MODULE - 2

Regular Expressions and Languages

2.1 Regular expressions
2.2 Finite automata and regular expressions
2.3 Pumping lemma for regular languages
2.4 Myhill Nerode theorem
— Minimization of DFA

Mrs. Vidya. J
Assistant Professor,
Dept. of CSE,
GITAM, Bengaluru Campus

2.1 Regular expressions

* The regular expressions are useful for representing certain sets of string in an algebraic fashion.
* Any terminal symbol, i.e., symbols ε and ϕ are regular expression.

Operators used in Regular expressions

1. Union operator: The union of two regular expressions R1 and R2, written as $R_1 + R_2$ is also a regular expression.
2. Concatenation operator: The concatenation of two regular expressions R1 and R2, written as $R1 \cdot R2$ is also a regular expression.
3. Kleene closure: If R1 is a regular expression, then $R1^*$ (the kleene closure of R1) is also a regular expression.

Definition:

A regular expression is recursively defined as follows:

1. ϕ is a regular expression denoting an empty language - $\{\}$.
2. ε - (epsilon) is a regular expression indicates the language containing an empty string - $\{\varepsilon\}$.
3. a is a regular expression which indicates the language containing only $\{a\}$.
4. If R is a regular expression denoting the language L_R and S is a regular expression denoting the language L_S, then
   (a) R+S is a regular expression corresponding to the language $L_R \cup L_S$.

(b) R.S is a regular expression corresponding to the language $L_R \cdot L_S$

(c) R^* is a regular expression corresponding to the language L_R^*

5. The expressions obtained by applying any of the rules from 1-4 are regular expressions.

The following table shows few examples of regular expressions and the language corresponding to these regular expressions.

Regular expression	Meaning
$(a+b)^*$	Set of strings of a's and b's of any length including the NULL string.
$(a+b)^*abb$	Set of strings of a's and b's ending with the string abb.
$ab(a+b)^*$	Set of strings of a's and b's starting with the string ab.
$(a+b)^*aa(a+b)^*$	Set of strings of a's and b's having a substring aa.
$a^*b^*c^*$	Set of string consisting of any number of a's (may be empty string also) followed by any number of b's (may include empty string) followed by any number of c's (may include empty string).
$a^+b^+c^+$ (or) $aa^*bb^*cc^*$	Set of string consisting of atleast one 'a' followed by string consisting of atleast one 'b' followed by string consisting of atleast one 'c'.
$aa^*bb^*cc^*$ (or) $a^+b^+c^+$	Set of string consisting of atleast one 'a' followed by string consisting of atleast one 'b' followed by string consisting of atleast one 'c'
$(a+b)^*(a+bb)$	Set of strings of a's and b's ending with either a or bb.

$(aa)^*(bb)^*b$	Set of strings consisting of even number of a's followed by odd number of b's.
$(0+1)^*000$	Set of strings of 0's and 1's ending with three consecutive zeros (or ending with 000)
$(11)^*$	Set consisting of even number of 1's.

Obtain the regular expression for the following:

1. $L = \{0, 1, 2\}$

Soln: The above set contains only 3 input symbols. We can read '0' or '1' or '2'. So the regular expression can be written as

$$\boxed{R.E = (0+1+2)}$$

2. $L = \{\varepsilon, ab\}$

Soln: The above language contains only 2 string i.e., 'ε' and 'ab'. So the regular expression is given by

$$\boxed{R.E = (\varepsilon + ab)}$$

3. $L = \{abb, a, b, bba\}$

Soln: The language L (finite language) contains 4 strings. We can read 'abb' or 'a' or 'b' or 'bba'. So, the regular expression is given by

$$\boxed{R.E = (abb + a + b + bba)}$$

4. $L = \{\varepsilon, 0, 00, 000, \cdots\cdots\}$

Soln: The above language L, contains strings of input symbol '0' (One or more occurrences of input symbol 0) including empty string 'ε'. i.e., the above language contains zero or more occurrences of input symbol '0'. So, the regular expression can be written as,

$$\boxed{R.E = 0^*}$$ Kleene star.

5. $L = \{a, aa, aaa, aaaa, \cdots\cdots\}$

Soln: The given language L contains one or more occurrences of input symbol 'a' i.e., Kleene plus. The regular expression is $\boxed{R.E = a^+}$

6. Obtain a regular expression over {a,b} that accepts strings of length equal to 2. (=2)

Soln: String with length 2, means we will get 2 positions in the string which can be filled either by 'a' symbol or 'b' symbol.

(a+b) (a+b) → 2 positions

So, the regular expression is,

$$R.E = (a+b)(a+b)$$

7. Obtain a regular expression over {a,b} that accepts the strings of length 5.

Soln:

$$R.E = (a+b)(a+b)(a+b)(a+b)(a+b)$$ → 5 positions

8. Obtain a regular expression over {a,b} that accepts strings starting with a.

Soln: Always the strings should start with 'a', later we can have any number of a's and b's. So, the regular expression is $R.E = a.(a+b)^*$

9. Obtain a regular expression over {a,b} that accepts strings ending with ab.

Soln: All the strings should end with 'ab', before ab we can have any number of a's and b's. Hence, the regular expression is given by $R.E = (a+b)^*.ab$

10. Obtain a regular expression over {a,b,c} that accepts strings having substring abc (or) containing abc.

Soln: All the strings should have 'abc' as substring. Before 'abc' you may have any number of a's & b's (or) c's and after 'abc' also we may get any number of a's (or) b's (or) c. We may have only 'abc' as string. The regular expression is given by $R.E = (a+b+c)^*abc(a+b+c)^*$

11. Obtain a regular expression over {a,b} that accepts strings starting and ending with different symbol.

Soln: There are 2 symbols in the input alphabet i.e, 'a' and 'b'.

(i) If the string is starting with 'a' then, it has to end with 'b'. → $a(a+b)^*b$

(or)

(ii) If the string is starting with 'b' then, it has to end with 'a'. → $b(a+b)^*a$

So, the regular expression is $RE = a(a+b)^*b + b(a+b)^*a$

12. Obtain a regular expression over {a, b} that accepts strings starting and ending with same symbol.

Soln: Since there are 2 input symbols 'a' and 'b', the strings may

(i) start with and end with 'a' ⇒ $a \cdot (a+b)^* \cdot a$

(or)

(ii) start with and end with 'b' ⇒ $b \cdot (a+b)^* \cdot b$

(or)

(iii) we may take the strings 'a' (or) 'b' (or) 'ε' ⇒ $a + b + \varepsilon$

(all the starting and ending with same symbol)

The final regular expression will be

$$R.E. = a \cdot (a+b)^* \cdot a + b \cdot (a+b)^* \cdot b + (a+b+\varepsilon)$$

13. Obtain a r.e over {a, b} that accepts strings with atleast one 'a' (or) with atleast one occurrence of 'a'.

Soln: According to the problem, minimum one occurrence of 'a' should be there. No restrictions of number of b's. And later we can have any number of a's or b's.

So, the regular expression can be written as

$$R.E = b^* a (a+b)^*$$

14. Obtain a r.e over {0, 1} that accepts strings with atmost one '1' (or) with atmost one occurrence of '1'.

Soln: That is, we can have maximum of one occurrence. No restrictions on number of 0's.

(i) Either we can read only 0's. → 0^*

(or)

(ii) Any number of 0's and one occurrence of input symbol '1'.
↳ 0^*10^*

The regular expression is given by

$$R.E = 0^* + 0^*10^*$$

15) Obtain a r.e over {a, b} that accepts strings with even number of a's followed by odd number of b's.

Soln: The language can be written as,

$L = \{ b, aab, aabbb, aaaab, aaaaaabbb, \dots \}$

(i) For even number of a's, the expression is $(aa)^*$

(ii) for odd number of b's, the expression is $(bb)^*b$

and a's should be followed by b's.

So, the regular expression is written as follows:

16) Design a r.e for the following languages over $\{a,b\}$

(i) language accepting strings of length exactly 2.

(ii) language accepting strings of length atleast 2.

(iii) language accepting strings of length atmost 2.

Sol^n:

(i) language accepting strings of length exactly 2.

Refer, problem number (6).

(ii) length atleast 2.

That means, minimum length has to be 2, we can have more that 2 length also.

* For length 2: $(a+b)(a+b)$ is the regular expression
* For any length: $(a+b)^*$ is the regular expression.

Concatenate both to get the regular expression for the language accepting strings of length atleast 2.

$$R.E = (a+b)(a+b).(a+b)^*$$

(iii) atmost 2: that means, maximum length can be 2.

So, the length of the strings can be 0, 1 or 2.

Maximum we will get 2 positions, each and every position can be filled with 'ε', 'a' or 'b'.

$L = \{\varepsilon, a, b, aa, ab, ba, bb\}$

Eg: to get ε: ε ε

to get a: a ε or ε a } like wise these 2 positions will be filled

to get ba: b a

So, the regular expression is

$$R.E = (a+b+\varepsilon)(a+b+\varepsilon)$$

17) Write a r.e for strings of a's accepting

(i) even length: 0, 2, 4, 6, ... $L = \{\varepsilon, aa, aaaa, aaaaaa, \ldots\}$

$$R\epsilon = (aa)^*$$

(ii) Odd length: 1, 3, 5, 7, ... $L = \{a, aaa, aaaaa, \ldots\}$

$$R.E = a.(aa)^*$$

18) Obtain a r.e for the strings of a's and b's such that always the third symbol from the right side of the string is 'a' and fourth symbol is b.

Soln:

Let us consider 4 positions from right side (last 4 positions) before this we can have any number of a's or b's.

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

4 3 2 1 → positions

↳ always it should be filled by 'a'

↳ always it should be filled by 'b'

remaining positions can be filled either by 'a' or 'b'.

So, the regular expression is $\boxed{R.E = (a+b)^* b\, a\, (a+b)(a+b)}$

19) Obtain a regular expression to accept the strings of a's and b's such that $|w| \bmod 3 = 0$ (or) length of the string is divisible by 3 (or) multiples of 3.

Soln: The length of the strings can be, 0, 3, 6, 9, 12, 15, ... and so on.

Regular expression for the strings of length 3 is given by

$(a+b)(a+b)(a+b)$

If we repeat the above expression, we will get strings of length divisible by 3. $\boxed{R.E = ((a+b)(a+b)(a+b))^*}$

Eg: $((a+b)(a+b)(a+b))^*$ → take 0 occurrance ⇒ length of the string is 0 (zero).

↳ 1 occurrence ⇒ length of the string is 3.

↳ 3 occurrences ⇒ length of the string is 9.

} all are multiples of 3.

20) Obtain a r.e to accept the strings over {a, b} such that

(i) $|w| \bmod 3 \cong 1$ (ii) $|w| \bmod 3 \cong 2$.

(i) After dividing the length of the string by 3, we should get '1' as remainder.

So, $\boxed{R.E = ((a+b)(a+b)(a+b))^* . (a+b)}$

(ii) After dividing the length of the string by 3, we should get '2' as remainder. So write regular expression for divisible by 3 and then concatenate with 2 positions (a+b)(a+b)

So, $\boxed{R.E = ((a+b)(a+b)(a+b))^* . (a+b)(a+b)}$ (or) (4) $\boxed{R.E = (a+b)(a+b) . ((a+b)(a+b)(a+b))^*}$

21) Obtain a r.e over {a,b} that accepts string with
(i) exactly 2 a's (ii) atleast 2 a's (iii) atmost 2 a's.
(iv) a's are even (v) no 2 a's should come together.
(vi) no 2 a's and 2 b's should come together.

Sol^n:

(i) exactly 2 a's: only 2 a's and we can have any number of b's.

$$R.E = b^*ab^*ab^*$$

(ii) atleast 2 a's: minimum there should be 2 a's, later we can have any number of b's and a's.

$$R.E = b^*ab^*a(a+b)^*$$

(iii) atmost 2 a's: that is, we can have zero occurrence of 'a', one occurrence or 2-occurrence of 'a' with any number of b's.

$$R.E = b^*(\varepsilon+a)b^*(\varepsilon+a)b^*$$

(iv) even number of a's.

Language can be written as,

$$L = \{\underbrace{b, bb, bbb, \dots}_{\text{zero-occurrence of a's}}, aab, aabb, aaaab, \dots\}$$

$(b^*ab^*ab^*)^*$ → it generates 2, 4, 6, 8, ... occurrences of a's.

b^* → it generates zero occurrence of a.

So, the regular expression is

$$R.E = (b^*ab^*ab^*)^*.b^*$$

(v) No 2 a's should come together. (or) there should not be 'aa' substring.

Language can be written as,

$$L = \{\varepsilon, b, bb, bbb, a, ab, aba, abab, ba, bab, baba, babab, \dots\}$$

* By seeing the above language, we can consider 'ab' or 'ba' to construct the regular expression.

Note:
But we can't consider both.
Eg: if we consider both, then there are chances to get 2 a's together.
$(ab+ba)^*$ ⇒ from this we can obtain baab string also, where (aa) 2 a's comes together.
So, consider either 'ab' or 'ba'.

* Let us consider 'ab', and if there is no 'a' in the string then, no way 2 a's will come together.

So, we can consider, $(b + ab)^*$ — it generates any number of (ab)'s.

it generates only b's

* Observe, the above expression, the strings can start either by 'a' or 'b' but always it ends with 'b' only. But there can be strings that end with 'a' also. So, we can write the regular expression as, $(b+ab)^*.a$

it always ends with a

So, the complete regular expression is

always ending with b — always ending with a.

$$R.E = (b+ab)^* + (b+ab)^*.a$$

We can take $(b+ab)^*$ common

$$\boxed{R.E = (b+ab)^*(\varepsilon + a)}$$

(or)

* Let us consider 'ba', and if there is no 'a' at all, then no chances of getting 2 a's together.

So, the regular expression can be, $(b+ba)^*$

the above expression, starts with 'b' only and ends with either 'a' or 'b'. But there can be strings that starts with 'a' also. So, we can write $a(b+ba)^*$

So, the complete regular expression is

$$R.E = a(b+ba)^* + (b+ba)^*$$

take $(b+ba)^*$ as common and write it right side only

$$\boxed{R.E = (a+\varepsilon)(b+ba)^*}$$

because the strings can start with 'a' also.

(vi) NO 2 a's and 2 b's should come together

* Very important *

Language can be written as,

$$L = \{\varepsilon, a, b, ab, ba, aba, bab, baba, abab, \ldots\}$$

Let us divide all these strings into 4 groups, based on the starting and ending symbol.

Example strings	starts with	ends with	Regular expressions
{a, aba, ababa, ···}	a	a	$(ab)^*a$ (or) $a(ba)^*$
{ab, abab, ababab, ···}	a	b	$(ab)^*$ (or) $a(ba)^*b$
{ba, baba, bababa, ···}	b	a	$b(ab)^*a$ (or) $(ba)^*$
{b, bab, babab, ···}	b	b	$b(ab)^*$ (or) $(ba)^*b$

We can take either 'ab' or 'ba' as building blocks to build regular expressions but not both.

either we need to choose first regular expression or 2nd. But not both.

So, the regular expression for all 4 different cases can be given as follows:

$= (ab)^*a + (ab)^* + b(ab)^*a + b(ab)^*$

take $(ab)^*$ common and write it leftside because it is prefix

$= (ab)^*(a+\varepsilon) + b(ab)^*a + b(ab)^*$

take $b(ab)^*$ common from last 2 terms, write it leftside because it is prefix.

$= (ab)^*(a+\varepsilon) + b(ab)^*(a+\varepsilon)$

take $(ab)^*(a+\varepsilon)$ common, it is not prefix so write it rightside

$$R.E = (\varepsilon + b)(ab)^*(a+\varepsilon)$$

(or) We can consider 'ba' terms $a(ba)^*$, $a(ba)^*b$, $(ba)^*$, $(ba)^*b$ to construct regular expression.

Then, the r.e will be $$R.E = (\varepsilon + a)(ba)^*(\varepsilon + b)$$

22) Obtain the regular expression for the given language

$L = \{a^n b^m \mid m+n \text{ is even}\}$

Soln: The r.e to accept even number of a's followed by even number of b's can be written as,

$$(aa)^*(bb)^*$$

NKT (result)
* even + even = even
* odd + odd = even

The r.e to accept odd number of a's followed by odd number of b's can be written as, $a(aa)^*b(bb)^*$

Both will generate even length strings, so the r.e is

$$R.E = (aa)^*(bb)^* + a(aa)^*b(bb)^*$$

23) Obtain a regular expression for accepting the strings of a's and b's such that every block of 4 consecutive symbols contains atleast 2 a's.

Soln: If we consider 4 consecutive symbols, there must be atleast 2 a's. Two a's can come anywhere as shown below.

$(a(a+b)(a+b)a)^*$ (or) $(a(a+b)a(a+b))^*$ (or) $((a+b)a(a+b)a)^*$

(or) $(aa(a+b)(a+b))^*$ (or) $((a+b)aa(a+b))^*$ (or) $((a+b)(a+b)aa)^*$

So, the regular expression is

$$R.E = (a(a+b)(a+b)a)^* + (a(a+b)a(a+b))^* + ((a+b)a(a+b)a)^* + (aa(a+b)(a+b))^* + ((a+b)aa(a+b))^* + ((a+b)(a+b)aa)^*$$

2.2 Finite automata and Regular expressions

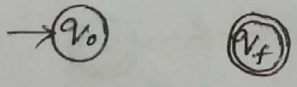
Theorem

Statement: "Let R be a regular expression then there exist a finite automata M, which accepts the language $L(R)$"

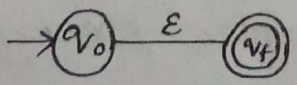
Prove that there exist a finite automata to accept a language $L(R)$ corresponding to regular expression R.

By the definition of regular expression, we have

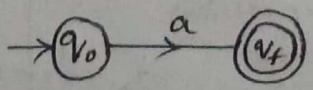
(i) ϕ - is a regular expression which accepts empty language, then its equivalent finite automata can be written as



(ii) ε - is a regular expression that accepts empty string, then its equivalent finite automata can be written as



(iii) a - is a regular expression that accepts symbol 'a', then its equivalent finite automata can be represented as

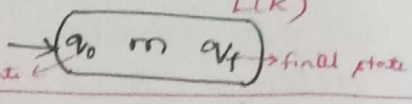


Now, let us consider R1 as a regular expression accepting the language $L(R1)$ and R2 as a regular expression accepting ⑥

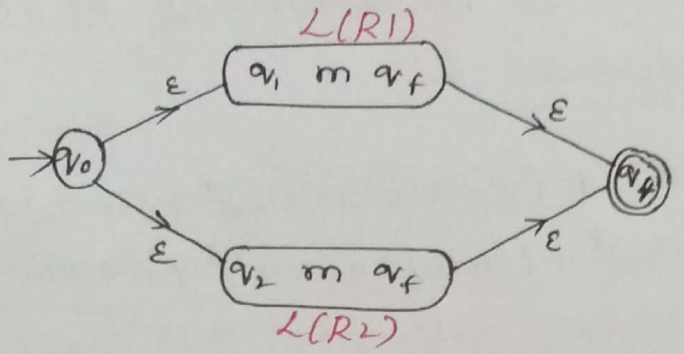
the language $L(R_2)$, then we have 3 cases:

(i) $R = R_1 + R_2$
(ii) $R = R_1 \cdot R_2$
(iii) $R = R_1^*$

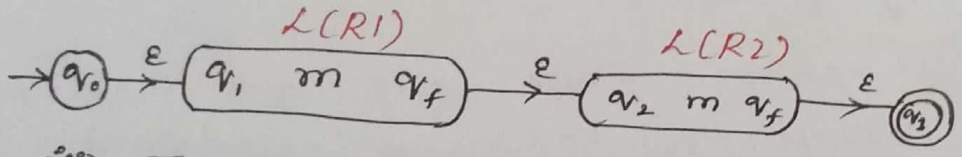
Note: The schematic diagram of any finite automata is represented as $L(R)$



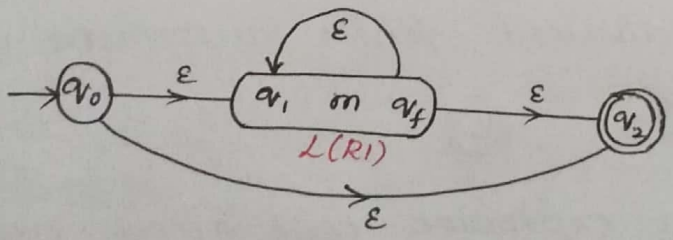
Case (i): If $R = R1 + R2$ is a regular expression then its regular language can be represented as $L(R1) + L(R2)$ then its equivalent finite automata will be



Case (ii): If $R = R1 \cdot R2$ is a regular expression accepting the language $L(R1) \cdot L(R2)$ then, its equivalent finite automata can be represented as follows:



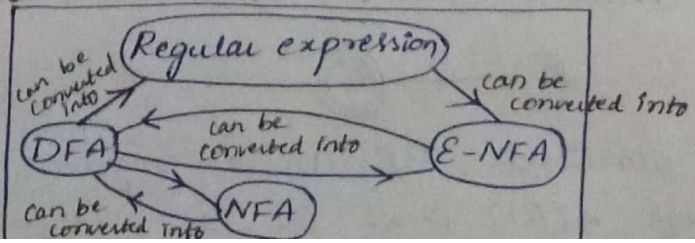
Case (iii): If $R = R_1^*$ is a regular expression then the language accepted is given by $L(R_1^*)$ and its equivalent finite automata will be,



Hence, by the definition of regular expression for all the cases of regular expression we have a finite automata.

Hence proved.

The relationship between DFA, NFA, ~~and~~ ε-NFA and regular expression is as shown below.

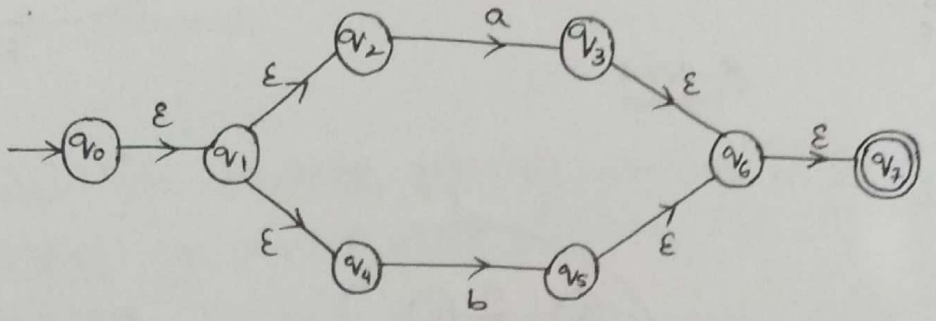


Construction of finite automata from regular expression

Problems:

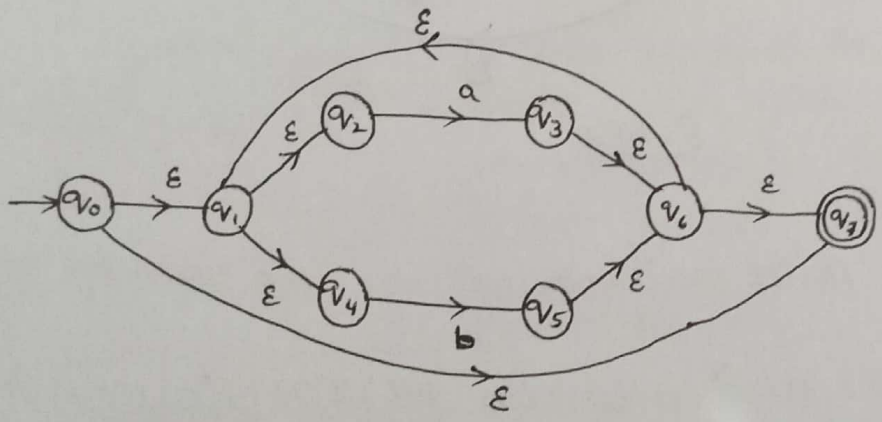
1) Construct a finite automata for the given regular expression $(a+b)^*$

Soln:

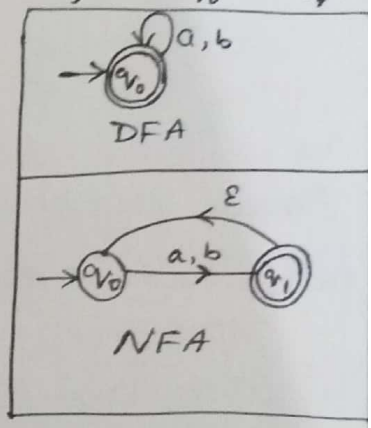


The above automata will read either 'a' or 'b'.

* To repeat this, take an ε-transition from q_6 to q_1.
* For zero occurrence, take another ε-transition from q_0 to q_7.

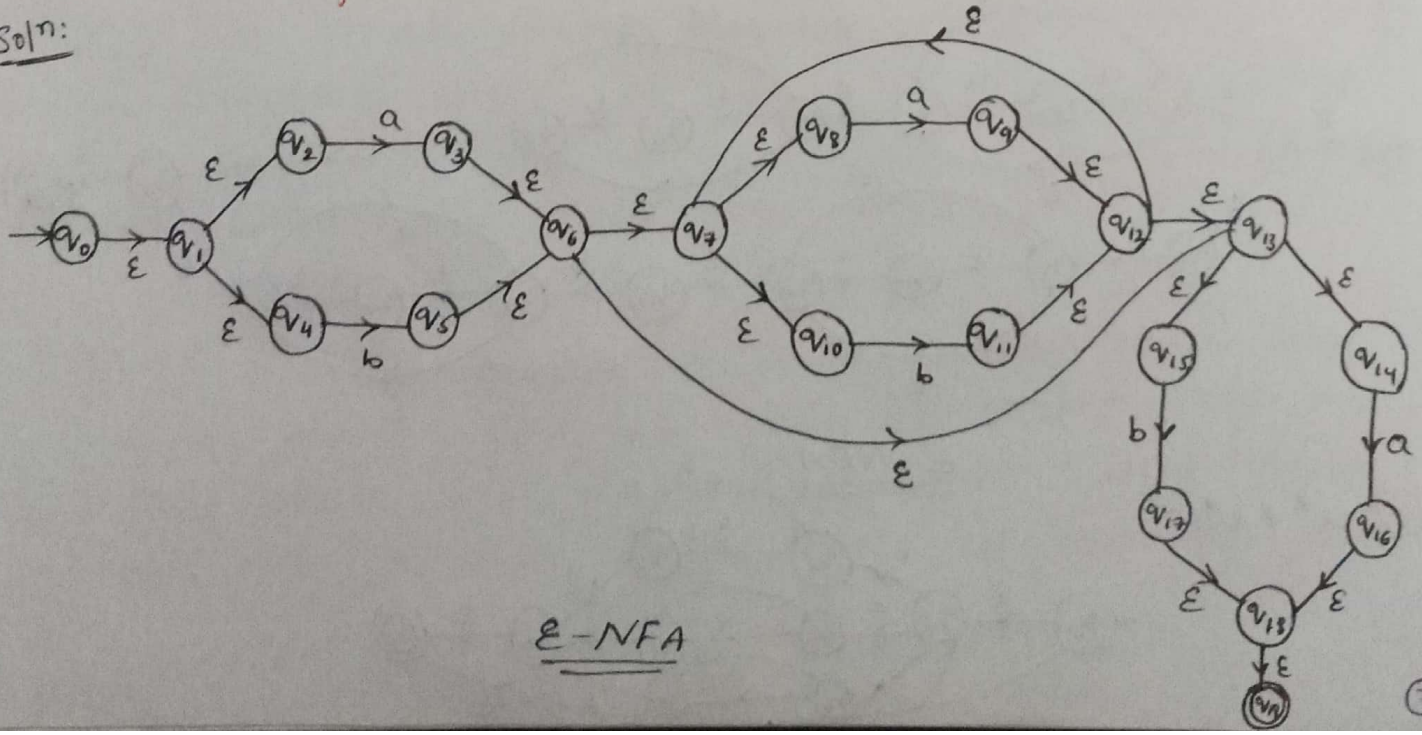


ε-NFA



2) Construct a finite automata for the r.e $(a+b)(a+b)^*(a+b)$.

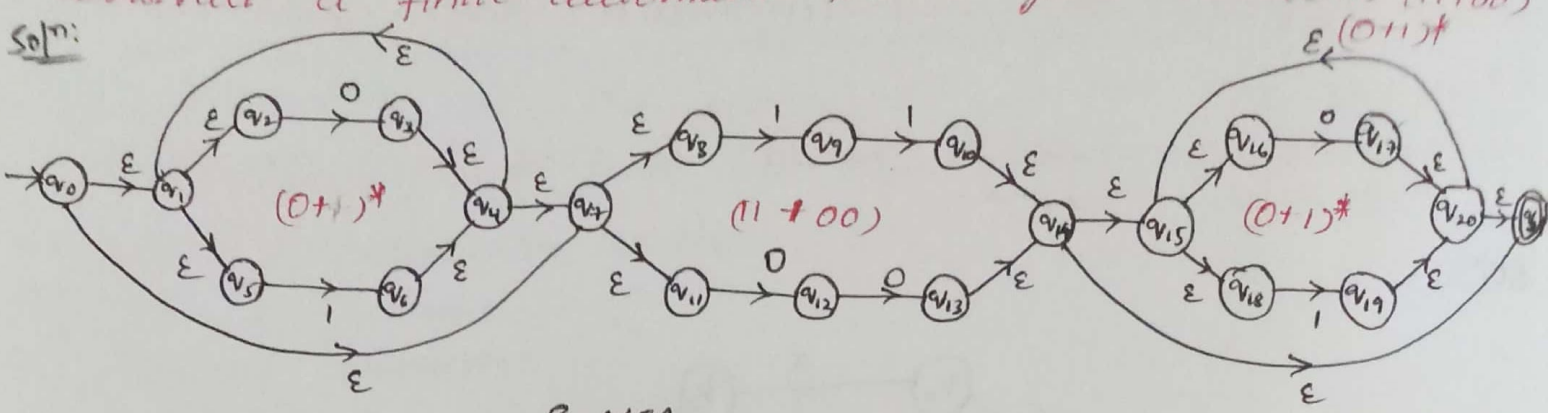
Soln:



ε-NFA

3) Construct a finite automata for the given r.e $(0+1)^*(11+00)$

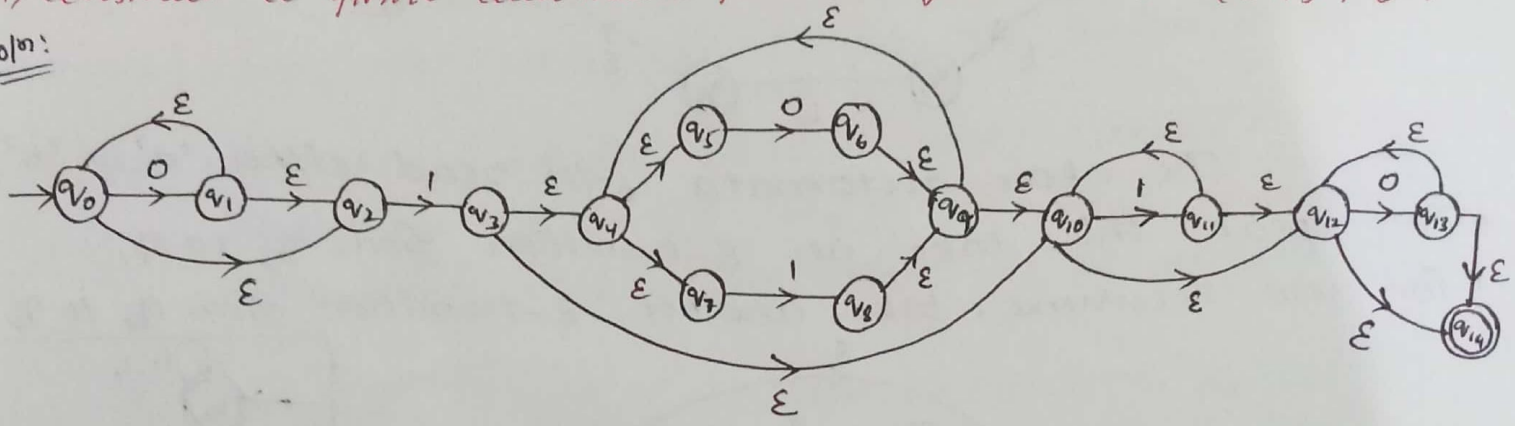
Soln:



ε-NFA

4) Construct a finite automata for the given r.e $0^*1(0+1)^*1^*0^*$.

Soln:



ε-NFA

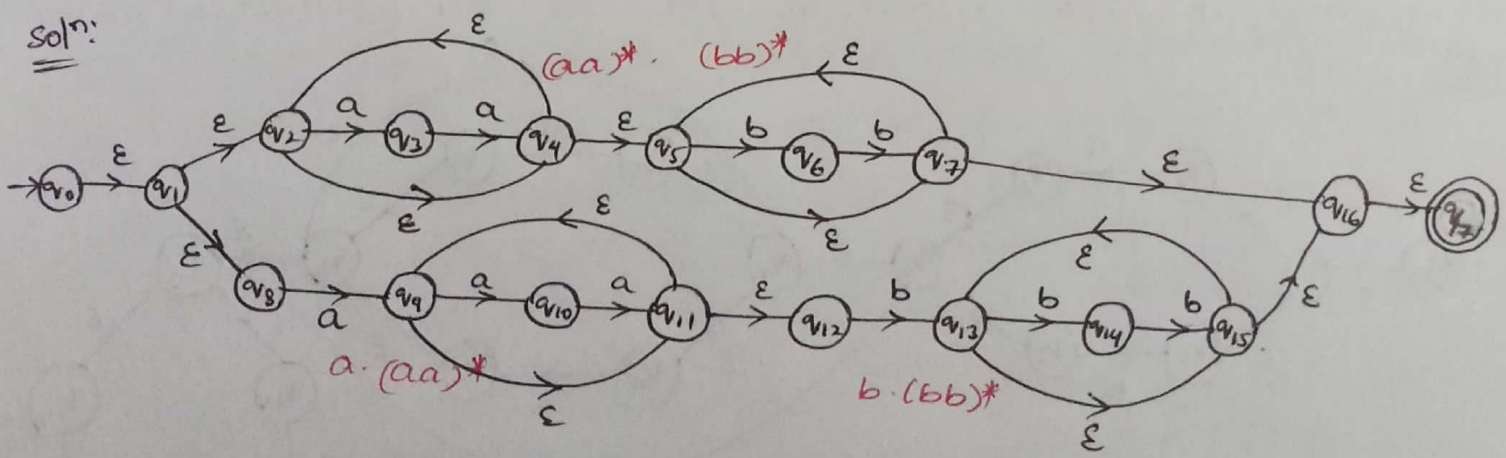
Exercise problems

1) Construct the finite automata for the following regular expressions.

(i) $(a+b)^*ab$ (ii) $(ab)^*(a+b)abb$ (iii) $(0+1)^*011(0+1)^*$

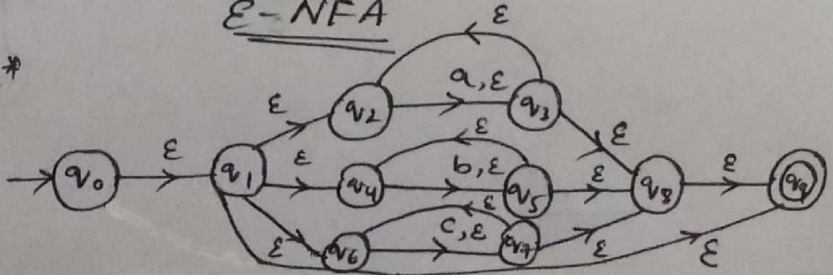
(iv) $(aa)^*(bb)^* + a(aa)^*b(bb)^*$ (v) $a^*+b^*+c^*$ (vi) $10+(0+11)0^*1$

Soln:



ε-NFA

(v) $a^*+b^*+c^*$



Conversion of Finite Automata into Regular Expression

The following are the ways to convert the given finite automata into its equivalent regular expression.

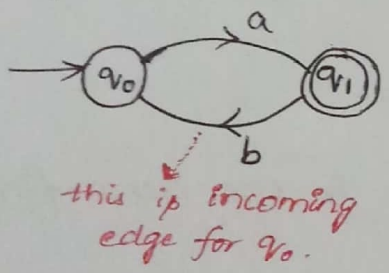
(i) State Elimination method
(ii) Arden's theorem
(iii) Kleene's theorem

1. State Elimination Method

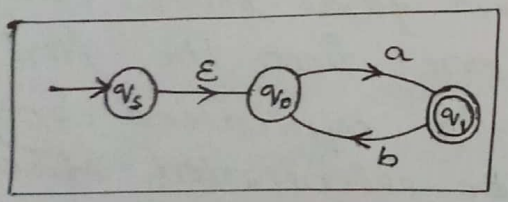
In order to convert finite automata into regular expression, we need to follow the below steps:

Step 1: The start state should not contain any incoming edges. If there is any incoming edge, then create a new start state.

Eg:



In this automata, q_0 is the start state and it has an incoming edge from state q_1 on input symbol 'b'. So, we need to add a new start state as shown below.



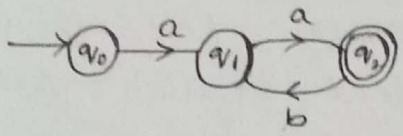
⇒ New state q_s is added to this finite automata

Step 2: The final state should not contain any

(a) outgoing edges. If there is any outgoing edge then create a new final state from the (old) existing final state and make it as a non-final state. (old final state)

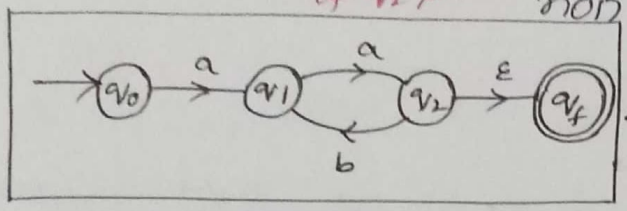
(b) If finite automata contains more than one final states, then we need to create only one final state from those final states and make all the previous final states as non-final state.

Eg: Step 2: (a) With only one final state.



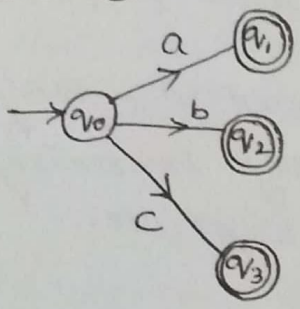
In this automata, q_2 is the final state. It has an outgoing edge with label b. So, add a new final state from q_2 and make q_2 as a non-final state as shown below:

outgoing edge of q_2 state.

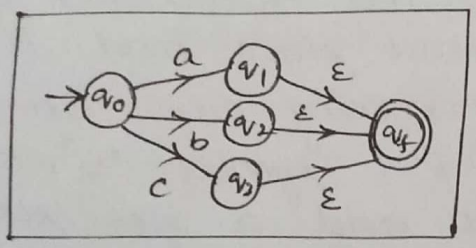


→ q_f is the new final state.

Step 2: (b) With more than one final states.



In this automata, there are 3 final states q_1, q_2 and q_3. Now, create only one final state from all these states. And make q_1, q_2 and q_3 as non-final states as shown.

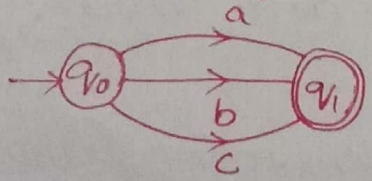


⇒ q_f is the new final state.

Step 3: Except start state and final state, eliminate the remaining states one-by-one from the finite-automata by deriving its equivalent regular expression. Trap state can be eliminated directly.

Example problems

1. Convert the given finite automata into its equivalent regular expression using State Elimination method.



Solⁿ:

Step 1: The start state q_0, is not having any incoming edge. So, no need to create new start state.

Step 2: (a) The final state q_1, is not having any outgoing edge. So, no need to create a new final state.

Step 3: Except start and final states, there is no other state to eliminate.

So, derive the regular expression.

(i) From q_0, by reading 'a' input symbol we can reach final state q_1.

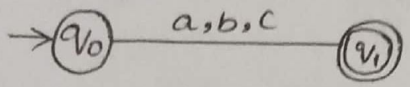
(or)

(ii) By reading 'b' input symbol also, we can reach from q_0 to q_1

(or)

(iii) By reading 'c' also, we can go from q_0 to q_1.

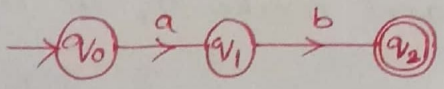
The finite automata can be written as,



Hence, the regular expression is given by,

$$R.E = a+b+c$$

② Convert the following finite automata into regular expression using state elimination method.

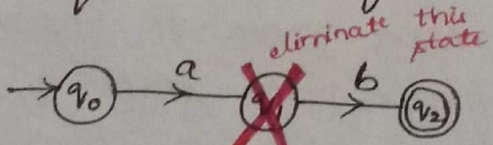


Soln:

Step1: The start state q_0, does not have any incoming edges. So, no need to create new start state.

Step 2: The final state q_2, does not have any outgoing edges. So, no need to create a new final state.

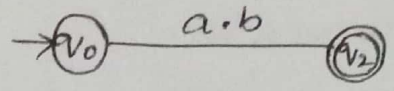
Step 3: Except the start state (q_0) and final state (q_2), eliminate the remaining state q_1, by considering its equivalent regular expression.



→ From q_0, by reading 'a' we were able to reach state q_1. From q_1 by reading 'b' we were able to reach state q_2 previously.

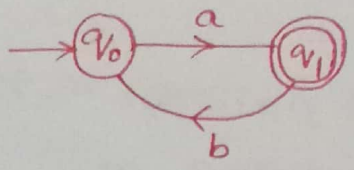
→ As we have eliminated state q_1, we can reach state q_2 directly from q_0 by reading 'a' followed by 'b'

The automata can be written as follows:

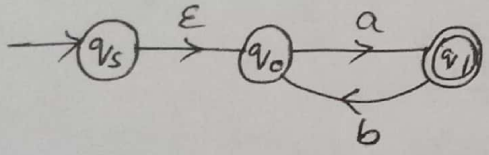


The regular expression is, $\boxed{R.E = a.b}$

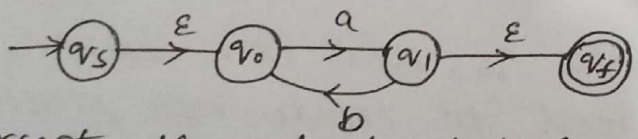
3. Convert the following finite automata into its equivalent regular expression using state elimination method.



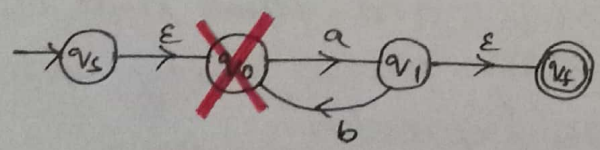
Sol: Step 1: The start state q_0, is having an incoming edge, so, add a new start state to the given automata as follows:



Step 2: The final state q_1 is having an outgoing edge. So, create a new final state from the existing final state as shown below and make q_1 as a non-final state.

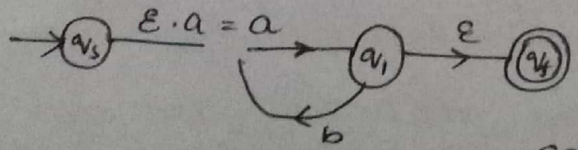


Step 3: Except the start state (q_s) and final state (q_f), eliminate the states q_0 and q_1, by deriving its equivalent regular expression.

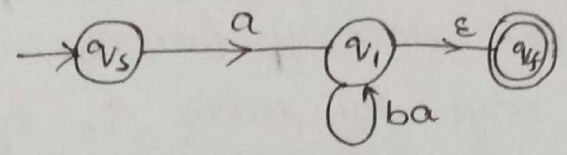
(i)  – eliminating state q_0.

→ From q_s by reading 'ε' we were able to reach q_0, further by reading 'a' we can reach the state q_1.

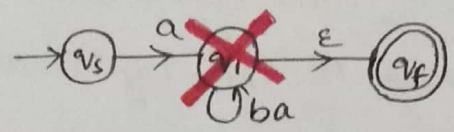
→ So, from q_s by reading 'ε·a' = 'a' we can reach q_1. From q_1 there is a edge going to q_0 state. But, q_0 is eliminated. By reading input symbol a after b we can come back to q_1 only. So, it will become a self loop.



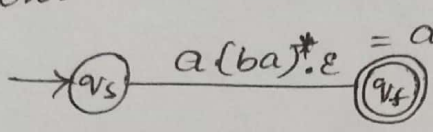
→ The finite automata can be written as follows:



(ii) Eliminate state q_1

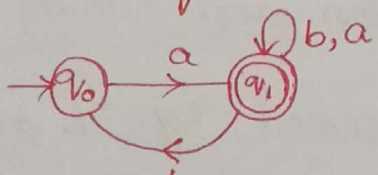


Now, there is a path from q_s to q_f. By seeing 'a' followed by any number of 'ba', followed by 'ε' we can reach final state q_f. The automata can be written as follows

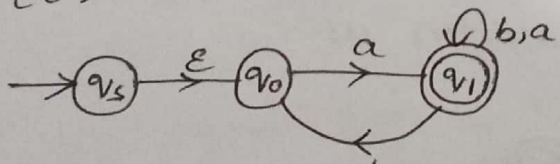


The Regular expression is, $\boxed{R.E = a(ba)^*}$

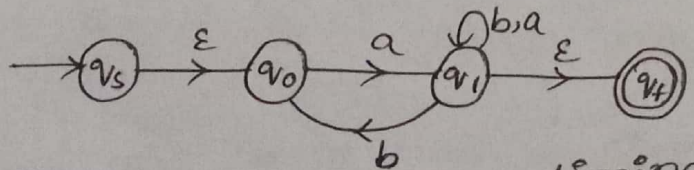
4. Obtain the regular expression from the given finite automata using state elimination method



Soln: Step 1: There is an incoming edge for the start state q_0. So, add a new start state as shown below.

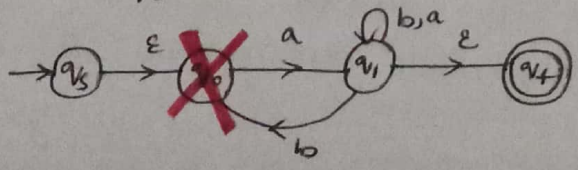


Step 2: There is an outgoing edge from final state q_1. So, add a new final state as shown below from q_1 and make q_1 as non final state.



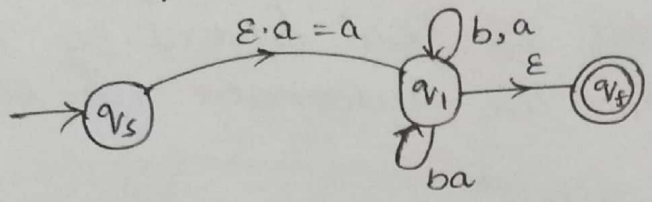
Step 3: Except q_s and q_f eliminate the remaining states.

(i) Eliminate q_0.

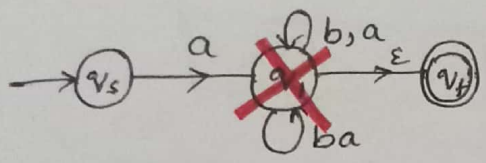


↦ Now, there exist a direct path from q_s to q_f, i.e, by reading "ε.a" we can reach state q_f from q_s.

↦ From q_1 there is an edge to state q_0. By reading 'b' symbol, we can reach q_0 from q_1, as q_0 is eliminated further by reading 'a' symbol we can come back to q_1 (starting from q_1 we are going back to q_1 - by reading 'ba') -so add self loop as shown below.



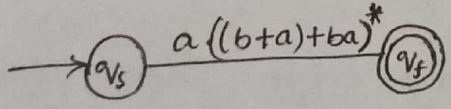
(ii) Now eliminate q_1 state.



↦ Now there exist a direct path from q_s to q_f. Consider the regular expression and eliminate q_1 state.

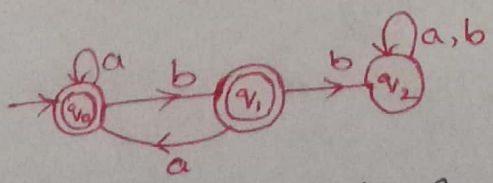
↦ From q_f, by reading 'a' followed by 'b' or 'a' any number of times or by reading 'ba' any number of times, followed by 'ε' we can reach q_f.

The automata can be written as



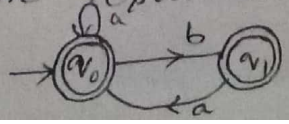
The regular expression is $\boxed{RE = a\cdot((b+a)+ba)^*}$

5) Obtain a regular expression for the given automata by using state elimination method.

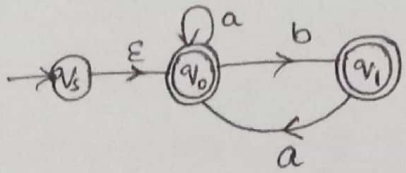


Soln:

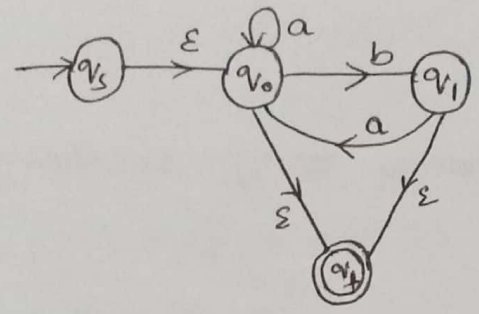
Step 1: The trap state q_2 can be eliminated directly. So the resultant finite automata is as shown below.



The start state q_0 has an incoming edge, so add a new start state as shown below

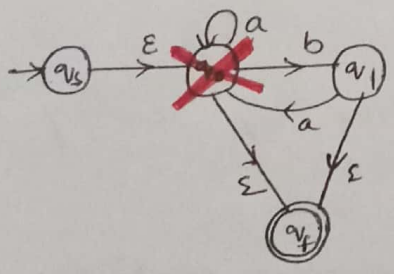


Step 2: There are 2 final states q_0 and q_1. Final states are having outgoing edges. So add one final state from both final states. Make the previous final states (q_0 and q_1) as non-final states.



Step3: Except start state q_s and final state q_f, eliminate the remaining states.

(i) eliminate state q_0.



→ Now, we can reach from q_s to q_1 directly by reading 'ε' followed by any number of a's followed by 'b'. so the expression is $\varepsilon \cdot a^*b = a^*b$.

→ From q_s, we can reach q_f directly, because q_0 is eliminated.

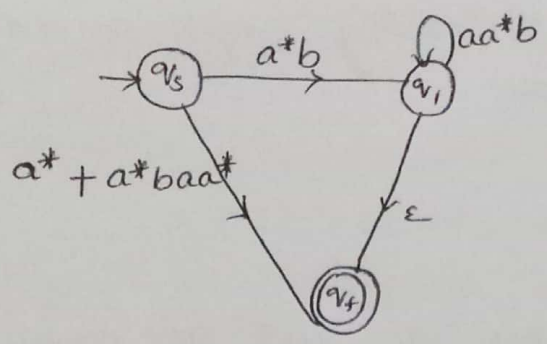
* Read ε (from q_s) followed by any number of a's, followed by ε and reach state q_f. i.e., $\varepsilon \cdot a^* \cdot \varepsilon = a^*$

(or)

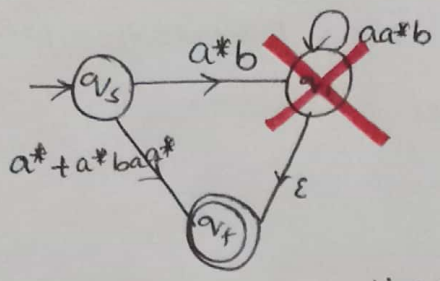
* Read ε and go to state q_0, read any number of a's followed by 'b' and reach q_1, then by reading 'a' come back to q_0 then take ε-transition and reach q_f.

i.e., $\varepsilon \cdot a^*b \cdot aa^* \cdot \varepsilon = a^*b \cdot aa^*$

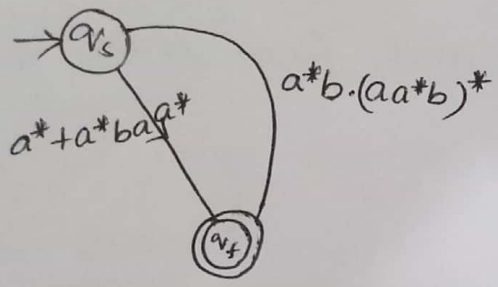
→ From state q_1, if we take transition of a, we will reach q_0. As q_0 is eliminated, read a^*b (after reading a) and come back to q_1. This is a self loop aa^*b.



(ii) Now, eliminate state q_1.



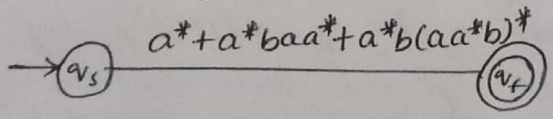
Now, there exist a direct path from q_s to q_f as shown below.



We can reach q_f from q_s either by reading

(i) $a^* + a^*baa^*$ (or)

(ii) a^*b followed by (aa^*b) any number of times.



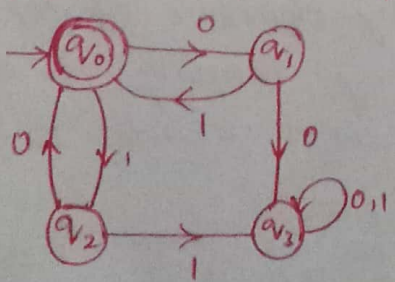
So, the regular expression is given by

$$RE = a^* + a^*baa^* + a^*b(aa^*b)^*$$

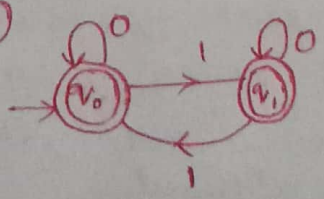
Exercise problems:

1) Convert the following finite automata into regular expression.

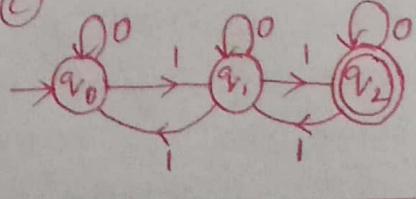
(a)



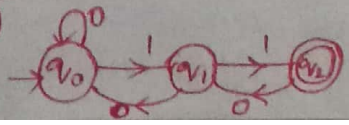
(b)



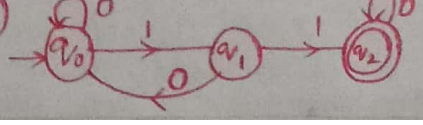
(c)



(d)



(e)



2. Arden's Theorem

Statement: "If P and Q are two regular expressions over Σ, and if P does not contain ε, then the following equation in R given by $R = Q + RP$ has a unique solution i.e., $R = QP^*$"

Consider the equation $R = Q + RP \rightarrow (1)$

replace, R with QP^*

So, equation (1) becomes

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

$$= Q + Q \cdot P^* \cdot P$$

Take Q as common

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

we have, $\boxed{\varepsilon + R^*R = R^*}$

$$\boxed{R = Q \cdot P^*}$$

Hence proved.

Now, let us prove that this is the unique solution

$$R = Q + RP$$

Now, replace R with $Q + RP$ in equation (1)

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

$$= Q + QP + RP^2$$

Once again replace R by $Q + RP$

$$= Q + QP + (Q + RP) \cdot P^2$$

$$= Q + QP + QP^2 + RP^3$$

replace R by $Q + RP$ once again

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

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

⋮ if we do this 'n' number of times, finally we get

$$= Q + QP + QP^2 + \cdots\cdots + QP^n + RP^{n+1}$$

Take Q as common

Now, replace R by $Q P^*$

$$= Q + QP + QP^2 + \cdots\cdots + QP^n + QP^*.P^{n+1}$$

Take Q as common

$$= Q[\varepsilon + P + P^2 + \cdots\cdots + P^n + P^*.P^{n+1}]$$

WKT, $\varepsilon + R^*R = R^*$

So, above equation becomes

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

Hence, we proved that, this is the only solution.

How to convert finite automata to regular expression by using Arden's theorem.

The following are the steps to be followed:

1. To get the regular expression from the automata we first create the equations for each state present in the finite automata.

[Note: Consider the incoming edges only to construct the equation for each state].

The state equation is in the form of

$$q_1 = q_1 w_{11} + q_2 w_{21} + \cdots\cdots + q_n w_{n1} + \varepsilon$$

(where q_1 is the initial / start state)

$$q_2 = q_1 w_{12} + q_2 w_{22} + \cdots\cdots + q_n w_{n2}$$

$$\vdots$$

$$q_n = q_1 w_{1n} + q_2 w_{2n} + \cdots\cdots + q_n w_{nn}$$

Where, w_{ij} is the regular expression representing the set of labels of edges (input symbols) from q_i to q_j.

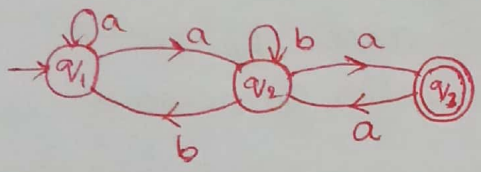
2. Then we solve these equations to get the equation for q_i in terms of w_{ij} and that expression is the required solution, where q_i is a final state.

Assumptions made while forming the regular expression

* The transition diagram should not have ε-transitions.
* It must have only one start state.

Problems:

1. Convert the given finite automata into its equivalent regular expression using Arden's theorem.



Soln:

→ We can directly apply the Arden's theorem, because it has no ε-transitions and it has only one start state.

→ There are 3 states in the given finite automata i.e., q_1, q_2 and q_3.

→ Now, write 3 equations for q_1, q_2 and q_3 as follows:

$$q_1 = q_1a + q_2b + \varepsilon \rightarrow (1)$$

$$q_2 = q_1a + q_2b + q_3a \rightarrow (2)$$

$$q_3 = \varepsilon + q_2a + \varepsilon \rightarrow (3)$$

Substitute q_3 value in equation (2)

$$q_2 = q_1a + q_2b + (q_2a)a$$

$$= q_1a + q_2b + q_2aa$$

take q_2 common from last 2 terms

$$R = Q + R \cdot P$$

$$q_2 = q_1a + q_2(b + aa)$$

The above equation is in the form $R = Q + RP$

where $R = q_2$, $Q = q_1a$, $P = (b + aa)$

The equation $R = Q + RP$ has only one solution i.e., $R = QP^*$

So, $\boxed{q_2 = q_1a \cdot (b + aa)^*} \rightarrow (4)$

Substitute q_2 value in equation (1)

$$q_1 = q_1 a + q_1 a(b+aa)^*.b + \varepsilon$$

Take q_1 common from first two terms

$R = R \cdot P + Q$

$$q_1 = q_1(a + a(b+aa)^*b) + \varepsilon$$

The above equation is in the form $R = Q + RP$

where, $R = q_1$, $P = (a + a(b+aa)^*b)$, $Q = \varepsilon$

The only solution is QP^*,

hence, $q_1 = \varepsilon \cdot (a+a(b+aa)^*b)^* = (a + a(b+aa)^*b)^*$

$$\boxed{q_1 = (a + a(b+aa)^*b)^*}$$

Similarly, find out q_3 value and q_2 value.

$q_2 = q_1 a (b+aa)^*$ equation (4) becomes.

Substitute q_1 value,

$$\boxed{q_2 = (a+a(b+aa)^*b)^* a(b+aa)^*}$$

$q_3 = q_2 a$ equation (3) becomes

Substitute q_2 value,

$$\boxed{q_3 = (a+a(b+aa)^*b)^* a(b+aa)^*.a}$$

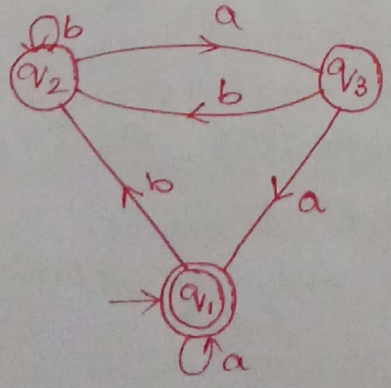
In the given finite automata, q_3 is the final state.

Hence, the value of q_3 is the required regular expression.

So, the regular expression is

$$\boxed{R.E = (a+a(b+aa)^*b)^* a(b+aa)^* a}$$

2) Convert the given finite automata into regular expression using Arden's rule.



Soln: Write the equations for all the states present in the given finite automata.

$q_1 = q_1a + \varepsilon + q_3a \rightarrow (1)$

$q_2 = q_1b + q_2b + q_3b \rightarrow (2)$

$q_3 = \varepsilon + q_2a + \varepsilon \rightarrow (3)$

Substitute q_3 value in equation (2)

$q_2 = q_1b + q_2b + q_2a \cdot b$

Take q_2 as common from last 2 terms

$q_2 = q_1b + q_2(b + ab)$

This is of the form $R = Q + RP$, where $R = q_2$, $P = (b+ab)$, and $Q = q_1b$

The solution will be $R = QP^*$,

hence, $\boxed{q_2 = q_1b(b+ab)^*} \rightarrow (4)$

Substitute q_3 value in equation (1)

$q_1 = q_1a + \varepsilon + q_2a \cdot a$

now, substitute q_2 value in the above equation.

$= q_1a + q_1b(b+ab)^*aa$

Take q_1 as common

$q_1 = q_1(a + b(b+ab)^*aa)$

The above equation is of the form $R = Q + RP$.

where, $R = q_1$, $Q = \varepsilon$, $P = (a + b(b+ab)^*aa)$

so, the solution is $R = QP^*$

$q_1 = \varepsilon \cdot (a + b(b+ab)^*aa)^*$

$\boxed{q_1 = (a + b(b+ab)^*aa)^*}$ substitute this in q_2

so, q_2 becomes, equation (4)

$\boxed{q_2 = (a + b(b+ab)^*aa)^* \cdot b(b+ab)^*}$

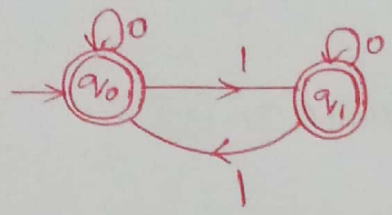
Substitute q_2 value in equation (3)

$\boxed{q_3 = (a + b(b+ab)^*aa)^* \cdot b(b+ab)^* \cdot a}$

In the given finite automata, final state is q_1, so the regular expression is value of q_1.

$\boxed{R.E = (a + b(b+ab)^*aa)^*}$

3) Convert the following finite automata into regular expression.



Soln:

Write the equations for the 2 states q_0 and q_1

$$q_0 = q_0 0 + q_1 1 \rightarrow (1)$$

$$q_1 = q_0 1 + q_1 0 \rightarrow (2)$$

Observe equation (2) which is of the form $R = Q + RP$

where, $R = q_1$, $Q = q_0 1$, $P = 0$

The solution is $R = QP^*$,

so q_1 becomes, $\boxed{q_1 = q_0 1 0^*} \rightarrow (3)$

Substitute q_1 value in equation (1)

$$q_0 = q_0 0 + q_0 10^* 1$$

Take q_0 as common,

$q_0 = q_0 (0 + 10^* 1)$ this is of the form $R = Q + RP$

where, $R = q_0$, $Q = \varepsilon$, $P = (0 + 10^* 1)$

the solution is $R = QP^*$

$$q_0 = \varepsilon \cdot (0 + 10^* 1)^*$$

$$\boxed{q_0 = (0 + 10^* 1)^*}$$

Substitute q_0 value in equation (3)

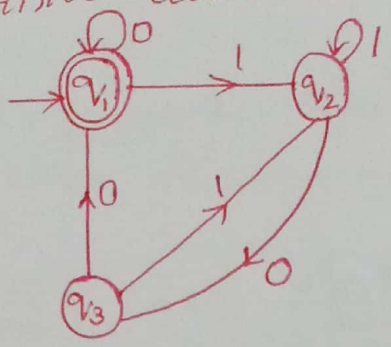
$$\boxed{q_1 = (0 + 10^* 1)^* 10^*}$$

In the given finite automata, both q_0 as well as q_1 are final states. So, we need to consider the value of q_0 or the value of q_1. The regular expression is given by,

q_0 (OR) q_1

$$\boxed{R \cdot E = (0 + 10^* 1)^* + (0 + 10^* 1)^* 10^*}$$

4) Obtain the regular expression for the given finite automata.



Soln: Write the equations for the 3 states present in the given finite automata.

$$q_1 = q_1 0 + \varepsilon + q_3 0 \rightarrow (1)$$

$$q_2 = q_1 1 + q_2 1 + q_3 1 \rightarrow (2)$$

$$q_3 = \varepsilon + q_2 0 + \varepsilon \rightarrow (3)$$

Substitute the value of q_3 in equation (2)

$$q_2 = q_1 1 + q_2 1 + q_3 1$$

$$= q_1 1 + q_2 1 + q_2 0 1$$

Take q_2 common, from last 2 terms.

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

This equation is of the form $R = Q + RP$,

where, $R = q_2$, $Q = q_1 1$, $P = 1 + 01$

The solution is $R = QP^*$,

$$\boxed{q_2 = q_1 1 \cdot (1+01)^*} \rightarrow (4)$$

Substitute q_2 value in equation (3)

$$\boxed{q_3 = q_1 1 \cdot (1+01)^* \cdot 0} \rightarrow (5)$$

Substitute q_3 value in equation (1)

$$q_1 = q_1 0 + \varepsilon + q_1 1 (1+01)^* 0 \cdot 0$$

Take q_1 as common,

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

This is of the form $R = Q + RP$ where, $R = q_1$, $Q = \varepsilon$,

$P = (0 + 1(1+01)^* 00)$

The solution is $R = QP^*$, so q_1 becomes

$$q_1 = \varepsilon \cdot (0 + 1(1+01)^* 00)^* = \boxed{(0 + 1(1+01)^* 00)^*} \rightarrow (6)$$

(15)

Substitute q_1 value in (4)

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

$$\boxed{q_2 = (0+1(1+01)^*00)^*.1(1+01)^*}$$

Substitute q_1 value in (5)

$$q_3 = q_1 1 \cdot (1+01)^*.0$$

$$\boxed{q_3 = (0+1(1+01)^*00)^*.1(1+01)^*0}$$

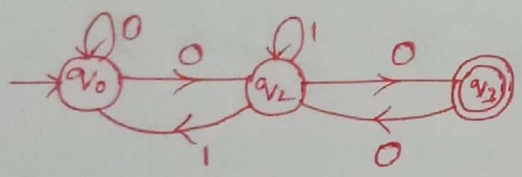
As q_1 is the final state, the regular expression will be

$$\boxed{R.E = (0+1(1+01)^*00)^*} \rightarrow q_1 \text{ value.}$$

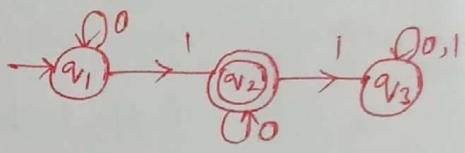
Exercise problems:

Obtain the regular expression for the following finite automata.

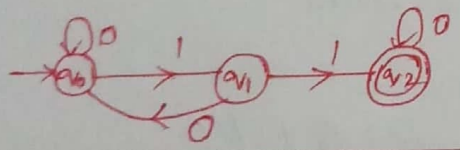
(a)



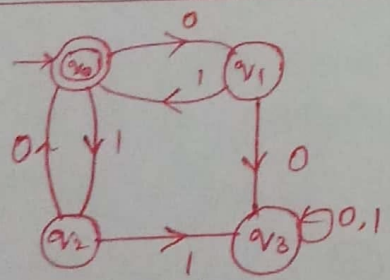
(b)



(c)



(d)



Soln: Equations

$$q_0 = \varepsilon + q_1 1 + q_2 0 + \varepsilon \rightarrow (1)$$

$$q_1 = q_0 0 + \varepsilon + \varepsilon + \varepsilon = q_0 0 \rightarrow (2)$$

$$q_2 = q_0 1 + \varepsilon + \varepsilon + \varepsilon = q_0 1 \rightarrow (3)$$

$$q_3 = \varepsilon + q_1 0 + q_2 1 + q_3(0+1) \rightarrow (4)$$

Substitute q_1 and q_2 values in equation (1)

$q_0 = q_0 0 \cdot 1 + q_0 \cdot 1 \cdot 0$ take q_0 as common

$\boxed{q_0 = q_0(01+10)}$ → this is of the form $R = Q + RP$ where $R = q_0$, $Q = \varepsilon$, $P = (01+10)$

The solution is $R = QP^*$

$q_0 = \varepsilon \cdot (01+10)^*$

The final state is q_0, so the regular expression is value of q_0 i.e., $\boxed{R.E = (01+10)^*}$

3. Kleene's theorem

Theorem: Let $M=(Q,\Sigma,\delta,q_0,F)$ be a finite automata recognizing the language L, then there exists an equivalent regular expression R for the regular language L such that $L=L(R)$

Proof: Let $Q=\{q_1,q_2,q_3,q_4,\ldots\ldots q_n\}$ be the set of states of machine where n is the number of states. The path from state i to j through an intermediate state whose number is not greater than k is given by regular expression R_{ij}^{k} as shown below.

$$R_{ij}^{(k)} = \{w \in \Sigma^* \mid w \text{ is a path from state } i \text{ to state } j \text{ that goes through an intermediate state whose number is not greater than } k\}$$

where $i>k$ and $j>k$. The string w can be written as

$$w=xy, \text{ where } |x|>0 \text{ and } |y|>0$$

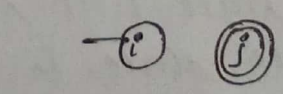
and $\delta^*(i,x)=k$ and $\delta^*(k,y)=j$

Basis: Let us consider $k=0$, this indicates that there is no intermediate states and the path from states i to j is obtained by the following 2 conditions.

Condition 1: There is a direct edge from state i to j where $i \neq j$. Here, a DFA with all the input symbols 'a' such that there is a transition from i to j is considered with the following cases:

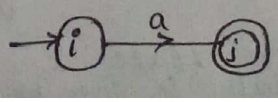
(a) No input symbol and the corresponding regular expression is given by,

$$\boxed{R_{ij}^{(0)} = \phi}$$



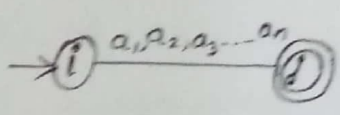
(b) If there is exactly one input symbol 'a', then the regular expression is given by,

$$\boxed{R_{ij}^{(0)} = a}$$



(c) If there are 'n' input symbols i.e; $a_1, a_2, \ldots a_n$ then its regular expression is given by,

$$R_{ij}^{(0)} = a_1 + a_2 + a_3 + \cdots + a_n$$



Condition 2: If $i=j$ indicates that there is only one state which itself is a final state and there doesnot exist any intermediate states.

case (a): No input symbol and the corresponding regular expression is given by $R_{ij}^{(0)} = \phi + \varepsilon$ → a self loop or a path of length zero, is denoted by ε.

case (b): If there is exactly one input symbol 'a' then the regular expression is given by, $R_{ij}^{(0)} = a + \varepsilon$

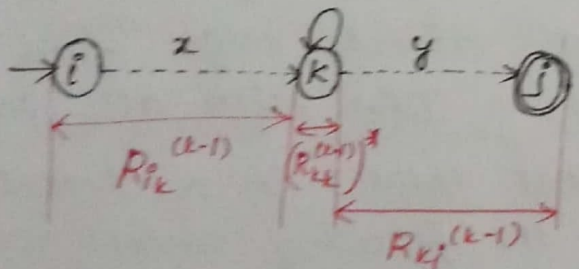
case (c): If there are 'n' input symbols i.e., $a_1, a_2, a_3 \cdots a_n$ then its regular expression is given by, $R_{ij}^{(0)} = a_1 + a_2 + \cdots + a_n + \varepsilon$

Induction:

Suppose there exist a path from i to j through a state which is not higher than k. This situation leads to 2 cases,

case (i): There exists a path from state i to j which does not go through k and the language accepted is given by $R_{ij}^{(k-1)}$ regular expression.

case (ii): There exists a path from state i to j through k as shown below.



The path from state i to j can be broken into several pieces,

(i) The path from state i to state k not passing through any intermediate state is given by $R_{ik}^{(k-1)}$

(ii) The path from state k to k not passing through any intermediate state is given by $(R_{kk}^{(k-1)})^*$

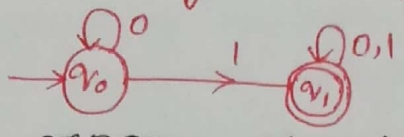
(iii) The path from state k to j not passing through any intermediate state is given by $R_{kj}^{(k-1)}$

So, the regular expression for the path from state 'i' to 'j' is given by,

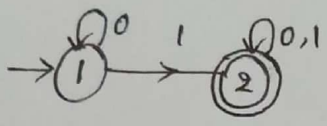
$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$

Problems:

1) obtain the regular expression for the given finite automata using kleene's theorem



Sol^n: Let us rename the states, $q_0 = 1$ and $q_1 = 2$. The above finite automata can be written as



From kleene's theorem, we have

* **Basis:** When $k = 0$

(i) $R_{11}^{(0)} = (0+\varepsilon)$

(ii) $R_{12}^{(0)} = 1$

(iii) $R_{21}^{(0)} = \phi$

(iv) $R_{22}^{(0)} = (0+1+\varepsilon)$

Note: If there is a loop, then add ε (epsilon) which denotes the length zero.

* **Induction:** The regular expression corresponding to the path from state i to state j through a state which is not higher than k is given by,

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \cdot \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$

When $k=1$, i.e., path from state i to j through a state not higher than 1). The various regular expressions obtained are as shown below.

* $R_{11}^{(1)}$ (i=1, j=1, k=1) $= R_{11}^{(0)} + R_{11}^{(0)} \cdot \left(R_{11}^{(0)}\right)^* \cdot R_{11}^{(0)}$

$= (0+\varepsilon) + (0+\varepsilon)\cdot(0+\varepsilon)^* \cdot (0+\varepsilon)$ ($R \cdot R^* = R^+$) $= (0+\varepsilon) + (0+\varepsilon)^+ \cdot (0+\varepsilon)$

$= (0+\varepsilon) + 0^* \cdot (0+\varepsilon)$ (0^*, because ε is there; we have, $R^*(R+\varepsilon) = R^*$)

$= (0+\varepsilon) + 0^*$

$R_{11}^{(1)} = 0^*$

(*) $R_{12}^{(1),k}$ (i → j) $= R_{12}^{(0)} + R_{11}^{(0)} \cdot (R_{11}^{(0)})^{*} \cdot R_{12}^{(0)}$

$= 1 + (0+\varepsilon) \cdot (0+\varepsilon)^{*} \cdot 1$

$= 1 + (0+\varepsilon)^{+} \cdot 1$

$= 1 + 0^{*} \cdot 1$

$\boxed{R_{12}^{(1)} = (\varepsilon + 0^{*}) \cdot 1 = 0^{*}1}$

(*) $R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)} \cdot (R_{11}^{(0)})^{*} \cdot R_{11}^{(0)}$ ($i=2, j=1, k=1$)

$= \phi + \phi \cdot (0+\varepsilon)^{*} \cdot (0+\varepsilon)$

$\boxed{R_{21}^{(1)} = \phi}$

(*) $R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} \cdot (R_{11}^{(0)})^{*} \cdot R_{12}^{(0)}$ ($i=2, j=2, k=1$)

$= (0+1+\varepsilon) + \phi \cdot (0+\varepsilon)^{*} \cdot 1$

$\boxed{R_{22}^{(1)} = (0+1+\varepsilon)}$

When $k=2$,

(*) $R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)} \cdot (R_{22}^{(1)})^{*} \cdot R_{21}^{(1)}$ ($i=1, j=1, k=2$)

$= 0^{*} + [0^{*}1 \cdot (0+1+\varepsilon)^{*} \cdot \phi]$

$= 0^{*} + \phi$

$\boxed{R_{11}^{(2)} = 0^{*}}$

(*) $R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} \cdot (R_{22}^{(1)})^{*} \cdot R_{22}^{(1)}$ ($i=1, j=2, k=2$)

$= 0^{*}1 + 0^{*}1 \cdot (0+1+\varepsilon)^{*} \cdot (0+1+\varepsilon)$

$= 0^{*}1 + 0^{*}1 \cdot (0+1+\varepsilon)^{+}$

$= 0^{*}1 + 0^{*}1 \cdot (0+1)^{*}$

$= 0^{*}1\,[\varepsilon + (0+1)^{*}]$

$\boxed{R_{12}^{(2)} = 0^{*}1 \cdot (0+1)^{*}}$

(*) $R_{21}^{(2)} = R_{21}^{(1)} + R_{22}^{(1)} \cdot (R_{22}^{(1)})^{*} \cdot R_{21}^{(1)}$ ($i=2, j=1, k=2$)

$= \phi + (0+1+\varepsilon)(0+1+\varepsilon)^{*} \cdot \phi$

$\boxed{R_{21}^{(2)} = \phi}$

(*) $R_{22}^{(2)} = R_{22}^{(1)} + R_{22}^{(1)} \cdot (R_{22}^{(1)})^* \cdot R_{22}^{(1)}$

$i=2, j=2, k=2$

$= (0+1+\varepsilon) + (0+1+\varepsilon) \cdot (0+1+\varepsilon)^* \cdot (0+1+\varepsilon)$

$= (0+1+\varepsilon) + (0+1)^*$

$\boxed{R_{22}^{(2)} = (0+1)^*}$

The number of states present in the given automata is 2. So, the maximum value of K will be 2.

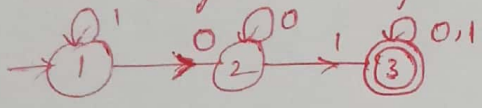
(*) The start state of automata is 1 and final state is 2, so the regular expression is given by

$\boxed{R_{12}^{(2)} = 0^*1(0+1)^*}$

and total number of states in the automata is 2 so, (k=2)

Exercise:

Obtain the regular expression for the below finite automata



Hint:

$R.E \Rightarrow R_{13}^{(3)} = R_{13}^{(2)} + R_{13}^{(2)} \cdot (R_{33}^{(2)})^* \cdot R_{33}^{(2)}$

value of $R_{13}^{(3)}$ is the regular expression.

Properties of Regular languages

The regular languages exhibit two types of properties:

(i) closure properties
(ii) Decision properties

1. Closure properties: Some new languages can be constructed from already existing languages using certain operations such as union, intersection, concatenation and so on. We can build automata for these new languages using some of the properties of regular languages. These properties are called closure properties.

Example: The intersection of 2 regular languages is regular. Thus, given 2 finite automata that recognized two different languages, we can construct an automaton that recognizes the intersection of these 2 languages. The automaton thus obtained may have more states than either of the 2 automata. Thus using the closure properties we can build more complex automata.

The closure properties of regular languages are as follows:

→ Union of two regular languages is regular
→ Intersection of two regular languages is regular
→ Complement of a regular language is regular
→ Closure (star) of a regular language is regular
→ The concatenation of two regular languages is regular
→ Difference of two regular languages is regular.
→ Reversal of a regular language is regular
→ A homomorphism of a regular language is regular
→ The inverse homomorphism of a regular language is regular.
→ Cross product

1. Closure under Union

Theorem: If $L1$ and $L2$ are 2 regular languages then $L_1 \cup L_2$ is regular.

Proof: If $L1$ and $L2$ are regular then, they have regular expressions $L1 = L(R1)$ and $L2 = L(R2)$. Then, $L_1 \cup L_2 = L(R_1 + R_2)$

Thus, we get $L_1 \cup L_2$ as regular language.

Example: $L1 = \{a, a^3, a^5, \ldots\}$ and $L2 = \{a^2, a^4, a^6, \ldots\}$

$L1 \cup L2 = \{a, a^2, a^3, a^4, a^5, \ldots\}$

Regular expression = $a \cdot a^*$ which is regular.

2. Closure under Concatenation

Theorem: If $L1$ and $L2$ are regular languages, then $L_1 \cdot L_2$ is also a regular language.

Proof: If $L1$ and $L2$ are regular then, they have regular expressions $L1 = L(R1)$ and $L2 = L(R2)$. Then $L_1 \cdot L_2 = L(R_1 \cdot R_2)$

Thus, we get $L_1 \cdot L_2$ as regular language.

Example:

Let $L_1 = \{ab^n \mid n \geq 0\}$ and $L_2 = \{b^n a \mid n \geq 0\}$

$L_1 \cdot L_2 = ab^n \cdot b^n a$

Regular expression $= ab^n a$ (starting with a and ending with a) which is regular.

3. Closure (star) of a regular language is regular.

Theorem: If L_1 is a regular language, then L_1^* is regular.

Proof: If L_1 is regular, then there exist a regular expression $L_1 = L(R_1)$. Then $L_1^* = L(R_1)^*$.

Thus, we get L_1^* as regular language.

Example: Let $L_1 = \{(a+b)\}$

then, $L_1^* = (a+b)^*$ is also regular.

4. Closure under Complementation:

Theorem: If L is regular language, then complementation of L denoted by $\bar{L}$ is also regular. In other words, the set of regular languages are closed under complementation.

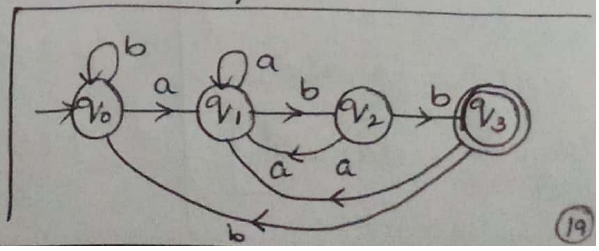
Proof: Let $M_1 = (Q, \Sigma, \delta, q_0, F)$ be a DFA which accepts the language L. Since, the language is accepted by a DFA, the language is regular.

Now, define the machine $M_2 = (Q, \Sigma, \delta, q_0, Q-F)$ which accepts $\bar{L}$. Note that there is no difference between M_1 and M_2 except the final states. The non-final states of M_1 are the final states of M_2 and final states of M_1 are the non-final states of M_2 and vice versa.

Thus, we have a machine M_2 which accepts all those strings denoted by $\bar{L}$ that are rejected by machine M_1.

So, a regular language is closed under complementation.

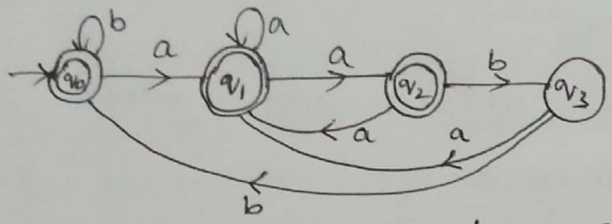
Example: The DFA to accept strings of a's and b's ending with abb is as shown.



According to the theorem, the regular languages are closed under complement. So, the complement of the given language is "strings of a's and b's that do not end with abb". By using the closure property of complement, the DFA can be obtained by following these steps:

* Making the final states in above DFA as non final states in new DFA.
* Making non-final states in above DFA as final states in new DFA.

The resulting DFA will accept the strings of a's and b's that do not end with abb, which is as shown below:



So, if L is a regular language over alphabet Σ, then $\overline{L} = \Sigma^* - L$ is also a regular language.

5. Closure under intersection:

Theorem: If $L1$ and $L2$ are regular languages, then they are closed under intersection.

Proof: Consider the language L_1 is regular. That means there exists some DFA M_1 that accepts $L1$. We can write $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$. Similarly being L_2 regular there is another DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Let L be the language obtained from $L_1 \cap L_2$. We can simulate $M = (Q, \Sigma, \delta, q, F)$

where, $Q = Q_1 \cap Q_2$

$\delta = \delta_1 \cap \delta_2$ a mapping function derived from both the DFA's.

$q \in Q$ which is initial state of machine M.

$F = F_1 \cap F_2$, the set of final states which is common for M_1 and M_2 machines.

Example:

$L_1 = \{a, a^2, a^3, a^4, a^5, a^6, \dots\}$

$L_2 = \{a^2, a^4, a^6, \dots\}$

$L_1 \cap L_2 = \{a^2, a^4, a^6, \dots\}$

Regular expression $= aa \cdot (aa)^*$ which is regular.

It is clear that there exists some DFA which accepts $L_1 \cap L_2$ i.e., L. Hence L is a regular language. This proves that if L_1 and L_2 are two regular languages then $L_1 \cap L_2$ is regular. In other words, the regular language is closed under intersection.

6. The reversal of a regular language is regular.

Theorem: If L is regular, then L^R is also regular.

Proof: Let L be the language corresponding to regular expression E. It is required to prove that there is another regular expression E^R such that $L(E^R) = (L(E))^R$. (which can be read as "language of E^R is the reversal of language of E")

Basic: By definition of regular expression E, we have:

* ϕ is a regular expression
* ε is a regular expression
* a is a regular expression

So, the reversal of regular expression E^R is given by

* $\{\phi\}^R = \phi$
* $\{\varepsilon\}^R = \{\varepsilon\}$
* $\{a\}^R = \{a\}$

Induction: Again, by definition of regular expressions, if E_1 and E_2 are regular expressions, then:

* $E_1 + E_2$ is a regular expression
* $E_1 \cdot E_2$ is a regular expression
* E_1^* is a regular expression

which results in three different cases. Now, let us prove each of these cases:

Case 1: $E = E_1 + E_2$. If $E = E_1 + E_2$ is a regular expression, then $E^R = E_1^R + E_2^R$ is a regular expression denoting the languages: $L(E^R) = L(E_1^R) \cup L(E_2^R)$

Case 2: $E = E_1 \cdot E_2$. If $E = E_1 \cdot E_2$ is a regular expression, then $E^R = E_1^R \cdot E_2^R$ is a regular expression denoting the languages: $L(E^R) = L(E_1^R) \cdot L(E_2^R)$

Case 3: $E = E_1^*$.

If $E = E_1^*$ is a regular expression, then

$E^R = E_1^R$ is a regular expression denoting the language,

$L(E^R) = L(E_1^R)$

Example: Let us consider a language L,

$L = \{001, 10, 11, 01\}$

$L^R = \{100, 01, 11, 10\}$

To prove that regular languages are closed under reversal.

$L = \{001, 10, 11, 01\}$ is a regular language over $\{0, 1\}$.

$L^R = \{100, 01, 11, 10\}$ is a language consisting of the reversals of the strings of L.

i.e., $L^R = \{100, 01, 11, 10\}$ which is also regular.

So, if L is regular then, L^R is also regular.

7. Closed under difference:

Theorem: If L_1 and L_2 are regular then the difference $L_1 - L_2$ is also regular.

Proof: Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ be the finite automata accepting the regular languages L_1 and L_2 respectively then, we say $L_1 - L_2$ is regular if and only if M_1 is accepted and M_2 is not accepted i.e., if (p, q) is the final state of the machine accepting $L_1 - L_2$ then $p \in F_1$ and $q \notin F_2$.

Example: $L_1 = \{a, a^2, a^3, a^4, a^5, \dots\}$

$L_2 = \{a^2, a^4, a^6, \dots\}$

$L_1 - L_2 = \{a, a^3, a^5, \dots\}$ which is also regular.

Regular expression = $a(aa)^*$

8. Closed under homomorphism:

Definition: Homomorphism

Let Σ and Γ are the set of alphabets. The homomorphic function $h: \Sigma \to \Gamma^*$ is called homomorphism i.e., a substitution where a single letter is replaced by a string.

If $w = a_1 a_2 a_3 \ldots\ldots a_n$, then $h(w) = h(a_1)h(a_2)\ldots h(a_n)$

If L is made of alphabets from Σ, then

$$h(L) = \{h(w) \mid w \in L\}$$

is called homomorphic image.

Example: ① Let $\Sigma = \{0,1\}$, $\Gamma = \{0,1,2\}$ and $h(0) = 01$, $h(1) = 112$. What is $h(010)$? If $L = \{00, 010\}$ what is homomorphic image of L?

By definition, we have

$$h(w) = h(a_1)h(a_2)\ldots\ldots h(a_n)$$

So, $h(010) = h(0).h(1).h(0)$

$= 0111201$

$L\{00,010\} = L(h(00), h(010))$

$= L(h(0).h(0), h(0)h(1)h(0))$

$= L(0101, 0111201)$

Therefore, $h(010) = 0111201$

$L(00, 010) = L(0101, 0111201)$ is the homomorphic image of L.

② If $h(0) = (00)$ and $h(1) = (101)$ then, find the $(0+1)^*(00)^*$

Soln: $(0+1)^* = (00 + 101)^*$

$(00)^* = (0000)^*$

$(0+1)^*(00)^* = (00+101)^*.(0000)^*$

Theorem: If L is a regular language and h is homomorphic function, then homomorphic image $h(L)$ is regular.

Proof: Let R be the regular expression and $L(R)$ be the corresponding regular language. We can easily find $h(R)$ by substituting $h(a)$ for each a in Σ. By definition of regular expression, $h(R)$ is a regular expression and so $h(L)$ is regular language. So, the regular language is closed under homomorphism.

9. Closed under inverse homomorphism

Theorem: If h is a homomorphism from alphabet S to alphabet T and L is a regular language over T, then $h^{-1}(L)$ is also a regular language.

Proof: Let h be a homomorphism and L be a language whose alphabet is the output language of h.

$$h^{-1}(L) = \{w \mid h(w) \text{ is in } L\}.$$

Example: Let L be the language of regular expression $(00+1)^*$ Let h be the homomorphism defined by $h(a) = 01$ and $h(b) = 10$. Then $h^{-1}(L)$ is the language of regular expression $(ba)^*$

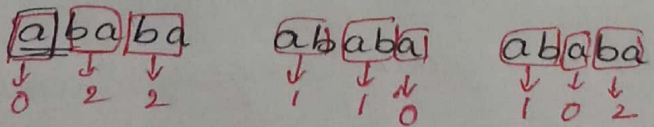
A homomorphism may be applied backwards i.e., given the $h(L)$, determine what L is. This is denoted as $h^{-1}(L)$, or the inverse homomorphism of L, which results in the set of strings w in Σ^* such that $h(w)$ is in L.

*Note that while applying the homomorphism to a string resulted in a single string, we may get a set of strings in the inverse.

Example: If h is the homomorphism from the alphabet $\{0,1,2\}$ to the alphabet $\{a,b\}$ defined by $h(0) = a$, $h(1) = ab$, $h(2) = ba$. Given L is the language $\{ababa\}$ what is $h^{-1}(L)$?

Soln: We can construct three strings that map into ababa:

$$h^{-1} = \{022, 110, 102\}$$

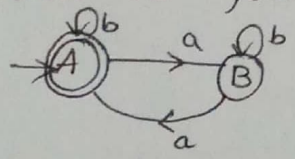
where, 

The result of h^{-1} is also a regular language.

10. Cross product:

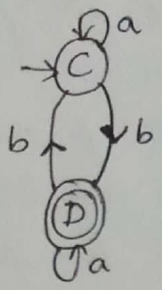
Example: Let $L_1 = \{\varepsilon, a^2, a^4, \cdots\}$ and $L_2 = \{b, b^3, b^5, b^7, \cdots\}$ where L_1 accepts even number of a's and any number of b's. L_2 accepts odd number of b's and any number of a's.

Let M_1 be the machine accepting L_1, which can be written as follows:



M_1

Let M_2 be the machine accepting L_2, which can be written as shown below,



$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$,

Consider $M = (Q, \Sigma, \delta, q_0, F)$ defined as follows

* $Q = Q_1 \times Q_2$
* $q_0 = \{q_1, q_2\}$
* $\delta([p_1, p_2), a] = (\delta_1(p_1, a), \delta_2(p_2, a))$
* $F = F_1 \times F_2$

From the above example,

$Q_1 = \{A, B\}$, $Q_2 = \{C, D\}$

$Q = Q_1 \times Q_2 = \{(A,C), (A,D), (B,C), (B,D)\}$

Consider the transitions.

① $\delta((A,C), a) = \delta(A, a), \delta(C, a)$
$= (B, C)$

$\delta((A,C), b) = \delta(A, b), \delta(C, b)$
$= (A, D)$

② $\delta((A,D), a) = \delta(A, a), \delta(D, a)$
$= (B, D)$

$\delta((A,D), b) = \delta(A, b), \delta(D, b)$
$= (A, C)$

③ $\delta((B,C), a) = \delta(B, a), \delta(C, a)$
$= (A, C)$

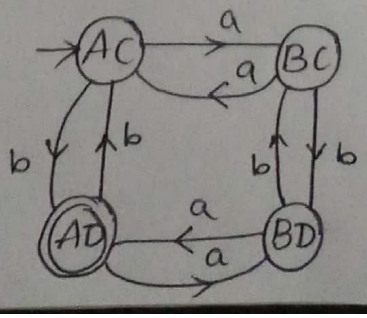
$\delta((B,C), b) = \delta(B, b), \delta(C, b)$
$= (B, D)$

④ $\delta((B,D), a) = \delta(B, a), \delta(D, a)$
$= (A, D)$

$\delta((B,D), b) = \delta(B, b), \delta(D, b)$
$= (B, C)$

The new states of machine, M are $\{AC, AD, BC, BD\} \rightarrow Q$

The DFA is given by,



This is also regular.

Applications of regular expressions

1. Regular expressions in UNIX:

Regular expressions are extensively used in UNIX operating system. But, certain short hand notations are used in UNIX platform using which complex regular expressions are avoided. For example grep (global regular expression print) used to search for a pattern of string.

2. Pattern matching:

Pattern matching refers to a set of objects with some common properties. We can match an identifier or a decimal number or we can search for string in the text.

3. Lexical analysis:

Regular expressions are extensively used in the design of lexical analyzer phase (this is the first phase of the compiler design). This phase scans the source program and recognizes all the tokens which are logically together.

To identify an identifier and an integer:

An identifier starts with a letter. This letter can be followed by combination of zero or more letters and digits i.e., an identifier can be a single letter followed by strings of letters and digits of any length and can be represented as

letter (letter + digit)*.

An integer can start with any of the sign +, − or ε (null string mean no sign) followed by one or more digits ranging from 0 to 9. This can be represented using a regular expression as Sd^+ or Sdd^* where S stands for sign and d stands for the digits from 0 to 9.

Identities of regular expression

1. $\phi + R = R + \phi = R$
2. $\phi R = R\phi = \phi$
3. $\varepsilon . R = R . \varepsilon = R$
4. $\varepsilon^* = \varepsilon$ and $\phi^* = \varepsilon$ → [closure of ϕ is ε]
   [closure of ε is ε itself]
5. $R + R = R$
6. $R^* . R^* = R^*$
7. $R . R^* = R^* . R = R^+$
8. $(R^*)^* = R^*$
9. $(\varepsilon + RR^*) = (\varepsilon + R^*R) = R^*$
10. $R^* . R + R = R^*R$
11. $\varepsilon + R^* = R^*$
12. $(R + \varepsilon)^* = R^*$
13. $R^* . (\varepsilon + R) = (\varepsilon + R) . R^* = R^*$
14. $(PQ)^* P = P(QP)^*$
15. $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$
    $= (P^* + Q)^* = (P + Q^*)^*$
    $= P^* . (Q . P^*)^* = Q^* . (P . Q^*)^*$
16. $(P + Q)R = PR + QR$
    and
    $R(P + Q) = RP + RQ$

Equivalence of two regular expressions

1. Prove that $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$ is equal to $0^*1(0+10^*1)^*$

Soln:

Consider, LHS part

$$LHS = (1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$$

Take $(1+00^*1)$ as common

$$=(1+00^*1)[\varepsilon+(0+10^*1)^*(0+10^*1)]$$

we have, $\varepsilon + R^*R = R^*$, so the above equation becomes

$$=(1+00^*1)(0+10^*1)^*$$

Observe the first term, $(1+00^*1)$, we can take '1' as common, but always the string is ending with '1' so, write '1' right side as shown below.

$$=(\varepsilon+00^*)\cdot 1\ (0+10^*1)^*$$

[WKT $\varepsilon + RR^* = R^*$]

$$= 0^*1\ (0+10^*1)^*$$

$$= RHS$$

Hence proved.

2. Show that $(0^*1^*)^* = (0+1)^*$

Soln:

Consider LHS part,

$$LHS = (0^*1^*)^*$$

WKT $(P+Q)^* = (P^*Q^*)^*$

where, $P=0$, $Q=1$

$$=(0^*1^*)^*$$

$$=(0+1)^*$$

$$=RHS$$

Hence proved.

(or)

If we write the language,

$L = \{\varepsilon, 0, 1, 01, 10, 010, 110, \ldots\}$

↓ the above language contains any string with 0's and 1's including ε,

Hence, $(0^*1^*)^* = (0+1)^*$

both accepts the above language

Hence, LHS = RHS

3. Show that $r^+ = r^* r^+$

Soln: Consider LHS,

$LHS = r^+$

WKT $r^+ = r^* r$, so substitute this value

$= r^* r$

$= r^* r^* . r$ WKT $R^* . R^* = R^*$

$= r^* . r^+$ (or) $R^* = R^* . R^*$

$= RHS$ $r^* r$ can be written as r^+

Hence proved

(or)

Consider, LHS i.e., r^+

the language is $L = \{r, rr, rrr, \ldots\ldots\}$

Consider, RHS i.e., $r^* . r^+$

the language is $L = \{r, rr, rrr, \ldots\ldots\}$

Both are accepting the same language.

Hence, proved LHS = RHS.

4. $(r+s)^* = (r+s^*)^*$

Soln: Consider LHS, $(r+s)^*$

$LHS = (r+s)^*$ WKT $(P+Q)^* = (P+Q^*)^*$

$= (r+s^*)^*$

$= RHS$

Hence proved.

(or)

Write the language for both LHS and RHS

$LHS = \{\varepsilon, r, s, rss, ssr, sr, rs, \ldots\ldots\}$

$RHS = \{\varepsilon, r, s, sr, rs, rss, ssr, \ldots\ldots\}$

Both have same strings. Hence LHS = RHS.

5. Show that $(r+s)^* \neq r^* + s^*$

Soln: Consider LHS,

$LHS = (r+s)^*$

Write the language, $L_1 = \{\varepsilon, r, s, rs, sr, rrsr, srsr, \ldots\}$

Consider $RHS = r^* + s^*$

Write the language, $L_2 = \{\varepsilon, r, s, rr, ss, rrr, sss, \ldots\}$

L_1 contains any combination of r and s including ε.

L_2 contains any number of r's or any number of s including ε. But there is no combination of r and s.

So, $LHS \neq RHS$

hence proved.

6. Prove that $r(s+t) = rs + rt$

Soln: We have an identity,

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

Consider $LHS = r(s+t)$

This can be written as

$= rs + rt$

$= RHS$

Hence proved.

Limitation of finite automata.

There are few problems for which we can not construct a DFA and still we want some solution to solve those problems:

for example:

(i) check for the matching parenthesis.

(ii) Count number of a's and then the number of b's - which is not possible by DFA; so DFA can not be used as a counter.

Limitations

1. A finite automata has finite number of states and so it does not have the capacity to remember arbitrary large amount of information.
2. Since it does not have memory, finite automata can not remember a long string.
   Example: to check for matching parentheses, check whether the string is a palindrome or not and so on are not possible using finite automata.
3. Finite automata or finite state machines have trouble recognizing various types of languages involving counting, calculating, storing the string.

The solution for all these problems is to have a machine which is more general than DFA which can recognize these set of languages.

Example: using grammers, push docon automata, tusing machines we can solve all these problems.

Decision property of regular language:

Using this property, we can decide whether two automata define the same language. If so, we can minimize the state of automata with as few states as possible. ~~This~~ This minimization of automaton is very important in the design of switching circuits. This is because, as the number of states of automaton decreases the size of the circuit and hence the cost decreases.

* Regular as well as non-regular languages exists, both are important. The following are some of the non-regular languages.

1. $L = \{w : w \in \{a,b\}^*$ and has equal number of a's and b's$\}$
2. $L = \{0^n 1^n$ for $n \geq 0\}$
3. $L = \{a^p$ / where $p \geq 2$ is a prime number
4. Language consisting of machine parenthesis.

There are so many languages which are not regular. We can prove that certain languages are not regular using a powerful tool called pumping lemma.

Pumping lemma for regular languages

Note: The language for which it is possible to design the finite automata is definitely a regular language.

Theorem: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automata and has n number of states. Let L be the regular language accepted by M. For every string S in L, the exists a constant p, such that $|S| \geq p$. Now, if the string S can be broken into three substrings x, y and z such that

$S = xyz$ satisfying the following conditions:

(*) $|y| \neq \varepsilon$ i.e., $|y| \geq 1$

(*) $|xy| \leq p$

then $xy^i z$ is in L for $i \geq 0$

i.e., 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) $xy^i z \in A$ for every $i \geq 0$

(ii) $|y| > 0$ (or) $|y| \geq 1$

(iii) $|xy| \leq p$

To prove that a language is not regular using pumping lemma, follow these steps:

(We prove by 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 such that $|y| \geq 1$, $|xy| \leq P$.
* show that $xy^iz \notin A$ for some i
* Then consider different ways that S can be divided into xyz. Show that none of these can satisfy all the 3 conditions at the same time.

[Note: Select i such that the resulting string is not in L].

S, cannot be pumped.

Hence, we can say L is not regular.

Problems:

1. Prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is not regular.

Soln: Assume A is regular,

then it should have a pumping length, say P.

* choose a string, $S = a^P . b^P$

let us consider $P = 7$.

So, the string S becomes aaaaaaabbbbbbb.

* Divide the string S into 3 parts say xyz.

S = aaaaaaabbbbbbb

case (i): The 'y' part comes with in a^p.

aa|aaaa|abbbbbbb

x | y | z

check all the 3 conditions:

* $xy^iz \in A$ for $i \geq 0$

Let us consider $i = 2$, so S becomes.

aa aaaaaaaa abbbbbbb which $\notin A$

(because number of $a^p \neq$ number of b^p)

* $|y| \geq 1$; $|aaaa| = 4 \geq 1$
* $|xy| \leq P$; $|\overset{x}{aa}\overset{y}{aaa}a| = 6 \leq 7$

} 2 conditions are satisfied, but 1st condition is not satisfied. So, $a^n b^n$ is not regular.

Case (ii): The 'y' part comes with in b's.

aaaaaaabbbbbbb
x y z

consider $i=2$, S becomes,

$S = aaaaaaabbbbbbbbbbb \notin A$ (because number of a's ≠ number of b's)

So, A is not regular.

Case (iii): The 'y' part contains 'a' as well as 'b' symbols.

aaaaaaabbbbbbb
x y z

consider $i=2$, S becomes,

$S = aaaaaaaabbaabbbbbbb \notin A$

So, A is not regular.

---

2) Prove that the language $A = \{yy \mid y \in \{0,1\}^*\}$ is not regular.

Soln: Let us assume A is a regular language. Then, it has a pumping length say P.

* choose a string $S = 0^P1.0^P1$ (where $y = 0^P1$, $yy = 0^P1.0^P1$

let us consider $P=4$.

So, S becomes $0^41.0^41 \Rightarrow S = 000010000 1$

* Divide the string S into 3 parts, say xyz

S = 000|0|1|0|0001
x y z

* check the conditions:

(i) $xy^iz \Rightarrow$ consider $i=2$, S becomes

$S = 000010010 0001 \notin A \Rightarrow$ condition is not satisfied

(ii) $|y| \geq 1$; $|010| = 3 \geq 1$

(iii) $|xy| \leq P$; $|0000\overset{x}{1}0\overset{y}{1}0| = 6 \leq 4 \Rightarrow$ condition is not satisfied.

From condition (i), we can see that $S \notin A$
Hence $xy^iz \notin A$.

Since, 000010010000 1 is not present in A, we can say A is not regular.

3) Prove that the language $L = \{ a^n b^{2n} \mid n > 0 \}$ is not regular.

Soln: Let us assume L is regular, then it has a pumping length p. $L = \{abb, aabbbb, aaabbbbbb, aaaabbbbbbbb, \ldots\}$

* choose a string $S = a^p . b^{2p}$, let us consider $p = 3$.

S becomes, $a^3 . b^6 = aaabbbbbb$

* Divide S into 3 parts, xyz

$S = aa|a|bb|bbbb$
$\quad\quad x \quad y \quad z$

* check the conditions:

(i) $xy^iz \Rightarrow$ consider $i = 2$

$S = aa\,abb\,abbbbbb \notin L$

Hence, the given language is not regular.

---

4) Show that $L = \{a^p \mid p$ is prime$\}$ is not regular.
↳ 1, 2, 3, 5, 7, 11 etc.

Soln: L can have these set of strings,

$L = \{a, aa, aaa, aaaaa, aaaaaaa, \ldots\}$

Let us assume L is a regular language, which has a pumping length p.

* choose a string S, a^p consider $p = 5$

$S = aaaaa$ (or) a^5

* Divide the string into 3 parts

$S = a|a|aa|a$
$\quad\quad x \quad y \quad z$

* check the conditions:

(i) $xy^iz \Rightarrow$ consider $i = 2$

$xy^2z = a\,aaaaaaa \notin L$
$\quad\quad (a^8)$

Hence, the given language is not regular.

Exercise problems:

1) Show that $L = \{0^n \mid n$ is a perfect square$\}$ is not regular.
↳ 1, 4, 9, 16, 25, ... etc.

hint: $L = \{0, 0000, 00000000, \ldots\}$

2) Show that $L = \{a^i b^j \mid i > j\}$ is not regular

3) Show that $L = \{a^n b^l \mid \text{where } n \neq l\}$ is not regular.

4) Prove that $L = \{a^n b^l c^{n+l} \mid (n,l) \geq 0\}$ is not regular.

5) Prove that $L = \{0^n 1^m 2^n \mid (n,m) \geq 1\}$ is not regular

6) Show that $L = \{a^{n^2} \mid n \geq 0\}$ is not regular.

Summary of pumping lemma

Step 1: For any given language take an example string accepting the language.

Step 2: The length of the string S should be greater than or equal to P. Divide the ~~language~~ string into 3 substrings xyz such that (i) $xy^i z \in$ the given language.

(ii) $|y| \geq 1$ or $y \neq \varepsilon$

(iii) $|xy| \leq p$

Step 3: Take different values of $i = 0, 2, 3, 4, \ldots$ to prove that the given language is not regular.

Decision properties: ~~It gives algorithms for deciding whether~~ For any given finite automata, if the number of states can be reduced by using some methods so that the original property of the finite automata is not changed.

Equivalence of two states: The language generated by a DFA is unique. But, there can exist many DFA's that accept the same language. In such cases, the DFA's are said to be equivalent. During computations, it is desirable to represent the DFA with fewer states. Since the space is proportional to the number of states of DFA. For storage efficiency, it is required to reduce the number of states and hence it is necessary to minimize the DFA. This can be achieved by finding the distinguishable and indistinguishable states.

Defn: Two states p and q are said to be equivalent (indistinguishable) iff $\delta(p,w)$ and $\delta(q,w)$ are final states or both are non-final states for all $w \in \Sigma^*$ i.e., if $\delta(p,w) \in F$ and $\delta(q,w) \in F$ then the states p and q are indistinguishable (or) if $\delta(p,w) \notin F$ and $\delta(q,w) \notin F$ then (p, q) are equivalent.

Distinguishable states: If there is atleast one string w such that $\delta(p,w) \in F$ and $\delta(q,w) \notin F$ or vice versa, then p and q are not equivalent and are called distinguishable states.

Minimization of DFA:

2 Methods: ① Table filling method ② Myhill Nerode theorem
② Partition method.

The following are the steps to be followed:

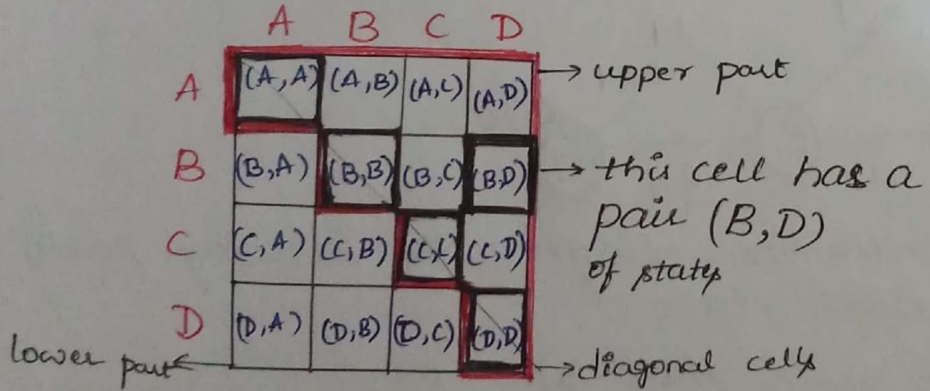
1. Draw a table for all pairs of states (P, Q)
2. Mark all pairs where $P \in F$ and $Q \notin F$ (final states set)
3. If there are any unmarked pairs (P, Q) such that $[\delta(P, x), \delta(Q, x)]$ is marked, then mark [P, Q] where 'x' is an input symbol.

   Repeat this until no more markings can be done.
4. Combine all the unmarked pairs and make them a single state in the minimized DFA.

How to draw a table for all pairs of states

Example: Consider, $Q = \{A, B, C, D\}$.

→ Now, we need to make pairs of states. Write all the states row-wise as well as column-wise as shown below.



→ If we observe the diagonal cells, only one state is used to form a pair i.e., (A, A), (B, B), (C, C) and (D, D). So, no need to consider these cells.

→ This table contains two cells for each pairs. Example: (A, B) and (B, A) both are same. But we will consider only one cell i.e., either (A, B) or (B, A).

→ So, we can divide the table diagonally and we can remove the upper part or lower part.

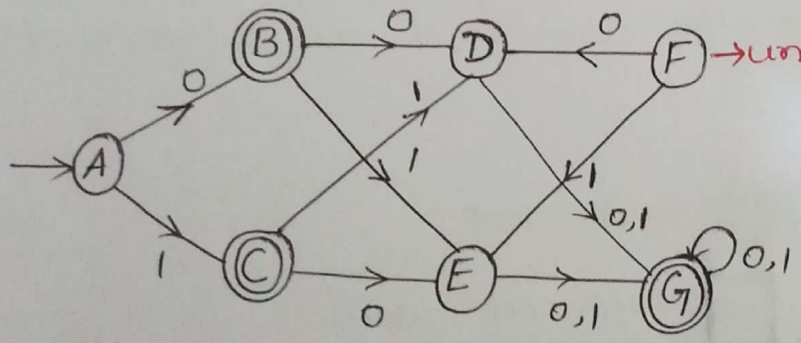
→ If the upper part is removed, then the table looks like this.

B	(B,A)		
C	(C,A)	(C,B)	
D	(D,A)	(D,B)	(D,C)
	A	B	C

Note: * If there is any unreachable states in the given finite automata, then it can be eliminated directly.

* Unreachable state: A state is said to be unreachable or not reachable if there is no path from start (initial) state to that state. And it has only outgoing edges but no incoming edges.

Example:



From the above transition diagram, the following states are reachable from start state.

(i) →A $\xrightarrow{0}$ B (we can reach state B)

(ii) →A $\xrightarrow{1}$ C (we can reach state C)

(iii) →A $\xrightarrow{0}$ B $\xrightarrow{0}$ D (we can reach state D)

(iv) →A $\xrightarrow{0}$ B $\xrightarrow{1}$ E (or) →A $\xrightarrow{1}$ C $\xrightarrow{0}$ E (we can reach state E)

(v) →A $\xrightarrow{0}$ B $\xrightarrow{1}$ E $\xrightarrow{0,1}$ G

(or) →A $\xrightarrow{1}$ C $\xrightarrow{0}$ E $\xrightarrow{0,1}$ G

(or) →A $\xrightarrow{0}$ B $\xrightarrow{0}$ D $\xrightarrow{0,1}$ G

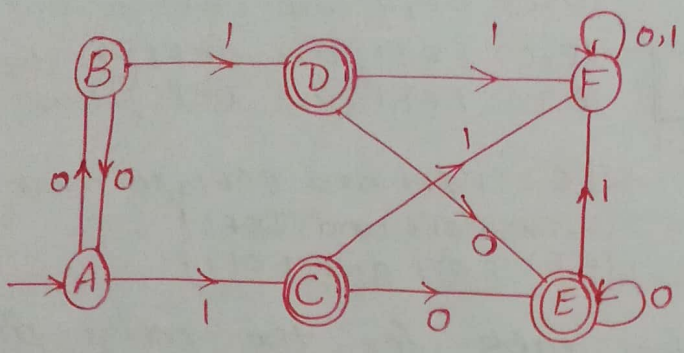
(or) →A $\xrightarrow{1}$ C $\xrightarrow{1}$ D $\xrightarrow{0,1}$ G

} (we can reach state G).

But there is no path from start state A to F state. so it is unreachable.

Problems:

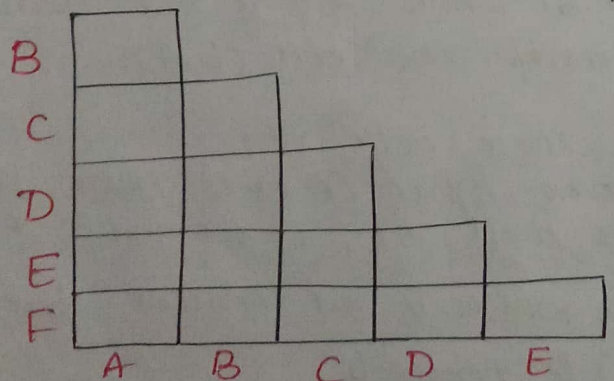
1) Minimize the given DFA using table filling method (or) Myhill Nerode theorem.



Soln: Let us write the transition table from the above transition diagram,

δ	0	1
→A	B	C
B	A	D
*C	E	F
*D	E	F
*E	E	F
F	F	F

Step 1: Draw a table for all pairs of states.

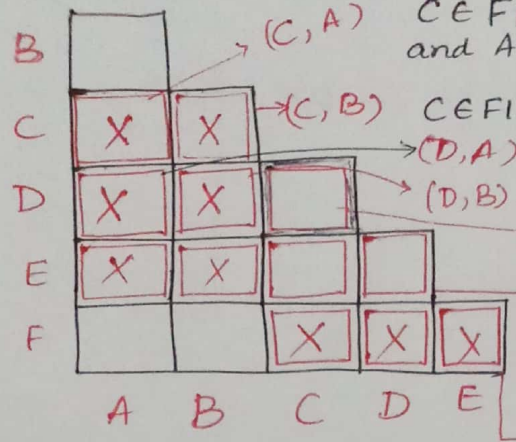


Step 2: Mark the pair (P,Q) such that $P \in F$ and $Q \notin F$ (or) $P \notin F$ and $Q \in F$.

Identify the final states, $F = \{C, D, E\}$.

Now, look at the rows and columns C, D, E; mark the cells such that $P \in F$ and $Q \notin F$ (or) $P \notin F$ and $Q \in F$, as shown.

Where $F1 = \{C, D, E\}$



(C, A) → C ∈ F1 and A ∉ F1 so you can mark this cell

(C, B) → C ∈ F1 and B ∉ F1, so mark this cell

(D, A): D ∈ F1 and A ∉ F1, so mark this cell

(D, B): D ∈ F1 and B ∉ F1, so mark this cell

(D, C): D ∈ F1 and C ∈ F1, so don't mark it.

(E, C): E ∈ F1 and C ∈ F1
(E, D): E ∈ F1 and D ∈ F1
} so don't mark these cells.

(F, C): F ∉ F1 and C ∈ F1
(F, D): F ∉ F1 and D ∈ F1
(F, E): F ∉ F1 and E ∈ F1
} so mark these cells.

Step 3: Now, write the transition table for the pairs in unmarked cells as follows:

Unmarked pairs	δ	0	1
(B, A)		(A, B)	(C, D)
(D, C)		(E, E)	(F, F)
(E, C)		(E, E)	(F, F)
(E, D)		(E, E)	(F, F)
(F, A)		(F, B)	(F, C)
(F, B)		(F, A)	(F, C)

δ(B, 0) = A
δ(A, 0) = B
} we will get the new pair (A, B)

Similary do it for other pairs also.

→ Now, we need to mark the remaining unmarked cells in such a way that, if $[\delta(p, x), \delta(q, x)]$ is marked, then mark (p, q) also.

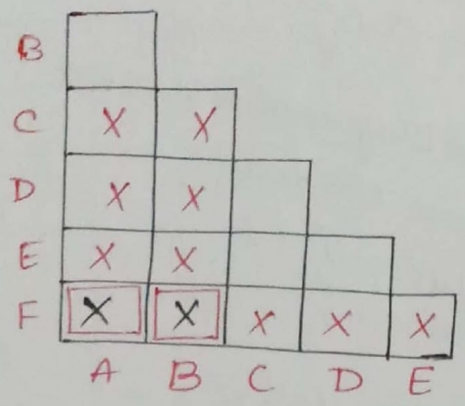
From the above table: consider the pairs

(i) (B, A): $[\delta(B, 0), \delta(A, 0)] = (A, B)$ where A, B is not marked so no need to mark the cell (B, A).

(ii) (D, C)
(E, C)
(E, D)
} No need to mark these cells. Because on seeing some input (0 or 1) these pairs (states) are going to same state i.e., (E, E) and (F, F)

(iii) (F, A): $[\delta(F, 0), \delta(A, 0)] = (F, B)$ which is not marked but $[\delta(F, 1), \delta(A, 1)] = (F, C)$ is marked.
So, mark the cell (F, A)

(iv) (F, B): $[\delta(F, 0), \delta(B, 0)] = (F, A)$ is not marked by $[\delta(F, 1), \delta(B, 1)] = (F, C)$ is marked.
So, mark the cell (F, B) as shown in the next table.



Repeat this until no more markings can be done. Write down the table for remaining unmarked cells.

δ	0	1	
(B,A)	(A,B)	(C,D)	→ (A,B) as well as (C,D) is not marked. So, don't mark (B,A)
(D,C)	(E,E)	(F,F)	same state. Don't mark the pairs (D,C), (E,C) and (E,D).
(E,C)	(E,E)	(F,F)	
(E,D)	(E,E)	(F,F)	

No more markings can be done.

Step 4: Combine all the unmarked pairs and make them a single state.

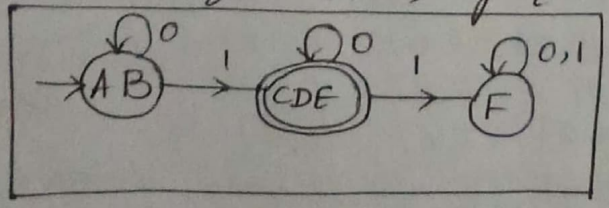
(i) (A, B) can be combined as a single state.
(ii) (D,C), (E,C) and (E,D) can be combined as single state. (∵ D and C are equivalent, E and C are equivalent, E and D are equivalent. Hence (C,D,E) is also equivalent.)
(iii) F can be taken as it is.

δ	0	1
→ (A,B)	(A,B)	(C,D,E)
* (C,D,E)	(C,D,E)	F
F	F	F

* The state A and B will be replaced by new state (A,B)
* Similarly C, D and E is replaced by (C,D,E) state.

↳ We have written this by seeing the original transition table.

Draw the minimized DFA, by seeing the above transition table.



Minimized DFA

2) Minimize the given DFA, using Myhill-Nerode theorem or Table filling method.

δ	0	1
→A	B	E
B	C	F
*C	D	H
D	E	H
E	F	I
*F	G	B
G	H	B
H	I	C
*I	A	E

For table filling method problems (in exam, you can follow these steps) (problem 2)

(write like this) ⇓

Soln:

Step 1: Draw a table for all pairs of states, and mark all the pairs where P∈F1 and Q∉F1 (or) P∉F and Q∈F.

$F1 = \{C, F, I\}$

Step 2:

B								
C	X	X						
D			X					
E			X					
F	X	X		X	X			
G			X			X		
H			X			X		
I	X	X		X	X		X	X
	A	B	C	D	E	F	G	H

Step 3: Note down, the unmarked cells as follows:

δ	0	1
(B,A)	(C,B)	(F,E)
(D,A)	(E,B)	(H,E)
(D,B)	(E,C)	(H,F)
(E,A)	(F,B)	(I,E)
(E,B)	(F,C)	(I,F)
(E,D)	(F,E)	(I,H)
(F,C)	(G,D)	(B,H)
(G,A)	(H,B)	(B,E)
(G,B)	(H,C)	(B,F)

δ	0	1
(G,D)	(H,E)	(B,H)
(G,E)	(H,F)	(B,I)
(H,A)	(I,B)	(C,E)
(H,B)	(I,C)	(C,F)
(H,D)	(I,E)	(C,H)
(H,E)	(I,F)	(C,I)
(H,G)	(I,H)	(C,B)
(I,C)	(A,D)	(E,H)
(I,F)	(A,G)	(E,B)

Now, mark the unmarked cells (P,Q) if $[\delta(p,x), \delta(q,x)]$ is marked.

Write the table once again and do the remaining markings.

B	X							
C	X	X						
D		X	X					
E	X		X	X				
F	X	X		X	X			
G		X	X		X	X		
H	X		X	X		X	X	
I	X	X		X	X		X	X
	A	B	C	D	E	F	G	H

Again, make a list of unmarked cells along with the transitions as shown below.

δ	0	1
(D,A)	(E,B)	(H,E)
(E,B)	(F,C)	(I,F)
(F,C)	(G,D)	(B,H)
(G,A)	(H,B)	(B,E)
(G,D)	(H,E)	(B,H)
(H,B)	(I,C)	(C,F)
(H,E)	(I,F)	(C,I)
(I,C)	(A,D)	(E,H)
(I,F)	(A,G)	(E,B)

Now, once again mark the unmarked cells (P, Q) if $[\delta(p,x), \delta(Q,x)]$ is marked.

But no more markings can be done.

So, combine the unmarked pairs and make them a single state in the minimized DFA.

In the above table, we have the sets

(D, A), (G, A), (G, D) - it can be combined as (A, D, G)

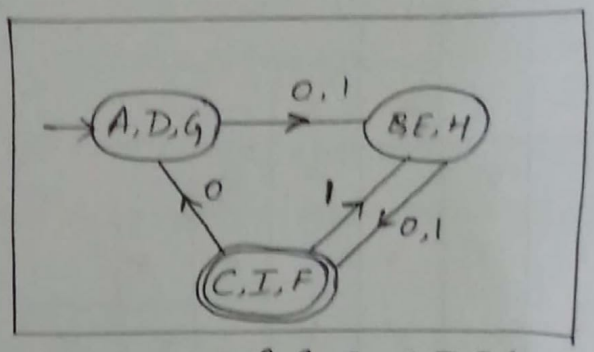
(E, B), (H, B), (H, E) - it can be combined as (B, E, H)

(F, C), (I, C), (I, F) - it can be combined as (C, I, F)

Here, the states (A, D, G), (B, E, H) and (C, I, F) all are distinguishable states and there are no more indistinguishable states.

The transition table for the minimized DFA can be written as follows:

δ	0	1
→(A, D, G)	(B, E, H)	(B, E, H)
(B, E, H)	(C, I, F)	(C, I, F)
* (C, I, F)	(A, D, G)	(B, E, H)



Minimized DFA

Exercise problems:

1) Minimize the following finite automata using table filling method.

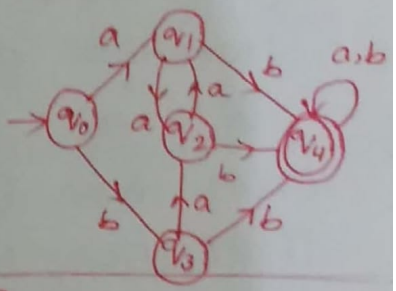
(a)

δ	a	b
→A	B	F
B	G	C
* C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

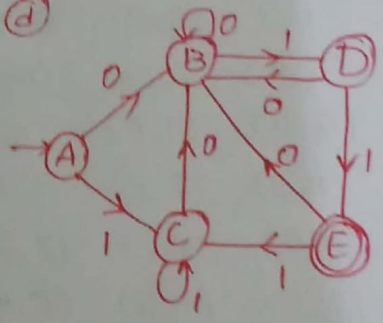
(b)

δ	0	1
→A	B	A
B	A	C
C	D	B
* D	D	A
E	D	F
F	G	E
G	F	G
H	G	D

(c)



(d)



Partition method

The following are the steps to be followed:

Step 1: All the states of δ are divided into 2 partitions - final states as one partition and non-final states as another partition, which is denoted by P_0. All the states in the partition P_0 are 0-(zero) equivalence. Take a counter n and initialize it with 0 (zero).

Step 2: Increment n by 1. For each partition P_n, divide the states in P_n into partitions if they are n-distinguishable. Two states within this partition X and Y are n-distinguishable if there is an input δ such that δ(X, S) and δ(Y, S) are (n-1) distinguishable.

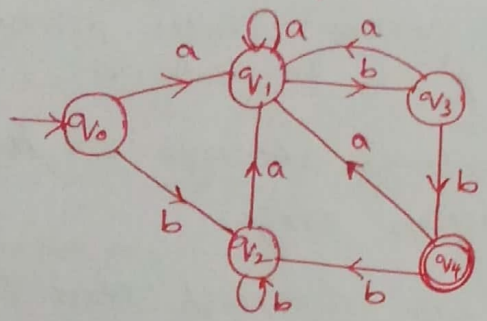
Step 3: If $P_n \neq P_{n-1}$, repeat step 2, otherwise go to step 4.

Step 4: Combine n^{th} equivalent sets (P_n) and make them the new states of the reduced DFA.

Note: In both table filling method as well as partition method we can eliminate the not reachable (or) unreachable states.

Problems:

1) Minimize the given DFA using partition method.



δ	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
$*\ q_4$	q_1	q_2

Soln: Find out the partition P_0,

(i) Zero - equivalence:

Make the non-final states as one partition and final states as another partition.

$P_0 = \{ \{q_0, q_1, q_2, q_3\}, \{q_4\} \}$

$\{q_0, q_1, q_2, q_3\}$ ↓ non-final states; $\{q_4\}$ ↓ final state

(ii) Now find out the partition P_1,

One - equivalence:

Consider P_0, $\{q_4\}$ - there is only one state in this partition so, no need to find equivalence.

$\{q_0, q_1, q_2, q_3\}$ - check equivalence among the states present in this partition.

* Let us consider the pair (q_0, q_1)

$\delta(q_0, a) = q_1$
$\delta(q_1, a) = q_1$
} same state

(or)

$\delta(\{q_0, q_1\}, a) = \{q_1, q_1\}$

$\delta(q_0, b) = q_2$
$\delta(q_1, b) = q_3$

(or)

$\delta(\{q_0, q_1\}, b) = \{q_2, q_3\}$

Consider P_0, $\{q_2, q_3\}$ comes under same partition i.e., $\{q_0, q_1, q_2, q_3\}$ so

they are equivalent. Hence, in P_1 it both the states will come under same partition $P_1 = \{\{q_0, q_1, \}, \{q_4\}\}$

* Now, let us consider q_2 state. If q_2 is equivalent with q_1 then it will be equivalent with q_0 also. (because already we have seen that $\{q_0, q_1\}$ are equivalent).

So, we can find equivalence between $\{q_0, q_2\}$ or $\{q_1, q_2\}$

$\delta(\{q_1, q_2\}, a) = \{q_1, q_1\} = q_1$ → same state so it may be equivalent. check with another transition also.

$\delta(\{q_1, q_2\}, b) = \{q_3, q_2\}$

↳ again $\{q_3, q_2\}$ comes under same partition of P_0 i.e., $\{q_0, q_1, q_2, q_3\}$

Note: While checking the equivalence, always we have to consider the (above) previous partition only.

$\{q_1, q_2\}$ are equivalent, so we can say that $\{q_0, q_2\}$ is also equivalent. Hence, in P_1 all these 3 states comes under same partition as shown below.

$$P_1 = \{\{q_0, q_1, q_2, \}, \{q_4\}\}$$

* Now, consider q_3 and for equivalence either with q_0, q_1 or q_2. (because we have seen that $\{q_0, q_1, q_2\}$ are equivalent). If q_3 is equivalent with any one of these states then it will be equivalent with the remaining states.

Consider the pair, $\{q_2, q_3\}$

$\delta(\{q_2, q_3\}, a) = \{q_1, q_1\} = q_1$ - same state

$\delta(\{q_2, q_3\}, b) = \{q_2, q_4\}$ - Observe P_0, q_2 and q_4 are in separate partitions. So P_1 becomes.

$$P_1 = \{\{q_0, q_1, q_2\}, \{q_3\}, \{q_4\}\}$$

q_2 is in $\{q_0, q_1, q_2, q_3\}$ and q_4 is in $\{q_4\}$

(iii) Now, find out two-equivalence or P_2:

Consider P_1, $\{q_3\}, \{q_4\}$ - single state, no need to find equivalence.

$\{q_0, q_1, q_2\}$ - find the equivalence among these states.

(*) $\{q_0, q_1\} \Rightarrow \delta(\{q_0, q_1\}, a) = \{q_1, q_1\} = q_1$

$\delta(\{q_0, q_1\}, b) = \{q_2, q_3\}$ ↓

in the partition P_1, q_2 and q_3 comes under separate group. So, $\{q_0, q_1\}$ are not equivalent. P_2 becomes

$$P_2 = \{\{q_0 \quad \{q_1\}, \{q_3\}, \{q_4\}\}$$

(*) $\{q_0, q_2\} \Rightarrow \delta(\{q_0, q_2\}, a) = \{q_1, q_1\} = q_1$

$\delta(\{q_0, q_2\}, b) = \{q_2, q_2\} = q_2$ } both are going to same state.

So, $\{q_0, q_2\}$ are equivalent.

Hence, P_2 becomes

$$P_2 = \{\{q_0, q_2\}, \{q_1\}, \{q_3\}, \{q_4\}\}$$

(iv) Now, find out three - equivalence (3-equivalence / P_3):

Consider P_2, $\{q_1\}, \{q_3\}, \{q_4\}$ are single states - no need to find out equivalence.

check with pair $\{q_0, q_2\}$

$\delta(\{q_0, q_2\}, a) = \{q_1, q_1\} = q_1$

$\delta(\{q_0, q_2\}, b) = \{q_2, q_2\} = q_2$ } they are going to same state - equivalent

Hence, P_3 is

$$P_3 = \{\{q_0, q_2\}, \{q_1\}, \{q_3\}, \{q_4\}\}$$

If we observe P_3 and P_2, both are same $\left[\frac{P_n = P_{n-1}}{P_3 = P_2}\right]$

So, we can stop over here.

* Now, combine the states present in the partition P_3.

$\{q_0, q_2\} \rightarrow$ 1 state

$\{q_1\} \rightarrow$ 1 state

$\{q_3\} \rightarrow$ 1 state

$\{q_4\} \rightarrow$ 1 state

Write the transition table for the minimized DFA.

δ	a	b
→ {q_0, q_2}	{q_1}	{q_0, q_2}
{q_1}	{q_1}	{q_3}
{q_3}	{q_1}	{q_4}
* {q_4}	{q_1}	{q_0, q_2}

Use the original transition table to get these transitions.

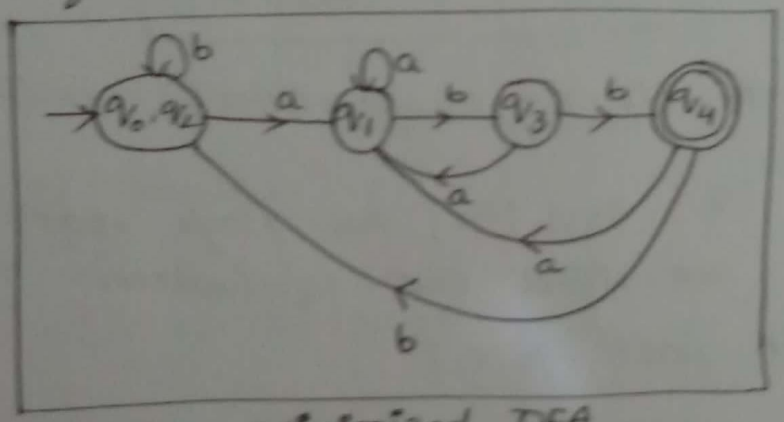
$\delta(q_0, a) = q_1$, new state is also $\{q_1\}$
$\delta(q_0, b) = q_2$, but new state is $\{q_0, q_2\}$

① Start state of original DFA is q_0, but here q_0 is combined with q_2 to form a new state. So, make $\{q_0, q_2\}$ as start state of minimized DFA.

② q_4 is the final state of original DFA, so $\{q_4\}$ is the final state of reduced DFA

③ As we have combined both q_0 and q_2. In the transition table, where ever q_0 or q_2 comes, it has to be replaced by $\{q_0, q_2\}$

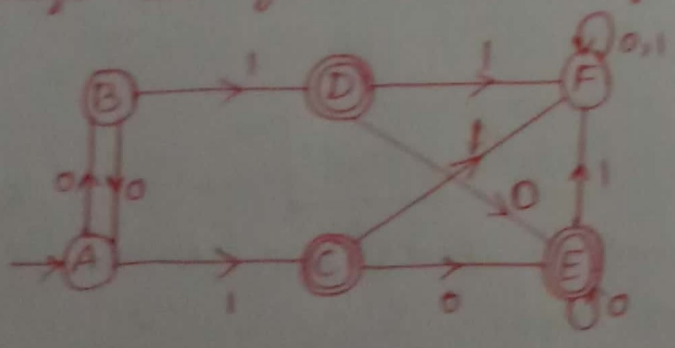
→ The minimized DFA is as follows:



Minimized DFA

(When there are more than one final states)

2) Minimize the given DFA using partition method.



δ	0	1
→ A	B	C
B	A	D
*C	E	F
*D	E	F
*E	E	F
F	F	F

Soln:

(i) 0-equivalence (zero equivalence / P_0):

$P_0 = \{\{A, B, F\}, \{C, D, E\}\}$ → set of final states

↓ set of non-final states

(ii) One-equivalence: P_1

$P_0 = \{A, B, F\}$ - check equivalence among the states

* $\{A,B\} \Rightarrow \delta(\{A,B\}, 0) = \{B,A\} \rightarrow$ B and A are in same partition i.e., $\{A,B,F\}$

$\delta(\{A,B\}, 1) = \{C,D\}$

↳ C and D are in same partition $\{C,D,E\}$

So, $\{A,B\}$ are equivalent.

$P_1 = \{\{A,B\}, \{C,D,E\}\}$

* Check with $\{B,F\}$ & $\{A,F\}$

$\{B,F\} \Rightarrow \delta(\{B,F\},0) = \{A,F\} \rightarrow$ A and F are in same partition i.e., $\{A,B,F\}$ of P_0.

$\delta(\{B,F\}, 1) = \{D,F\}$

↳ where are D and F are in separate partitions.

D comes under $\{C,D,E\}$

F comes under $\{A,B,F\}$

So, F is neither equivalent with B nor A.

$P_1 = \{\{A,B\}, \{F\}, \{C,D,E\}\}$

Now, let us consider $\{C,D,E\}$

* $\{C,D\} \Rightarrow \delta(\{C,D\}, 0) = \{E,E\} \rightarrow$ same state - present in partition $\{C,D,E\}$

$\delta(\{C,D\}, 1) = \{F,F\} \rightarrow$ same state

↳ present in partition $\{A,B,F\}$

So, $\{C,D\}$ are equivalent.

$P_1 = \{\{A,B\}, \{F\}, \{C,D\}\}$

Check for E, $\{D,E\} \Rightarrow \delta(\{D,E\}, 0) = \{E,E\} =$ E present in $\{C,D,E\}$

$\delta(\{D,E\}, 1) = \{F,F\} =$ F present in $\{A,B,F\}$ of P_0.

E is equivalent with D and so it is equivalent with C.

$P_1 = \{\{A,B\}, \{F\}, \{C,D,E\}\}$

(iii) two-equivalence / 2-equivalena / P_2:

Consider P_1, $\{\{A,B\}, \{F\}, \{C,D,E\}\}$

* $\{A,B\} \Rightarrow \delta(\{A,B\}, 0) = \{B,A\} \rightarrow$ present in partition $\{A,B\}$ of P_1.

$\delta(\{A,B\}, 1) = \{C,D\} \rightarrow$ comes under partition $\{C,D,E\}$ of P_1.

{A, B} are equivalent.

Check with {C, D, E}:

* {C, D} ⇒ δ({C, D}, 0) = {E, E} = E → comes under {C, D, E} of P_1

δ({C, D}, 1) = {F, F} = F → comes under {F} of P_1

So, P_2 becomes

$$P_2 = \{\{A, B\}, \{F\}, \{C, D\}\}$$

* Check with {D, E}. ⇒ δ({D, E}, 0) = {E, E} = E → Present in {C, D, E} of P_1

δ({D, E}, 1) = {F, F} = F → present in {F} of P_1.

{D, E} are equivalent and {C, D} are equivalent.

So, {C, D, E} are equivalent. P_2 becomes,

$$P_2 = \{\{A, B\}, \{F\}, \{C, D, E\}\}$$

(iv) 3-equivalence / P_3:

Consider, $P_2 = \{\{A, B\}, \{F\}, \{C, D, E\}\}$

* Check with {A, B}, ⇒ δ({A, B}, 0) = {B, A} → present in the same partition {A, B} of P_2.

δ({A, B}, 1) = {C, D} → present in the same partition {C, D, E} of P_2.

{A, B} are equivalent. P_3 is,

$P_3 = \{\{A, B\}, \{F\}, \{C, D, E\}\}$

* Check with {C, D, E}

{C, D} ⇒ δ({C, D}, 0) = {E, E} = E → comes under same partition {C, D, E} of P_1.

δ({C, D}, 1) = {F, F} = F → comes under {F} of P_1.

{C, D} are equivalent.

$P_3 = \{\{A, B\}, \{F\}, \{C, D\}\}$

* {D, E} ⇒ δ({D, E}, 0) = {E, E} = E → present in partition {C, D, E} of P_2.

δ({D, E}, 1) = {F, F} = F → present in partition {F} of P_2.

{D, E} are equivalent and {C, D} are equivalent.

Hence, {C, D, E} are equivalent.

$$P_3 = \{\{A, B\}, \{F\}, \{C, D, E\}\}$$

$P_3 = P_2$. So, we can stop and combine the partitions of P_3.

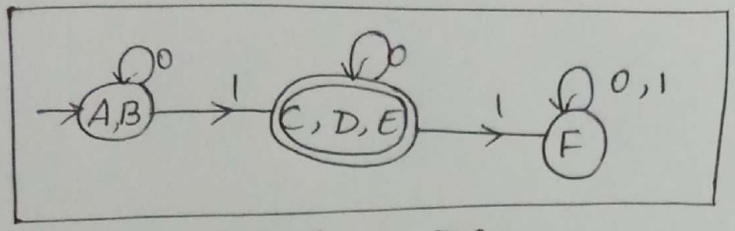
New states ⇒ {A, B}, {F}, {C, D, E}

Draw the transition table and transition diagram for minimized DFA.

δ	0	1
→ {A,B}	{A,B}	{C,D,E}
{F}	{F}	{F}
* {C,D,E}	{C,D,E}	{F}

⊛ A and B are combined to form a single state. So where ever A or B comes, replace it by {A,B}.

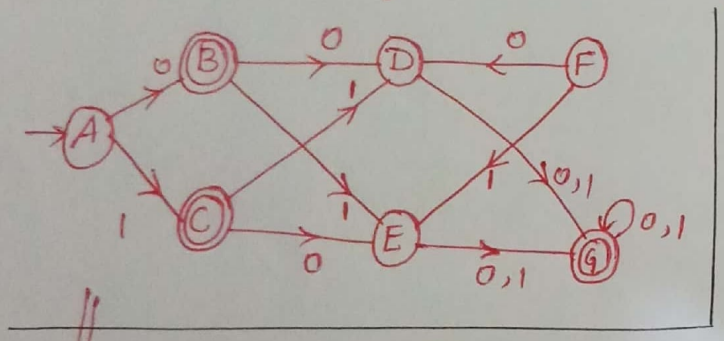
⊛ C, D and E are combined as one state. So where ever C, D or E comes replace it by {C,D,E}



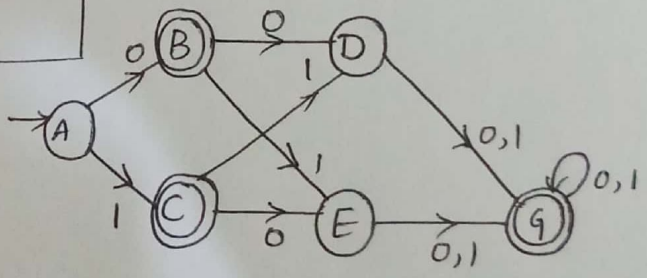
Minimized DFA.

Exercise problems

1) Minimize the given DFA using partition method.



In this problem F is unreachable state. So, it can be eliminated directly. We will get the following DFA.



Minimization of DFA, when there are not reachable states.

Now, follow the same steps and minimize this DFA.

2)

δ	a	b
→ A	B	D
B	C	E
C	B	E
D	C	E
* E	E	E