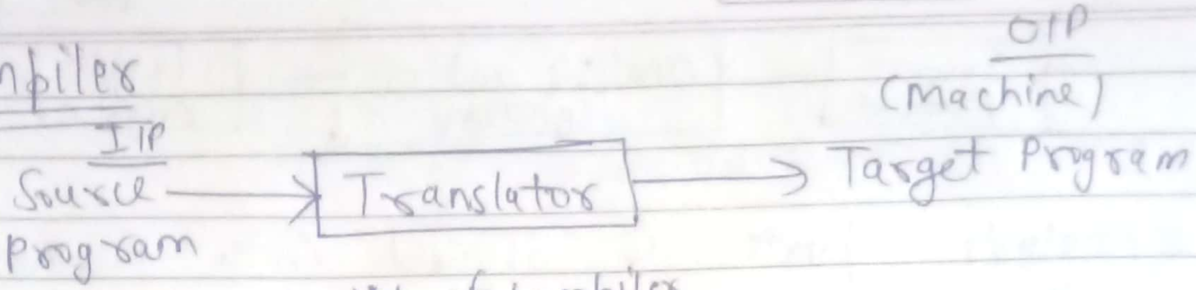
Unit - 1

Ullman

* Compiler



Model of Compiler

Compiler is a translator and it is a program which takes one language as I/P and translates into an equivalent another language.

Assembler is also a translator but the diff b/w these two is in the case of assembler converts the assembly languge into machine language whereas compiler converts ~~machine lang~~ high level language into machine language.

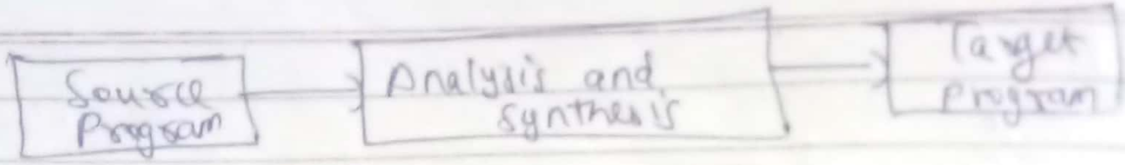
Compiler is also called Analysis and Synthesis Model. This compiler process is called Compilation.

Compilation can be done in two parts—

(i) Analysis

Source Program is read and broken down into smaller pieces. Thereafter the meaning and syntax of the pieces is determined. Then it will generate intermediate code equivalent to their syntax. (source Program)

(ii) Synthesis



Analysis parts is divided into further 3 parts -

(a) Lexical (Tokens)
(b) Syntax (hierarchical)
(c) Semantic

(a) Lexical

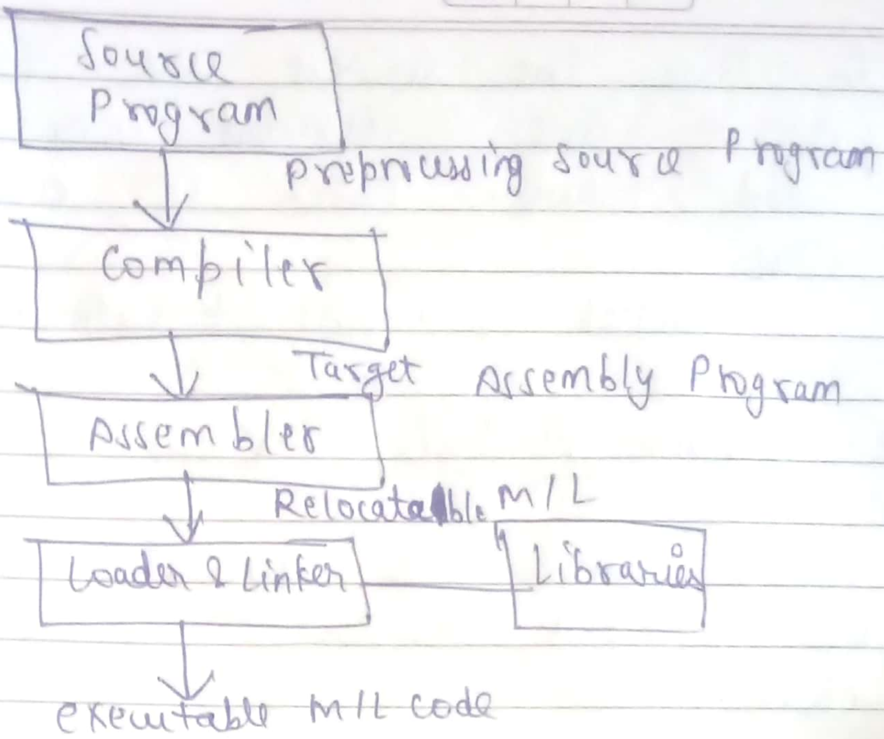
Source Program is broken into stream of strings and they are called token. The collection of character having some meaning.

(b) Syntax

These tokens are arranged in hierarchical structure. (tree)

(c) Semantic

Means the meaning of the tokens is determined with semantic part analysis.



The task of loader is to perform the relocation of object code and the memory allocated to store them.

Compiler must be portable.

Analysis of source Program can be determined by 3 phases -

(a) Linear phase

When the string is reada it will start From Left.

(b) Hierarichical phase

* Phase of compiler

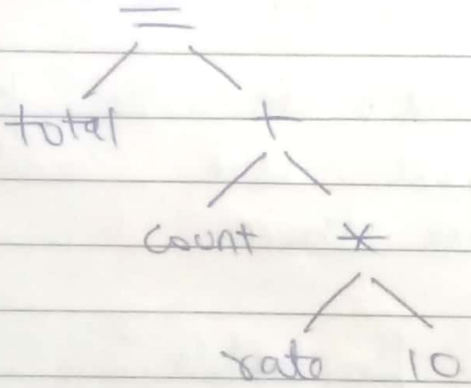
1. Lexical Analysis (scanning)

In this the source are scanned & source program are divided into group of string called tokens. Token is a sequence of charac-cters

total = count + rate * 10

2. Syntax Analysis (Parsing)

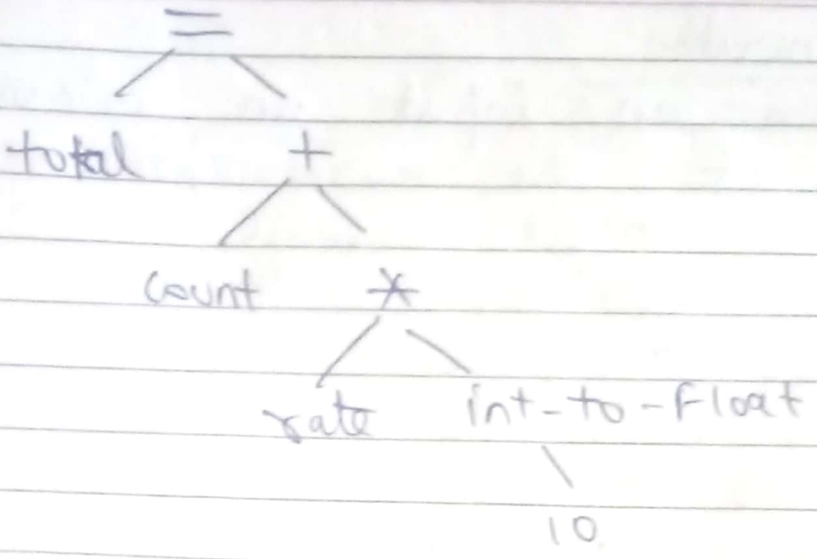
Tokens which are generated in lexical analysis are grouped together to form hierarchy stru-cture



E - Expression
E ⇐ Identifier
E ⇐ $E_1 + E_2$
E ⇐ $E_1 - E_2$
E ⇐ (E)

3. Semantic Analysis

The syntax analyzer uses the syntax tree and the info. in the symbol table to check the source program for semantic consistency with the language definition.



Intermediate code generation

```
total = count + rate * 10
    t1 := int to real (10)
    t2 := rate * t1
    t3 := count + t2
    total := t3
```

Point to be noted about three-address instruction:

(i) Each three-address assignment instructions has at most one operator on the right side

(ii) The compiler must generate a temporary name to hold value computed by a Three-address instruction

(iii) Some "3-address instruction" like the 1st & last above have fewer than 3 operands.

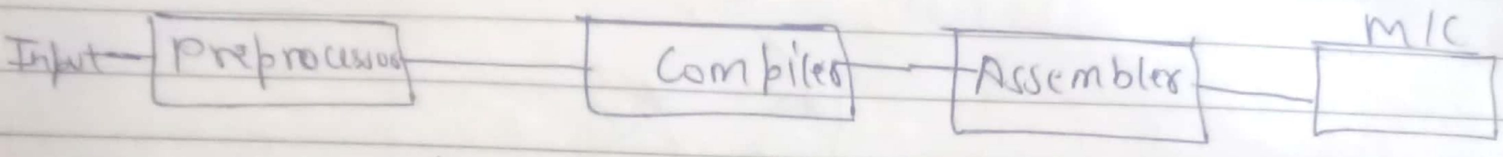
Code optimization

In this phase attempts to improve the intermediate code so that better target code will result.

```
MOV   id2 , R1
 ADD   R2 , R1
 MOV   R1 , id1
```

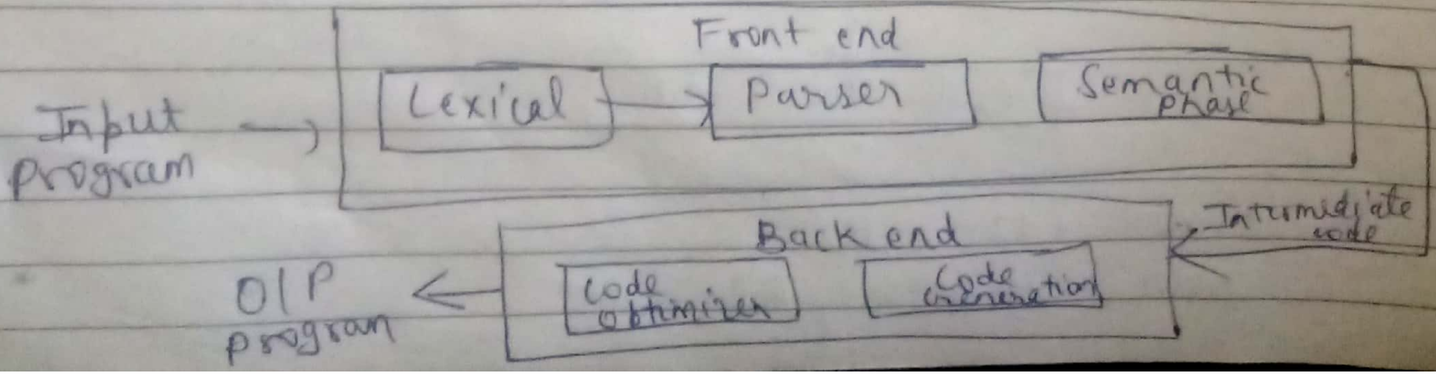
* Cousins of compiler

1. Preprocessor
2. Assembler
3. Linker & Loader



Compiler	Interpreter
→ It is more efficient In this all the code will be compiled at a time	When a source program is modified interpretation will start from 1st line. It is less efficient.
→ It produces the object code	→ Object code is not generated.
→ It is not portable	→ It is ~~not~~ portable
→ It is more complex design Large memory is required.	→ Improved debugging environment.

* Grouping of Phases



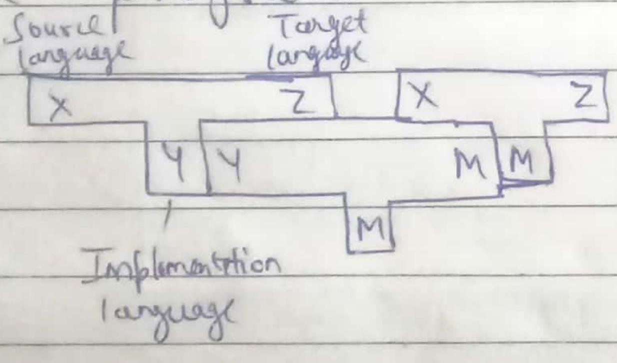
* Passes

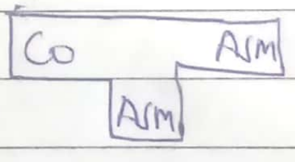
Many phases an be grouped in one pass.

There are many factors which affect no. of passes
(i) Forward reference
(ii) Storage limitation
(iii) Optimization
Compile require more than one pass to complete the execution.

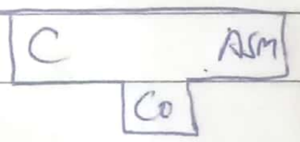
* Boot-strapping

It is a process in which we use the simple language to translate more complicated program

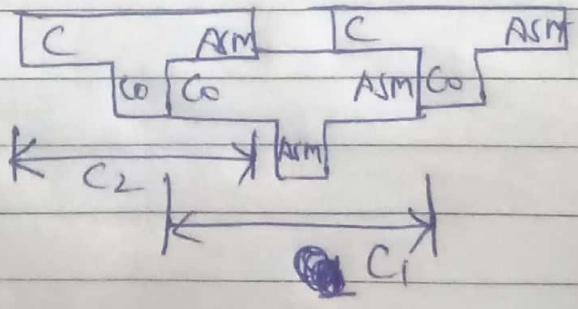




Compiler 1



Compiler 2

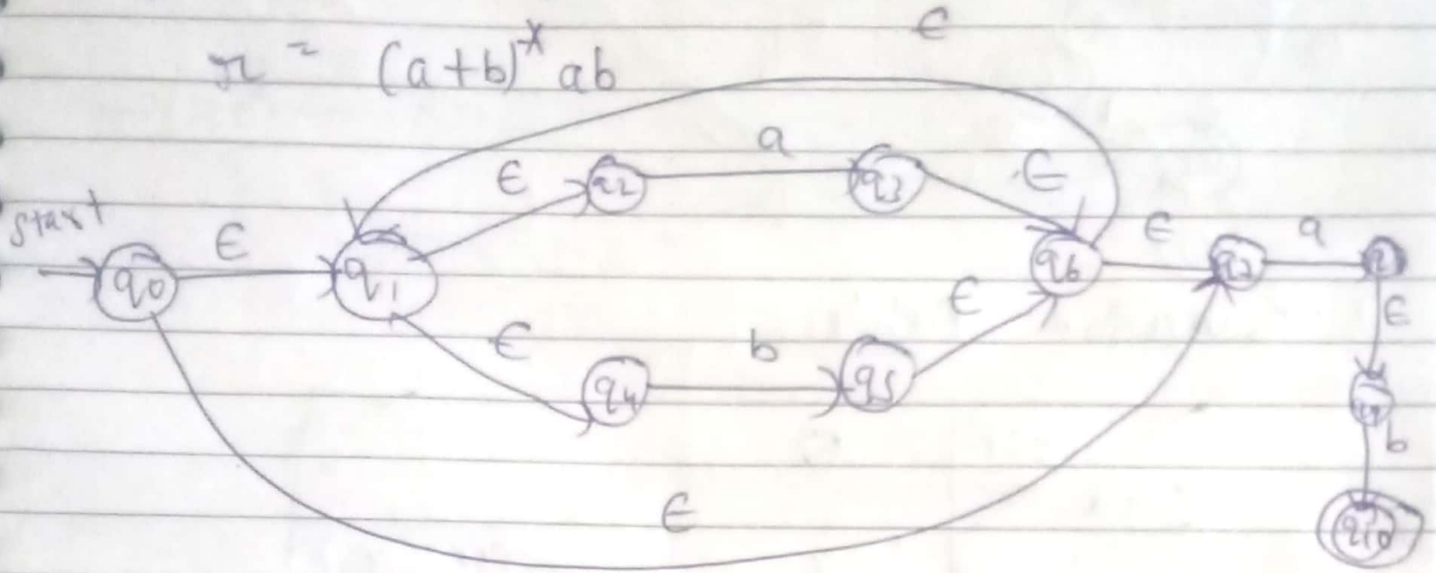


$(a+b)^+$ = not including null

$(a+b)^*$ = including null

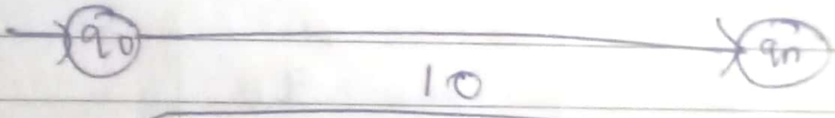
Q Construct a regular expression for the languge containing

$r = (a+b)^* ab$

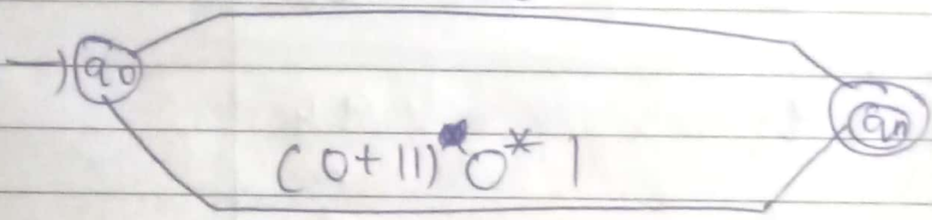


$10 + (0+11)\,0^*\,1$

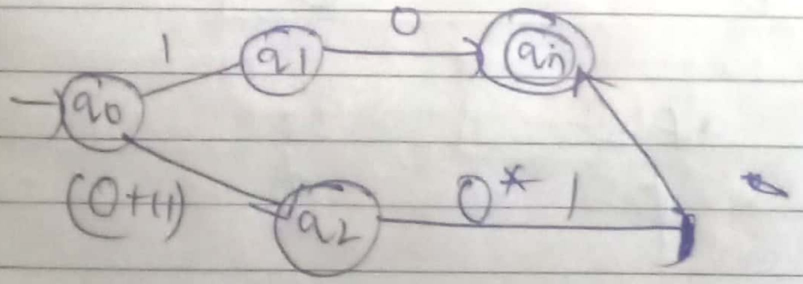
①



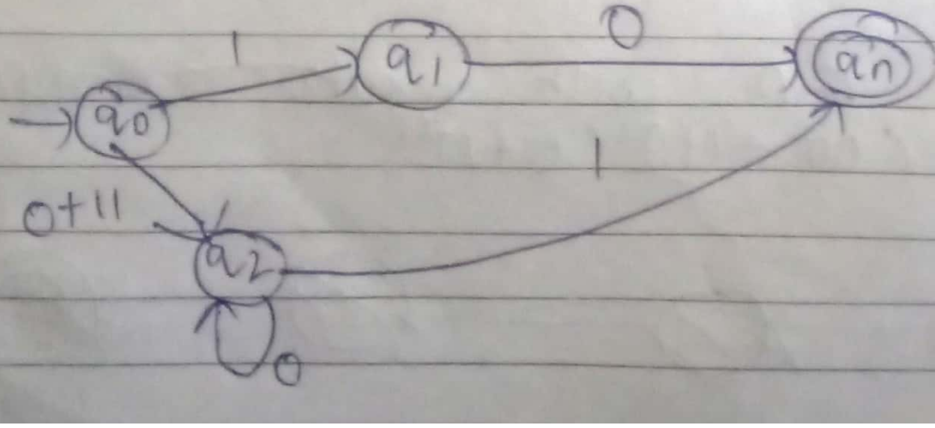
②

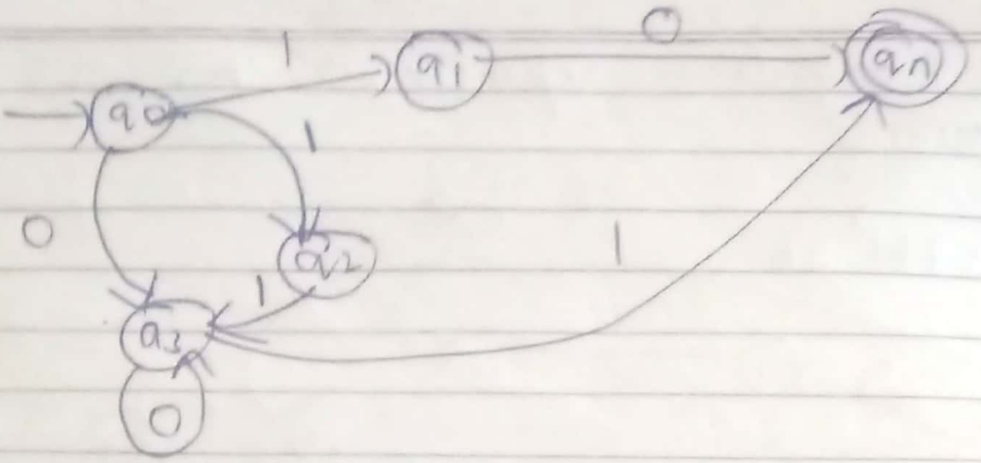


③



④

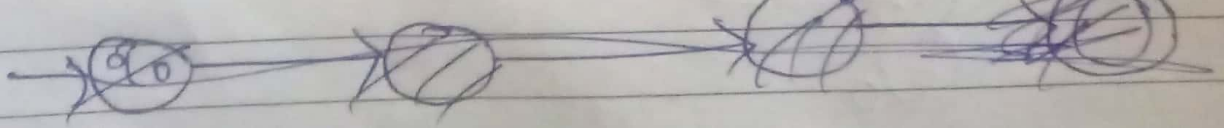


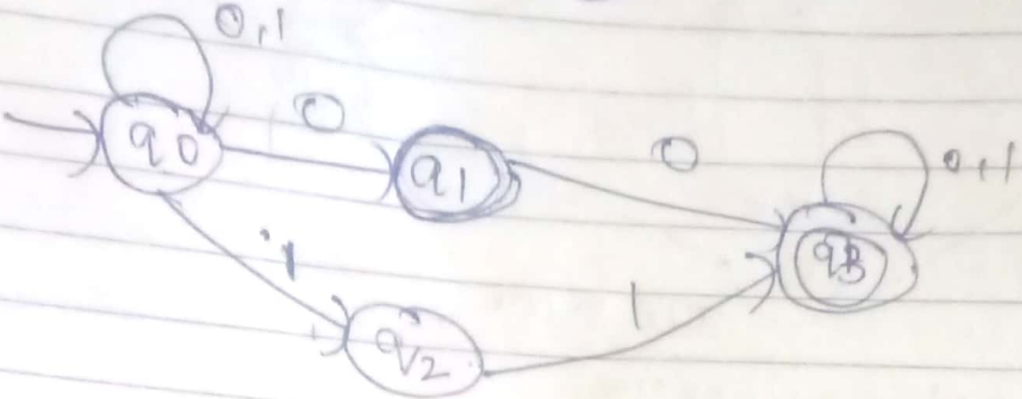


Transition Table

I/P state	0	1
q0	q3	{q1 q2}
q1	q4	φ
q2	φ	q3
q3	q3	q4
q4	φ	φ
[q1 q2]	qf	q3
qf	φ	φ

Q Construct FA with regular expression $(0+1)^* \, (00+11) \, (0+1)^*$





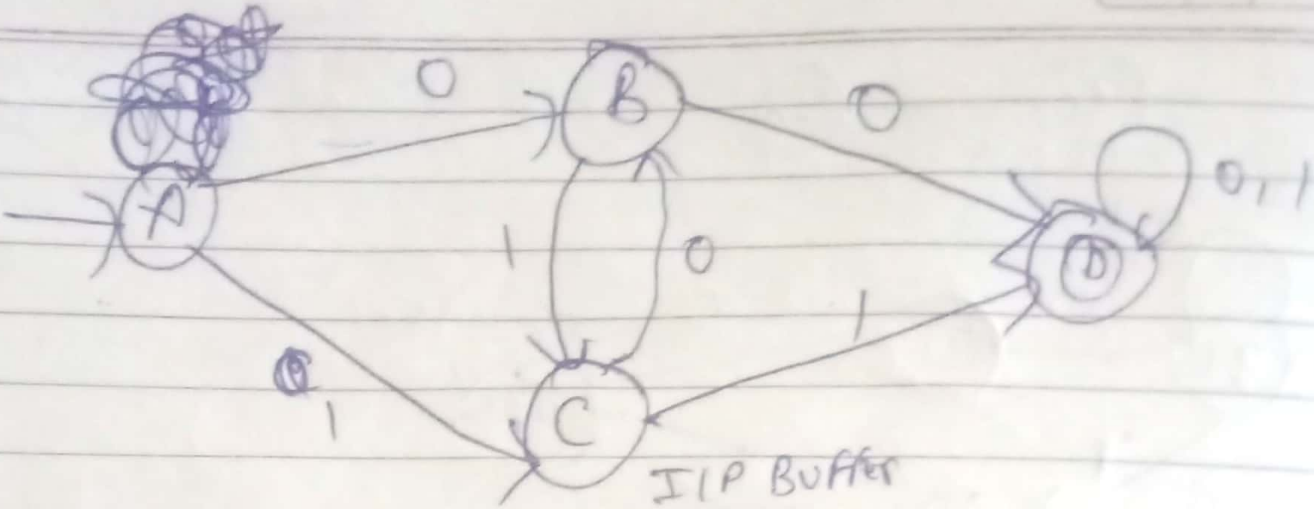
I/P state	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	q_3	–
q_2	–	q_3
q_3	q_3	q_3

	0	1
$[q_0]$	$[q_0, q_1]$	$[q_0\ q_2]$
$[q_0, q_1]$	$[q_0\ q_1\ q_3]$	$[q_0\ q_2]$
$[q_0\ q_2]$	$[q_0\ q_1]$	$[q_0\ q_2 q_3]$
$[q_0\ q_2 q_3]$	$[q_0\ q_1\ q_3]$	$[q_0 q_2\ q_3]$
$[q_0\ q_1 q_3]$	$[q_0\ q_1\ q_3]$	$[q_0 q_2 q_3]$

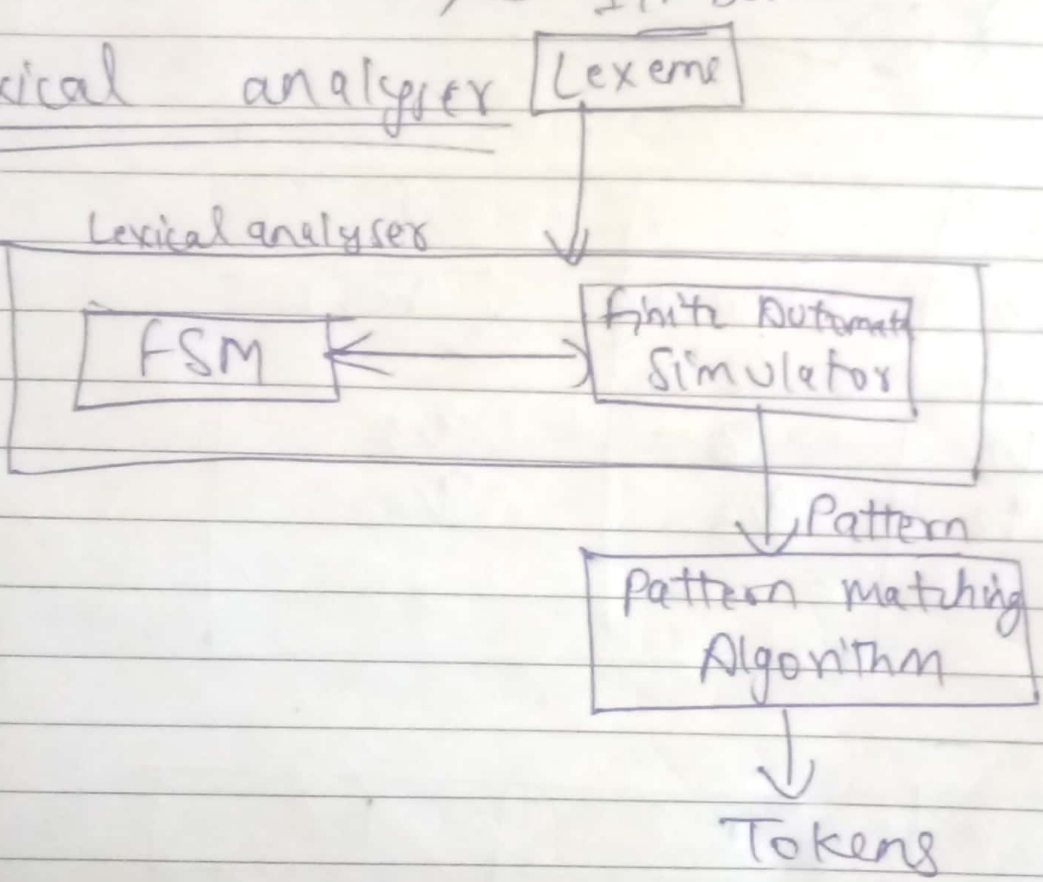
Equivalent

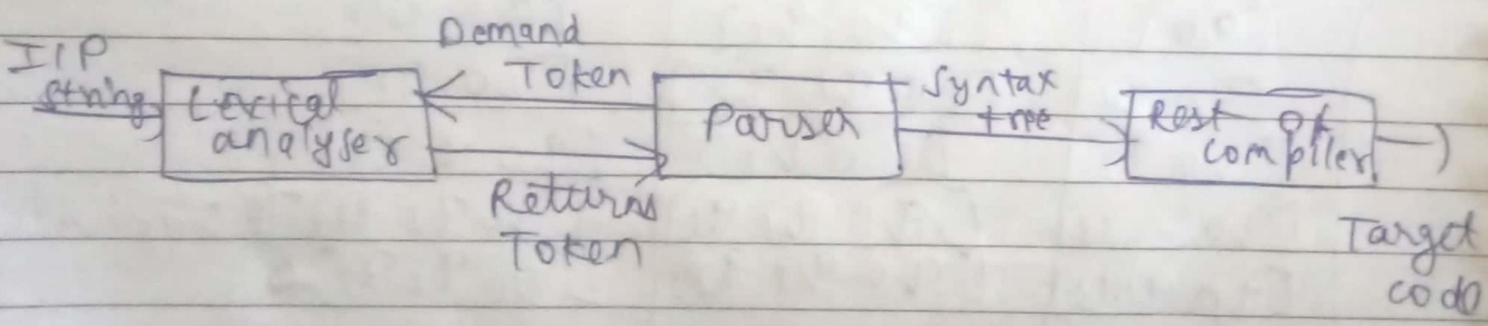
$A = [q_0]$ $B = [q_0\ q_1)$ $C = [q_0\ q_2]$

$D = [q_0\ q_1\ q_3]$



* Lexical analyser





Q Let G be a CFG Production rules are given

S → aB / bA

A → a / aS / bAA

B → b / bS / aBB

Derive the string. aaabbabbba using above grammer.

LMD

S → aB

S → aaBB (B → aBB)

S → aaaBBB (B → aBB)

S → aaabBB (B → b)

S → aaabbSB (B → bS)

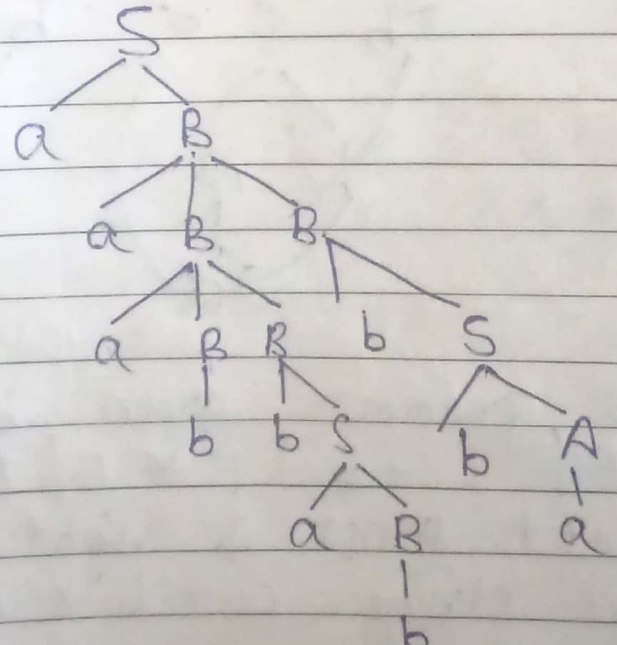
S → aaabbaBB (S → aB)

S → aaabbabB (B → b)

S → aaabbabbS (B → bS)

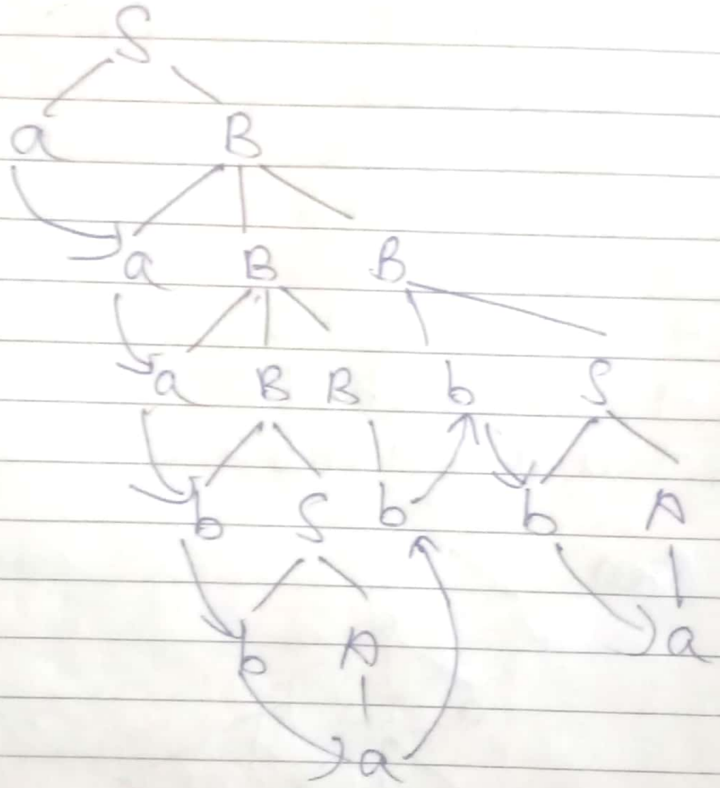
S → aaabbabbbA (S → bA)

S → aaabbabbba (A → a)



RMD

S → aB
→ aaBB
→ aaBbB
→ aaBbbA
→ aaBbba
→ aaaBBbba
→ aaaBbbba
→ aaabSbbba
→ aaabbA bbba
→ aaa bbabbba



Q Consider the grammer given below

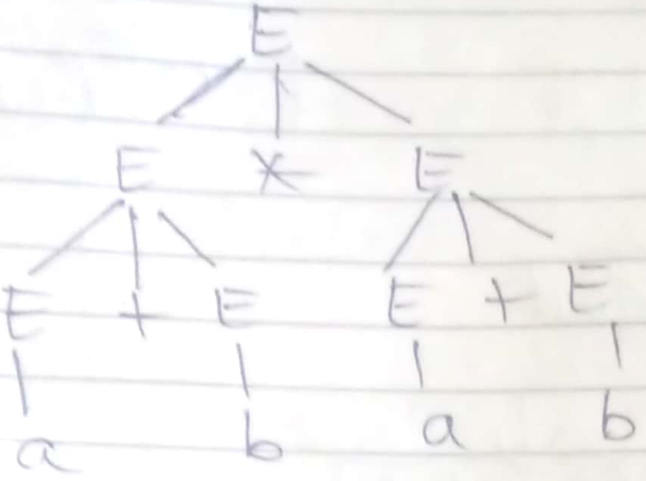
E → E + E | E - E | E * E | E/E | a|b

obtain the left most & right most derivation for the string a+b*a+b

E → E * E
E → E + E * E (E → E + E)
E → E + E * E + E (E → E + E)

E → E + E * E + E

E → a + b * a + b   (E → a , E → b, E → a , E → b)



RMD

Q Show that following grammer is amb-iguous.   abab

S → aSbS

S → bSaS

S → ∈

S → aSbS

S → abSaSbS   (S → bSaS)

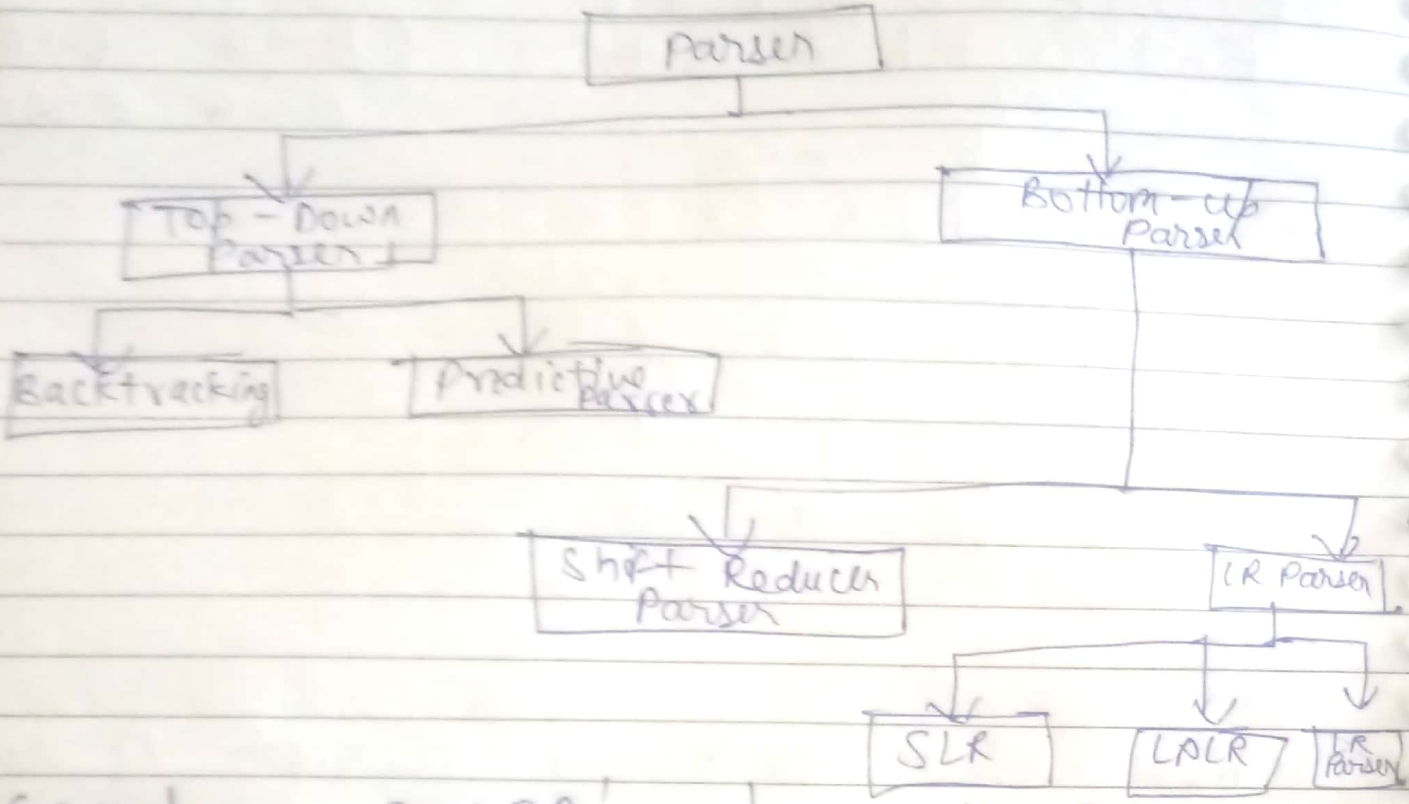
S → abaSbS   (S → ∈)

S → ababS   (S → ∈)

S → abab

Unit-2

Parsing



$A \rightarrow A\alpha$

$A \rightarrow \beta$

General

$A \rightarrow BA'$

$A' \rightarrow \alpha A'$

$A' \rightarrow \epsilon$

$T \rightarrow \overset{A}{T} \overset{\alpha}{* F} \mid \overset{B}{F}$

$T \rightarrow FT'$

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

Q Consider the following grammer

$A \rightarrow ABd / Aa \mid a$

$B \rightarrow Be \mid b$

Remove the left recursion.

Ans (i) $A \rightarrow ABd / a$

$A \rightarrow aA'$

$A' \rightarrow BdA'$

$A' \rightarrow \epsilon$

$A \rightarrow Aa \mid a$

$A \rightarrow aA'$

$A' \rightarrow aA' / \epsilon$

(2) $B \rightarrow Bc / b$

$B \rightarrow bB'$

$B' \rightarrow cB' / \epsilon$

* Left Factoring

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 / \beta_2$

$S \rightarrow iEtS / iEtSeS \mid a$

$S \rightarrow iEtSS' / a$

$S' \rightarrow eS / e$

$E \rightarrow b$

Q Do the left factoring in the following grammer.

$A \rightarrow aAB / aA / a$

$B \rightarrow bB / b$

Ans $A \rightarrow \alpha A'$

$B \rightarrow bB \mid b$

$\downarrow \quad \downarrow\downarrow \quad \downarrow$

$A \quad \alpha \; \beta_1 \; \alpha$

$B \rightarrow bB'$

$B' \rightarrow B / \epsilon$

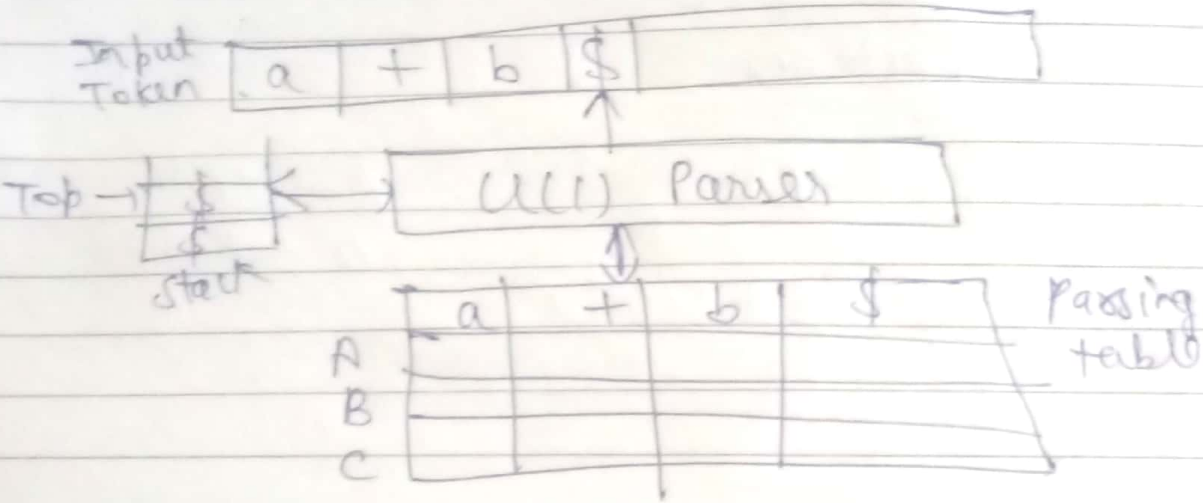
$A \rightarrow aA'$

$A' \rightarrow \beta_1 \mid \beta_2 / \beta_3$ $\quad A' \rightarrow AB / A / \epsilon$

* Ambiguity

There should be no ambiguity in the grammer.

There are two types of Predictive Parser
1. Recursive Decent
2. LL(1) Parser



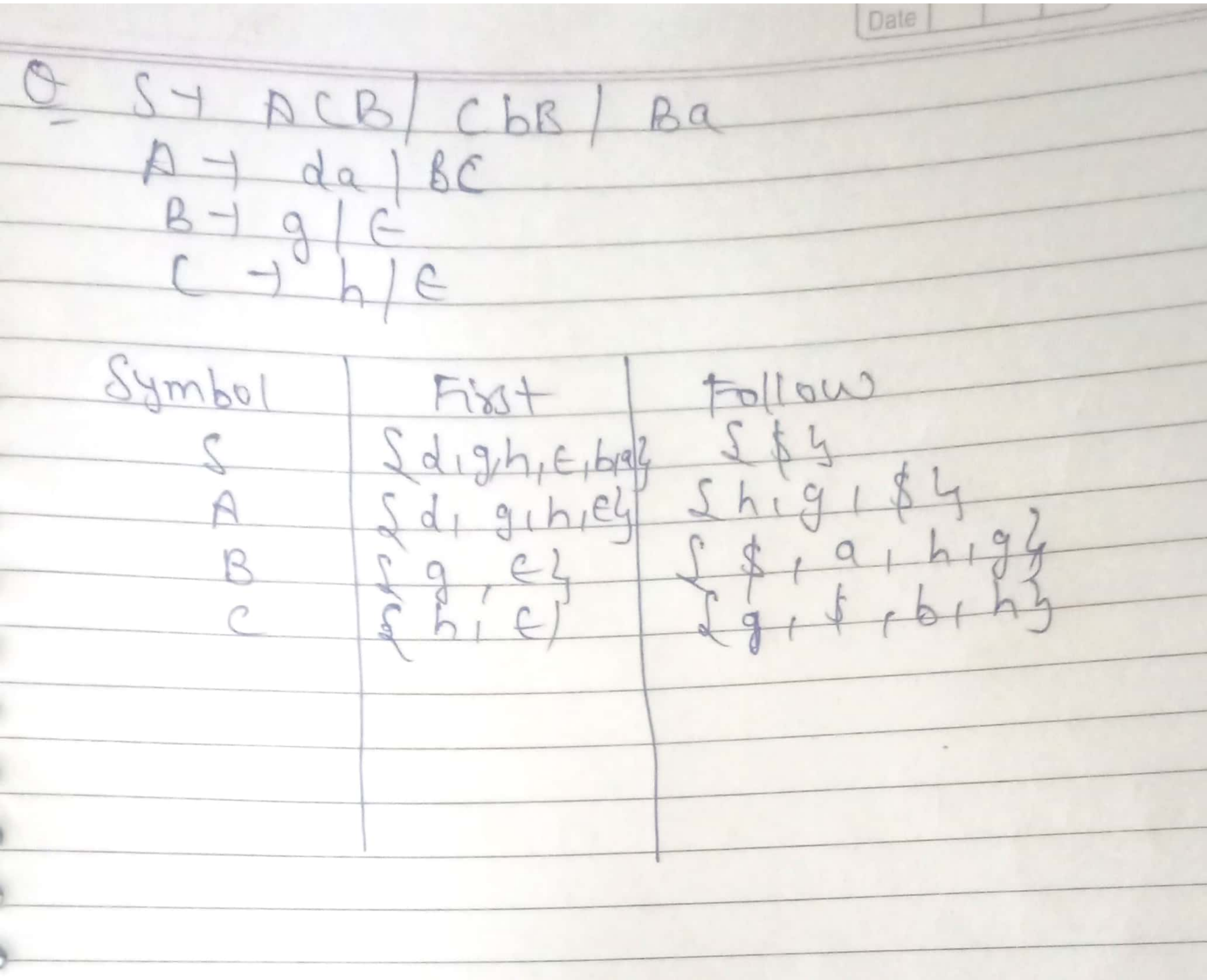
* First function

First(α)

Q S → ~~AB~~CDE
A → a | ϵ
B → b | ϵ
C → c
D → d | ϵ
E → e | ϵ

Non terminal

Symbol	First	Follow
S	{a, b, c}	{$}
A	{a, ϵ}	{b, c}
B	{b, ϵ}	{c}
C	{c}	{d, e, $}
D	{d, ϵ}	{e, $}
E	{e, ϵ}	{$}



Date

Q. S → ACB / CbB / Ba
A → da | BC
B → g / ε
C → h / ε

Symbol	First	Follow
S	{d, g, h, ε, b, a}	{$}
A	{d, g, h, ε}	{h, g, $}
B	{g, ε}	{$, a, h, g}
C	{h, ε)	{g, $, b, h}