*F \mid (S) \mid a$
$F \to (S) \mid a$

Ex. Simplify the grammar $G = (V, T, P, S)$ defined by following production
$S \to AS/B \mid \epsilon$
$A \to a\ SA \mid a$
$B \to SbS \mid A \mid bb \mid \epsilon$

⇒ <u>First removing ϵ-production</u>

Here, $S \to \epsilon$ and $B \to \epsilon$ are null variables

DATE

→ To eliminate $S \to \epsilon$

$S \to ASB$
$S \to AB$
$A \to aSA$
$A \to aA$
$B \to SbS$
$B \to Sb$
$B \to bS$
$B \to b$

New production rule is

$S \to ASB \mid AB$
$A \to aSA \mid aA \mid a$
$B \to SbS \mid Sb \mid bS \mid A \mid bb \mid b$

→ To eliminate $B \to \epsilon$

$S \to ASB \mid AB$
$S \to As \mid A$

New production rule is

$S \to ASB \mid AB \mid As \mid A$
$A \to aSA \mid aA \mid a$
$B \to SbS \mid Sb \mid bS \mid A \mid bb \mid b$

DATE

Second, Removing ε-productn

The unit productions are.

$S \to A$, ~~$A \to B$~~ $B \to A$

So, ~~$S \to B$~~ is also unit production by transitn.

Now, new production is.

$S \to ASB \mid AB \mid As \mid aSA \mid aA$

$A \to aSA \mid aA \mid a$

$B \to .Sbs \mid sb \mid bs \mid aSA \mid aA \mid bb \mid a$

Removal of useless symbols (Reduction of grammar)

Note: Symbols that are not derivable or reachable are useless.

CFG are reduced in two phases:-

Phase 1: Derivation of an equivalent grammar G', from the CFG, G, such that each variable derives some terminal string.
(Removal of underivable symbols & their production)

Procedure:-

Step1: Include all the symbols W_1, that derives some terminal and initialize $i=1$

Step2: Include symbols W_{i+1}, that derives W_i

Step3: Increment i and repeat step 2, until $W_{i+1} = W_i$

Step4: Include all production rules that have W_i in it.

(Removable of unreachable symbols & their productions)

Phase 2 : Derivation of an equivalent grammar G_1'', from the CFG, G_1', such that each Symbol appears in a sentential form.

Procedure:

Step 1: Include the Start Symbol in Y_1 and initialize $i=1$
Step 2: Include all symbols Y_{i+1}, that can be derived from Y_i and include all production rules
Step 3: Increment i and repeat Step 2, until $Y_{i+1} = Y_i$

NOTE: all Symbols : both terminals & non-terminals (Variables) not present in Y_{i+1} can be removed.

Example: Find a reduced grammar equivalent to the grammar G, having production rules P
$P: S \to Ac \mid B,\ A \to a,\ C \to c \mid Bc,\ E \to aA \mid e$

⇒ Phase 1: (Removal of Underivable Symbols)
Terminal Symbols $T = \{a, c, e\}$

$W_1 = \{A, C, E\}$
is the set which includes all the symbols that derive the terminal symbols.

$W_2 = \{A, C, E, S\}$
is the set which includes all the symbols that can derive the symbols present in the set W_1.

$W_3 = \{A, C, E, S\}$
" " " " " W_2

$\therefore G' = \{(A, C, E, S), \{a, c, e\}, P, (S)\}$ where,

P: $S \to AC,\ A \to a,\ C \to c,\ E \to aA \mid e$

Phase 1: (Removal of unreachable symbols)

$Y_1 = \{S\}$

$Y_2 = \{S, A, C\}$

in Y_2 are the symbols that can be derived from the symbol in Y_1 (see G')

$Y_3 = \{S, A, C, a, c\}$

in Y_3 are the " " " " symbols in Y_2

$Y_4 = \{S, A, C, a, c\}$

Since $Y_3 = Y_4$ have same symbols. So, we stop here

$G'' = \{(A, C, S), \{a, c\}, P, \{S\}\}$ which is reduced form of grammar G, where:

P: $S \to AC,\ A \to a,\ C \to c$

Q. Let's take another example for phase 2 only:

$S \to aAB \mid bA$

$A \to aAa \mid bCa$

$B \to Ca \mid AC$

$C \to Ab \mid cB$

$D \to aAB \mid dD$

$Y_1 = \{S\}$

$Y_2 = \{S, A, B, a, b\}$

$Y_3 = \{S, A, B, C, a, b\}$

$Y_4 = \{S, A, B, C, a, b, D\}$

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

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

$Y_5 = \{\text{same}\}$

P: S → aAB | bA
A → aAa | bCa
B → Ca | AC
C → Ab | CB

DATE

Note:

Formal definition of reachability.

$A \overset{*}{\Rightarrow} W$, A is non-terminal ($A \in V$)
W is a terminal string ($W \in T^*$)
Such a symbol A is derivable.

$S \overset{*}{\rightarrow} \alpha A \beta$, $\alpha, \beta \in (V \cup T)^*$
S is a start symbol. $A \in V$
Such a symbol A is reachable.

$$S \overset{*}{\Rightarrow} \alpha A \beta \overset{*}{\Rightarrow} W$$

↓ reachability ↓ derivability

Useless productions:

The productions of grammar involving/containing useless symbols are called useless productions. Such productions must be removed from grammar to obtain a simplified or reduced grammar.

PAGE

DATE

Chomsky Normal Form (CNF)

In Chomsky Normal Form (CNF) we have a restriction on the length of RHS; which is; elements in RHS should either be two variables or a Terminal

A CFG is in Chomsky Normal Form if the productions are in the following forms:

$A \to a$

$A \to BC$ Where, A, B, C are non-terminal & a is a terminal

Thus, a grammar in CNF is one which shouldn't have
→ ∈-production
→ Unit production
→ useless symbols

⇒ Steps to convert a given CFG to CNF:

Step1: If the Start Symbol S occurs on some right side, create a new Start Symbol S' and a new production $S' \to S$

Step2: Remove Null Productions.

Step3: Remove Unit Productions.

Step4: Replace each production $A \to B_1 \dots B_n$ Where $n > 2$, with $A \to B_1 C$ where $C \to B_2 \dots B_n$.
Repeat this step for all productions having two or more symbols on the right side.

Step5: If the right side of any production is in the form $A \to aB$ where 'a' is a terminal & A and B are non-terminals, then the production is replaced by $A \to XB$ & $X \to a$. Repeat this for every production $A \to aB$

CLASSMATE PAGE

DATE

Ex: Convert the following CFG to CNF:
P: S → ASA | aB | ε, A → B | S, B → b | ε

⇒ Step-1: Since S appears in RHS, we add a new state S' and S' → S is added to the production rule:

P: S' → S, S → ASA | aB | ε, A → B | S, B → b | ε

Step-2: Remove the Null production.

Removal of A → ε

S → ASA | aB
S → AS | SA | S | aB

New production rule:

S' → S, S → ASA | aB | AS | SA | S |
A → B | S, B → b | ε

Removal of B → ε

S → aB
S → a

A → B | S
A → S

∴ New production rule:

S' → S, S → ASA | aB | a | AS | SA | S
A → B | S, B → b

Step-3: Removing the unit production.

S → S, S' → S, A → B, A → S

After removing S → S: P: S' → S, S → ASA | aB | a | AS | SA
A → B | S, B → b

DATE

After Removing $S' \to S$: P: $S' \to ASA \mid aB \mid a \mid As \mid sA$,
$S \to ASA \mid aB \mid a \mid As \mid sA$
$A \to B \mid S$, $B \to b$

After Removing $A \to B$: P: $S' \to ASA \mid aB \mid a \mid As \mid sA$
$S \to ASA \mid aB \mid a \mid As \mid sA$
$A \to b \mid S$, $B \to b$

After Removing $A \to S$: P: $S' \to ASA \mid aB \mid a \mid As \mid sA$
$S \to ASA \mid aB \mid a \mid As \mid sA$
$A \to b \mid ASA \mid aB \mid a \mid As \mid sA$
$B \to b$

Step: Now find out the productions that has more than TWO variables in RHS.

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

After removing these, we get

P: $S' \to Ax \mid aB \mid a \mid As \mid sA$
$S \to Ax \mid aB \mid a \mid As \mid sA$
$A \to b \mid Ax \mid aB \mid a \mid As \mid sA$
$B \to b$
$X \to SA$

Step: Now Change the productions $S' \to aB$, $S \to aB$ & $A \to aB$

Finally, P: $S' \to Ax \mid YB \mid a \mid As \mid sA$
$S \to Ax \mid YB \mid a \mid As \mid sA$

PAGE

DATE

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

$B \to b$

$X \to SA$

$Y \to a$

Which is the required CNF for the given CFG.

Ex: Remove the nullable symbols:

$S \to ACbB$

$A \to BC$

$B \to b \mid \epsilon$

$C \to D \mid \epsilon$

$D \to d$

⇒ Nullable variables are

$B \to \epsilon$, $C \to \epsilon$, $A \to BC$.

Removing $B \to \epsilon$

$S \to ACbB \mid Acb$

$A \to BC \mid C$

$B \to b$

$C \to D$

$D \to d$

Removing $C \to \epsilon$

$S \to ACbB \mid AbB \mid Acb \mid Ab$

$A \to BC \mid C \mid B$

$B \to b$

$C \to D$

$D \to d$

Removing $A \to BC$ (i.e. $A \to \epsilon$)

$S \to ACbB \mid AbB \mid ACb \mid Ab \mid CbB \mid bB \mid Cb \mid b$
$A \to BC \mid C \mid B$
$B \to b$
$C \to D$
$D \to d$

Q2. Remove the nullable symbols

$S \to aSb \mid aAb$
$A \to \epsilon$

=) Nullable variable is $A \to \epsilon$, removing it

$S \to aSb \mid aAb \mid ab$

Q: Remove the nullable:

$S \to AB$
$A \to aAA \mid \epsilon$
$B \to bBB \mid \epsilon$

Note :- Here $S \to AB$. so $S \leftrightarrow \epsilon$ so, the language itself contains ϵ, so no need to remove $S \to \epsilon$

Removing $A \to \epsilon$

$S \to AB \mid B$
$A \to aAA \mid aA \mid a$
$B \to bBB$

Removing $B \to \epsilon$

$S \to AB \mid B \mid A \mid \epsilon$
$S \to aAA \mid aA \mid a$
$B \to bBB \mid bB \mid b$

Ex: Remove the unit production

$S \to aA \mid B$
$A \to ba \mid bb$
$B \to A \mid bba$

$\Rightarrow$ $S \to aA \mid ba \mid bb \mid bba$
$A \to ba \mid bb$
$B \to ba \mid bb \mid bba$

Removing unreachable symbols.

$\Rightarrow$ $S \to aA \mid ba \mid bb \mid bba$
$A \to ba \mid bb$

DATE

Ex: Convert following grammar into CNF

$S \to aA \mid bB$
$A \to aAAA \mid a$
$B \to bBB \mid b$

⇒ Find out the productions that has more than two variables in RHS.

$A \to aAAA$, $B \to bBB$

After removing these, we get modified rule as:-

$S \to aA \mid bB$
$A \to XY \mid a$
$B \to ZB \mid b$
$X \to aA$
$Y \to AA$
$Z \to bB$

Now, Change the productions:-

$S \to aA$, $S \to bB$, $X \to aA$, $Z \to bB$

Modified rule:

$S \to MA \mid NB$
$A \to XY \mid a$
$B \to ZB \mid b$
$X \to MA$
$Y \to AA$
$Z \to NB$
$M \to a$
$N \to b$

Since it is in CNF. This is the final grammar.

DATE

Greibach Normal Form (GNF)

A CFG is in Greibach Normal form if all its production rules are in the form

$A \to aV^*$
$A \to a.$

where, A is a non terminal
a is a terminal
V is a set of non terminals

Ex. $X \to bABCDE$
$X \to c$

or $X \to bX_1X_2X_3X_4$
$X \to b$

Steps to convert a given CFG to GNF:

Step-1: Check if the given CFG has any unit productions or Null productions and remove if there are any

Step-2: Check whether the CFG is already in CNF and convert it to CNF if it is not.

Step-3: Change the name of the Non-Terminal Symbols into some A_i in ascending order of i

Ex: $S \to CA \mid BB$
$B \to b \mid SB$
$C \to b$
$A \to a$

CNF

Note: $S' \to S$
$S \to XA \mid BB$
$B \to b \mid SB$
$C \to b$
$A \to a$

→ Step 1 & 2 are satisfied already

DATE

Now, replace S with A_1, C with A_2, A with A_3, B with A_4

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-Terminals 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 $i \geq j$

$A_4 \rightarrow b \mid A_1A_4$

$A_4 \rightarrow b \mid A_2A_3 \mid A_4A_4 \mid A_4$

$A_4 \rightarrow b \mid bA_3$

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b \mid A_1A_4$

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

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

Step 5: Remove left recursion. (of the form Ax)

Introduce a new variable to remove the left recursion

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

Introduce. $Z \rightarrow A_4A_4Z \mid A_4A_4$

New production rules:

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

DATE

Now the grammar:

$A_1 \to A_2A_3 \mid A_4A_4$
$A_4 \to b \mid bA_3A_4 \mid bz \mid bA_3A_4z$
$z \to A_4A_4 \mid A_4A_4z$
$A_2 \to b$
$A_3 \to a$

Since A_1 is not still in GNF,

Now, $A_1 \to bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bzA_4 \mid bA_3A_4zA_4$
$A_4 \to b \mid bA_3A_4 \mid bz \mid bA_3A_4z$
$z \to bA_4 \mid bA_3A_4A_4 \mid bzA_4 \mid bA_3A_4zA_4 \mid$
$bA_4z \mid bA_3A_4A_4z \mid bzA_4z \mid bA_3A_4zA_4z$
$A_2 \to b$
$A_3 \to a$

Which is in GNF.

Ex: Convert the following CFG to GNF (Another method)

$S \to SS$
$S \to aSb$
$S \to ab$

If not in CNF, do this method.

⇒ Step-1: Introduce non-terminals A, B for a, b

$A \to a$
$B \to b$

$S \to SS$
$S \to ASB$
$S \to AB$.

Step-2: Introduce an order among non-terminals by renaming them as follows

Note: A Grammar G(V, T, P, S) is left recursive if it has a production in the form $A \to A\alpha \mid \beta$

Where A, β non terminal

α = string composed from $(V \cup T)^*$

$A_1 \to A_1 A_1$

$A_1 \to A_2 A_1 A_3$

$A_1 \to A_2 A_3$

$A_2 \to a$

$A_3 \to b$.

Step-4: Check if the non-terminals are in ascending order.

$A_1 \to \cancel{A_2 A_1 A_3} A_1 A_1$

$A_1 \to A_2 A_1 A_3$

$A_1 \to A_2 A_3$

$A_2 \to a$

$A_3 \to b$

It can be written as

$A_1 \to \cancel{A_2 A_1 A_3} A_1 A_1 \mid A_2 A_1 A_3 \mid A_2 A_3$

$A_2 \to a$

$A_3 \to b$

Step-5: Remove the left recursion, introduce new non terminal Z such that,

$Z \to A_1 Z \mid A_1$

Then, new rule is

$A_1 \to A_2 A_1 A_3 Z \mid A_2 A_3 Z \mid A_2 A_1 A_3 \mid A_2 A_3$

$Z \to A_1 Z \mid A_1$

$A_2 \to a$

$A_3 \to b$

Which is still not in GNF.

$A_1 \to a A_1 A_3 Z \mid a A_3 Z \mid a A_1 A_3 \mid a A_3$

$Z \to a A_1 A_3 Z Z \mid a A_3 Z Z \mid a A_1 A_3 Z \mid a A_3 Z \mid a A_1 A_3 Z \mid a A_3 Z \mid a A_1 A_3 \mid a A_3$

$A_2 \to a$, $A_3 \to b$

DATE

Ex: Remove Left Recursive production from following

$S \to AA \mid 0$
$A \to AAs \mid 0S \mid 1$

⇒ Introduce a new variable z such that

$Z \to Asz \mid As$

Now, the new rules

$S \to AA \mid 0$
$A \to 0Sz \mid 1z \mid 0S \mid 1$
$z \to Asz \mid AS$

Ex: Remove left recursive production.

$E \to E+T \mid T$
$T \to T*F \mid f$
$F \to (E) \mid a$

⇒ Since, there are two left recursions $E \to E+T$ and $T \to T*F \mid F$.
Introduce two new non-terminals Y and Z such that

$Y \to +TY \mid +T$
$Z \to *fz \mid *f$

So, new rule is

$E \to TY \mid T$
$Y \to +TY \mid +T$
$T \to fz \mid f$
$z \to *fz \mid *f$
$F \to (E) \mid a$

Ex: Remove the left recursion and convert it into GNF

$S \to AA \mid 0$

$A \to SS \mid 1$

:) The grammar is already in CNF, so do by 1st method.

Replace S with A_1 A with A_2

$A_1 \to A_2 A_2 \mid 0$

$A_2 \to A_1 A_1 \mid 1$

Alter the rules so that the non-terminals are in ascending order.

$A_1 \to A_2 A_2 \mid 0$

$A_2 \to A_2 A_2 A_1 \mid 0A_1 \mid 1$

Now, removing the left recursion

Introduce new variable z such that

$Z \to A_2 A_1 Z \mid A_2 A_1$

Now, new rule is

$A_1 \to A_2 A_2 \mid 0$

$A_2 \to 0A_1 Z \mid 1Z \mid 0A_1 \mid 1$

$Z \to A_2 A_1 Z \mid A_2 A_1$

It's still not in GNF

$A_1 \to 0A_1 ZA_2 \mid 1ZA_2 \mid 0A_1 A_2 \mid 1A_2 \mid 0$

$A_2 \to 0A_1 Z \mid 1Z \mid 0A_1 \mid 1$

$Z \to 0A_1 ZA_1 Z \mid 1ZA_1 Z \mid 0A_1 A_1 Z \mid 1A_1 Z \mid 0A_1 ZA_1 \mid 1ZA_1 \mid 0A_1 A_1 \mid 1A_1$

(no need to do this) DATE

Now, you can replace A_1 with S and A_2 with A to match with original gramma

$S \to 0SZA \mid 1ZA \mid 0SA \mid 1A \mid 0$
$A \to 0SZ \mid 1Z \mid 0S \mid 1$
$Z \to 0SZSZ \mid 1ZSZ \mid 0SSZ \mid 1SZ \mid 0SZS \mid 1ZS \mid 0SS \mid 1S$

Ex: Remove left recursion and convert into CNF

$S \to AB$
$A \to BS \mid b$
$B \to SA \mid a$

⇒ It is already in CNF so do by 1st method

$A_1 \to A_2 A_3$
$A_2 \to A_3 A_1 \mid b$
$A_3 \to A_1 A_2 \mid a$

Check if non-terminals are in ascending order

$A_1 \to A_2 A_3$
$A_2 \to A_3 A_1 \mid b$
$A_3 \to A_2 A_3 A_2 \mid a$
$A_3 \to A_3 A_1 A_3 A_2 \mid bA_3A_2 \mid a$

∴ new rule:

$A_1 \to A_2 A_3$
$A_2 \to A_3 A_1 \mid b$
$A_3 \to A_3 A_1 A_3 A_2 \mid bA_3A_2 \mid a$

classmate PAGE

DATE

Since, there is a left recursion $A_3 \to A_3A_1 \mid A_3A_2$

So, introduce a new variable z such that

$$z \to A_1A_3A_2\,z \mid A_1A_3A_2$$

So, new rules

$$A_1 \to A_2A_3$$
$$A_2 \to A_3A_1 \mid b$$
$$A_3 \to bA_3A_2\,z \mid az \mid bA_3A_2 \mid a$$
$$z \to A_1A_3A_2\,z \mid A_1A_3A_2$$

It is still not in GNF.

$$A_1 \to A_3A_1A_3 \mid bA_3$$
$$A_2 \to bA_3A_2zA_1 \mid azA_1 \mid bA_3A_2A_1 \mid aA_1 \mid b$$
$$A_3 \to bA_3A_2z \mid az \mid bA_3A_2 \mid a$$
$$z \to A_1A_3A_2z \mid A_1A_3A_2$$

It is still not in GNF

$$A_1 \to bA_3A_2zA_1A_3 \mid azA_1A_3 \mid bA_3A_2A_1A_3 \mid aA_1A_1A_3 \cancel{\mid bA_1A_3 \mid bA_3}$$

$$A_2 \to bA_3A_2zA_1 \mid azA_1 \mid bA_3A_2A_1 \mid aA_1 \mid b$$
$$A_3 \to bA_3A_2z \mid az \mid bA_3A_2 \mid a$$
$$z \to bA_3A_2zA_1A_3A_3A_2z \mid azA_1A_3A_3A_2z \mid bA_3A_2A_1A_3A_3A_2z \mid aA_1A_1A_3A_3A_2z \mid bA_3A_2zA_1A_3A_3A_2 \mid azA_1A_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2 \mid aA_1A_1A_3A_3A_2$$

PAGE

DATE

Backus-Naur Form (BNF)

It is a notation technique used for context free grammar. This is used for specifying the syntax of the language. It can be expressed as

<Symbol> ::= Exp1 | Exp2 | Exp3 ------

Where,

Symbols non-terminals

and Exp is a sequence of symbols (terminals & non-terminals)

Here the concept is similar to CFG, only the difference, instead of using symbol (→) in a production, we use symbol (::=). We enclose all non terminals in only brackets <>.

Ex:

<integer> ::= <digit> | <integer> <digit>

<digit> ::= 0 | 1 | 2 | ---- | 9

Ex: The BNF for identifiers as as:-

<Identifier> ::= <letter or underscore> <Identifier> <symbol>

<letter or underscore> ::= <letter> | -

<symbol> ::= <letter or underscore> | <digit>

<letter> ::= a | b | ---- z

<digit> ::= 0 | 1 | 2 | ---- | 9

Context Sensitive Grammar (CSG)

A CSG is a formal grammar in which left hand sides and right hand sides of any production may be surrounded by a context of terminal and non-terminal symbols.

A formal grammar, $G = (V, T, P, S)$ is context sensitive if all the production 'p' are of the form

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

where $A \in V$

$\alpha \in V$

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

$\gamma \in (V \cup T)^+$

The name context sensitive is explained by α and β that form the context of A & determine whether A can be replaced with γ or not. Thus the production $A \rightarrow \gamma$ can only be applied in contexts where α occurs to left of A & β occurs to right of A.

Ex: $\{a^n b^n c^n \mid n \geq 1\}$ is a context sensitive language defined as:

$S \rightarrow aSBC \mid aBC$

$CB \rightarrow BC$

$aB \rightarrow ab$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

DATE

Chomsky hierarchy

Chomsky hierarchy represents the class of languages that are accepted by the different machine. It was described by Noam Chomsky in 1956

According to Chomsky hierarchy grammar is divided into 4 types as follows:

1. Type 0 is known as unrestricted grammar
2. Type 1 is known as Context-sensitive grammar
3. Type 2 is known as context-free- grammar
4. Type 3 Regular grammar

Here, the restriction of grammar increases from going to type 0 from type 3. The more restriction increases, the more easier the grammar's language becomes and requires less powerful machine. If grammar is simple (type-0), it requires powerful machine as Turing machine to accept the language generated by grammar.

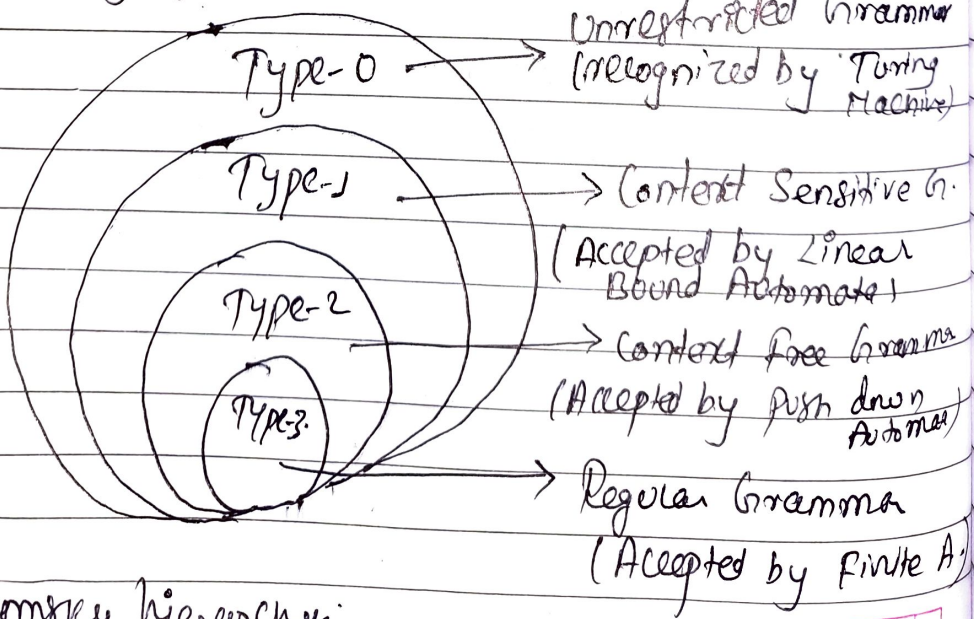


Fig: Chomsky hierarchy

PAGE

DATE

Type-0-grammar (Recursive enumerable grammar / Unrestricted grammar / phase structured grammars.)

→ They include all formal grammar. They are recognized by Turing machine. It is used to generate recursive enumerable language (REL) accepted by TM.

Formally, $G = \{V, T, P, S\}$

where $p: \alpha \to \beta$

where, α : $(V+T)^* V (V+T)^*$

where, V: Variables

T: Terminals.

and β is $(V+T)^*$.

Here, there must be at least one variable on the left side of the production. Ex: $Sab \to ba$

$A \to S$

Variables are S, A, & Terminals. a, b

Type-1-grammar (Context Sensitive grammar / length increasing grammar / non-Contracting grammar)

→

It generates CSL which is accepted by Linear Bounded Automata (LBA).

Here, all type 1 grammar should be Type 0. Grammar production is in the form of: $\alpha \to \beta$ where,

β is $(V+T)^+$ & $|\alpha| \leq |\beta|$

Here, α cannot be $\in$ and length of $\alpha \leq$ length of β

Ex: $S \to AB$ $\quad B \to b$

$AB \to abc$

Type-2-grammar (Context free Grammar)

→ It generates CFG accepted by PDA. It should be type 1. The left-hand side of production can have only one variable & there is no restriction on β.

DATE

Production is in the form of $\alpha \to \beta$ where $|\alpha| = 1$

Ex: $S \to AB$
$A \to a$
$B \to b$

$\alpha \in V$
$\beta \in (V+T)^*$

* Type-3 Grammar
→ It generates regular languages accepted by FA. These languages are exactly all languages that can be accepted by FA. It is the most restricted form of gramma
$G = \{V, T, P, S\}$
It should be in the given form only

$V \to VT/T$ (left-regular grammar)
or
$V \to TV/T$ (right-regular grammar)

Ex: $S \to a$. This is strictly regular gramma

Note: There is another form of regular grammar called extended regular gramm.
$V \to VT^*/T^*$ (extended left-regular gr.
or
$V \to T^*V/T^*$ (extended right reg-gr.)

DATE

Pumping Lemma for Context Free Language (CFL)

If A is a context Free Language, then, A has a pumping length 'p' such that any string 's' where $|s| \geq p$ may be divided into 5 pieces $s = uvxyz$ such that the following conditions might be true.

i) uv^ixy^iz is in A for every $i \geq 0$
ii) $|vy| > 0$
iii) $|vxy| \leq p$

Pumping Lemma (for CFL) is used to prove that a language is NOT context Free.

To prove that a language is Not Context Free using pumping lemma we follow the below steps:-
(we prove using contradiction)

→ Assume that A is context free
→ It has a pumping length (say p)
→ All Strings longer than p can be pumped $|s| \geq p$
→ Now find a string 's' in A such that $|s| \geq p$
→ Divide S into uvxyz
→ Show that $uv^ixy^iz \notin A$ for some i
→ Then consider the ways that S can be divided into uvxyz.
→ Show that none of these can satisfy all the 3 pumping conditions at the same time.
→ S cannot be pumped == CONTRADICTION

Ex: Show that $L = \{a^N b^N c^N \mid N > 0\}$ is not Context free.

⇒ i) Suppose that L is context free

ii) Let pumping length = P exists such that any string $s \in L$ is

$$s = a^P b^P c^P, \quad |s| = 3p > p$$

iii) Let's divide s into uvxyz.

Assume that $p = 4$, then

$$s = a^4 b^4 c^4$$

$$\therefore s = aaaabbbbcccc$$

Now, let's see all possible ways in which we can divide s into uvxyz

Case 1: V and Y each contain only 1 type of symbol

$s = \underbrace{a}_{u}\underbrace{a}_{v}\underbrace{aa}_{x}\underbrace{b}_{y}\underbrace{bbbcc}_{z}cc$

Note: $|vxy| = 9 \Rightarrow p = 4$
Now, make sure $|vxy| \leq p$
but it's not necessary

Case 2: Either V or Y has more than one kind of symbol

$s = \underbrace{a}_{u}\underbrace{aa}_{v}\underbrace{a}_{x}\underbrace{b}_{y}\underbrace{bbb}_{z}cccc$

iv) Now, we show that $uv^i xy^i z \notin L$ for some i

$uv^i xy^i z$ Let's take $i = 2$

Case 1: $s = aaaaaabbbbbccccc \Rightarrow a^6 b^4 c^5 \notin L$

Case 2: $s = aaaaabbaabbbbbcccc \notin L$

Hence, rule isn't satisfied. So, it contradicts our assumption. Hence, it is not context free.

DATE

Ex: Show that $L = \{ww \mid w \in \{0,1\}^*\}$ is not Context free.

=) i) Suppose that L is context free.

ii) Let pumping length = P exists such that any string $s \in L$ is: $S = 0^P 1^P 0^P 1^P$

iii) Let's divide S into uvxyz

Assume that P = 5, then

$$S = 0^5 1^5 0^5 1^5$$

$$S = 00000\,11111\,00000\,11111$$

Now, let's see the possible ways in which we can divide S into uvxyz.

Case 1: vxy does not straddle a boundary.

$\underbrace{00000}_{u}\,\underbrace{11111}_{vxy}\,\underbrace{00000\,11111}_{z}$

Here, $|vxy| = 3 \leq P = 5$

So, condition (3) is satisfied.

$|vy| = 2 > 0$ (condition (2) is satisfied)

Now, we show that $uv^i xy^i z \notin L$ for some i

$uv^i xy^i z$ Lets take $i = 2$

$$S = uv^2xy^2z$$

$$= 0\,00001\,11\,11\,1000\,00111$$

$$= 0^5 1^7 0^5 1^5 \notin L$$

So, it doesn't satisfy condition (i)

Here, case 1 already doesn't satisfy, so no need to check other cases.

So, it contradicts our assumption. And, Hence L Cannot be pumped so it is not Context free.

classmate

PAGE

DATE

Application of pumping lemma in CFL

→ Pumping lemma is used to check whether a grammar is is context free or not.
→ Used in some rules of thumb.

Closure properties of CFL

Given certain languages are context free and a language L is formed from them by certain operation, like union of the two, then L is also context free. These theorem / lemmas are often called closure properties of context free language.

Some of the principle closure properties for context free languages are:

1) Union (closed)

The CFL is closed under union.
Let's suppose two CFL L_1 and L_2 and corresponding grammar are G_1 and G_2. Let's start symbol for G_1 is S_1 and G_2 is S_2.

Let's take a string for S_1 where a followed by equal no. of b. & for S_2 is b followed by equal no. of c (ie. $L_1: a^n b^n$, $L_2: b^n c^n$ n≥

$L_1 \rightarrow G_1$
$L_2 \rightarrow G_2$

$S_1 \rightarrow a S_1 b \mid \epsilon$
$S_2 \rightarrow b S_2 c \mid \epsilon$

Their union is: $S \rightarrow S_1 \mid S_2$

which is closed under union

DATE

iii) Concatenation (closed)

⇒ The CFL is closed under concatenation.

Let's take same example as above.

$S_1 \rightarrow a S_1 b \mid \epsilon$

$S_2 \rightarrow b S_2 c \mid \epsilon$

Then, concatenation: $S \rightarrow S_1 . S_2$

So, it is closed under concatenation

iii) Kleene closure (closed)

⇒ The CFL is closed under kleene closure.

In above example.

$L_1 \rightarrow G_1$

$L_2 \rightarrow G_2$

$L_1^* \rightarrow G_{L_1}$

$S \rightarrow S_1 S \mid \epsilon$

which is closed.

iv) Intersection (not closed)

⇒ The CFL is not closed under intersection.

Let's suppose L_1 is $a^n b^n c^m$

L_2 is $a^m b^n c^n$

Then, $L_1 \cap L_2$ is $a^n b^n c^n \rightarrow CSL$

context sensitive.

So, it can't be context free. So, it is not closed under intersection

v) Complementation (not closed)

⇒ Take above same example: Lets suppose CFL is closed under complement and $L_1 \cap L_2 = \overline{L_1} \cup \overline{L_2}$

Suppose $L_1 \cap L_2$ is closed under complement, then, $\overline{L_1}$ is CFL, $\overline{L_2}$ is CFL, then $\overline{L_1} \cup \overline{L_2}$ is CFL, which is

DATE

Complement becomes CFL.

So, $L_1 \cap L_2$ becomes CFL. Which contradicts the property of intersection.

So, it is not closed.

Questions asked from this Chapter

Q. Explain about Chomsky's Hierarchy about the language and grammars. (2078 - 5 marks) (2071 - 5 marks) (2067 - 5 marks)

Q. Convert the following grammar into Chomsky Normal Form. (2078 - 5 marks)

$S \rightarrow aBSb \mid a \mid aAb$

$A \rightarrow bS \mid aAAb \mid \epsilon$

Q. Define Chomsky Normal Form and Greibach Normal form in reference to CFG. Give a suitable example each. (2076 - 5 marks)

Q. Convert following to CNF: $S \rightarrow abSb \mid aa$. Explain CNF (2071 - 5 marks)

Q. Convert the following CFG into CNF

$S \rightarrow aAa \mid bBb \mid \epsilon$

$A \rightarrow C \mid a$

$B \rightarrow C \mid a$

$C \rightarrow CDE \mid \epsilon$

$D \rightarrow A \mid B \mid ab$

(2073 - 5 marks)

Q. Explain closure properties of CFL with examples. (2069 - 10 marks)

Q. Explain about Unrestricted Grammar. How they differ with CFG? Explain. (2071 - 5 marks, 2067 - 5 marks)

Q. Give a detailed description of ambiguity in context free grammar. (2069 - 5 marks)

Q. Prove that CFG is ambiguous. (2070 - 5 marks)

Q. What is CFG? Define CFG for palindromes with alphabet $\{0,1\}$. (2073 - 5 marks)

Q. Define regular grammar? what is the relation of Regular grammar with other Grammar? (2074 - 5 marks)