Unit-3

Regular Expressions

Regular Languages → A language is said to be a regular language if and only if some finite state machine recognizes it.

The languages which are not recognized by any finite state machine and which require memory are not regular languages.

For Example:- Let we have ababbababb. As we see this language should follow the rule that, the first five letters ababb is repeated again. So, in order to repeat next time we should have information about what to be repeated (i.e, ababb in this case) which requires memory to store ababb. As we know that memory of FSM is very limited and it cannot store or count string, so this language can not be designed using FSM making it a language that is not regular.

⊛ **Regular Expression** → Regular expressions are those algebraic expressions used for describing regular languages, the languages accepted by finite automation. Regular expressions offer a declarative way to express the strings we want to accept.

Any regular expression is composed of two components; symbols and operators. Symbol, Σ is a set of inputs & operators are union (U), concatenation (.), Kleen Closure (*), Positive closure (+).

Example: Consider a regular expression $(0 \cup 1)01^*$; where 0,1 are symbols and $\cup$, * are the operators. Then, the language described by this expression is the set of all binary strings:

→ that start with either 0 or 1. (indicated by $(0 \cup 1)$ ∪ means or operation)

→ for which the second symbol is 0

→ and that end with zero or more number of 1's. (Indicated by 1^*)

Hence, the language described by this expression is {00, 001, 00111 ... , 10, 101, 10111, ...}.

Scanned by TapScanner

Formal Definition of Regular Expression:

Let Σ be an alphabet, the regular expression over the alphabet Σ are defined inductively as follows;

⇒ ϕ is a regular expression representing empty language.

⇒ ϵ is a regular expression representing the language of empty strings.

⇒ If 'a' is a symbol in Σ, then 'a' is a regular expression representing the language {a}.

⇒ If 'r' and 's' are the regular expressions representing the language L(r) and L(s) then:

→ r ∪ s is a regular expression denoting the language L(r) ∪ L(s).

→ r.s is a regular expression denoting the language L(r).L(s).

→ r* is a regular expression denoting the language (L(r))*

→ (r) is a regular expression denoting the language (L(r)).

Operators of Regular Expressions:

i) Union (∪/ |): If L_1 and L_2 are any two regular languages then,

$L_1 \cup L_2 = \{s \mid s \in L_1, \text{ or } s \in L_2\}$

e.g. let $L_1 = \{00, 11\}$, $L_2 = \{\epsilon, 10\}$

$L_1 \cup L_2 = \{\epsilon, 00, 11, 10\}$

Nothing, this simply means or operation. i.e s such that s belongs to L_1 or s belongs to L_2

but this symbol used here denotes empty string

ii) Concatenation (.): If L_1 and L_2 are any two regular languages then,

$L_1 \cdot L_2 = \{L_1 \cdot L_2 \mid L_1 \in L_1 \text{ and } L_2 \in L_2\}$

e.g. $L_1 = \{00, 11\}$, $L_2 = \{\epsilon, 11\}$

$L_1 \cdot L_2 = \{11\}$.

Concatenation denotes and operation

iii) Kleen Closure (*): If L is any regular language then, Kleen closure of L is;

$L^* = L_0 \cup L_1 \cup L_2 \cup L_3 \cup \ldots$

i.e, $L^* = \bigcup_{i=0}^{\infty}$

i.e, set of all strings of length 0 or more

* has highest precedence
• has next higher precedence
∪/|/+ has lowest precedence.

iv) Positive Closure (+): If L is any regular language then, Positive closure of L is: $L^+ = L_1 \cup L_2 \cup L_3 \cup \ldots$

i.e, $L^+ = L^* - L_0$

i.e, set of all strings expect of length 0 or empty

Scanned by TapScanner

⊛ Applications of Regular Expression:

i) Validation → Determining that a string compiles with a set of formatting constraints, like email address validation, password validation etc.

ii) Search and Selection → Identifying a subset of items from a larger set on the basis of a pattern match.

iii) Tokenization → Converting a sequence of characters into words, tokens (like keywords, identifiers) for later interpretation.

iv) lexer → Used as a lexer/tokenizer in lexical analysis step of compilers.

⊛. Algebraic laws for regular expression:

r, s, l, m, n used all letters are notations for any regular expression

1) Commutativity:

$r+1 = 1+r$ i.e, $r \cup 1 = 1 \cup r$

but $r.1 \neq 1.r$

2) Associativity

$1+(r+s) = (l+r)+s$

Also $1.(r.s) = (1.r).s$

3) Distributive law:

$l(m+n) = lm + ln$

Also, $(m+n)l = ml + nl$

4) Identity law:

$r+\phi = \phi+r = r$ i.e, $\phi \cup r = r$,

i.e, $\epsilon.r = r = r.\epsilon$ ← empty symbol

Also, $\epsilon + r^*r = r^*$

5) Annihilator:

$\phi.r = r.\phi = \phi$

6) Idempotent law of union:

$r+r = r$

7) Law of closure:

$(r^*)^* = r^*$

Also, closure of $\phi = \phi^* = \epsilon$

Also, closure of $\epsilon = \epsilon^* = \epsilon$

Also, Positive closure of r, $r^* = rr^*$.

Scanned by TapScanner

⊛ Regular Expression Examples:

1. Describe the following sets as Regular Expressions.

a) $\{0,1,2\}$ b) $\{\wedge, ab\}$ c) $\{abb, a, b, bba\}$ d) $\{\wedge, 0, 00, 000, \ldots\}$

e) $\{1, 11, 111, 1111, \ldots\}$

Solution,

a) $\{0,1,2\}$

$\Rightarrow R = 0+1+2$ ← Since this set can contain 0 or 1 or 2 not other than that ← or means addition operation.

b) $\{\wedge, ab\}$ ← empty symbol $\wedge$ and $\in$ are same any can be used

$\Rightarrow R = \wedge.ab$ ← during empty symbol we write using . operator

c) $\{abb, a, b, bba\}$ ← Since this set contains any of abb or a or b or bba not other than that so, + operator.

$\Rightarrow R = abb+a+b+bba$

d) $\{\wedge, 0, 00, 000, \ldots\}$ ← i.e, all the string that can be formed using 0 along with empty symbol i.e, all means closure * sign.

$\Rightarrow R = 0^*$

e) $\{1, 11, 111, 111, \ldots\}$ ← i.e, all string that can be formed by 1. But it is not closure of 1. Since closure should contain empty symbol $\wedge$ or $\in$ as well.

$\Rightarrow R = 1^+$ ← 1^+ denotes closure of 1 excluding empty symbol.

2. Consider $\Sigma_1 = \{0,1\}$, some regular expressions over Σ_1;

Understand only

a) 0^*10^* ← 0^* means any number of 0's including empty. i.e, this means it contains single 1 because there's no possibility of other 1 comming.

b) $\Sigma_1^* 1 \Sigma_1^*$ ← This means it contains at least one 1. Since Σ_1^* means all string that can be formed using $\Sigma_1 = \{0,1\}$ of any length.

c) $\Sigma_1^* 001 \Sigma_1^*$ ← this means contains string 001 as substring.

d) $(\Sigma_1\Sigma_1)^*$ or $((0+1)^*.(0+1)^*)$ ← this means string of even length.

e) $1^*(01^*01^*)^*$ ← this means string containing even number of zeros

f) $0^*10^*108108$ ← this means string with exactly three 1's

g) $(1+0)^*.001.(1+0)^* + (1+0)^*.(100).(1+0)^*$ ← this means string that have either 001 or 100

h) $1^*(0+\epsilon).1^*.(0+\epsilon).1^*$ ← this means strings that have at most two 0's within it

i) $(1+0)^*.(11)^*$ ← this means string ending with 11

Scanned by TapScanner

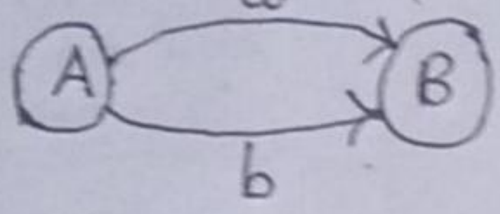
⊛ Equivalence of Regular Expression and Finite Automata:

For the conversion of regular expression to its equivalent finite automata we have some important basic rules which are as follows:-

i) When we have expressions like (a+b)
OR (a∪b)
OR (a|b)

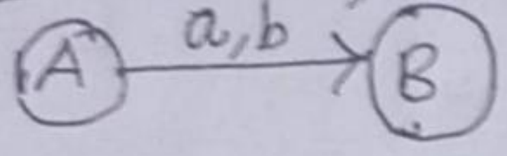
since +, ∪ & | operation denote same thing i.e, OR operation

then, we draw it as,



since a+b, a∪b, a|b means OR operation so on getting any of them can go to next state

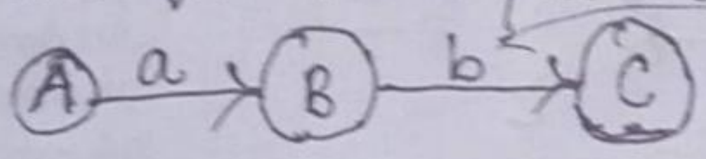
OR



same thing drawn with single transition line

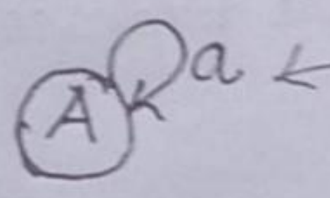
ii) If we have the expression of the form (a.b) then we simply draw new state for each input as follows:-

simply (ab)



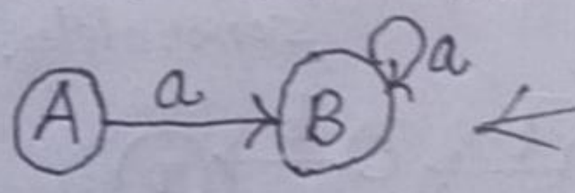
. means and operation so seperate states are made

iii) If we have the expression of the form a* then we create a transition that goes to the state itself without creating next state as follows:-



Since a* denotes zero to any numbers of a's.
i.e, a* = {ε, a, aa, aaa, ...}
ε denotes empty.

iv) If we have the expression of the form a^+ then we draw as follows:-



Since a^+ means any number of a's excluding zero.
i.e, $a^+ = \{a, aa, aaa, ...\}$.
This means at least 1 a should come. so we draw it as like this.

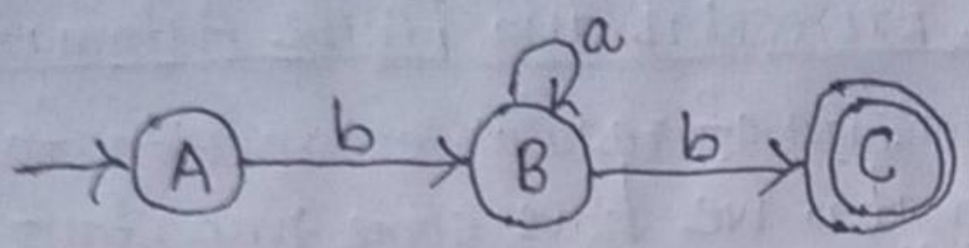
Example 1: Convert the following regular expressions to their equivalent Finite Automata.

i) ba*b
ii) (a+b)c
iii) a(bc)*

Scanned by TapScanner

ii) ba*b

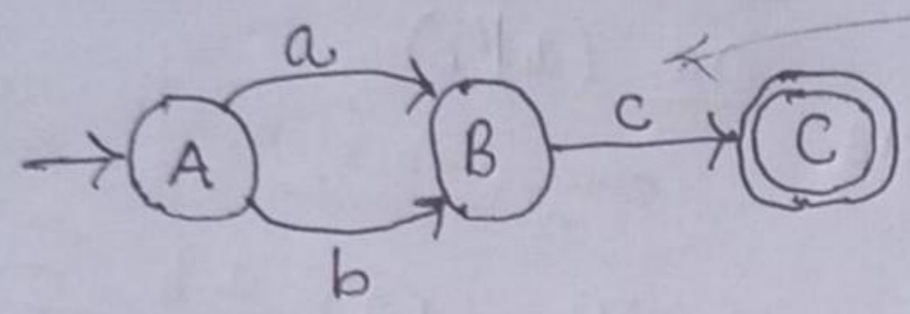
Solution:



Since for first a there is no any operation so we directly go to next state (according to rule ii). Then we have a* in which we applied rule (iii) and finally for b again rule (ii) used. b input is final so C is final state.

iii) (a+b)c

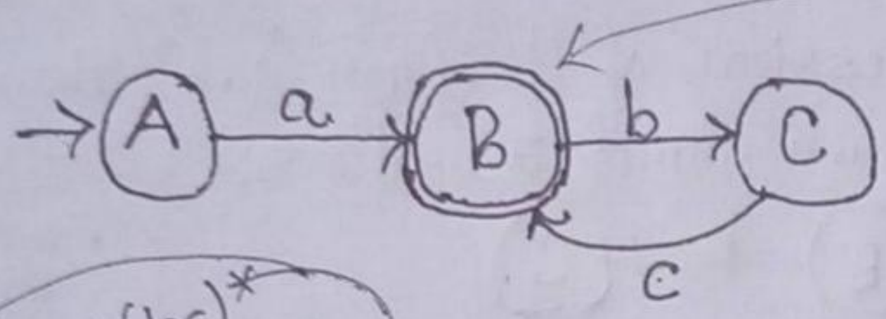
Solution:-



Since c is just and (i.e, dot operation) with (a+b). so directly sent to new state according to rule (ii)

iii) a(bc)*

Solution:-



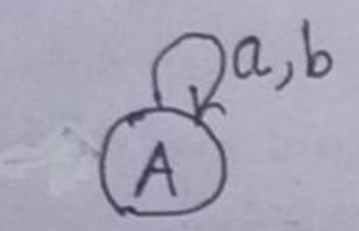
think (bc)* as form (a)* i.e, as one

Here a(bc)* means it contains strings like a, abc, abcbc, abcbcbc. B becomes final state because we may also get single a only as our string. Now if B is final state C is the final string on our question so, C on getting input c goes to final state B.

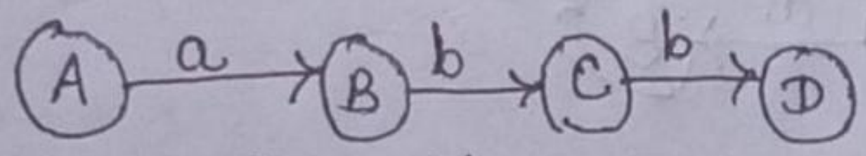
Example2: Convert the following Regular Expression to its equivalent Finite Automata:

(a|b)* (abb|a+b)

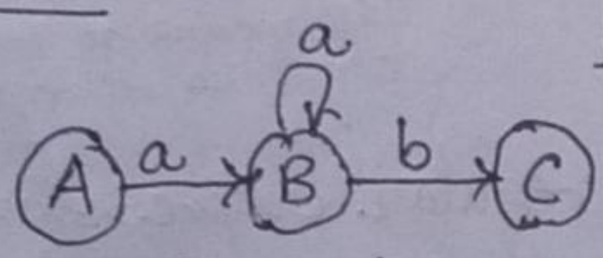
Solution:- For making easier to solve let we divide given expression in three parts as (a|b)*, abb & a+b and draw for each seperately.



for (a|b)*



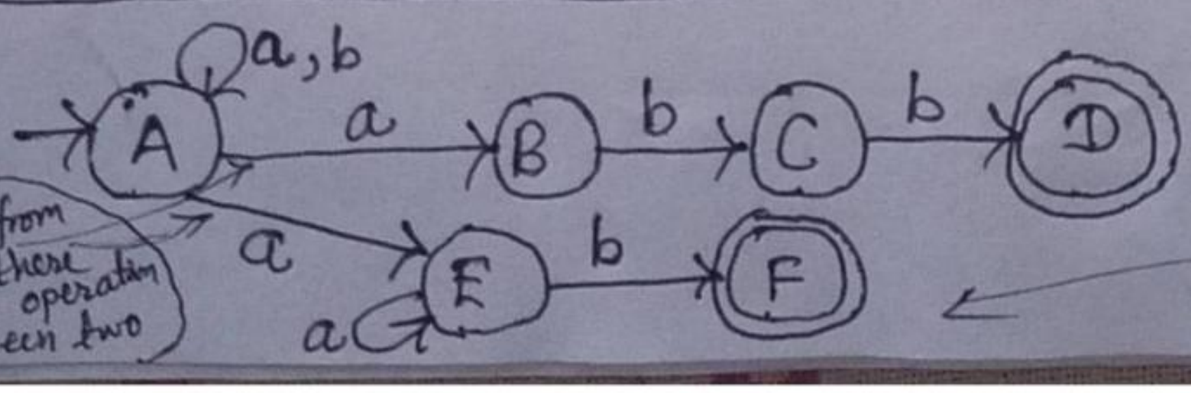
for abb



for a+b

for easier I did this we can directly draw if we know

Now we combine all of them in one as follows:-



we draw directly from A since there is and operation between two

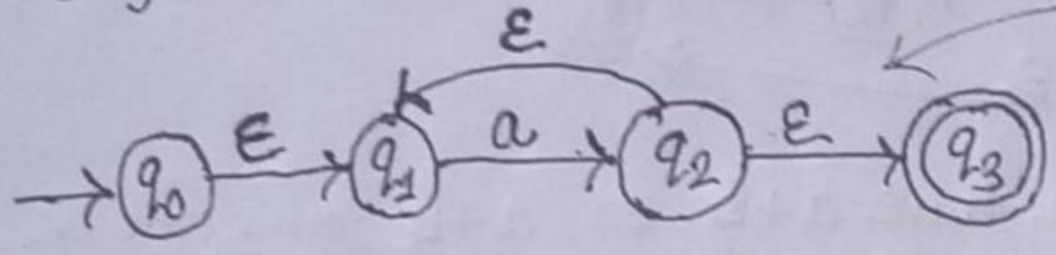
D and F are final states and came from A seperately because there is | i.e, OR operation between aab and a+b. So, on getting both we get final state

Scanned by TapScanner

⊛ Reduction of Regular Expression to ε-NFA:- [Imp]

For the conversion of regular expression to its equivalent ε-NFA we have some important basic rules as we used before (including now ε) as follows:-

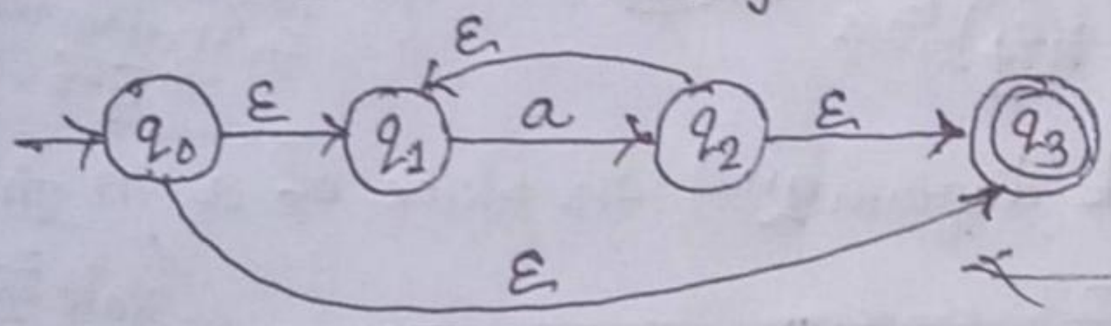
i) When we have expression of the form a^+ then its ε-NFA is as follows:-



यसमा अगाडि पछाडि ε input मा state परिवर्तन हुन् । q_2 बाट empty input मा फेरि q_1 मा जान मिल्ने भएकाले गर्दा any number of a we can get

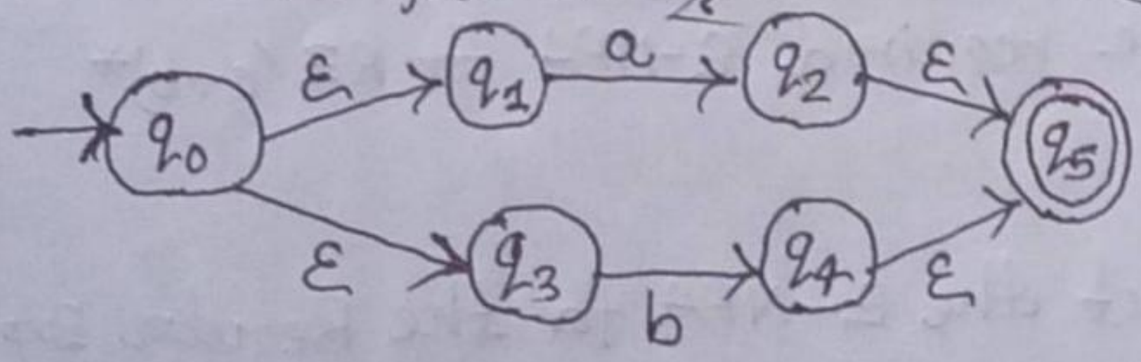
rule हरु हो यो यति याद गर्नै पर्ने ।

ii) When we have expression of the form a^* then its ε-NFA is as follows:-



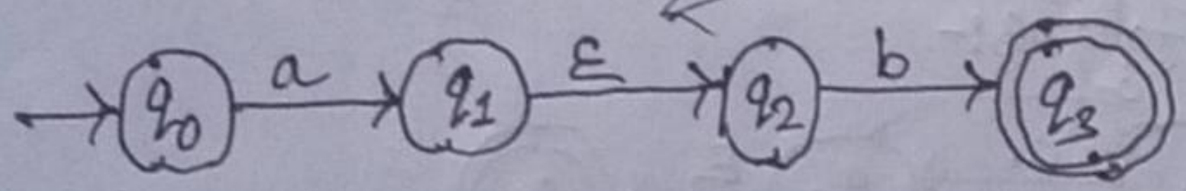
same diagram as for a^+. Only difference is this transition line from start state to final state on getting ε as input. This is because a^* contains any number of a's including zero (i.e, no input or ε).

iii) When we have expression of the form $(a+b)$ OR $(a \cup b)$ OR $(a|b)$ then its ε-NFA is as follows:-



a or b हुँदा त्यसैले हुने एक state बाट अर्को मा seperately र अगाडि र ~~पछाडि~~ ε states

iv) When we have expression of the form $(a.b)$ or simply (ab), then its ε-NFA is as follows:-



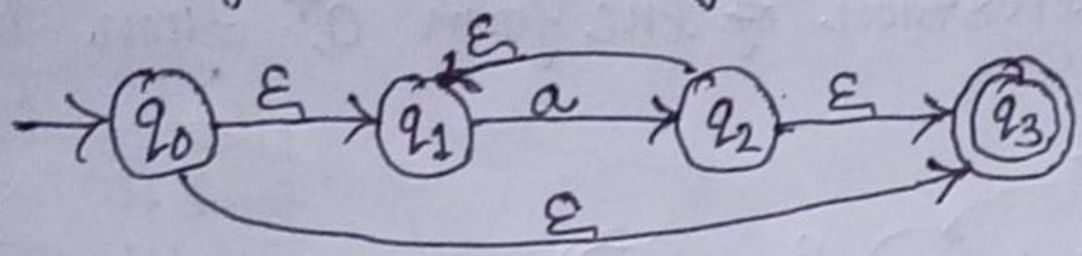
ab simply आको बेला without any operation then यसरि बिचमा ε ले seperate हुन्छ

Scanned by TapScanner

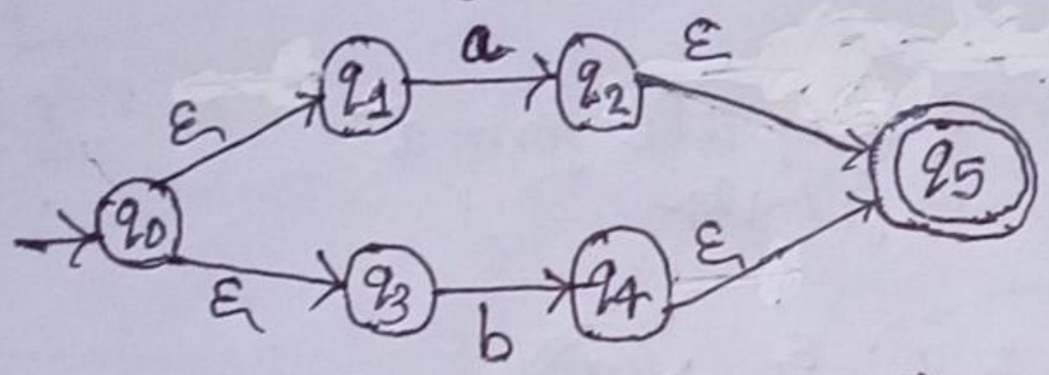
Example 1: Construct the ε-NFA for the Regular Expression $(a+b)^*$ where the alphabets are [a, b].

Solution:-

Let first we think $(a+b)^*$ as a^*, then according to the rule ε-NFA for a^* is as follows:-

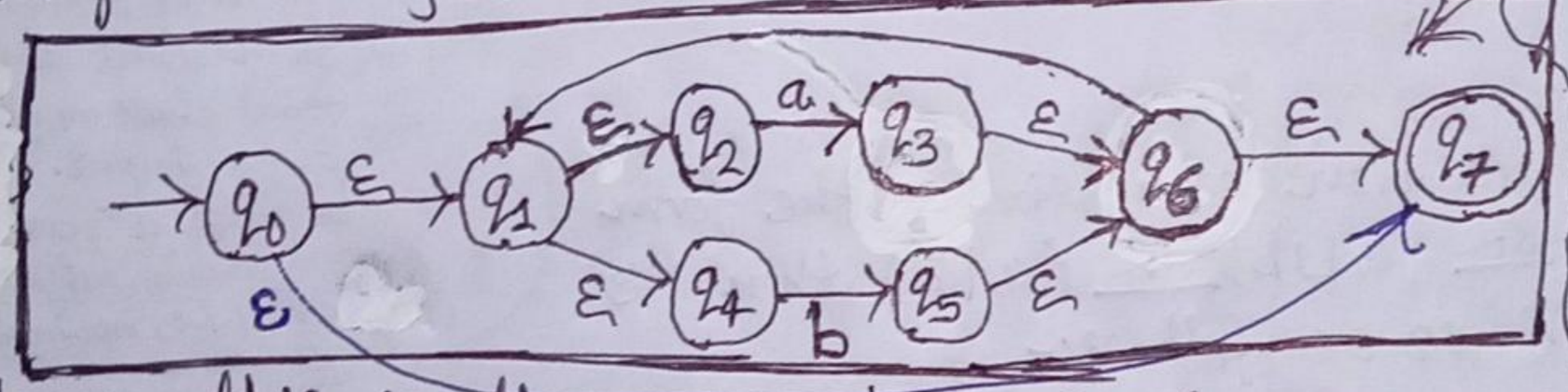


But instead of a we have a+b so, a+b form can be represented according to the rule in ε-NFA as follows:



बुझ्नको लागि यसरि stepwise गरेका। यो सब rough मा गरि direct final diagram बनाउदा पनि हुन्छ solution मा

Now, substituting a+b (diagram 2nd) in place of a in first diagram as follows:-



final diagram यो हो भनि जनाउन box मा

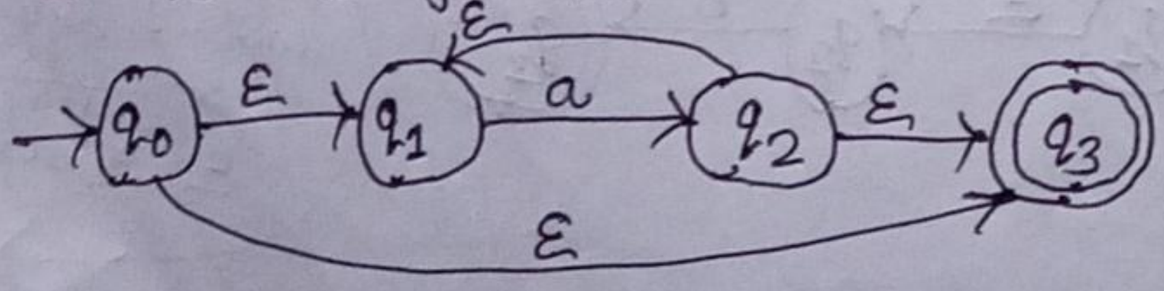
State को name हरु जे दिएनि हुन्छ unique to each but draw गरिपछि last मा नाम दिदा सजिलो हुन्छ

Hence, this is the required ε-NFA for RE $(a+b)^*$

Example 2:- Construct the ε-NFA for the Regular Expression $(00+11)^*$

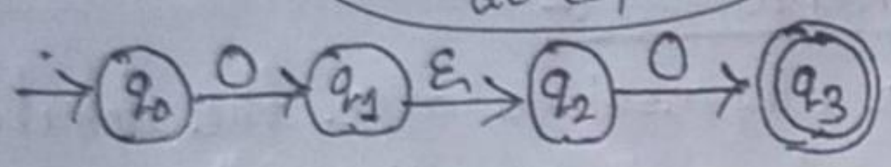
Solution:-

Let first we think $(00+11)^*$ whole as a^*, then ε-NFA for a^* is as follows:-

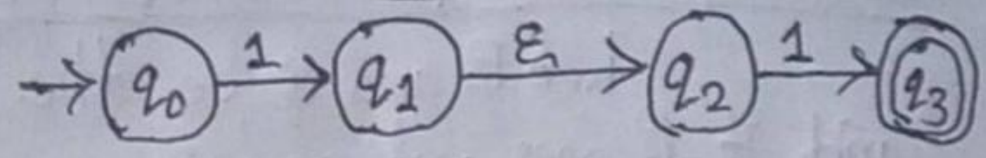


But instead of a we have 00+11. Again two inputs 00 or 11 can not remain at same place in diagram so we subdivide 00 and 11 and draw for each seperately as follows;

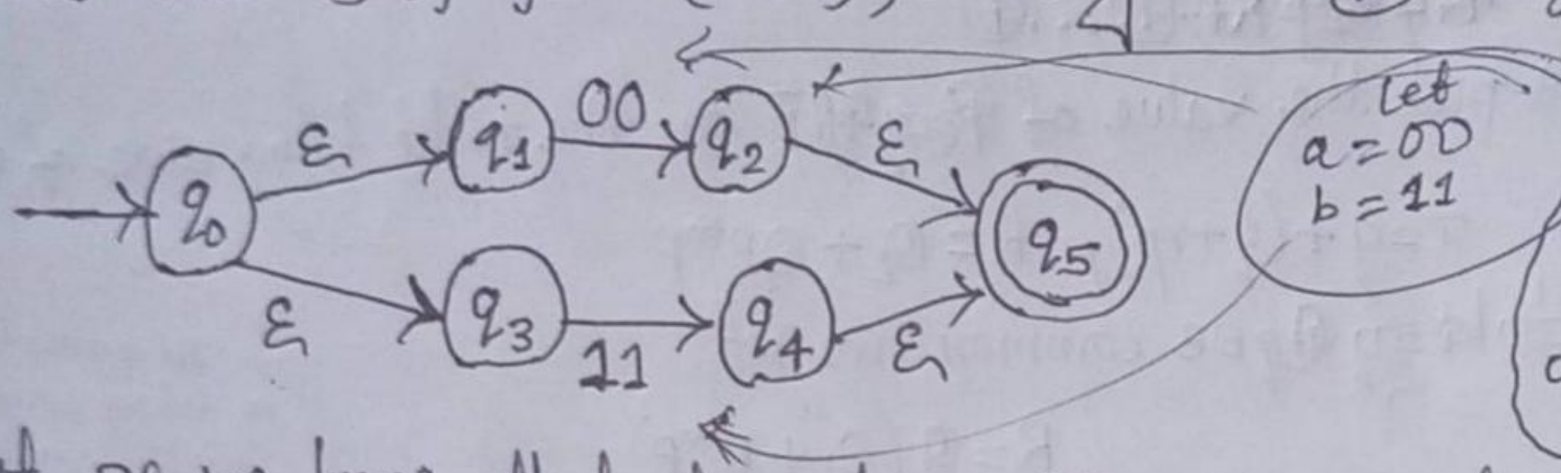
For 00 ← ab form so use rule (iv) to expand



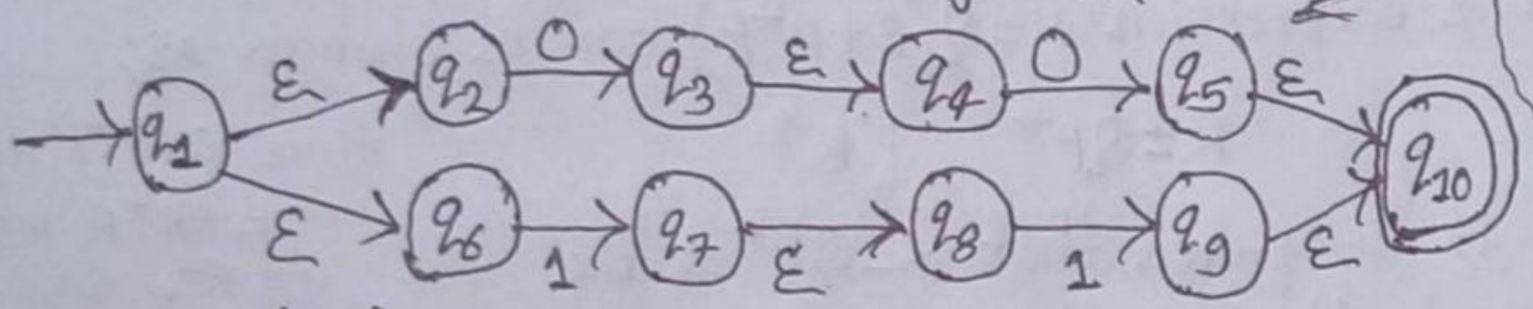
For 11 ← ab form rule (iv) used



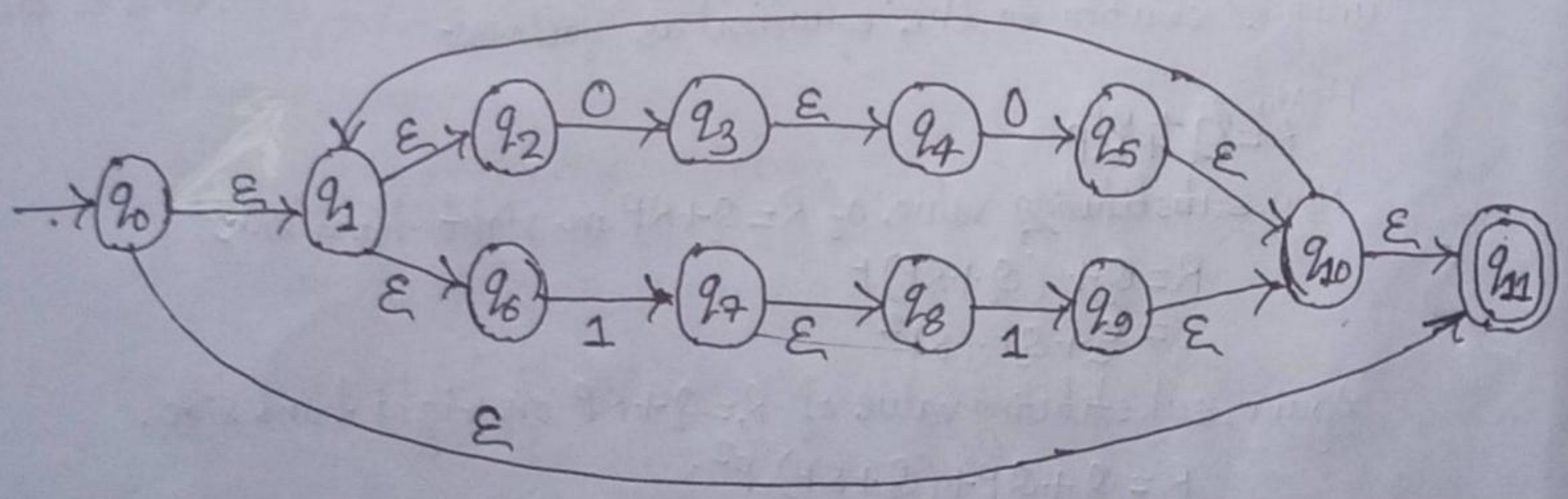
Now we got for 00 and 11, but we have 00+11 inplace of a so, we combine both. If we consider 00 as a and 11 as b then this becomes of form (a+b), so using rule (iii) we get,



But as we know that two strings 00 or 11 can not sit together as in above diagram so we replace them by the diagram we draw for 00 and for 11 as follows;-



Now we got diagram for (00+11) now we substitute it to our first figure in place of a, and hence we get required ε-NFA as follows;-



Note:- Practice more questions [i.e, kec book examples] yourself.

Scanned by TapScanner

⊛. Arden's Theorem:-

Let P and Q be the regular expressions over the alphabet Σ, and if P does not contain any empty string ε, then the following equation in R given by $R = Q + RP$ has a unique solution i.e. $R = QP^*$.

Proof:

(a) Here, $R = Q + RP$ ..... (i)

Let us put the value of $R = QP^*$ on the right hand side of relation (i)

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

Now, taking Q as common we get;

$$R = Q(\varepsilon + P^*P)$$

In general math on taking common if nothing remains we use to put 1 but here if nothing remains we put ε i.e, empty symbol

Now, from the algebraic laws or regular identities that we discussed earlier we have $\varepsilon + R^*R = R^*$.
The above expression $R = Q(\varepsilon + P^*P)$ can be changed as;

$$R = QP^*$$

Since $\varepsilon + P^*P$ is of form $\varepsilon + R^*R$ which equals to R^*. So, $\varepsilon + P^*P$ is replaced by P^*.

Hence it is proved that $R = QP^*$ is the solution to the equation $R = Q + RP$. This proves the first part of Arden's Theorem.

यो solution हो भनि माथि prove गरियो हो, अब unique solution हो भनि prove गरिन्छ।

(b) Now we prove the second part of Arden's theorem (i.e, this is the unique solution to the equation) as follows;-

Here,
$R = Q + RP$

Now, substituting value of $R = Q + RP$ on right hand side:

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

Again, substituting value of $R = Q + RP$ on right hand side.

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

Continuing in the same way, we will get as;

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

Since $\varepsilon + P + P^2 + P^3 + ...$ is the closure of P i.e, P^* So,

$$R = QP^*$$

Hence, now we proved that this is the unique solution for equation $R = Q + RP$.

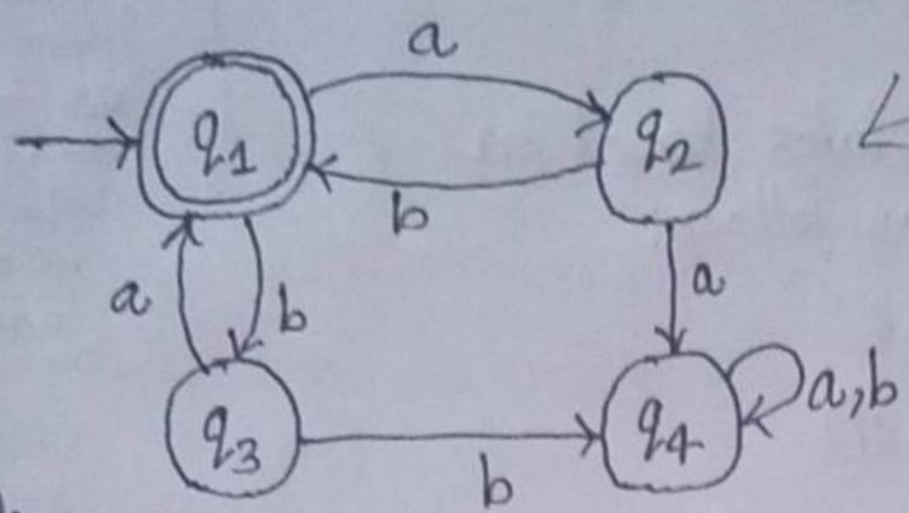
Scanned by TapScanner

(Page 20 continue)

⊛. Conversion of DFA to Regular Expression:-

For solving this we should have knowledge of Arden's theorem and algebraic laws for RE.

Example 1: Find the Regular Expression for the following DFA.



i.e, Example of single final state, Later we will discuss for more than one.

Solution:-

Here, equations for each states based on incomming transitions are as follows:-

$$q_1 = \varepsilon + q_2 b + q_3 a \quad \text{——} (i)$$

$$q_2 = q_1 a \quad \text{——} (ii)$$

$$q_3 = q_1 b \quad \text{——} (iii)$$

$$q_4 = q_2 a + q_3 b + q_4 a + q_4 b \quad \text{——} (iv)$$

start, incomming transition comming from nowhere so ε.

$q_2 b$ means q_1 has incomming transition which is q_2 on getting input b similarly for $q_3 a$

q_4 on getting a,b goes to q_4 so, $q_4 a + q_4 b$ done. seperately for a & b both.

Now we take eqn (i). (i.e, final state q_1)

$$q_1 = \varepsilon + q_2 b + q_3 a$$

Putting values of q_2 and q_3 from eqn (ii) and (iii)

$$q_1 = \varepsilon + q_1 ab + q_1 ba$$

Taking q_1 common:

$$q_1 = \varepsilon + q_1 (ab + ba)$$

$q_3 = \varepsilon + q_1 (ab+ba)$
$R = Q + RP$

Now this is of the form $R = Q + RP$ so, we can write it in the form $R = QP^*$ according to Arden's theorem.

i.e, $q_1 = \varepsilon \cdot (ab+ba)^*$

If we think ababa as one let R then it is of form $\varepsilon \cdot R$

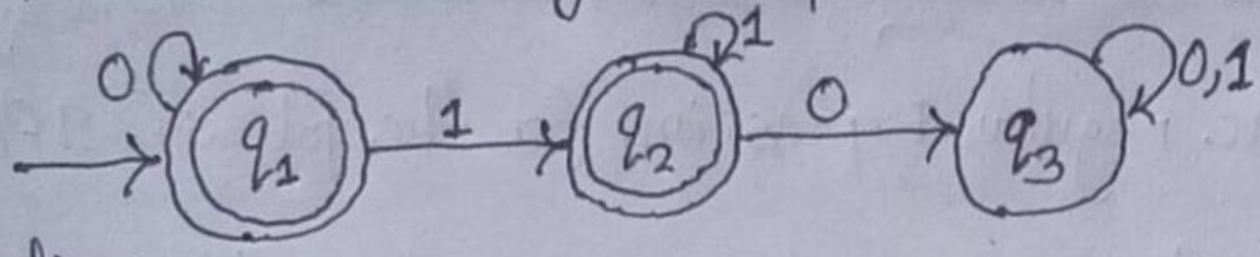
Now, according to the algebraic laws for regular expressions we know that $\varepsilon \cdot R = R$. So, above expression can be written as;

$$q_1 = (ab+ba)^*$$

Since q_1 is final state and we get expression for q_1. Hence $q_1 = (ab+ba)^*$ is the required Regular Expression for given DFA.

Scanned by TapScanner

Example2: Find the Regular Expression for the following DFA.



Example with more than one final state. For solving this we find R.E for all final states one by one as we did before then finally we do union (i.e, +) to get final RE

Solution:-

Here, equations for each states, based on incomming transitions are as follows;

$$q_1 = \varepsilon + q_1 0 \quad \text{—— (i)}$$

$$q_2 = q_1 1 + q_2 1 \quad \text{—— (ii)}$$

$$q_3 = q_2 0 + q_3 0 + q_3 1 \quad \text{—— (iii)}$$

Let we take final state q_1:

$$q_1 = \varepsilon + q_1 0$$

where, $q_1 = R$, $\varepsilon = Q$, $0 = P$

Now, this is of the form $R = Q + RP$, so using Arden's theorem we write it in the form $R = QP^*$ as follows:-

$$q_1 = \varepsilon . 0^*$$

$$q_1 = 0^* \quad \text{(iv)}$$ [Since, from algebraic laws for R.E we know that $\varepsilon . R = R$].

Again, let we take final state q_2:

$$q_2 = q_1 1 + q_2 1$$

$$q_2 = 0^* 1 + q_2 1$$ [Putting value of $q_1 = 0^*$ from eq" (iv)]

$q_2 = 0^*1 + q_2 1$, $R = Q + RP$

Now, if we think 0^*1 as one let Q, then this expression is of form $R = Q + RP$. So, using Arden's theorem we can write it in the form $R = QP^*$.

i.e, $q_2 = 0^* 1 (1)^*$

Now the Final Regular Expression can be obtained by the union of both final states:

i.e, $R.E = 0^* + 0^* 1 (1)^*$

$$R.E = 0^* (\varepsilon + 11^*)$$ [0^* Taken common]

According to the algebraic laws for R.E we have $\varepsilon + RR^* = R^*$.

So, $R.E = 0^* 1^*$ ← since $\varepsilon + 11^* = 1^*$

Hence, the required Regular Expression is $R = 0^*1^*$ for given DFA.

Scanned by TapScanner

⊛. Pumping Lemma for Regular Expression:-

Application of pumping lema is to prove that a language is not regular

Pumping lemma is used to prove that a language is not regular. It cannot be used to prove that a language is regular.

Statement: If A is a Regular language, then A has a Pumping length 'p' such that any string 'S' where $|S| \geq P$ may be divided into 3 parts $S = xyz$, such that the following conditions must be true:

i) $xy^iz \in A$ for every $i \geq 0$
ii) $|y| > 0$
iii) $|xy| \leq p$

i.e, on increasing y any number of times $i \geq 0$, then the string obtained must also belong to A.

i.e, length of y should be greater than 0;

i.e, length of x & y together should be less than or equal to pumping length.

To prove tha a language is not Regular using Pumping lemma follow the following steps: (We prove using Contradiction) :-

→ Assume that A is regular.
→ It has to have a Pumping Length (say P)
→ All strings longer than P can be pumped $|S| \geq P$.
→ Now find a string 'S' in A such that $|S| \geq P$.
→ Divide S into xyz.
→ Show that $xy^iz \notin A$ for some i.
→ Then consider all ways that S can be divided into xyz.
→ Show that none of these can satisfy all the 3 pumping conditions at the same time.
→ S cannot be Pumped == CONTRADICTION.

Example1:- Using Pumping lemma prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is Not Regular.

Proof:

Assume that A is regular.
Pumping length = P.

Scanned by TapScanner

Now, $S = a^p b^p$

Now, we need to divide S into three parts x, y, z, For that let we assume pumping length P = 7. Then we can write the string S as;

S = aaaaaaabbbbbbb.

Now, let us see all the possible ways in which we can divide this S into three parts x, y, z for that let we take cases as follows:-

तिनवटै condition मिले मात्रै satisfy हुन्छ यसटा condition पनि नमिले satisfy हुन्छ।

take x & y length in such a way that $|xy| \leq P$ Also, $|y| > 0$.

Case-I: The y is in the 'a' part.

i.e, aaaaaaabbbbbbb
x y z

Case-II: The y is in the 'b' part.

i.e, aaaaaaabbbbbbb
x y z

Case-III: The y is in the 'a' and 'b' part.

i.e, aaaaaaabbbbbbb
x y z

this means 2 times i.e, y string is repeated one time more

Now for each cases we check condition xy^iz. he we take i = 2.
i.e, xy^2z.

For Case-I:

aaaaaaaaabbbbbbb
x y y z
y^2

No. of a's = 11
No. of b's = 7
No. of a's ≠ No. of b's
So, $xy^2z \notin A$.

does not belong to A

For Case-II

aaaaaaabbbbbbbbb

No. of a's = 7
No. of b's = 11
No. of a's ≠ No. of b's.
So, $xy^2z \notin A$.

Case II with y part repeated similarly case-III

सबै case मा ∉A आ does not follow question pattern भए सबै regular हुन्छन्

i.e, पहिलो a हुने मात्रै हुनु पर्छ्यो त्यसपछि b हुने त्र mix हुन

For Case-II

aaaaaaabbaabbbbbb

Since this string does not follow the pattern $a^n b^n$ as given by question. So, this does not lie in our language.

Scanned by TapScanner

So, we proved that xy^iz on taking $i=2$ all cases does not lie in our language. ~~Hence the given language is not regular.~~ This means S cannot be pumped which leads to contradiction. Hence, the language is not regular.

Example 2:- Using Pumping Lemma prove that the language $A = \{yy \mid y \in \{0,1\}^*\}$ is not regular.

i.e, yy means first part of language must be same as 2nd part. where y has 0,1..

Proof: Assume that A is regular, then it must have a pumping length. Let Pumping length be 'P'.

We formed String S in A such that $|S| \geq P$. & first and 2nd part made same i.e, 0^P1.

Now,

$$S = 0^P 1 0^P 1$$

0 मा मात्रै power P राख्दा $|S| \geq P$ condition satisfy हुन्छ So, सजिलो को लागि 0 मा मात्रै राखेको x string बनाको $0^P1\,0^P1$ solve गर्दा सजिलो हुन्छ भनेर

Now, we need to divide S into three parts x, y, z, for that let we assume pumping length P=7. Then we can write the string S as;

1st part (y) 2nd part (y).

$$S = \overbrace{00000001}\,\overbrace{00000001}$$

Now let we see all the possible ways in which we can divide this S into three parts x, y, z for that we take cases as;

Case-I; The y is in the first part.

i.e, $\underbrace{0}_{x}\,\underbrace{0000}_{y}\,\underbrace{0100000001}_{z}$

Here, $|y| > 0$ & $|xy| \leq P$ i.e, $|xy| \leq 7$

Case-II: The y is in the second part.

i.e, $\underbrace{00000001}_{x}\,\underbrace{0000}_{y}\,\underbrace{01}_{z}$

Case-III: The y is in the first and second part.

i.e, $\underbrace{00000001}_{x}\,\underbrace{00}_{y}\,\underbrace{00000001}_{z}$.

Now for each cases we check condition xy^iz. Let we take $i=2$. i.e, xy^2z.

For Case-I $\underbrace{000000000001}_{\text{first part}}\,\underbrace{00000001}_{2^{nd}\text{ part.}}$

Scanned by TapScanner

Here, we see that first part is not equal to second part. i.e, it does not follow pattern YY. So, this does not lie in the language A. This means S cannot be pumped which leads to contradiction. Hence the language is not regular.

⊛. Closure Properties of Regular Languages:-

i) Closure under Union → If L and M are regular languages, so is L∪M. Let L and M be the languages of regular expressions R and S respectively. Then R+S is a regular expression whose language is L∪M.

ii) Closure under Intersection → If L and M are regular languages, so is L∩M. Let L and M be the languages of regular expressions R and S respectively then R∩S is a regular expression whose language is L∩M.

iii) Closure under Concatenation → If L and M are regular languages, so is L.M. If L and M be the languages of regular expressions R and S respectively then R.S is a regular expression whose language is L.M.

iv) Closure under Kleene closure → If L is the regular language of regular expression R then R^* is a regular expression whose language is L^*.

v) Closure under complement → The complement of a language L (with respect to an alphabet Σ such that Σ^* contains L) is $\Sigma^* - L$. Since Σ^* is surely regular by the property of closure under Kleen closure, the complement of a regular language is always regular.

Scanned by TapScanner

⊛. Minimization of DFA: (Table Filling Algorithm) :-

also known as Myhill-Nerode Theorem

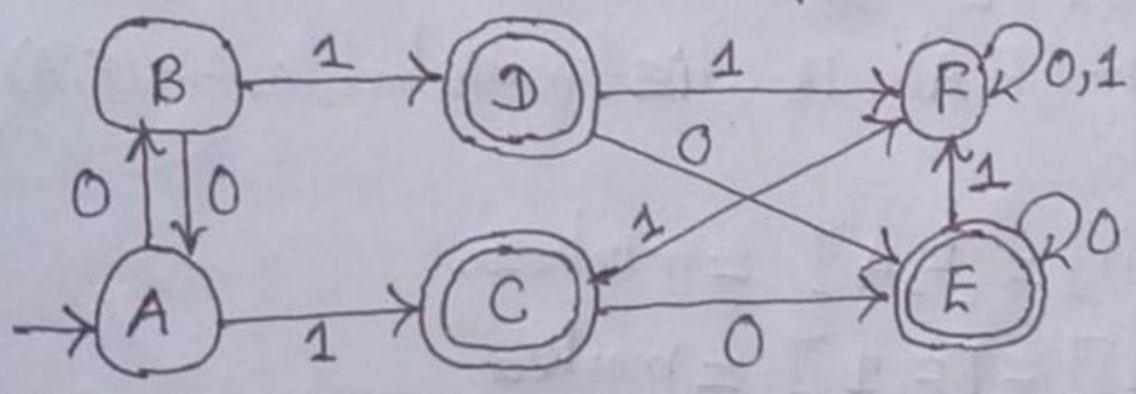
Steps:-

1) Draw a table for all pairs of states (P, Q).
2) Mark all pairs where $P \in F$ and $Q \notin F$.
3) If there are any Unmarked pairs (P, Q) such that $[\delta(P,x), \delta(Q,x)]$ is marked, then mark [P, Q] and repeat this process until no markings can be done.
4) Combine all the Unmarked Pairs and make them a single state in the minimized DFA.

Pair मा कुनै एक final state को set मा पर्‍यो र अर्को पर्दैन भने Mark गर्ने। दुबै state पर्ने वा पर्दैननु भने नगर्ने

i.e, Check a pair where it goes which is obtained by state P & Q on getting particular input x.

Example :- Minimize the following DFA using Table Fill Algorithm.



Solution:-

	A	B	C	D	E	F
A						
B						
C	✓	✓				
D	✓	✓				
E	✓	✓				
F	✓	✓	✓	✓	✓	

Full table मध्येको माथि diagonally divide गरि upper part राख्ने repeat गरेको same pair जस्ते AA, FF, (यसैगरि AB र BA यउटै हो so, एउटा pair BA check गर्दा पुग्यो so, repeating rooms eliminated & pair checked and ticked.

FA and FB are empty not marked initially. they are later marked after checking pairs which unmarked.

Now, we check the Unmarked pairs as follows:-

(F,E) pair मा
F → final state हो जस्न
E → final state हो
यसरि फरक भए mark हुन्छ यउटै भए हुन्न

For pair (B, A)

input 0 taken for pair

$[\delta(B,0), \delta(A,0)] = [A,B] \Rightarrow$ unmarked

[A, B] table मा mark छैन त्यसैले unmarked राखेको। marked भको भए marked राखिन्थ्यो

input 1 taken for pair

$[\delta(B,1), \delta(A,1)] = [D,C] \Rightarrow$ unmarked

DC unmarked in table so, unmarked

Scanned by TapScanner

For pair DC

$[\delta(D,0), \delta(C,0)] = [E,E]$ = unmarked ← table मा सबभन्दा पनि unmarked

$[\delta(D,1), \delta(C,1)] = [F,F]$ = unmarked

For pair EC

$[\delta(E,0), \delta(C,0)] = [E,E]$ = unmarked

$[\delta(E,1), \delta(C,1)] = [F,F]$ = unmarked

For pair ED

$[\delta(E,0), \delta(D,0)] = [E,E]$ = unmarked

$[\delta(E,1), \delta(D,1)] = [F,F]$ = unmarked

For pair FA

$[\delta(F,0), \delta(A,0)] = [F,B]$ = unmarked.

$[\delta(F,1), \delta(A,1)] = [F,C]$ = marked

So, the pair (F,A) is distinguishable. (i.e, (F,A) is marked now)

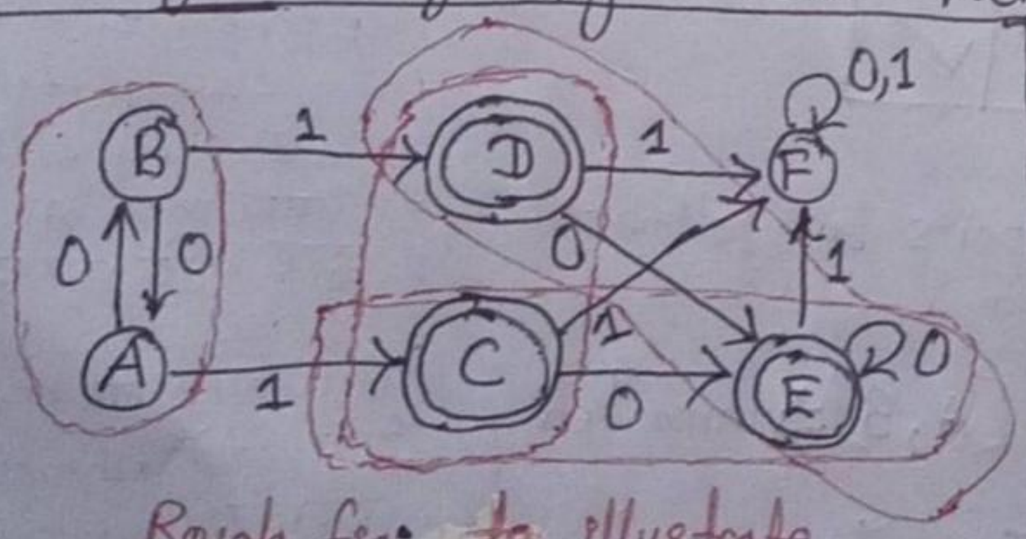
For pair FB

$[\delta(F,0), \delta(B,0)] = [F,A]$ = marked.

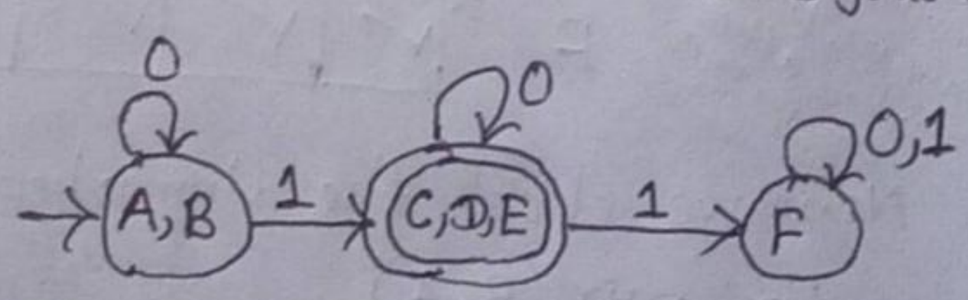
$[\delta(F,1), \delta(B,1)] = [F,D]$ = marked

So, the pair (F,B) is distinguishable. (i.e, (F,B) is marked now)

Now, finally we combine unmarked pairs (B,A), (D,C), (E,C), (E,D) as one single state. i.e, (B,A) is now single state. But in (D,C), (E,C), (E,D) there is common C in (D,C), (E,C) and also common E in (E,C), (E,D) this shows all these 3 pairs can also be combined as one single state, which is illustrated by a rough figure below. Hence, the minimized DFA is as follows:-



Rough fig:- to illustrate (D,C), (E,C), (E,D) as one state since on combining them each of them are overlapping (i.e, common).



यो rough fig. rough मा बनाउने क्रम मा जरुरी छैन / कुनै सजिलो लाई मात्र

Scanned by TapScanner