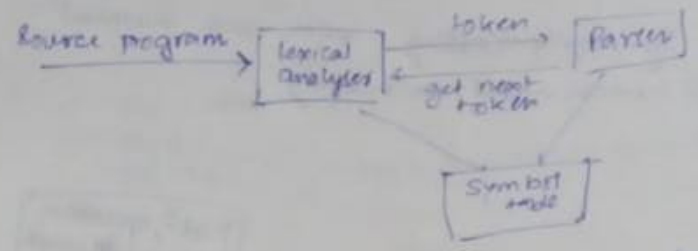
lexical analysis



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

Pattern: A set of strings each described by a rule is called a pattern associated with a token.

Eg:

token	lexeme (eg of token)	Pattern (rules)
id	D2, count	letter followed by a letter or digit

- Set of symbols : input-alphabet : Σ
  sequence of symbols : string
  language : set of strings.
  prefix : delete from trailing edge
  suffix : " " leading edge

  Substring → maintains order
  Subsequence → not bound to maintain order.

Regular expression ⇒ notation

a	{a}
b	{b}

operations on language

$L = \{a, b \ldots z, A, B, C \ldots Z\}$

$D = \{0, 1 \ldots 9\}$.

$L \cup D$ ← Union, concatenation, closure

Kleene's closure → $L^* = \bigcup_{i=0}^{\infty} L^i$

Positive " → $L^+ = \bigcup_{i=1}^{\infty} L^i$

Σ^+ Σ^*

$a^* = \{\epsilon, a, aa, aaa, \ldots\}$

$a^+ = \{a, aa, aaa, \ldots\}$

$L(L \cup D)^*$

$a(a|b) \rightarrow aa$
ab

$a(a|b)^* \rightarrow a$
aaa
abb

14/7/22

Informal analysis → communication b/w client developers.

Structured analysis →

Cardinality (entity): no of instances of an entity related to no of instances of another entity.

Ordinality: whether a relationship is mandatory or not.

Primary Attribute : identifies each entity uniquely.

Derived "

data specification ⇒ ERD

process " ⇒ DFD

control " ⇒ STD

Regular expression for id : $L(L|D)^*$

$L(r_1+r_2) = L(r_1) \cup L(r_2)$

$L(r_1 r_2) = L(r_1) L(r_2)$

$r|s$	$L(r) \cup L(s)$	
rs	$L(r) L(s)$	
r^*	$L(r)^*$	
reg exp.	reg lang.	
1. ϵ	$\{\epsilon\}$	
2. a	$\{a\}$	
3. $a|b$	$\{a, b\}$	
4. ab	$\{ab\}$	
5. a^*	$\{\epsilon, a, aa, aaa \ldots\}$	

① Set of all strings a's and b's of length 2.

② Set of all strings of 0 ~~or~~ or more b's

③ Set containing a string a and all string consisting of 0 or more a's followed by a b

④ Give the reg exp of ~~exp~~ exactly one a

$\Sigma = \{a, b\}$.

⑤ Give the reg exp for atleast 2 a.

① aa bb ab ba

→ $(a|b)(a|b)$

② b^*

③ $a|((a)^*b)$

④ $b^*(a)b^*$

⑤ ~~$(a)^*a$~~ $(a|b)^*a(a|b)^*a(a|b)^*$

Regular definition:

Qs.

① Define reg. exp. such that it should contain atleast one double letter.

② Write the reg. exp. over alphabet $\{0, 1\}$ for the set of string with even no. of 0's followed by odd no. of 1's for the language.

(3) Write the reg exp. for the language in which words ending with either aa or b

(4) Write the regular exp for all string with even no of 0's followed by odd no of 1's.

(4) Write regular exp. for the language that the set of all strings that begin/end with 00 or 11

(5) Write reg. exp for the set of all strings in which both the no of a's & b's are even

13/7/22

If Σ is an alphabet of basic symbols, then a regular defn. is a sequence of defn of the form →

$$\left.\begin{array}{l} d_1 \rightarrow r_1 \\ d_2 \rightarrow r_2 \\ d_3 \rightarrow r_3 \\ \vdots \\ d_n \rightarrow r_n \end{array}\right\} \rightarrow \text{reg exp}^n$$

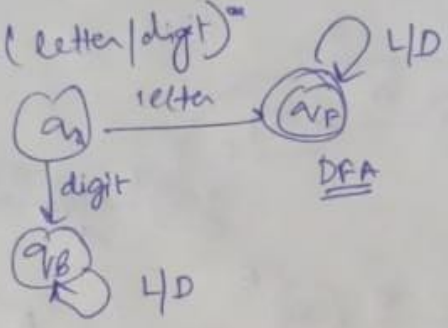
distinct name ←

letter = a|A| . . . |z|Z

digit = 0|1| . . . |9

id = letter(letter|digit)*

Trans. diag.



DFA

NFA: $Q \times \{\Sigma \cup \epsilon\} \rightarrow 2^Q$

DFA: $Q \times \Sigma \rightarrow Q$

(*) Conversion of a NFA to DFA:-

Subset construction:-

(i) ϵ closure (s)

(ii) ϵ closure (T)

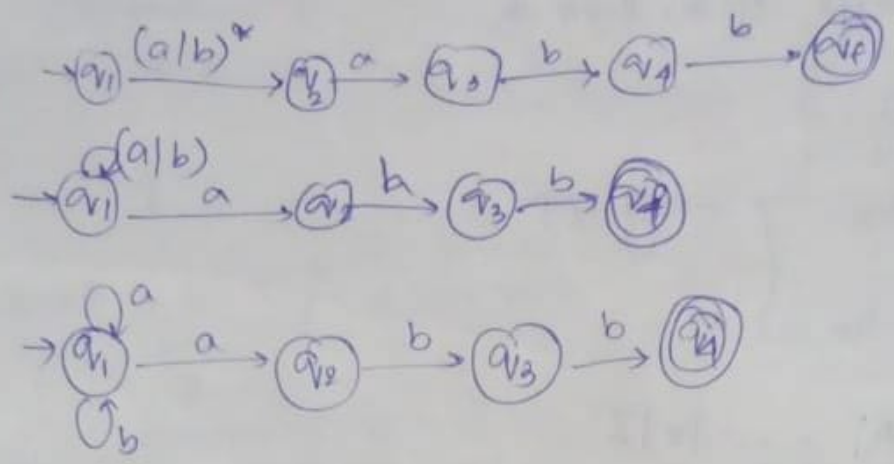
(iii) move (T, a)

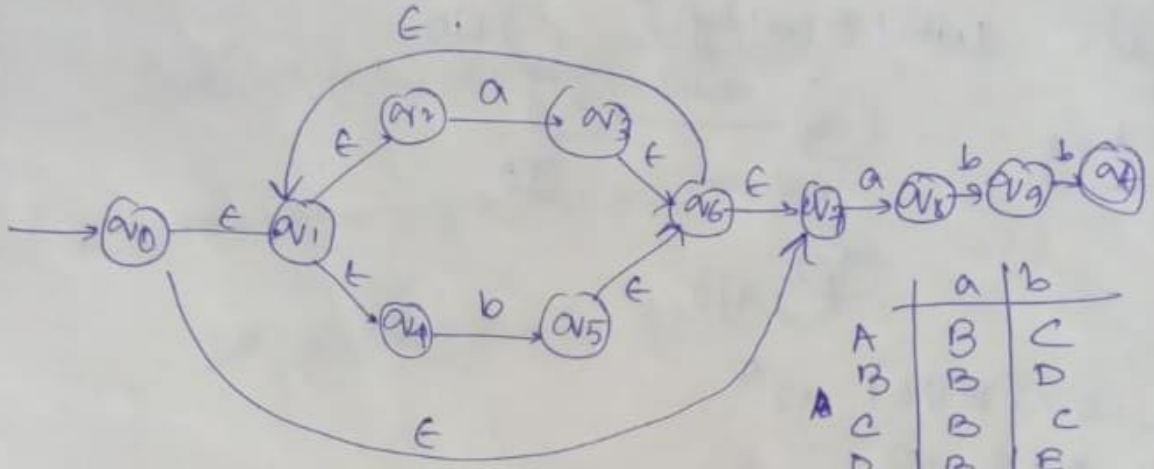
① set of NFA states reachable from NFA state s on ϵ transition alone.

② set of NFA states reachable from some NFA states s in set T on ϵ transition.

③ set of NFA states to which there is a transition on i/p symbol a from some states s in T

$(a|b)^* abb$





	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

$A = \epsilon\text{-closure}(\{v_0\}) = \{v_0, v_1, v_2, v_4, v_7\}$

$B = \epsilon\text{-closure}(\{v_3, v_8\}) = \{v_1, v_2, v_3, v_4, v_6, v_7, v_8\}$

$C = \epsilon\text{-closure}(\{v_5\}) = \{v_1, v_2, v_4, v_5, v_6, v_7\}$

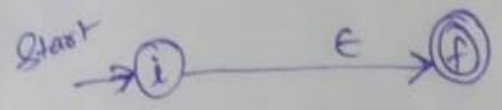
$D = \epsilon\text{-closure}(\{v_5, v_9\}) = \{v_1, v_2, v_4, v_5, v_6, v_7, v_9\}$

$E = \epsilon\text{-closure}(\{v_5, v_{10}\}) = \{v_1, v_2, v_4, v_5, v_6, v_7, v_{10}\}$

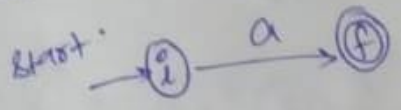
regular expression to NFA by Thompson construction

$(a|b)^*$

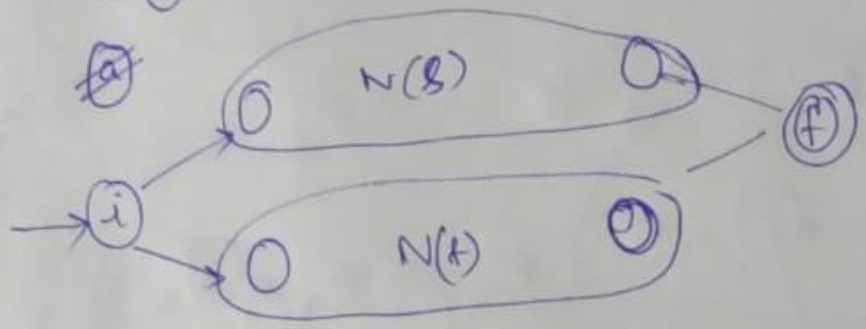
1. For expression ϵ

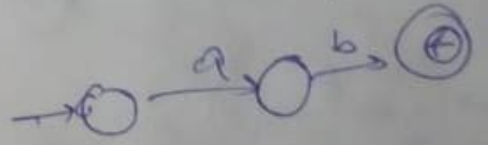


2. For expression a



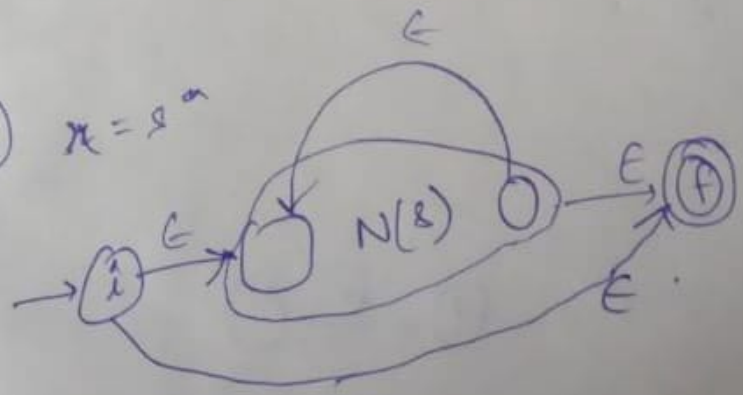
3 (a) $r = s|t$.



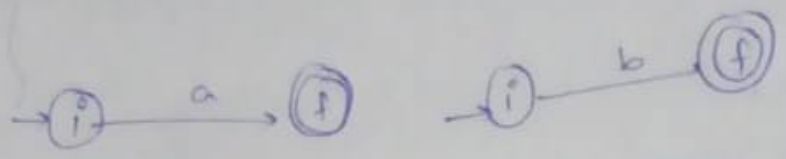


(b) $r = st$.

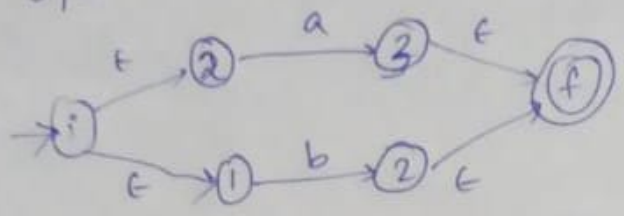
(c) $r = s^*$



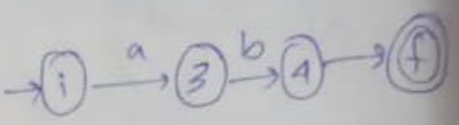
$(a|b)^* ab$



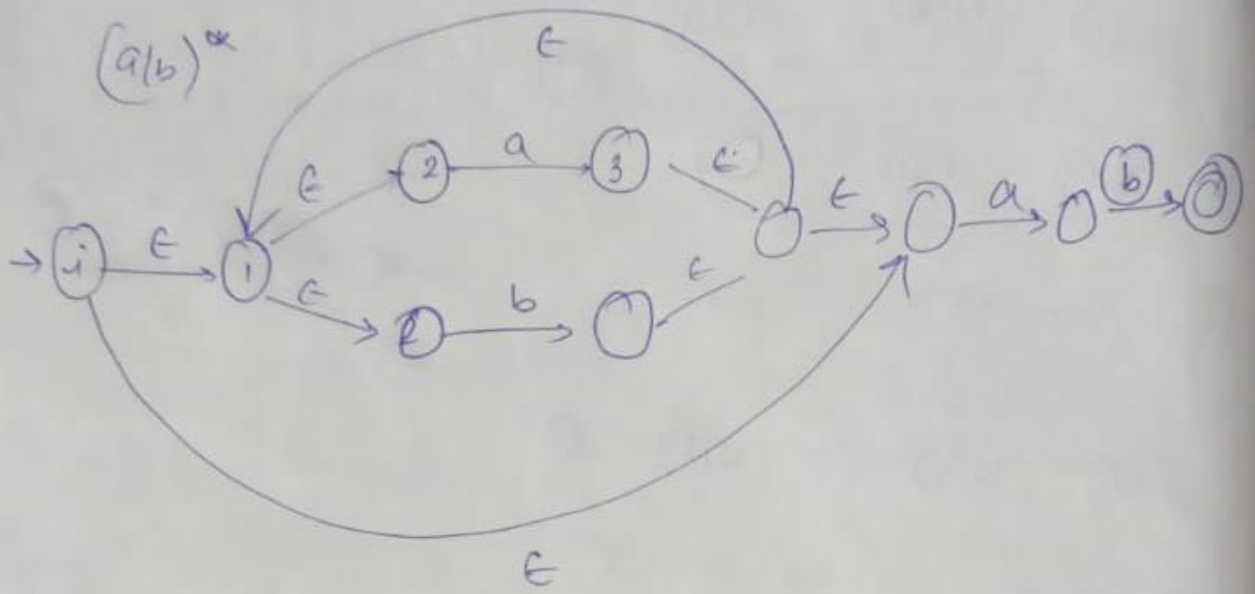
$a|b$



ab

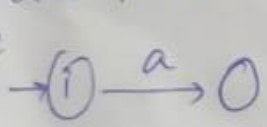


$(a|b)^*$

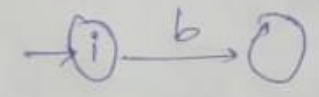


$aa^* | bb^*$

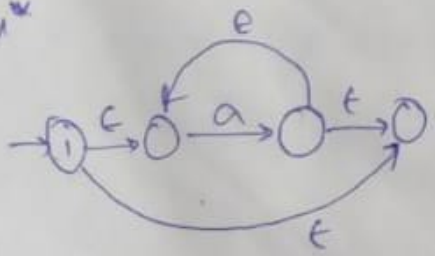
a



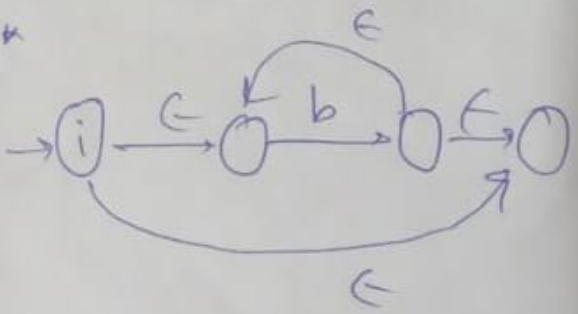
b



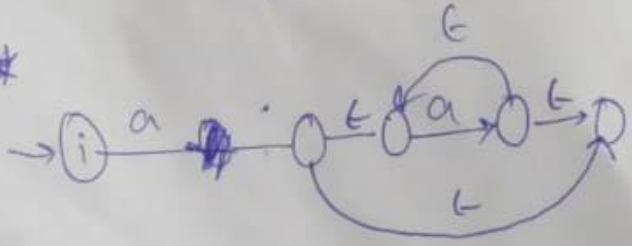
a^*



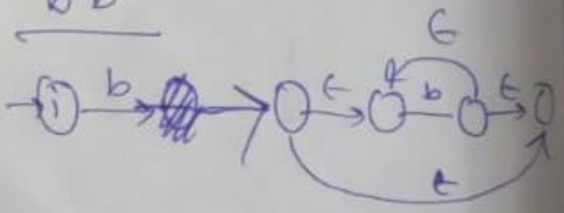
b^*



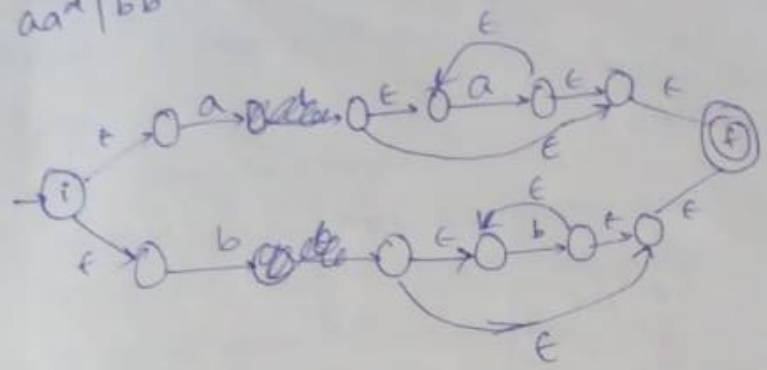
aa^*



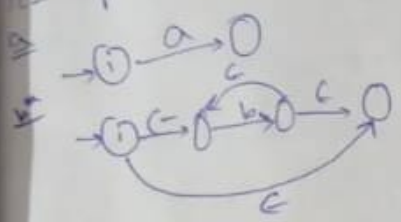
bb^*



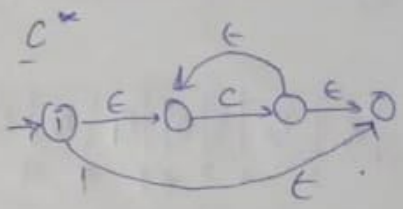
$aa^*|bb^*$



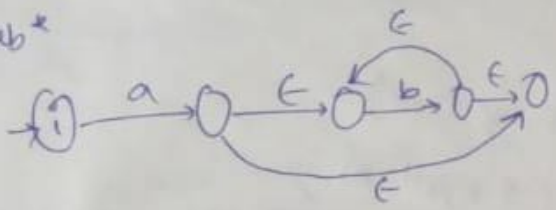
$a|ab^*|c^*$



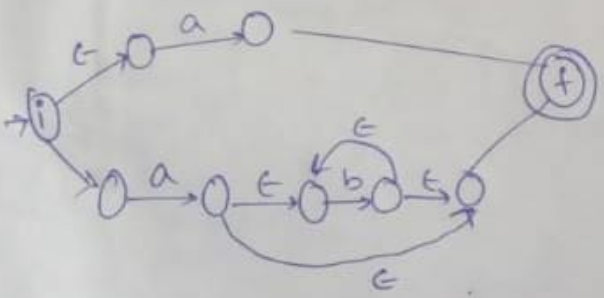
c^*

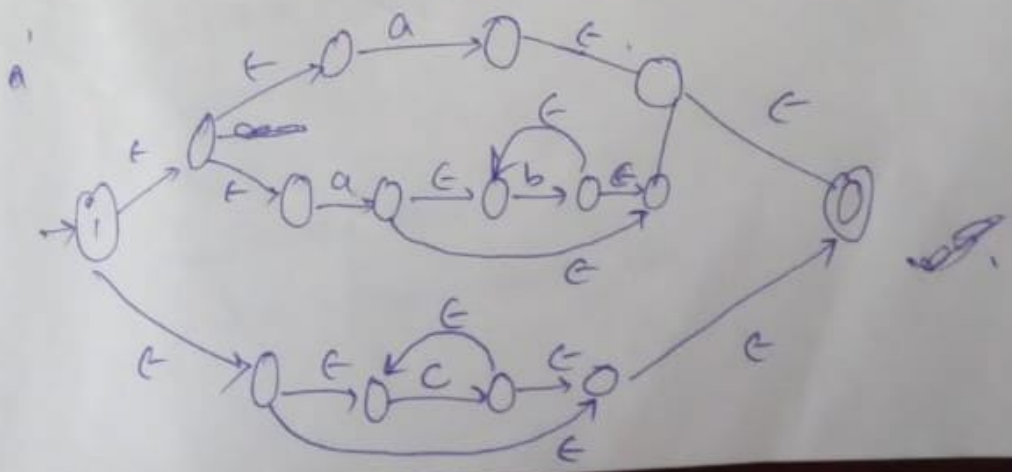


ab^*



$a|ab^*$



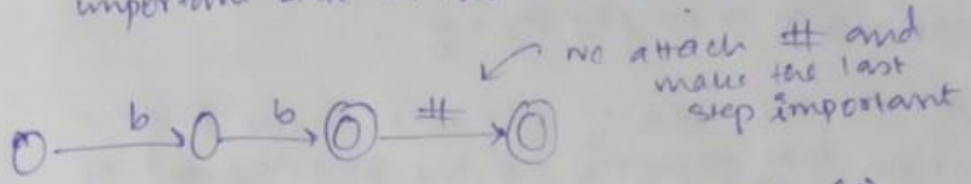


25/7/22

(1) Reg. exp to DFA.

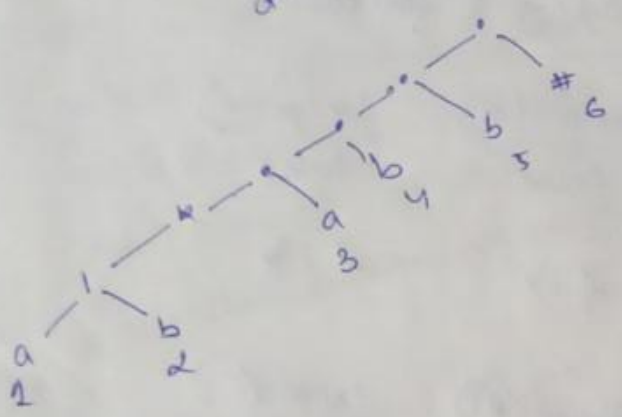
Step (i) we construct augmented regular exp. $(r)\#$

important state → no ε transition.

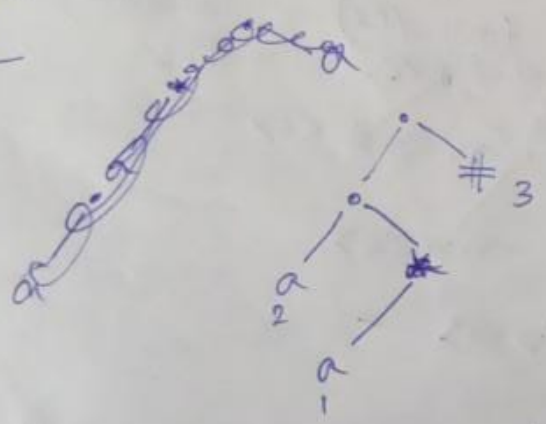


Step (ii) Construct syntax tree → cat node (·), or node (|), * node (*)

Eg: $(a|b)^* abb\#$



aa^*



Step (iii) label every leaf node by unique integer no.

Step (v) construct 4 tables.

(1) nullable (n)

(2) first pos (n)

(3) last pos (n)

(4) follow pos (P)

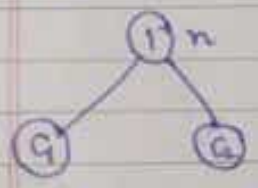
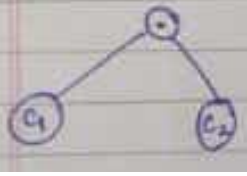
* Step 4:

* Nullable(n) is true for a syntax tree node (n) iff the sub-expression represented by n # ε in its language. That is subexpression can be made null or empty string.

* Firstpos(n) is the set of positions in the subtree rooted at n that corresponds to the first symbol of atleast one string in the language of the sub expression rooted at n. (1,2,3)

* Lastpos(n) is the set of positions in the subtree rooted at n that corresponds to the last symbol of atleast " " at n.

Rules for computing nullable firstpos and lastpos.

Node n	nullable (n)	firstpos(n)	lastpos(n)
A leaf labeled ε	true	ϕ	ϕ
A leaf with position i	false	{i}	{i}
	nullable (c_1) or nullable(c_2)	firstpos(c_1) ∪ firstpos (c_2)	lastpos(c_1) ∪ lastpos(c_2)
	nullable (c_1) and nullable (c_2)	if (nullable(c_1)) firstpos(c_1) ∪ firstpos (c_2)	if (nullable(c_1)) lastpos(c_1) ∪ lastpos(c_2) else

What is nullable?

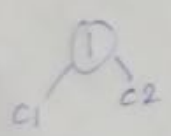
Nullable(n) is true for a syntax tree node n iff the sub-expression represented by n has ∈ in its language i.e. the sub-expression can be made null or empty string.

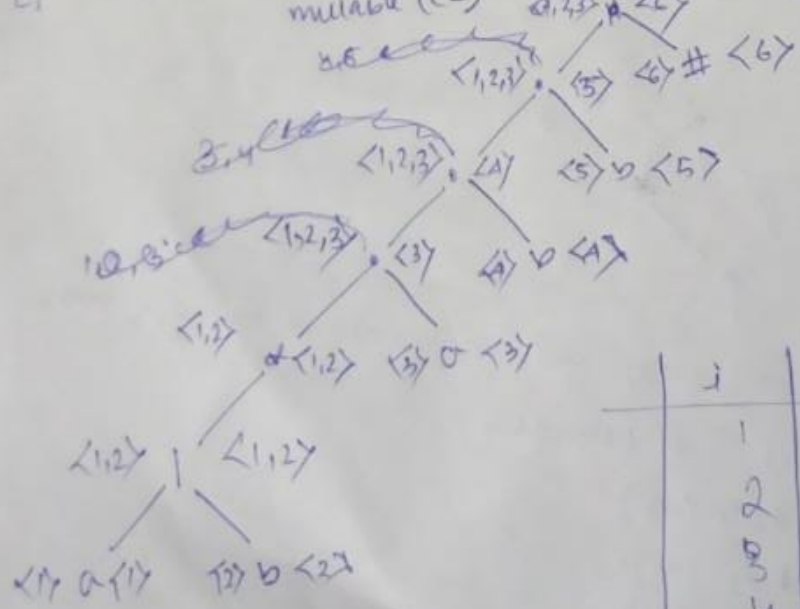
Firstpos(n) is the set of positions in the subtree rooted at n that correspond to the first symbol of atleast one string in the language of the sub exp. rooted at n.

Lastpos(n)

It is the set of positions in the subtree rooted at n that correspond to the last symbol of . . .

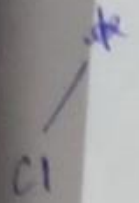
Rules for computing nullable, lastpos & firstpos.

Node n	nullable(n)	firstpos(n)	lastpos(n)
A leaf labelled ∈	true	false	false
A leaf with position i	false	{i}	{i}
	~~false~~ nullable(c1) or nullable(c2)	firstpos(c1) ∪ firstpos(c2)	lastpos(c1) ∪ lastpos(c2)



5, 6

i	followpos(i)
1	{1, 2, 3}
2	{1, 2, 3}
3	{4}
4	{5}
5	{6}
6	~~{}~~ —

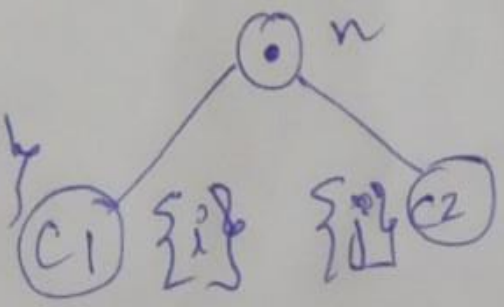
node	nullable(n)	firstpos(n)	lastpos(n)
	nullable(c1) and nullable(c2)	• if nullable(c1) firstpos(c1) ∪ firstpos(c2) • else firstpos(c1)	• if nullable(c2) lastpos(c2) ∪ lastpos(c1) • else lastpos(c2)
	true	firstpos(c1)	lastpos(c1)

example :

Followpos (i)

→ If n is a ⊙ node with left child c1 and right child c2 and i is a position in lastpos(c1) then all position in firstpos(c2) are in followpos(i)

→ If n is a * node and i is a position in lastpos(n) then all position in firstpos(n) are in followpos(i)



⇒ follow(i) = {j}

⇒ followpos(i) = {j}.

{j} ⊙ {i} ⇒ follow(i) = {j}

(v) Construct DFA

State	a	b
A	B	A
B	~~A~~ B	C
C	~~A~~ B	D
D	~~A~~ B	A

I/P

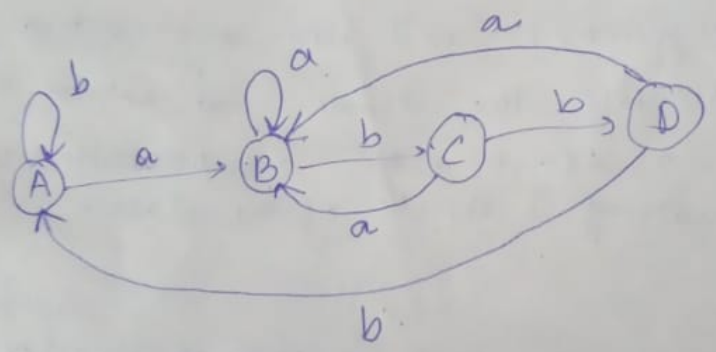
firstpos(root) = ~~<1,2,3>~~ = A

<1,2,3,4> = B

<1,2,3,5> = C

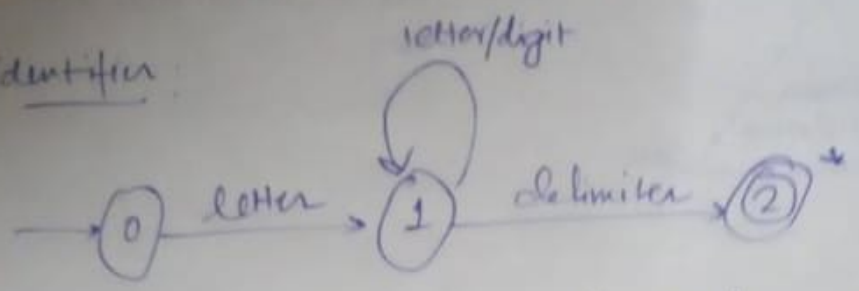
<1,2,3,6> = D

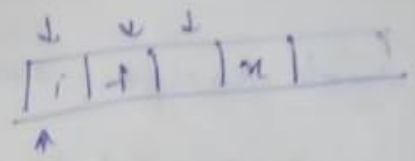
∴ D ⇒ accepting state.



Design of a lexical analyser

Identifier





Beginning pointer
Look ahead pointer.

State 0 : c = GETCHAR()
if LETTER(c) then
get state 1.
else FAIL

State 1 : c = GETCHAR()
if (LETTER c) or DIGIT(c) then get
state 1.

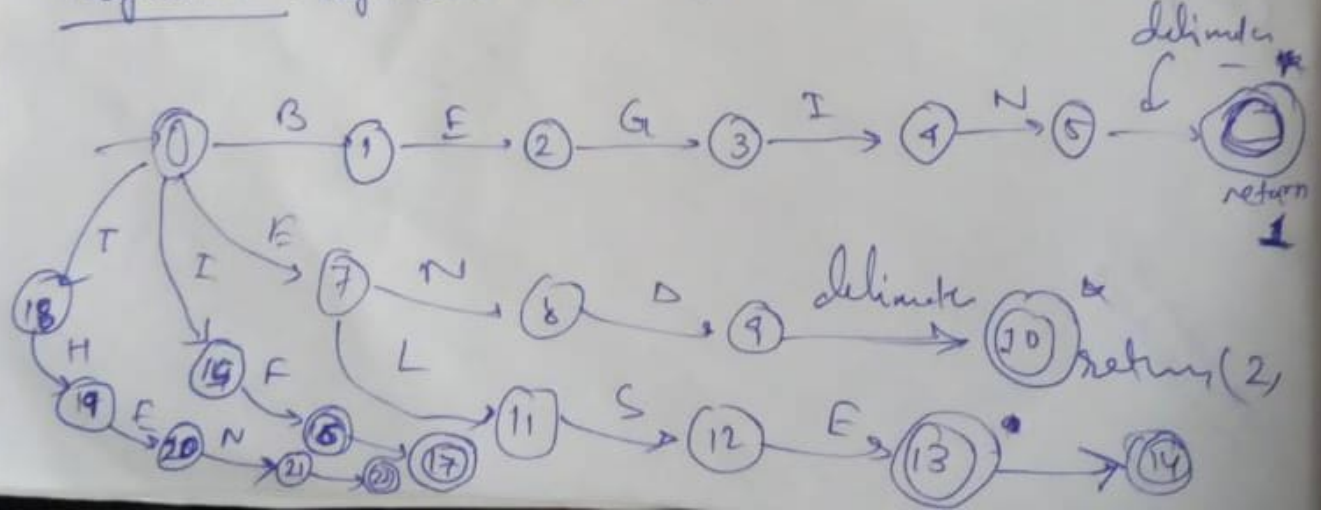
else if DELIMETER(c) then go
to state 2

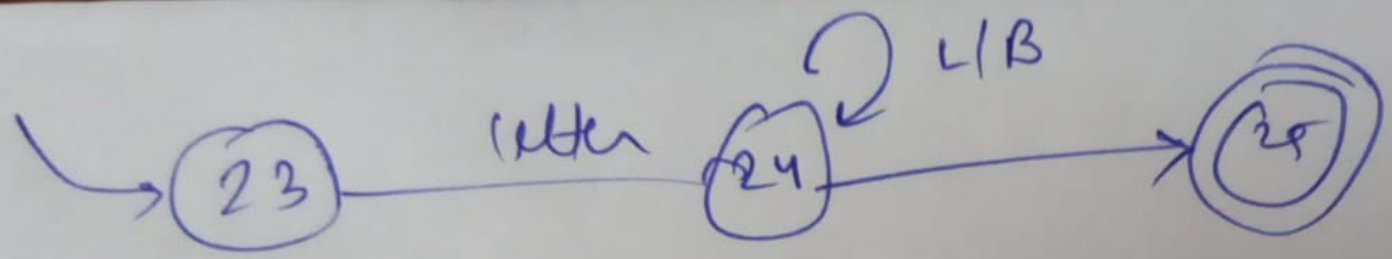
else
FAIL

State2 : RETRACT()
return (id, install()),

* - retract

Keyword : begin end else if then




23
letter
24
L/B
25