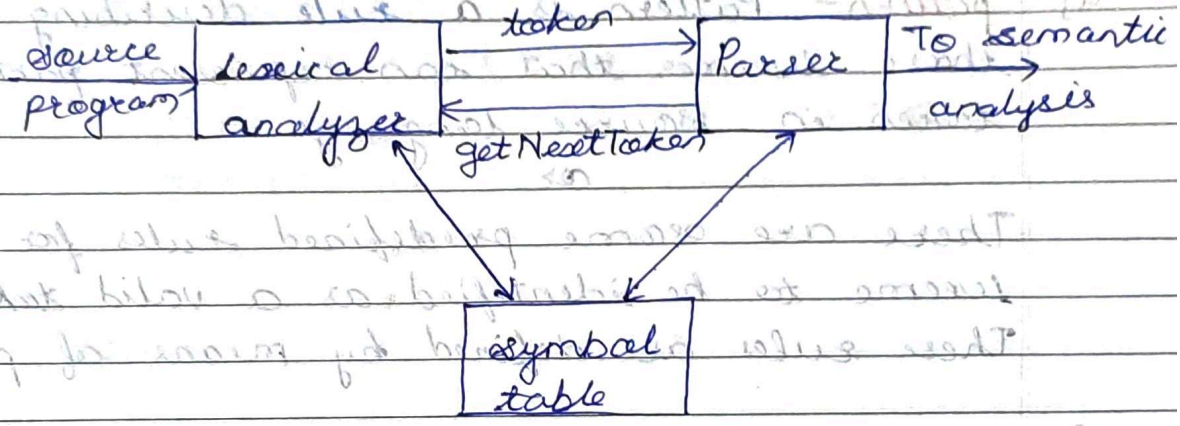
CC

Unit 2 - Compiler Course -

Lexical analysis

1] The role of lexical analyser



why to separate lexical analysis & parsing :-
① simplicity of design
② improving compiler efficiency
③ enhancing compiler portability

2] Tokens - A token is a sequence of characters that can be treated as a unit/single logical entity.

Ex - keywords (for, if, while)
identifiers (variable name, fun name)
operators (+, -, *, ++) etc.
separators (, ;) etc.

3] lexeme - It is a sequence of characters in source program that is matched by pattern for a token.

or

A sequence of input characters that comprises a single token is called lexeme.
Ex - "float", "-", "723", ";".

4] pattern - Pattern is a rule describing all those lexemes that can represent particular token in source language.

or

There are some predefined rules for every lexeme to be identified as a valid token. These rules are defined by means of pattern.

5] count number of tokens.

① int max(int i); = 7

② int main()
{
// 2 Variables declared
int a, b;
a = 10;
return 0;
}

= 18 comments not counted as tokens.

③ printf("Never give up");

= 5 string is considered as one token

Compiler

D D M M Y Y Y Y

④ printf("%d Hello", &x);

= 8

⑤ int main()
{ int a=10, b=20;
printf("sum is = %d", a+b);
return 0;
}

= 27

6] Attributes for tokens.

E = M * C * * 2

<id, pointer to symbol table entry for E>
<assign-op>
<id, pointer to symbol table entry for M>
<mult-op>
<id, pointer to symbol table entry for C>
<exp-op>
<number, int value 2>

7] Lexical errors.

Some errors are out of power of lexical analyzer to recognize:

fi (a==f(2)) ...

However it may be able to recognize errors like: d=2r.

Such errors are recognized when no pattern

for tokens matches a character sequence.

Types of errors detected by lexical analyzer

① long identifiers
Ex - int sum_of_two_numbers_entered_by_user;

② too long numerical literals
Ex - int x = 987654321;

③ badly formed numerical literals
Ex - char x = Vanita@123

④ input characters that are not present in source language.

⑤ spelling mistakes
Ex - printf

Error recovery

① Panic mode: successive characters are ignored until we reach to a well defined token.

② delete one character from the remaining input

③ insert a missing character into remaining input

④ replace a character by another character

⑤ transpose two adjacent characters.

8] Input buffering

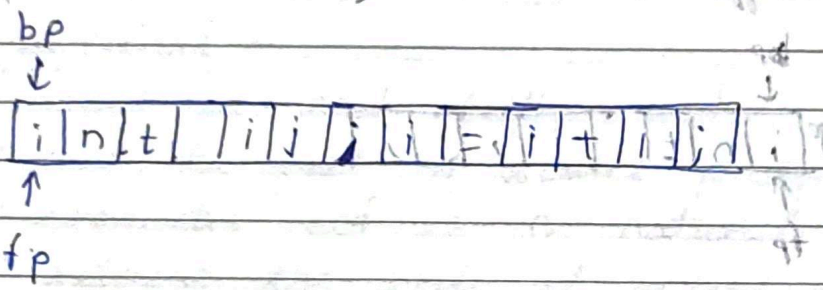
The lexical analyzer scans the input string from left to right one character at a time.

It uses two pointers -

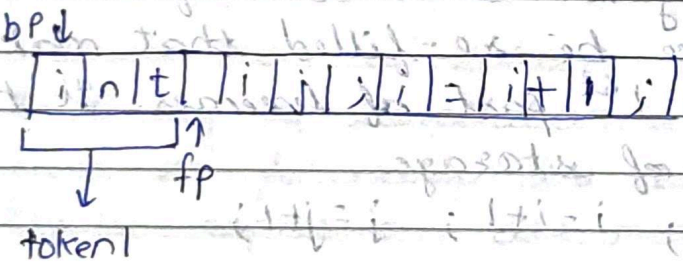
① begin pointer (bp) - from where read/write head starts scanning or to read tokens

② forward pointer (fp) - which moves a step ahead.

Ex - int i, j;
i = i+1 ;



Initially both the pointer points the first character of input string.



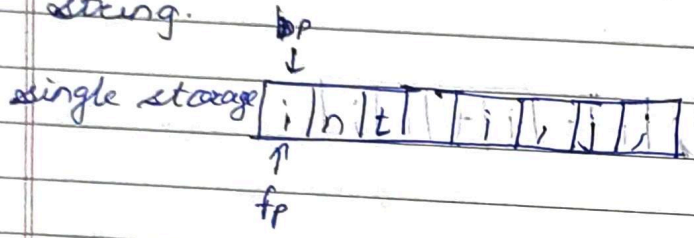
A forward pointer moves ahead to search the end of lexeme as soon as the blank space is encountered. It indicates the end of lexeme

The input character thus read from secondary storage but reading in this way from secondary

storage is costly, hence buffering scheme is used.
A block of data is first read into a buffer & then scanned by lexical analyzer.
Buffer means secondary storage.

Two methods are used:-
① one buffer scheme
② two buffer scheme.

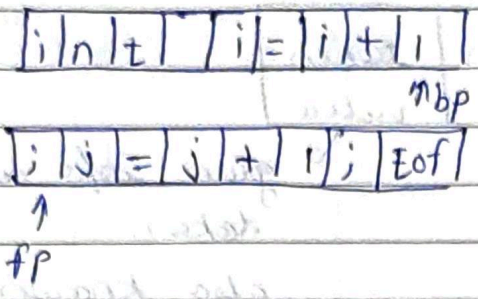
one buffer scheme :- In one buffer scheme, only one buffer is used to store the input string.



But the problem with this scheme is that if lexemes are very long then it process the buffer boundary to scan rest of lexemes. The buffer has to be re-filled that makes overwriting the 1st part of lexemes. It has limited amt of storage.
Ex - int i, j ; i = i+1 ; j = j+1 ;

Two buffer scheme :- 2 buffers are used to store the input string. 1st & 2nd buffer are scanned alternatively. When the end of 1st buffer is used, 2nd buffer is filled.

Ex - int i, j; i=i+1; j=j+1;



2nd buffer is used to improve execution speed of buffer.

Sentinel -

- added at each buffer end
- can't be part of source program
- character eof is a natural choice.
- retains the role of entire input end.
- when appears other than at end of buffer it means that input is at an end.

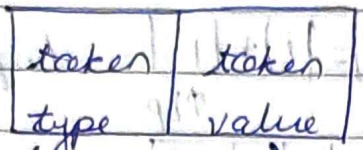
9] specification of tokens.

In theory of compilation regular expressions are used to formalize the specification of tokens. Regular expressions are means for specifying regular lang.

Ex - letter_(letter|digit)*

Each reg exp is a pattern specifying the form of strings.

10] Recognition of token.



tells the category of token - Ex- identifier, constant, etc

gives info about token. also known as token value.

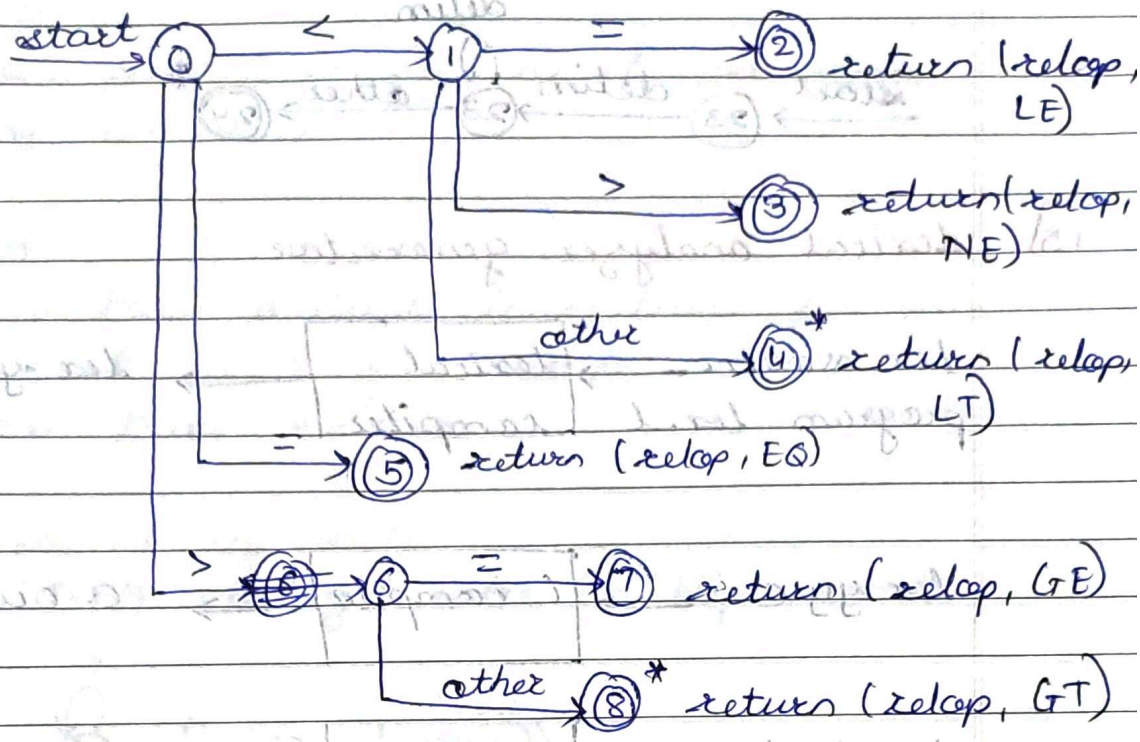
stmt → if expr then stmt
| if expr then stmt else stmt
| ε

expr → term relop term
| term

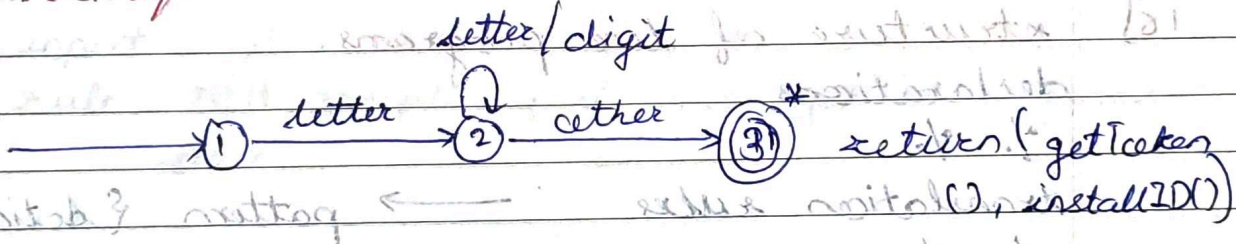
term → id
| number.

digit → [0-9]
digits → digit+
number → digit(.digits)? (E[+-]? digit)?
letter → [A-Z a-z]
id → letter (letter | digit)*
if → if
then → then
else → else
relop → < | > | <= | >= | = | <>
ws → (blank | tab | newline)+.

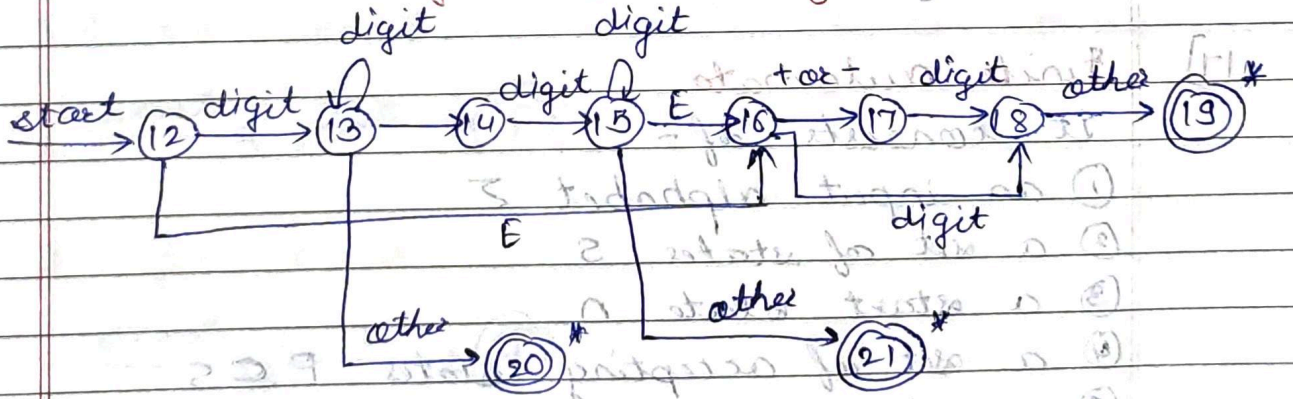
1] Transition diagram for relop



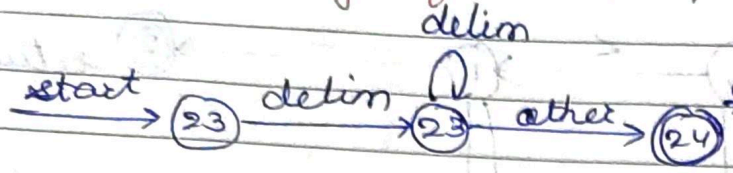
2] Transition diagram for reserved words & identifiers.



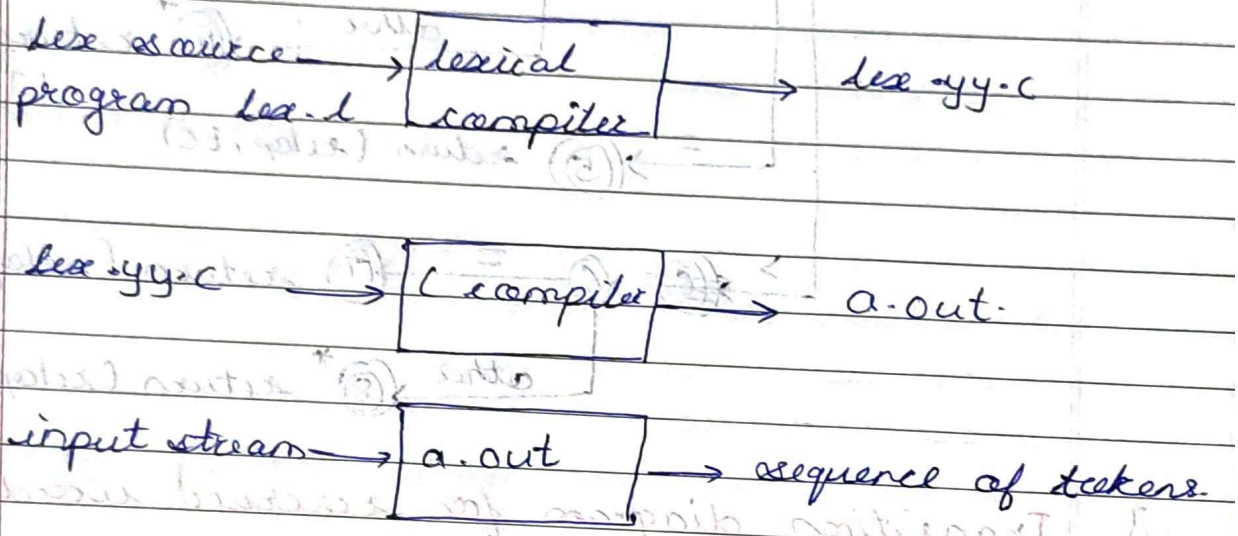
3] Transition diagram for unsigned numbers



14] Transition diag for whitespace



15] lexical analyser generator.



16] structure of lex programs.

declarations
%. %.
translation rules → pattern {Action}.
%. %.
auxiliary functions

17] Finite automata

It consists of -

① an input alphabet Σ
② a set of states S
③ a start state n
④ a set of accepting states $F \subseteq S$
⑤ a set of transition state → input state

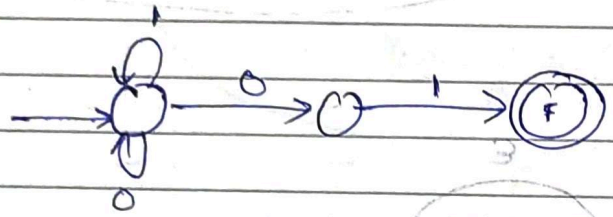
18] DFA & NFA.

- DFA.
  - one transition per input per state
  - no ε-moves.

- NFA
  - can have multiple transitions for one input in a given state
  - can have ε-moves.

19] acceptance of NFA.

An NFA can get into multiple states.

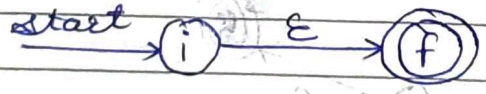


input : 101

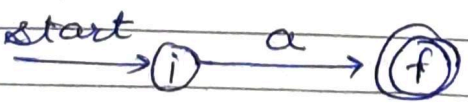
rule : NFA accepts if it can get in a final state.

20] Regular exp to NFA (Thomson's construction)

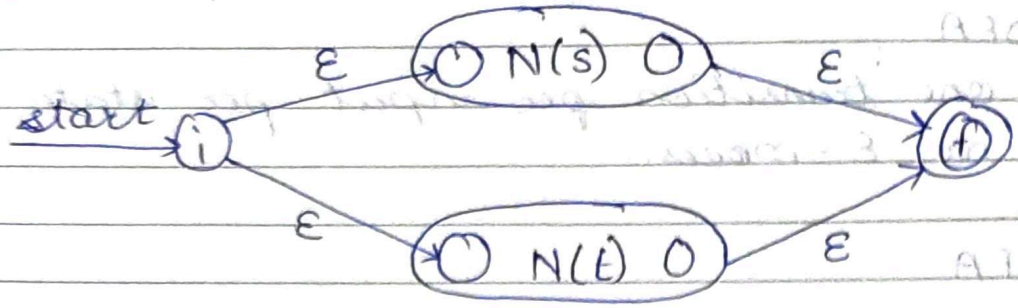
① for $r = \varepsilon$



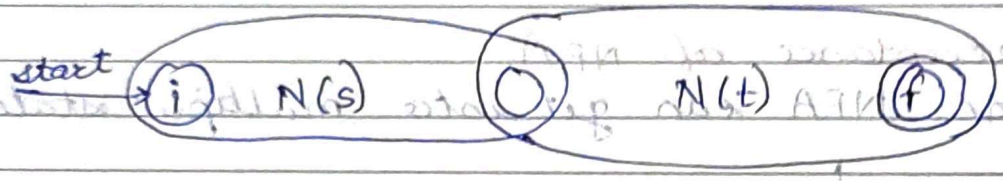
② for $r = a$



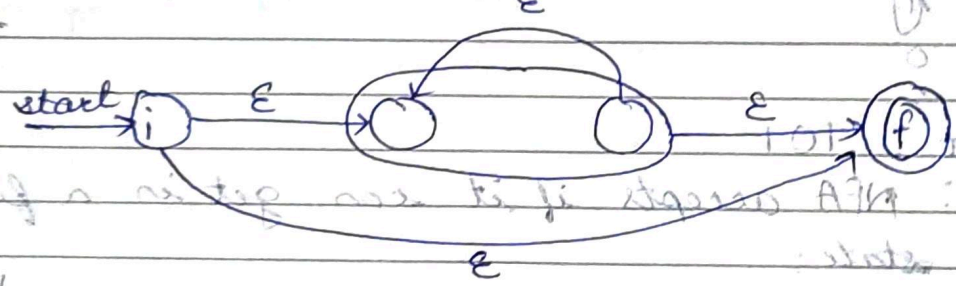
③ s|t , N(s|t).



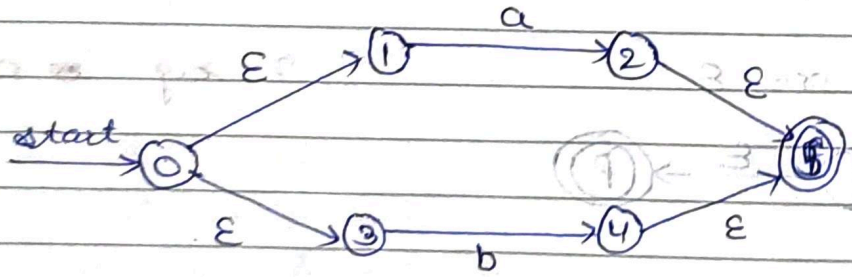
④ N(st)



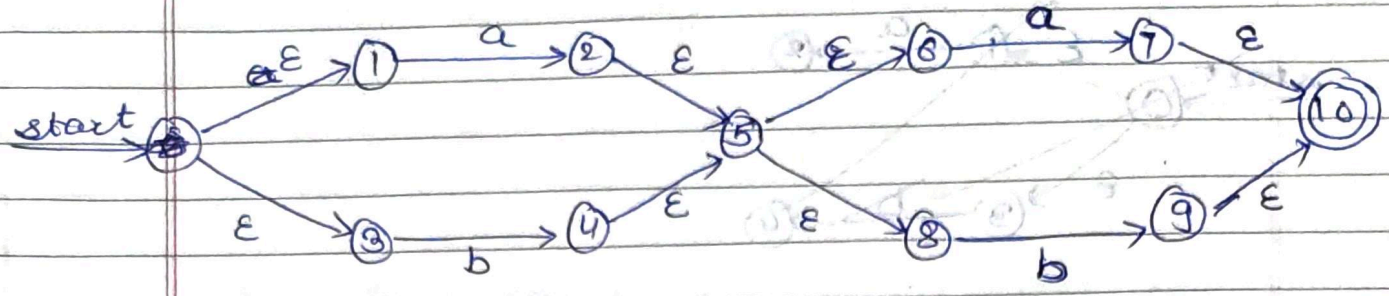
⑤ s^* , N(s^*)



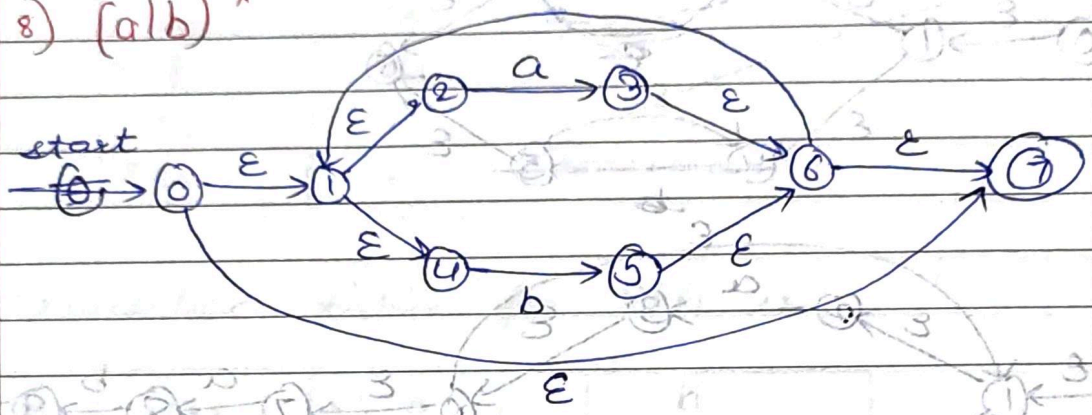
⑥ (a|b)



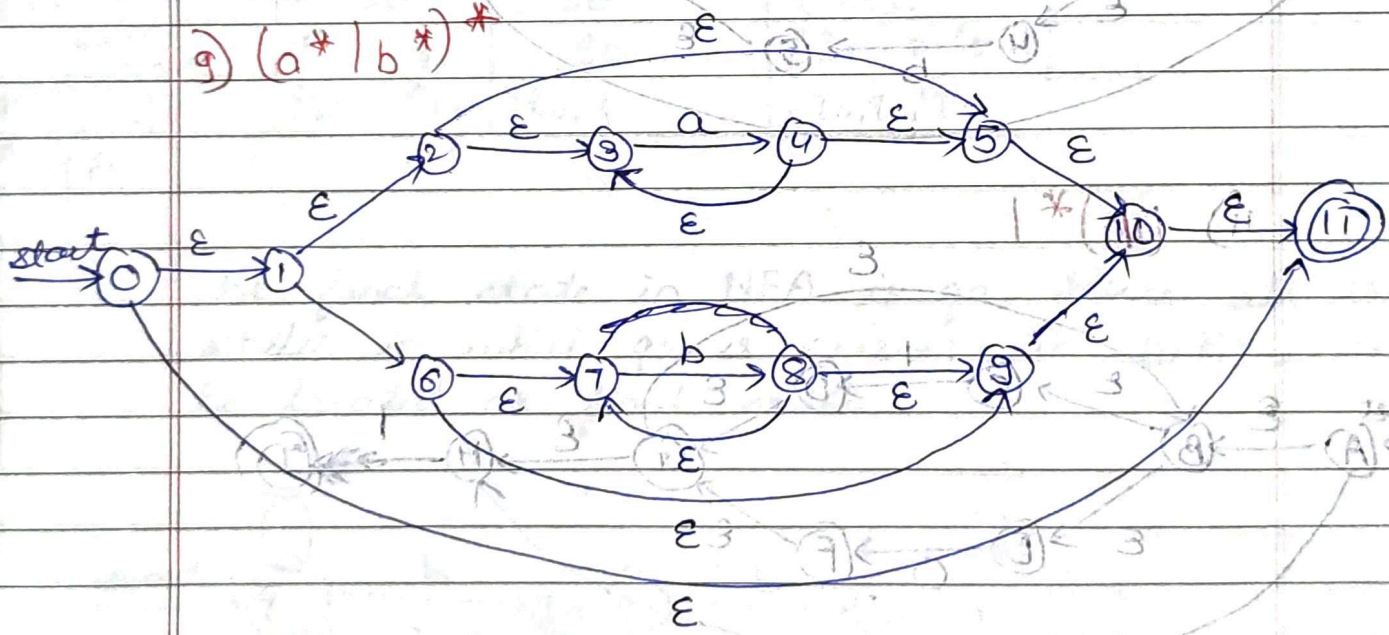
7) (a|b) (a|b).



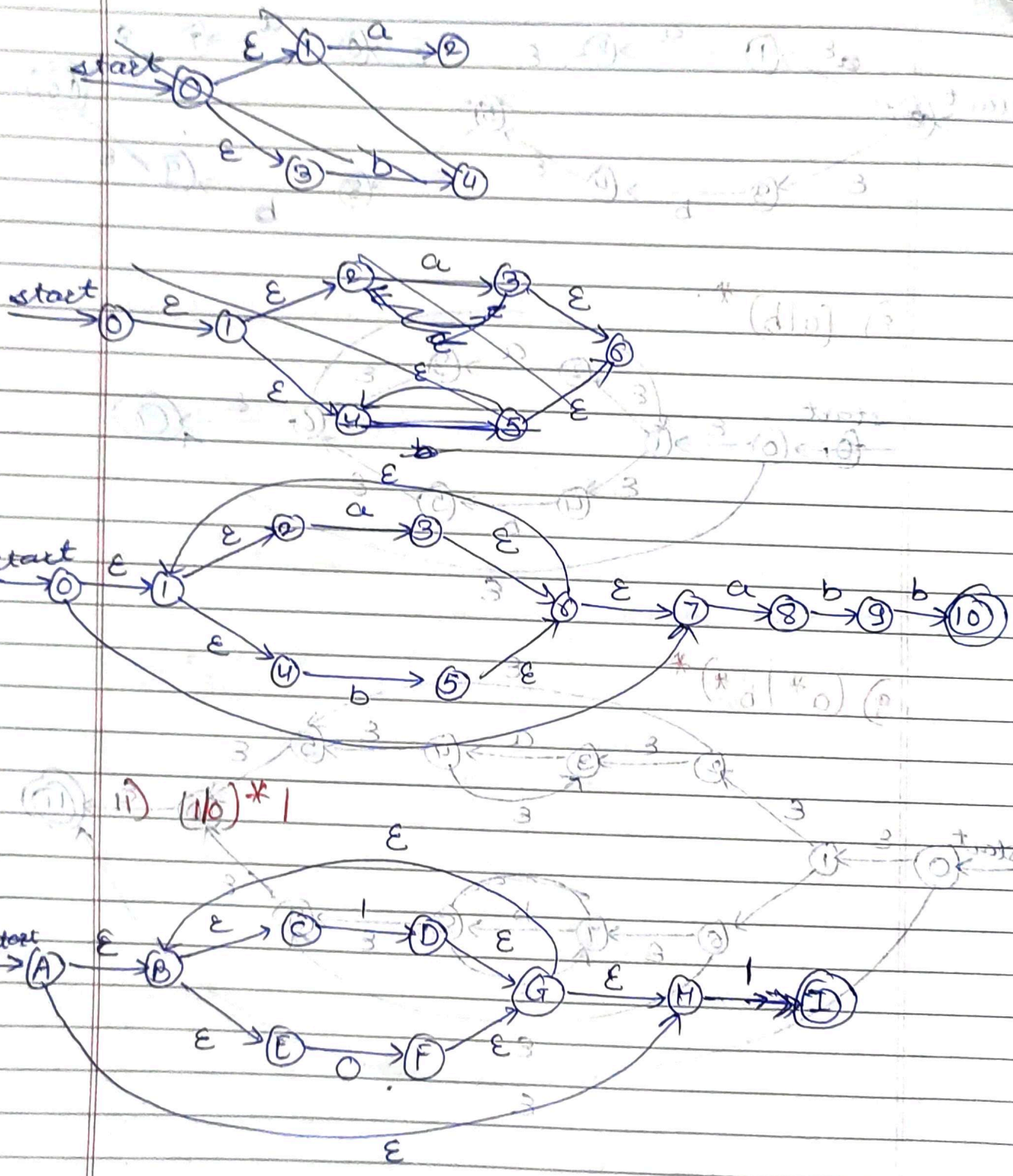
8) (a|b)*.



9) (a*|b*)*



10) $(a|b)^* \cdot abb$



11) $(1|0)^* 1$

Max number of states in DFA = 2

Q. Explain CC tools with example.

→ There are many types of software tools that have been developed to create one or more phases of compiler. These tools are called compiler construction tools. These tools use specialized language for defining & executing components & can use algorithms that are completely refined.

Following are example of CC tools-

① scanner generators –
This tool takes regular exp as input & create lexical analysis. The fundamental lexical analysis is produced by finite automata.
Ex - LEX

② Parser generators -
This software produces syntax analysis which takes input in the form of syntax of programming lang. depending on context free grammar.

③ Syntax-directed translation engines -
These software tools offer an intermediate code by using parse tree. It has goal of associating one or more translations with each node of parse tree.

(4) automatic code generators -
Takes intermediate code & converts them into machine language.

Q. What is role of lexical analysis?

→ - The lexical analysis is first phase of compiler where a lexical analyzer operate as an interface between source code & rest of phases of compiler.
- It reads the input characters of the source program, groups them into lexemes and produces sequence of tokens for each lexeme.
- The tokens are sent to parser for syntax analysis.
- The lexical analyzer also interacts with the symbol table while passing tokens to parser.

PAGE NO.:
DATE: / /

Q. Write note on attributes of token.

→
- A token is pair of a token name & an optional token value.
- A pattern is a description of the form that lexemes of a token may take.
- A lexeme is a sequence of characters in the source program that matches the pattern for a token.
- When more than one lexeme matches a pattern, the lexical analyzer must provide additional information about the particular lexeme that matched to the subsequent phases of the compiler.
- The lexical analyzer collects information about tokens into their associated attributes.
- Example - The program statement

  E = M * C ** 2

  yields the following token - attribute pairs :-

① <id, pointer to symbol table entry for E>
② <assign - op>
③ <id, pointer to symbol table entry for M>
④ <mult - op>
⑤ <id, pointer to symbol table entry for C>
⑥ <exp - op>
⑦ <number, integer value 2>.