SR 22

St. Peter’s Engineering College (Autonomous) Dullapally (P), Medchal, Hyderabad – 500100. QUESTION BANK					Dept.	:	CSE(AIML)
					Academic Year 2023-24		
Subject Code	**:**	**AS22-66PC02**	**Subject**	**:**	**AUTOMATA THEORY & COMPILER DESIGN**		
Class/Section	**:**	**B. Tech.**	**Year**	**:**	**II**	**Semester**	**: II**

BLOOMS LEVEL					
Remember	**L1**	**Understand**	**L2**	**Apply**	**L3**
Analyze	**L4**	**Evaluate**	**L5**	**Create**	**L6**

*****

Q. No	Question (s)	Marks	BL	CO
	UNIT – II			
1	Define Empty Set.	**1M**	L1	C222.1
	Define Null String.	**1M**	L1	C222.1
	Define Identity Rules for Regular Expressions.	**1M**	L1	C222.1
	Define Regular sets.	**1M**	L1	C222.1
	Define Parse Tree.	**1M**	L1	C222.1
2	Discuss about Regular Sets and Regular Language with examples.	**3M**	L2	C222.1
	Discuss about Grammars and Derivation Trees with examples.	**3M**	L2	C222.1
	Discuss about Context Free Grammars with examples.	**3M**	L2	C222.1
	Discuss about Ambiguous Grammars with example.	**3M**	L2	C222.1
	Derive the Regular Expression for the following DFA,	**3M**	L2	C222.1
3	State and Prove Arden’s Theorem.	**5M**	L5	C222.1
	Construct an NFA and NFA-Ɛ for the regular expression 11+00.	**5M**	L6	C222.1
	Discuss about regular grammar, right linear grammar and left linear with examples.	**5M**	L2	C222.1
	Draw a Parse Tree for the Language L={$a^n b^n$, n>=0} and the CFG with Productions are S-> aSb, S-> Ɛ.	**5M**	L6	C222.1
	Construct an NFA-Ɛ for the regular expression 110(0+1)*.	**5M**	L6	C222.1
4	**a)** Construct an NFA-Ɛ for the regular expression (0+1)*11.	**5M**	L6	C222.1

	b) Explain in detail about Parse Trees, Left and Right Most Derivations.	**5M**	L4	C222.1
	Construct a DFA, NFA and NFA-Ɛ for any regular expression	**10M**	L2	C222.1
	State and Prove Pumping Lemma.	**10M**	L5	C222.1

ANSWERS

1.

Define Empty Set.
Empty Set is a Set with no String, denoted as { }.

Define Null String.
Null String is a string with length zero, denoted with { ε}.

Define Identity Rules for Regular Expressions.
Define Regular sets.

Regular Sets: Any set represented by a regular expression is called a regular set.
If a, b are the elements of Σ, then regular expressions
a denotes the set {a}.
a + b denotes the set {a, b}.

Regular Expressions: Regular Expressions are useful for representing certain sets of strings in an algebraic fashion. RE describes the language accepted by finite automata.
Regular Expression over Σ: Any terminal symbol/element of Σ is RE
Example: Φ, ϵ, a in Σ
Φ is a regular expression and denotes the empty set.
ϵ is a regular expression and denotes the set {ϵ}.
a is a regular expression and denotes the set {a}.

Identity Rules for Regular Expressions:
P and Q are two equivalent regular expressions (i.e., P and Q represent the same set of strings), then to simplify the regular expressions, the following identity rules can be used:
1. Φ + R = R, e + R = R + e
2. eR = Re = R
3. R + R = R
4. RR* = R*R = R+

Define Parse Tree.
A parse tree (also known as a syntax tree) is a tree representation that shows how a string derived from a formal grammar is syntactically constructed. It breaks down the structure of a string according to the rules of the grammar and shows how the string can be generated from the start symbol.

2.

Discuss about Regular Sets and Regular Language with examples.

Regular Expressions are useful for representing certain sets of strings in an algebraicfashion. RE describes the language accepted by finite automata.

Any set represented by a regular expression is called a regular set.
If a, b are the elements of Σ, then regular expressionsa denotes the set {a}.
a + b denotes the set {a, b}.

A regular language is a language that can be expressed with a regular expression or a deterministic or non-deterministic finite automata or state machine. A language is a set of strings which are made up of characters from a specified alphabet, or set of symbols.

Example 1:
Write the regular expression for the language accepting all the string which are starting with 1 and ending with 0, over ∑ = {0, 1}.
Solution:
In a regular expression, the first symbol should be 1, and the last symbol should be 0. The r.e. is as follows:

1. R = 1 (0+1)* 0

Example 2:
Write the regular expression for the language starting and ending with a and having any having any combination of b's in between.
Solution:
The regular expression will be:

1. R = a b* b

Discuss about Grammars and Derivation Trees with examples.

Derivation tree is a graphical representation for the derivation of the given production rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from a given set of production rules. The derivation tree is also called a parse tree.
Parse tree follows the precedence of operators. The deepest sub-tree traversed first. So, the operator in the parent node has less precedence over the operator in the sub-tree.
A parse tree contains the following properties:

1. The root node is always a node indicating start symbols.
2. The derivation is read from left to right.
3. The leaf node is always terminal nodes.
4. The interior nodes are always the non-terminal nodes.

Example 1:
Production rules:

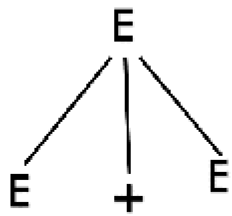
1. E = E + E
2. E = E * E

3. E = a | b | c

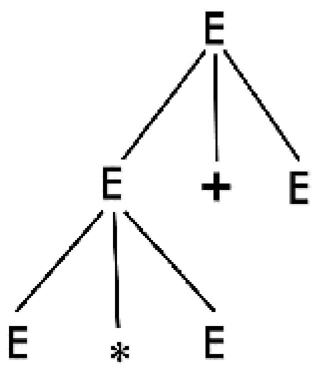
Input

1. a * b + c

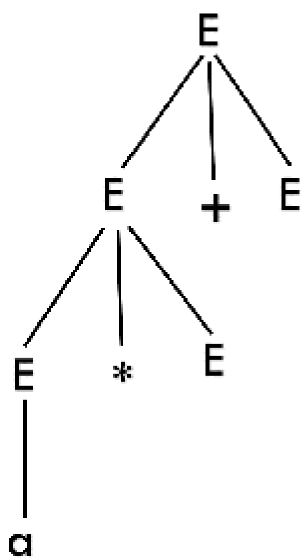
Step 1:



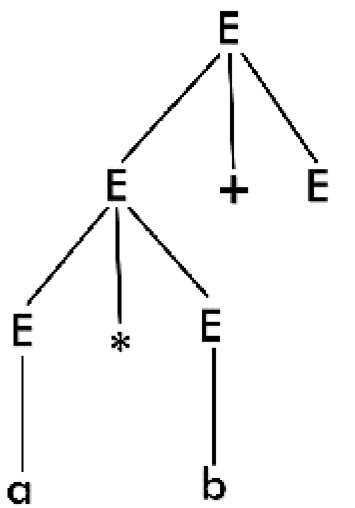
Step 2:



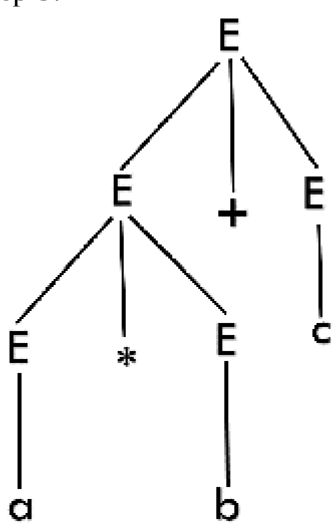
Step 2:



Step 4:



Step 5:



Discuss about Context Free Grammars with examples.

Context free grammar is a formal grammar which is used to generate all possible strings in a given formal language.

Context free grammar G can be defined by four tuples as:

1. G= (V, T, P, S)

Where,

G describes the grammar

T describes a finite set of terminal symbols.

V describes a finite set of non-terminal symbols

P describes a set of production rules

S is the start symbol.

In CFG, the start symbol is used to derive the string. You can derive the string by repeatedly replacing a non-terminal by the right hand side of the production, until all non-terminal have been replaced by terminal symbols.

Example:

L= {wcw^R | w € (a, b)*}

Production rules:

1. S → aSa
2. S → bSb
3. S → c

Now check that abbcbba string can be derived from the given CFG.

1. S ⇒ aSa
2. S ⇒ abSba
3. S ⇒ abbSbba
4. S ⇒ abbcbba

By applying the production S → aSa, S → bSb recursively and finally applying the production S → c, we get the string abbcbba.

Discuss about Ambiguous Grammars with example.

A grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string. If the grammar is not ambiguous, then it is called unambiguous.

If the grammar has ambiguity, then it is not good for compiler construction. No method can automatically detect and remove the ambiguity, but we can remove ambiguity by re-writing the whole grammar without ambiguity.

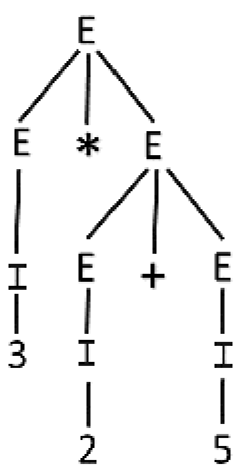
Example 1:

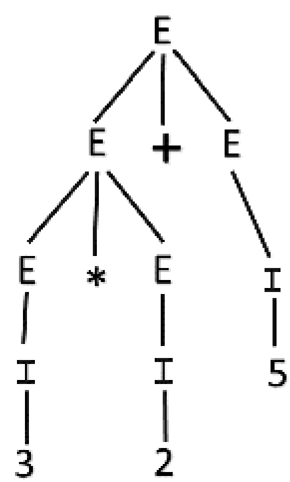
Let us consider a grammar G with the production rule

1. E → I
2. E → E + E
3. E → E * E
4. E → (E)
5. I → ε | 0 | 1 | 2 | ... | 9

Solution:

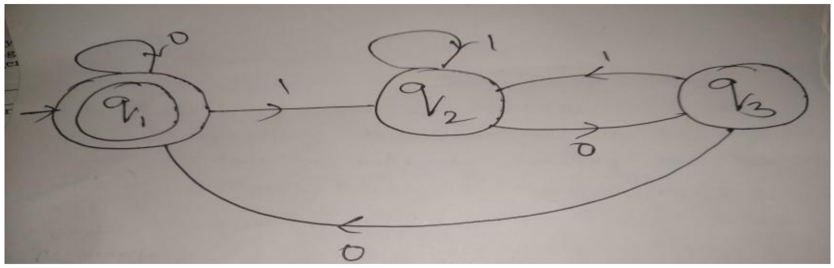
For the string "3 * 2 + 5", the above grammar can generate two parse trees by leftmost derivation:



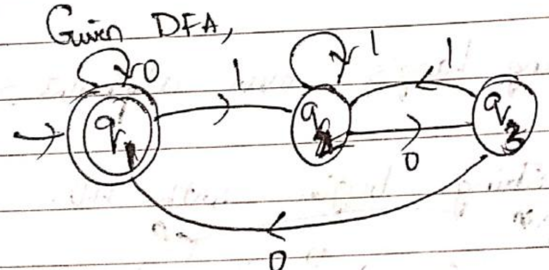


Since there are two parse trees for a single string "3 * 2 + 5", the grammar G is ambiguous.

Derive the Regular Expression for the following DFA,



Given DFA,



Write Eq of each st

$q_1 = q_1 0 + q_3 0 + \epsilon$ → (1)

$q_2 = q_1 1 + q_2 1 + q_3 1$ → (2)

$q_3 = q_2 0$ → (3)

→ For initial st, 'ε' is written because without taking any I/p, we start from initial st.

Substitute (3) in (2),

(3) $\Rightarrow q_2 = q_1 1 + q_2 1 + q_2 01$

$q_2 = q_1 1 + q_2(1+01) \rightarrow (4)$

$R = Q + RP$

$q_2 = q_1 1(1+01)^*$

$R = q_2, \quad Q = q_1 1, \quad P = (1+01)$

$\Rightarrow \quad R = QP^*$

(4) $\Rightarrow q_2 = q_1 1 (1+01)^* \rightarrow (5)$

||ly we will solve it for eq (1),

(1) $\Rightarrow q_1 = q_1 0 + q_3 0 + \epsilon$

$= \cancel{q_1 0}$

Substitute (3) in (1),

$\Rightarrow q_1 = q_1 0 + q_2 00 + \epsilon \rightarrow (6)$

$= \cancel{q_1 0 + q}$

(5) in (6),

$q_1 = q_1 0 + q_1 1(1+01)^* 00 + \epsilon$

$q_1 = q_1 (0 + 1(1+01)^* 00) + \epsilon$

$R = q_1, \quad Q = \epsilon, \quad P = (0 + 1(1+01)^* 00)$

$R = RP + Q$

$R = QP^*$

$q_1 = \epsilon (0 + 1(1+01)^* 00)^*$

$\cancel{q_1 = (0+1(}$

$$\boxed{\therefore q_1 = (0 + 1(1+01)^* 00)^*}$$

→ ~~Because~~ Why we derived the eq in the form of q_1 means here q_1 is the Final st.

3.
State and Prove Arden's Theorem.

Arden's Equation

$R = Q + RP$
$R = QP^*$

→ This is to check the Equivalence b/w 2 RE's.
→ In the Conversion of DFA to RE.

Condition to apply Arden's Theorem.

→ FA should not contain ϵ-transitions.
→ FA " have only one initial st.

Statement: If P & Q are two RE's & P does not have any ϵ-transitions, then the Eq $R = Q + RP$ will have unique solution $R = QP^*$. → (1)

Proof: $R = Q + RP$

Apply Eq (1)

$R = Q + (QP^*)P$
$R = Q\epsilon + QP^*P$
$= Q(1 + P^*P)$
[∵ In RE's, $\epsilon = 1$} Automata Concepts, $\epsilon = 1$)
$= Q(\epsilon + P^*P)$
[∵ Rule 8: $\epsilon + r^*r = \epsilon + rr^* = r^*$)

$\boxed{R = QP^*}$

$\boxed{\therefore R = QP^*}$

→ Let us derive the proof in the other way.
- We know that RE is recursive

Proof: $R = Q + RP \rightarrow (2)$

Apply Eq (2)

$R = Q + (Q+RP)P$

$R = Q + QP + RP^2 \rightarrow (3)$

Now again (2) in (3),

$R = Q + QP + (Q+RP)P^2$

$= Q + QP + QP^2 + RP^3 \rightarrow (4)$

Now (2) in (4),

$R = Q + QP + QP^2 + (Q+RP)P^3$

$R = Q + QP + QP^2 + QP^3 + RP^4$.

So, ~~we~~ if we substitute $R = Q + \cancel{RP}\, RP$ in every step then

$R = Q(\epsilon + P + P^2 + P^3 + \ldots\ldots)$

(∵ According Automata $\epsilon = 1$)

(∵ $\Sigma^* = \epsilon + \Sigma^1 + \Sigma^2 + \Sigma^3 \ldots\ldots$)

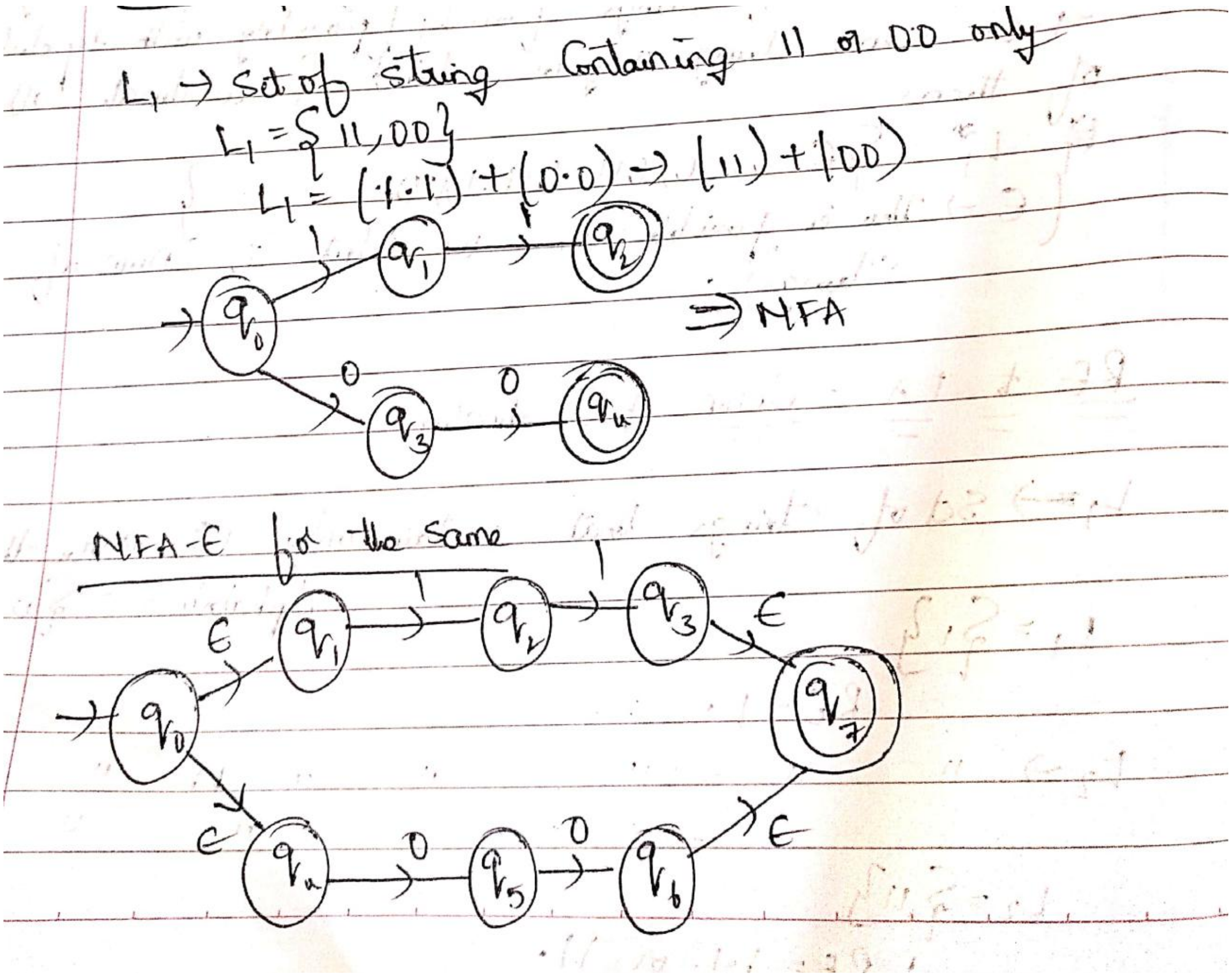
$\boxed{\therefore R = QP^*}$

$$\boxed{\begin{aligned} \therefore\ & R = Q + RP \\ & R = QP^* \end{aligned}}$$

This is Arden's Theorem

Hence Proved

Construct an NFA and NFA-Ɛ for the regular expression 11+00.



Discuss about regular grammar, right linear grammar and left linear with examples.

Regular grammar: The regular grammars generate strings of regular languages if the grammar is rightlinear or left linear.

Regular grammar *is a four-tuple G* = (*N*, Σ, *P*, *S*), *where*

1. *N is an alphabet called the set of* **nonterminals.**
2. Σ *wan alphabet called the set of* **terminals**, *with* Σ ∩ *N* = Ø.
3. *P is a finite set of* **productions** *or* **rules** *of the form A* → *w*, *where A* ∈ *N and w* ∈ Σ**N* ∪ Σ*.
4. *S is the* **start symbol**, *S* ∈ *N*.

The productions must be in the form:

A ⇢ xB

A ⇢ x

A ⇢ Bx

Left linear grammar (LLG):

In LLG, the productions are in the form if all the productions are of the form
A ⇢ Bx
A ⇢ x
Where A, B ∈ V and x ∈ T*

Right linear grammar (RLG):

In RLG, the productions are in the form if all the productions are of the form
A ⇢ xB
A ⇢ x
where A,B ∈ V and x ∈ T*

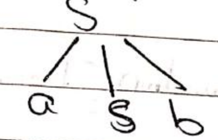
Draw a Parse Tree for the Language L={$a^n b^n$, n>=0} and the CFG with Productions are S-> aSb, S-> Ɛ.

$L_1 = \{a^n b^n, n \geq 0\}$

→ The given Lang 'L' is not a Reg-Lang but

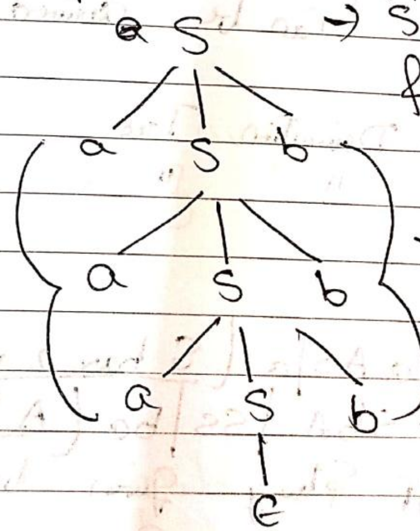
we can write CFG for this Lang
So $L_1 = \{\epsilon, ab, aabb, aaabbb, \ldots\}$
So, the CFG for the given Lang 'L' is
S → aSb
S → ε
Non-Terminals are considered as a fun
S → aSb
→ The fun 'S', ~~is~~ check for the ~~Terminal~~ I/P Sym 'a' & ~~ag~~ call 'S' & then check for I/P Sym 'b'.
(or) S is 'ε'.

→ So we can write a tree for the I/p String aaabbb for this production i.e



Now again expand 'S'

aaabbb ⇒



→ 'S' expand as S→aSb & call for 'a' check for 'a' & call 'S' & check for 'b'

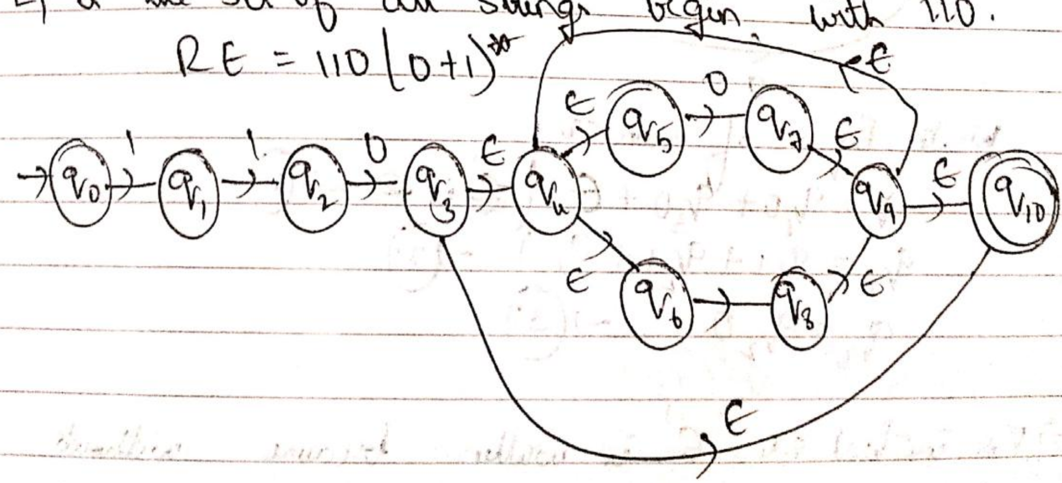
→ ||ly again expand S → aSb and expand S→aSb

→ And finally expand S→ϵ.

Construct an NFA-Ɛ for the regular expression 110(0+1)*.

L_1 is the set of all strings begin with 110.

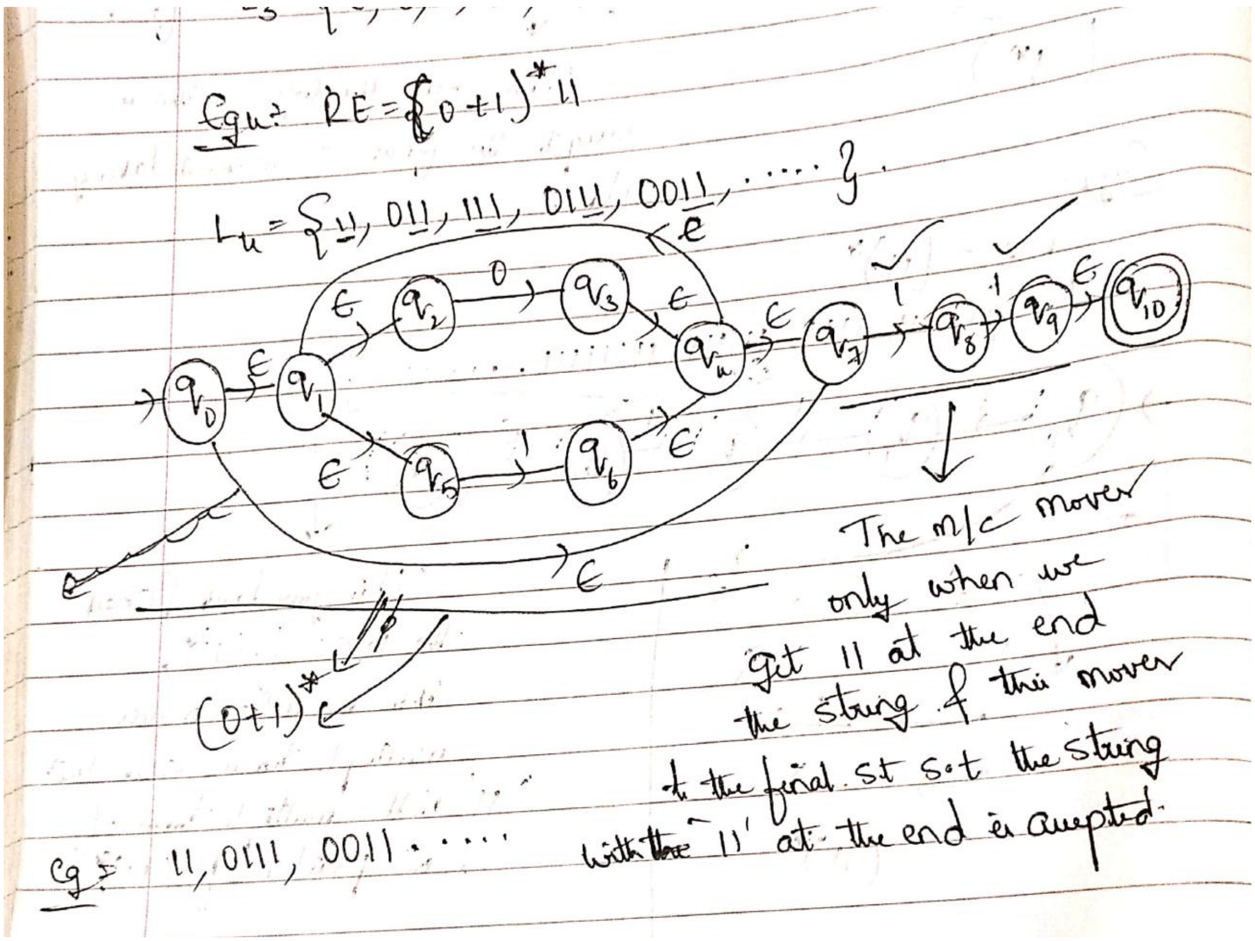
RE = 110(0+1)*



4.

a) Construct an NFA-Ɛ for the regular expression (0+1)*11.	
b) Explain in detail about Parse Trees, Left and Right Most Derivations.	
Construct a DFA, NFA and NFA-Ɛ for any regular expression	
State and Prove Pumping Lemma.	

a) Construct an NFA-Ɛ for the regular expression (0+1)*11.



b) Explain in detail about Parse Trees, Left and Right Most Derivations.

Derivations mean replacing a given string's non-terminal by the right-hand side of the production rule. The sequence of applications of rules that makes the completed string of terminals from the starting symbol is known as derivation. The parse tree is the pictorial representation of derivations. Therefore, it is also known as derivation trees. The derivation tree is independent of the other in which productions are used.

A parse tree is an ordered tree in which nodes are labeled with the left side of the productions and in which the children of a node define its equivalent right parse tree also known as syntax tree, generation tree, or production tree.

A Parse Tree for a CFG G =(V,∑, P,S) is a tree satisfying the following conditions –

- Root has the label S, where S is the start symbol.
- Each vertex of the parse tree has a label which can be a variable (V), terminal (Σ), or ε.
- If $A \rightarrow C_1,C_2 \ldots\ldots C_n$ is a production, then $C_1,C_2 \ldots\ldots C_n$ are children of node labeled A.
- Leaf Nodes are terminal (Σ), and Interior nodes are variable (V).
- The label of an internal vertex is always a variable.
- If a vertex A has k children with labels $A_1,A_2 \ldots\ldots A_k$,then $A \rightarrow$

$A_1,A_2 \ldots\ldots A_k$ will be production in context-free grammar G.

Yield − Yield of Derivation Tree is the concatenation of labels of the leaves in left to right ordering.

Example1 − If CFG has productions.

S → a A S | a

S → Sb A | SS | ba

Show that S ⇒ *aa bb aa & construct parse tree whose yield is aa bb aa.

Solution

S $\Rightarrow^{lm}$ lm a A−−A_ S

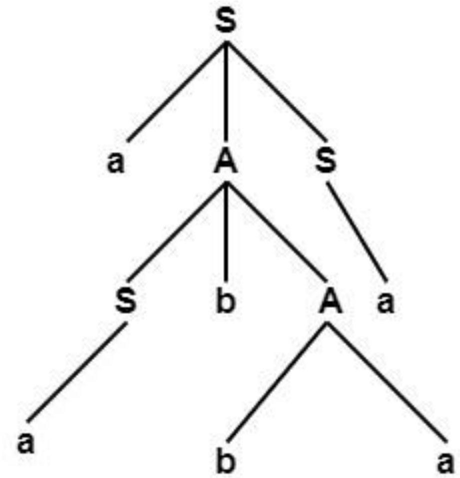
⇒ a Sb−−−Sb_ A S

⇒ aa b A−−A_ S

⇒ aa bba S−−S_

∴ S ⇒ * aa bb aa

Derivation Tree



Yield = Left to Right Ordering of Leaves = aa bb aa

Example2

Consider the CFG

S → bB | aA

A → b | bS | aAA

B → a |aS | bBB

Find (a) Leftmost

- Rightmost Derivation for string b aa baba. Also, find derivation Trees.

Solution

- **Leftmost Derivation**

S⇒b B−−B_

⇒ bb B−−B_B

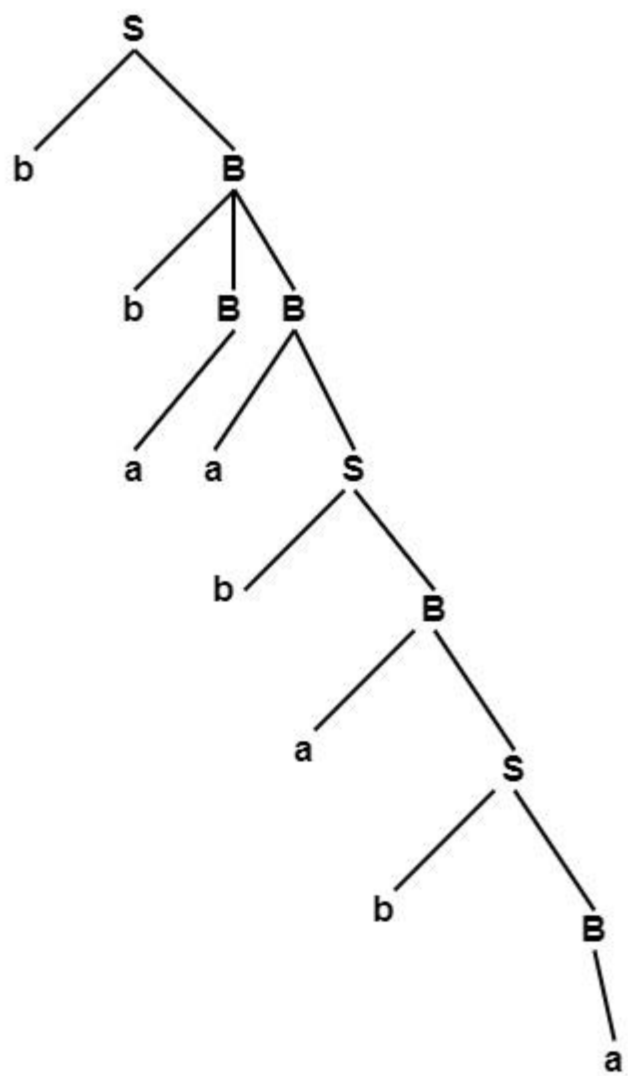
⇒ bbaB−−B_

⇒ bbaaS−−S_

⇒ bb aabB−−−bB_

⇒ bb aa b aS−−−aS_
⇒ bb aa bab B−−B_
⇒ bb aa ba ba

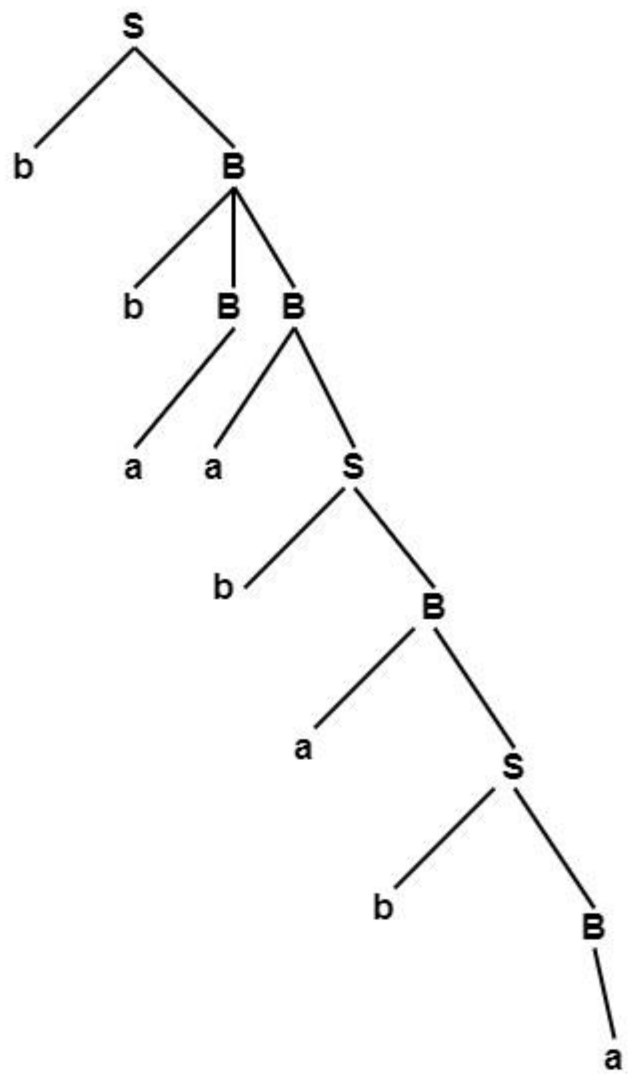
Derivation Tree for Leftmost Derivation



- **Rightmost Derivation**

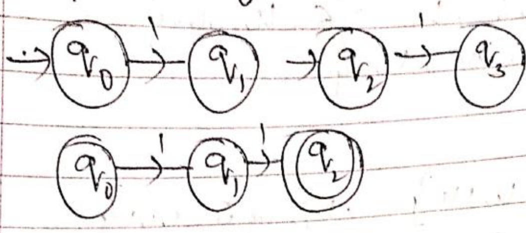
S ⇒ bB−−B_
⇒ bb BB−−B_
⇒ bbBaS−−S_
⇒ bbBabB−−B_
⇒ bbBabaS−−S_
⇒ bbBababB−−B_
⇒ bbB−−B_abab a
⇒ bbaababa

Derivation Tree for Rightmost Derivation

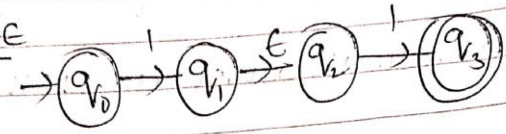


Construct a DFA, NFA and NFA-Ɛ for any regular expression.

Equivalence of RE to FA



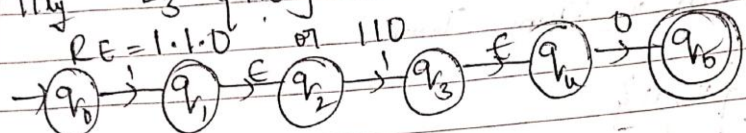
NFA-ε



for $RE = 1.1 \mid 11.$

IIIly $L_3 = \{110\}$

$RE = 1.1.0$ or 110

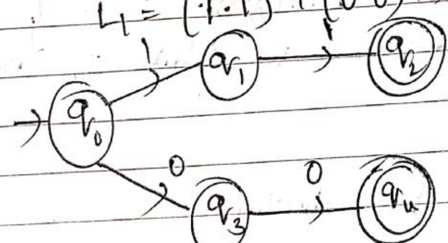


Union Operator

$L_1 \rightarrow$ Set of string Containing 11 or 00 only

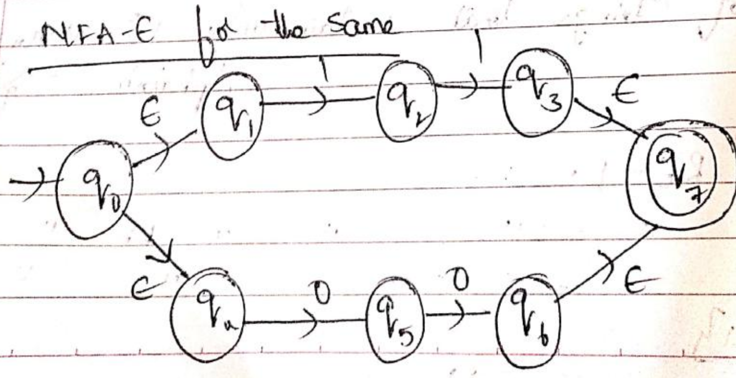
$L_1 = \{11, 00\}$

$L_1 = (1.1) + (0.0) \rightarrow (11) + (00)$



$\Rightarrow$ NFA

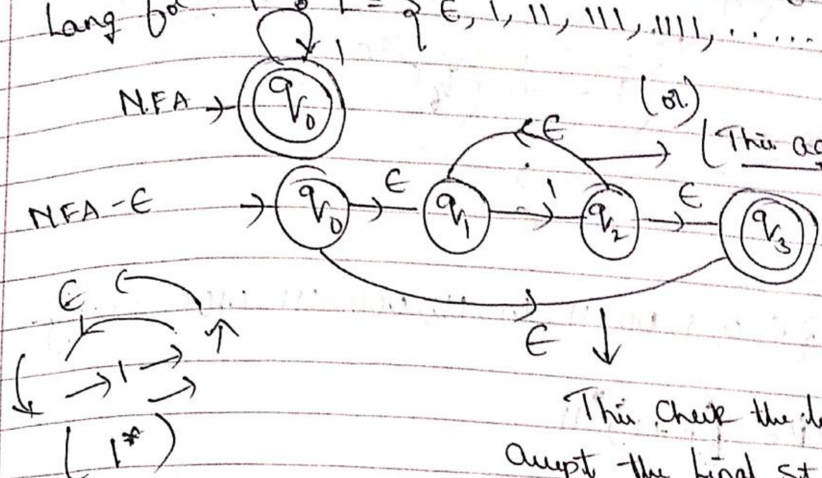
NFA-ε for the same



RE to FA – Closure

$\Sigma = \{0,1\}$ over the Alphabet, the RE $= 1^*$

Lang for 1^* : $L = \{\epsilon, 1, 11, 111, 1111, \ldots\ldots\}$.



(or)

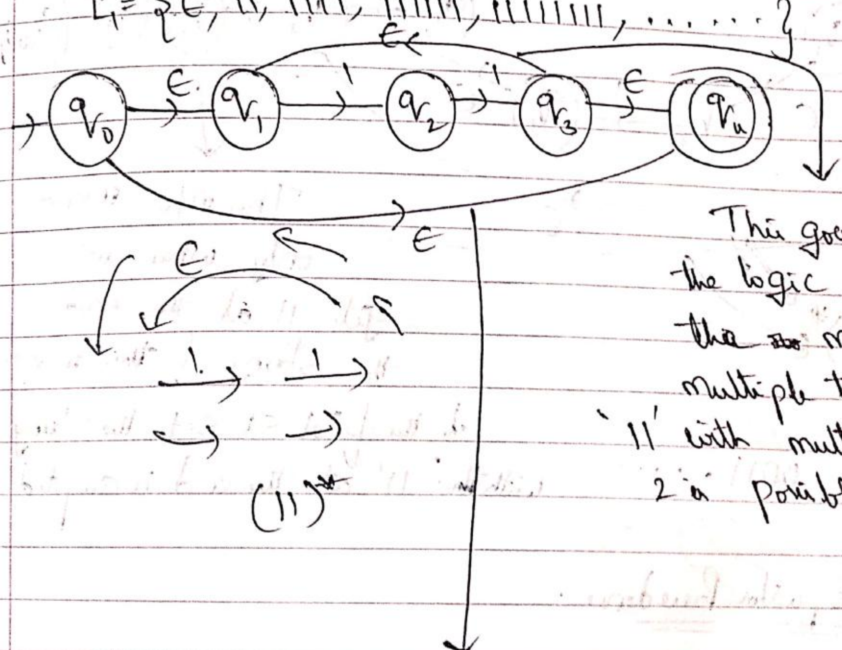
(This again goes back & check the logic once again for 1^*)

This check the logic, this is accept the final st without taking I/p.

Eg2:

$RE = (11)^*$

$L_1 = \{\epsilon, 11, 1111, 111111, 11111111, \ldots\ldots\}$



This goes back & check the logic for $(11)^*$, the M/c moves multiple times such that '11' with multiple ~~times~~ of 2 is possible for $(11)^*$.

This check the logic, this accept the final st without taking the I/p.

State and Prove Pumping Lemma.

→ ~~Lang~~ Lang is a common " " Symbol
→ String are " " " repeated
Stmt → If a substring of a string is repeated many times & if the resultant string is also available in a lang 'L' then we can say it as Regular.
→ And repeating in the ~~string~~ substring is Called as "Pumping Lemma"
→ Repeating means Pumping
→ Lemma " Substring.

~~Steps~~: Steps:
→ Consider lang as a Regular
→ Assume a Constant 'c' & select the string 'w' from L such that $|w| \geq c$. (∵ length $(w) \geq c$)
→ Divide the w as xyz (∵ 3 strings).
→ $|y| > 0$
→ $|xy| \leq c$
→ for $i \geq 1$ every string xy^iz belongs to 'L'.
→ ~~No~~
Now here all these Conditions need to satisfied & if any one of the Condition is ~~so~~ violated then the given Lang is not a Regular Lang.

Example
→ Now prove that the Lang $L = \{a^n b^n \mid n \geq 0\}$ is not Regular.

Proof :-

$L = \{ \epsilon, ab, aabb, aaabbb, \ldots \}$.

→ Now let the constant 'c' be $c = 6$.

So, $w = aaabbb$, $|w| = 6 \geq c$. ✓

Now we have to divide 'w' into 3 Substrings.

$w = xyz$ where x, y, z are 3 ~~Strings~~ Substrings

$w = aaabbb$

now let $x = aa$, $y = ab$, $z = bb$. (Example1)

∴ Now $|y| = 2 > 0$ ✓

∴ $|xy| = |aabb| = 4 \leq c$ ✓

→ If $i = 0$,

$xy^0z = xz = aabb$ ✓ (∵ aabb is a Substring in L).

If $i = 1$, $xy^1z = aaabbb$ ✓

If $i = 2$, $xy^2z =$ ~~aaabbb~~ aaababbb ⊗

Now here 'aaababbb' is not available in the Lang 'L'.

~~here~~

→ L is equal num of a's followed by equal num of b's. So, this substring is not available in 'L'.

So, here the string aaababbb is not a Valid string.

→ So, now we will change the substrings ~~x~~ x, y & z

Example2:

$x = a$, $y = aa$, $z = bbb$. ($|w| = 6 \geq c$)

∴ Now $|y| = 2 > 0$ ✓

$|\cancel{xy}| = |xy| = 3 \leq c$ (aaa) ✓

If $i = 0$, $xy^0z = xz = abbb$ ⊗

∴ So, here the ~~St~~ substring 'abbb' substring is not available in the given Lang 'L'.

Example3: $x = aaa$, $y = b$, $z = bb$

Date

$\therefore |y| = b = 1 > 0$ ✓

$|xy| = |aaab| = 4 \leq c$.

If $i=0$, $xy^0z = xz = aaabb$ $\times$

$\therefore$ So aaabb is not a string in the given Lang 'L'.

$\therefore$ By these 3 examples, we can simply prove that the given Lang 'L' is not Regular.