Shatt Alarab university callage - Departement of computer science

COMPUTATIONAL THEORY

Seconed stage

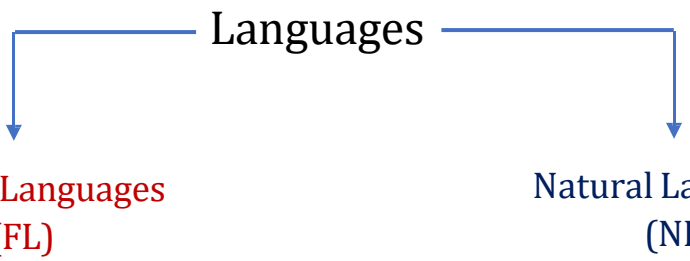
Scholastic Year

2020-2021

Lecture (1)
Formal Languages

Head Lines

- *Alphabet*
- *Sentences or strings*
- *Rules*
- *Languages*



The FL is a set of strings. In turn, a string is a finite sequence of letters from some alphabet.

The main characteristics are:

- It doesn't care about meaning of sentences.
- It has a syntax system only.
- Examples: programming languages , sets , strings , …..
- Sentences : (C++ PL) " cout>>X ". (Set) "00 , 01 , 10 , 11"

The Nl is the language of the declaiming and understanding of humans.

The main characteristics are:

- Each sentence should have a meaning to belong for the language.
- It has two systems, syntax and semantic systems.
- Examples: Arabic , English , Japanese ,….
- Sentences : (English) "Ahmad is going to the school"

- ***Alphabet (∑)***

Definition (Alphabet): An alphabet Σ is any finite set. We often write Σ = {a1, . . . , ak}.

The ai are called the symbols of the alphabet.

Examples: Σ = {a} , Σ = {a, b, c} , Σ = {0, 1} , Σ = {α, β, γ, δ, ϱ, λ, ϕ, ψ, ω, μ, ν, ρ, σ, η, ξ, ζ }

The alphabet of a formal language consist of symbols, letters, or tokens. So, the alphabet of *L* is the set of all symbols that may occur in any string in *L*. For example, if *L* is the set of all variable identifiers in the programming language C, *L*'s alphabet is the set { a, b, c, ..., x, y, z, A, B, C, ..., X, Y, Z, 0, 1, 2, ..., 7, 8, 9, _ }.

- ***String***

The symbols of alphabet that concatenate into strings of the language. Each string concatenated from symbols of this alphabet is called a word, (or sentence) and the words that belong to a particular formal language are sometimes called well-formed words or well-formed formulas.

For example, using the binary alphabet {0,1}, the strings ε, 0, 1, 00, 01, 10, 11, 000, …}

Example: if ∑={a,b,c} then some of language strings are {abc, aaba, baacc, aabc, …..}

- ***Rule and Language***

Rule is the written condition that is defined over an alphabet to describe a language.

Example:

Let ∑={a,b} and the rule is (all strings has 3 letters length }

Then Language (L) ={aaa, aab, aba, baa, abb, bba, bab, bbb}

Example:

Let ∑={0,1}

Rule={Each word begin with 00 and end with 11 and has 6 letters}

Then L={000011, 000111, 001011, 001111}

Example:

∑={1,x,2,y}

Rule=(Each words has all alphabet symbols in one recurrence, and begin with number)

L={1x2y, 12xy, 1xy2, 12yx, 1y2x, 1yx2, 21xy, 21yx, 2x1y, 2xy1, 2y1x, 2yx1}

Note: Empty string denoted by ε or ^ , is belongs to any formal language, the length of empty word is 0.

If ∑={} then L={ø}

Example:

∑={0,1}

Rule=(all words has odd '1' and even '0}

L={1 001 00111 010 11100 1101000 …..}

Lecture (3)

Formal Languages

Head Lines

- *Regular Expressions and formal languages*

Regular Expressions (RE) and formal languages(FL)

A regular Expression(denoted it by RE) is a mathematical notation of the linguistic operations to describe a formal language.

Closure Properties of Regular Expression

Regular Expressions are used to denote regular languages. An expression is regular if:

- RE=ϕ is a regular expression for regular language ϕ.
- RE=ε is a regular expression for regular language {ε}.
- If a ∈ Σ (Σ represents the input alphabet), RE=a is regular expression with language {a}.
- (Union property) If a and b are regular expression, RE=a + b is also a regular expression with language {a,b}.
- (concatenation property) If a and b are regular expression, RE=ab (concatenation of a and b) is also regular.
- (power closer property) If a is regular expression, RE=a^x (x times a) is also regular.

- (plus closer property) If a is regular expression, RE=a^+ (1 or more times a) is also regular.
- (Kleene closer) If a is regular expression, RE=a* (0 or more times a) is also regular.

Note: Kleene closer property = plus closer property + ε

Definition (regular Language (RL)): A language is regular if it can be expressed in terms of regular expression.

Closure Properties of Regular Languages

Union : If L1 and If L2 are two regular languages, their union L1 ∪ L2 will also be regular. For example, L1 = $\{a^n \mid n \geq 0\}$ and L2 = $\{b^n \mid n \geq 0\}$

L3 = L1 ∪ L2 = $\{a^n \cup b^n \mid n \geq 0\}$ is also regular.

Intersection : If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular. For example,

L1= $\{a^m b^n \mid n \geq 0$ and $m \geq 0\}$ and L2= $\{a^m b^n \cup b^n a^m \mid n \geq 0$ and $m \geq 0\}$

L3 = L1 ∩ L2 = $\{a^m b^n \mid n \geq 0$ and $m \geq 0\}$ is also regular.

Concatenation : If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular. For example,

L1 = $\{a^n \mid n \geq 0\}$ and L2 = $\{b^n \mid n \geq 0\}$

L3 = L1.L2 = $\{a^m . b^n \mid m \geq 0$ and $n \geq 0\}$ is also regular.

Kleene Closure : If L1 is a regular language, its Kleene closure L1* will also be regular. For example,

L1 = (a ∪ b)

L1* = (a ∪ b)*

Complement: If L(G) is regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L(G) from all possible strings. For example,

L(G) = {a^n | n > 3}

L'(G) = {a^n | n <= 3}

Some RE Examples

Regular Expressions	Regular Set
(0 + 10*)	L = { 0, 1, 10, 100, 1000, 10000, … }
(0*10*)	L = {1, 01, 10, 010, 0010, …}
(0 + ε)(1 + ε)	L = {ε, 0, 1, 01}
(a+b)*	Set of strings of a's and b's of any length including the null string. So L = { ε, a, b, aa , ab , bb , ba, aaa…….}
(a+b)*abb	Set of strings of a's and b's ending with the string abb. So L = {abb, aabb, babb, aaabb, ababb,.....}
(11)*	Set consisting of even number of 1's including empty string, So L= {ε, 11, 1111, 111111,.....}
(aa)*(bb)*b	Set of strings consisting of even number of a's followed by odd number of b's , so L = {b, aab, aabbb, aabbbbb, aaaab, aaaabbb,.....}
(aa + ab + ba + bb)*	String of a's and b's of even length can be obtained by concatenating any combination of the strings aa, ab, ba and

	bb including null, so L = {aa, ab, ba, bb, aaab, aaba, ……..}

Example:

Describe the formal language that can be defined by the following expression:

RE=$0^{*} + 1^{+}$?

L={^ 0 00 000 …. 1 11 111 …….}

Example:

Describe the formal language that can be defined by the following expression:

RE=(0+1)* ?

L={^ 0 1 00 11 01 10 11 000 001 010 011 100 110 111 ……}

General Grammar's Questions

Q1: Let G1 is a grammar,
G1=({a,b,c},{S,T,O},S,P) where p:

S→aSa | Tbb | Occ | ^ , Tb→ac , Oc→bc

Then:

1- Describe the G1 language ?

2- Determine five sentential forms and two directly drives for G1?

3- Classify G1 by using chomsky' hierarchy ?

- Q2: Let G2 is a grammar, G1=({0,1},{S,L},S,P) where p:
- S→00S | 01L | 11S | ^ , L→0 | 1 | 10S

Then:

- 1- Describe the G2 language ?
- 2- Determine five sentential forms and two directly drives for G2?
- 3- Classify G2 by using chomsky' hierarchy ?

Question: Describe the formal language that can be defined by:

$\sum=\{a,b,c\}$

R.E.=$a^*+b^+(ab)^2$

Solution:

R.E.=RE1+RE2+RE3

RE1→ a^* , L={ ^, a, aa, aaa, aaaa,…}

RE2→b^+ , L={b, bb, bbb, bbbb,….}

RE3→$(ab)^2$, L={abab}

RE= $a^*+b^+(ab)^2$, L={^, a, aa, aaa, aaaa,…, b, bb, bbb, bbbb,….,abab}

Question: Describe the formal language that can be defined by

$\sum=\{0,1\}$

RE=$(01^*+10^*)^+$

RE1=01^* , L={0, 01, 011, 0111,….}

RE=10^* , L={1, 10, 100, 1000,….}

RE=$(01^*+10^*)^+$ →1,2,3,…

L={0,1,00,11,01,10,0101,1010,0110,000,111,0110111100,….}

Note: Two regular expressions are equivalent if they describe the same formal language.

Example: Show if the following expressions are equivalent or not:

RE1=$(0+1)^*$

RE=$(01)^*$

L(RE1)={^, 0,1,00,01,10,11,000,001,010,011,100,101,110,111,0000,..}

L(RE2)={^,01,0101,010101,01010101,…}

Then the RE1 is not equivalent to the RE2.

H.W. If you have the RE1=$aa^*+ b^+$, then choose the equivalent regular expression(s) ,for RE1from following Res:

1) RE=a^++b^+

2) RE=a$(a^+$+$b^+)^+$

3) RE= a$(a+b)^+$

Theorem (Kleen's star): If ∑={x}, Then $\sum^* = \sum^{**} = \sum^{***} = \ldots$

Example: Let ∑={0}, Then

$\sum^*=\{0\}^*=\{\wedge, 0, 00, 000, 0000, \ldots\}$

$\sum^{**}=\{0\}^{**}=\{\wedge, \wedge\wedge, \wedge\wedge\wedge,\ldots,0,00,000,0000,\ldots,00,0000,000000,\ldots, 000,000000, 00000000, \ldots\}$ then, after reduce words redundancy, we get the following words set:

= {^, 0,00,000,0000,…} = $\sum^*$

And so on($\sum^{***}$, $\sum^{***}$, ….)

Proof:

All words of the set ∑ are belonging in the set $\sum^*$, and all words of the set $\sum^*$ are belonging in the set $\sum^{**}$, and so on for $\sum^{***}$,$\sum^{****}$,….

→ $\sum \subseteq \sum^* \subseteq \sum^{**} \subseteq \sum^{***}$ ….

And $\sum^{***} \subset \sum^{**} \subset \sum^* \subseteq \sum$

Then $\sum^* = \sum^{**} = \sum^{***} = \ldots$

Evaluating the RE from the formal language (Set of words)

To evaluate RE from set of words, We should notice the following points:

1- The similar parts in all words.
2- The redundancy in all words.
3- The used alphabet in the words.

Example: Evaluate the RE for the following formal language:

L={011, 0101, 01001, 010001, 0100001,…}

The similar (constant) parts are in the beginning and ending of each word, and the redundancy part in the middle.

L={01^1, 0101, 01001, 010001, 0100001,…}

Then RE=010^*1

Example: Evaluate the RE for the following formal language:

L={ab, c, aab, cd, aaab, cdd, aaaab, cddd,….}

The constant parts in all words are: b in the end of some words, and c in the begin of others.

L={ab, c, aab, cd, aaab, cdd, aaaab, cddd,….}

RE=$a^+b + cd^*$

H.W.

Evaluate the Res of each language in the following:

1- L={^, ab, abab, ababab,…}
2- L={010, 101, 0110, 1101, 01110, 11101,…
3- L={02, 102, 022, 11002, 0222, 1110002,…}

Product of sets

If we have two sets of strings(words) S and R , then we can define the set of product S*R and R*S as in the following:

S*R : {all words that first part belong in the S, and second part belong in the R}, this definition is correct on R*S.

Example: If S={0110, 01, 11, 101} , and R={a, ab, bba}

Then compute S*R and R*S

S*R={0110a, 01a, 11a, 101a, 0110ab, 01ab, 11ab, 101ab,0110bba, 01bba, 11bba, 101bba}

R*S={a0110, a01, a11, a101, ab0110, ab01, ab11, ab101, bba0110, bba01, bba11, bba101}

From above example, we show S*R ≠ R*S

Example: If X={^, x, xx}, and Y={yy, yyy}, then compute X*Y and Y*X

X*Y={yy, yyy, xyy, xyyy, xxyy, xxyyy}

Y*X={yy, yyx, yyxx, yyy, yyyx, yyyxx}

X*Y≠Y*X

H.W:

If X={^, 0, 00, 000}, and Y={^, 1, 11, 111}, then compute X*Y and Y*X?

Example: describe the language that can be defined by ∑={0,1,2}, and:

1- RE1=$0^2 1^3 2^2$
2- RE2=$0^i 1^j 2^k$, where i= odd number less than 8, j=i-1,k=i+j
3- RE3=$01^{odd} + 12^{even} + 0^x$, where $x=2^2+1$

L1={0011122}

L2={02 , 0001122222 , 000001111222222222 , 00000001111112222222222222}

L3={01, 0111 ,011111 ,…, 122, 12222, 1222222,…., 00000}

Grammars

A grammar is a quadruple component. It's forming from four finite sets:

- A finite set of symbols called an alphabet(small letters), and denoted it by (∑).
- A finite set of capital letters called variable (V).
- Subset of variable set (V) is called start variable (S).
- A finite set of production rules (P), in form : α → β , where α,β ϵ $(\sum U V)^*$

So, G=(∑ , V , S , P)

Note: Each word must be derived from derivation, begin with start variable and end with the word. In this case the word is belonging in the formal language that describe by the grammar.

Example: If G=({a},{S,A},{S},P)

Where p: S→a , S→ aA , aA → aa , aA → S

Then describe the language that can be defined by G?

1- S → a (a word)
2- S → aA → aa (a word)
3- S → aA → S → a

So, L={a , aa }

Example: If G=({0,1},{S,A,B},{S},P) , where P :

S →0A , S →1B , 0A →00S , 0A → 0 , 1B→11S, 1B→1

Then describe the language that can be define by G?

1- S→0A→00S→000A→000 (a word)
2- S→0A→0 (a word)
3- S→1B→1 (a word)
4- S→1B→11S→111B→111 (a word)
5- S→0A→00S→001B→001 (a word)

So, L={0,000,1,111,001,….}

Question : Let G=({a,b},{X,Y,Z},{Z},P) , where P:

Z→aXb , Z→aaYbb , Xb→ aa , aX→aZ , YB→bb , aY→aZ

Then describe the formal language that can be defined by G?

Headlines

- *Examples of the general grammars*
- *Definitions (sentential forms, directly drives, derivation)*

Let $G_1 = (\{0,1\}, \{S,T,O,I\}, S, P)$, where P contains the following productions

$$\begin{aligned} S &\rightarrow OT \\ S &\rightarrow OI \\ T &\rightarrow SI \\ O &\rightarrow 0 \\ I &\rightarrow 1 \end{aligned}$$

As we shall see, the grammar G_1 can be used to describe the set $\{0^n1^n | n \geq 1\}$.

Let $G_2 = (\{0,1,2\}, \{S,A\}, S, P)$, where P contains the following productions

$$\begin{aligned} S &\rightarrow 0SA2 \\ S &\rightarrow \epsilon \\ 2A &\rightarrow A2 \\ 0A &\rightarrow 01 \\ 1A &\rightarrow 11 \end{aligned}$$

This grammar G_2 can be used to describe the set $\{0^n1^n2^n \geq n \geq 0\}$.

Example (grammar in English natural language)

To construct a grammar G3 to describe English sentences, one might let the alphabet ∑ comprise all English words rather than letters. V would contain nonterminals that correspond to the structural components in an English sentence, such as <sentence>, <subject>, <predicate>, <noun>, <verb>, <article>, and so on. The start symbol would be <sentence>. Some typical productions are:

<sentence> → <subject><predicate>
<subject> → <noun>
<predicate> → <verb><article><noun>
<noun> → mary
<noun> → algorithm
<verb> → wrote
<article> → an

The rule <sentence> → <subject><predicate> models the fact that a sentence can consist of a subject phrase and a predicate phrase. The rules <noun> → mary and <noun> → algorithm mean that both "mary" and "algorithm" are possible nouns. This approach to grammar, stemming from Chomsky's work, has influenced even elementary-school teaching.

Definitions

A sentential form

A sentential form of G is any string of terminals and nonterminals, i.e. a string over ∑U V .

The directly derives

Let G=(∑,V,S,P) be a grammar, and let γ_1, γ_2 be two sentential forms of G. We say that :

γ_1 directly derives γ_2, written $\gamma_1 \Rightarrow \gamma_2$, if $\gamma_1 = \sigma\alpha\tau$, $\gamma_2 = \sigma\beta\tau$, and $\alpha \rightarrow \beta$

Is a production in P.

For example, the sentential form $00S11$ directly derives the sentential form $00OT11$ in grammar G_1, and $A2A2$ directly derives $AA22$ in grammar G_2.

A derivation

Let γ_1 and γ_2 be two sentential forms of a grammar G. We say that :

γ_1 drives γ_2: written $\gamma_1 \Rightarrow^* \gamma_2$ if there exists a sequence of (zero or more) sentential forms $\sigma_1, \ldots, \sigma_n$ such that :

$$\gamma_1 \Rightarrow \sigma_1 \Rightarrow \cdots \Rightarrow \sigma_n \Rightarrow \gamma_2.$$

The sequence $\gamma_1 \Rightarrow \sigma_1 \Rightarrow \cdots \Rightarrow \sigma_n \Rightarrow \gamma_2$ is called a **derivation** of γ_2 from γ_1.

For example, in grammar G_1, $S \Rightarrow^* 0011$ because

$$S \Rightarrow \underline{O}T \Rightarrow 0\underline{T} \Rightarrow 0S\underline{I} \Rightarrow 0\underline{S}1 \Rightarrow 0\underline{O}I1 \Rightarrow 00\underline{I}1 \Rightarrow 0011$$

and in grammar G_2, $S \Rightarrow^* 001122$ because

$$S \Rightarrow 0\underline{S}A2 \Rightarrow 00\underline{S}A2A2 \Rightarrow 00\underline{A}2A2 \Rightarrow 0012\underline{A}2 \Rightarrow 0011\underline{A}22 \Rightarrow 001122.$$

A Subject Examples of the conversion from RE into RG

Q1/ Find the regular grammar that equivalent to the regular expression (RE=01100^{+}) ?

Solution/

G=({0,1},{S,A},S,P)

S→0110A , A→0A | 0

Q2/ Find the regular grammar that equivalent to the regular expression (RE=abc^{*}) ?

Solution/

G=({a,b,c},{S,A},S,P)

S→abA , A→cA | ^

Q3/ Find the regular grammar that equivalent to the regular expression (RE=$(01)^{*} + 1^{+}$) ?

Solution/

G=({0,1},{S,A},{S,A},P)

S→01S | 1A | ^ , A→1A | ^

Q4/ Find the regular grammar that equivalent to the regular expression (RE=$00110^{*} + 1101^{+}$) ?

Solution/

G=({0,1},{S,A,B},{S,A},P)

S→0011A | 110B | ^ , A→0A | ^ , B→1B | 1

Q5/ Find the regular grammar that equivalent to the regular expression (RE=$(a+b)^+$?

Solution/

G=({a,b},{S},S,P)

S→aS | bS | a | b

Q6/ Find the regular grammar that equivalent to the regular expression (RE= $(0+1)^*$ 101+ bba^+ ?

Solution/

G=({0,1,a,b},{S,A},S,P)

S→0S | 1S | 101 | bbA , A→aA | a

Finite Automata

Definition: A Finite Automata(FA) (also called a finite state machine FSM) is a graph consists of six finite sets:

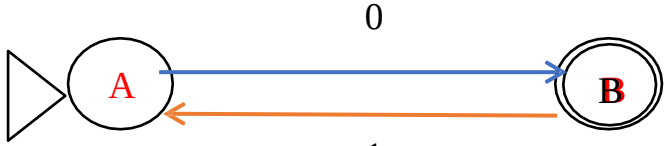
- An alphabet ($\sum$) is a finite set of terminal symbols which are labels of transitions.
- A finite set of state is represented in the graph by circles ◯, each circle labeled alphabetically or numerically.
- A finite set of transitions is represented in the graph by arrows → Each arrow is labeled by alphabet symbol.
- A subset of states set called starting states are remarked by ▷

  Or (-).
- A subset of states set called final states are remarked by double circles ◎ or (+).

Note: Each FA described a formal language. And it considered as word or sequence recognition.

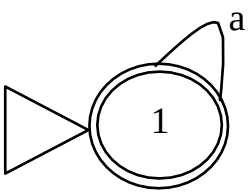
Note: sequence is recognized by a FA if it starts with the starting state and ends with the final state.

Example: a FA represents a RE=$0(10)^*$

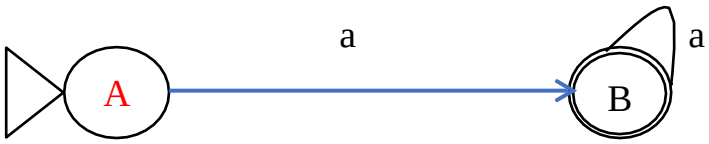


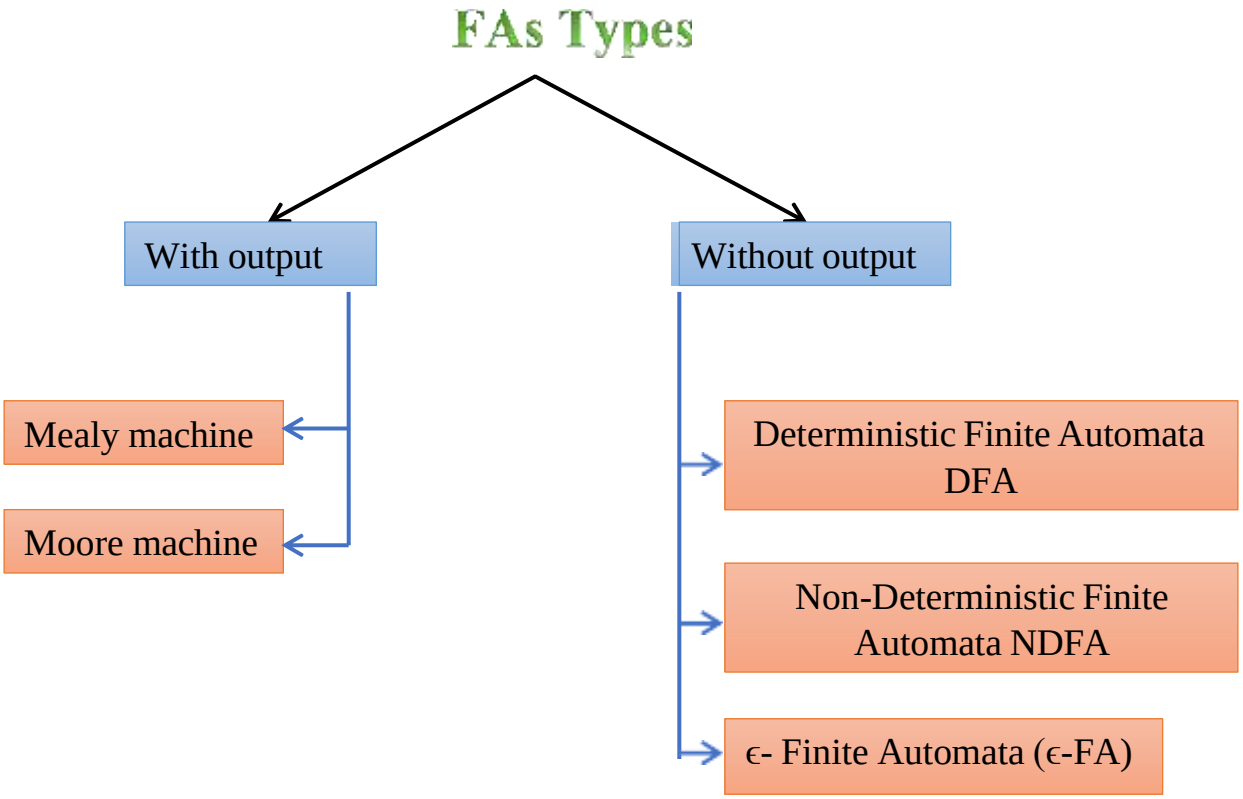
FA

e: a FA represents the RE=a^*



Example: a FA represents the RE= a^+

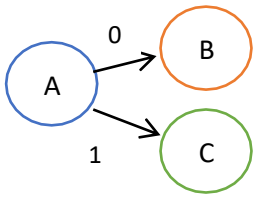




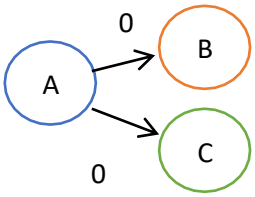
Finite Automata:

Two categories of Finite Automata:

- Deterministic Finite Automata (DFA)
  - The machine can exist in only one state at any given time
- Non-deterministic Finite Automata (NFA)
  - The machine can exist in multiple states at the same time

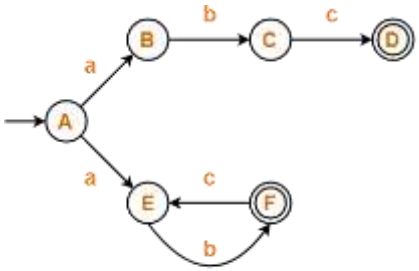


DFA



NFA

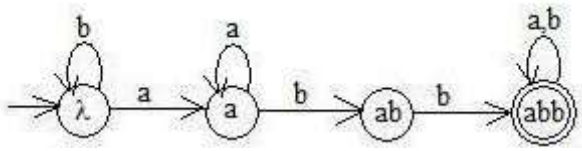
Example:



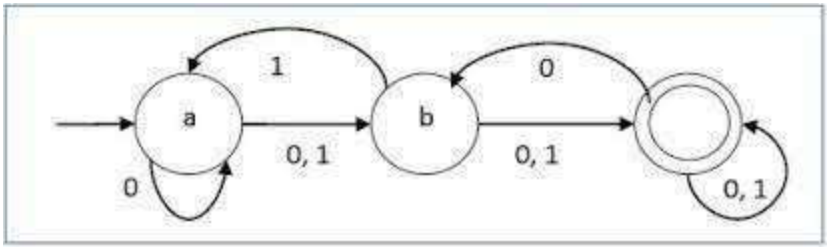
Example of Non-Deterministic Finite Automata

(Without Epsilon)

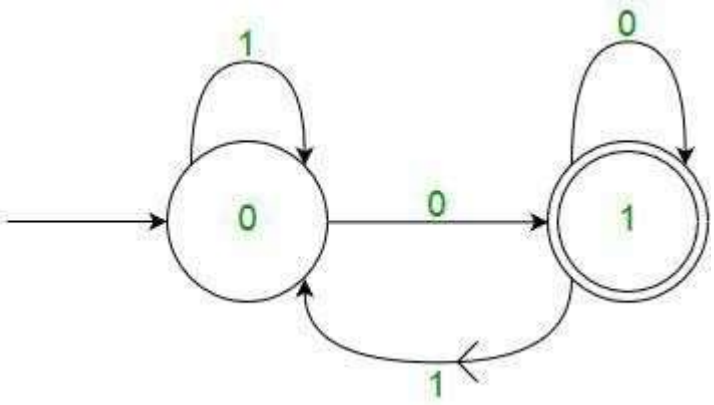
Example



Example



Example (Describe the formal language by using FA)



- First word (0) the path of states is → 01
- Second word (10) the path of states is →001
- Third word (110) the path of states is → 0001
- And consequently, for other words.

So, the language L={0,10,110,…}

Q: Show if the sequence "1100011010" is recognized via the above machine?

Answer:

We can follow the sequence of states for the bits sequence that should be starting with the start state:

bit	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**
state	0	0	1	1	1	0	0	1	0	1

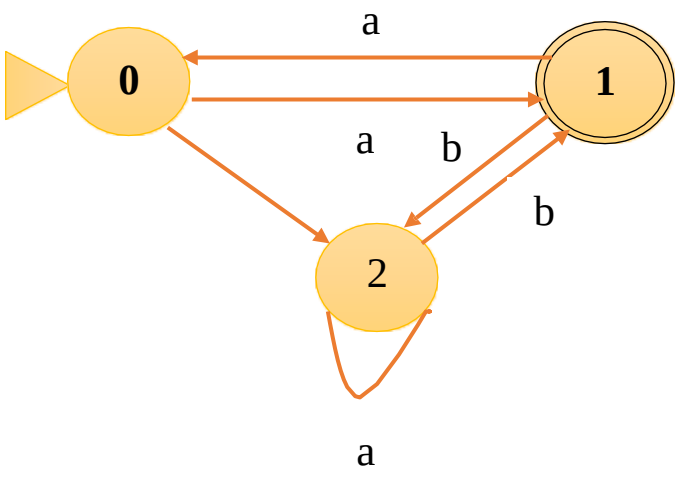
And the last state, in the table, which the sequence ends with it, it's a final state, so the bits sequence is recognized via FA.

Question:

Show if the following sequences are recognized via the following machine or not?

1-"aaaabbaabb"

2-"bbbaabaaab"



DFA

Solution:

1-"aaaabbaabb"

bit		a	a	a	a	b	b	a	a	b	b
state	0	1	0	1	0	2	1	0	1	2	1

And the last state, in the table, which the sequence ends with it, it's a final state, so the bits sequence is recognized via FA.

abaaab"

bit		b	b	b	a	a	b	a	a	a	b
state	0	2	1	2	2	2	1	0	1	0	2

And the last state, in the table, which the sequence ends with it, it's not a final state, so the bits sequence isn't recognized via FA.

Transition Table (TT)

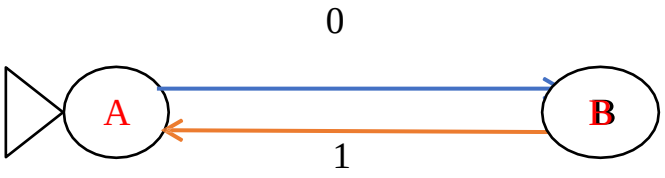
The transition table describes the transitions of states with each input alphabet symbols, the head of rows represents states and the head of columns represents the alphabet symbols.

Example/ for the DFA in the above:

Σ / S	a	b
0-	1+	2
1+	0-	2
2	2	1+

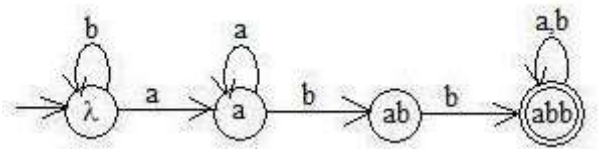
TT

Example/ write the transition table for the following DFA:



Σ / S	**0**	**1**
A-	B+	-
B+	-	A-

Example/ write the transition table for the following FA:



Σ / S	**a**	**b**
λ-	a	λ-
a	a	ab
ab	-	abb+
abb	abb+	abb+

TT

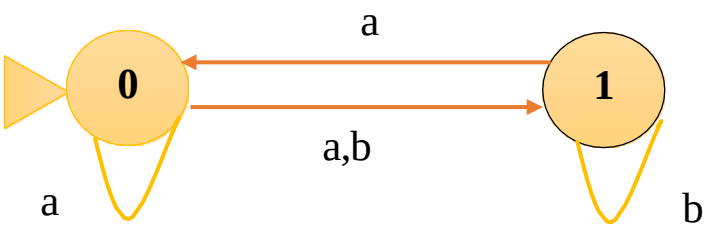
Note: We can draw the FA graph by using transition graph.

Convert a NFA into a DFA

Three steps to convert a NFA into DFA:

- Find the T.T. of the NFA.
- Evaluate the new state(s) and it transitions in the table.
- Draw the DFA and eliminate the death parts in the graph(that not describing words).

Example: Convert the following NFA into DFA



NFA

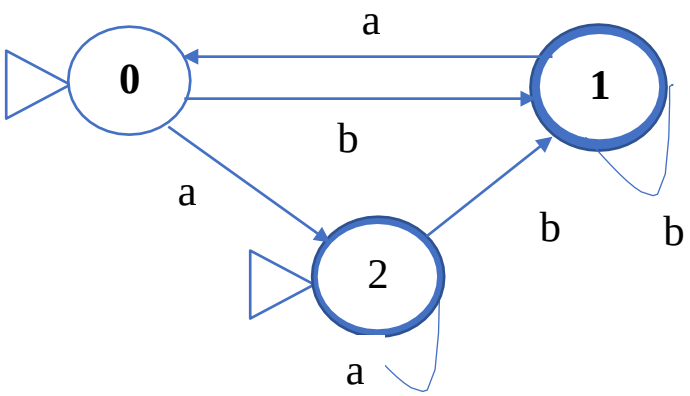
Find the T.T. and evaluate the new states:

New state

Σ / S	a	b
0-	(0,1)	1
1+	0-	1+
(0,1)	(0,1)	1

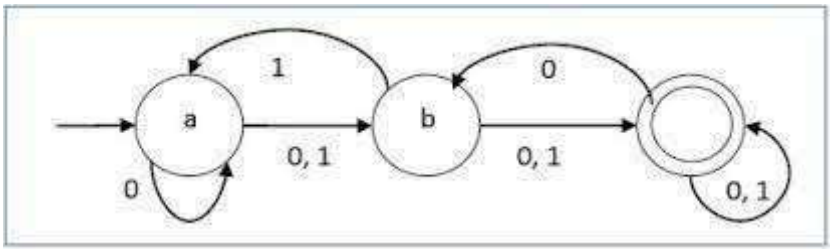
Rename to the state (2) and remark it as start and final state(because o
the state 0 is start and the state 1 is final)

Draw the DFA:



DFA

Example: convert the following NFA into DFA



NFA

Find the T.T. and evaluate the new states:

S \ Σ	0	1	
a-	(a,b)	b	
b	c	(a,c)	
c+	(b,c)	c	
(a,b)	(a,b,c)	(a,b,c)	In tl e
(b,c)	(b,c)	(a,c)	first addition
(a,c)	(a,b,c)	(b,c)	
(a,b,c)	(a,b,c)	(a,b,c)	In tl e addition

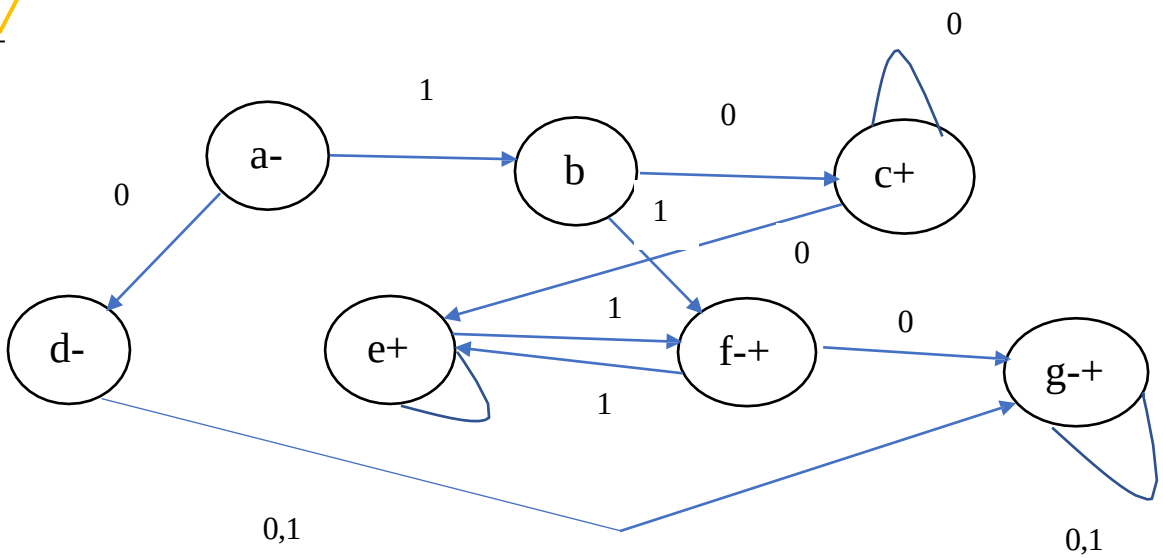
d-

e+

f-+

g-+

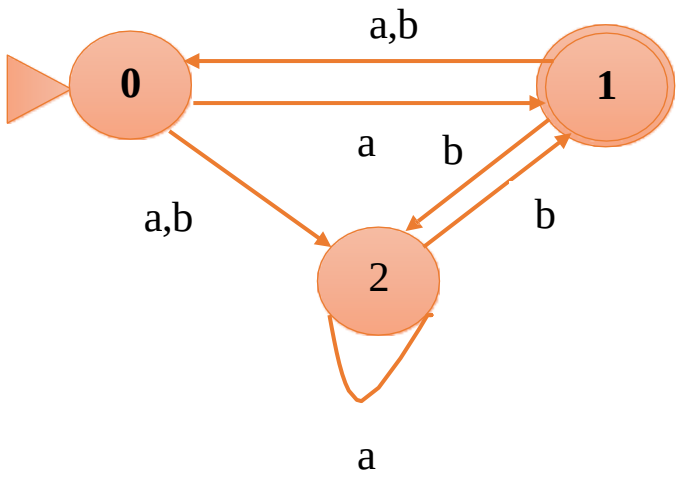
Draw the DFA



DFA

QUESTIONs:

Q1/Convert the following NFA into DFA?



NDFA

Solution:

New states

First step: Finding the T.T. and evaluating the new states.

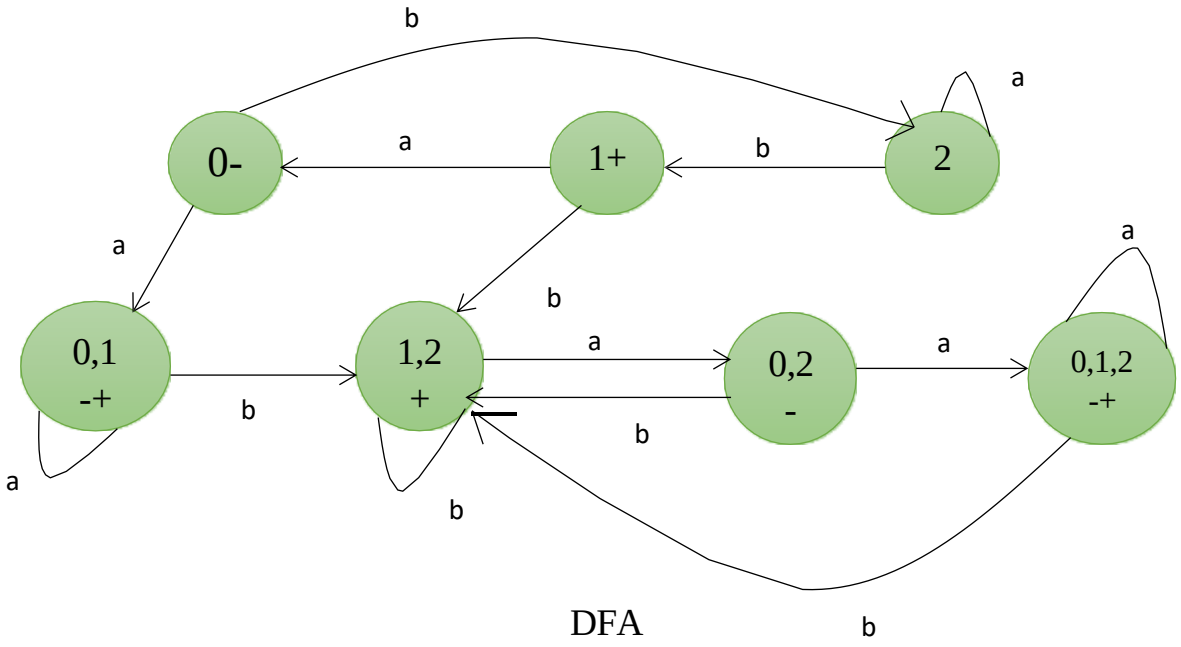
S \ Σ	a	b
0-	(0,1)	2
1+	0	(1,2)
2	2	1
(0,1)-+	**(0,1)**	**(1,2)**
(1,2)+	**(0,2)**	**(1,2)**
(0,2)-	**(0,1,2)**	**(1,2)**
(0,1,2)-+	**(0,1,2)**	**(1,2)**

New state

New state

T.T.

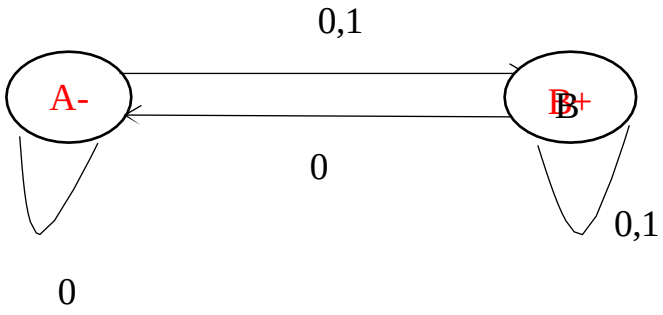
step: draw the automata



DFA

Note: There's no passive part that does not produce words (does not accept strings).

Q2/Convert the following NFA into DFA?



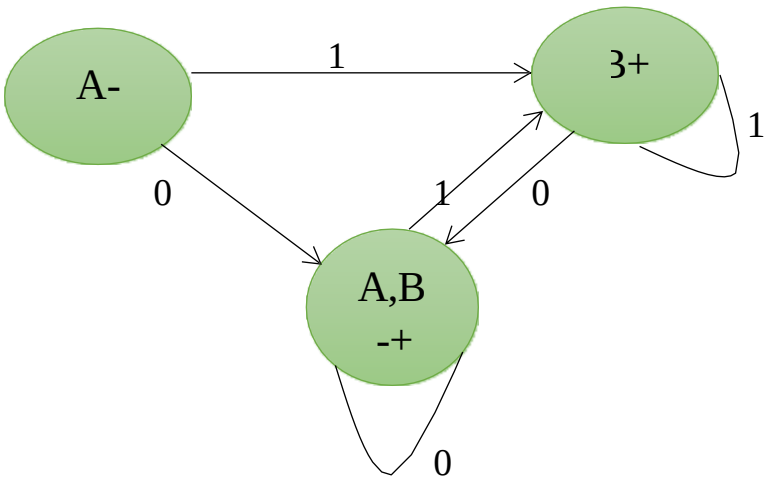
First step: Evaluate the T.T.

S \ Σ	**0**	**1**
A-	A,B	B
B+	A,B	B
A,B-+	A,B	B

T.T.

New state

Second step: Draw the automata



DFA

Note: There's no passive part that does not prod uce words (does not accept strings).

Transition Graphs

Transition graph can be interpreted as a flowchart for an algorithm recognizing a language. A transition graph consists of three things:

1. A finite set of states, at least one of which is designated the start state and some of which are designated as final states.
2. An alphabet Σ of possible input symbols from which the input strings are formed.
3. A finite set of transitions that show the change of state from the given state on a given input.

A successful path through the transition graph is a series of edges forming a path beginning at the start state and ending at one of the final states.

Definition of a Transition Graph

A transition graph is defined by a 5-tuple:

•A finite set of states, Q.

•A finite set of input symbols, Σ.

•A non-empty set set of start states, S ⊆ Q.

•A set of final or accepting states F ⊆ Q.

•A finite set, δ of transitions, (directed edge labels) (u, s, v), where u, v ∈ Q and s ∈ Σ

The Language Accepted by a Transition Graph

•Let A = (Q, Σ, S, F, δ) be a transition graph.

•A successful path in A is one that starts in some start state and ends in some accepting state of A.

•Let P be the set of all successful paths in A.

•Let L be the set of words that are the concatenation of the sequence of edge labels of A corresponding to some successful path in A.

• The language accepted by A, denoted L(A) = L.

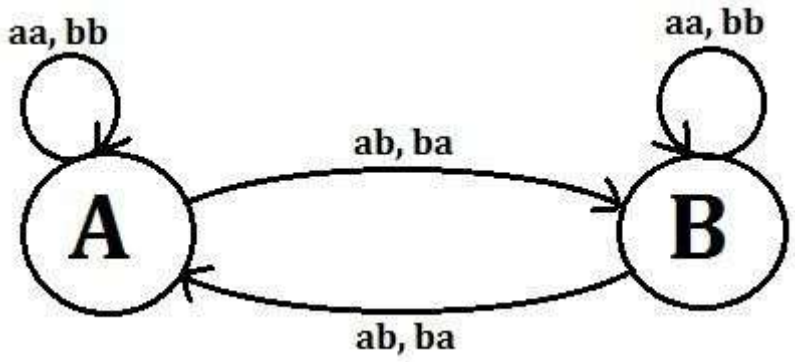
Some Observations

•If there is no factoring of a word w that is the concatenation of edge labels of a successful path in A, then w ∉ L(A)

•Every finite automaton can be viewed as a transition graph.

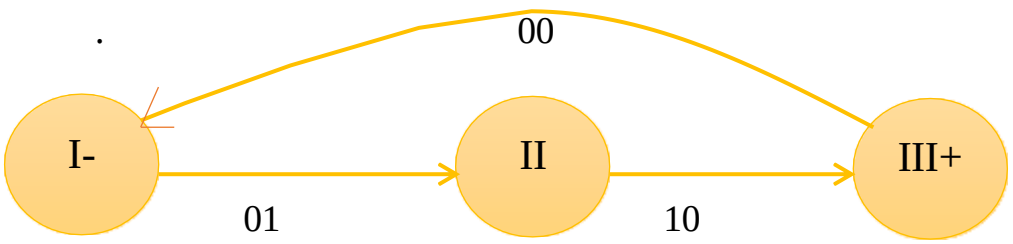
• Since the reverse is not true, transition graphs generalize finite automata.

GTG examples:

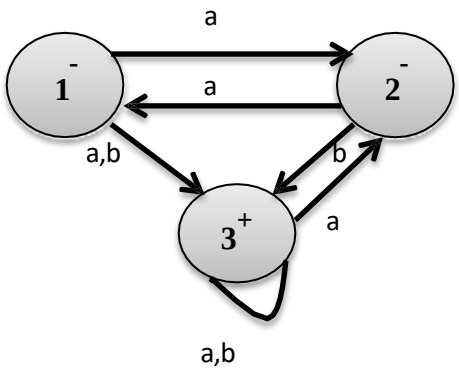


*

.



مثال : (التحويل من DFA الى NDFA) حول الماكنة من نوع NDFA أدناه الى DFA :



NDFA

الخطوة الاولى:انشاء الجدول الانتقالي الذي يعبر عن الماكنة NDFA .

States \ ∑	a	B
1 -	2,3	3
2 -	1	3
3 +	2,3	3

الخطوة الثانية: أضافة وحساب الانتقالات للحالات الجديدة المزدوجة الجديدة التي تظهر بالجدول الانتقالي.(وفي المثال ظهرت الحالة 2،3).

States \ ∑	a	b
1 -	2,3	3
2 -	1	3
3 +	2,3	3
2,3	1,2,3	3

	a	b
2	1	3
3	2,3	3
2,3	1,2,3	3

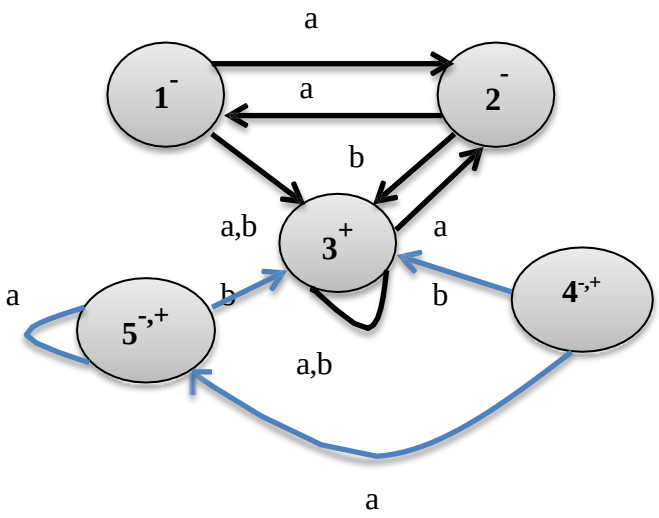
- اضافة الحالة الجديدة التي ظهرت (1,2,3) الى الجدول الانتقالي.

			b
	1^-	2,3	3
	2^-	1	3
	3^+	2,3	3
$4^{-,+}$	2,3	1,2,3	3
$5^{-,+}$	1,2,3	1,2,3	3

	a	b
1	2,3	3
2	1	3
3	2,3	3
1,2,3	1,2,3	3

- لم تظهر حالة جديدة ، نقوم بتأشير حالات البداية والنهاية الجديدة ، ونعيد تسميتها لتتناسق مع الترقيم المتبع.
(كل حالة جديدة تشترك فيها حالة بداية فهي حالة بداية ، وكل حالة جديدة تشترك فيها حالة نهاية فهي حالة نهاية)

الخطوةالثالثة: اعادة رسم الماكنة الناتجة .



DFA

- تقييم الماكنة (الرسم) الناتجة ، اذا كان هناك جزء ميت (لا يوصف كلمات :اي مسار لا يبدأ بحالة بداية ولا ينتهي بحالة نهاية) . الماكنة الناتجة لا يوجد بها جزء ميت ، اذا هي ماكنة DFA المنشودة.

** ** ** ** ** ** ** ** ** ** ** ** **

المخطط الانتقالي Transition Graph (TG)

وهي ماكنة محددة تتألف من خمسة مجاميع محددة، وهي:

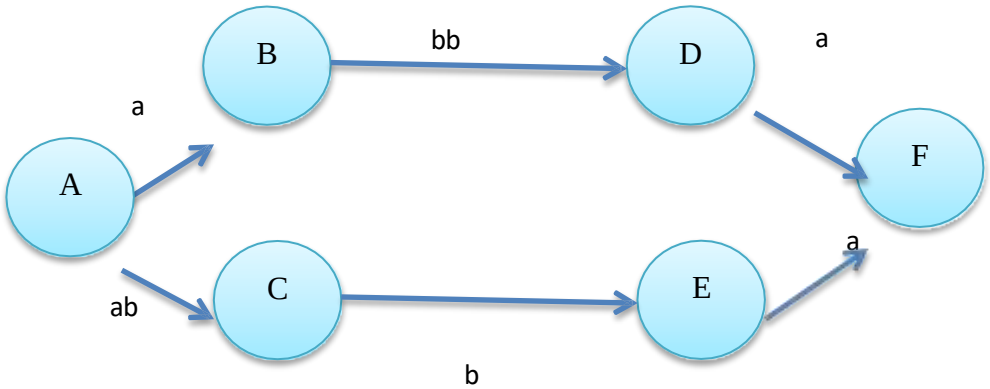
1) مجموعة محدودة من الحالات States تمثل بدوائر داخل المخطط ، وترمز ترميزا ابجديا اورقميا ويكتب داخل الدائرة.
3) مجموعة محدودة من الرموز تكون رموز أبجدية الادخال ∑ ، وهي رموز صغير او ارقام تؤلف السلاسل المعنونة لأسهم الانتقالات بين الحالات.
2) مجموعة جزئية من مجموعة الحالات تؤشر على انها حالات بداية ، تبدأ منها عملية انتاج كلمات اللغة ، وتؤشر بالمخطط اما بعلامة (-) او وضع مثلث على الحالة.
4) مجموعة جزئية من مجموعة الحالات تؤشر على انها حالات نهاية ، تنتهي عندها عملية توصيف كلمات اللغة وتؤشر بالمخطط اما بعلامة (+) او وضع دائرة مضاعفة على الحالة.
5) مجموعة محدودة من قواعد الانتقال بين الحالات ، تؤشر على شكل اسهم موجهة يتم بها الانتقال من حالة الى اخرى اعتمدا على سلسلة رمزية من ابجدية الادخال.

ملاحظة: الفروقات بين FA و TG :

1) تتألف عناوين الاسهم في FA من رمز واحد من رموز الابجدية ، اما عناوين الاسهم في TG فتتألف من سلسلة رمزية من واحد فأكثر من رموز الابجدية. لذلك تعتبر FA هي مجموعة جزئية من TG.

TG

3) يمكن توصيف الكلمة نفسها في TG بأكثر من طريق بينما هذا غير ممكن في FA



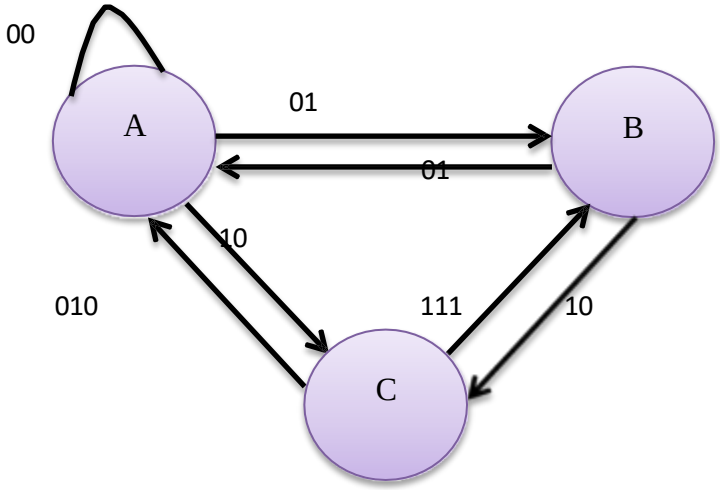
مثال TG يوصف الكلمة abba بطريقين مختلفين .

مثال : ارسم المخطط الانتقالي المصف ادناه:

TG=({A,B,C},{0,1},{A,B},{B,C},δ)

Where δ are:

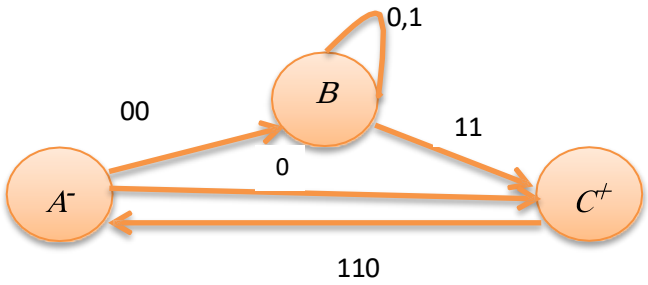
(A,00)=A , (A,01)=B , (A,10)=C , (B,01)=A , (B,10)=C , (C,010)=A , (C,111)=B



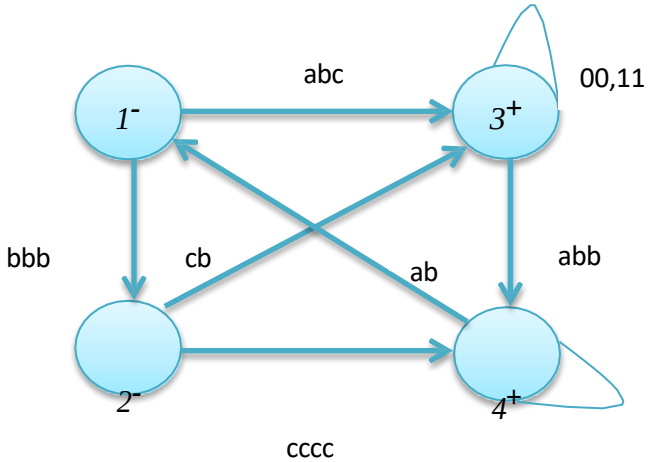
TG

مثال 1:

TG



مثال 2:

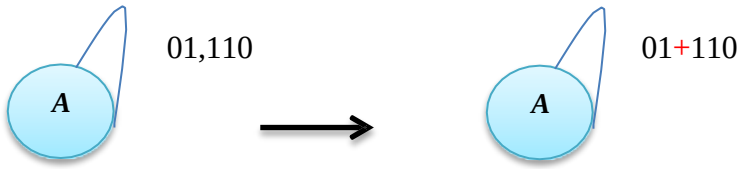


a,c

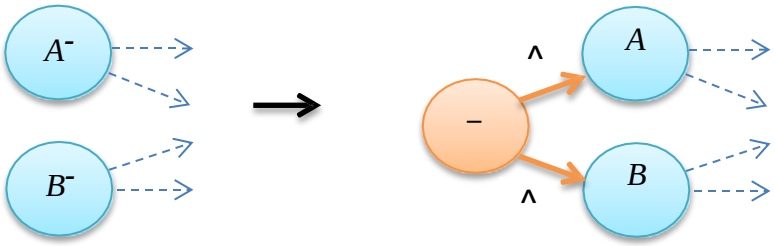
كيفية حساب تعبير نظامي Regular Expression لمخطط انتقالي Transition Graph

وتوصف بأنها خوارزمية أنشائية Constructive Algorithm، حيث نقوم من خلالها بأنشاء التعبير النظامي المكافئ للمخطط الانتقالي ، وتتكون من خطوات أربع رئيسية:

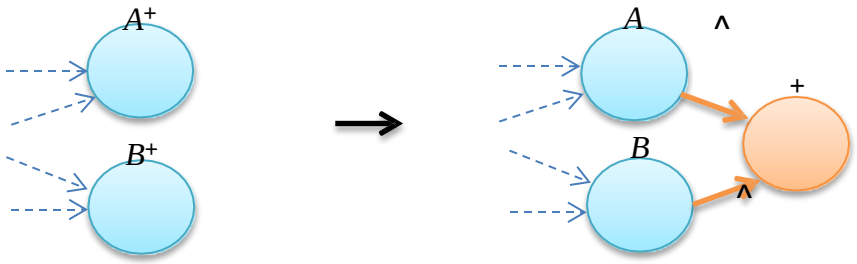
- تغيير العلامة (,) التي تربط ترميزات أدلة الاسهم (الانتقالات) في المخطط بالعلامة(+) اينما وجدت .



- وهذه الخطوة لا تنفذ اذا كان المخطط يحتوي على حالة بداية واحدة. أذا كان المخطط يحتوي على أكثر من حالة بداية واحدة ، فنقوم بتوحيد حالات البداية من خلال أضافة حالة بداية وربطها بحالات البداية للمخطط بواسطة أسهم(انتقالات) تحمل دليل (^) رمز الفراغ .



- وهذه الخطوة لا تنفذ اذا كان المخطط يحتوي على حالة نهاية واحدة. أذا كان المخطط يحتوي على أكثر من حالة نهاية واحدة ، فنقوم بتوحيد حالات النهاية من خلال أضافة حالة نهاية وربط حالات النهاية بها من خلال أسهم(انتقالات) تحمل دليل (^) رمز الفراغ .



- أختزال حالات وانتقالات المخطط بحيث يعوض الاختزال بأنتقالة (سهم) مباشر من الجزء السابق الى الجزأ اللاحق. والتوقف في حالة الوصول الى حالة بداية واحدة وحالة نهاية واحدة، ويكون الدليل الموجود على السهم الرابط بينهما هو التعبير النظامي المكافئ للمخطط الانتقالي .

حالات التعويض ان وجدت بالمخطط:

A)

01 11

A *B* *C*

A *C*

101

01 11

A *B* *C*

$01(101)^*11$

A *C*

B)

00

A *B*

11

00+11

A *B*

C) 10 $00(10)^*11$

00

A *B*

11

A

D) 00 $((00)^*+(2)^*+(1)^*)^*$

A

2

1

A

E) 22 $00(22)^*11$

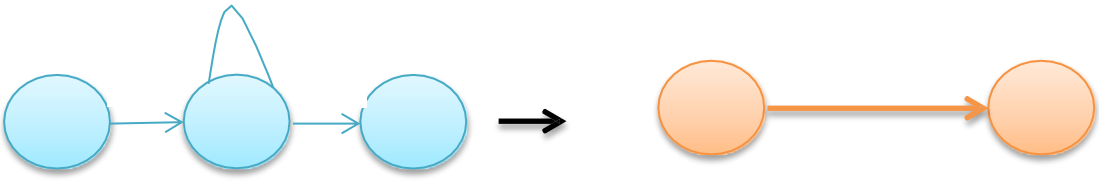
00 33

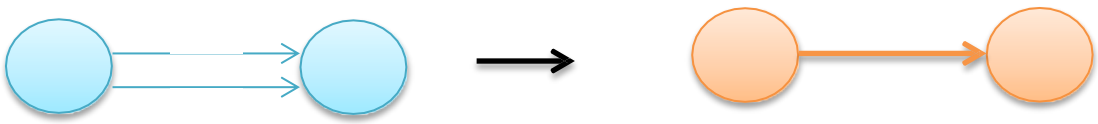
A *B* *C*

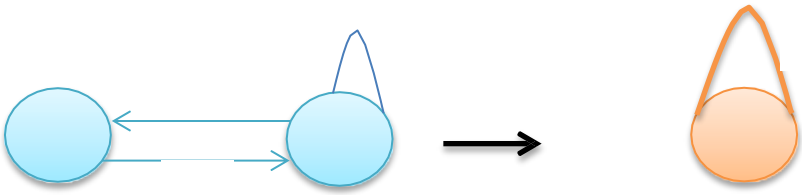
11

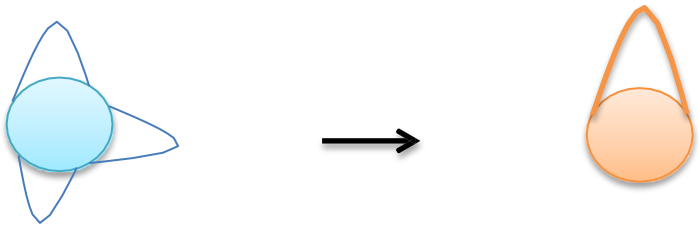
00(22)*33

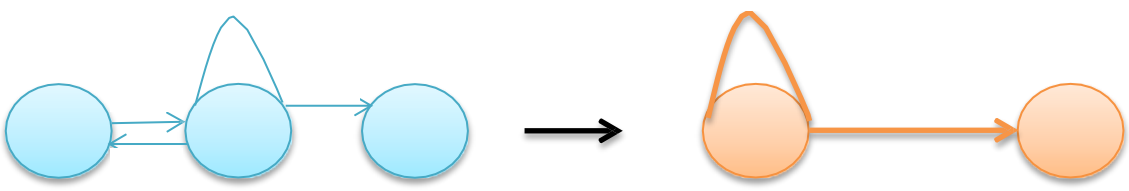
A *C*





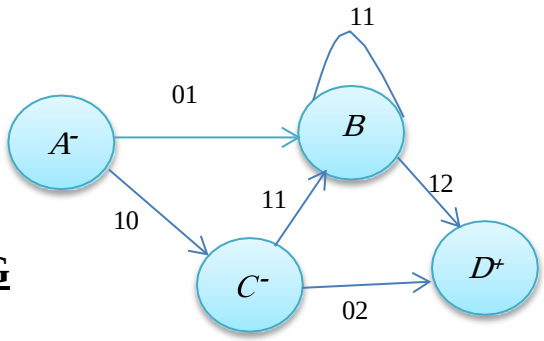




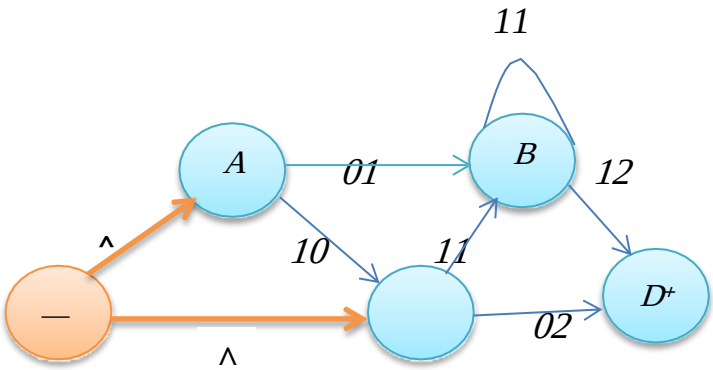


مثال : جد التعبير النظامي الذي يكافئ المخطط الانتقالي الاتي:

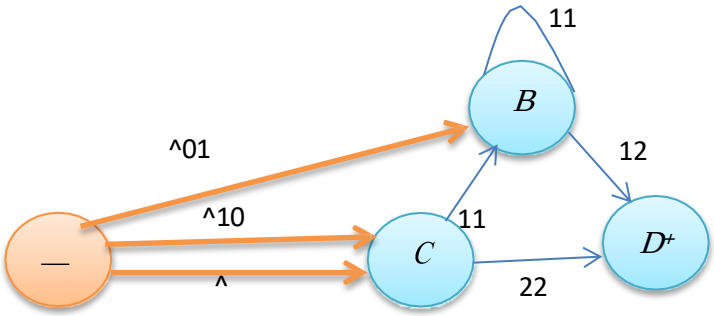
TG



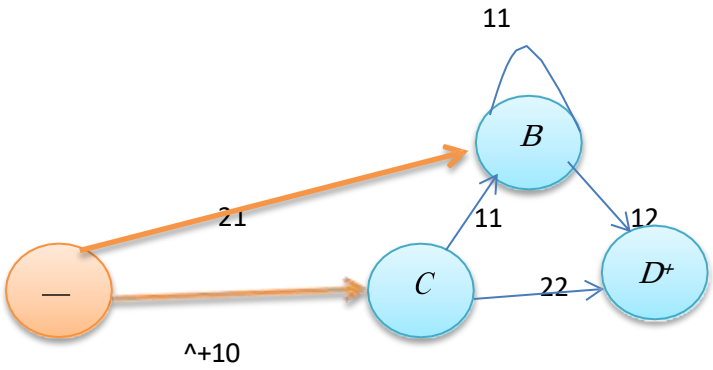
- بما ان لدينا حالتين بداية ، نقوم بتوحيدهما وذلك بخلق حالة بداية واحدة وربطها بهما بالرمز null ، والغاء التأشير من حالات البداية القديمة.



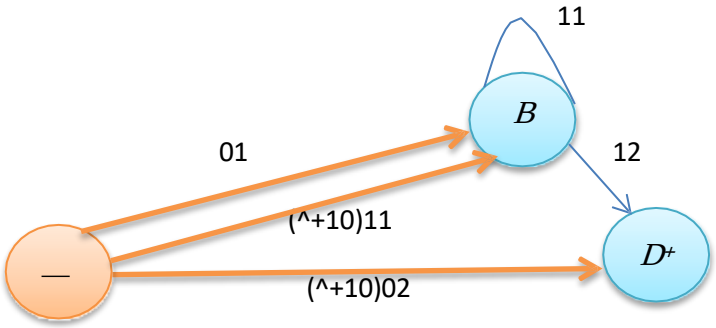
- لدينا حالة نهاية واحد اذا ليست هناك حاجة لتنفيذ خطوة توحيد حالات النهاية.
- نقوم بالاختيار لاي حالة من الحالات سيتم اختزالها(ما عدا حالتي البداية والنهاية)،سنختار الغاء الحالة *A* (ممكن *C* او *B*) ، هناك مسارين يجب تعويضهما عند الغاء الحالة *A* هما *_AC* , *_AB* :



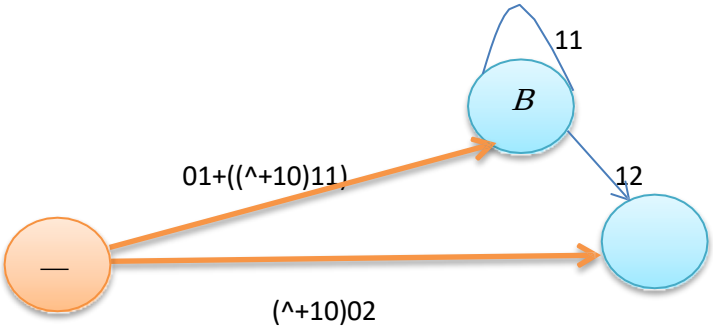
- توحيد الاسهم المتوازية من حالة البداية الى الحالة *C* :



* نختار الحالة C لكي نقوم بألغائها ، يتم تعويض مسارين هما $CD\_$, $CB\_$ بمسارين مباشرين.

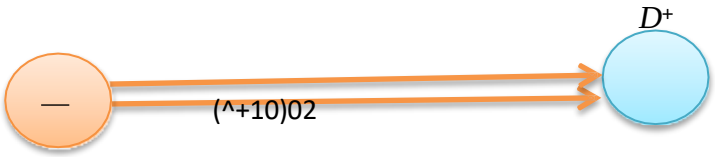


- توحيد المسارين الممتدين من حالة البداية الى الحالة B بمسار واحد:



- الحالة الاخيرة المهيئة للالغاء هي الحالة *B* وتعويض المسارات بمسار واحد من حالة البداية الى حالة النهاية:

$$(01+((\wedge+10)11)(11)^{*}\ 12$$

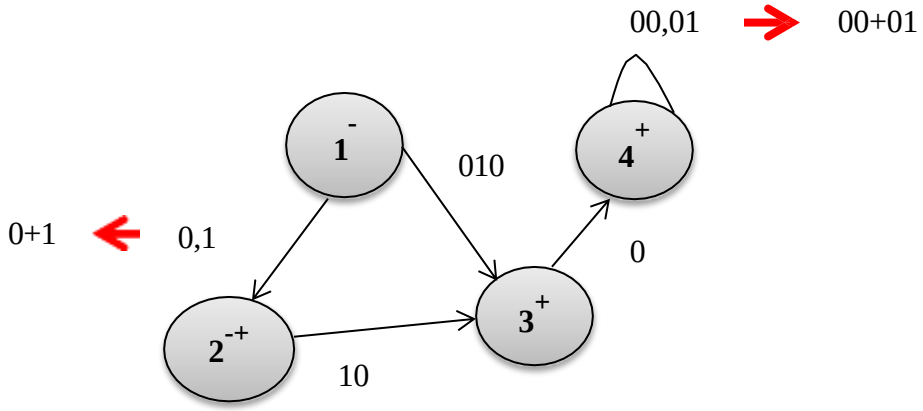


- توحيد المساران المتوازيان اللذان يربطان حالة البداية بحالة النهاية بمسار واحد.

$$((01+((\wedge+10)11)(11)^{*}\ 12) + ((\wedge+10)02)$$

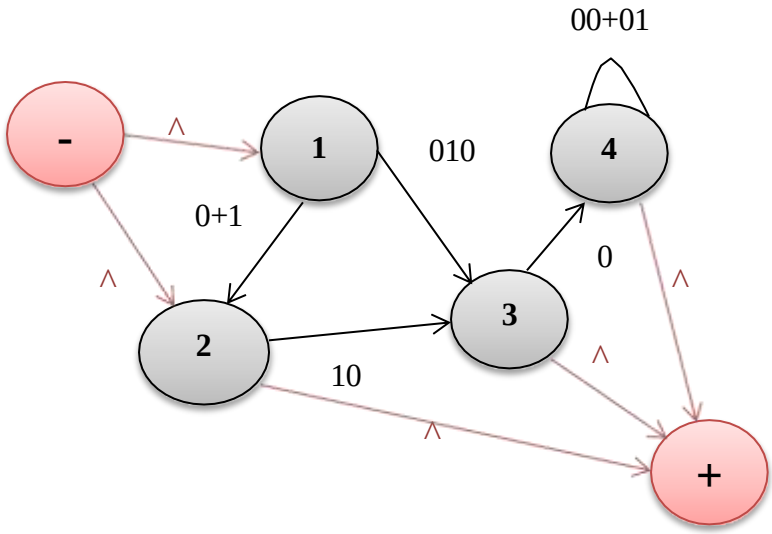
Then the R.E.= $((01+((\wedge+10)11)(11)^{*}\ 12) + ((\wedge+10)02)$

Q/ By using the constructive algorithm, evaluate the RE that equivalent to the following TG:



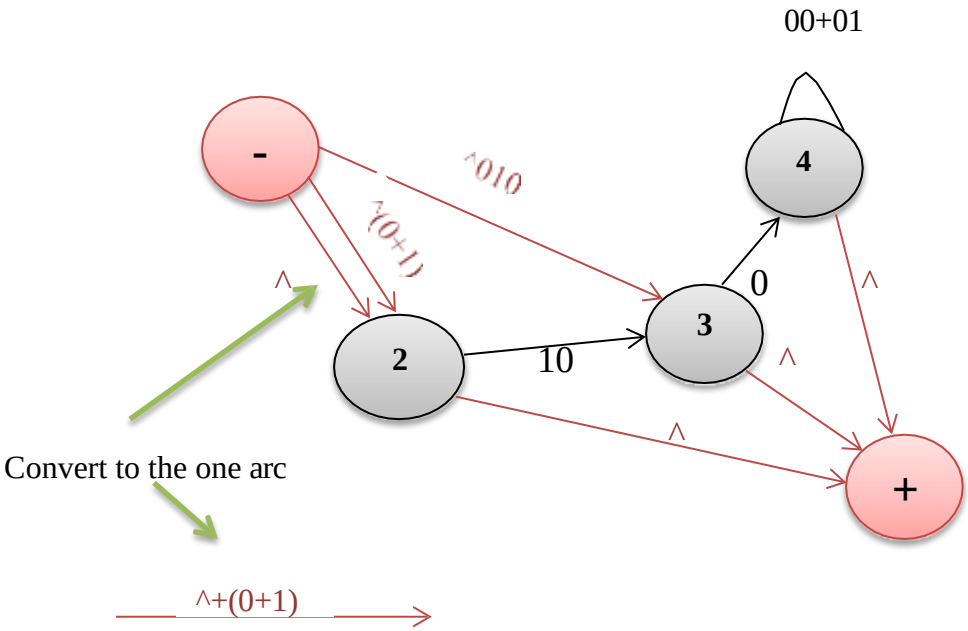
First step: convert each (,) sign into (+).

Second step: Because of there are more than one start state and more than final state, we should add a new one start and one final states.

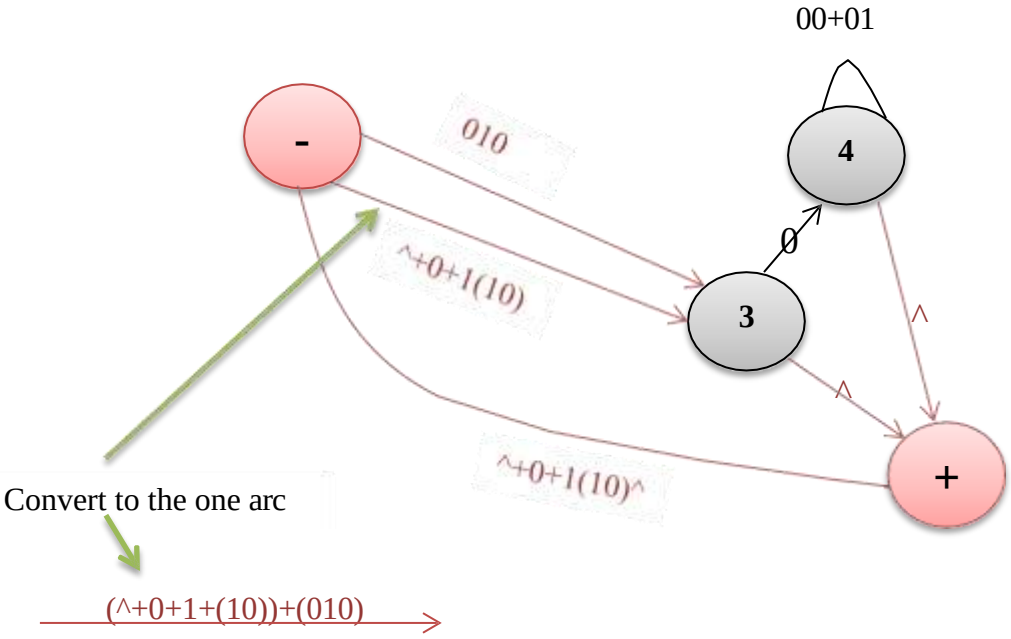


Third step: (Eliminate steps) repeat the step that eliminating state(s) or arc(s) (or both) in each step until remains the start and final state.

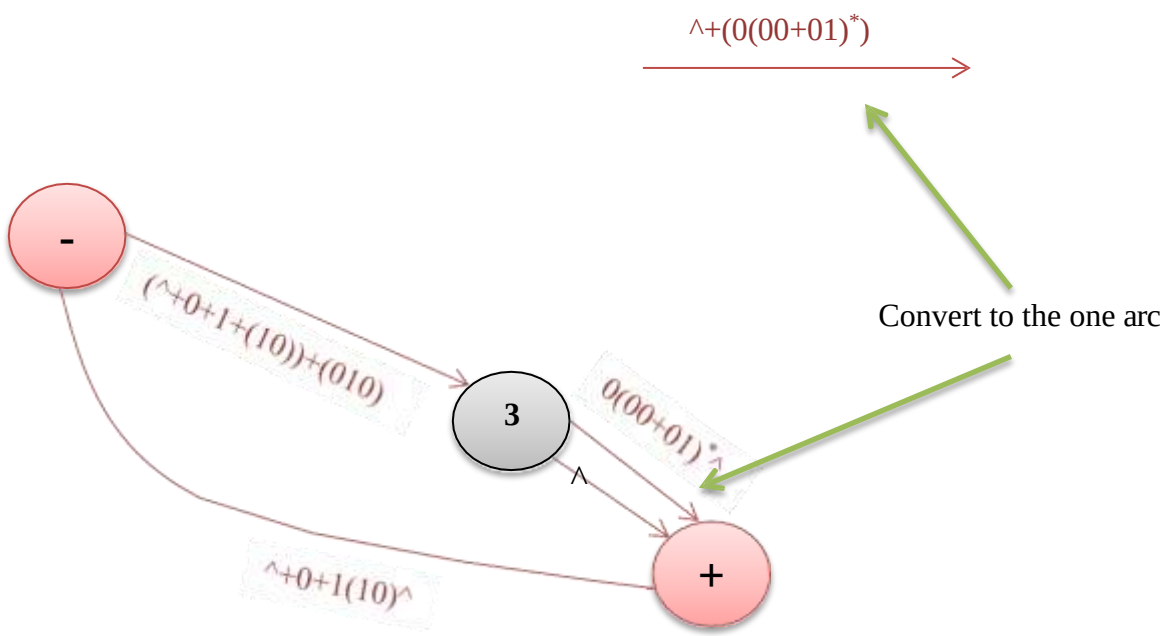
Eliminate state(1): there are two paths we will make up for it: (-,1,2) and(-,1,3)



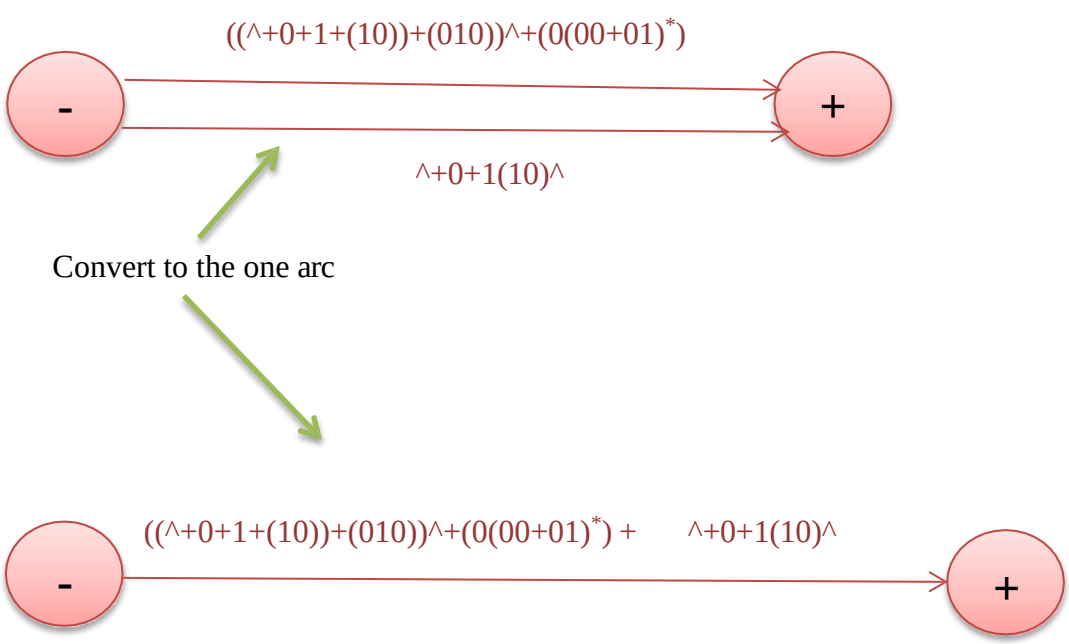
Eliminate state(2): there are two paths we will make up for it: (-,2,3)and(-,2,+):



Eliminate state(4):



Eliminate state(3):



RE= ((^+0+1+(10))+(010))^+(0(00+01)*) + (^+0+1(10))

Finite Automata with output

Goal: designing a mathematical model for a computer where the input string represents the program and data. The state of the machine could be its output. Machines should be capable of doing more.

A Finite State Machine with output is a 5-tuple $(Q, \sum, Z, \delta, \lambda)$

Q: is finite nonempty set of states

$\sum$: is finite nonempty set of input symbols

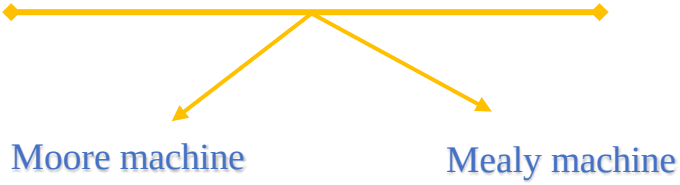
Z: is finite nonempty set of output symbols

δ: transition function (maps Q x$\sum$ into Q)

λ: output function (maps Q x$\sum$ into Z)

Note: Assume it has 2 tapes; input tape and output tape. It writes an output string of length n (output tape) when reading an input string of length n (input tape).

Finite Automata with output



The Moore machine

Definition

A **Moore machine** is a collection of five things:

1. A finite set of states $q_0, q_1, q_2, \ldots$ where q_0 is the start state
2. An alphabet of letters for forming the input string

$$\Sigma = \{a \ b \ c \ \ldots\}$$

3. An alphabet of possible output characters

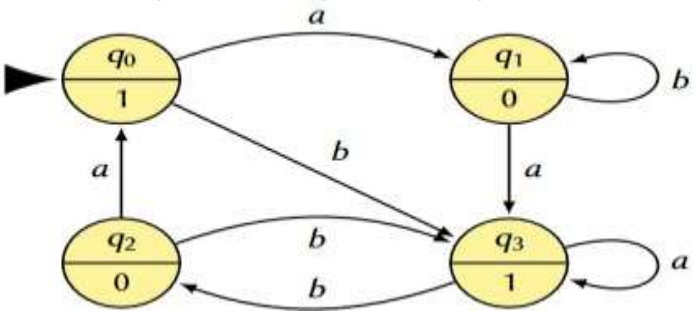
$$\Gamma = \{x \ y \ z \ \ldots\}$$

4. A transition table that shows for *each* state and *each* input letter which state is reached
5. An output table that shows which character from Γ is printed by each state as it is *entered* (this means when we "start" execution we print the character from the start state)

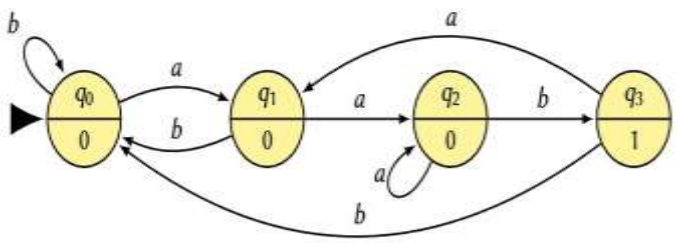
Example of a Moore machine

Input: $\Sigma = \{a \ b\}$ Output: $\Gamma = \{0 \ 1\}$ States: q_0, q_1, q_2, q_3

Old	Output	After a	After b
q_0	1	q_1	q_3
q_1	0	q_3	q_1
q_2	0	q_0	q_3
q_3	1	q_3	q_2



Example: count how many times *aab* occurs



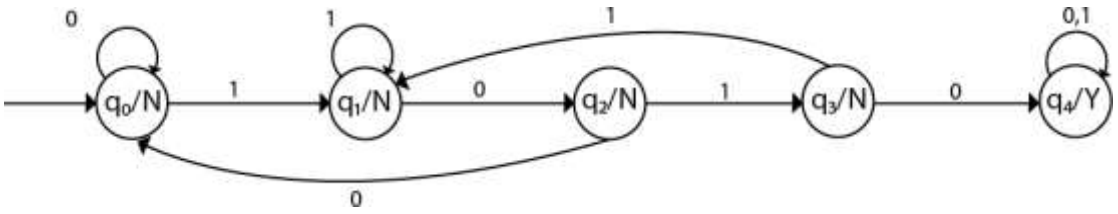
Tracing *aababbaabb*

Input		*a*	*a*	*a*	*b*	*a*	*b*	*b*	*a*	*a*	*b*	*b*
State	q_0	q_1	q_2	q_2	q_3	q_1	q_0	q_0	q_1	q_2	q_3	q_0
Output	0	0	0	0	1	0	0	0	0	0	1	0

- The number of substrings *aab* in the input string will be exactly the number of 1's in the output string.
- Given a language *L* and an FA that accepts it, we can "print" 0 in any non-final state and 1 in any final state.
- The 1's in any output sequence mark the end positions of all substrings *aab*
- If we remove the output ability and mark output states with 1 as final and output states with 0 as non-final, we convert the Moore machine to a standard FA

Q// Design a Moore machine with the input alphabet {0, 1} and output alphabet {Y, N} which produces Y as output if input sequence contains 1010 as a substring otherwise, it produces N as output.

The Moore machine will be:



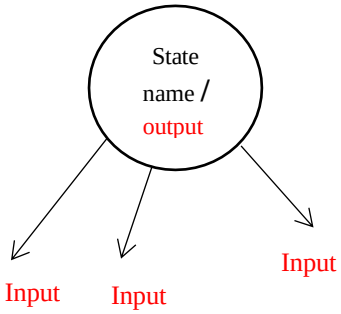
Some notes of FA with output

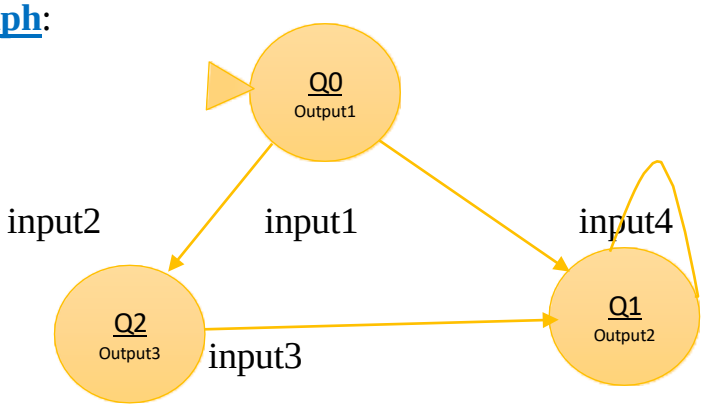
Note1: Each Machine has one or more start state but hasn't a final state. Therefore, instead of describing a formal language, it accepts a specific string as an input to give the appropriate string as output.

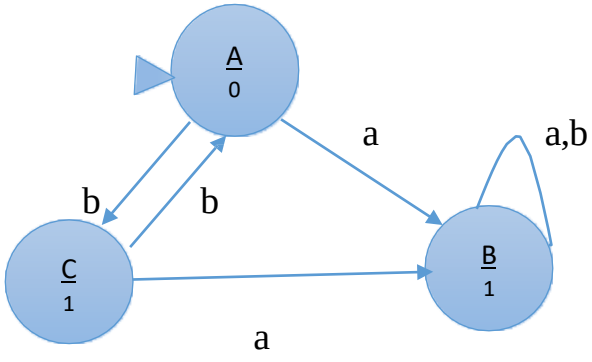
Note2: In case the machine has more than one start state, we should evaluate the behavior of the machine with each start state, if it's not specified.

Note3: The essential difference that appears in graphs of two type machines Moore and Mealy, is the position of the output symbol, which depend on it when the machine produces the output symbol to the buffer.

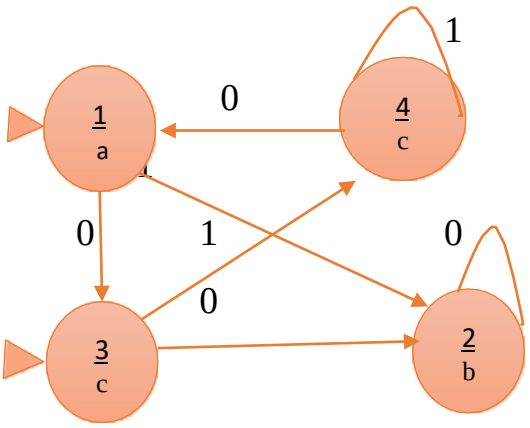
Moore machine overall graph:





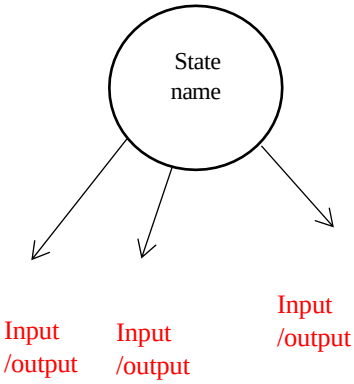


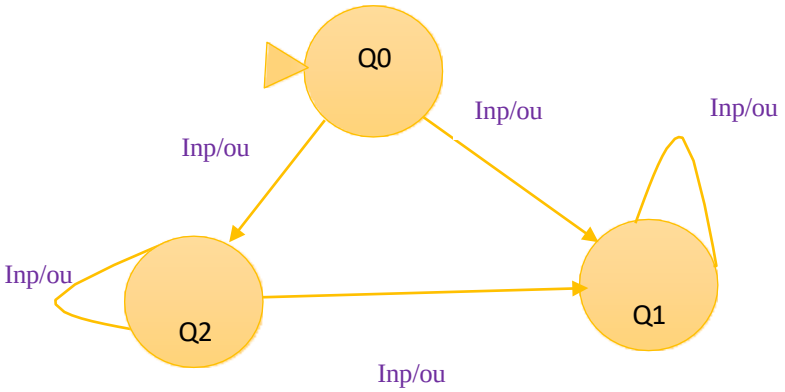
Moore machine

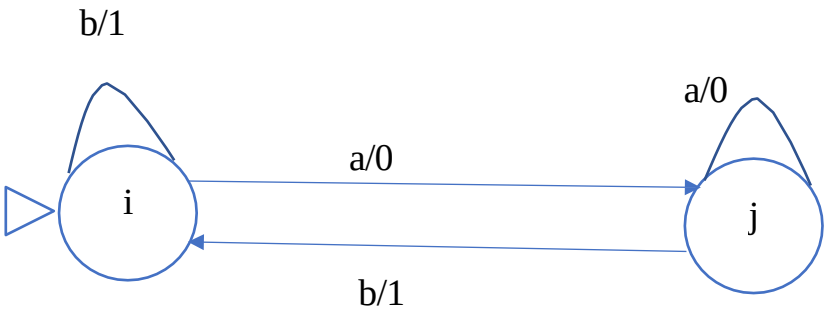


Moore machine

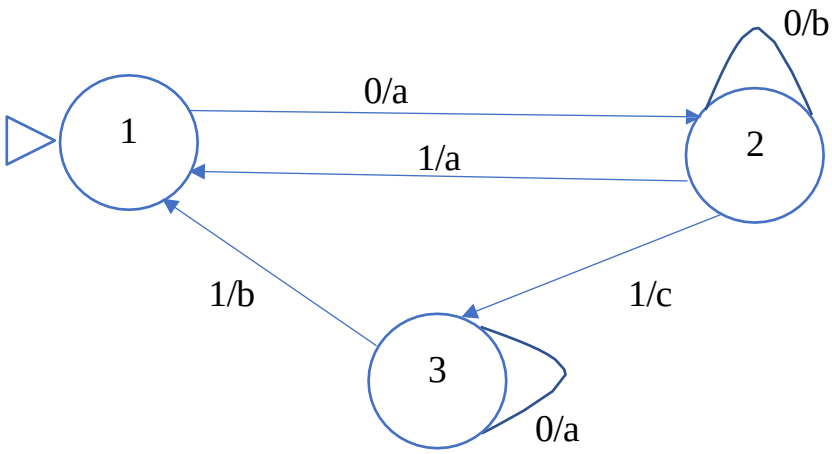
nachine overall graph:







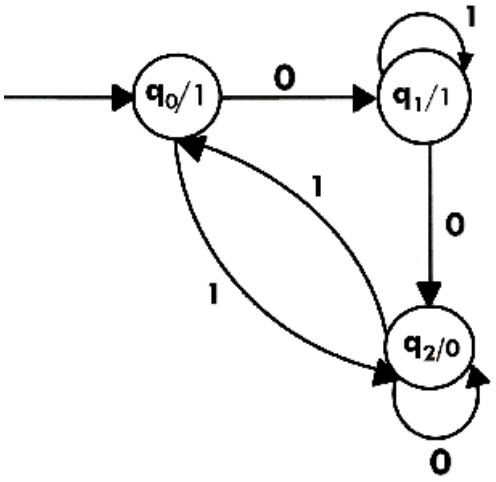
Mealy machine



Mealy machine

Q1: Evaluate the output sequence for the input sequence "1001011100" by using the following Moore machine:

The state diagram for Moore Machine is

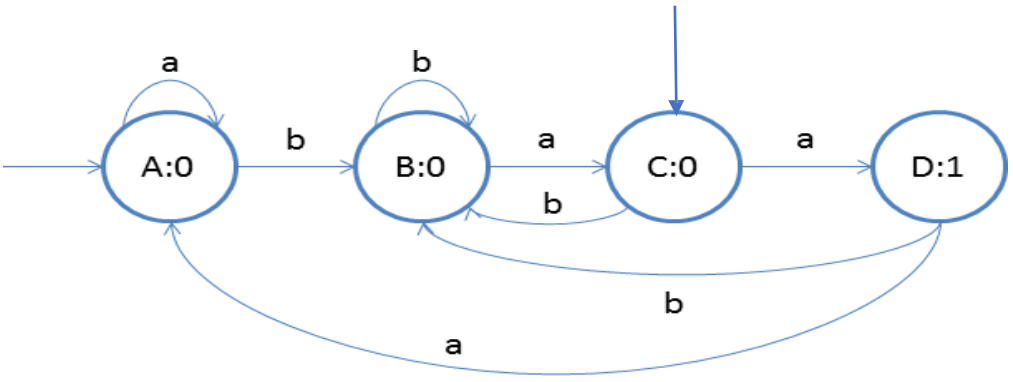


***When start state q0*:**

input		1	0	0	1	0	1	1	1	0	0
start	q0	q2	q2	q2	q0	q1	q1	q1	q1	q2	q2
output		1	0	0	0	1	1	1	1	1	0

The Output sequence: 1000111110

Q2: Evaluate the output sequence for the input sequence "aabbabbbaaab" by using the following Moore machine:



When A is the start state:

Input		a	a	b	b	a	b	b	b	a	a	a	b
State	A	A	A	B	B	C	B	B	B	C	D	A	B
output		0	0	0	0	0	0	0	0	0	0	1	0

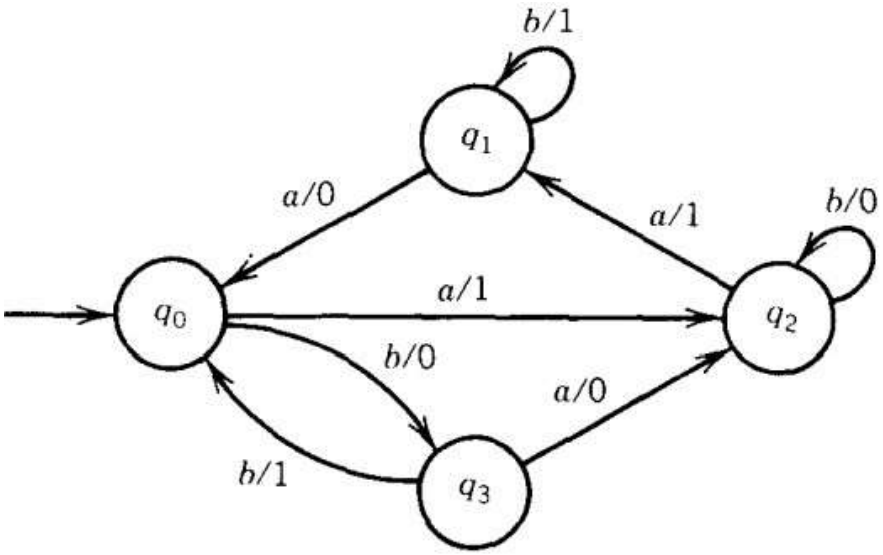
The output sequence: 000000000010

When C is the start state:

Input		a	a	b	b	a	b	b	b	a	a	a	b
State	C	D	A	B	B	C	B	B	B	C	D	A	B
output		0	1	0	0	0	0	0	0	0	0	1	0

The output sequence: 010000000010

Q3: Evaluate the output sequence for the input sequence "aababbabb" by using the following Mealy machine:

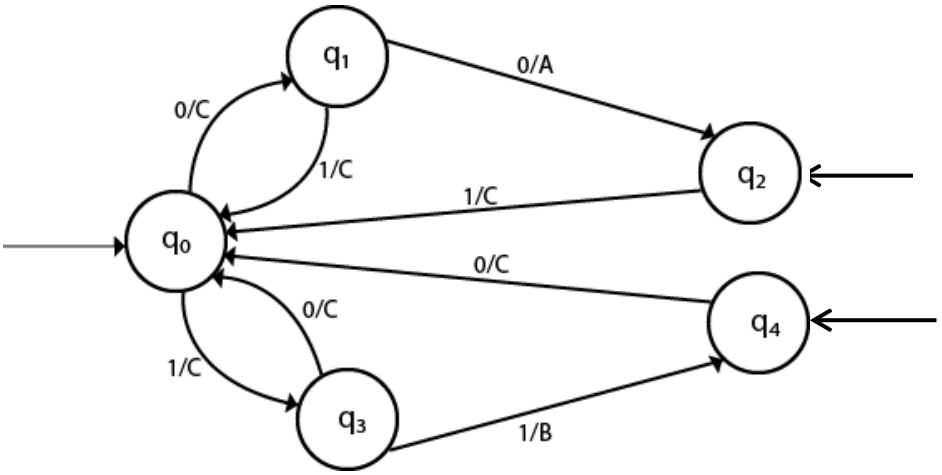


When q0 is the start state:

Input		a	a	b	a	b	a	b	b
State	q0	q2	q1	q1	q0	q3	q2	q2	q2
output		1	1	1	0	0	0	0	0

The output sequence: 11100000

Q4: Evaluate the output sequence for the input sequence "1001001110" by using the following Mealy machine:



When q0 is the start state:

Input		1	0	0	1	0	0	1	1	1	0
State	q0	q3	q0	q1	q0	q1	q2	q0	q3	q4	q0
output		C	C	C	C	C	A	C	C	B	C

The output sequence: CCCCCACCBC

H.W.// Evaluate the same input sequence with start states q2 and q4?

Note: A machine graph can contains the two type of nodes.

Conversion from Mealy machine to Moore Machine

In Moore machine, the output is associated with every state, and in Mealy machine, the output is given along the edge with input symbol. To convert Moore machine to Mealy machine, state output symbols are distributed to input symbol paths. But while converting the Mealy machine to Moore machine, we will create a separate state for every new output symbol and according to incoming and outgoing edges are distributed.

The following steps are used for converting Mealy machine to the Moore machine:

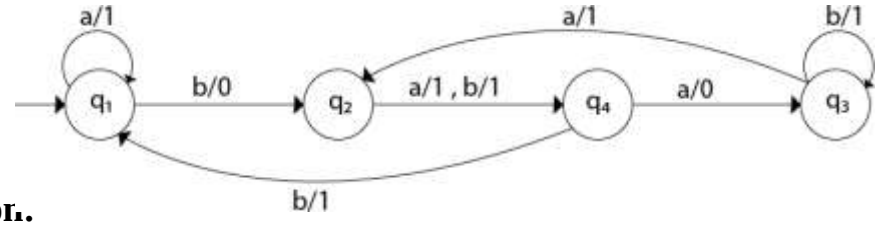
Step 1: For each state(Qi), calculate the number of different outputs that are available in the transition table of the Mealy machine.

Step 2: Copy state Qi, if all the outputs of Qi are the same. Break qi into n states as Qin, if it has n distinct outputs where n = 0, 1, 2....

Step 3: If the output of initial state is 0, insert a new initial state at the starting which gives 1 output.

Example 1:

Convert the following Mealy machine into equivalent Moore machine.



Solution..

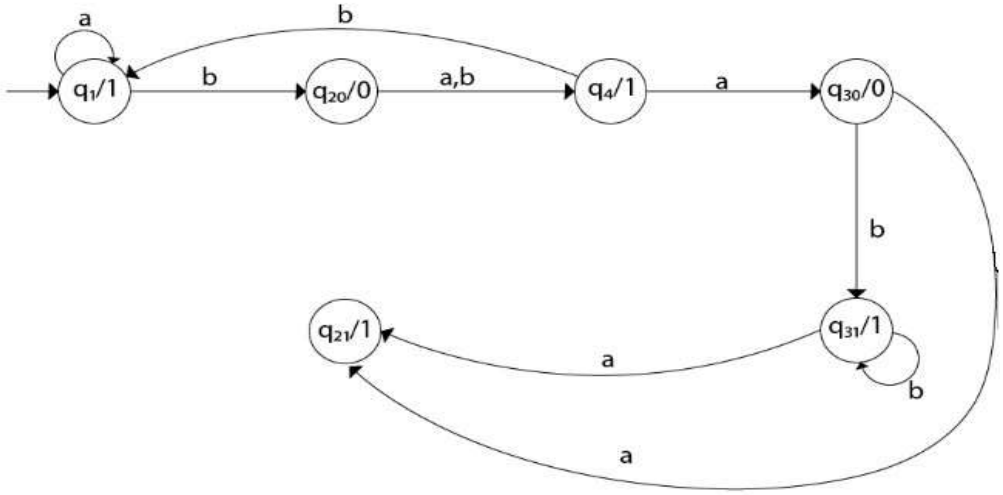
Transition table for above Mealy machine is as follows:

Present State	Next State			
	a		b	
	State	O/P	State	O/P
q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

- For state q1, there is only one incident edge with output 0. So, we don't need to split this state in Moore machine.
- For state q2, there is 2 incident edge with output 0 and 1. So, we will split this state into two states q20(state with output 0) and q21(with output 1).
- For state q3, there is 2 incident edge with output 0 and 1. So, we will split this state into two states q30(state with output 0) and q31(state with output 1).
- For state q4, there is only one incident edge with output 0. So, we don't need to split this state in Moore machine.

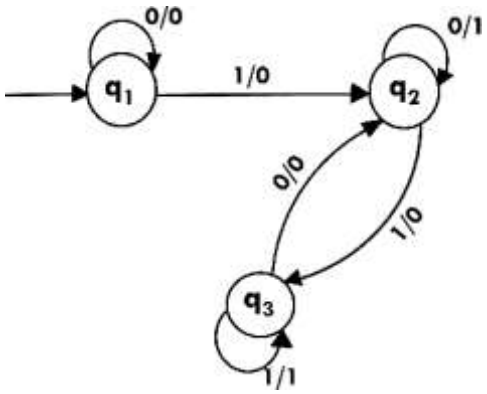
Transition table for Moore machine will be:

Present State	Next State		Output
	a=0	a=1	
q_1	q_1	q_2	1
q_{20}	q_4	q_4	0
q_{21}	ø	ø	1
q_{30}	q_{21}	q_{31}	0
q_{31}	q_{21}	q_{31}	1
q_4	q_3	q_4	1



Example 2:

Convert the following Mealy machine into equivalent Moore machine.



Solution:

Transition table for above Mealy machine is as follows:

Present State	Next State 0		Next State 1	
	State	o/P	State	o/P
q_1	q_1	0	q_2	0
q_2	q_2	1	q_3	0
q_3	q_2	0	q_3	1

and 1. So we will create two states for these states. For q2, two states will be q20(with output 0) and q21(with output 1). Similarly, for q3 two states will be q30(with output 0) and q31(with output 1).

Transition table for Moore machine will be:

Present State	Next State 0	Next State 1	O/P
q_1	q_1	q_{20}	0
q_{20}	q_{21}	q_{30}	0
q_{21}	q_{21}	q_{30}	1
q_{30}	q_{20}	q_{31}	0
q_{31}	q_{20}	q_{31}	1

Transition diagram for Moore machine will be: