## Finite Automata

A finite automaton is a mathematical (model) abstract machine that has a set of “states” and its “control” moves from state to state in response to external “inputs”. The control may be either “deterministic” meaning that the automation can’t be in more than one state at any one time, or “non deterministic”, meaning that it may be in several states at once. This distinguishes the class of automata as DFA or NFA.

‐ The DFA, i.e. Deterministic Finite Automata can’t be in more than one state at any time.

‐ The NFA, i.e. Non-Deterministic Finite Automata can be in more than one state at a time.

The finite state machines are used in applications in computer science and data networking. For example, finite-state machines are basis for programs for spell checking, indexing, grammar checking, searching large bodies of text, recognizing speech, transforming text using markup languages such as XML & HTML, and network protocols that specify how computers communicate.

Push

off

on

Push

Fig: - Finite automaton modeling an on/off switch

## Deterministic Finite Automata (DFA)

A deterministic finite automaton is defined by a quintuple (5-tuple) as (Q, ∑, δ, q0, F).

Where,

Q = Finite set of states, ∑ = Finite set input symbols, δ = A transition function that maps Q × ∑  Q q0 = A start state; q0 ∈ Q F = Set of final states; F ⊆ Q.

E. g: -

a

F

S

### General Notations of DFA

There are two preferred notations for describing this class of automata;

‐ Transition Table

‐ Transition Diagram

#### 1) Transition Table: -

Transition table is a conventional, tubular representation of the transition function δ that takes the arguments from Q × ∑ & returns a value which is one of the states of the automation. The row of the table corresponds to the states while column corresponds to the input symbol. The starting state in the table is represented by  followed by the state i.e. q, for q being start state, whereas final state as \*q, for q being final state. The entry for a row corresponding to state q and the column corresponding to input *a,* is the state δ (q, a). For example:

1. Consider a DFA;

Q = {q0, q1, q2, q3} ∑ = {0, 1}

q0 = q0 F = {q0} δ = Q × ∑  Q

Then the transition table for above DFA is as follows:

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| \*  q0 | q2 | q1 |
| q1 | q3 | q0 |
| q2 | q0 | q3 |
| q3 | q1 | q2 |

This DFA accepts strings having both an even number of 0’s & even number of 1’s.

1. DFA accepting all strings over {0, 1} having substring 01.

Let, Q = {q0, q1, q2}, ∑ = {0, 1}, q0 = q0, F = { q1}

|  |  |  |
| --- | --- | --- |
| δ: | 0 | 1 |
|  q0 | q2 | q0 |
| \*q1 | q1 | q1 |
| q2 | q2 | q1 |

#### 2) Transition Diagram: -

A transition diagram of a DFA is a graphical representation where; (or is a graph)

‐ For each state Q, there is a node represented by circle,

‐ For each state q in Q and each input a in ∑, if δ (q, a) = p then there is an arc from node q to p labeled a in the transition diagram. If more than one input symbol cause the transition from state q to p then arc from q to p is labeled by a list of those symbols.

‐ The start state is labeled by an arrow written with “start” on the node.

‐ The final or accepting state is marked by double circle.

‐ For the example I considered previously, the corresponding transition diagram is:

0

q

0

q

2

q

1

q

3

0

0

0

1

1

1

1

‐ Similarly for example II, the transition diagram is:

1 0

Start

0

1

q

1

q

0

q

2

0, 1

**How DFA process strings?**

The first thing we need to understand about a DFA is how DFA decides whether or not to “accept” a sequence of input symbols. The “language” of the DFA is the set of all symbols that the DFA accepts. Suppose a1, a2, …… an is a sequence of input symbols. We start out with the DFA in its start state, q0. We consult the transition function δ also for this purpose. Say δ (q0, a1) = q1 to find the state that the DFA enters after processing the first input symbol a1. We then process the next input symbol a2, by evaluating δ (q1, a2); suppose this state be q2. We continue in this manner, finding states q3, q4, …, qn such that δ (qi-1, ai) = qi for each i. if qn is a member of F, then input a1, a2, --- an is accepted & if not then it is rejected.

#### Extended Transition Function of DFA: -

The extended transition function of DFA, denoted by is a transition function that takes two arguments as input, one is the state q of Q and another is a string w∈ ∑\*, and generates a state p ∈ Q. This state p is that the automaton reaches when starting in state q & processing the sequence of inputs w.



i.e. (q, w) = p



Let us define by induction on length of input string as follows:



*Basis step:*   i.e. from state q, reading no input symbol stays at the same (q, ε) = q state.



*Induction:* Let w be a string from ∑\* such that *w* = *xa*, where *x* is substring of *w* without last symbol and *a* is the last symbol of *w*, then

(q, w) = δ ( (q, x), a)



Thus, to compute (q, w), we first compute (q, x), the state the automaton is in after processing all but last symbol of w. let this state is p, i.e. (q, x) = p.

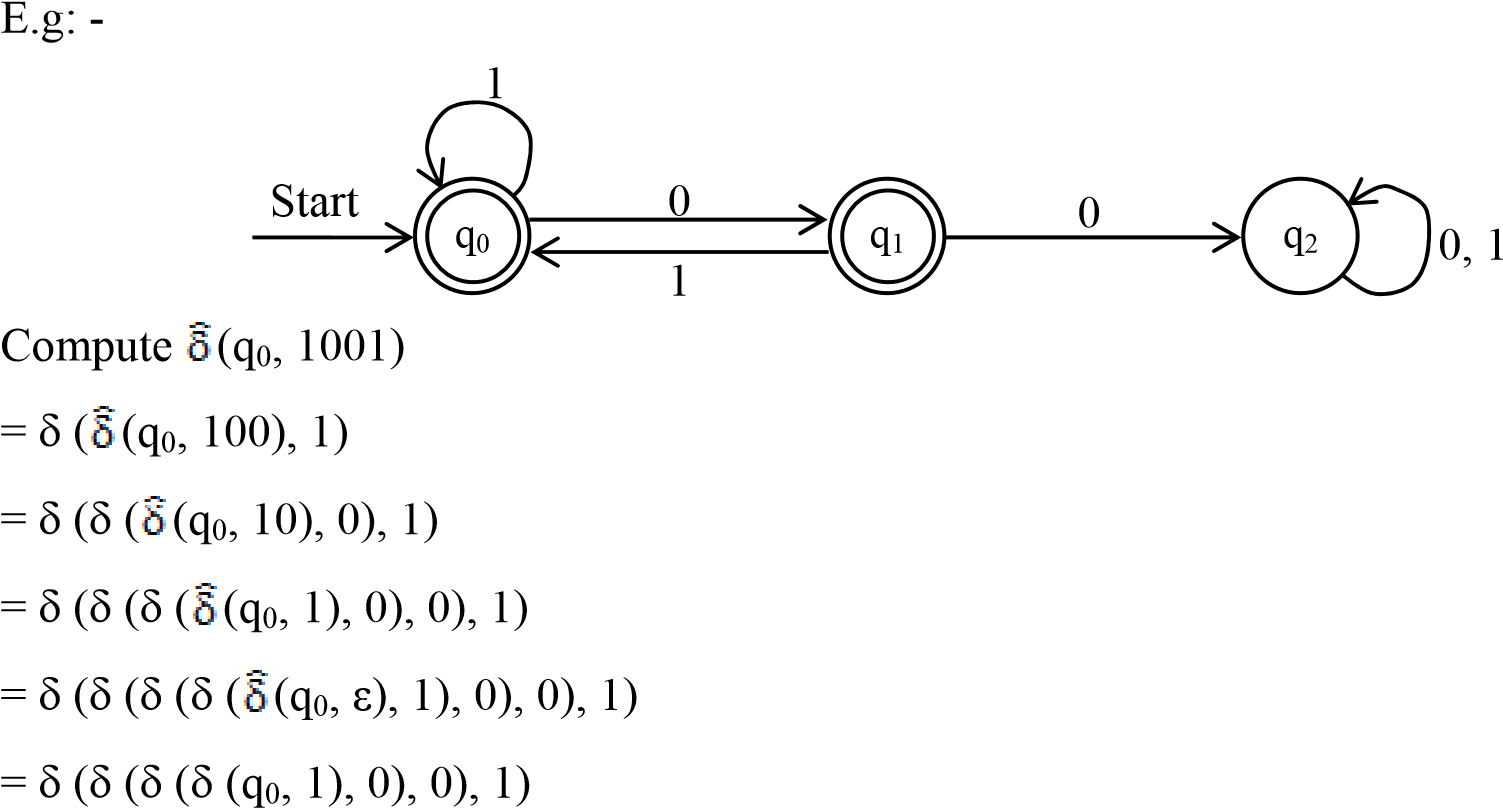


Then, (q, w) is what we get by making a transition from state p on input a, the last symbol of w.



i.e. (q, w) = δ (p, a)





= δ (δ (δ (q0, 0), 0), 1)

= δ (δ (q1, 0), 1)

= δ (q2, 1)

= q2, *so accepted*.

Compute (q0, 101)



= δ ((q0, 10), 1)



= δ (δ ((q0, 1), 0), 1)



= δ (δ (δ ((q0, ε), 1), 0), 1)



= δ (δ (δ (q0, 1), 0), 1)

= δ (δ (q0, 0), 1)

= δ (q1, 1)

= q0, *so not accepted*.

#### String accepted by a DFA

A string x is accepted by a DFA (Q, ∑, δ, q0, F) if; (q, x) = p ∈ F.



#### Language of DFA

The language of DFA M = (Q, ∑, δ, q0, F) denoted by L (M) is a set of strings over ∑\*

that are accepted by M.

i.e; L (M) = {w/ (q0, w) = p ∈ F}



That is; the language of a DFA is the set of all strings w that take DFA starting from start state to one of the accepting states. The language of DFA is called regular language.

**Examples:**

* Construct a DFA, that accepts all the strings over ∑ = {a, b} that do not end with

ba.

q

2

q

0

q

1

Start

b

a

a

b

b

a

* DFA accepting all string over ∑ = {0, 1} ending with 3 consecutive 0’s.

1  DFA over {a, b} accepting {baa, ab, abb}

q

2

q

3

q

1

q

0

Start

0

0

1

0

1

1

0

q

6

q

3

q

5

q

0

q

1

q

2

q

4

Start

b

b

a

b

a, b

b

a

a

a

b

a, b

a

* DFA accepting zero or more consecutive 1’s.

i.e. L (M) = {1n / n = 0, 1, 2, ……}

1

Start

0

q

1

q

0

0, 1

* DFA over {0, 1} accepting {1, 01}

0

Start

0

1

0

, 1

q

3

q

2

q

1

q

0

0,1

1

* DFA over {a, b} that accepts the strings ending with abb.

q

2

q

3

q

1

q

0

Start

a

b

a

a

b

b

b

a

## Non-Deterministic Finite Automata (NFA)

A non-deterministic finite automaton is a mathematical model that consists of:

‐ A set of states Q, (finite)

‐ A finite set of input symbols ∑, (alphabets)

‐ A transition function that maps state symbol pair to sets of states.

‐ A state q0 ∈ Q, that is distinguished as a start (initial) state.

‐ A set of final states F distinguished as accepting (final) state. F ⊆ Q.

Thus, NFA can also be interpreted by a quintuple; (Q, ∑, δ, q0, F) where δ is Q × ∑ = 2Q.

Unlike DFA, a transition function in NFA takes the NFA from one state to several states just with a single input. For example;

0, 1

Start

0

1

q

1

q

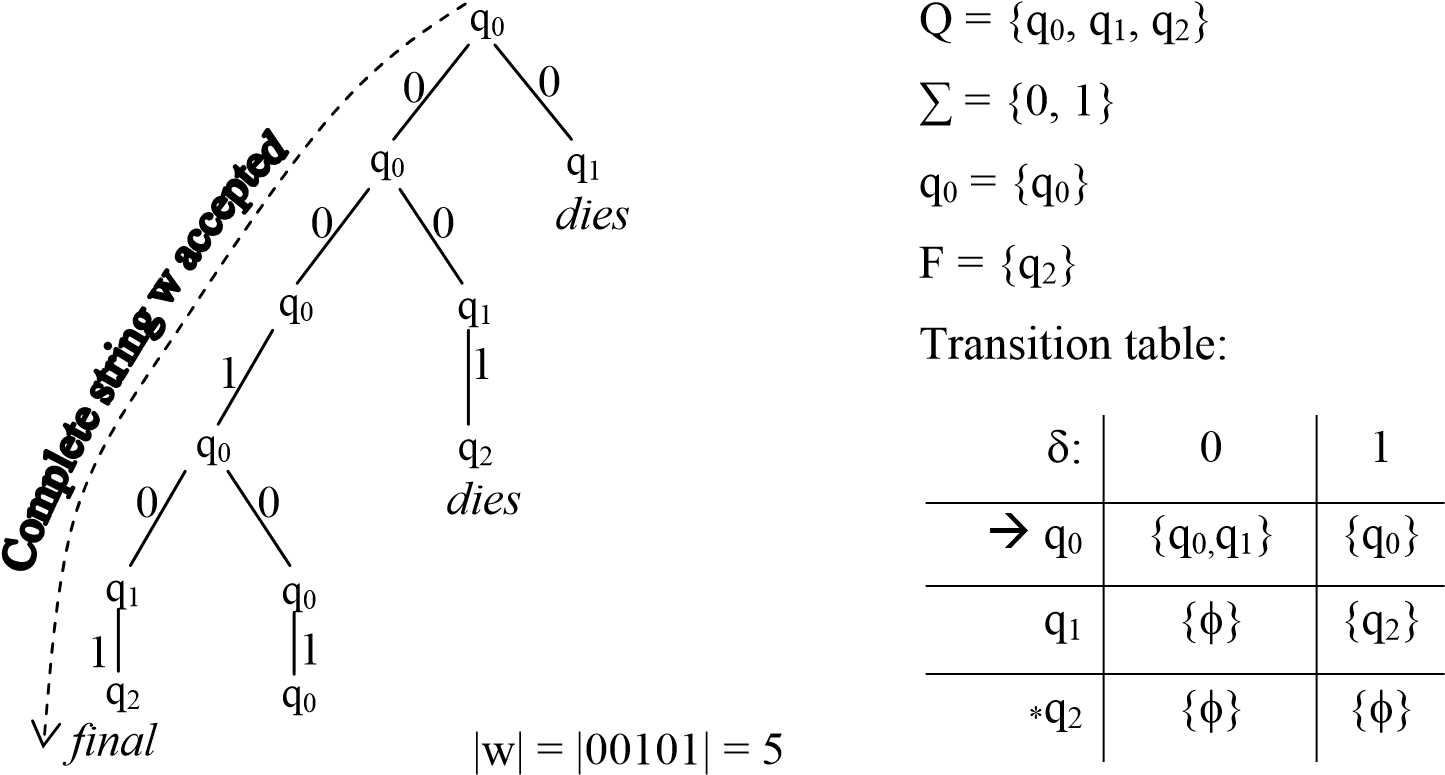
2

q

0

Fig: - NFA accepting all strings that end in 01.

Here, from state q1, there is no any arc for input symbol 0 & no nay arc out of q2 for 0 & 1. So, we can conclude in a NFA, there may be zero no. of arcs out of each state for each input symbol. While in DFA, it has exactly one arc out of each state for each input symbol. For input sequence w = 00101, the states the NFA can be in during the processing of the input are as:



# NFA over {0, 1} accepting strings {0, 01, 11}.

q

1

q

2

q

0

Start

0

,

1

1

Transition table: 0

|  |  |  |
| --- | --- | --- |
| δ: | 0 | 1 |
|  q0 | {q0,q2} | {q1} |
| q1 | {φ} | {q2} |
| \*q2 | {φ} | {φ} |

Computation tree for 01;

q0

q2

*dies*

q

1

0

0

1

q2

*Final, so 01 is accepted*

Computation tree for 0110

q0

q2

*dies*

q

1

0

0

1

q2

*dies, so 0110 is not accepted*

### Extended Transition Function of NFA: -

The extended transition function of NFA, denoted by is a transition function that takes two arguments as input; a state q ∈ Q & a string w and returns a set of states that the NFA is in, if it starts in q & processes the string w.



*Definition by Induction Hypothesis:*

*Basis Step:* (q, ε) = {q} i.e. reading no input symbol remains into the same state.

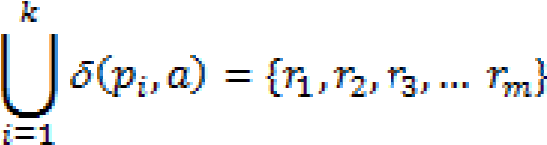


*Induction:* Let w be a string from ∑\* such that w = xa, where x is a substring of w without last symbol a.

Also let,

(q, x) = {p1, p2, p3, …pk}



&

Then, (q, w) = {r1, r2, r3, …rm}



Thus, to compute (q, w) we first compute (q, x) & then following any transition from each of these states with input a.



Consider,

0, 1

Start

0

1

q

1

q

2

q

0

Now, computing 01101; *Solution:*

(q0, 01101)



(q0, ε) = {q0}



(q0, 0) = {q0, q1}



(q0, 01) = δ (q0, 1) ∪ δ (q1, 1) = {q0} ∪ {q2} = {q0, q2}



(q0, 011) = δ (q0, 1) ∪ δ (q2, 1) = {q0} ∪ {φ} = {q0}



(q0, 0110) = δ (q0, 0) = {q0} ∪ {q1} = {q0, q1}



(q0, 01101) = δ (q0, 1) ∪ δ (q1, 1) = {q0} ∪ {q2} = {q0, q2}



So, accepted

**Examples:**

* Construct a NFA over {a, b} that accepts strings having *aa* is substring.

a, b

q

1

q

2

q

0

Start

a

a

a, b

* NFA for strings over {0, 1} that contain 0110 or 1001.

q

7

q

0

q

1

q

2

q

4

q

5

q

3

q

6

Start

0

, 1

0

, 1

0

0

0

0

1

1

1

1

* NFA over {a, b} that have a as one of the last 3 characters.

a, b

Start

a

b

q

3

q

0

q

1

q

2

b

* NFA over {a, b} that accepts strings stating with *a* and ending with *b*.

a, b

q

1

q

2

q

0

Start

a

b

### Language of NFA: -

The language of NFA, M = (Q, ∑, δ, q0, F), denoted by L (M) is;

L (M) = {w/ (q, w) ∩ F ≠ φ}



i.e. L (M) is a set of strings w in ∑\* such that (q, w) contains at least one accepting



state.

Example:

# Design a NFA for the language over {0, 1} that have at least two consecutive 0’s or

’s.

1

⇒

q

1

q

2

q

0

q

3

q

4

Start

0

0

1

0

0

, 1

1

0

, 1

0

, 1

1

Now, compute 10110; *Solution:*

(q0, 10110)



(q0, ε) = {q0}



(q0, 1) = {q1, q3}



(q0, 10) = δ (q1, 0) ∪ δ (q3, 0) = {q2} ∪ {q1} = {q1, q2}



(q0, 101) = δ (q1, 1) ∪ δ (q2, 1) = {q3} ∪ {q2} = {q2, q3}



(q0, 1011) = δ (q2, 1) ∪ δ (q3, 1) = {q2} ∪ {q4} = {q2, q4}



(q0, 10110) = δ (q2, 0) ∪ δ (q4, 0) = {q2} ∪ {q4} = {q2, q4}



So, accepted

### Equivalence of NFA & DFA

‐ We now show that DFAs & NFAs accept exactly the same set of languages. That is; non-determinism does not make a finite automaton more powerful.

‐ To show that NFAs and DFAs accept the same class of language, we show;

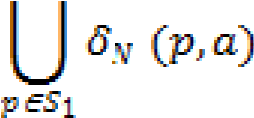
 Any language accepted by a NFA can also be accepted by some DFA. For this we describe an algorithm that takes any NFA and converts it into a DFA that accepts the same language. The algorithm is called “subset construction algorithm”.

The key idea behind the algorithm is that; the equivalent DFA simulates the NFA by keeping track of the possible states it could be in. Each state of DFA corresponds to a subset of the set of states of the NFA, hence the name of the algorithm. If NFA has n-states, the DFA can have 2n states (at most), although it usually has many less.

#### Steps in Subset construction: -

To convert a NFA, N = (QN, ∑, δN, q0, FN) into an equivalent DFA D = (QD, ∑, δD, q0, FD), we have following steps.

* The start state of D is the set of start states of N i.e. if q0 is start state of N then D has start state as {q0}.
* QD is set of subsets of QN i.e. QD = 2QN. So, QD is power set of QN. So if QN has n states then QD will have 2n states. However, all of these states may not be accessible from start state of QD so they can be eliminated. So QD will have less than 2n states.
* FD is set of subsets S of QN such that S ∩ FN ≠ φ i.e. FD is all sets of N’s states that include at least one final state of N.

For each set S1 ⊆ QN & each input *a* ∈ ∑, δD (S1, a) =

i.e. for any state {q0, q1, q2, … qk} of the DFA & any input a, the next state of the DFA is the set of all states of the NFA that can result as next states if the NFA is in any of the state’s q0, q1, q2, … qk when it reads a. *For example:*

0, 1

Start

0

1

q

1

q

2

q

0

|  |  |  |
| --- | --- | --- |
| δ: | 0 | 1 |
| A | A | A |
| B | E | B |
| C | A | D |
| \*D | A | A |
| E | E | F |
| \*F | E | B |
| \*G | A | D |
| \*H | E | F |

|  |  |  |  |
| --- | --- | --- | --- |
|  | δ: | 0 | 1 |
| A | φ | φ | φ |
| B |  {q0} | {q0,q1} | {q0} |
| C | {q1} | φ | {q2} |
| D | \*{q2} | φ | φ |
| E | {q0, q1} | {q0, q1} | {q0, q2} |
| F | \*{q0, q2} | {q0, q1} | {q0} |
| G | \*{q1, q2} | φ | {q2} |
| H | \*{q0, q1, q2} | {q0, q1} | {q0, q2} |

i.e.

The equivalent DFA is:

E

F

B

Start

0

1

1

0

0

1

Other states are ignored as they are not reached starting from start state.

⇒ 1 0

Start

0

1

0

1

q

{

0

}

q

{

0

, q

1

}

{

q

0

, q

2

}

#### Questions: 1

Convert the NFA to DFA:

r

q

p

s

Start

1

, 1

0

0

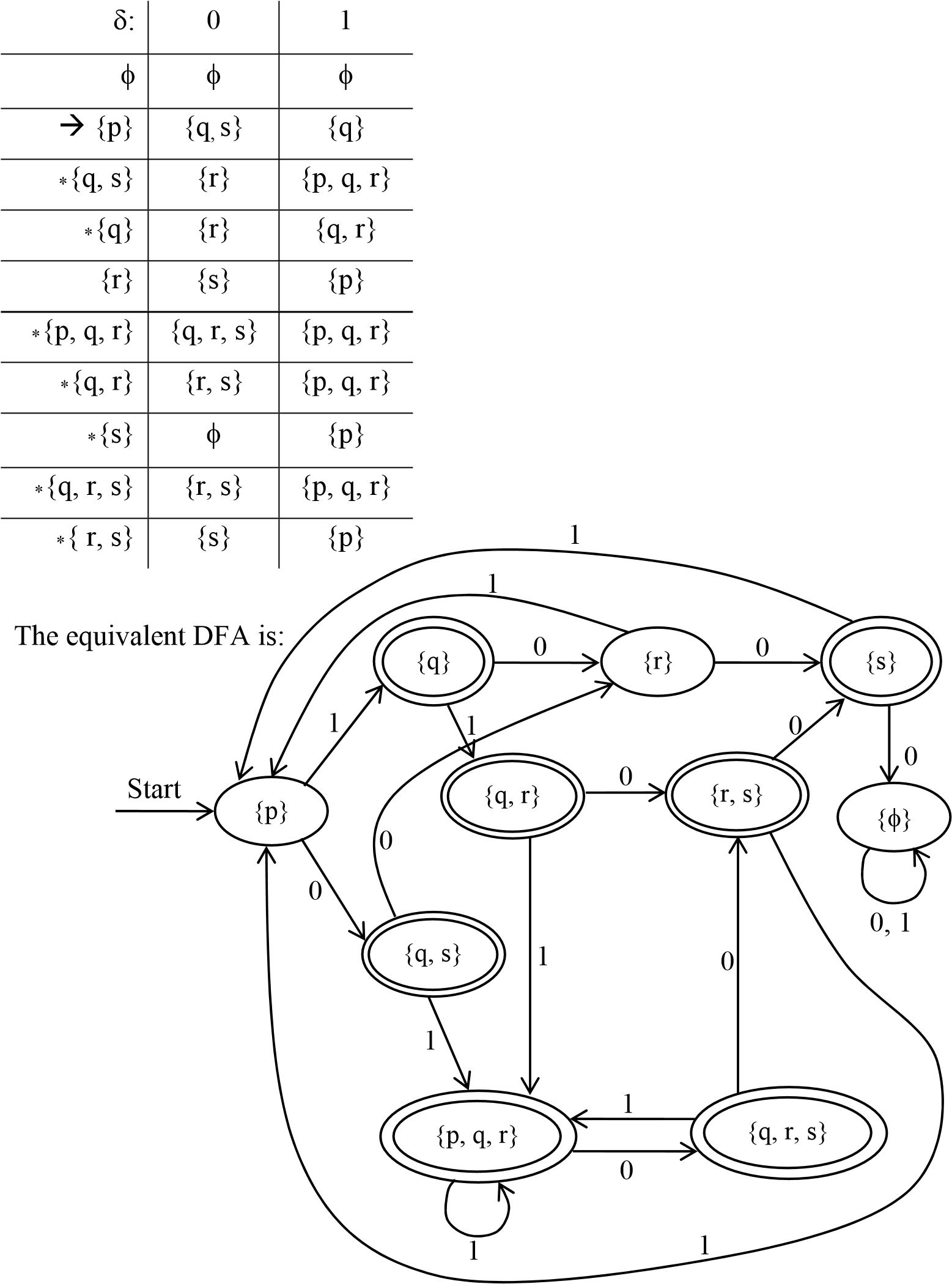
, 1

1

0

0

Now through subset construction, we have the DFA as:



# Covert following NFA to DFA:

r

s

p

q

Start

a

b

b

a

a

a

# Convert the following NFA to DFA:

0, 1

r

s

p

q

Start

0

0

, 1

0

0, 1

## NFA with ε-transition (ε-NFA)

This is another extension of finite automation. The new feature that it incorporates is, it allows a transition on ε, the empty string, so that a NFA could make a transition spontaneously without receiving an input symbol.

A NFA with ε-transition is defined by five tuples (Q, ∑, δ, q0, F), where;

Q = set of finite states ∑ = set of finite input symbols q0 = Initial state, q0 ∈ Q F = set of final states; F ⊆ Q

δ = a transition function that maps;

Q × ∑ ∪ {ε} 2Q

δ (q, ε) = {p1, p2, …… pk} (set of all states that can be reached from q with input ε). ***For example;***

a b

⇒

{a, b, ab, aab, abb, …}

⇒

abb, bab}

{

p

q

Start

ε

x

y

p

t

s

q

r

w

u

v

Start

ε

a

b

ε

ε

b

b

a

ε

b

**ε-closure of a state:**

ε-closure of a state ‘q’ can be obtained by following all transitions out of q that are labeled ε. After we get to another state by following ε, we follow the ε-transitions out of those states & so on, eventually finding every state that can be reached from q along any path whose arcs are all labeled ε.

Formally, we can define ε-closure of the state q as; *Basis:* state q is in ε-closure (q).

*Induction:* If state q is reached with ε-transition from state q, p is in ε-closure (q). and if there is an arc from p to r labeled ε, then r is in ε-closure (q) and so on.

### Extended Transition Function of ε-NFA: -

The extended transition function of ε-NFA denoted by is defined by;



i) BASIS STEP: - (q, ε) = ε-closure (q) ii) INDUCTION STEP: -

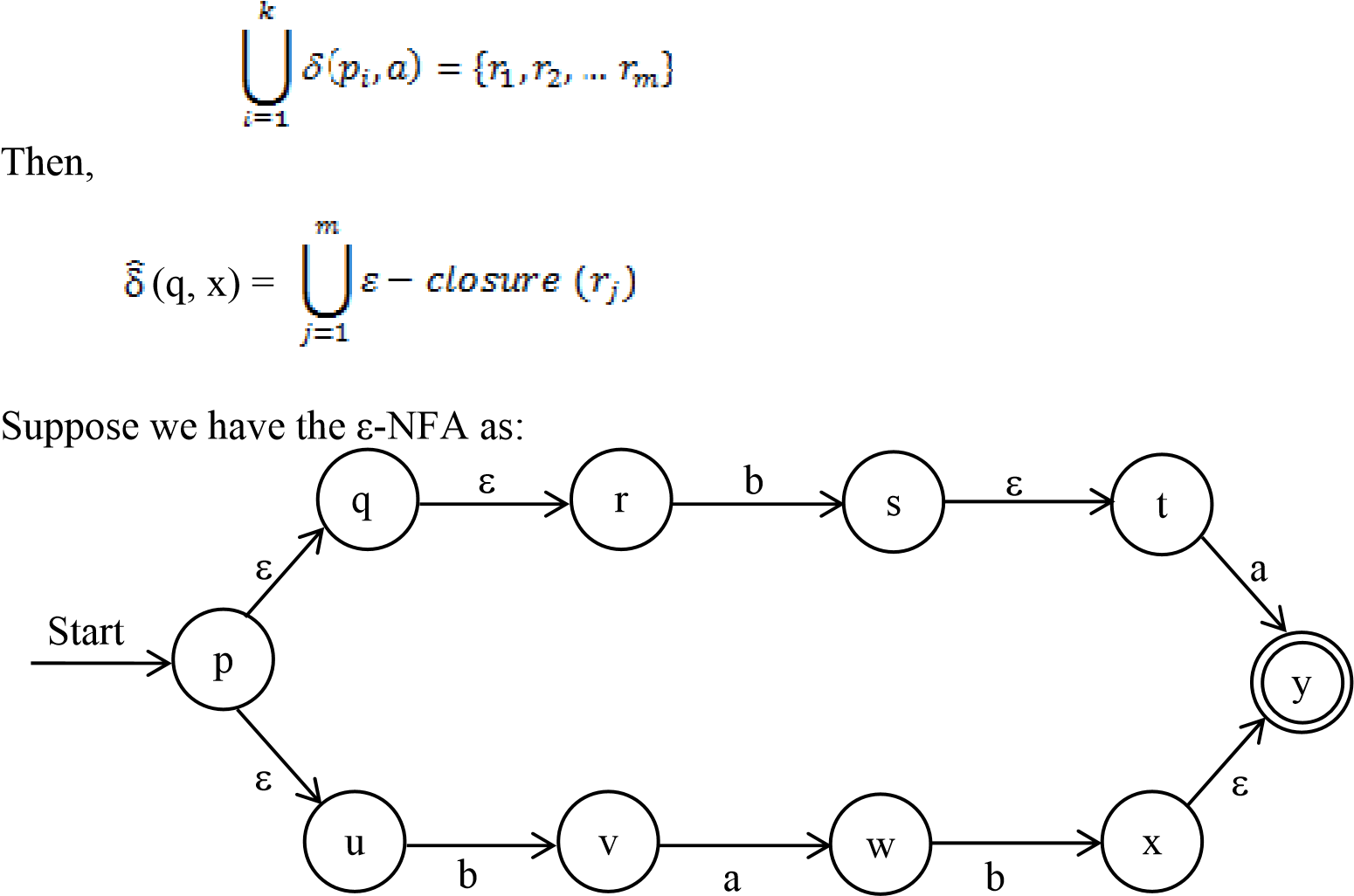


Let w = xa be a string, where x is substring of w without last symbol a and a ∈ ∑

but a ≠ ε.

Let (q, x) = {p1, p2, … pk} i.e. pi’s are the states that can be reached from q following path labeled x which can end with many ε & can have many ε. Also let,





*Now, compute for string ba.*

1. (p, ε) = ε-closure (p) = {p, q, r, s}



1. (p, b) = δ (p, b) ∪ δ (q, b) ∪ δ (r, b) ∪ δ (s, b)



= φ ∪ φ ∪ {u} ∪ {t} = {u, t}

(u, ε) ∪ (t, ε) = ε-closure (u) ∪ ε-closure (t)



∴(p, b) = {u} ∪ {t, x} = {u, t, x}



1. (p, ba) = δ (u, a) ∪ δ (t, a) ∪ δ(x, a)



= {v}∪ φ ∪ {y}

(v, ε) ∪ (y, ε) = ε-closure (v) ∪ ε-closure (y)



= {v} ∪ {y}

∴ (p, ba) = {v, y} *Accepted*



*Compute for string bab*

1. (p, ε) = ε-closure (p) = {p, q, r, s}



1. (p, b) = δ (p, b) ∪ δ (q, b) ∪ δ (r, b) ∪ δ (s, b)



= φ ∪ φ ∪ {u} ∪ {t} = {u, t}

(u, ε) ∪ (t, ε) = ε-closure (u) ∪ ε-closure (t)



∴(p, b) = {u} ∪ {t, x} = {u, t, x}



1. (p, ba) = δ (u, a) ∪ δ (t, a) ∪ δ(x, a)



= {v}∪ φ ∪ {y} = {v, y}

(v, ε) ∪ (y, ε) = ε-closure (v) ∪ ε-closure (y)



= {v} ∪ {y}= {v, y}

∴(p, ba) = {v, y}



1. (p, bab) = δ (v, b) ∪ δ (y, b)



= {w} ∪ φ

= {w}

(w, ε) = ε-closure (w)



= {y} *Accepted*.

### Conversion of ε-NFA into NFA & DFA: -

#### 1) ε-NFA to NFA: -

To construct NFA from a given ε-NFA;

* Here, we have to do is to eliminate the ε- transitions somehow, so that the resulting NFA will have no more ε-transitions. For this we do as below:
* Take start state of ε-NFA as start state of NFA. If ε-NFA accepts ε, then mark

start state as a final state.

* Take final state of ε-NFA as final state of NFA.
* Perform δN (q, a) = ε-closure (δ (ε-closure (q), a))

Where, δN = transition function of resulting NFA.

For example:

D

E

A

B

C

Start

ε

0

1

1

0

ε

0

|  |  |  |  |
| --- | --- | --- | --- |
|  | ε | 0 | 1 |
|  A | {B, D} | {A} | φ |
| B | φ | {C} | {E} |
| C | φ | φ | {B} |
| D | φ | {E} | φ |
| \*E | φ | φ | φ |

Now, the NFA is:

Start state = A Now,

δN (A, 0) = ε-closure (δ (ε-closure (A), 0)) = ε-closure (δ ({A, B, D}, 0))

= ε-closure ({A, C, E}) = {A, B, C, D, E} δN (A, 1) = ε-closure (δ (ε-closure (A), 1))

= ε-closure (δ ({A, B, D}, 1))

= ε-closure ({E})

= {E}

δN (B, 0) = ε-closure (δ (ε-closure (B), 0))

= ε-closure (δ ({B}, 0))

= ε-closure ({C})

= {C}

δN (B, 1) = ε-closure (δ (ε-closure (B), 1))

= ε-closure (δ ({B}, 1))

= ε-closure ({E})

= {E}

δN (C, 0) = ε-closure (δ (ε-closure (C), 0))

= ε-closure (δ ({C}, 0))

= ε-closure ({φ})

= φ

δN (C, 1) = ε-closure (δ (ε-closure (C), 1))

= ε-closure (δ ({C}, 1))

= ε-closure ({B})

= {B}

δN (D, 0) = ε-closure (δ (ε-closure (D), 0))

= ε-closure (δ ({D}, 0))

= ε-closure ({E})

= {E}

δN (D, 1) = ε-closure (δ (ε-closure (D), 1))

= ε-closure (δ ({D}, 1))

= ε-closure ({φ})

= φ

δN (E, 0) = ε-closure (δ (ε-closure (E), 0))

= ε-closure (δ ({E}, 0))

= ε-closure ({φ})

= φ

δN (E, 1) = ε-closure (δ (ε-closure (E), 1))

= ε-closure (δ ({E}, 1))

= ε-closure ({φ})

= φ

Thus, the resulting NFA is:

D

E

A

B

C

0

0

Start

0

0

, 1

1

0

0

0

#### 1 2) ε-NFA to NFA: -

Given an ε-NFA E = (Q, ∑, δ, q0, FD), to construct a DFA equivalent to E, let D = (Q’,

∑, δ’, q0’, F’) is a DFA equivalent to E.

Here,

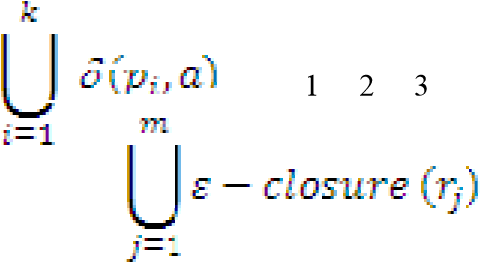
Q’ = is a set of subsets of Q.

q0’ = ε-closure (q0)

F’ = set of those states that contains at least one accepting state of E.

i.e. F’ = {S/S ⊆ Q’ & S ∩ FD ≠ φ} δ’ = is a transition function computed as; δ’ (s, a), where S is a member of Q’ & a ∈ ∑.

Let S = {p1, p2, … pk}

Compute  = {r , r , r , …, rm}

Then, δ’ (s, a) =

# For the following ε-NFA, configure equivalent DFA:

D

E

A

B

C

Start

ε

0

1

1

0

ε

0

1

The start state of given ε-NFA is A, So start state for DFA will be; q0’ = ε-closure (A)

= {A, B, D} Here, δ' ({A, B, D}, 0) = ε-closure (δ ({A, B, D}, 0))

= ε-closure ({A, C, E})

= {A, B, C, D, E}

δ' ({A, B, D}, 1) = ε-closure (δ ({A, B, D}, 1))

= ε-closure ({D, E})

= {D, E}

δ' ({A, B, C, D, E}, 0) = ε-closure (δ ({A, B, C, D, E}, 0))

= ε-closure ({A, C, E})

= {A, B, C, D, E}

δ' ({A, B, C, D, E}, 1) = ε-closure (δ ({A, B, C, D, E}, 1))

= ε-closure ({B, D, E})

= {B, D, E}

δ' ({D, E}, 0) = ε-closure (δ ({D, E}, 0))

= ε-closure ({E})

= {E}

δ' ({D, E}, 1) = ε-closure (δ ({D, E}, 1))

= ε-closure ({D}) = {D}

& we calculate remaining states in same way.

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
|  {A, B, D} | {A, B, C, D, E} | {E, D} |
| \*{A, B, C, D, E} | {A, B, C, D, E} | {B, D, E} |
| \*{D, E} | {E} | {D} |
| \*{B, D, E} | {C, E} | {E, D} |
| \*{E} | φ | φ |
| {D} | {E} | {D} |
| \*{C, E} | φ | {B} |
| {B} | {C} | {E} |
| {C} | φ | {B} |
| φ | φ | φ |

This transition table gives the final DFA of above ε-NFA. # Convert following ε-NFA to NFA & DFA

a b c

q

1

q

2

q

0

Start

ε

ε

*1) Constructing the equivalent NFA:*

Start state of NFA = Start state of ε-NFA

= q0

Final state of NFA = Final state of ε-NFA

= q2

For this, we compute as; δN (q0, a) = ε-closure (δ (ε-closure (q0), a))

= ε-closure (δ ({q0, q1, q2}, a))

= ε-closure ({q0}) = {q0, q1, q2}

δN (q0, b) = ε-closure (δ (ε-closure (q0), b))

= ε-closure (δ ({q0, q1, q2}, b))

= ε-closure ({q1}) = {q1, q2}

δN (q0, c) = ε-closure (δ (ε-closure (q0), c))

= ε-closure (δ ({q0, q1, q2}, c))

= ε-closure ({q2})

= {q2}

δN (q1, a) = ε-closure (δ (ε-closure (q1), a))

= ε-closure (δ ({q1, q2}, a))

= ε-closure ({φ})

= {φ} δN (q1, b) = ε-closure (δ (ε-closure (q1), b))

= ε-closure (δ ({q1, q2}, b))

= ε-closure ({q1})

= {q1, q2}

δN (q1, c) = ε-closure (δ (ε-closure (q1), c))

= ε-closure (δ ({q1, q2}, c))

= ε-closure ({q2})

= {q2}

δN (q2, a) = ε-closure (δ (ε-closure (q2), a))

= ε-closure (δ ({q2}, a))

= ε-closure ({φ})

= {φ} δN (q2, b) = ε-closure (δ (ε-closure (q2), b))

= ε-closure (δ ({q2}, b))

= ε-closure ({φ})

= {φ} δN (q2, c) = ε-closure (δ (ε-closure (q2), c))

= ε-closure (δ ({q2}, c))

= ε-closure ({q2})

= {q2}

The final DFA is as follows:

q

1

q

2

q

0

Start

a, b

a

b

c

b, c

a, b, c

*2) Constructing the equivalent DFA:*

Start state of DFA = ε-closure (start state of ε-NFA)

= ε-closure (q0) = {q0, q1, q2}

Now,

δ' ({q0, q1, q2}, a) = ε-closure (δ ({q0, q1, q2}, a))

= ε-closure ({q0}) = {q0, q1, q2}

δ' ({q0, q1, q2}, b) = ε-closure (δ ({q0, q1, q2}, b)) = ε-closure ({q1}) = {q1, q2} δ' ({q0, q1, q2}, c) = ε-closure (δ ({q0, q1, q2}, c))

= ε-closure ({q2}) = {q2} Similarly, δ' ({q1, q2}, a) = ε-closure (δ ({q1, q2}, a)) = ε-closure ({φ}) = {φ} δ' ({q1, q2}, b) = ε-closure (δ ({q1, q2}, b)) = ε-closure ({q1}) = {q1, q2} δ' ({q1, q2}, c) = ε-closure (δ ({q1, q2}, c))

= ε-closure ({q2}) = {q2} Similarly, δ' ({q2}, a) = ε-closure (δ ({q2}, a)) = ε-closure ({φ}) = {φ} δ' ({q2}, b) = ε-closure (δ ({q2}, b)) = ε-closure ({φ}) = {φ} δ' ({q2}, c) = ε-closure (δ ({q2}, c)) = ε-closure ({q2}) = {q2} The equivalent DFA is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| δ: | a | b | c |
| \* {q0, q1, q2} | {q0, q1, q2} | {q1, q2} | {q2} |
| \*{q1, q2} | φ | {q1, q2} | {q2} |
| \*{q2} | φ | φ | {q2} |
| φ | φ | φ | φ |

φ

{

q

0

, q

2

}

{

q

2

}

{

q

0

, q

0

q

2

}

Start

a, b

a

b

c

c

c

a

a, b, c

b

**Theorem 1:**

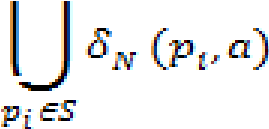
*For any NFA, N = (QN, ∑,* δ*N, q0, FN) accepting language L* ⊆ *∑\* there is a DFA D =*

*(QD, ∑,* δ*D, q0’,FD) that also accepts L i.e. L (N) = L (D).*

#### Proof: -

The DFA D, say can be defined as;

QD = 2QN , q0’ = {q0}

Let S = { p1, p2, p3, … pk} ∈ QD. Then for S ∈ QD & a ∈ ∑, δD (s, a) = 

FD = {S / S ∈ QD & S ∩ FN ≠ φ}

The fact that D accepts the same language as N is as; for any string w ∈ ∑\*;

N (q0, w) = δD’ (q0’, w)



Thus, we prove this fact by induction on length of w.

*Basis Step:*

Let |w| = 0, then w = ε,

∴  D (q0’, w) =  D (q0’, ε) = q0’ = {q0}



Also,

N (q0, w) =  N (q0, ε) = {q0}



∴  D (q0’, w) =  N (q0, w) is true |w| = 0



*Induction Step:*

Let |w| = n + 1 is a string such that w = xa & |x| = n, |a| = 1; a being last symbol.

Let the inductive hypothesis is that x satisfies.

Thus,

D (q0’, x) =  N (q0, x), let these states be {p1, p2, p3, … pk}



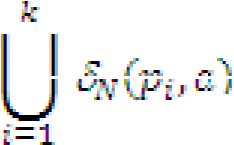
Now,

N (q0, w) =  N (q0, xa)



= δN ( N (q0, x), a)



= δN ({p1, p2, p3, … pk}, a) [Since, from inductive step] =     --------------------- (I)

Also,

D (q0’, w) =  D (q0’, xa)



= δD ( D (q0’, x), a)



= δD ( N (q0, x), a) [Since, by the inductive step as it is true for x]



= δD ({p1, p2, p3, … pk}, a) [Since, from inductive step]

Now, we know from subset construction, it tells us that, δD ({p1, p2, p3, … pk}, a) = 

Thus, we conclude,

D (q0’, w) = -------------------- (II)



Here, from (I) & (II),

N (q0, w) =  D (q0’, w)



Hence, if this relation is true for |w| = n, then it is also true for |w| = n + 1.

∴DFA D & NFA N accepts the same language.

i.e. L (D) = L (N)

Proved

**Theorem 2:**

A language L is accepted by some NFA if L is accepted by some DFA.

**Proof:**

Consider we have a DFA D = (QD, ∑, δN, q0, FD).

This DFA can be interpreted as a NFA having the transition diagram with exactly one choice of transition for any input.

i.e. N = (QN, ∑, δN, q0’, FN) where, QN = QD, FN = FD, q0’ = q0

And δN defined as if δN (q, a) = p then δN (q, a) = {p}.

Then to show if L is accepted by D then it is also accepted by N, it is sufficient to show, for any string w ∈ ∑\*,

D (q0, w) =  N (q0, w)



We can proof this fact using induction on length of the string.

*Basis step: -*

Let |w| = 0 i.e. w = ε

∴ D (q0, w) =  D (q0, ε) = q0



N (q0, w) =  N (q0, ε) = {q0}



∴  D (q0, w) =  N (q0, w) for |w| = 0 is true.



*Induction: -*

Let |w| = n + 1 & w = xa. Where |x| = n & |a| = 1; a being the last symbol.

Let the inductive hypothesis is that it is true for w = x.

∴if  D (q0, x) = p, then  N (q0, x) = {p}



i.e.  D (q0, x) =  N (q0, x)



Now,

D (q0, w) =  D (q0, xa)



= δD ( D (q0, x), a)



= δD (p, a) [ from inductive step]

= r, say

Now,

N (q0, w) =  N (q0, xa)



= δN ( N (q0, x), a)



= δN ({p}, a) [  from inductive step]

As the NFA N is constructed from the DFA D, where transition diagram of NFA consists exactly one choice of transition for any input. So, taking input a, from any state, the transition of NFA will be same as DFA.

i.e.  N ({p}, a) = {r}



∴ N ({p}, a) = {r}



Hence,  D (q0, w) =  N (q0, w)



**Proved.**

# A Language L is accepted by some DFA if and only if L is accepted by some NFA.