ToppersNotes

GATE

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

VOLUME-VI

COMPILER DESIGN & THEORY OF COMPUTATION

Sierra Innovations Pvt. Ltd.

Contents

COMPILER DESIGN

Chomsky
↓
MIT , MF
dep

Grammer : set of rules

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

Ex→ $G = \left(\underbrace{\{S, A, B\}}_{\text{variable}}, \underbrace{\{a, b\}}_{\text{Ter.}}, \underbrace{\left\{ \begin{array}{l} S \to AB \\ A \to a \\ B \to b \end{array} \right\}}_{\text{Prod.}}, \underbrace{S}_{\text{Start}} \right)$

A/c to Chomsky 4 types of grammers -

1. Type 0
2. Type 1
3. Type 2
4. Type 3

Compiler ⇒ set of rul

Type 0 :
(Unrestricted G.)

$$\boxed{\alpha \to \beta}$$

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

Type 1 :
(CSG)

$$\boxed{\alpha \to \beta}$$

$\alpha, \beta \in (V+T)^*$ where $|\alpha| \le |\beta|$

Type-2 :
(CFG)

$$\boxed{A \to \beta}$$

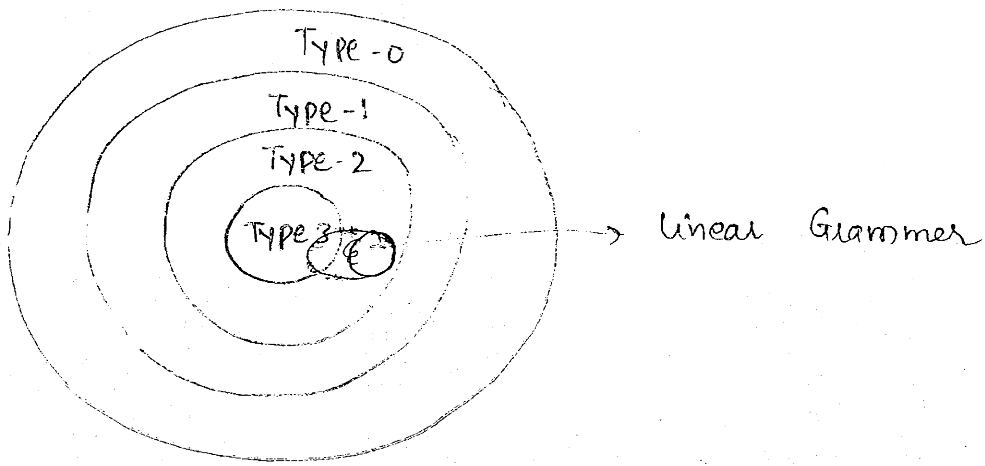
$A \in V$, $\beta \in (V+T)^*$

Type 3 :
(Regular G.)

$A \rightarrow T^* V \mid T^*$ — Right linear

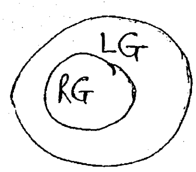
(or)

$A \rightarrow V T^* \mid T^*$ — Left linear



* By default every grammer is Type 0.

Linear Grammer :



$$A \rightarrow T^* V T^* \mid T^*$$

Q→① Consider CFG for the language $L = \{a^m b^n \mid m, n \geq 1\}$

$S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

② $L = \{a^m b^n c^n \mid m, n \geq 1\}$

$S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow bBc \mid bc$

③ $L = \{(a+b)^* \, abb \, (a+b)^*\}$

$S \rightarrow A\,abb\,B$	$S \rightarrow AabbA$	$S \rightarrow AabbA$
$A \rightarrow aA \mid bA \mid \epsilon$	$A \rightarrow \epsilon \mid a \mid b \mid AA$	$A \rightarrow aA \mid bA \mid$
$B \rightarrow aB \mid bB \mid \epsilon$		

④ $L = \{a^i b^{i+j} c^j \mid i, j \geq 1\}$

$S \rightarrow AB$

$A \rightarrow aAb \mid ab$

$B \rightarrow bBc \mid bc$

⑤ L = { Set of all balanced parenthesis }

$S \rightarrow (S) \mid \epsilon \mid SS$

((()))
()()()()
(((((()()))))

Q-> 5 $L_1 = \{a^m b^n c^n \mid m,n \geq 1\} \Rightarrow$ CFG $\Rightarrow$ CFL

$L_2 = \{a^m b^m c^n \mid m,n \geq 1\} \Rightarrow$ CFG $\Rightarrow$ CFL

$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$ $\Rightarrow$ no - CFG $\Rightarrow$ not CFL

CFLs are not closed under intersection.

* ⑦ $L = \{a^i b^j c^k \mid i \neq j \text{ (or) } j \neq k\}$ $i,j,k \geq 1$

$i \neq j$: $i<j$ or $i>j$; $j \neq k$: $j<k$ or $j>k$

$S \to \overset{i<j}{DBC} \mid \overset{i>j}{ADC} \mid \overset{j<k}{AEC} \mid \overset{j>k}{ABE}$

$A \to a \mid aA$ $a+$

$B \to b \mid bB$ $b+$

$C \to c \mid cC$ $c+$

$D \to ab \mid aDb$ $a^n b^n$

$E \to bc \mid bEC$ $b^n c^n$

$L = a^n b^n c^n \Rightarrow$ not CFL

$\bar{L} =$ CFL

} CFLs are not closed under complement.

⊛ Intersection of two CFLs need not be CFL.

Ex→ $L_1 : a^n b^n \mid n \geq 1 \Rightarrow$ DCFL , $L_2 = a^n b^{2n} \mid n \geq 1 \Rightarrow$ DCFL

$L_1 \cup L_2 = \{ a^n b^n \mid a^n b^{2n} , n \geq 1 \}$ ⇒ CFL but not DCFL

⇒ DCFLs are not closed under union.

Ex→ $L = \{ c a^n b^n \mid d a^n b^{2n} , n \geq 1 \} \Rightarrow$ DCFL ✓, CFL ✓

Q→ 8 L = { set of all arithmatic expressions over the alphabet a, b }

~~S → a | b | E+E~~

S → id | E+E | E−E | E/E | E*E | (E)

Q→ 9 L = { Set all palindromes over the alphabet (a, b) }

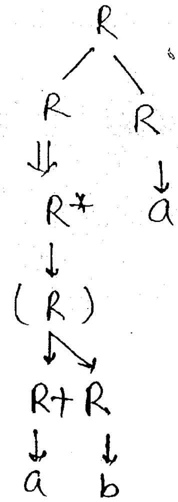
S → aSa | bSb | b | a | ϵ

Q→ 10 L = { set of all regular expressions over the alphabet (a, b) }

~~S → aS | bS | SS | ϵ~~

R → ϵ | a | b | R+R | R·R | (R) | R* |

$(a+b)^* \cdot a \Rightarrow$



(11) L = { set of all boolean expressions }

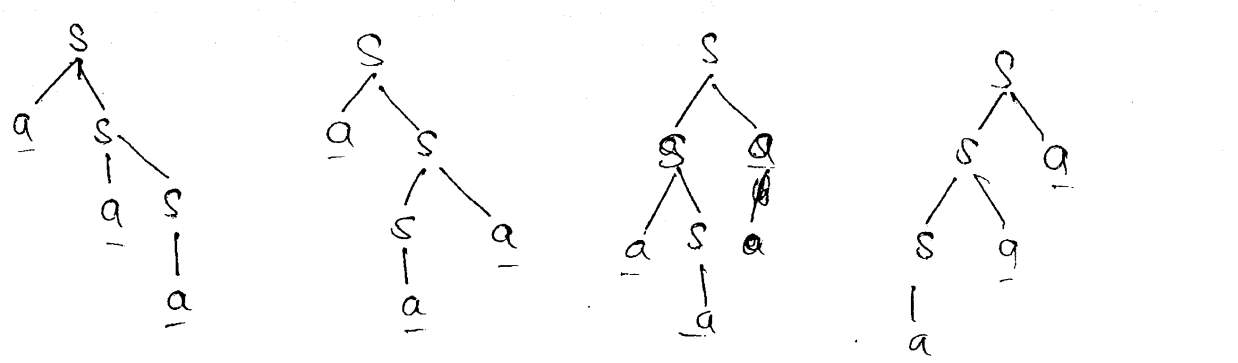
$B \rightarrow T \mid F \mid B \text{ and } B \mid B \text{ or } B \mid \text{not } B \mid (B)$

Q→ ~~Check whether following grammers are ambiguous or not ?~~

Q→① Consider the following grammer -

$S \rightarrow aS \mid Sa \mid a$

i/p: $w = aaa$, how many derivation trees possible for i/p w ?



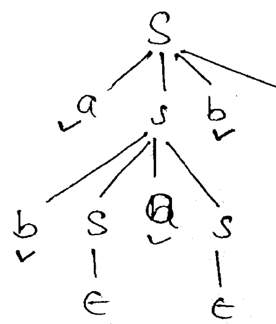
⇒ ~~Ambiguous~~ Above grammer is ambiguous grammer because to derive w=aaa more than one parse trees are possible.

② Check given grammer is ambiguous or not?

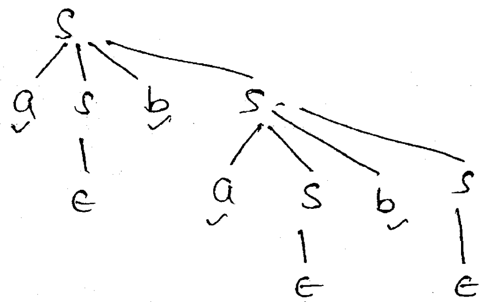
$S \rightarrow aSbS \mid bSaS \mid \epsilon$

$w = abab$

⇒



⇒ abab



⇒ abab

⇒ ambiguous grammer.

⊛ parser → LR parser
→ LL parser

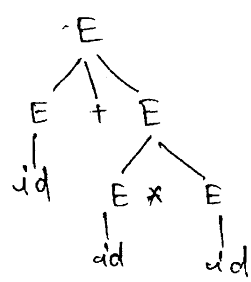
LR → using rightmost derivation
↓
scanning i/p from left to right

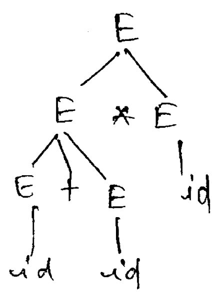
LR → using leftmost derivati
↓
scanning i/p from L to R.

③ $E \rightarrow id \mid E+E \mid E*E$

$w = id + id * id$

⇒





⇒ ambiguous grammer.

Note : ⊛ To check given grammer is ambiguous grammer or not there is no algorithm so it is UNDECIDABLE problem.

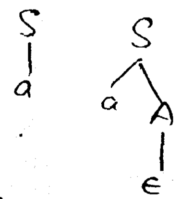
⊛ Every regular grammer may or may not be unambiguous.

$S \to aS \mid a$

⇓

unambiguous

$S \to aA \mid a$
$A \to \epsilon \mid aA$



⇒ ambiguous

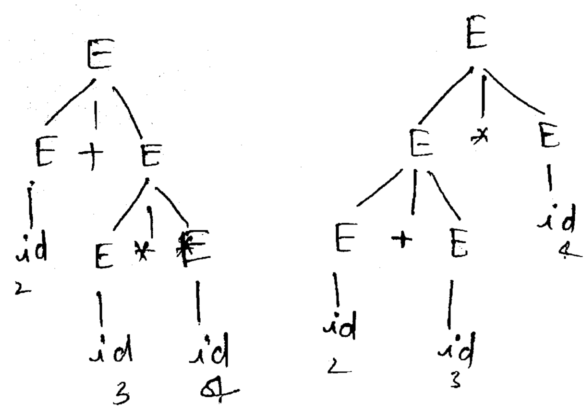
⊛ Every regular language is unambiguous (because at least there exist a Regular grammer which is unambiguous)

Converting ambiguous grammer into equivalent unambiguous grammer :

$E \to id \mid E+E \mid E*E$

$w = id + id * id$
$\quad\;\; 2 + 3 * 4$

→ id is having highest priority.



⇒ 14 ⇒ 20

A/C to C language, * is having highest priority than

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id.$

here * is having highest priority

*, + ⇒ left to right associativity

→ $E \to E+T \mid E-T \mid T$

$T \to T*F \mid T/F \mid T$

$F \to G\uparrow F \mid G$

$G \to (E) \mid id$

priority ( ) > ↑ > * = / > + = −

↑ associativity ⇒ Right to left

$2\uparrow \underline{3\uparrow 2} \Rightarrow 2^{9} = 2$

Q→ Construct unambiguous CFG for the following rules

	priority	Associativity
↑ →	lowest ①	L to R
*, +	②	* : R to L, + : L to R
/	③	R to L
−	④	L to R
		All

⇒ ~~$E \to E-T \mid T$~~

~~$T \to F/T$~~

~~$F \to G*F \mid F+G$~~

$A \to A\uparrow B \mid B$

$B \to C*B \mid B+C \mid C$

$C \to D/C \mid D$

$D \to D-E \mid E$

$E \to id$

Q→ Construct equivalent unambiguous grammer for -

$$R \rightarrow \epsilon \mid a \mid b \mid R+R \mid R \cdot R \mid R^{*} \mid (R)$$

$R \rightarrow R+S \mid S$

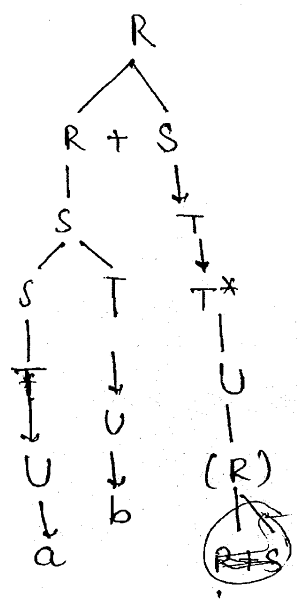
$S \rightarrow S \cdot T \mid T$

$T \rightarrow T^{*} \mid U$

$U \rightarrow (R) \mid \epsilon \mid a \mid b$

highest priority

$a^{*} \cdot b + a$



$\Rightarrow a \cdot b + (a \cdot b)^{*}$

Q→ Construct equivalent unambiguous grammer for -

$$B \rightarrow T \mid F \mid \text{not } B \mid (B) \mid B \text{ and } B \mid B \text{ or } B$$

B→ ~~B or S~~ | S

~~S→~~

$B \rightarrow B \text{ or } C \mid C$

$C \rightarrow C \text{ and } D \mid D$

$D \rightarrow \text{not } D \mid E$

$E \rightarrow (B) \mid T \mid F$.

or
and
not
( )
B

Recursion:

Left Recursion

$A \rightarrow A\alpha \mid T^*$

T^* ⇓ Termination (set of terminals)

Right Recursion

$A \rightarrow \alpha A \mid T^*$

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

→ Every parser will scan i/p from left to right so left recursion creates problem for parser.

$A \rightarrow Aa \mid b$ ⇒ left Recursion

$w = baaa$ ⇒

$A \rightarrow Aa \rightarrow Aa \rightarrow Aa \rightarrow Aa$ ⇐

$A \rightarrow aA \mid b$

$w = aaab$

$A \rightarrow aA \rightarrow aA \rightarrow aA \rightarrow b$

⊛ Because of left recursion some (Top down parsers) parsers are going to be in infinite loop so we have to eliminate left recursion.

Elimination of left Recursion :

Ex→ ① $E \to E+T \mid T$ $\longrightarrow$ $E \to E\underbrace{+T}_{E'} \mid T$ $\Rightarrow T,\ T+T,\ T+T+T, \ldots$

$T \to T*F \mid F$

$F \to id \mid (E)$

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

} converted to right recursion because right recursion dont create problem

$T \to T\underbrace{*F}_{T'} \mid F$

$T \to FT'$

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

② $S \to (L) \mid a$ ✓ ; eliminate left Recursion

$L \to L,S \mid S$

$S \to (L) \mid a$

$L \to SL'$

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

③ $S \to aBDh$ ✓ eliminate left Recursion

$B \to Bb \mid h$ $\Longrightarrow$ $B \to hB'$

$B' \to bB' \mid \epsilon$.

$D \to EF$ ✓

$E \to g \mid \epsilon$ ✓

$F \to f \mid \epsilon$ ✓

Toppersnotes
Unleash the topper in you

(4) Eliminate left Recursion from -

$$A \to Aa \mid Ab \mid Ac \mid Ad \mid e \mid f \mid g$$

$$\Rightarrow \quad A \to eA' \mid fA' \mid gA'$$

$$A' \to aA' \mid bA' \mid cA' \mid dA' \mid \epsilon$$

(5) Eliminate L.R. from -

$$S \to Abc \mid Ade \mid f \mid g$$

$$A \to Aa \mid Ab \mid Ac \mid Ad \mid Se \mid i \mid j \mid k$$

⇒ (4) direct recursions

(2) indirect recursions.

$$\Rightarrow \quad S \to Abc \mid Ade \mid f \mid g$$

$$A \to Aa \mid Ab \mid Ac \mid Ad \mid Abce \mid Adee \mid fe \mid ge \mid i \mid j \mid k$$

⇒ (6) direct Recursion

$$\Rightarrow \quad A \to feA' \mid geA' \mid iA' \mid jA' \mid kA'$$

$$A' \to aA' \mid bA' \mid cA' \mid dA' \mid bceA' \mid deeA' \mid \epsilon$$

Left Factoring :

Ex→ ① $S \rightarrow a\alpha_1 \mid a\alpha_2 \mid a\alpha_3$

$w = a\alpha_3$

From the above grammer to generate string w all the productions are fighting because for the given string first character ⓐ is given by every production. This problem is known as <u>left factoring</u>.

Elimination of left factoring :

$S \rightarrow a\alpha_1 \mid a\alpha_2 \mid a\alpha_3$

$$\Rightarrow \left.\begin{cases} S \rightarrow aS' \\ S' \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \end{cases}\right\} \text{left factored grammer}$$

② $S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b$

$\Rightarrow$ $S \rightarrow \cancel{iEtS} \mid iEtSS' \mid a$

$S' \rightarrow eS \mid \epsilon$

$E \rightarrow b$.

Toppersnotes
Unleash the topper in you

(*) $|ab| = 2$, $\boxed{|\epsilon| = 0}$ $\epsilon \rightarrow$ Zero length string

(3) $S \rightarrow a \mid ab \mid abc \mid abcd \mid e \mid f \mid g$

$\Rightarrow$ $S \rightarrow aS' \mid e \mid f \mid g$

$S' \rightarrow \epsilon \mid bC'$

$C' \rightarrow \epsilon \mid cD'$

$D' \rightarrow \epsilon \mid d$