UNIT_1

Introduction to finite automata: *The central concepts of automata theory, Structural representation of FA, Types of FA, Conversion of NFA to DFA, NFA with epsilon to NFA without epsilon conversion.*

Regular Expression: *Introduction to Regular language, Algebraic laws for regular expressions, Conversion of FA to RE, Conversion of RE to FA, Pumping lemma for Regular language.*

==========

Introduction to Finite Automata _Fundamentals – Terminology

Automation: means automatic machine / self-controlled

- Ex: computer, ATM

Why we study it:

- It allows us to think systematically about what machine do without going into Hardware details.
- Learning of languages and computational techniques.
- Designing of theoretical models for machines.
- It is core subject of computer science.

To understand a machine Language we should know

- How we are going to give input to machine
- How machine takes the input
- What are the limitations of machine?

Central concepts of automata theory

1. Symbol

- Something that has some meaning.
- Can't be further divided.

Eg: { a, b, 0, 1, +, ?, :, +, ++ }

---

2. Alphabet

- Has finite set of symbols?
- Denoted by ∑

Eg : ∑ = { 0, 1 }
∑ = { a, b }
∑ = {[#, @ }
∑ { 0. . .9 }
∑ eng = { A, B, …..Z, a, b . . .z }

∑ bin = { 0, 1}
∑ dec = { 0, 1 . . .9 }
∑ empty = Ø

3. String / Word / sentence

- Finite sequence of symbols over ∑
- Denoted by w.

∑ = { 0, 1 }
W = 101, w = 1100, w = 111 00 11
∑ { a, b }
W = aba, w = bab, w = aabbaa

---

4. String operations

- Empty string is denoted by ε / л / λ
- String length: * # of symbols in the word.
- Denoted by |w|

---

5. Concatenation: with Є

Є. W = w. Є = w.

Eg : w = 110110 w = Є 011100 Є
|w|= 6 |w| =6
w = Є 00 ЄЄ 011 Є 00
|w| = 7

Don't consider Є while calculating length of a string.

---

6. Prefix

- The first letter of each string is fixed. But ending may be anywhere.

Eg: w = abcd
Prefix(w) = {Є, a, ab, abc, abcd}

---

7. Suffix

- The last letter of each string is fixed. But start may be anywhere.

Eg: w = abcd
Suffix(w) = {Є, d, cd, bcd, abcd}

---

8. Subsequence

- Starting and ending is anywhere.

Eg: w = abcd
Subsequence(w)= { Є, a, b, c, d, ab, ac, ad, bc, bd, cd, abc, abd, bcd, abcd}

---

9. Substring

- Letters of consecutive symbols.

Eg: w = abcd
Substrings(w) = {Є, a, b, c, d, ab, bc, cd, abc, bcd, abcd}

---

10. Concatenation

- Concatenation of two strings w1 and w2
- W1 is followed by w2 without any space b/w them

Eg : w1 = Success w2 = fully
w1.w2 = successfully

---

11. Proper substring:

- All substrings of w except itself.

Eg : FLAT
Proper substrings: {Є, F, L, A, T, FL, LA , AT, FLA, LAT}

12.Palindrome

- Can be read w either side identically.

Eg : w = { 0, 1 }
w = 0110, w = 10001, w = 111000111

---

13. Power Alphabet

- The power alphabet, $\sum^k$
- K denotes set of all strings of length k.

Eg: let $\sum$ = {0, 1}

$\sum^0 = \{Є\}$ $\sum^1 = \{0, 1\}$ $\sum^2 = \{00, 01, 10, 11\}$

---

14. Kleene closure

- All strings of length 0 or more instances.
- Denoted by $\sum^*$

$$\sum^* = \sum^0 U \sum^1 U \sum^2 \ldots\ldots$$

$$\bigcup_{i\geq 0}^{\infty} \sum^i$$

$$\sum^* \sum^* = \sum^*$$
$$\sum^+ + \sum^0 = \sum^*$$
$$\sum^+ + Є = \sum^*$$

---

15. Positive Closure

- $\sum^+$ denotes all strings of length 1 or more.

$$\sum^+ = \sum^1 U \sum^2 U \sum^3 \ldots\ldots$$

$$\sum^* \cap \sum^+ = \sum^+$$
$$\sum^* U \sum^+ = \sum^*$$
$$\sum^* \sum^+ = \sum^+$$
$$\sum^+ \sum^+ = \sum^+$$
$$\sum^* - Є = \sum^+$$

a. Є = a

$$\longrightarrow \bigcup_{i\geq 0}^{\infty} \sum^i$$

---

16. Set operation:

- Set is a collection of distinct objects.
- Set is denoted by S.

Eg :1. S = { 0,1, 2, 3, 4 } → finite set
2. S = { 0, 1, 2, 3, 4 . . . } → Infinite set
3. S = { } → Empty | Null set

---

17. Set size | cardinality:

- The number of objects in the set

Denoted by |S|
Eg : s = {[1, 2, 3 } → |S|= 3

---

18. Finite | countable set:

- Every element of the set is countable

Eg: S = {10, 20, 30, 40}

19. Infinite | countably Infinite set

- Basically, finite set.
- Every object is countable for sometimes.

Eg: set of natural Numbers.

---

20. Intersection

- Common elements from the given sets.

A = {1, 2, 3, 4, 5} B = {2, 6, 7, 8} A ∩ B = {2}

---

21. Union

- Unique elements from the given sets.

A U B = {1, 2, 3, 4, 5, 6, 7, 8}

---

22. Cartesian product

- Multiply each element of one set by another element of another set.

Eg : A = { 1, 2 } , B = {a, b }
A * B = { (1, a), (1, b), (2, a), (2, b) }

---

23. Language: (Finite and Infinite)

- subset of $\sum^*$
- Denoted by L.

Finite Language: Finite | countable number of objects in the set.
Eg: $\sum$ = {a, b}
$\sum^2$ = {aa, ab, ba, bb}

Infinite Language: Infinite number of elements in the set.

Eg: All strings of starts with a.

{a, ab, aa, aab, aab, aba,……}

$\sum$ = {a, b } → L_1 = { a, aa, ab, aab, abb……}

L_2 = { } → Φ

L_3 = { ε }

L_4 = {a, b } → { ε, ab, aabb, aaabb…….}

L_5 = {a^n | n>= 0 } { a^0, a^1, a^2, a^3,……. }

L_6 = {a^n b^m | n, m >= 1 }

= {ab, aab, abb, aaab,……}

---

23. Automata

- Mathematical model of computation.
- Abstract machine to perform computation.

Eg : Can generate primes

Can check primes.

24. Types of Automata

- Finite Automaton
- Pushdown Automaton
- Linear Bounded Automaton
- Turing Machine

---

25. Grammar

- Generates valid strings using rules.

---

26. Types of languages, Automata and grammar

Language	Automata	Grammar
Type 0[REL]	TM	UG
Type 1[CSL]	LBA	CSG
Type 2[CFL]	PDA	CFG
Type 3[Regular Language]	FA	RG

Practice Questions

1. Let $\sum$ = {0, 1} How many strings of length n are possible over $\sum$.

   A) n B) n+1 C) 2^n D) 2n

---

2. w be a string of length n. How many suffixes are possible for w.?

   A) n B) n+1 C) 2^n D) 2n

3. Let $\sum$ = {0, 1, 2} How many strings of length 3 or less are possible?

A) 8 B) 9 C) 18 D) None

$L^3 = 3^3$

$L^2 = 3^2$

$L^1 = 3^1$ $27+9+3+1 = 40$

$L^0 = 3^0$

---

4. Let ∑ = {0, 1}, Let L = {01, 110, 100}

i) 0100110010
ii) 100110100110
iii) 110100011110
iv) 1010101110

Which of the combination of strings belong to L^*?

A) i & ii B) ii & iv C) iii & iv D) None (only ii)

Structural Representation of Finite Automata

- A finite automaton (FA) can be represented structurally as a directed graph, also called a state transition diagram or state machine diagram.
- The graph consists of vertices (nodes) and edges. In an FA, the vertices represent the states of the machine and the edges represent the transitions between states based on input symbols.
- The FA has one initial state, represented by an arrow pointing to the state. The initial state is where the machine begins processing input symbols.
- The machine may have one or more accepting states, indicated by a double circle around the state. An accepting state is where the machine accepts the input string.
- The edges of the graph represent transitions between states. Each edge is labeled with an input symbol, indicating the symbol that causes the machine to transition from one state to another. For a deterministic finite automaton (DFA), each state has exactly one edge leaving it for each input symbol in the alphabet.
- For a non-deterministic finite automaton (NFA), a state may have multiple edges leaving it for the same input symbol or even have epsilon (ε) transitions, which means the machine can transition to another state without reading an input symbol.

Finite Automata

- Finite automata are used to recognize patterns.
- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
- At the time of transition, the automata can either move to the next state or stay in the same state.

- Finite automata have two states, **Accept state** or **Reject state**. When the input string is processed successfully, and the automata reached its final state, then it will accept.

→ In other words, a FA is a 5-tuple system.
M = (Q, ∑, δ, q_0, F) where

Q: Finite set of states
∑: Input alphabet
δ: Transition function mapping from Q X ∑ → Q
q_0: Initial state
F: Set of final states

A FA is represented in two ways
1. State Diagram
2. Transition state Table

State Diagram

→ can also be called transition diagram | transition system.

→ It is a direct labelled graph in which nodes are corresponding to states and directed edges are corresponding to transition states.

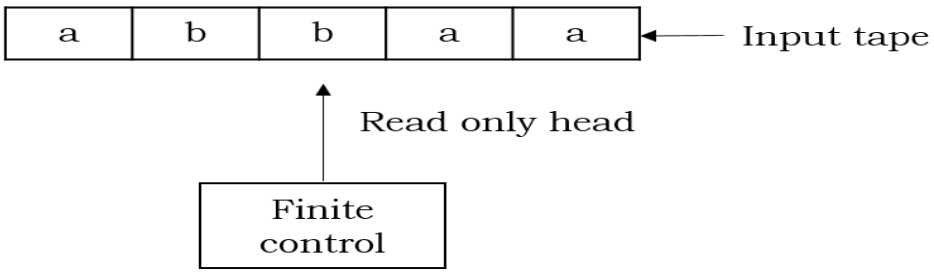
→ Initial states are denoted by → 

→ final state is denoted by 

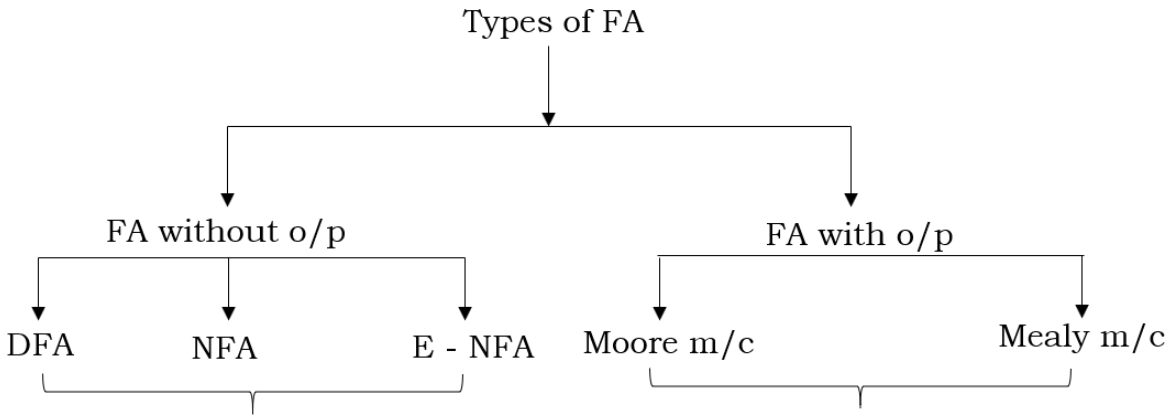
State Transition Table

- In transition table, rows are corresponding to state and columns are corresponding to i/p.
- For each combination of present state and present i/p, corresponding entry specifies next state of the system.

Configuration | Block Diagram of FA



- A FA consists of Input tape, read only head and Finite Control.
- The i/p tape consists of set of cells each hols one i/p symbol.
- The read only head moves left to right and scans one symbol at a time.
- FC determines the next state of the system using symbol under read only head and the present state of the system.



Applications

1. Used to check spellings In Editor
2. Lexical analysis in compilers
3. General Applications
4. To model the behavior of electrical ckts
5. To model the behaviour of digital ckts
6. To model the behaviour of mechanical devices
7. To solve puzzles.

Applications

1. Computing1's complement
2. Computing 2's compliment
3. Adder | Subtractor
4. Counter
5. Sequence Generator/detector

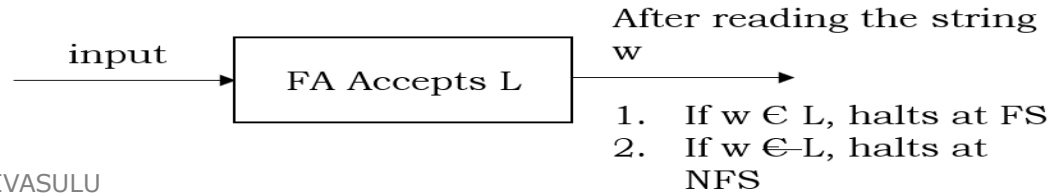
Equality

All types of FAS are equal. Equivalence means not the same but they perform the same task.

L (FA) = L (DFA) = L (NFA) = L (ε- NFA) = RL.

FA $\overset{!}{\cong}$ DFA $\overset{!}{\cong}$ NFA $\overset{!}{\cong}$ ε- NFA $\overset{!}{\cong}$ = RL.

FA without o/p

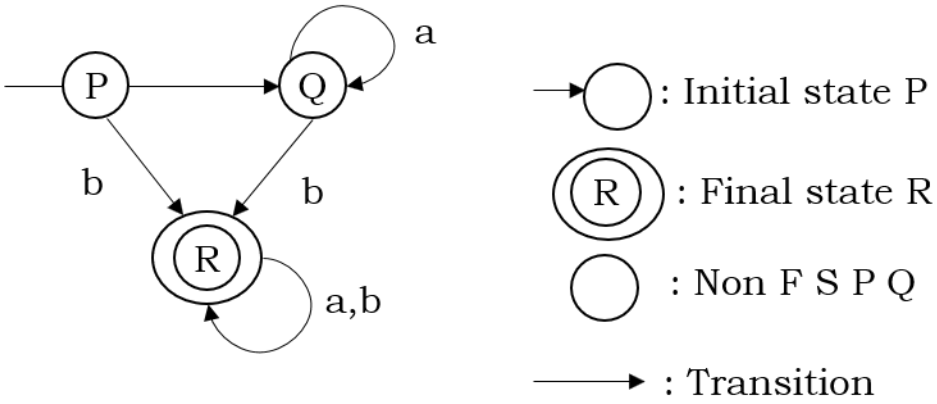


FA Representation

DFA . . . . . $\delta\ (Q \times \sum) \rightarrow Q$
NFA . . . . . $\delta\ (Q \times \sum) \rightarrow 2^Q$
ϵ – NFA . . . $\delta\ (Q \times \sum \cup \{\epsilon\} \rightarrow 2^Q$

1. State Diagram
2. Transition Table
3. Transition Function

State Diagram



Transition Table

Staes	a	b
P	Q	R
Q	Q	R
R	R	R

String Acceptance by FA

- If there Exists a path from initial state to final state the string w is said to be accepted.
- In other words, w is accepted by FA
  iff $\delta\ (q_0, w) \rightarrow$ for some P in F.

Language Accept

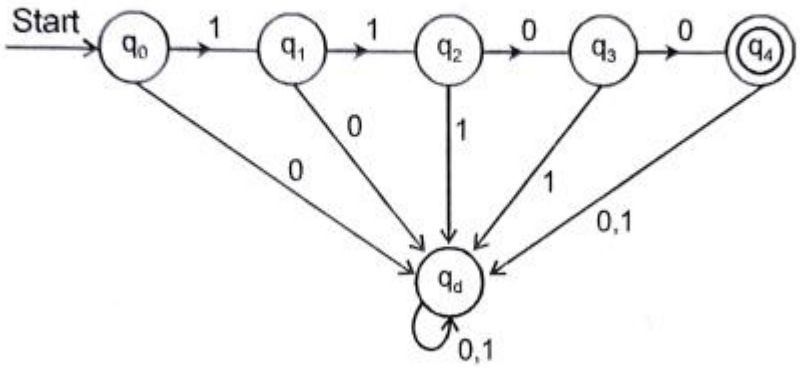
- A FA is said to be accepted a lang if all the strings in the lang are accepted.
- All the strings that are not in the lang are rejected by the FA.

DFA PROBLEMS

Problem 1:

Design DFA which accepts string 1100 only

Solution:

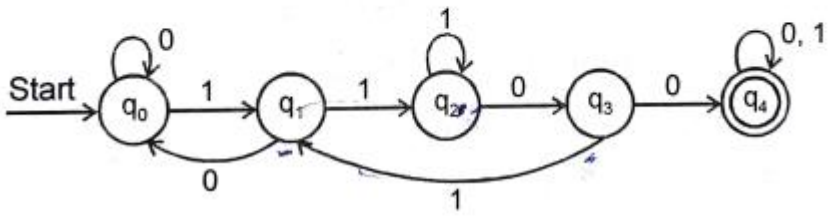


Where q_d is called Dummy state, trap state or Dead state.

Problem 2:

Design a DFA which accepts set of all strings contains 1100 as substring, where $\Sigma = \{0,1\}$

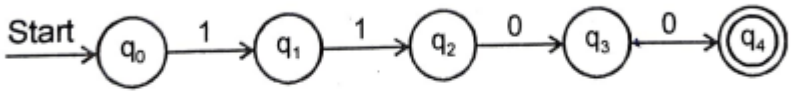
Solution:



Procedure:

Step 1:

1100 is a sub string itself so draw the finite automata for 1100



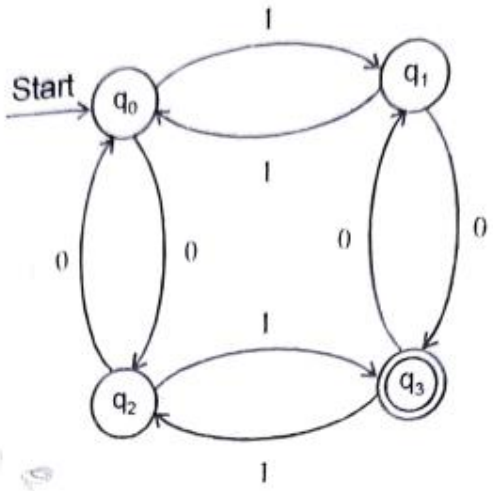
Step 2:

q_0 on 1 already path is there to q_1 and q_0 on o should stay in the q_0 because 1100 substring may have any number of 0's before it. q_1 on 0 sub string matching string is failed so it checks from the starting state q_0. q_2 on 1 string is failed so it should return back to the starting state but after two 1's any number of 1's can be there. So it returns in the same state. q_3 on 1 string matching failed so it returns back to the starting state but it already see in first alphabet of substring so it returns to q_1. After subset is matched any input can be accepted so q_4 on 0 or 1 remains in the same state.

Problem 3:

Design a DFA which accepts set of all strings containing odd number of 0's and odd number of 1's.

Solution:



Procedure:

Step 1:

Select all combinations of even and odd number of 0's and 1's as states ie, q_0 even number of 0's and even number of 1's

q_1 : even number of 0's and odd number of 1's.

q_2 : odd number of 0's and even number of 1's.

q_3 : odd number of 0's and odd number of 1's.

q_3 contains odd number of 0's and odd number of 1's so make it as a final state.

Step 2:

q_0 on 0 ie, on even number of zeros and even number of 1's if we apply another 0 it becomes odd number of 0's. So it moves to odd number of 0's and even number of is state ie, q_2 similarly for other states and inputs also,

Problem 4:

Design DFA which accepts set of all strings containing mod3 is zero where string is treated as binary numbers.

Solution:

Step 1:

Take all posible reminders as states

Step 2:

For any binary string if we add a bit at LSB the previous value gets doubled this can

be genarlized as 2 × n+a where n is previous number and a is the bit added.

So (2 × n+a) mod3 ie 2 × n mod3 + a mod3

ie 2 × (State number or Reminder)+a

Step 3:

Substitute as the above and get the values.

Solution:

For |3| the remainders are 0,1,2. So let us consider 0 at q_0,

1 at q_1 and 2 at q_2 .

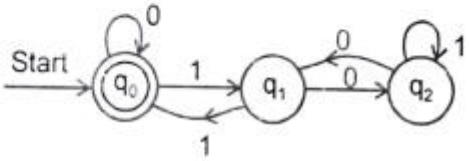
$\Sigma = \{0,1\}$ and base =2.

$$(q_0, 0) = 2\times0+0 = 0 \rightarrow q_0$$
$$(q_0, 1) = 2\times0+1 = 1 \rightarrow q_1$$
$$(q_1, 0) = 2\times1+0 = 2 \rightarrow q_2$$
$$(q_1, 1) = 2\times1+1 = 0 \rightarrow q_0$$
$$(q_2, 0) = 2\times2+0 = 1 \rightarrow q_1$$
$$(q_2, 1) = 2\times2+1 = 2 \rightarrow q_2$$

from the above,

$(q_1, 1) = 2\times1+1 = 3\%3 = 0$

Base.

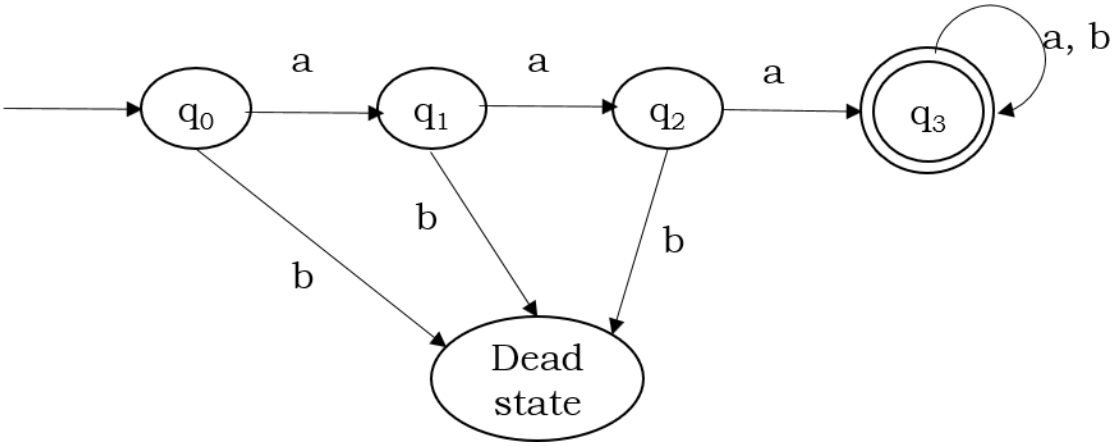


Practice QuestionsMODEL :1
(start | End | contain)

1. Dead state exists
2. # states : n+2

1. No dead state
2. # states +1

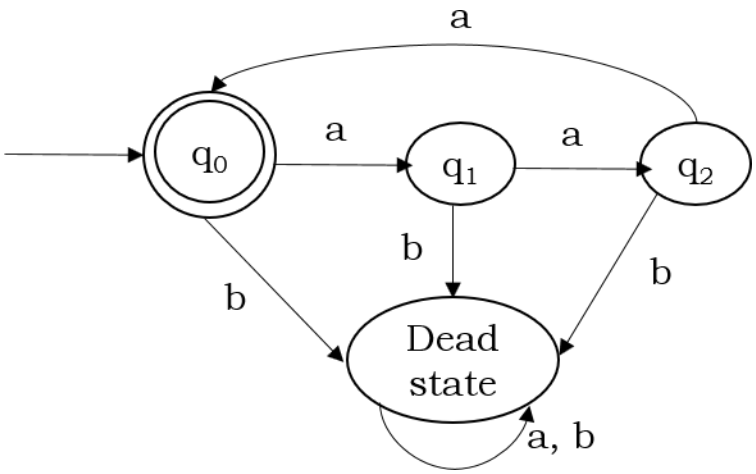
1. Design DFA for the lang RE: aaa (a+b) * → 5 states



Exercise 1: Construct DFA for the RE: (a+b) * abb (a+b)* → contain
Exercise 2: Construct DFA for the RE: (a+b)* aaa → Ends with aaa Need 3+1 states.

Model : 2
(Language over 1 Symbol)

2. L = (aaa)* over ∑ = { a, b }

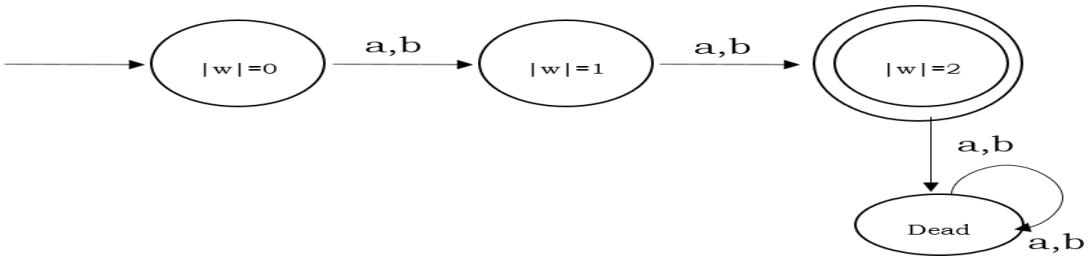


Exercise 1: L = { a^m, b^n, c^L | m, n, L >= 1 }

Exercise 2: L = { a^m, b^n, c^L | m, n, L >= 0 }

Model: 3
(Length and Divisible problems)

3. L = {String length is exactly 2 } over ∑ = { a, b }*



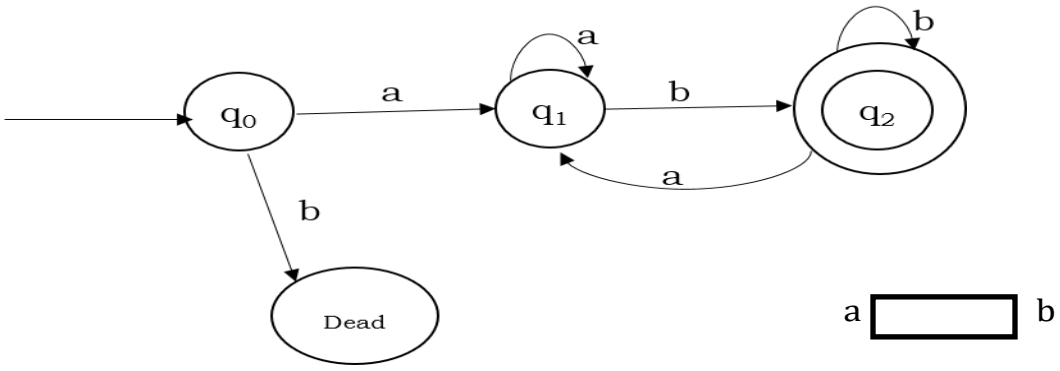
Exactly n length	Almost = |w| <= 2	Atleast = |w| >= 2
= n+2 states	n+2 states	n+1 states
Dead state exists	Dead state exists	No Dead state

Exercise 1:L = { w|w Є {[a, b }, |w| <= 2 }

Exercise 2: L = { w|w Є { a, b }*, |w| >= 2 }

MODEL: 4
(Multiple conditions)

4. L = { starts with a and ends with b }



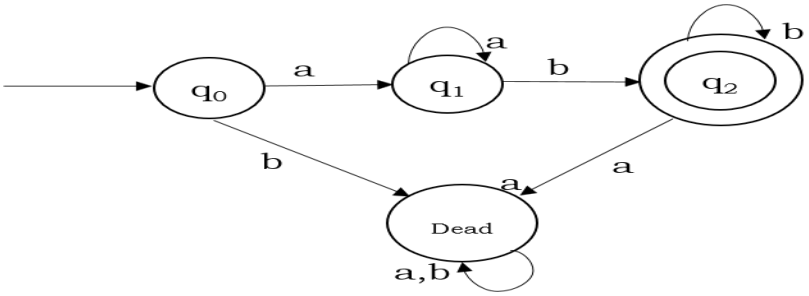
Exercise 1: L = {[strings start and end with different symbols} over ∑ = {a, b}
Exercise: 2: L = (aa+bb) (a+b)* ..... starts with aa or bb.

MODEL: 5
(Position Based problems)

Exercise 1: L = { 2nd symbol from LHS is a }
Exercise 2: L = { a^3 b w a^3 / w Є { a, b }* }

MODEL : 6
(Sequence Based problems)

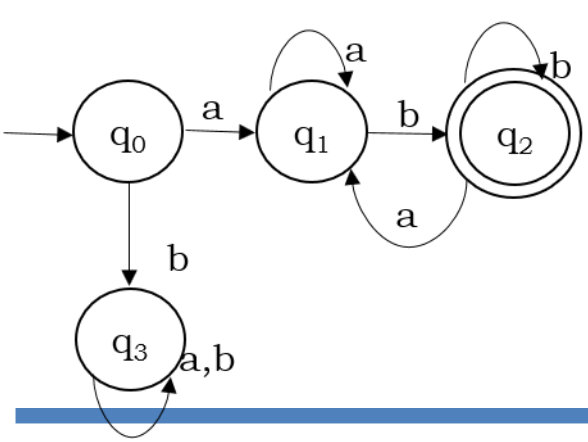
5. L = { $a^+ b^+$ }

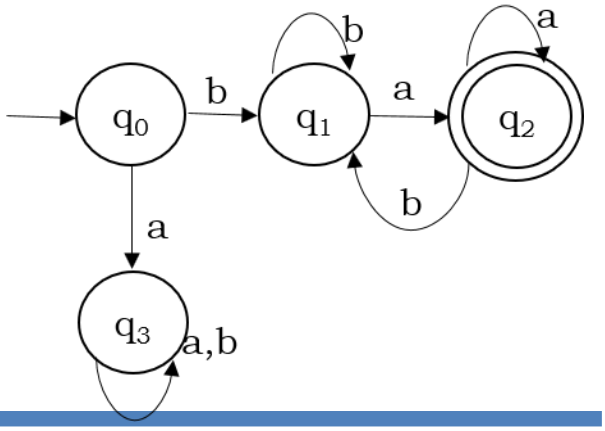


6. UNION of two FAs

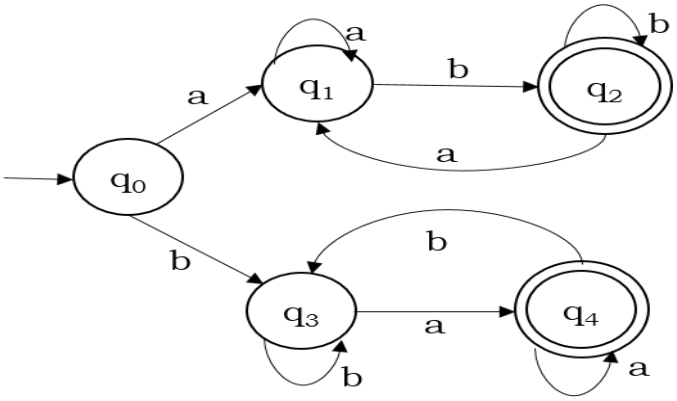
L_1 = { start with a and end with b } = { ab, aab, abb, aaab . . . }

L_2 = { start with b and end a } = { ba, bba, bbba, baaa . . }





Union of the above FA (starting and ending with different Symbols) $L_1 \cup L_2$

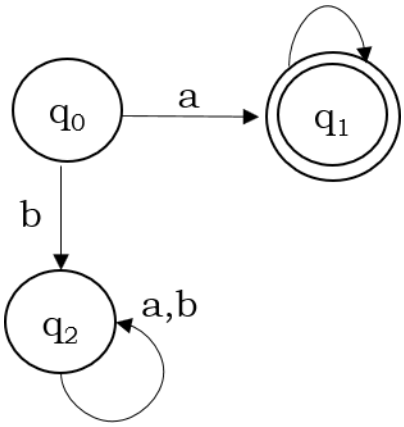


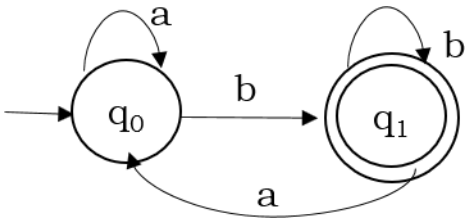
---

7. Concatenation of two FA.

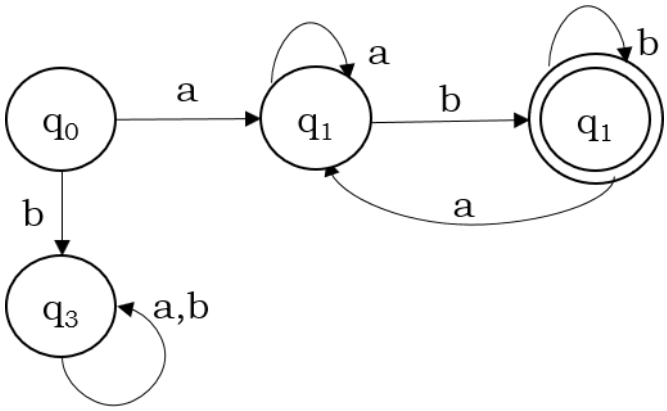
L_1 = {[start with a }
= { a, aa, ab, aaa . . . }

L_2 = {Ending with b }
+ { b, ab, aab, bbb . . . }





Concatenation of L_1 . L_2 (start with a and End with b)

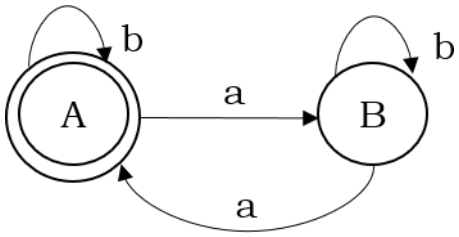


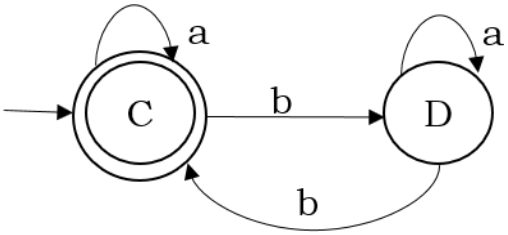
---

8. Cross product

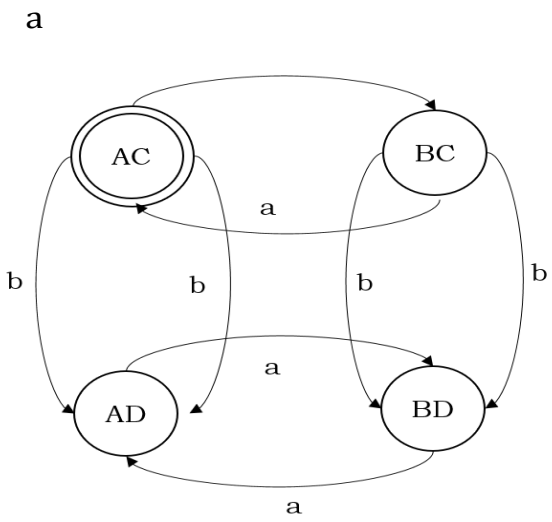
Even no of a's
L_1 = { aa, aba, aab . . . }

Even no of b's
L_2 = { bb, bab, bba . . . }





L_1 X L_2 = { AB X CD } = { AC, AD, BC, BD }



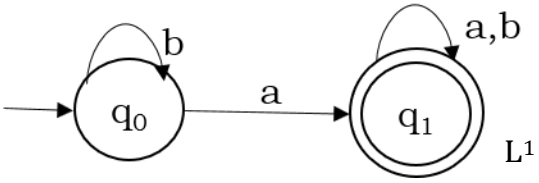
---

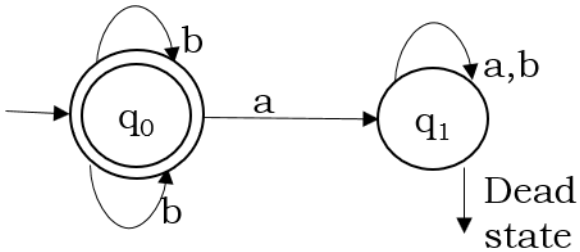
9. Compliment of language L_1

L_1 = {containing a }
= { a, aa, ba, bab, . . . }

$\bar{L}_1$ = { does not contain a }
= { Є, b, bb, bbb, . . . }

i) Final to non-final
ii) Non- final to final



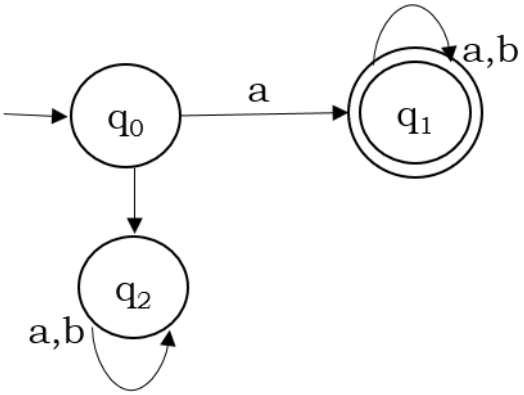


==========

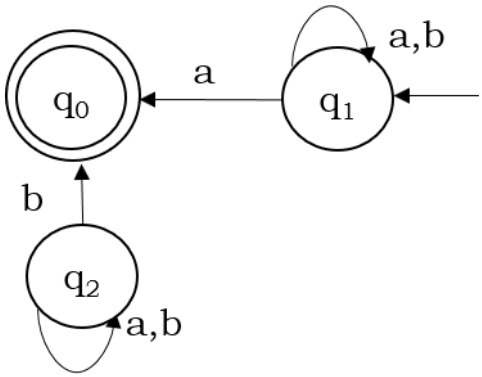
10. **Reversal:** start with a

L = { a, aa, ab, aaa . . . }

L^R = {Reverse of strings}
= { a, aaa, ba, aaa . . . }



(Ending with a . . .)



(Staring with a . . .) Its NFA

Since no transition q_0

i) Final to initial & initial to final.
ii) No change in Symbols
iii) Reverse transition.

Non-Deterministic Finite Automata

- In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine.
- In other words, the exact state to which the machine moves cannot be determined. Hence, it is called Non-deterministic Automaton.
- As it has finite number of states, the machine is called Non-deterministic Finite Machine or Non-deterministic Finite Automaton.

Formal Definition of an NDFA

An NDFA can be represented by a 5-tuple $(Q, \sum, \delta, q_0, F)$ where –

- **Q** is a finite set of states.
- **∑** is a finite set of symbols calling the alphabets.
- **δ** is the transition function where $\delta: Q \times \sum \rightarrow 2^Q$

(Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)

- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of an NDFA: (same as DFA)

An NDFA is represented by digraphs called state diagram.

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

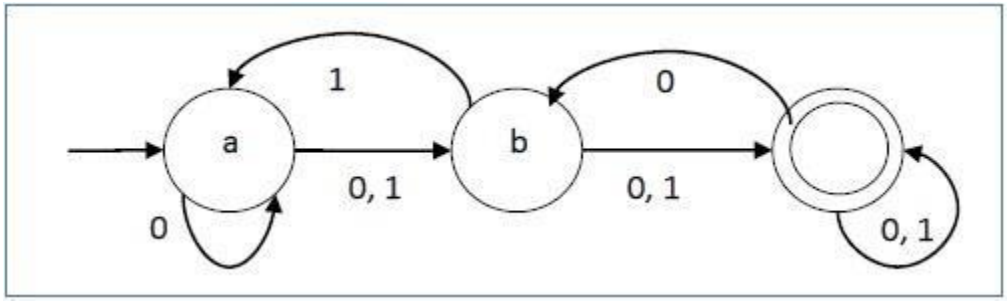
Example

Let a non-deterministic finite automaton be →

- Q = {a, b, c}
- ∑ = {0, 1}
- q_0 = {a}
- F = {c}

The transition function δ as shown below

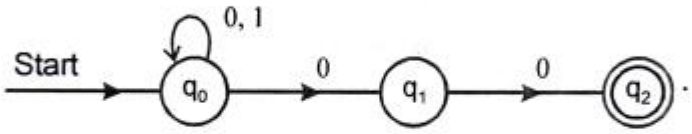
Present State	Next state for Input 0	Next state for Input 1
a	a, b	b
b	c	a, c
c	b, c	c



Problem 1:

Design an NFA to accept set of strings over alphabet set {0, 1} and ending with two consecutive O's.

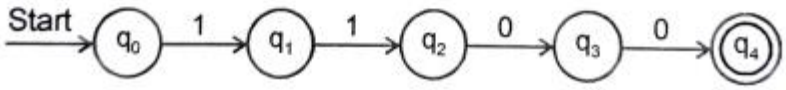
Solution:



Problem 2:

Design NFA which accept string 1100 only

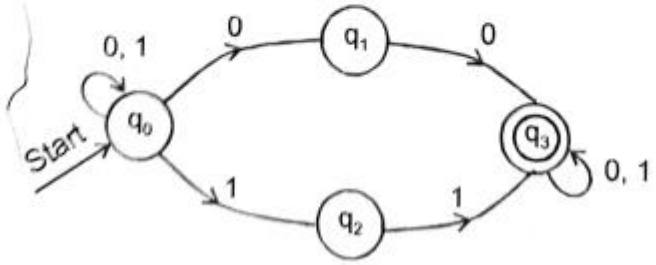
Solution:



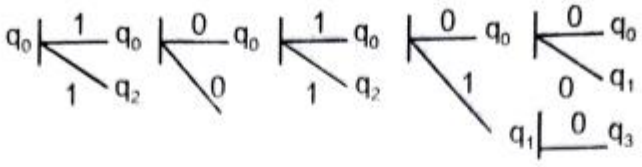
Problem 3:

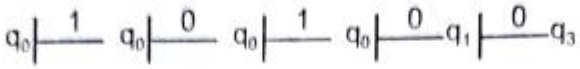
Design a NFA to accept the strings with 0's and 1's such that string contains either two consecutive 0's or two consecutive 1's.

Solution:



Consider a string 10100

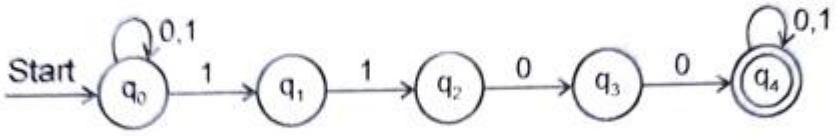


The path is 

Given string is accepted as transition reaches to final state, q_3.

Problem 4:

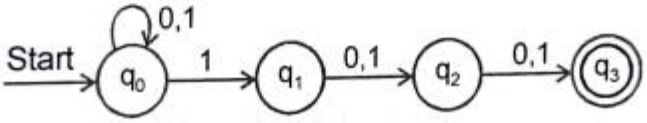
Design NFA which accepts set of all strings containing 1100 as substring.



Problem 5:

Design NFA which accepts set of all strings containing 3rd symbol from right side is 1.

Solution:



DFA vs NDFA

The following table lists the differences between DFA and NDFA.

DFA	NDFA
The transition from a state is to a single particular next state for each input symbol. Hence it is called *deterministic*.	The transition from a state can be to multiple next states for each input symbol. Hence it is called *non-deterministic*.
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
Backtracking is allowed in DFA	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.

NDFA to DFA Conversion

Problem Statement

Let X = (Qx, Σ, δx, q0, Fx) be an NDFA which accepts the language L(X). We have to design an equivalent DFA Y = (Qy, Σ, δy, q0, Fy) such that L(Y) = L(X). The following procedure converts the NDFA to its equivalent DFA:

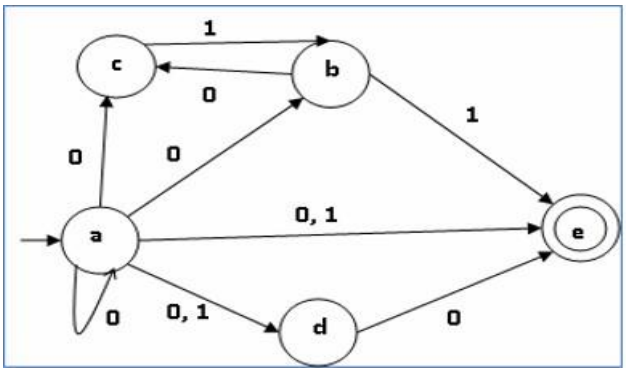
Algorithm

Input:	An NDFA
Output:	An equivalent DFA
Step 1	Create state table from the given NDFA.
Step 2	Create a blank state table under possible input alphabets for the equivalent DFA.
Step 3	Mark the start state of the DFA by q0 (Same as the NDFA).
Step 4	Find out the combination of States {Q0, Q1,... , Qn} for each possible input alphabet.
Step 5	Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.

Step 6 The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Example

Let us consider the NDFA shown in the figure below.



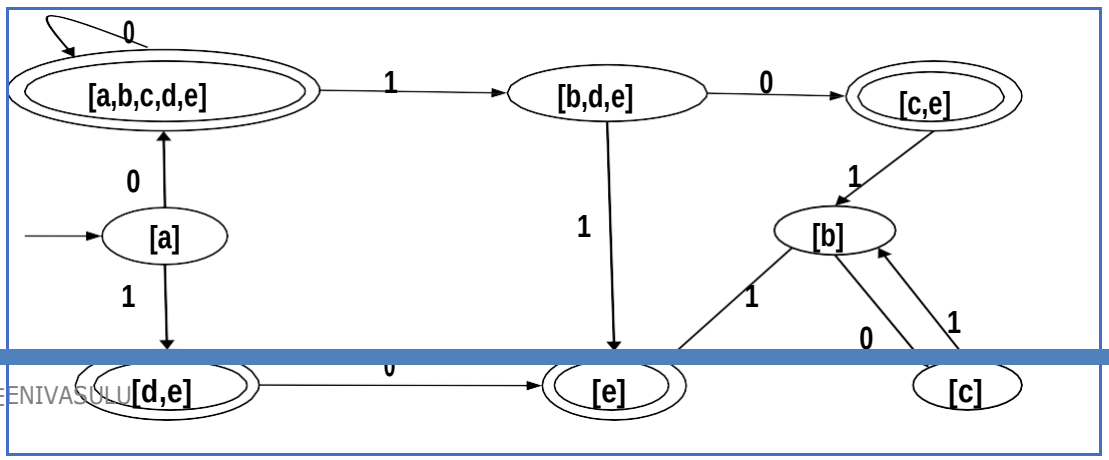
q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
b	{c}	{e}
c	∅	{b}
d	{e}	∅
e	∅	∅

Using the above algorithm, we find its equivalent DFA. The state table of the DFA is shown in below.

Transition Table for DFA

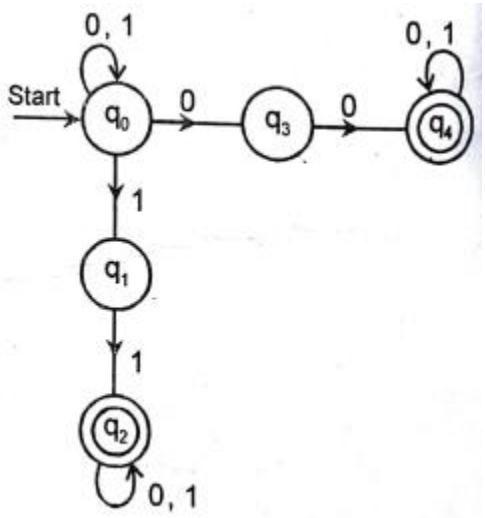
q	δ(q,0)	δ(q,1)
[a]	[a, b, c, d, e]	[d,e]
[a, b, c, d, e]	[a, b, c, d, e]	[b, d, e]
[d, e]	[e]	∅
[b, d, e]	[c, e]	[e]
[e]	∅	∅
[c, e]	∅	[b]
[b]	[c]	[e]
[c]	∅	[b]

The state diagram of the DFA is as follows:



Example 1:

Convert the following NFA to DFA



Transition table for the given NFA diagram is

State \ i/ps	0	1
q_0	q_0, q_3	q_0, q_1
q_1	ϕ	q_2
((q_2))	q_2	q_2
q_3	q_4	ϕ
((q_4))	q_4	q_4

First we make the starting state of NFA as the starting state of DFA. Apply 0 and 1 as i/p on state q_0. Now keep the o/p's in 2nd and 3rd columns. Next take the new generated states which we placed in 3rd and 4th columns keep them in first. Column and again apply 0's and 1's as i/p. Repeat the process until no new state is left.

DFA's transition table

The deterministic automation M_2 equivalent to the above M_1, defined as

$M_2 = (2^Q, \{0, 1\}, \delta, [q_0], F)$

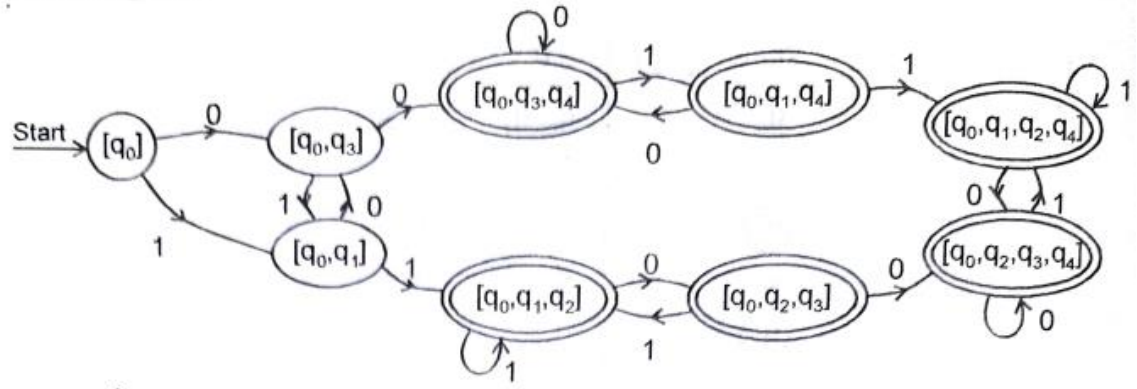
$F = \{[q_0, q_3, q_4], [q_0, q_1, q_4], [q_0, q_1, q_2, q_4]$

$[q_0, q_1, q_2], [q_0, q_2, q_3], [q_0, q_2, q_3, q_4]\}$

δ is defined as

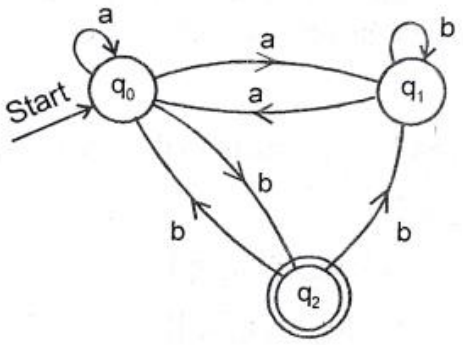
state/ip	0	1
$[q_0]$	$[q_0, q_3]$	$[q_0, q_1]$
$[q_0, q_3]$	$[q_0, q_3, q_4]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_3]$	$[q_0, q_1, q_2]$
$[q_0, q_3, q_4]$	$[q_0, q_3, q_4]$	$[q_0, q_1, q_4]$
$[q_0, q_1, q_2]$	$[q_0, q_2, q_3]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_4]$	$[q_0, q_3, q_4]$	$[q_0, q_1, q_2, q_4]$
$[q_0, q_2, q_3]$	$[q_0, q_2, q_3, q_4]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2, q_4]$	$[q_0, q_2, q_3, q_4]$	$[q_0, q_1, q_2, q_4]$
$[q_0, q_2, q_3, q_4]$	$[q_0, q_2, q_3, q_4]$	$[q_0, q_1, q_2, q_4]$

DFA Diagram:



Example 2:

Construct a DFA diagram to the NFA given below



NFA

Solution:

Let equivalent DFA, $M_2 = (Q', \Sigma, \delta', q'_0, F')$

of given NFA, $M_1 = (Q, \Sigma, \delta, q_0, F)$

Q' is $\{q_0, q_2, [q_0, q_1], [q_1, q_2], q_d\}$

F' is $\{q_2, [q_2, q_1]\}$

q'_0 of DFA is q_0 of NFA

Σ of DFA Σ of NFA

δ is defined as $\delta'(q_0, a) = \{q_0, q_1\}$

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

$\delta'([q_0, q_1], a) = \delta(q_0, a) \cup \delta(q_1, a)$

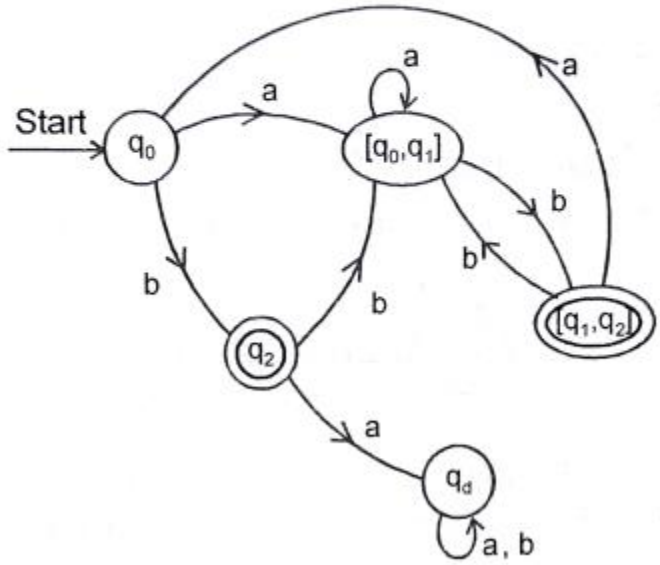
$= \{q_0, q_1\} \cup \{q_0\}$

$= \{q_0, q_1\}$

$\delta'([q_0, q_1], b) = \delta(q_0, b) \cup \delta(q_1, b)$

$= \{q_2\} \cup \{q_1\} = \{q_1, q_2\}$

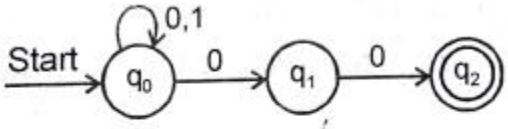
$$\delta'([q_1, q_2], a) = \delta(q_1, a) \cup \delta(q_2, a)$$
$$= \{q_0\}$$
$$\delta'([q_1, q_2], b) = \delta(q_1, b) \cup \delta(q_2, b)$$
$$= \{q_1\} \cup \{q_0, q_1\} = \{q_0, q_1\}$$



DFA

Example 3:

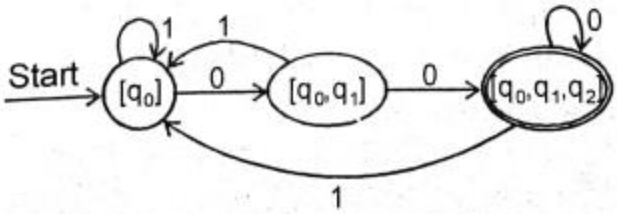
Convert NFA to DFA,



Solution:

	0	1
$[q_0]$	$[q_0,q_1]$	$[q_0]$
$[q_0,q_1]$	$[q_0,q_1,q_2]$	$[q_0]$
$[q_0,q_1,q_2]$	$[q_0,q_1,q_2]$	$[q_0]$

The DFA is



Conversion of Epsilon-NFA to NFA

Non-deterministic Finite Automata (NFA) is a finite automata having zero, one or more than one moves from a given state on a given input symbol. Epsilon NFA is the NFA which contains epsilon move(s)/Null move(s). To remove the epsilon, move/Null move from epsilon-NFA and to convert it into NFA, we follow the steps mentioned below.

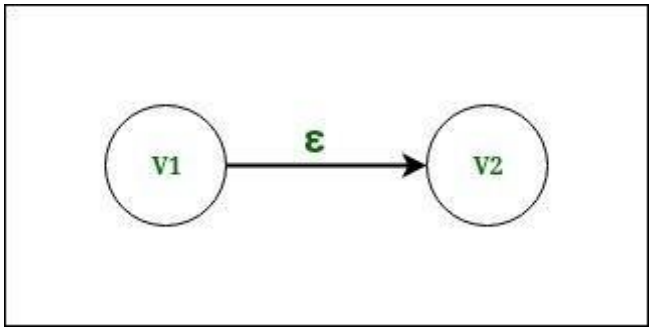


Figure – Vertex v1 and Vertex v2 having an epsilon move

Step-1:

Consider the two vertexes having the epsilon move. Here in Fig.1 we have vertex v1 and vertex v2 having epsilon move from v1 to v2.

Step-2:

Now find all the moves to any other vertex that start from vertex v2 (other than the epsilon move that is considering). After finding the moves, duplicate all the moves that start from vertex v2, with the same input to start from vertex v1 and remove the epsilon move from vertex v1 to vertex v2.

Step-3:

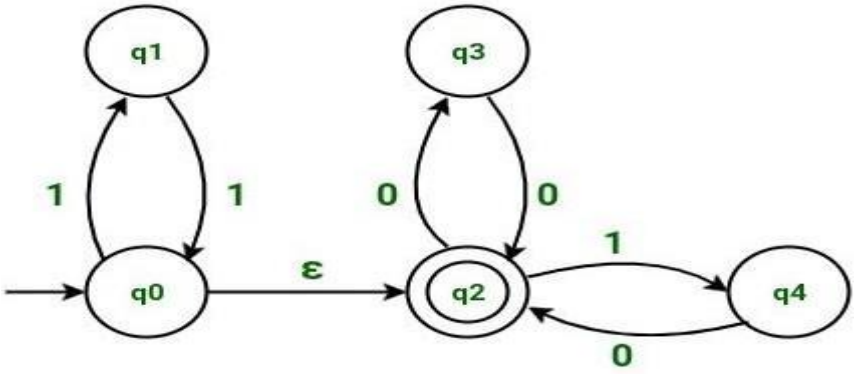
See that if the vertex v1 is a start state or not. If vertex v1 is a start state, then we will also make vertex v2 as a start state. If vertex v1 is not a start state, then there will not be any change.

Step-4:

See that if the vertex v2 is a final state or not. If vertex v2 is a final state, then we will also make vertex v1 as a final state. If vertex v2 is not a final state, then there will not be any change. Repeat the steps(from step 1 to step 4) until all the epsilon moves are removed from the NFA.

Now, to explain this conversion, let us take an example.

Example: Convert epsilon-NFA to NFA. Consider the example having states q0, q1, q2, q3, and q4.

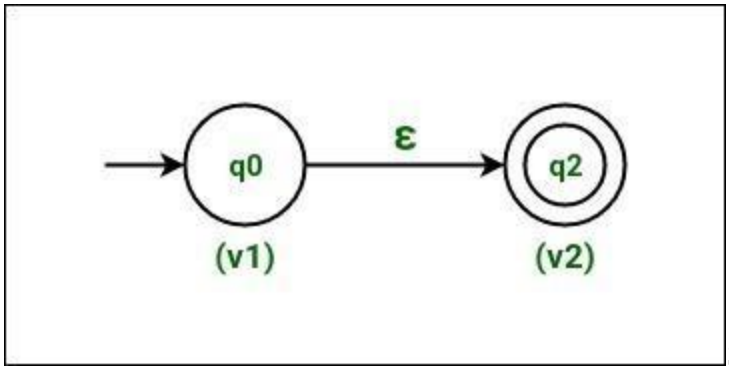


In the above example, we have 5 states named as q0, q1, q2, q3 and q4. Initially, we have q0 as start state and q2 as final state. We have q1, q3 and q4 as intermediate states.

Transition table for the above NFA is:

States/Inputs	INPUT 0	INPUT 1	INPUT EPSILON
q0	–	q1	q2
q1	–	q0	–
q2	q3	q4	–
q3	q2	–	–
q4	q2	–	–

According to the transition table above,
state q0 on getting input 1 goes to state q1.
State q0 on getting input as a null move (i.e. an epsilon move) goes to state q2.
State q1 on getting input 1 goes to state q0.
Similarly, state q2 on getting input 0 goes to state q3, state q2 on getting input 1 goes to state q4.
Similarly, state q3 on getting input 0 goes to state q2.
Similarly, state q4 on getting input 0 goes to state q2.
We can see that we have an epsilon move from state q0 to state q2, which is to be removed.
To remove epsilon move from state q0 to state q1, we will follow the steps mentioned below.
Step-1:
Considering the epsilon move from state q0 to state q2. Consider the state q0 as vertex v1 and state q2 as vertex v2.



S

Figure – State q0 as vertex v1 and state q2 as vertex v2

Step-2:

Now find all the moves that starts from vertex v2 (i.e. state q2).

After finding the moves, duplicate all the moves that start from vertex v2 (i.e state q2) with the same input to start from vertex v1 (i.e. state q0) and remove the epsilon move from vertex v1 (i.e. state q0) to vertex v2 (i.e. state q2).

Since state q2 on getting input 0 goes to state q3.

Hence on duplicating the move, we will have state q0 on getting input 0 also to go to state q3.

Similarly state q2 on getting input 1 goes to state q4.

Hence on duplicating the move, we will have state q0 on getting input 1 also to go to state q4.

So, NFA after duplicating the moves is:

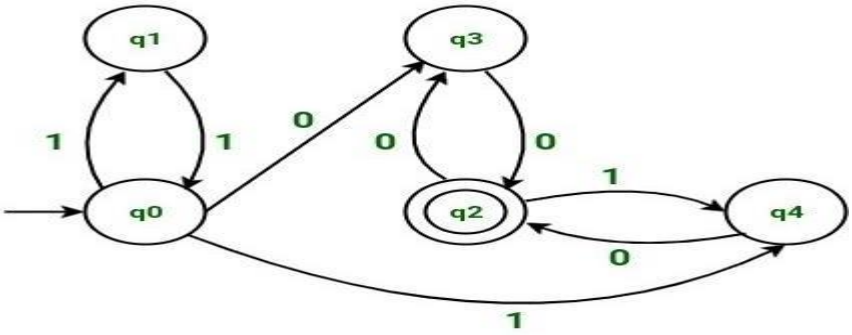


Figure – NFA on duplicating moves

Step-3:

Since vertex v1 (i.e. state q0) is a start state. Hence we will also make vertex v2 (i.e. state q2) as a start state.

Note that state q2 will also remain as a final state as we had initially.

NFA after making state q2 also as a start state is:

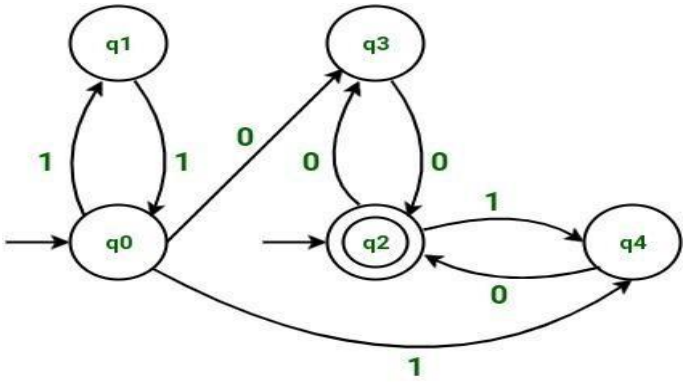


Figure – NFA after making state q2 as a start state

Step-4:

Since vertex v2 (i.e. state q2) is a final state. Hence we will also make vertex v1 (i.e. state q0) as a final state.

Note that state q0 will also remain as a start state as we had initially.

After making state q0 also as a final state, the resulting NFA is:

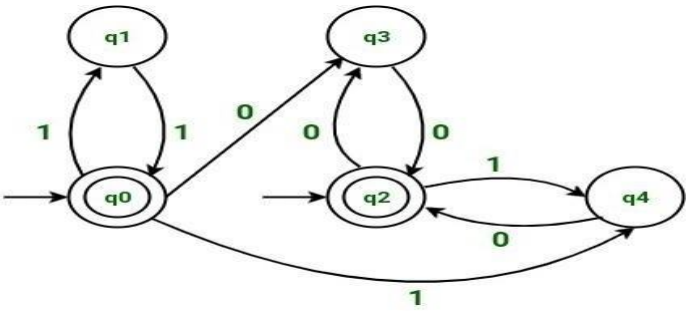


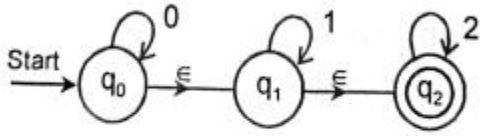
Figure – Resulting NFA (state q0 as a final state)

The transition table for the above resulting NFA is:

STATES/INPUT	INPUT 0	INPUT 1
q0	q3	q1,q4
q1	–	q0
q2	q3	q4
q3	q2	–
q4	q2	–

Example:

Convert NFA with ϵ-moves in figure given below to equivalent NFA without ϵ-moves



Solution:

From the definition of ϵ-closure,

ϵ-closure(q_0) = $\{q_0, q_1, q_2\}$

ϵ-closure (q_1) = $\{q_1, q_2\}$

ϵ-closure (q_2) = $\{q_2\}$

There q_2 is in ϵ-closure(q_0) that means without reading any i/p symbol transition directly moves from q_0 to q_2 directly. Similarly, q_2 is the only final state for given NFA with ϵ-moves

So, F of NFA with ϵ is $\{q_2\}$ and F′ of NFA without ϵ is $\{q_0, q_1, q_2\}$

$\hat{\delta}(q_0, 0)$ = ϵ-closure($\delta(\hat{\delta}(q_0, \epsilon), 0)$)

= ϵ-closure($\delta(\{q_0, q_1, q_2\}, 0)$)

= ϵ-closure($\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$) = ϵ-closure($\{q_0\} \cup \phi$)

= ϵ-closure(q_0) = $\{q_0, q_1, q_2\}$

$\hat{\delta}(q_0, 1)$ = ϵ-closure($\delta(\hat{\delta}(q_0, \varepsilon), 1)$)

= ϵ-closure($\delta(\{q_0, q_1, q_2\}. 1)$)

= ϵ-closure($\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$)

= ϵ-closure($\phi \cup \{q_1\} \cup \phi$)

= ϵ-closure (q_1) = $\{q_1, q_2\}$

$\hat{\delta}(q_0, 2)$ = ϵ-closure($\delta(\hat{\delta}(q_0, \epsilon, 2)$)

= ϵ-closure($\delta(\{q_0, q_1, q_2\}, 2)$)

= ϵ-closure (q_1) = $\{q_1, q_2\}$

$\hat{\delta}(q_1, 0)$ = ϵ-closure($\delta(\hat{\delta}(q_1, \epsilon), 0)$)

= ϵ-closure($\delta(\{q_1, q_2\}, 0)$)

= ϵ-closure (ϕ) = ϕ

The state q_1 is not consuming the i/p symbol '0' so, no need to perform this step.

$\hat{\delta}(q_1, 1)$ = ϵ-closure($\delta(\hat{\delta}(q_1, \epsilon), 1)$)

= ϵ-closure($\delta(\{q_1, q_2\}, 1)$)

$= \in\text{-closure}(q_1) = \{q_1, q_2\}$

$\hat{\delta}(q_0, 2) = \in\text{-closure}(\delta(\hat{\delta}(q_0, \in), 2))$

$= \in\text{-closure}(\delta(\{q_2\}, 1))$

$= \in\text{-closure}(\delta(q_2, 1))$

$= \in\text{-closure}(q_2) = \{q_2\}$

Similarly,

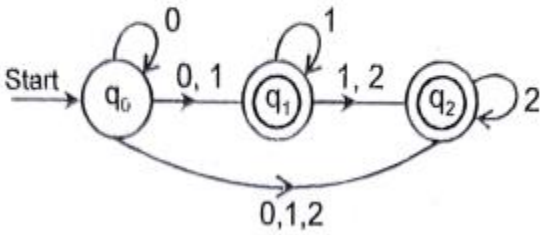
$\hat{\delta}(q_1, 2) = \{q_2\}$

$\hat{\delta}(q_2, \in) = \{\phi\}$

$\hat{\delta}(q_2, 1) = \phi$

$\hat{\delta}(q_2, 2) = \{q_2\}$

Using above information of δ' we can construct the transition diagram for NFA without $\in$.

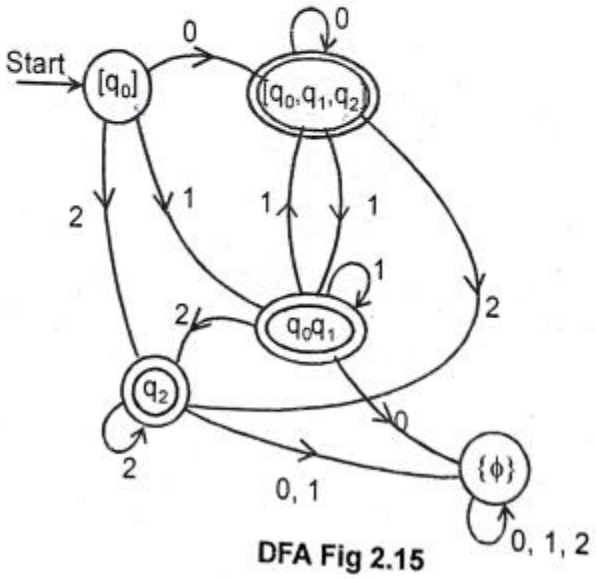


Construction of DFA from ∈-NFA:

i) Construct first NFA from NFA with ∈-moves

ii) NFA without ∈ in the above figure is converted into equivalent DFA.

The DFA is



DFA Fig 2.15

Converting Regular Language to Regular Expressions

- Regular sets| Reg lang: Recognized by FA.
- Regular Expressions
  → Mathematical representations of lang accepted by FA.
  →Regular sets are expressed in simple algebraic forms.
  → Lang accepted by FA can be represented in RE.

- Applications of RE
  → pattern matching algorithm.
  → search Engines (google, yahoo)
  → Text Editors.
  → Web programming forms.

PRACTICE QUESTIONS

Reg set | Language	Reg Exp
1. L = { }	Φ
2. L = { ε }	ε
3. L = { a }	a
4. L = {ab, ba }	ab + ba or ab| ba

5. L = { ab, ba }* U { abb }	(ab + ba)* + abb
6. L = { ab, ba } ab	(ab+ba)* aab
7. L = { ε, a, b, aa, ab, baa, bb . . . }	(a+b)*
8. L = { ε, a, b, c, aa, ab, aac, baa . . .}	(aa+b+c)*
9. ∑ = { 0 } L = { ε, 0, 00, 000 . . . }	0*
10. L = { 0, 1 }* 00	(0+1)* 00

11. Set of strings begin with 1 and Ending with 0.
RE : 1 (0+1)* 0

12. Set of strings containg in 3 consecutive OS.
RE : (0+1)* 000 (0+1)*

13. Set of all strings such that 10th symbol from the right is 1.
RE : (0+1)* 1 (0+1)^9

14. Set of all strings such that 3rd symbol from LHS is 1
RE : (0+1)^2 1 (0+1)*

15. # 1's followed by exactly 2 consecutive Os followed by # 1's.
    RE : $1^* 00\ 1^*$

16. Set of all strings containing atleast two 0's.
    RE : $1^* 01^* 0\ (\ 0+1)^*$ or $(\ 0+1\ )^* 0\ (\ 0+1)^* 0\ (\ 0+1)^*$

17. Set of all strings containing atmost two 0's
    RE : $1^* (\ 0+\varepsilon\ ) 1^* (\ 0+\varepsilon\ )^* 1$

18. Set of strings of length 2.
    RE : $(\ 0+1\ )(\ 0+1\ ) = (\ 0+1\ )^2$

19. Strings of length 2 or more
    RE : $(\ 0+1\ )(\ 0+1\ )(\ 0+1\ )^*$ or $(\ 0+1)^2 (\ 0+1\ )^*$

20. Strings of length 3 or less
    RE : $(\ 1+0+\varepsilon\ )(\ 1+0+\varepsilon\ )(\ 1+0+\varepsilon\ )$

Algebraic Laws for Regular Expressions

Given R, P, L, Q as regular expressions, the following identities hold:

1. $\emptyset^* = \varepsilon$
2. $\varepsilon^* = \varepsilon$
3. $RR^* = R^*R$
4. 4. $R^*R^* = R^*$
5. 5. $(R^*)^* = R^*$
6. $RR^* = R^*R$
7. $(PQ)^*P = P(QP)^*$

8. $(a+b)^* = (a^*b^*)^* = (a^*+b^*)^* = (a+b^*)^* = a^*(ba^*)^*$

9. $R + \emptyset = \emptyset + R = R$ (The identity for union)

10. $R\varepsilon = \varepsilon R = R$ (The identity for concatenation)

11. $\emptyset L = L\emptyset = \emptyset$ (The annihilator for concatenation)

12. $R + R = R$ (Idempotent law)

13. $L(M + N) = LM + LN$ (Left distributive law)

14. $(M + N)L = LM + LN$ (Right distributive law)

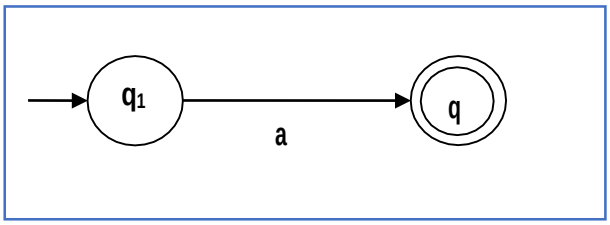
15. $\varepsilon + RR^* = \varepsilon + R^*R = R^*$

Construction of a Finite Automata from a Regular Expression

We can use Thompson's Construction to find out a Finite Automaton from a Regular Expression. We will reduce the regular expression into smallest regular expressions and converting these to NFA and finally to DFA.

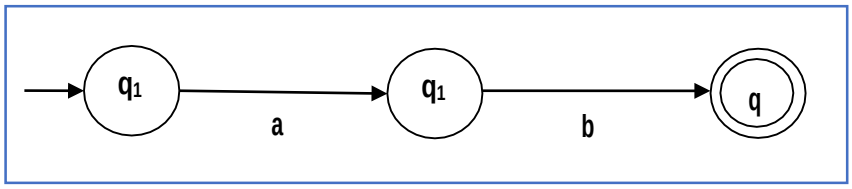
Some basic RA expressions are the following:

***Case 1*:** For a regular expression 'a', we can construct the following FA:



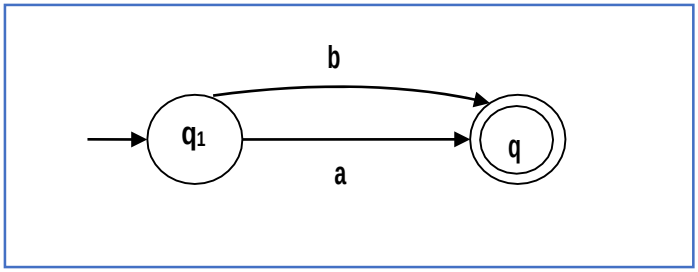
Finite automata for RE = a

***Case 2*:** For a regular expression 'ab', we can construct the following FA:



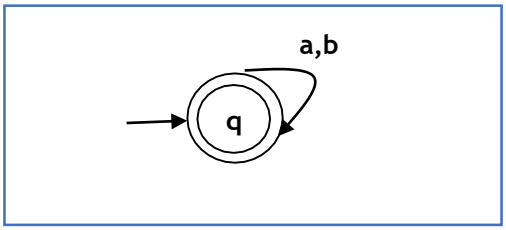
Finite automata for RE = ab

***Case 3*:** For a regular expression (a+b), we can construct the following FA:



Finite automata for RE= (a+b)

***Case 4*:** For a regular expression (a+b)*, we can construct the following FA:



Finite automata for RE= (a+b)*

Method:

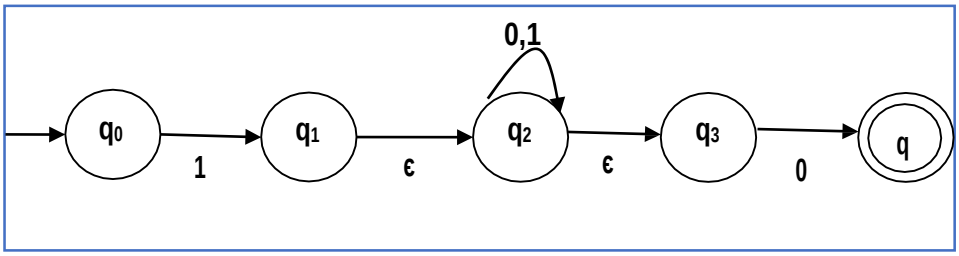
Step 1: Construct an NFA with Null moves from the given regular expression.

Step 2: Remove Null transition from the NFA and convert it into its equivalent DFA.

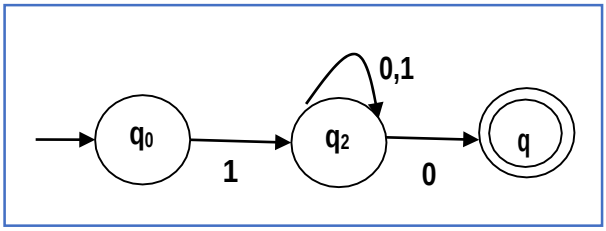
Problem **Convert the following RA into its equivalent DFA: 1 (0 + 1)* 0**

***Solution*:**

We will concatenate three expressions "1", "(0 + 1)*" and "0"



NDFA with NULL transition for RA: 1 (0 + 1)* 0



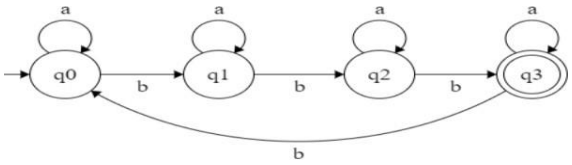
Now we will remove the **ϵ** transitions. After we remove the **ϵ** transitions from the NDFA, weget the following:NDFA without NULL transition for RA: 1 (0 + 1)* 0

It is an NDFA corresponding to the RE: 1 (0 + 1)* 0. If you want to convert it into a DFA, simplyapply the method of converting NDFA to DFA discussed in Chapter 1.

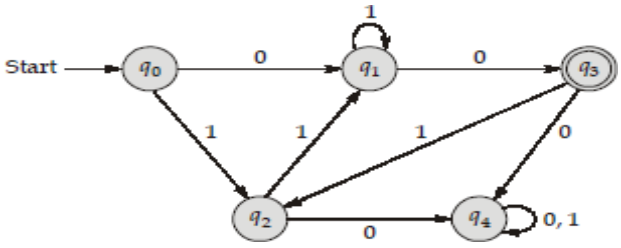
Examples:

Construct DFA for the regular expression given.

1. a (a + b) * a
2. a (a + b) * b
3. a (a + b)$^+$ a
4. a (a + b)$^+$ b
5. a*
6. a* b*
7. Construct a DFA for the regular language: ending with aab
8. Find the language accepted by the given DFA



9. Consider the following DFA.



10. Consider the following strings. Which of these below strings are accepted by the DFA?
    A. 011110 B. 101011 C. 010110 D. 1110110

==========

Pumping Lemma

- Pumping lemma is used to prove that a language is Not Regular.
- But can't be used to prove that a language is regular.
-

If L is a Regular Language, L has a pumping length P such that any word W where |w|>= p may be divided into 3 parts w = x y z then the following condition must be true.

1) x y z ε L for every i > 0, the string xy^iz is also in L
2) |y| > 0 or |y| $\neq \varepsilon$
3) |xy| < = P

To prove that a language is not regular using pumping lemma follow the steps.

- Assume that L is regular.
- The pumping length is n.
- Words of length greater than p can be pumped |w| >= n
- Divide w into x y z.
- Show that x y $\notin$ L f for some i.
- So, none can satisfy all the above 3 conditions at a time.
- So, the assumption is wrong, proof by contradiction.

1. Prove that the language L = $\{a^n b^n \mid n >= 0\}$is not regular using pumping lemma.

Proof: Assume that L = $\{a^n b^n \mid n >= 0\}$is regular

Pumping length = K

w = $a^K b^k$ → aaaaaaaabbbbbbb

Case1: The y is in the 'a' part

a a a a a a a b b b b b b b b x y z → x y^2 z

x y z a a a a a a a a a a a a b b b b b b b

11 a's ≠ 7 b's

Case2: The y is in the 'b'; part

a a a a a a a b b b b b b b x y^1 z → x y^2 z

x y z a a a a a a a b b a a b b b b b b

Not even the pattern = a^k b^k

Language L is not regular.

==========

2. Prove that the language L = {y y|y ∈{0, 1} * is not regular by using pumping lemma. Assume that L is regular

The pumping length = n.

W = 0^n 1 0^n 1

x y z k = 7

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 → x y^i z → x y^2 z

x y z 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

110's L. ∉

|y| > 0

|xy| < k Not satisfied.

Contradiction by proof. So, L is not regular.

==========

3.

RL or non-RL? → pumping lemma for RL.

For any RL, L ∃ an integer is dependent on L

Set ∀ Z ∈ L and |z| >= n

i) z = u v w then u v^i w Є L ∀ i >= 0

ii) |uv| <= |z|

iii) v >= 1 or |v| # 0. u, v, w, z are words ∈

Example 1: L = {a^m b^m | m >= 1} pumping means: u v' w

u v^2 w

u v^3 w . . . .

// counting, storing, comparing not possible

// finite lang. No problem.

Let assume L is a RL

Z Є L

Z = a^k b^k → |z| >= n, we don't know the length of n or k

u v w chooses a k s.t |a^k b^k | = 2K >= n

1.Z → a^{k-1} a b^k → length of v >= 1

2. | u v | = | a^{k-1} a | = | a^k | = k < = 2k

Check | v | ≠ 0, but 1

At first, it is assumed L is a RL → Proof by contradiction → L is non-RL

Hence uv^i w Є L ∀ I >= 0

i = 2: u v w

a^{k-1} a b^k = a^{k+1} b^k

Now it is m+1 not equal to a^m b^m L

L = {a^m b^m | m >= 1} if i changes

==========

Eg2: L = {a^m b^p | m > p}

Let assume L is a RL → by proof by contradiction assumption is incorrect. L is a non-RL

z ∈ L

z = a^{k+1} b^k choose k s.t |z| >= n

2k+1 >= n

a^{k+1} b b^{k-1}

1) |z| >= n satisfied

2) |uv| <= |z|

|v| ≠ 0,

3) Then u v^i w Є L ∀ I >= 0

Choose i = 2

u v w

a^{k+1} b^2 b^{k-1} ∈ L

Because m > p

Proof by contradiction

Hence L is non regular

==========

Eg 3: L = {a^{3n} | n >= 1}

1) z Є L

u v w

a^{3n-1} a Є → can be

but v = 1

or

v ≠ 0

2) |u v| < = | z |

3) Now check

U v w Є L ∀ I >= 0

Choose I = 2:

u v w

a^{3n-1} a^2 Є → a^{3n+1} ∉ L

because L = a^{3n} but ↑ say a's and b's are equal.
By proof by contradiction. it is not.
Special case of pumping lemma when ∑ = {a}
Singleton set.
Length of strings must follow AP for a L tube a RL.

u v^i w → a $(aa)^i$ a → {a a, a a a a, a a a a a a}
2 —4 — 6 ———

==========

Eg 4: L = $\{a^{3n} \mid n >= 1\}$ ∑ = {a}
a^3, b^6, a^9, a^{12} ..... f n = 1
RL n = 2
3 6 9 12 . . .. n = 3
Diff: 3 3 3 n = 4

Eg 6: L = $\{a^{2n+1} \mid n >= 0\}$ → RL
1, 3, 5, 7 .....AP
If n = 0
n = 1
n = 2
n = 3
1, 3, 5, 7 . . ..

==========

Eg 7: L = $\{a^{n3+n2+1} \mid n >= 0\}$ → non RL if n =0
1, 3, 13 . . . Not AP n = 1
N = 2

Eg 8: L = $\{a^{2n} \mid n >= 0\}$ Non-RL If n = 0
n = 1
n = 2
1, 2, 4, 8, 16 . . ≠ AP n = 3

Eg 9: L = $\{a^p \mid p$ is a prime$\}$ non-RL
Prime numbers 2, 3, 5, 7, 11.....≠ AP
1 2 2 4 ..... ≠ AP

==========00oo00==========