

Unit-3Regular 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.

④ 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 (\cup), concatenation (\cdot), 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.

→ for which the second symbol is 0

→ and that end with zero or more number of 1's.

Indicated by $(0 \cup 1)$
 \cup means OR operation

Indicated by 1^*

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

Formal Definition of Regular Expression:

- Let Σ be an alphabet, the regular expression over the alphabet Σ are defined inductively as follows;
- \emptyset is a regular expression representing empty language.
 - E 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:
 - $r \cup s$ is a regular expression denoting the language $L(r) \cup L(s)$.
 - $r.s$ 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 ($\cup / |$): 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

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

$$L_1 \cdot L_2 = \{l_1 l_2 \mid l_1 \in L_1 \text{ and } l_2 \in L_2\}$$

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

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

but this symbol used here denotes empty string

Concatenation denotes and operation

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

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

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

* has highest precedence
• has next higher precedence
 $\cup / | / +$ has lowest precedence.

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

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

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

② Applications of Regular Expression:

- Validation → Determining that a string complies with a set of formatting constraints, like email address validation, password validation etc.
- Search and Selection → Identifying a subset of items from a larger set on the basis of a pattern match.
- Tokenization → Converting a sequence of characters into words, tokens (like keywords, identifiers) for later interpretation.
- Lexer → Used as a lexer/tokenizer in lexical analysis step of compilers.

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

③ Algebraic laws for regular expression:

1) Commutativity:

$$r+s = s+r \text{ i.e., } r \cup s = s \cup r \\ \text{but } r.s \neq s.r$$

2) Associativity

$$r+(s+t) = (r+s)+t \\ \text{Also } r.(s.t) = (r.s).t$$

3) Distributive law:

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

$$\text{Also, } (m+n)l = ml + nl$$

4) Identity law:

$$r+\emptyset = \emptyset + r = r \text{ i.e., } \emptyset \cup r = r,$$

$$\text{Also, } \epsilon \cdot r = r = r \cdot \epsilon$$

empty symbol

5) Annihilator:

$$\emptyset \cdot r = r \cdot \emptyset = \emptyset$$

6) Idempotent law of union:

$$r+r=r$$

7) Law of closure:

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

$$\text{Also, closure of } \emptyset = \emptyset^* = \epsilon$$

$$\text{Also, closure of } \epsilon = \epsilon^* = \epsilon$$

$$\text{Also, Positive closure of } r, r^* = rr^*.$$

★ Regular Expression Examples:

1. Describe the following sets as Regular Expressions.

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

e) $\{\lambda, 11, 111, 1111, \dots\}$

Solution :

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

$$\Rightarrow R = 0 + 1 + 2$$

Since this set can contain 0 or 1 or 2 not other than that
+ means addition operation.

b) $\{\lambda, ab\}$

empty symbol λ and ε are same any can be used

$$\Rightarrow R = \lambda \cdot ab \cdot$$

during empty symbol we write using · operator

c) $\{abb, a, b, bba\}$

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

Since this set contains any of abb or a or b or bba not other than that so, + operator.

d) $\{\lambda, 0, 00, 000, \dots\}$

$$\Rightarrow R = 0^*$$

i.e., all the string that can be formed using 0 along with empty symbol i.e., all means closure * sign.

e) $\{\lambda, 11, 111, 1111, \dots\}$

$$\Rightarrow R = 1^+$$

i.e., all string that can be formed by 1. But it's not closure of 1. Since closure should contain empty symbol λ or ε as well.

1^+ denotes closure of 1 excluding empty symbol.

2. Consider $\Sigma = \{0, 1\}$, some regular expressions over Σ :

a) $0^* 1 0^*$

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 coming.

b) $\Sigma^* 1 \Sigma^*$

This means it contains at least one 1.

Since Σ^* means all string that can be formed using $\Sigma = \{0, 1\}$ of any length.

c) $\Sigma^* 001 \Sigma^*$

this means contains string 001 as substring.

d) $(\Sigma \Sigma)^*$ or $((0+1)^* \cdot (0+1)^*)$

this means string of even length.

e) $1^* (01^* 01^*)^*$

this means string containing even number of zeros

f) $0^* 1 0^* 1 0 8 1 0 8$

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

understand only

* 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:-

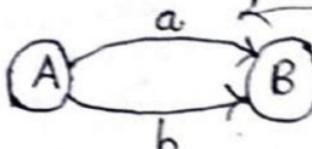
i) When we have expressions like $(a+b)$

OR $(a \cup b)$

OR $(a \mid b)$

since $+$, \cup operation
denote same thing
i.e., or operation

then, we draw it as,



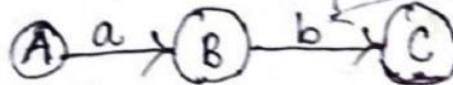
since $a+b$, $a \cup b$, $a \mid b$ means OR operation
so on getting any of them can go to next state

OR



same thing
drawn with
single transition
line

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



• means AND operation
so separate states
are made

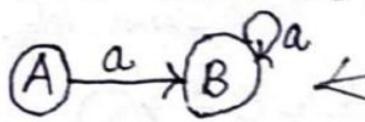
iii) If we have the expression of the form a^* then we create a transition that goes to the state itself without creating next state as follows:-



Since a^* denotes zero
to any number of a 's.
i.e., $a^* = \{\epsilon, a, aa, aaa, \dots\}$

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., $a^+ = \{a, aa, aaa, \dots\}$.
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!

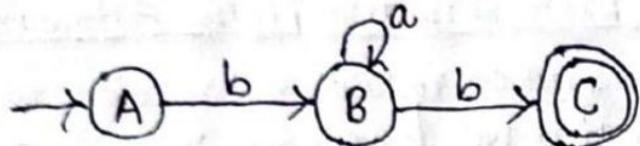
i) ba^*b

ii) $(a+b)c$

iii) $a(bc)^*$

i) $b a^* b$

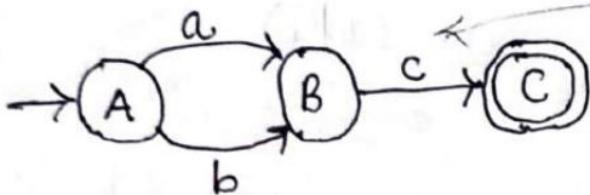
Solution:



Since for first a there is no any operation so we directly go to next state (according to rule 11). Then we have a^* in which we applied rule 10 and finally for b again rule 11 used. b input is final so C is final state.

ii) $(a+b)c$

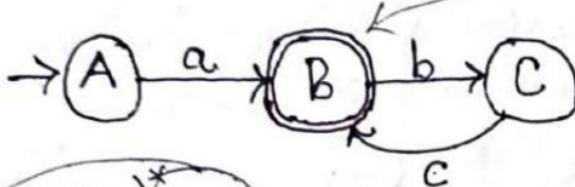
Solution:



Since C is just and (i.e., dot operation) with $(a+b)$. so directly sent to new state according to rule 11

iii) $a(bc)^*$

Solution:



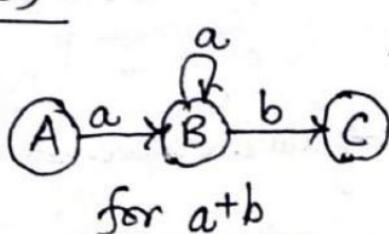
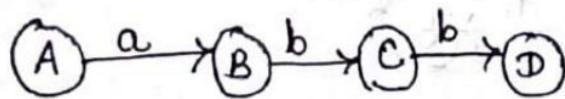
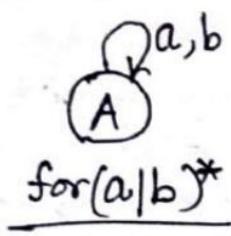
think $(bc)^*$
as form $(ab)^*$
i.e., as one

Here $a(bc)^*$ means it contains strings like a, abc, abcbc, abc bc bc. 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, on getting input c goes to final state B.

Example 2: Convert the following Regular Expression to its equivalent Finite Automata:

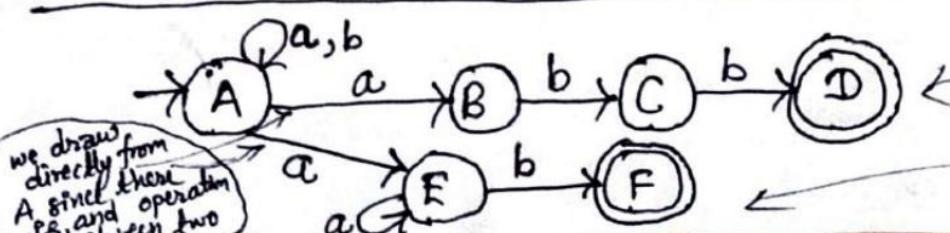
$(a|b)^* (abb|a^+b)$

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



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

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



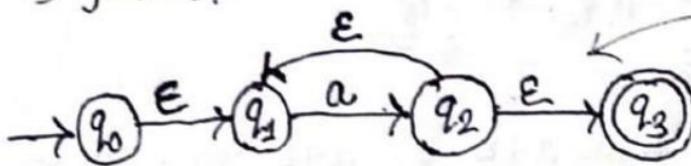
we draw directly from
A since there
is operation
between two

D and F are final states and came from A separately because there is OR operation between abb and a^+b . so, on getting both we get final state

Q. Reduction of Regular Expression to E-NFA :- [Imp]

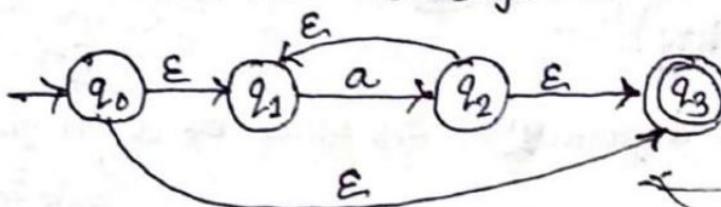
For the conversion of regular expression to its equivalent E-NFA we have some important basic rules as we used before (including now E) as follows:-

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



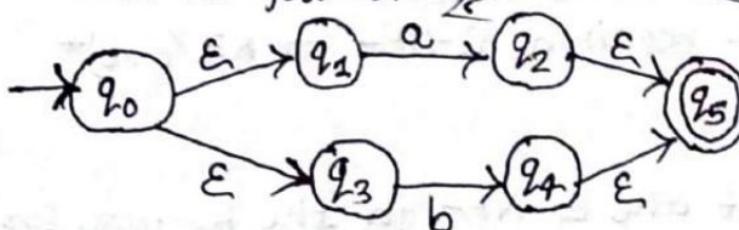
प्राप्ति अंगारे प्राप्ति E
input मा state परिवर्तन
करने । q_2 का empty input
मा पर्ने q_1 मा जाने को
प्राप्ति जानी any number
of a we can get

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



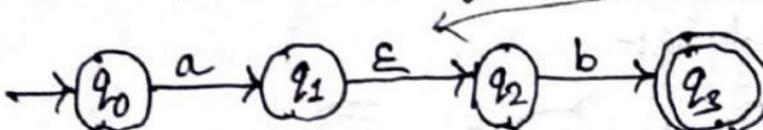
rule हमें दो यो
प्राप्ति ग्राह जानी पर्दी,

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



a or b E प्राप्ति
एवं एक state को
अकेले मा separately
प्राप्ति र उपरांत 2
एकाकी E states

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

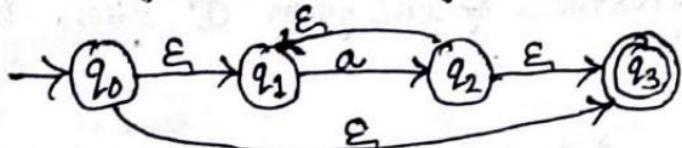


ab simply जानी बोला
without any operation
then चर्चित बिटामा
E के separate terms

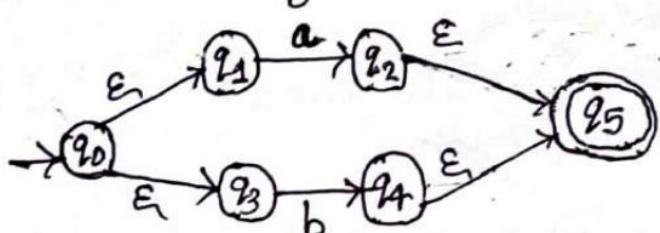
Example 1: Construct the ϵ -NFA for the Regular Expression $(a+b)^*$ where the alphabets are $[a,b]$.

Solution:-

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

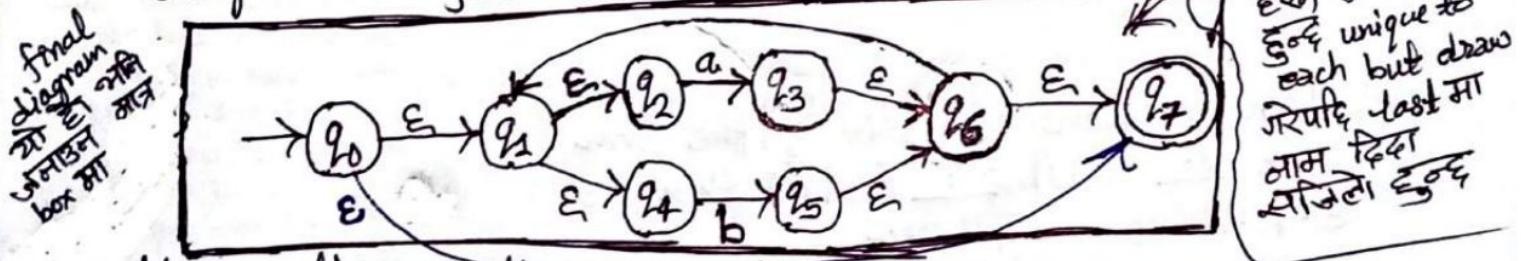


But instead of a we have $a+b$ so, $a+b$ form can be represented according to the rule in ϵ -NFA as follows:



कुल तरीके ताकि गम्भीर
step wise तरीका 1 तो सब
rough में तरीके direct
final diagram तैयार
तो कूप से solution में

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

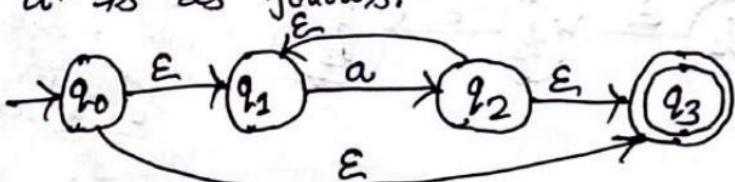


Hence, this is the required ϵ -NFA for RE $(a+b)^*$

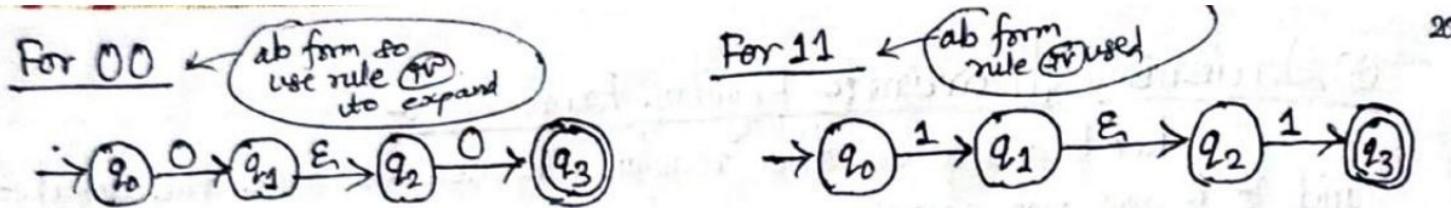
Example 2:- Construct the ϵ -NFA for the Regular Expression $(00+11)^*$

Solution:-

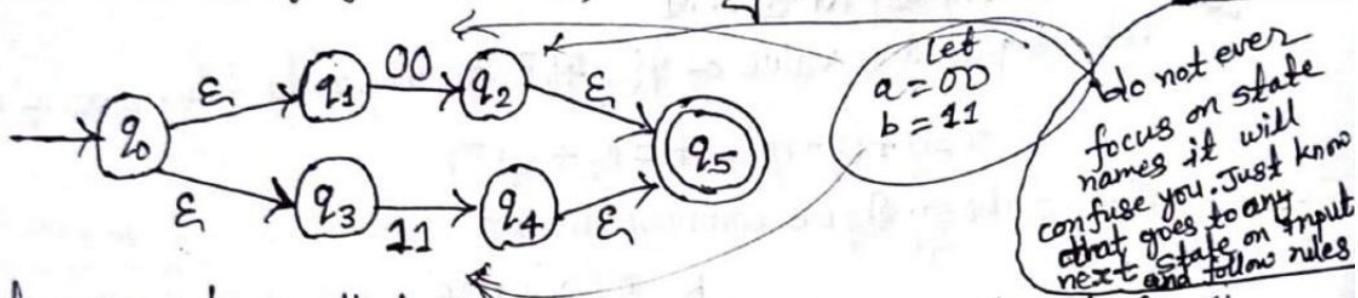
Let first we think $(00+11)^*$ whole as a^* , then ϵ -NFA for a^* is as follows:-



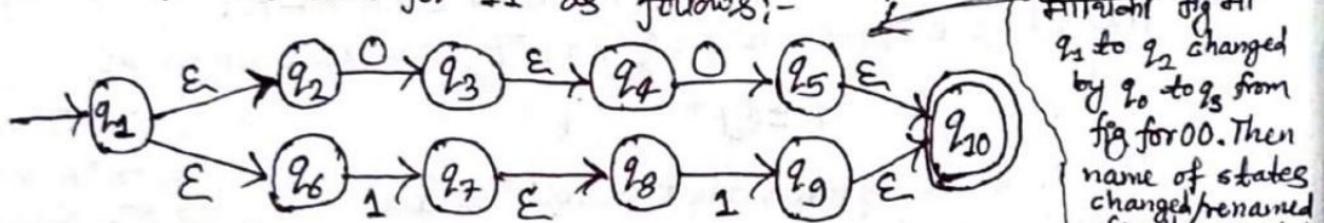
But instead of a we have $00+11$. Again two inputs 00 or 11 can not remain at same place in diagram so we subdivide 00 and 11 and draw for each separately as follows;



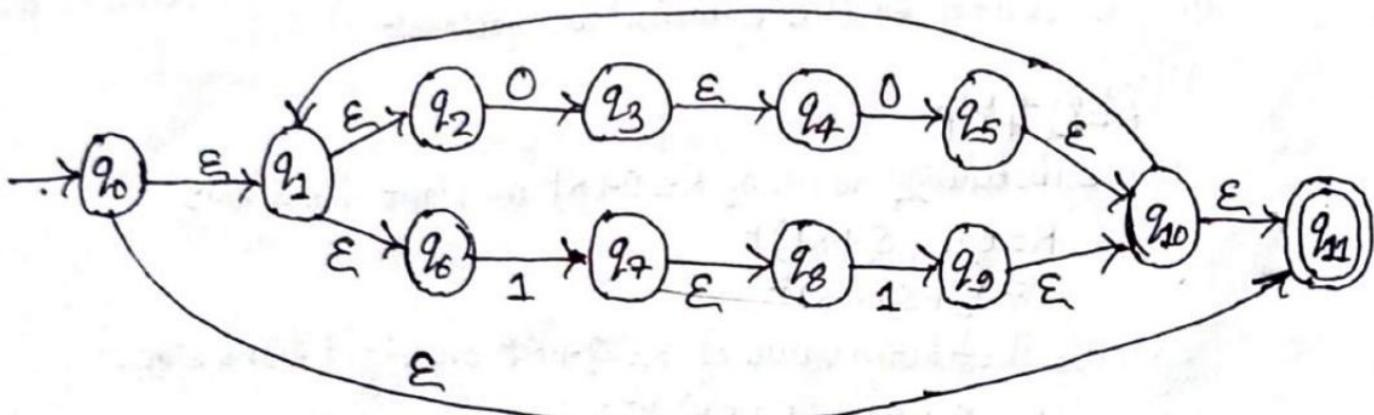
Now we got for 00 and 11, but we have 00+11 instead of a so, we combine both. If we consider 00 as a and 11 as b then this becomes of form (a+b), so using rule (iii) we get,



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 q_f 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.

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, $R = Q + RP \dots \text{①}$

Let us put the value of $R = QP^*$ on the right hand side of relation (1)

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

Now, taking Q as common we get;

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

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 $\epsilon + R^*R = R^*$.

The above expression $R = Q(\epsilon + P^*P)$ can be changed as;

$$R = QP^* \leftarrow$$

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.

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

दो भाग से प्रमाण
प्रमाण करना है, अब
एक समान उत्तर का प्रमाण
करना है।

(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:

$$\begin{aligned} R &= Q + (Q + RP)P \\ &= Q + QP + RP^2 \end{aligned}$$

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

$$\begin{aligned} R &= Q + QP + (Q + RP)P^2 \\ &= Q + QP + QP^2 + RP^3 \end{aligned}$$

Continuing in the same way, we will get as;

$$\begin{aligned} R &= Q + QP + QP^2 + RP^3 + \dots \\ &= Q(\epsilon + P + P^2 + P^3 + \dots) \end{aligned}$$

Since $\epsilon + P + P^2 + P^3 + \dots$ is the closure of P i.e., P^* so,

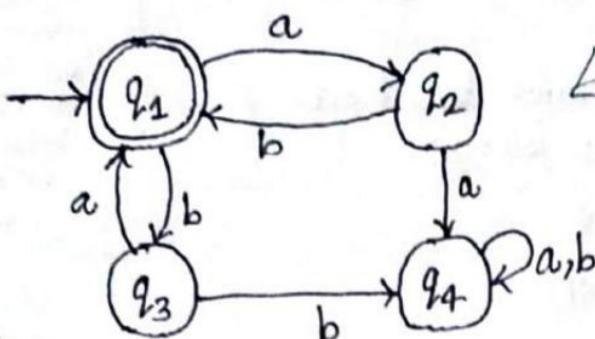
$$R = QP^*$$

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

Q. 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 incoming transitions are as follows:-

$$q_1 = \epsilon + q_2 b + q_3 a \quad \text{--- (1)}$$

$$q_2 = q_1 a \quad \text{--- (2)}$$

$$q_3 = q_1 b \quad \text{--- (3)}$$

$$q_4 = q_2 a + q_3 b + q_4 a + q_4 b \quad \text{--- (4)}$$

Now we take eqn (1). (i.e., final state q_1)

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

Putting values of q_2 and q_3 from eqn (2) and (3)

$$q_1 = \epsilon + q_1 ab + q_1 ba$$

Taking q_1 common:

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

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

$$\text{i.e., } q_1 = \epsilon \cdot (ab + ba)^*$$

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

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

start - incoming
transition coming
from nowhere so ϵ .

$q_2 b$ means q_1 has
incoming transition
which goes to 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.
separately for a & b both.

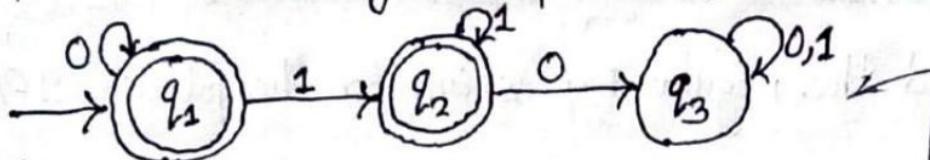
$$q_3 = \epsilon + q_1(ab + ba)$$

$$R = Q + RP$$

If we think $abba$ as
one let R then it is
of form $\epsilon \cdot R$

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.

Example 2: Find the Regular Expression for the following DFA.



Solution:-

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

$$q_1 = \epsilon + q_1 0 \quad \text{--- (1)}$$

$$q_2 = q_1^1 + q_2^1 \quad \text{--- (2)}$$

$$q_3 = q_2 0 + q_3 0 + q_3 1 \quad \text{--- (3)}$$

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 R.E

Let we take final state q_1 :

$$q_1 = \epsilon + q_1 0$$

where,
 $q_1 = R$
 $\epsilon = Q_1$
 $0 = P$

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

$$q_1 = \epsilon \cdot 0^*$$

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

Again, let we take final state q_2 :

$$q_2 = q_1^1 + q_2^1$$

$$q_2 = 0^* 1 + q_2^1 \quad \text{[Putting value of } q_1 = 0^* \text{ from eq" (1)]}$$

Now, if we think $0^* 1$ as one let Q_2 , then this expression is of form $R = Q + RP$. So, using Arden's theorem we can write it in the form $R = QP^*$.

$$\text{i.e., } q_2 = 0^* 1 (1)^*$$

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

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

$$\text{i.e., } R.E = 0^* + 0^* 1 (1)^*$$

$$R.E = 0^* (\epsilon + 11^*) \quad \text{[} 0^* \text{ Taken common]}$$

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

So,

$$R.E = 0^* 1^* \quad \text{since } \epsilon + 11^* = 1^*$$

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

* Pumping lemma for Regular Expression:-

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:

$$\rightarrow xy^i z \in A \text{ for every } i \geq 0$$

$$\text{if } |y| > 0$$

$$\text{if } |xy| \leq P$$

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 $x + y$ together should be less than or equal to pumping length.

To prove that 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 $xy^i z \notin A$ 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.

Example 1:- 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.

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

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

$$S = aaaaaaaaabbbbbbbb.$$

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

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

i.e., $\underbrace{aaaaaaaa}_{x} \underbrace{a}_{y} \underbrace{bbbbbbb}_{z}$

condition satisfied
परन्तु सत्य है कि यह
condition परन्तु परन्तु सत्य है कि

take $x+y$ length in
such a way that $|xy| \leq p$
Also, $|y| \geq 0$.

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

i.e., $\underbrace{aaaaaaaa}_{x} \underbrace{abbbbbb}_{y} \underbrace{bbb}_{z}$

This means 2
times its string
is repeated
one time more

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

i.e., $\underbrace{aaaaaa}_{x} \underbrace{aabb}_{y} \underbrace{bbbbb}_{z}$

Now for each cases we check condition xy^2z . Here we take $p=2$.

For Case-I:

$$\underbrace{aaaaaaa}_{x} \underbrace{aaaaa}_{y} \underbrace{abb}_{z}$$

y^2

No. of a's = 11

No. of b's = 7

No. of a's \neq No. of b's

So, $xy^2z \notin A$.

does not belong to A

For Case-II

$$aaaaaaaabb bbbbbbbbbb$$

No. of a's = 7

No. of b's = 11

No. of a's \neq No. of b's

So, $xy^2z \notin A$.

case II with
b part repeated
Similarly case-III

For Case-III

$$aaaaa aabb aabb bbbbb$$

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

i.e., परन्तु a एवं b में
इन परमाणुसमान
बहुलता मिल

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

$$A = \{yy \mid y \in \{0,1\}^*\}$$

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 '.

Now,

$$S = 0^p 1 0^p 1$$

We formed string S in A such that $|S| \geq p$. & first and 2nd part made same i.e., $0^p 1$.

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

$$S = \underbrace{0000000}_1 \underbrace{0000000}_1$$

O मात्रे power p रखते
 $|S| \geq p$ condition satisfy है।
 So, सभिलो के लिए O मा
 टनको राखेको 2 string
 बनाको $0^p 1 0^p 1$ solve हो।
 सभिलो ही सु भगेर

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.

$$\text{i.e., } \underbrace{000000}_x \underbrace{01}_{y} \underbrace{00000001}_z$$

Here $|y| > 0$
 $\& |xy| \leq p$
 $\therefore |xy| \leq 7$

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

$$\text{i.e., } \underbrace{00000001}_{x} \underbrace{00000001}_{y} \underbrace{1}_{z}$$

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

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

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

For Case-I

$$\underbrace{000000000001}_{\text{first part}} \underbrace{0000000001}_{\text{2nd part}}$$

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.

② Closure Properties of Regular Languages:-

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

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

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 Kleene closure, the complement of a regular language is always regular.

* Minimization of DFA: (Table Filling Algorithm) :-

also known as
Myhill-Nerode theorem 23.

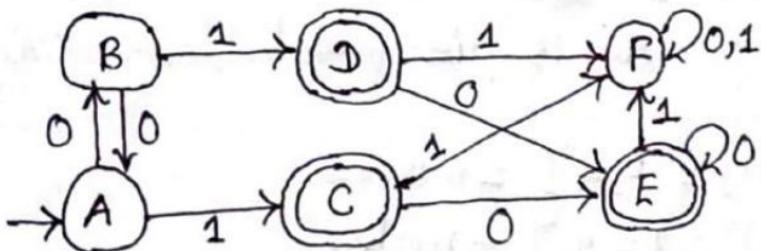
Steps:-

- 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 $[S(P, x), S(Q, x)]$ 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 कौने सके Final state को set मा पढ़ र अकों पढ़ने ले Mark राने। दुबे state पर्हन या पढ़नने जोन नहीं होती।

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	✓	✓	✓	✓	✓	✓

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

Full table में गोले उपर
diagonally divide जरुर upper
part र repeat जरुर same
pair जैसे AA, FF, (मैं गोले)
AB र BA गोले हो सो,
अउरा pair BA check जाये
पुराएं सो, repeating
rooms eliminated
& pair checked
and ticked.

Now, we check the Unmarked pairs as follows:-

For pair (B, A)

Input 0 taken
for pair

$$[S(B, 0), S(A, 0)] = [A, B] \rightarrow \text{unmarked}$$

input
1 taken
for pair

$$[S(B, 1), S(A, 1)] = [D, C] \rightarrow \text{unmarked}$$

(F, E) pair मा
F → final state होता है।
E → final state होता है।
असरि करके भर mark
होने वाले भर होता है।

$[A, B]$ table मा mark
होने वाले unmarked
राखेको। marked भाको
भर mark राखिन्दो।

DC unmarked in table
so, unmarked

For pair DC

$$[S(D,0), S(C,0)] = [E, E] = \text{unmarked}$$

$$[S(D,1), S(C,1)] = [F, F] = \text{unmarked}$$

table में अमार्क लिए
unmarked

For pair EC

$$[S(E,0), S(C,0)] = [E, E] = \text{unmarked}$$

$$[S(E,1), S(C,1)] = [F, F] = \text{unmarked}$$

For pair ED

$$[S(E,0), S(D,0)] = [E, E] = \text{unmarked}$$

$$[S(E,1), S(D,1)] = [F, F] = \text{unmarked}$$

For pair FA

$$[S(F,0), S(A,0)] = [F, B] = \text{unmarked}$$

$$[S(F,1), S(A,1)] = [F, C] = \text{marked}$$

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

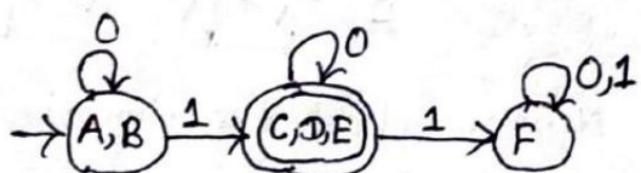
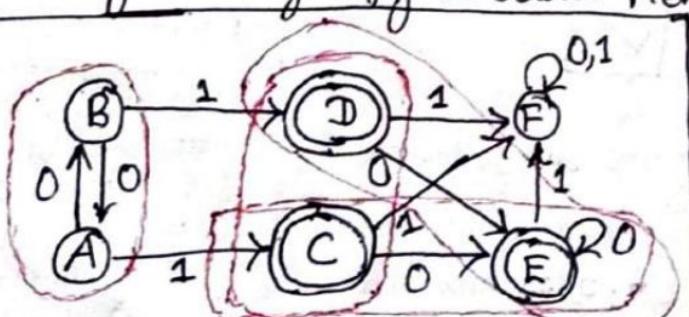
For pair FB

$$[S(F,0), S(B,0)] = [F, A] = \text{marked}$$

$$[S(F,1), S(B,1)] = [F, D] = \text{marked}$$

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).

शीरू rough fig. rough
मात्र बनाउने सबै
जसरी होल / लुप्त समिले। लाई मात्र