2 – Non Deterministic Finite Automata

2.1Nondeterministic Finite Automata.

- An NFA can be in several states at once.
- Each NFA accepts a language that is also accepted by some DFA.
- NFAs are often easier than DFAs.
- We can always convert an NFA toa DFA.

The difference between the DFA and NFA is the type of transition function δ

- For NFA δ returns *a set of zero, one or more states.*
- For DFA δ *returns exactly one state.*

2.1.1 NFA: Formal definition:

A nondeterministic finite automaton (NFA) is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states.
- Σ is a finite set of input symbols.
- $q_0 \in Q$ is the start state.
- F $(F \subseteq Q)$ is the set of final or accepting states.
- δ, the transition function is a function that takes a state in Q and an input symbol in Σ as arguments and returns a subset of Q.

$\Delta(q, a) = S$, where 'q' is the current state, 'a' the next input symbol and 'S' is the set of states the NFA enters after the transition ($S \subseteq Q$).

Example 1: Construct an NFA to accept all the binary strings that end with 01.

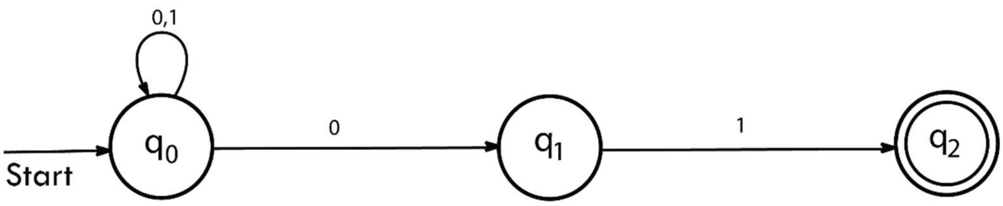

Fig 2.1

The above NFA is specified formally as

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, start state is q_0, $F = \{q_2\}$

Transition Function is represented by the following table.

δ	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2*	$\emptyset$	$\emptyset$

Table 2.1

2.1.2 The extended transition function:

The extended transition function $\hat{\delta}$ of an NFA takes a state '*q*' and a string '*w*' as inputs and returns the set of states that the NFA is in, if it starts in state '*q*' and processes the string '*w*'.

Formal definition of the extended transition function:

Definition by induction on the length of the input string

Basis: $\hat{\delta}(q, \varepsilon) = q$ (string length is 0)

If we are in a state '*q*' and read no inputs, then we are still in state '*q*'.

Induction:

- Suppose '*w*' is a string of the form '*xa*'; that is '*a*' is the last symbol of '*w*', and '*x*' is the rest of '*w*'.
- Also suppose that $\hat{\delta}(q, x) = \{p_1, p_2, \ldots, p_k\}$ i.e., the set of states the NFA is in after reading the string 'x'
- Let $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \ldots, r_m\}$ i.e., the set of states the NFA is in after reading the last symbol 'a'
- Then $\hat{\delta}(q, w) = \{r_1, r_2, \ldots, r_m\}$

2.1.3 The language of an NFA:

The language of a NFA $A = (Q, \Sigma, \delta, q_0, F)$, denoted L (A) is defined by

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

The language of A is the set of strings $w \in \Sigma^*$ such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.

Example 2: Construct an NFA to accept all the binary strings that start with 01.

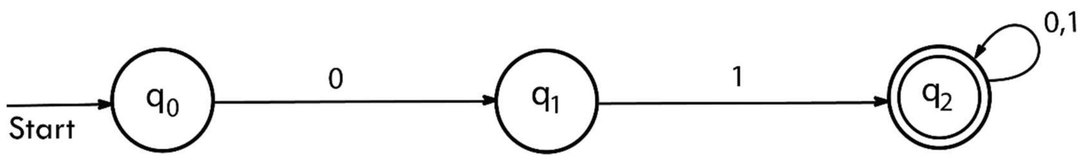

Fig 2.2

Example 3: Construct an NFA to accept all the strings on Σ = {a, b} such that the second symbol is a.

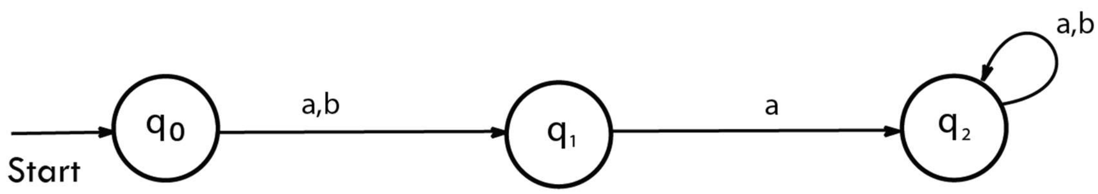

Fig 2.3

Example 4: Construct an NFA to accept all the strings on Σ = {a, b} starting and ending with a.

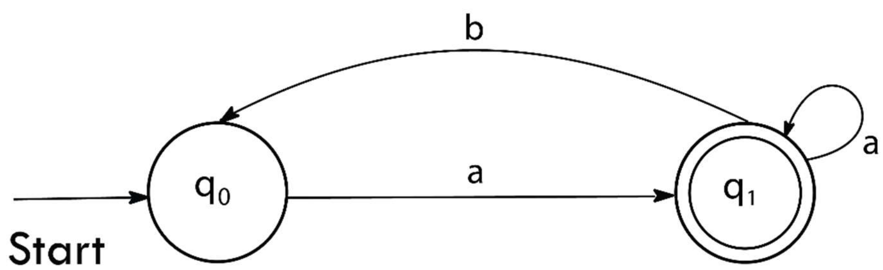

Fig 2.4

Example 5: Construct an NFA to accept all the strings on Σ = {0, 1} starting with 0 and ending with 1.

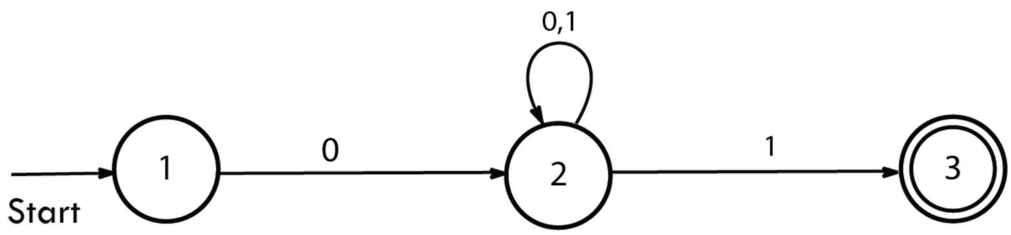

Fig 2.5

Example 6: Construct an NFA to represent the language

L = {awa | w ∈ {a, b}*} i.e., all strings with the first and last symbol as '**a'** and these two a's separated by any string on Σ = {a, b}

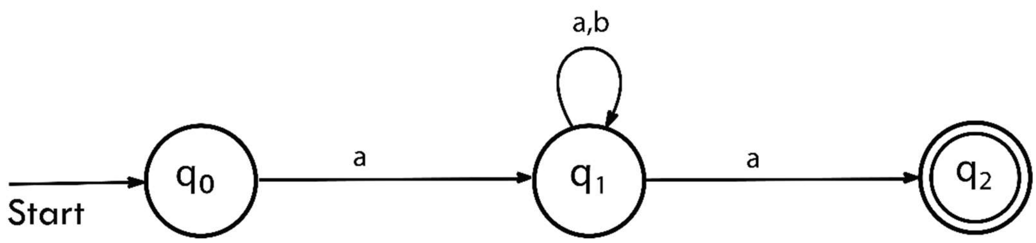

Fig 2.6

Example 7: Construct an NFA to represent the language

L = {aawbb | w ∈ {a, b}*} i.e., all strings starting with at least two a's and ending with at least two b's.

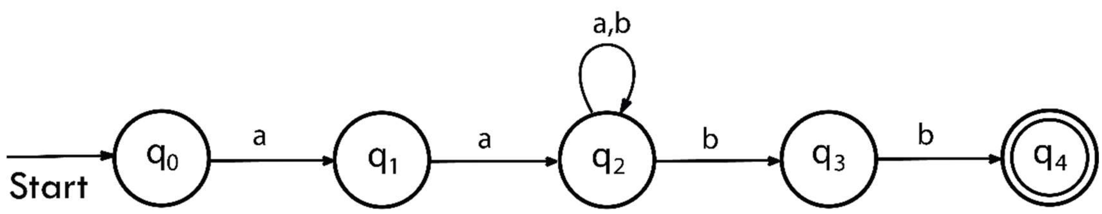

Fig 2.7

Example 8: Construct an NFA to represent the language

L = {w | w ∈ {a, b}*} i.e., all strings containing the substring ***ab***at least once.}

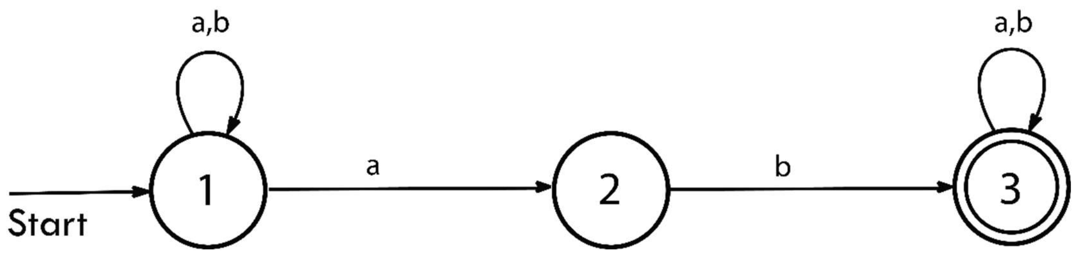

Fig 2.8

Example 9: Construct an NFA to all binary strings such that the third symbol from the end is 0.

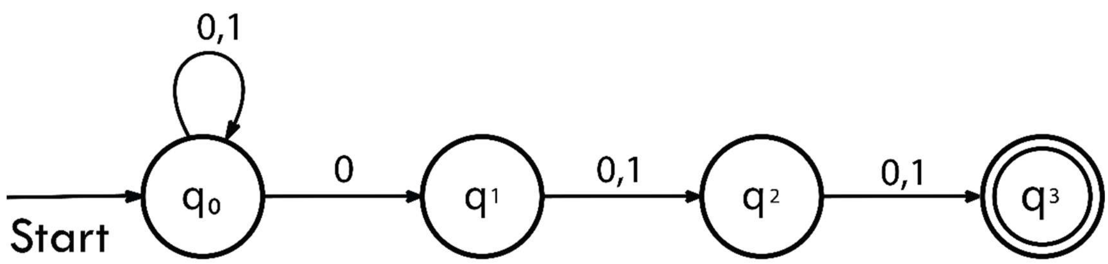

Fig 2.9

2.2Equivalence of Deterministic and Nondeterministic Finite Automata:

- Every language that can be described by some NFA can also be described by some DFA.
- The DFA in practice has about as many states as the NFA, although it has more transitions.
- In the worst case, the smallest DFA can have 2^n (for a smallest NFA with n states).

Proof: DFA can do whatever NFA can do:

The proof involves an important construction called "*subset construction*" because it involves constructing all subsets of the set of states of NFA. (How one formally describes one automaton in terms of the states and transitions of another).

2.2.1 From NFA to DFA: (Subset Construction Method)

Input: NFA

Output: An equivalent DFA

- We have a NFA $N = (Q_N, \Sigma, \delta_N, q_N, F_N)$.
- The goal is the construction of a DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ such that $L(D) = L(N)$.
- Input alphabets are the same.
- The start state in D is the set containing only the start state of N i.e., $q_D = \{q_N\}$.
- Q_D is the set of subsets of Q_N, i.e., Q_D is the power set of Q_N. If Q_N has n states Q_D will have 2^n states. Often, not all of these states are accessible from the start state. Inaccessible states can be “thrown away”, so effectively, the number of states of D may be much smaller than 2^n.
- F_D is the set of subsets S of Q_N such that $S \cap F_N = \emptyset$. That is, F_D is all sets of N’s states that include at least one accepting state of N.
- For each set $S \subseteq Q_N$ and for each input symbol $a \in \Sigma$,

  $\delta_D(S, a) = \bigcup_{p\ in\ S} \delta_N(p, a)$

  - To compute $\delta_D(S, a)$ we look at all the states p in S.
  - See what states the NFA goes to from ‘p’ on input ‘a’.
  - Take the union of all those states.

Example:NFA to accept strings on $\Sigma = \{0, 1\}$ ending with 01.

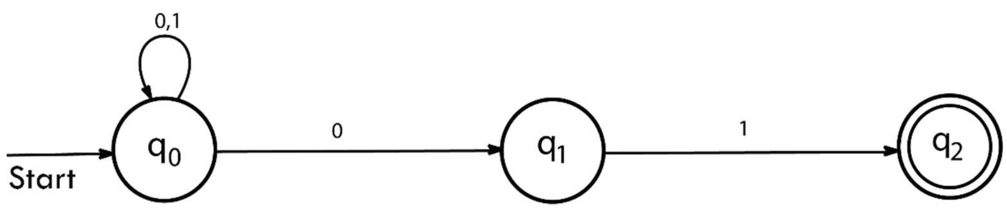

Fig 2.10

Convert the above NFA to its equivalent DFA.

$Q_N = \{q_0, q_1, q_2\}$

$Q_D = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

$\Sigma = \{0, 1\}$ same for both NFA and DFA.

$q_D = \{q_0\}$ the start state for DFA

$F_D = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

δ_D	0	1
∅	∅	∅
→$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	∅	$\{q_2\}$
$\{q_2\}$*	∅	∅
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$*	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1, q_2\}$*	∅	$\{q_2\}$
$\{q_0, q_1, q_2\}$*	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Table 2.1

To make the point simpler, we can give new names for the states e.g., A for Ø, B for $\{q_1\}$ and soon. The transition table for the DFA becomes

δ_D	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

Table 2.2

In the above DFA the start state is B and the only states reachable from the start state B are B, E and F. The other five states are not accessible and hence may be ignored.

δ_D	0	1
→B	E	B
E	E	F
F*	E	B

Table 2.3

The subset construction method (Converting NFA to DFA) takes exponential time i.e., to convert an n state NFA to DFA, 2^n states are to be examined. This can be avoided by performing “lazy evaluation” on subsets i.e., listing only the states accessible from the start state.

2.2.2 From NFA to DFA: (Lazy Evaluation Method)

Basis: The subset containing the start state of NFA is accessible.

Induction: Suppose the subset S is accessible then for each input $a \in \Sigma$, compute the set of states $\delta_D(S, a)$; these sets of states will also be accessible.

Example: Convert the NFA in Fig to its equivalent DFA using lazy evaluation method.

Start state of DFA is $\{q_0\}$

Step 1: $\delta_D(\{q_0\}, 0) = \delta_N(q_0, 0) = \{q_0, q_1\}$

$\delta_D(\{q_0\}, 1) = \delta_N(q_0, 1) = \{q_0\}$

One of the two states computed above $\{q_0\}$ is “old”.

The state $\{q_0, q_1\}$ is new and transitions from it must be considered.

Step 2: $\delta_D (\{q_0, q_1\}, 0) = \delta_N (q_0, 0) \cup \delta_N (q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$

$\delta_D (\{q_0, q_1\}, 1) = \delta_N (q_0, 1) \cup \delta_N (q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

One of the two states computed above $\{q_0, q_1\}$ is “old”.

The state $\{q_0, q_2\}$ is new and transitions from it must be considered.

Step 3: $\delta_D (\{q_0, q_2\}, 0) = \delta_N (q_0, 0) \cup \delta_N (q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$

$\delta_D (\{q_0, q_2\}, 1) = \delta_N (q_0, 1) \cup \delta_N (q_2, 1) = \{q_0\} \cup \emptyset = \{q_0\}$

Both the states $\{q_0, q_1\}$ and $\{q_0\}$ are “old”.

The subset construction has converged.

We know all the accessible states and their transitions.

The resulting DFA is

δ_D	0	1
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}^*$	$\{q_0, q_1\}$	$\{q_0\}$

Table 2.4

Example: Convert the following NFA to DFA.

δ_N	0	1
$\rightarrow p$	{p, q}	{p}
q	{r}	{r}
r	{s}	$\emptyset$
*s	{s}	{s}

Table 2.5

Start state of DFA is {p}

Step 1: $\delta_D (\{p\}, 0) = \delta_N (p, 0) = \{p, q\}$

$\delta_D(\{p\}, 1) = \delta_N(p, 1) = \{p\}$

One of the two states computed above {p} is "old".

The state {p, q} is new and transitions from it must be considered.

Step 2: $\delta_D(\{p, q\}, 0) = \delta_N(p, 0) \cup \delta_N(q, 0) = \{p, q\} \cup \{r\} = \{p, q, r\}$

$\delta_D(\{p, q\}, 1) = \delta_N(p, 1) \cup \delta_N(q, 1) = \{p\} \cup \{r\} = \{p, r\}$

Both the states computed above {p, q, r} and {p, r} are new and transitions from them must be considered.

Step 3: $\delta_D(\{p, r\}, 0) = \delta_N(p, 0) \cup \delta_N(r, 0) = \{p, q\} \cup \{s\} = \{p, q, s\}$

$\delta_D(\{p, r\}, 1) = \delta_N(p, 1) \cup \delta_N(r, 1) = \{p\} \cup \emptyset = \{p\}$

The state {p} is old and the state {p,q, s} is new.

Hence transitions from {p, q, s} must be considered.

Step 4: $\delta_D(\{p, q, r\}, 0) = \delta_N(p, 0) \cup \delta_N(q, 0) \cup \delta_N(r, 0) = \{p, q\} \cup \{r\} \cup \{s\} = \{p, q, r, s\}$

$\delta_D(\{p, q, r\}, 1) = \delta_N(p, 1) \cup \delta_N(q, 1) \cup \delta_N(r, 1) = \{p\} \cup \{r\} \cup \emptyset = \{p, r\}$

The state {p, r} is "old". The state {p, q, r, s} is new.

Therefore transitions from {p, q, r, s} must be considered.

Step 5: $\delta_D(\{p, q, s\}, 0) = \delta_N(p, 0) \cup \delta_N(q, 0) \cup \delta_N(s, 0) = \{p, q\} \cup \{r\} \cup \{s\} = \{p, q, r, s\}$

$\delta_D(\{p, q, s\}, 1) = \delta_N(p, 1) \cup \delta_N(q, 1) \cup \delta_N(s, 1) = \{p\} \cup \{r\} \cup \{s\} = \{p, r, s\}$

The state {p, q, r, s} is "old". The state {p, r, s} is new.

Therefore transitions from {p, r, s} must be considered.

Step 6: $\delta_D(\{p, r, s\}, 0) = \delta_N(p, 0) \cup \delta_N(r, 0) \cup \delta_N(s, 0) = \{p, q\} \cup \{s\} \cup \{s\} = \{p, q, s\}$

$\delta_D(\{p, r, s\}, 1) = \delta_N(p, 1) \cup \delta_N(r, 1) \cup \delta_N(s, 1) = \{p\} \cup \emptyset \cup \{s\} = \{p, s\}$

The state {p, q, s} is "old". The state {p, s} is new.

Therefore transitions from {p, s} must be considered.

Step 7: δ_D ({p, q, r, s}, 0) = δ_N (p, 0) ∪ δ_N (q, 0) ∪ δ_N (r, 0) ∪ δ_N (s, 0) = {p, q} ∪ {r} ∪{s}∪{s} = {p, q, s}

δ_D ({p, q,r, s}, 1) = δ_N (p, 1) ∪δ_N (q, 1) ∪δ_N (r, 1) ∪ δ_N (s, 1) = {p} ∪{r} ∪ ∅∪ {s} = {p, r, s}

Both the states {p, q, r, s} and {p, r, s} are "old".

Step 8: δ_D ({p, s}, 0) = δ_N (p, 0) ∪ δ_N (s, 0) = {p, q} ∪ {s} = {p, q, s}

δ_D ({p, s}, 1) = δ_N (p, 1) ∪ δ_N (s, 1) = {p} ∪ {s} = {p, s}

Both the states {p, q, s} and {p, s} are "old".

The equivalent DFA is

δ_D	0	1
→{p}	{p, q}	{p}
{p, q}	{p, q, r}	{p, r}
{p, r}	{p, q, s}	{p}
{p, q, r}	{p, q, r, s}	{p, r}
*{p, q, s}	{p, q, r, s}	{p, r, s}
*{p, r, s}	{p, q, s}	{p, s}
*{p, q, r, s}	{p, q, r, s}	{p, r, s}
*{p, s}	{p, q, s}	{p, s}

Table 2.6

2.3Finite Automata with Epsilon- Transitions (ε– NFA)

- Ɛ- NFA is an extension of NFA.
- The new feature is that a transition on Ɛ is allowed i.e., make a transition spontaneously, without receiving an input symbol.
- This capability does not change the class of languages that is represented by finite automata, but it gives us some added programming convenience.

2.3.1 The formal Notation For anƐ – NFA

- Ɛ – NFA is represented exactly like an NFA, with one exception: the transition on Ɛ.
- Formally we represent an Ɛ – NFA A by A = (Q, Σ, δ, q_0, F) where all components have their same interpretation as for NFA, except that δ is now a function that takes arguments:
    - A state in Q and
    - A member of Σ ∪{Ɛ} we require that Ɛ cannot be a member of Σ.
- δ (q, a) =S, where
  'q' is the current state,
  'a' the next input symbol or Ɛ and
  'S' is the set of states the Ɛ -NFA enters after the transition (S⊆ Q).

2.3.2 Epsilon-Closures

Ɛ –CLOSURE(q) is the set of all the states that can be reached from 'q' only through Ɛ transitions.

We define Ɛ –CLOSURE(q)recursively, as follows

Basis: State 'q' is in Ɛ –CLOSURE(q)

Induction: If state 'p' is in Ɛ –CLOSURE (q), and there is a transition from 'p' to 'r'labelledƐ, then 'r' is in Ɛ –CLOSURE (q).

2.3.3 Extended Transitions and Languages for Ɛ-NFA

$\hat{\delta}$ (q, w) is the set of all the states that can be reached from 'q' after reading the string 'w'.

The definition of $\hat{\delta}$ (q, w):

Basis: $\hat{\delta}$(q, Ɛ) = ECLOSE (q) or Ɛ –CLOSURE (q)

Induction: Suppose 'w' is the string ofthe formxa, where a ∊ Σ is the last symbol of w and x is the string consistingof all other symbols.

We compute$\hat{\delta}$(q, w) as follows

Let $\hat{\delta}$(q, x) = {p_1, p_2 . . .p_k} i.e., the set of states that can be reached from q following a path labelled x.

Let $\cup_{i=1}^{k} \delta(pi, a) =$ {r_1, r_2, . . .r_m }. The r_j are some of the states we can reach from q along paths labelled w. The additional states we can reach are found from the $r_j$by followingƐ- labelledtransitions in the step below

Then $\hat{\delta}$ (q, w) = $\cup_{i=1}^{m} \mathcal{E} - closure(ri)$

2.3.4 The language of anƐ-NFA

Let theƐ-NFAE= (Q, Σ, δ, q_0, F)

Then the language ofE

$$L(E) = \{w \mid \hat{\delta}(q, w) \cap F \neq \emptyset\}$$

2.3.5 Eliminating Ɛ-Transitions

Let E = (Q_E, Σ, δ_E, q_0, F_E) be anƐ-NFA.

Then the equivalent DFA D = (Q_D, Σ, δ_D, q_D, F_D) is defined as follows.

1) Q_D is the set of subsets of Q_E
2) q_D = ECLOSE(q_0)
3) F_D = {S | S is in Q_D and S ∩ F_E≠∅}. Those sets of states that contain at least one accepting state of E.
4) δ_D(S, a) is computed, for all a in Σ and sets S in $Q_E$by
    - Let S = {p_1, p_2, . . .,pk}
    - Compute$\cup_{i=1}^{k} \delta E(pi, a)$; let this set be {$r_1$, r_2,⋯ ,r_m}
    - Then δ_D(S, a) = $\cup_{j=1}^{m} ECLOSE(rj)$

Example: Consider the following $\mathcal{E}$- NFA

δ_N	$\mathcal{E}$	a	b	c
→p	∅	{p}	{q}	{r}
q	{p}	{q}	{r}	∅
*r	{q}	{r}	∅	{p}

Table 2.7

a) Compute $\mathcal{E}$-closure of each state.

b) Convert the automaton to DFA.

Solution:

$\mathcal{E}$-closure (p) = {p}

$\mathcal{E}$-closure (q) = {p, q}

$\mathcal{E}$-closure (r) = {p, q, r}

Converting to DFA

q_d the start state of DFA = $\mathcal{E}$-closure (p) ={p}

Step1: Transition from {p}

δ_E ({p}, a) = {p}

δ_D ({p}, a) =$\mathcal{E}$-closure {p}= {p}

δ_E ({p}, b) = {q}

δ_D ({p}, b) = $\mathcal{E}$-closure ({q}) = {p, q}

δ_E ({p}, c) = {r}

δ_D ({p}, c) = $\mathcal{E}$-closure (r) = {p, q, r}

{p} is an "old" state, whereas {p, q} and {p, q, r} are new states.

Hence transitions from {p, q} and {p, q, r} must be considered.

Step2: Transitions from {p, q}

δ_E (p, a) ∪δ_E (q, a) = {p, q}

δ_D ({p, q}, a) = Ɛ-closure (p)∪ Ɛ-closure (q)= {p}∪ {p, q} = {p, q}

δ_E (p, b)∪δ_E (q, b) = {q}∪ {r} = {q, r}

δ_D ({p, q}, b) = Ɛ-closure (q) ∪ Ɛ-closure (r) = {p, q} ∪{p, q, r} = {p, q, r}

δ_E (p, c) ∪ δ_E (q, c) = {r} ∪ø= {r}

δ_D ({p, q}, c) = Ɛ-closure (r) = {p, q, r}

{p, q} is an “old” state, whereas and {p, q, r} is a new state.

Hence transitions from {p, q, r} must be considered.

Step 3: Transitions from {p, q, r}

δ_E (p, a) ∪ δ_E (q, a) ∪ δ_E (r, a) = {p, q, r}

δ_D ({p, q, r}, a) = Ɛ-closure (p) ∪Ɛ-closure (q) ∪Ɛ-closure (r) = {p} ∪ {p, q} ∪ {p, q, r} = {p, q, r}

δ_E (p, b) ∪ δ_E (q, b) ∪ δ_E (r, b) = {q, r}

δ_D ({p, q, r}, b) = Ɛ-closure (q) ∪Ɛ-closure (r) = {p, q} ∪ {p, q, r} = {p, q, r}

δ_E (p, c) ∪ δ_E (q, c) ∪ δ_E (r, c) = {p, r}

δ_D ({p, q, r}, c) = Ɛ-closure (p) ∪Ɛ-closure (r) = {p, q, r}

{p} and {p, q} and {p, q, r} are all “old” states.

Hence no new transitions need to be considered.

The equivalent DFA is

δ_D	a	b	c
→{p}	{p}	{p, q}	{p, q, r}

{p, q}	{p, q}	{p, q, r}	{p, q, r}
*{p, q, r}	{p, q, r}	{p, q, r}	{p, q, r}

Table 2.8