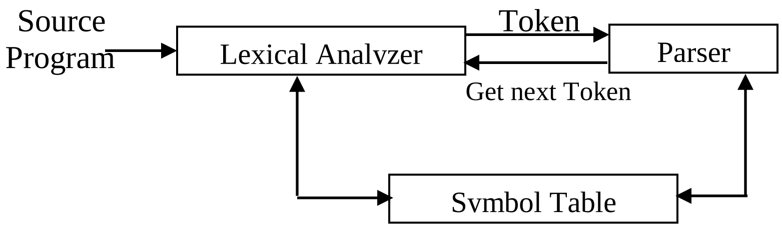
Role of Lexical Analyzer: -

The main task is to read the input characters and produce as output a sequence of tokens that the parser uses for syntax analysis.



After receiving a “get next token” command from the parser, the lexical analyzer reads input characters until it can identify a next token.

Token:-

Token is a sequence of characters that can be treated as a single logical entity. Typical tokens are,

(a) Identifiers
(b) Keywords
(c) Operators
(d) Special symbols
(e) Constants

Pattern:-

A set of strings in the input for which the same token is produced as output, this set of strings is called pattern.

Lexeme:-

A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.

Finite Automata

Definition: -

A recognizer for a language is a program that takes as input a string x and answers “yes” if x is a sentence of the language and “no” otherwise.

A better way to covert a regular expression to a recognizer is to construct a generalized transition diagram from the expression. This diagram is called finite automation.

A finite automation can be,

1. Deterministic finite automata
2. Non-Deterministic finite automata

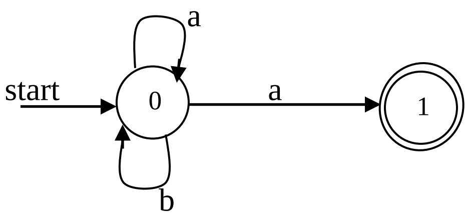
1. **Non – deterministic Finite Automata**:- [**NFA**]

A NFA is a mathematical model that consists of,

1. a set of states S
2. a set of input symbol Σ
3. a transition function δ
4. a state S_0 that is distinguished as start state
5. a set of states F distinguished as accepting state. It is indicated by double circle.

Example:-

The transition graph for an NFA that recognizes the language (a/b)* a



The transition table is,

State	Input Symbol	
	a	b
0	0,1	0
1	-	-

2. **Deterministic Finite Automata**: - [**DFA**]

A DFA is a special case of non – deterministic finite automata in which,

1. No state has an ε – transition
2. For each state S and input symbol there is atmost one edge labeled a leaving S.

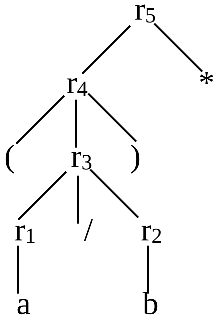
PROBLEM: -

1. Construct a non – deterministic finite automata for a regular expression (a/b)*

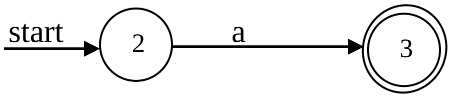
Solution;-

r = (a/b)*

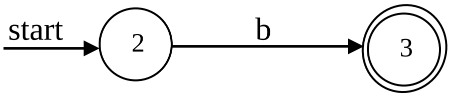
Decomposition of (a/b)* (parse tree)



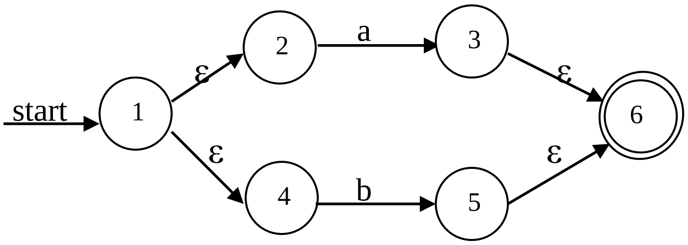
For r_1 construct NFA



For r_1 construct NFA

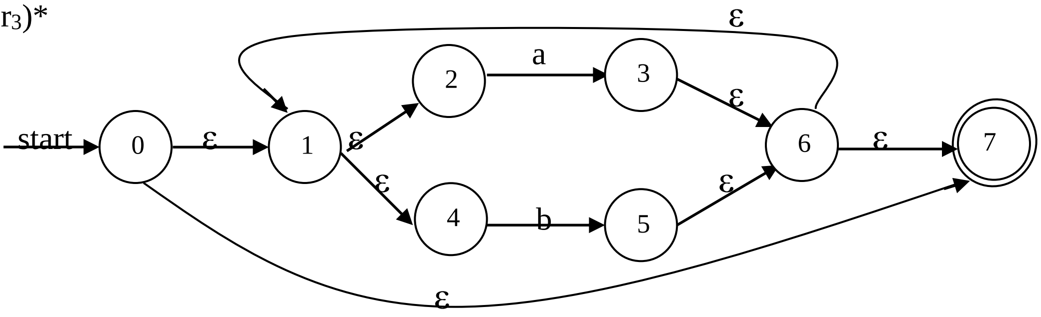


NFA for $r_3 = r_1/r_2$



NFA for r_4, that is (r_3) is the same as that for r_3.

NFA for $r_5 = (r_3)^*$

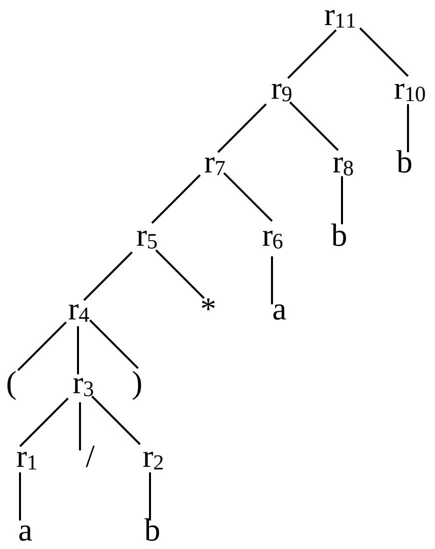


2. Construct a non – deterministic finite automata for a regular expression (a/b)*abb

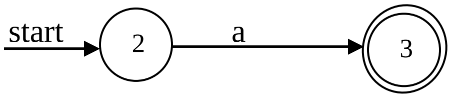
<u>Solution</u>;-

r = (a/b)*

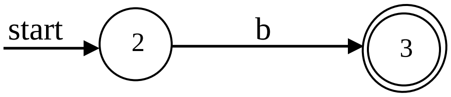
Decomposition of (a/b)* abb (parse tree)



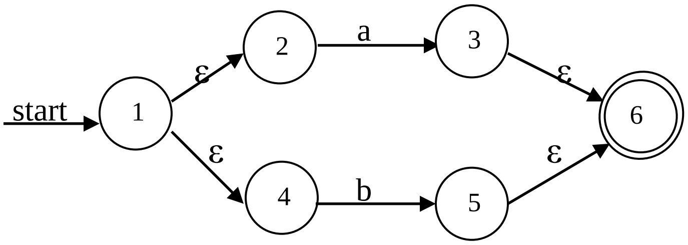
For r_1 construct NFA



For r_1 construct NFA

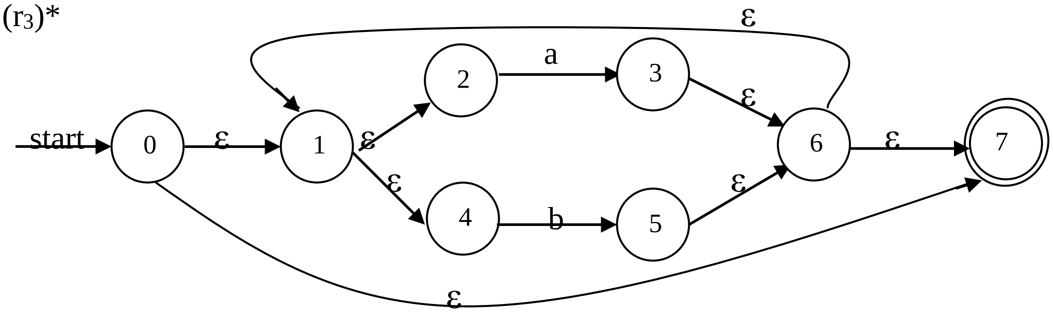


NFA for $r_3 = r_1/r_2$

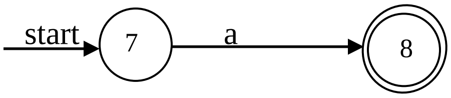


NFA for r_4, that is (r_3) is the same as that for r_3.

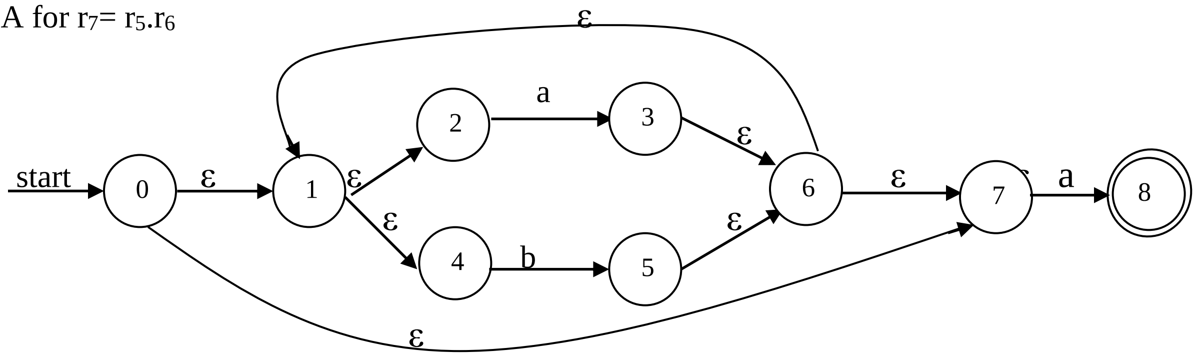
NFA for $r_5 = (r_3)^*$



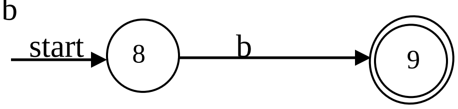
NFA for r_6=a



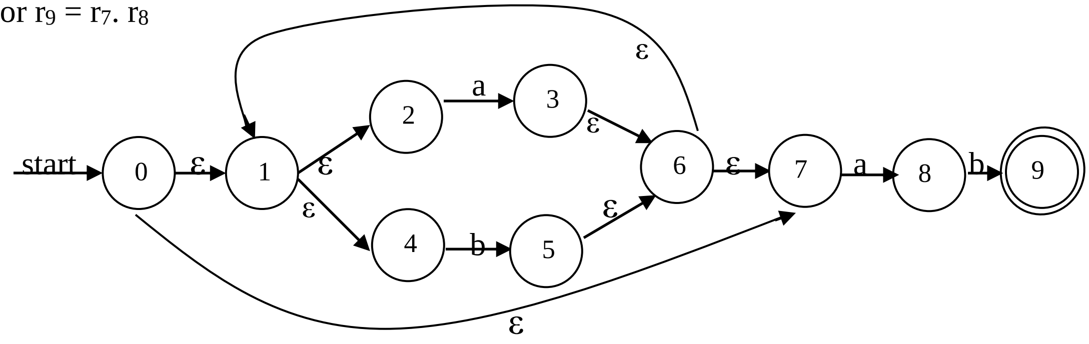
NFA for r_7= r_5.r_6



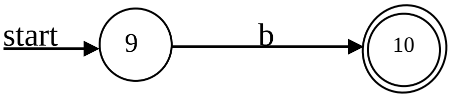
NFA for r_8 = b



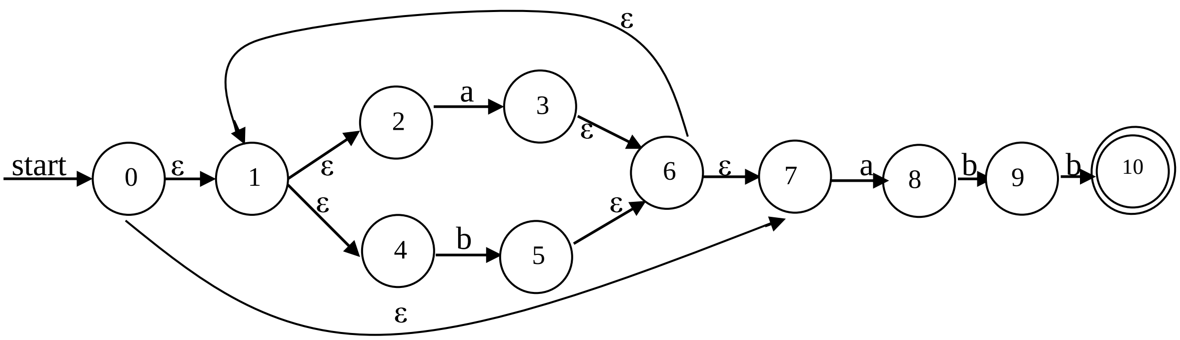
NFA for r_9 = r_7. r_8



NFA for r_{10} =b



NFA for r_{11} = r_9.r_{10} = (a/b)* abb



References

1. J. Tremblay, P.G. Sorenson,"The Theory and Practice of Compiler Writing “, McGRAW-HILL,1985.
2. W.M. Waite, L.R. Carter,"An Introduction to Compiler Construction",Harper Collins,New york,1993
3. A.W. Appel,"Modern Compiler Implementation in, CambridgeUniversity Press,1998
4. Internet Papers
5. Aho, R. Sethi, J.D. Ullman," Compilers- Principles, Techniques and Tools"Addison-Weseley, 2007