Date
26/09/13
Wednesday.

UNIT - II

Context - free Grammars.

* Before the definition of grammar. Two types of sentences in English with a view to formalising the construction of these sentences. The sentences we consider are those with a Noun and a verb, or those $\bar{c}$ a noun - verb and adverb (such as Ram ate quickly)

here word - 'Ram', 'ate', 'quickly'

we can replace Ram by any other person name.

Verb by another verb. & adverb by adverb.

∴ we can get other grammatically correct sentences.

So Ram ate quickly can be given as <noun> <verb> <adverb>

here <noun> <verb> <adverb> is not a sentence but only the description of particular type of sentence.

• we actually get grammatically correct sentences.

Values are called terminals like adverb - slowly ← terminal.

eg: Let S be a variable denoting a sentence. Now, we can form the following rules to generate two types of sentences.

S → <noun> <verb> <adverb>
S → <noun> <verb>
<noun> → Sam / Ram / Rita / Rahul
<verb> → ran / ate / ....
<adverb> → slowly / quickly etc.

} called production rules.

Grammar :→ phrase structure grammar is 4 tuple where $\langle V_N, \Sigma, P, S \rangle$

where

* V_N = is a finite non-empty set whose elements are variables (non-terminals), generally denoted by capital letter.

* Σ = is a finite non-empty set whose elements are called terminals, generally represented by small letter, symbol, operator etc.

* $V_N \cap \Sigma = \phi$

* S is a special variable called start variable.

* P is a finite set whose elements are

  $\alpha - \beta$ where α & β are string on $(V_N \cup \Sigma)$ α has atleast one symbol from V_N

The elements of P are called production | production rules | rewriting rules.

Language generated by grammer ÷ L(G) sentences

G_1 & G_2 are equivalent if $L(G_1) = L(G_2)$

Content free - Grammer.

is a formal grammar in which every production rule is of the form $V \rightarrow w$ where V is single Non Te & w is string of terminals and / or non terminals.

$$A \rightarrow \alpha \quad \text{where} \quad \alpha \in (V_N \cup \Sigma)^*$$

languages generated by content free grammars are known as the content-free language.

Language is set of string generated by a grammar.

- CFG's are theorical basis for the syntax of most prog. languages.

- Type 2 grammer
- automata → push down automata
- Complexity → Quadr
- language L2 : content-free ($a^n b^n$)

Derivation Trees.

* when deriving a sequence of tokens......

- More than one NT may be present and can be expand
- A leftmost Derivatn chooses the leftmost NT to expand
- A Rightmost Derivatn chooses the Rightmost NT —"—

arse Tree | Derivation Tree

- is a graphical representation for a derivation.
- filters out by choice regarding replacement order.
- Rooted by the start symbol.
- interior nodes represent nonterminal
- leaf nodes represent terminal or ϵ
- A node can have children $X_1, X_2 \ldots X_n$ if a rule $A \rightarrow X_1 X_2 X_n$ exists.

$S \rightarrow S+S \mid S*S \mid a \mid b$

Derivation Tree for $a*b + a*b$

$S \rightarrow S+S$

$S \rightarrow S*S+S$

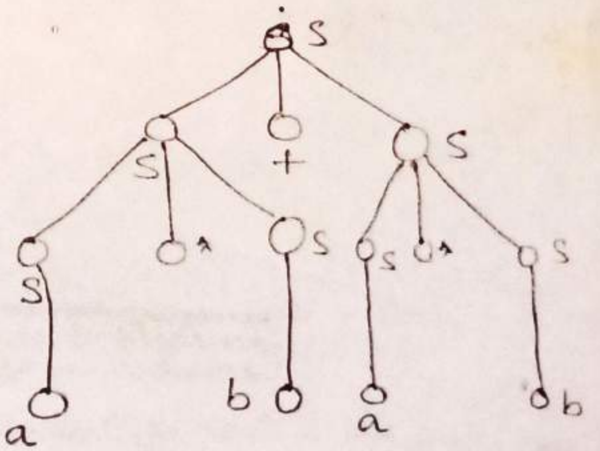
$S \rightarrow a*S+S$

$S \rightarrow a*b+S$

$S \rightarrow a*b+S*S$

$S \rightarrow a*b+a*S$

$S \rightarrow a*b+a*b$



Defn:- Derivatn Tree | Parse Tree are a way to represent the generatn of string in a grammar. They also give info about the structure of string.

Pre-Order and Post Order Traversal.

a parse tree has unique LMD and RMD

LMD is PreOrder Traversal correspond to top-down parsing

* RMD is PostOrder Traversal correspond to bottom up parsing.

Now LMD for $a*b+a*b$

$S \to S+S$

$S \to S*S+S$

$S \to a*S+S$

$S \to a*S+S$

$S \to a*b+S$

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

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

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

RMD for same string.

$S \to S+S$

$S \to S+S*S$

$S \to S+S*b$

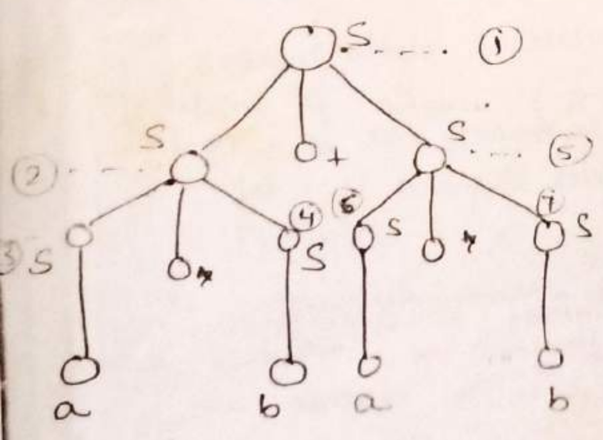
$S \to S+a*b$

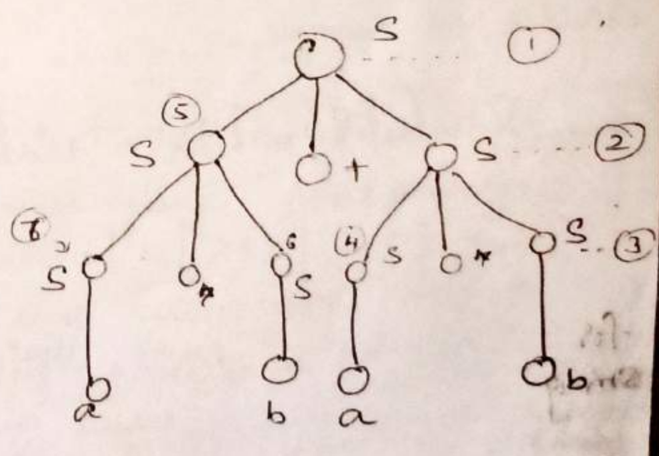
$S \to S*S+a*b$

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

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

Derivatn Tree / parse tree.





Numerical prob. on Ambiguity:

find string that has 2 LMD

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

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

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

2> $S \to Ab \mid aaB$

$A \to a \mid Aa$

$B \to b.$

Ambiguous Grammars

* A grammer is called ambigious if
  - It permits a terminal string to have more than 1 parse
  - This means also, more than 1 LMD for a given string
  - Also, more than one RMD for the same string.
  - Ambiguous grammar should be avoided.
  - No algo, that detects ambiguity in any CFG.
  - To handle ambiguity in expression......
    * The precedence and associativity of operators specify order of evaluatn
    * Higher precedence operators are evaluated 1st.
    * equal precedence operators are evaluated acc to associativity
      * Left to right or Right to left.

Consider, for example.

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

$S \rightarrow S+S \mid S*S \mid a \mid b$

for string $a + a*b$ prove that grammar is ambiguous.

Soln:- To prove grammar is ambiguous we will create two LMD for a given string.

if we found 2 diff LMD then grammar is ambiguous.

1> $S \Rightarrow S+S$

$a+S$

$a+S*S$

$a+a*S$

$a+a*b$

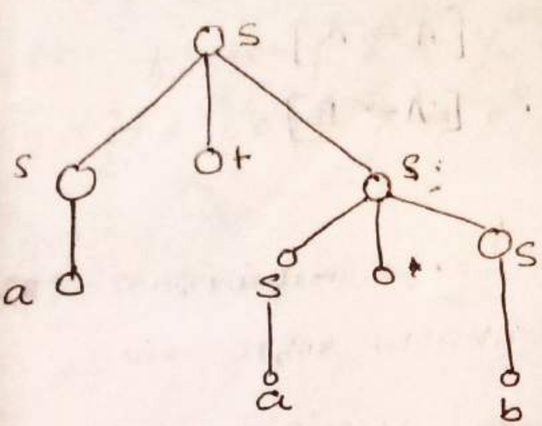
2> $S \Rightarrow S*S$

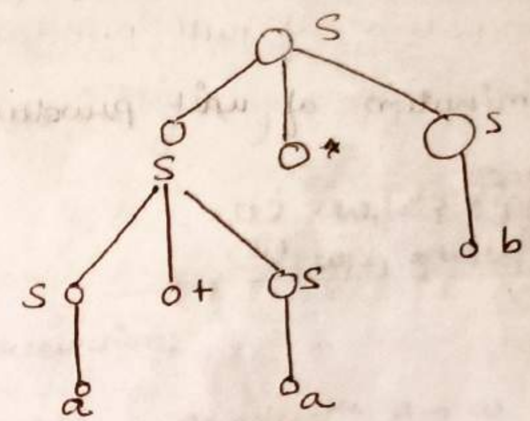
$S+S*S$

$a+S*S$

$a+a*S$

$a+a*b$





Hence Proved.

Several ways to resolve the ambiguity :-

* Rewrite the grammar so that it is no longer ambiguous yet still accepts the same language. (not always possible)
* introduce extra rules that allows the program to decide which of multiple parse Trees to use. These are called disambiguating rules.
* An ambiguous grammar may signal problems with language design and the prog lang itself might be changed.

Date
27 June, 2013

SIMPLIFICATION OF CFG'S

In CFG, & it may not be necessary to use all the symbol in $V_N \cup \Sigma$, or all the productions in P for deriving sentences. So, when we study CF language L(G). we try to eliminate those symbol and production in G which are not useful for derivation of sentences.

* Construction of Reduced grammars
* Elimination of null production $[A \rightarrow \wedge]$
* Elimination of unit production. $[A \rightarrow B]$

Construction of Reduced Grammars

* Theorem:→ If G is a CFG such that $L(G) = \phi$, we can find an equivalent grammar G' such that variable in G' derives some more terminal string.

* Theorem:→ for every CFG $G = (V_N, \Sigma, P, S)$, we can constr an equivalent grammar $G' = (V_N', \Sigma', P', S)$, such that every symbol in $V_N \cup \Sigma'$ appears in some sentential form (i.e. for every X in $V_N' \cup \Sigma'$ there exist α such that $S \underset{G'}{\overset{*}{\Rightarrow}} \alpha$ and X is a symbol in the string α)

(i) $w_1 = \{S\}$
(ii) $w_{i+1} = w_i \cup \{X \in V_N \cup \Sigma\}$

(a) Construction of $\underline{\underline{V_N'}}$ through $\underline{RHS}$,

we define $W_i \subseteq V_N$ by recursion:

$W_1 = \{A \in V_N \mid$ there exists a production $A \to w$ where $w \in \Sigma^*\}$ if $W_1 = \phi$ then $L(G) \neq \phi$

$W_{i+1} = W_i \cup \{A \in V_N \mid$ there exists some production $A \to \alpha$ with $\alpha \in (\Sigma \cup W_i)^*\}$

(b) Construction of P'

$P' = \{A \to \alpha \mid A, \alpha \in (V_N' \cup \Sigma)^*\}$

$V_N' = W_K$

we can define $G' = (V_N', \Sigma, P', S)$

if $S \notin V_N$ then $L(G) = \phi$

$V_N' = V_N \cap W_K$
$\Sigma' = \Sigma \cup W_K$
$P' = \{A \to \alpha \mid A \in W_K\}$

Find a reduced grammar equivalent to the grammar whose production are -

$S \to AB \mid CA$, $B \to Bc \mid AB$, $A \to a$, $C \to aB \mid b$

Step1: $W_1 = \{A, C\}$ as $A \to a$ & $C \to b$ are production with terminal string on RHS.

$W_2 = \{A, C\} \cup \{A_1 \mid A_1 \to \alpha$ with $\alpha \in (\Sigma \cup \{A, C\})^*\}$

$= \{A, C\} \cup \{S\}$ as we have $S \to CA$

$W_3 = \{A, C, S\} \cup \{A_1 \mid A_1 \to \alpha$ with $\alpha \in (\Sigma \cup \{S, A, C\})^*\}$

$W_3 = \{A, C, S\} \cup \phi$

$W_3 = \{A, C, S\}$

As $W_3 = W_2$ $\therefore$ stop here.

Now $V_N' = W_2 = \{S, A, C\}$

$\therefore P' = \{A_1 \to \alpha \mid A_1, \alpha \in (V_N' \cup \Sigma)^*\}$

$= \{S \to CA, A \to a, C \to b\}$

Thus,
$G_1 = (\{S, A, C\}, \{a, b\}, P', S)$

applying other theorem:→ (i.e. proof of reduced grammar)

$W_1 = \{S\}$

as we have production $S \to CA$

and $S \in W_1, W_2$

$W_2 = \{S\} \cup \{A\, C\}$

Now $A \to a$ & $C \to b$ are production with A, C

$W_3 = \{S, A, C, a, b\}$ $\therefore$ G' is the reduced gram

Elimination of NULL production.

Theorem: If $G = (V_N, \Sigma, P, S)$ is a context-free-grammar, then we can find a CFG grammar G_1 having no null production such that $L(G_1) = L(G) - \{\Lambda\}$.

we have to construct $G_1 = \{V_N, \Sigma, P', S)$

Step1: Construction of the set of nullable variables:

(i) $W_i = \{A \in V_N \mid A \to \Lambda \text{ is in } P\}$

(ii) $W_{i+1} = W_i \cup \{A \in V_N \mid$ there exists a production $A \to \alpha$ with $\alpha \in W_i^* \}$

⋮

Step2: construction of P': $A \to X_1 X_2 \dots X_n$ is in P.

the production $A \to \alpha_1 \alpha_2 \dots \alpha_K$ are included in P'

where $\alpha_j = X_i$ if $X_i \notin W$

(ii) gives several production in P'.

The production are obtained either by not erasing any nullable variable on RHS.

Consider the grammar G whose productions are

$S \rightarrow aS \mid AB$
$A \rightarrow \Lambda$
$B \rightarrow \Lambda$
$D \rightarrow b$

Construct a grammar G_1 without null production generating $L(G) - \Lambda$

Solution

Step 1 Construction of the set W of all nullable variables.

$W_1 = \{ A_1 \in V_N \mid A_1 \rightarrow \Lambda \text{ is a production in } G \}$

$\therefore W_1 = \{A, B\}$

$W_2 = W_1 \cup \{ A_1 \in V_N \mid \text{there exists a production } A \rightarrow \alpha \text{ with } \alpha \in W_i^* \}$

$W_2 = \{A, B\} \cup \{S\}$ as $S \rightarrow \underline{AB}$ W_i^*

$W_3 = \{A, B, S\}$

$W_2 = W_3 = \{S, A, B\}$

Step 2 Construction of production:

Eliminate $\boxed{A \rightarrow \Lambda \ \& \ B \rightarrow \Lambda}$ ×

$\therefore$ Production: $S \rightarrow aS \mid AB$
$D \rightarrow b$

Now new production: $S \rightarrow a$
$S \rightarrow A$
$S \rightarrow B$

$$P' = \boxed{\begin{array}{l} S \rightarrow aS \mid AB \\ S \rightarrow a \mid A \mid B \\ D \rightarrow b \end{array}}$$

$\therefore G_1 = (\{S, A, B, D\}, \{a, b\}, P', S)$

Elimination of UNIT Production

Theorem: If G is CFG, we can find a context-free grammar G_1 which has no null production or unit production such that $L(G_1) = L(G)$

Step 1: Prerequisite: Elimination of null production.

Step 2: Construction of the set of variables derivable from A.

Define $W_i(A)$ recursively as follows:

$W_0(A) = \{A\}$

$W_{i+1}(A) = W_i(A) \cup \{B \in V_N \mid C \to B \text{ is in } P \text{ with } C \in W_i(A)\}$

⋮

Step 3: Construction of A-production in G_1.

(i) Remove all production of the form $(NT \to NT)$

(iii) $A \to \alpha$ where $B \to \alpha$ is in G with $B \in W(A)$ and $\alpha \notin V_N$.

Now the required grammar is $G_1 = (V_N, \Sigma, P_1, S)$

Let G be

$S \to AB$
$A \to a$
$B \to C/b$
$C \to D$
$D \to E$
$E \to a$

Eliminate unit productn:

<u>Solution</u> : (i) Elimination of null productn
No, null productn in given grammar.

<u>Step2</u> : Constructn of set of variables

$W_0(S) = \{S\} \cup \phi$ $\quad$ $W_0(B) = \{B\}$

$W_0(A) = \{A\} \cup \phi$ $\quad$ $W_1(B) = \{B\} \cup \{C\} = \{B, C\}$

$W_0(E) = \{E\} \cup \phi$ $\quad$ $W_2(B) = \{B, C\} \cup \{D\} = \{B, C, D\}$

$W_3(B) = \{B, C, D, E\}$

$\Rightarrow W(S) = \{S\}$ $\quad$ $W_4(B) = W_3(B)$

$W(A) = \{A\}$

$W(E) = \{E\}$ $\quad$ $\therefore W(B) = \{B, C, D, E\}$

Similarly, $W_0(C) = \{C\}$ $\quad$ $W_0(D) = \{D\}$

$W_1(C) = \{C\} \cup \{D\}$ $\quad$ $W_1(D) = \{D, E\}$

$W_2(C) = \{C, D\} \cup \{E\}$ $\quad$ $W_2(D) = W_1(D)$

$W_3(C) = \{C, D, E\}$ $\quad$ $W_0(E) = \{E\}$

$W_2(C) = W_3(C)$ $\quad$ $W_1(E) = W_0(E)$

Step2: productn are

$S \to AB$, $A \to a$, $E \to a$

$B \to b/a$, $C \to a$, $D \to a$

By constructn G_1 has no unit productions.

NORMAL FORMS FOR CONTEXT-FREE GRAMMA[R]

* Chomsky Normal form (CNF) Good for Parsing & proving the[orems]
  * Restrict the no. of terminals and non-termin[als] on RHS.
  * The key advantage is that in CNF, every derivatn of a string of n letters has exactly (2n-1) steps.
* Greibach Normal form (GNF)
  * Eliminating left Recursion from the grammar.
  * every CFG can be converted into GNF there is a constructn ensuring that the resulting normal form grammar is of size atmost $O(n^4)$ where n is the size of the original grammar.
  * conversion can be used to prove that every CFL can be accepted by a NDPDA. (NDPA)
  * provides a justification of operatal prefix-notatn usually employed in algebra.

Summary :-

* every CFG that doesn't generate the empty string can be simplified to the CNF and GNF
* The derivation tree in grammar in CNF is a binary tree.
* In the GNF, a string of length n has a derivation of exactly n steps.
* Grammars in normal form can facilitate proofs.
* CNF is used as starting point in the algorithm CYK.

people.cs.clemson.edu/~goddard/texts/theoryofcomputation/9a.pdf

CHOMSKY NORMAL FORM

A CFG is said to be in CNF if every production in grammar is of the form.

1. $A \rightarrow BC$ i.e. N.T. $\rightarrow$ N.T. x N.T.
2. or $A \rightarrow a$ i.e. N.T. $\rightarrow$ single terminal
3. if ε in $L(G)$, then $S \rightarrow \varepsilon$ is a production and S doesn't appear on RHS.

Uses of CNF:→

Allows determination of:

- Membership problem
  - is string w a member of language $L(G)$?
- Emptiness Problem
  - is $L(G) = \phi$?
- ~~Fitness~~ Finiteness problem
  - is language $L(G)$ finite?

Advantage:→

↳ Parsers can use binary tree

↳ Exact length of derivation is known.

used by parsing algorithm (called CYK algo)

Steps to convert a CFG into CNF.

1. Elimination of null production
2. elimination of unit production
3. Convert the grammer into required form.

↳

*· cs.okstate.edu/~kmg/class/5313/fall12/notes/eleven.pdf

For every terminal or α that appears in a body of length 2 or more create a new variable that has only one production.

example:→

$S \to aAbB$

$A \to aA \mid a$

$B \to bB \mid b$

Production Required format

N.T. → N.T. * N.T.

N.T. → single terminal

No Problem with production

$A \to a$ and $B \to b$ directly added to $\boxed{P_1}$

Now, $S \to aAbB$ not allowed

∴ new substitution for terminal symbols. ∴ 2 new production $X_a \to a$ and $X_b \to b$ added to $\boxed{P_1}$

becomes

$S \to \underbrace{X_a A}\,\underbrace{X_b B}$ Now, again substitute

through this

$A \to aA$ becomes $A \to X_a A$

and $B \to bB$ becomes $B \to X_b B$ } added to $\boxed{P1}$

$X_{aA} \to X_a A$

$X_{bB} \to X_b B$ } and ~~$X_b B$~~ added.

∴ $S \to X_a A X_b B$ converted to $S \to X_{aA} X_{bB}$

∴ P_1 Production are :→

$\boxed{S \to X_{aA} X_{bB}}$

$X_{aA} \to X_a A$ $\quad X_a \to a$

$X_{bB} \to X_b B$ $\quad X_b \to b$

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

CNF

$G = \{ V, T, P_1, S \}$

Greibach Normal Form

- A CFG is in GNF if each rule has one these form

i. $A \rightarrow a\alpha$

ii. $A \rightarrow a$

<u>procedure:→</u>

* convert the grammar into CNF I.
* then appy GNF Rule:-

step1: Rename the variable like $A_1, A_2 \ldots A_n$.

Now: write the producth in new name formate.

step2: Remove left factoring

* to eliminate this check the producth

or appy lemma 6.1 to test on c producth

I we eliminate left factoring.

Lemma 6.1 :→ $\boxed{A_i \rightarrow A_j \gamma \text{ , where } j \geq i}$ ← required form of lemma

else convert it into

$A_i \rightarrow A_i \gamma$ such that $j \geq i$

* that means if in any producth

$$A_i \rightarrow A_j \gamma$$

if $i > j$ then convert the producth in form

$$A_i \rightarrow A_i \gamma$$

Now form #

Now apply lemma on the products that we generate from the previous lemma.

we have to eliminate left factoring here by converting it into right one.

i/p
$$\begin{cases} A \rightarrow A\alpha_1 \mid A\alpha_2 \ldots \mid A\alpha_n & \rightarrow \text{eliminate this type of products.} \\ A \rightarrow \beta_1 \mid \beta_2 \ldots \ldots \beta_n & \rightarrow \text{No prob.} \end{cases}$$

Now, we have to change $A \rightarrow A\alpha_1 \ldots \mid A\alpha_n$

Solution: • By adding these new products

$A \rightarrow \beta_1 Z \mid \beta_2 Z \ldots \ldots \mid \beta_n Z$

$Z \rightarrow \alpha_1 Z \mid \alpha_2 Z \ldots \ldots \mid \alpha_n Z$

$Z \rightarrow \alpha_1 \mid \alpha_2 \ldots \ldots \alpha_n$

∴ o/p ⇒
$$\begin{cases} A \rightarrow \beta_1 \mid \beta_2 \ldots \mid \beta_n \\ A \rightarrow \beta_1 Z \mid \beta_2 Z \ldots \mid \beta_n Z \\ Z \rightarrow \alpha_1 Z \mid \alpha_2 Z \ldots \mid \alpha_n Z \\ Z \rightarrow \alpha_1 \mid \alpha_2 \ldots \alpha_n \end{cases}$$

where α's are the $(\Sigma \cup V_N)^*$ starts with A itself

β's are the $(\Sigma \cup V_N)^*$ not starts with A.

* Now convert all the products in the same format by substituting the value of these converted product.

* finally, the grammar we have is in the GNF.

Construct a grammar in GNF (Greibach normal form) equivalent to the grammar →

↳ $S \to AA \mid a$

$A \to SS \mid b$

Step1: Rename the variable

$S = A_1$ and $A = A_2$

∴ production are

$A_i \to A_j$

$A_1 \to A_2A_2 \mid a$ → here $i < j$ then no prob

$A_2 \to A_1A_1 \mid b$ → $i > j$ then convert it

apply lemma 6.1 on $A_2 \to A_1A_1$ by subsituting A_1 value

production

$A_2 \to A_2A_2A_1 \mid aA_1$ Now $i = j$ ∴ no prob

Now if $A_i \to A_i$ then we will apply another lemma 6.2

$A_2 \to A_2A_2A_1 \mid aA_1 \mid b$

here $\alpha = A_2A_1$ $B_1 = aA_1$ $B_2 = b$

production becomes:

$A_2 \to aA_1 \mid b$

$A_2 \to aA_1Z_2 \mid bZ_2$

$Z_2 \to A_2A_1$, $Z_2 \to A_2A_1Z_2$

Now, substituting these values of A_2 in other production all can be converted into required GNF form.

$Z \rightarrow \cancel{aA_1} | b$

$Z_2 \rightarrow A_2 A_1 Z_2$

$\Downarrow$

$Z_2 \rightarrow aA_1A_1Z_2 \mid bA_1Z_2 \mid aA_1Z_2A_1Z_2 \mid bZ_2A_1Z_2$

Now, substituting values in A_1.

$A_1 \rightarrow \underline{A_2 A_2} \mid a$ no prob.

$A_1 \rightarrow aA_1A_2 \mid bA_2 \mid aA_1Z_2 \mid bZ_2A_2 \mid a$

finally, grammar production are —

$A_1 \rightarrow aA_1A_2 \mid bA_2 \mid aA_1Z_2 \mid bZ_2A_2 \mid a$

$A_2 \rightarrow aA_1 \mid b \mid aA_1Z_2 \mid bZ_2$

$Z_2 \rightarrow aA_1A_1Z_2 \mid bA_1Z_2 \mid aA_1Z_2A_1Z_2 \mid bZ_2A_1Z_2$

$Z_2 \rightarrow aA_1A_1 \mid bA_1 \mid aA_1Z_2A_1 \mid bZ_2A_1$

* in lemma 6.1 we try to eliminate Non-terminal from the RHS of a production (like $A \rightarrow A\alpha$)

pg no-206

* lemma 6.2 pg no 206. KLP Mishra.

* Construction of Grammar.

$$L = \{ wcw^T \mid w \in \{a,b\}^* \}$$

L is generated by recursion as follows.

$c \in L$ if $~~x~~ \in L$ then $wxw^T \in L$

$S \to aSa$
$S \to bSb$
$S \to c$

$\therefore S \to aSa$
$absba$
$abcba$

$L = \{ \underset{NA}{aa}, aca, abba, abcba, \ldots$
$\underset{NA.}{bb}, bca, baab, bacab, \ldots \}$

for ww^T

$S \to aSa$
$S \to bSb$
$S \to \wedge$

* Grammar for $a^n b^n$ $n \geq 1$.

$L = \{ab, a\underline{ab}b, aa\underline{ab}bb \ldots\}$

$S \to aSb$

$S \to ab$

* $a^j b^n c^n \mid n \geq 1, j \geq 0$.

$L = \{bc, bbcc, \ldots$

$abc, abbcc, \ldots$

$aabc, aaabc, \ldots\}$.

indicates independent

where b & c are dependent.

pg no. 115

$\therefore$ $S \to AbSc \mid bc$ or $S \to aS \mid A$

$A \to aA \mid \wedge$ $A \to bAc \mid bc$

* Construct a grammar G generating $\{a^n b^n c^n \mid n \geq 1\}$

$S \to aSBC \mid aBC$

$CB \to BC$

$aB \to ab$

$bC \to bc$

every regular grammar is context-free but not all con-
free grammars are regular.

- no inaccessible symbol
- no unproductive symbol
- no ʌ production
- no cycles.

* every pattern language is context sensitive.

<u>Theorem</u>:- Membership problem for Type-0 grammar is undecidable in general, but its decidable given any context sensitive grammar. For CFG the problem is decidable in polynomial time and for regular grammars, linear time.

* formal languages are studied in linguistic and computer sci.
* The human brain contains a limited set of rules for organizing language, known as Universal grammar
* in computer science, formal languages are used for the precise defn of prog. languages and therefore, in the development of compilers.

*

4. Appⁿ :→ theoretical computer Science, theoritical linguis formal semantics, mathematical logic.

Formal Grammar

is a way to describe a formal language. In computer science, it is specifically a generative grammar, usually expressed in top down Backus Normal Form. The syntactically correct strings of the language are generated by free application of grammatical rules of substitution.
A grammar for a language is not unique whether the language be formal or not.
Assume a start symbol S analogous to an empty set, then the substitution or production rules generate new strings from known strings. Cf. Formal Grammar Wikipedia entry for formal definition and examples.

Lindenmayer Systems (L-Systems) Cf. also Wiki

The production rules encapsulate the underlying processes that result in potential states. In their sequences of applications to produce and given string, the sequence is not necessarily unique; different sequences or paths though the string S(A*) may yield the same string. When different paths are possible, the grammar is called ambiguous.

Chomsky Hierarchy of Formal Grammars

The definition of formal languages is so general that the collection as a whole fails to exhibit what one might call "interesting structure". The very same thing happens in topology, where Hausdorff supplies a set of additional axioms of increasing stricture (and structure) to create topological spaces with more specific interest. Chomsky does likewise with formal languages:

Type 0: (more powerful than any other type of grammer)
Unrestricted Grammars are all formal grammars.
They are exactly all those that can be accepted by some Turing machine. Charact. recursively enumberable language.

Type 1: $|\alpha| \leq |\beta|$. $\alpha \to \beta$ no restriction.
Context Sensitive Grammars. recognized by linear bounded auto.
These have rules of the form
aAb → agb $S \to \Lambda$ is allowed if S is not appear on RHS of any rule.
where to make the substitution, the preexistence of prefix 'a' and postfix 'b' is required; that is the 0 context to which the rule is sensitive.

Type 2: (recog. by LR & LL parser) recognized by Non-Deterministic PushDown Auto.
Context Free Grammars. (pumping lemma used to prove not CFG)
These have production rules all of which do not have any of the contextual restrictions of type 1 grammars. used to make parsing easier.
$A \to \alpha$ where $\alpha \in (V_N \cup \Sigma)^*$
• no null production • no cycle.

Type 3: Regular Grammars (Right linear)
Are formal grammar (N, T, P, S) where all the production rules in P are of one of the forms:

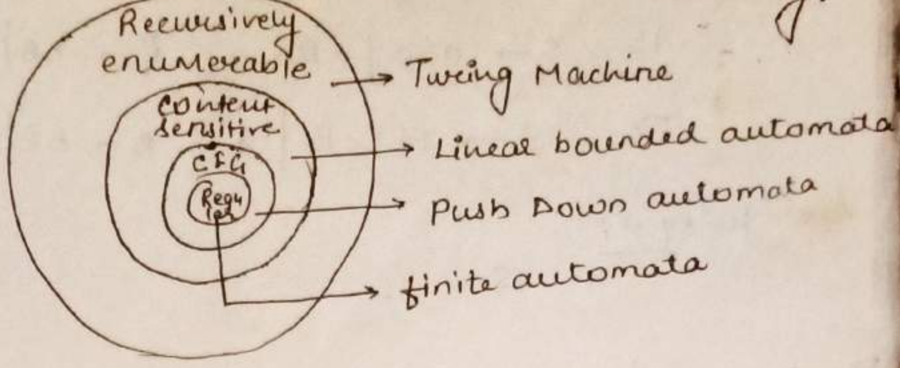
A → a
A → aB or A → Ba
A → ε

Regular languages are recognizable by finite state automata.

→ ~~aaa~~. language generated $A \to \Lambda | a | aB$ nothing else are accepted by finite state m/c than that.
→ commonly expressed using regular expression.
→ to prove lang is not regular then pumping lemma is used

hierarchy :→.

en.wikipedia.org/wiki/chomsky hierarchy.



TYP-0 ⇒ include all formal grammars. They generate exactly all languages that can be recognized by a Turing Machine. These lang are also known as Recursively enumberal languages.

diff from recursive language which can be decided by an always-halting Turing M/c.

recursive lang is b/w Type-0 & Type-1.

recursively enumerable (also recognizable, partially. decidable, semidecidable or Turing acceptable)

recursive language, which requires that the Turing M/c halt in all cases.

Type 0 ⇒ closed under union, intersection, concatenation and kleene closure, but not under complementatn.

Type 2 ⇒ closed under union, concatenation and kleene closure, but not under intersectn or complementatn.

Type 1 ⇒ & Type 3 ⇒ CS & RL are closed under all of the five operatn i.e. union, ∩, concatinatn, kleene closure as well as complementation.

Q. Show that $L = \{0^i 1^i \mid i \geq 1\}$ is not regular.

Solution

Now $L = \{01, 0011, 000111, 00001111, \ldots\}$

that means L is a language that will generate the string with equal no. of zeroes and ones where no. of zeroes are followed by no. of ones.

Now as we have to prove its not regular then we will prove this by contradiction and use of Pumping lem.

Step1: Suppose L is regular. Let n be the no. of states in the finite automaton accepting L.

Step2: we want to find i so that $xy^iz \notin L$.

here $w = 000111$

Now we have to divide string into 3 parts such that $|xy| \leq n$ and $|y| \neq 0$

now here y again has 3 cases

* it may contain only '0'
* it may contain only '1'
* it may contain both 0 & 1 i.e. (01)

Case1 $00(0)^i111 \Rightarrow$ at $i=2$ $00(0)^2111$

$000011 \notin L$

Case2 $000(1)^i111 \Rightarrow$ at $i=2$ $000(1)^211$

$0001111 \notin L$

Case3 $00(01)^i11 \Rightarrow$ at $i=2$ $00(01)^211$

$00010111 \notin L$

$\therefore$ Hence Proved ~~gra~~ L is not regular.

P:- A solution can be found in polynomial Time

NP:- " ———— Varified in polynomial Time

Reduce - problem P to Q. की तुलना में कठिन नहीं

P is "no harder than" Q

How to Transform:

instances of P to instances of Q in polynomial time

such that Q: "yes" iff P "yes"

NP-Hard: Every problem $P \in NP \leq_P Q$

NPC: Q is NP-Hard and $Q \in NP$.