Unit 2

Finite State Automata and Regular Expression

Introduction

Regular expressions

- Are a formal notation for characterizing the text sequence.
- Regular expressions are an algebraic way to describe languages.
- Are mainly used in web search examples, word processors.

Finite automata

- Are formal (or abstract) machines for recognizing patterns.
- Are mathematical device used to implement regular expressions.
- These machines are used extensively in compilers and text editors, which must recognize patterns in the input

Regular Expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions

- Regular expressions are a very powerful tool to do string matching and processing
- RE search requires pattern and corpus of text.
- Allows you to do things like:
  - Match a string that starts with a lowercase letter, then is followed by 2 numbers and ends with “ing” or “ion”
  - Replace all occurrences of one or more spaces with a single space
  - Split up a string based on whitespace or periods or commas or …

Terminology of Languages:

- *An alphabet is a finite, non-empty set of symbols*
- We use the symbol ∑ (sigma) to denote an alphabet
- Examples:
  - Binary: ∑ = {0,1}
  - All lower case letters: ∑ = {a,b,c,..z}
  - Alphanumeric: ∑ = {a-z, A-Z, 0-9}

Terminology of Languages

- *A string or word is a finite sequence of symbols chosen from ∑*
- ***Empty string is ε (or "epsilon")***
- Length of a string *w,* denoted by "|*w*|", is equal to the *number of (non- ε) characters in the string*
  - *E.g., x = 010100* *|x| = 6*
  - *x = 01 ε 0 ε 1 ε 00 ε* *|x| = ?*

- *xy = concatenation* of two strings *x* and *y*

Terminology of Languages

L is a said to be a language over alphabet ∑, only if L $\subseteq \sum^*$

➔ this is because ∑* is the set of all strings (of all possible length including 0) over the given alphabet ∑

Examples:

1. Let L be *the* language of all strings consisting of *n* 0's followed by *n* 1's:
   L = {ε, 01, 0011, 000111,…}
2. Let L be *the* language of all strings of with equal number of 0's and 1's:
   L = {ε, 01, 10, 0011, 1100, 0101, 1010, 1001,…}

Examples:

- L_1 = {a,b,c,d} L_2 = {1,2}
- L_1L_2 = {a1,a2,b1,b2,c1,c2,d1,d2}
- $L_1 \cup L_2$ = {a,b,c,d,1,2}
- L_1^3 = all strings with length three (using a,b,c,d)
- L_1^* = all strings using letters a,b,c,d and empty string
- L_1^+ = doesn't include the empty string

Regular Expressions: Literals

- Basic regular expression consists of a single literal character
- We can put any string in a regular expression
- /test/ - matches any string that has “test” in it
- /this class/ - matches any string that has “this class” in it
- /Test/. case sensitive: matches any string that has “Test” in it

Regular Expressions: Meta characters

[]	A set of characters (character class)
\	Signals a special sequence (can also be used to escape special characters)
.	Any character (except newline character)
^	Starts with (anchor)
$	Ends with (anchor)
*	Zero or more occurrences
+	One or more occurrences
?	Zero or one occurrence
{}	Exactly the specified number of occurrences
|	Either or
\b	Word boundary (anchor)
\B	Non boundary (anchor)

Anchors are special characters that anchor regular expressions to particular places in string.

Operator precedence hierarchy

Parenthesis	()
Counters	* + ? {}
Sequences and anchors	the ^my end$
Disjunction	|

Regular Expressions: Character Classes

- A set of characters to match:
  - put in brackets: []
  - [abc] matches a single character a or b or c
- Can use - to represent ranges
  - [a-z] is equivalent to [abcdefghijklmnopqrstuvwxyz]
  - [A-D] is equivalent to [ABCD]
  - [0-9] is equivalent to [0123456789]

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

Pattern	Matches	
`[A-Z]`	An upper case letter	Drenched Blossoms
`[a-z]`	A lower case letter	my beans were impatient
`[0-9]`	A single digit	Chapter 1: Down the Rabbit Hole

- **What would the following match?**
- a) /[Tt]est/
- any string with “Test” or “test” in it

b) /[0-9][0-9][0-9][0-9]/

matches any four digits, e.g. a year

- c)/[aeiou][0-9]/
- matches a1 or e4

Regular Expressions: Character Classes

- Can also specify a set NOT to match:
- ^ means all characters EXCEPT those specified
- Carat means negation only when written first in []
- [^a] all characters except ‘a’
- [^0-9] all characters except numbers
- [^A-Z] not an upper case letter

Pattern	Matches	
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither ‘S’ nor ‘s’	I have no exquisite reason”
a^b	The pattern a carat b	Look up a^b now

Regular Expressions: Advanced Operators

Aliases for common set of operators:

RE	Expansion	Match	Example Patterns
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric or underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

Regular expression operators for counting

RE	Match
`*`	zero or more occurrences of the previous char or expression
`+`	one or more occurrences of the previous char or expression
`?`	exactly zero or one occurrence of the previous char or expression
`{n}`	*n* occurrences of the previous char or expression
`{n,m}`	from *n* to *m* occurrences of the previous char or expression
`{n,}`	at least *n* occurrences of the previous char or expression

What would the following match?

- /19\d\d/
- would match any 4 digits starting with 19
- /\s\s/
- matches anything with two adjacent whitespace characters
- /\s[aeiou]..\s/
- any three letter word that starts with a vowel

Regular Expressions

- * matches zero or more of the preceding character
- /ba*d/
- matches any string with:
- bd bad baad baaad
- /A.*A/
- matches any string starts and ends with A
- + matches one or more of the preceding character
- /ba+d/
- matches any string with:
- bad baad baaad baaaad

Regular Expressions

- ? zero or 1 occurrence of the preceding
- /fights?/
- matches any string with “fight” or “fights” in it
- {n,m} matches n to m inclusive
- /ba{3,4}d/
- matches any string with
- baaad
- baaaad

Pattern	Matches	
colou?r	Optional previous char	color colour
oo*h!	0 or more of previous char	oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n		begin begun begun beg3n

Regular Expressions:beginning and end

- ^ marks the beginning of the line
- $ marks the end of the line
- /test/ test can occur anywhere
- /^test/ must start with test
- /test$/ must end with test
- /^test$/ must be exactly test

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$	The end.
.$	The end? The end!

Question Hour

- What is the regular expression if we wanted to match:
- This is very interesting
- This is very very interesting
- This is very very very interesting

A. /This is very+ interesting/
B. /This is (very)+interesting/

- Repetition operators only apply to a single character.
- Use parentheses to group a string of characters.

Regular Expressions: More Disjunction

- The pipe | for disjunction

Pattern	Matches
`groundhog|woodchuck`	`woodchuck`
`yours|mine`	`yours`
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	`Woodchuck`

Introduction to Finite-State-Automata

Finite Automata

- Theory of automata is a theoretical branch of computer science and mathematics.
- It is the study of abstract machines and the computation problems that can be solved using these machines. The abstract machine is called the **automata.**
- **Automata** is a abstract machine which takes some string as input and this input goes through a finite number of states and may enter in the final state.
- An automaton with a finite number of states is called a **Finite automaton.**
- This automaton consists of **states** and **transitions.**
- The **State** is represented by **circles**, and the **Transitions** is represented by **arrows.**

Finite Automata

- Finite automata are used to recognize patterns.
- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
- At the time of transition, the automata can either move to the next state or stay in the same state.
- Finite automata have two possible final states, Accept state or Reject state
- When the input string is processed successfully, and the automata reached its final state, then it will be accepted.

Formal Definition of FA

- A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:
- Q: finite set of states
- ∑: finite set of the input symbol
- q0: initial state
- F: final state
- Δ(q,i): Transition function

Transition Diagram

- A **transition diagram or state transition diagram** is a directed graph which can be constructed as follows:
- There is a node for each state in Q, which is represented by the circle.
- There is a directed edge from node q to node p labeled a if **δ(q, a) = p**
- In the start state, there is an arrow with no source.
- Accepting states or final states are indicating by a double circle.

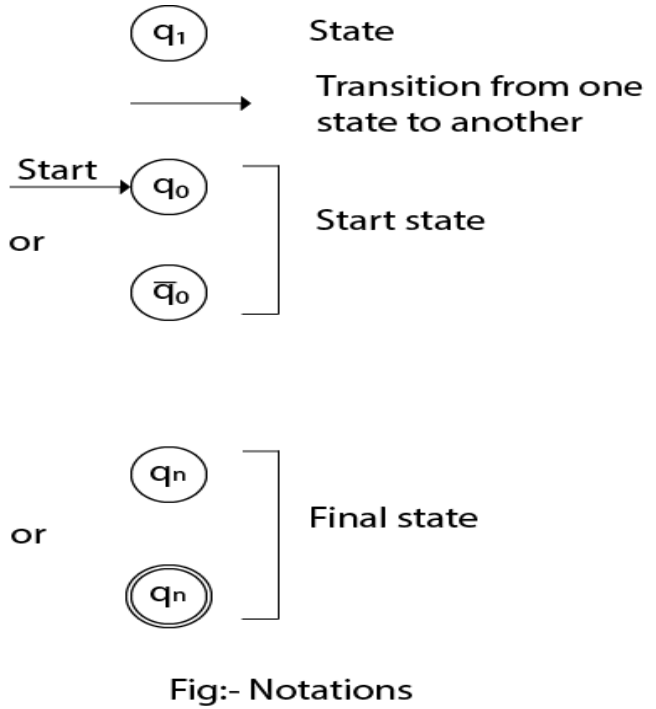
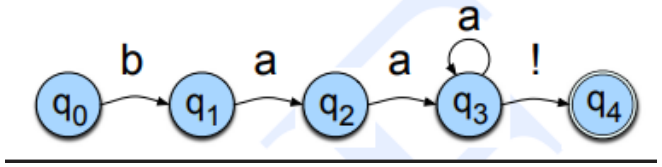


Fig:- Notations

Figure 1:Notations

FSA and State transition table



	Input		
State	b	a	!
0	1	Ø	Ø
1	Ø	2	Ø
2	Ø	3	Ø
3	Ø	3	4
4:	Ø	Ø	Ø

Types of Automata

- There are two types of finite automata:
    1. DFA (Deterministic Finite Automata)
    2. NFA (Non-deterministic Finite Automata)

1. DFA

DFA refers to **deterministic finite automata**. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

2. NFA

NFA stands for non-deterministic finite automata. It is used to transmit any number of states for a particular input. It can accept the null move.

There can be multiple final states in both DFA and NFA.

DFA

- In DFA, there is only one path for specific input from the current state to the next state
- DFA does not accept the null move, i.e., the DFA cannot change state without any input character.
- DFA can contain multiple final states. It is used in Lexical Analysis in Compiler.
- In the following diagram, we can see that from state q0 for **input a,** there is only one path which is going to q1. Similarly, from q0, there is only one path for input b going to q2.

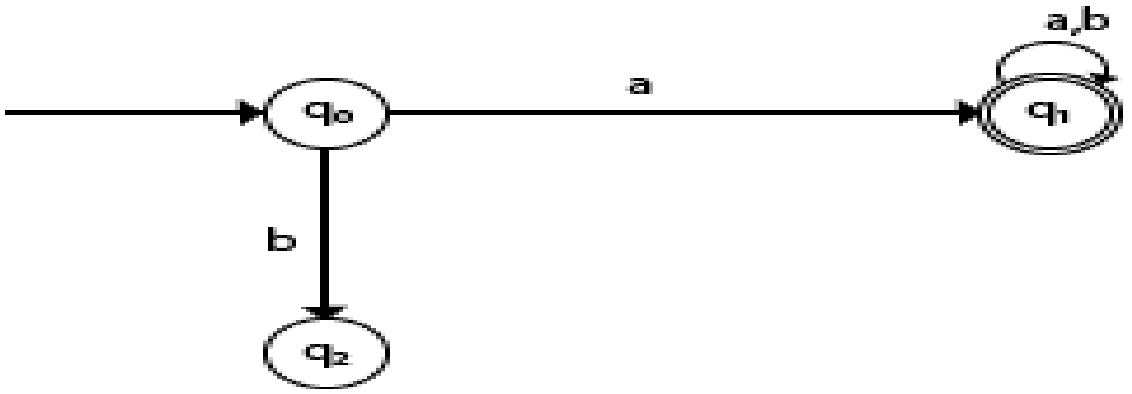


Fig: 2 DFA

Graphical Representation of DFA

- A DFA can be represented by digraphs called state diagram. In which:

1. The state is represented by vertices.
2. The arc labeled with an input character show the transitions.
3. The initial state is marked with an incomming arrow.
4. The final state is denoted by a double circle.

Tabular Representation of DFA

- **Transition Table**
- The transition table is basically a tabular representation of the transition function. It takes two arguments (a state and a symbol) and returns a state (the "next state")
- A transition table is represented by the following things:
  - Columns correspond to input symbols
  - Rows correspond to states
  - Entries correspond to the next state
  - The start state is denoted by an arrow with no source
  - The accept state is denoted by a star

Example for DFA

Example 1: Construct a DFA with ∑ = {0, 1} that accepts all strings starting with 1.

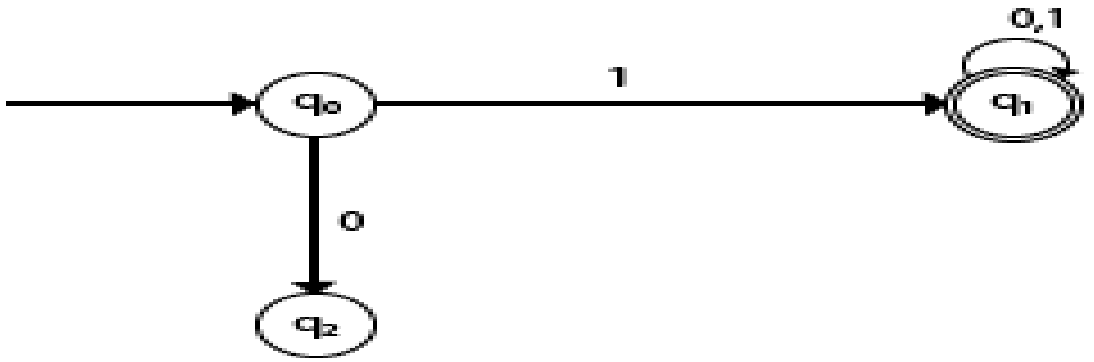


Fig: Transition diagram

d

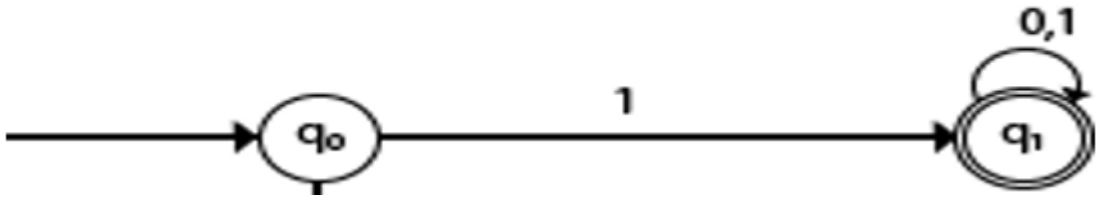


Fig: DFA without Dead State

Figure 3 and 4

- In the diagram, machine initially is in start state **q0** then on receiving input 1 the machine changes its state to **q1**
- From **q0** on receiving 0, the machine changes its state to **qd,** which is the **dead /trap state** (usually not represented)
- From **q1** on receiving input 0, 1 the machine changes its state to q1, which is the final state
- The possible input strings that can be generated are 10, 11, 110, 101, 111……., that means all string starts with 1

Example for DFA

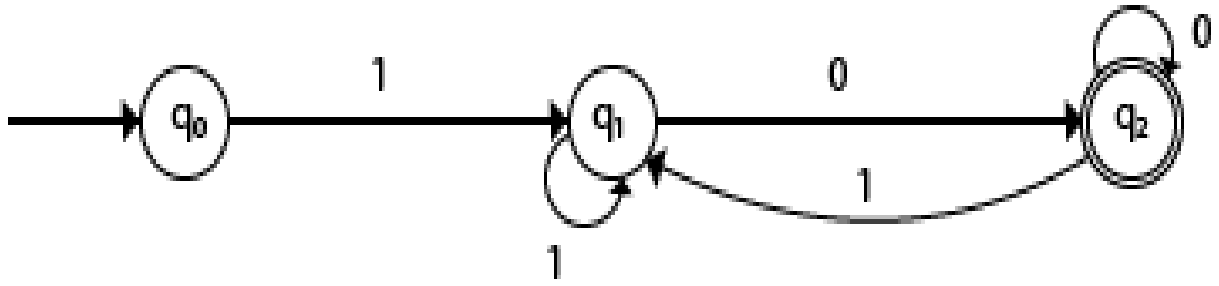
- Transition table for Example 1:

Present State	Next state for Input 0	Next State of Input 1
→q0	-	q1
*q1	q1	q1

Figure 5: Transition table for example 1

Example for DFA

- Example 2:Design a DFA with ∑ = {0, 1} accepts those string which starts with 1 and ends with 0



- In state q1, if we read 1, we will be in state q1, but if we read 0 at state q1, we will reach to state q2 which is the final state
- In state q2, if we read either 0 or 1, we will go to q2 state or q1 state respectively
- Note that if the input ends with 0, it will be in the final state.

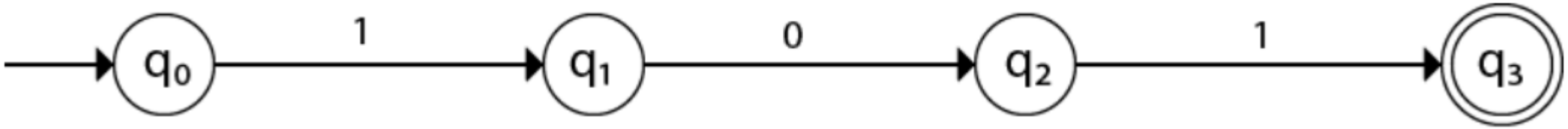
Example for DFA

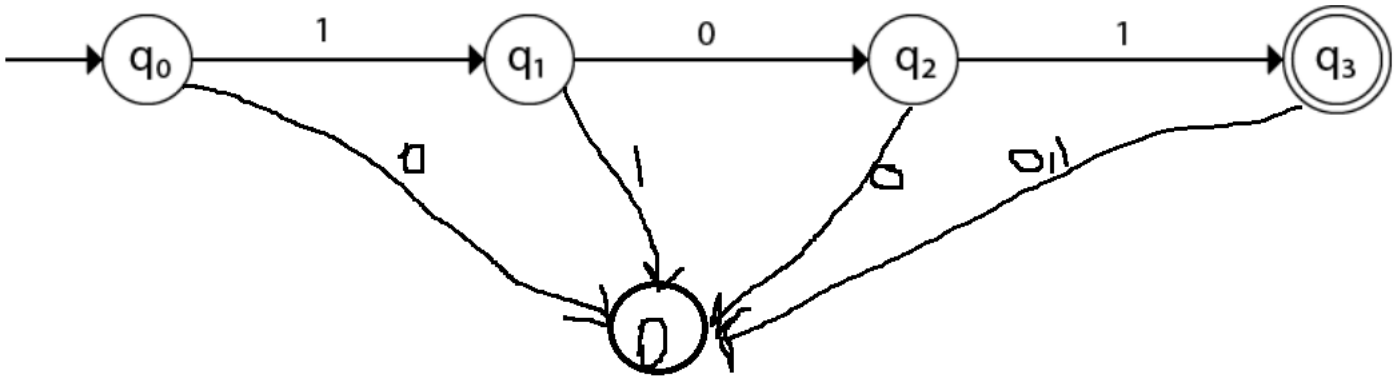
- Transition Table for Example 2:

Present State	Next state for Input 0	Next State of Input 1
→q0	-	q1
q1	q2	q1
*q2	q2	q1

Example for DFA

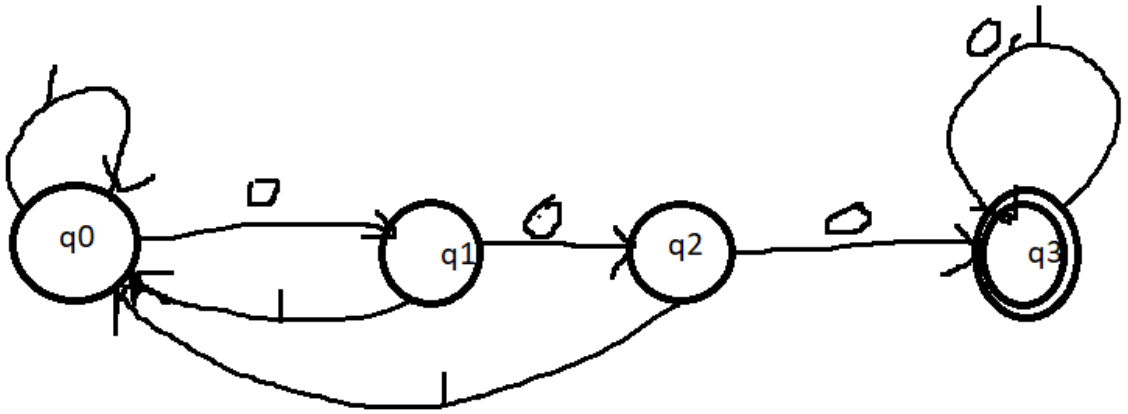
- Example 3:Design a DFA with ∑ = {0, 1} accepts the only input 101





Example for DFA

- Example 4: Design DFA with ∑ = {0, 1} accepts the set of all strings with three consecutive 0's.



NFA

- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state
- Every NFA is not DFA, but each NFA can be translated into DFA
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition
- The transition without consuming an input symbol are called ε transitions

NFA

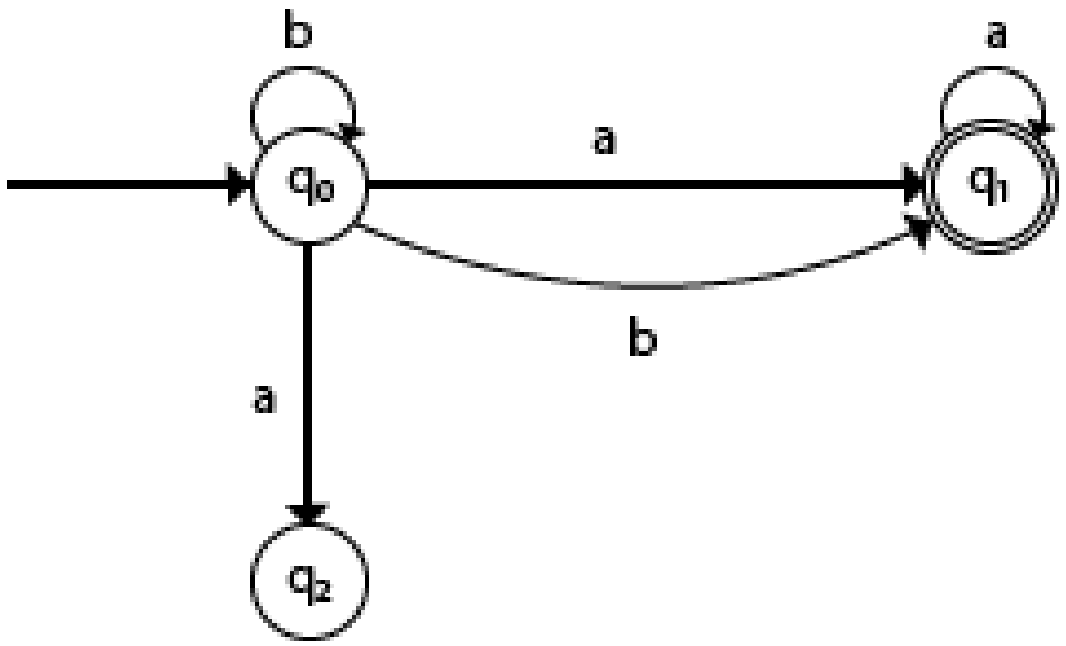
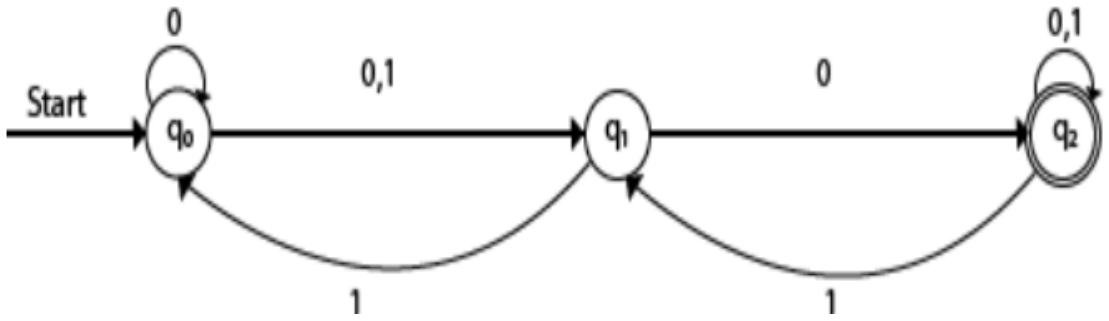


Fig:- NDFA

- In the figure, we can see that from state q0 for input a, there are two next states q1 and q2, similarly, from q0 for input b, the next states are q0 and q1
- Thus it is not fixed or determined that with a particular input where to go next
- Hence this FA is called non-deterministic finite automata.

NFA

Example 1



Present State	Next state for Input 0	Next State of Input 1
→q0	q0, q1	q1
q1	q2	q0
*q2	q2	q1, q2

NFA

- Example 2: NFA with ∑ = {0, 1} accepts all strings begin with 01

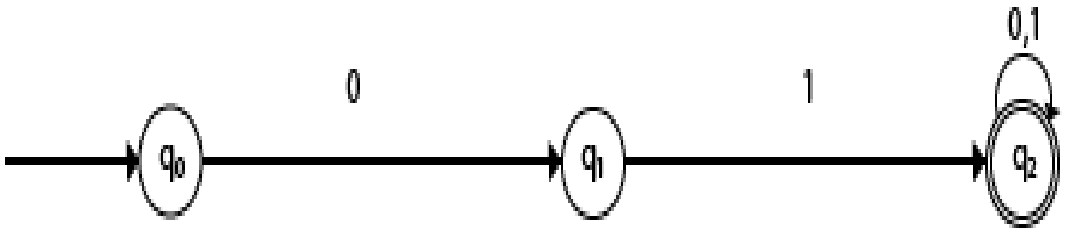
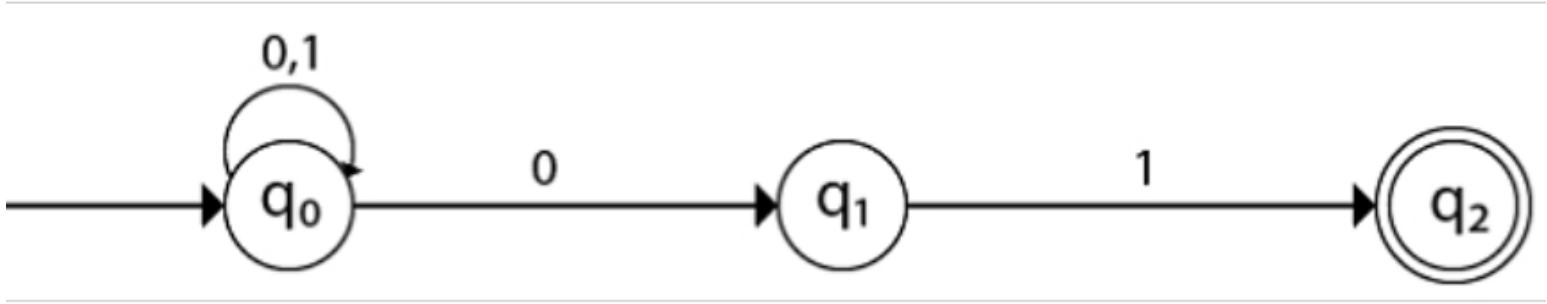


Fig: NFA

Present State	Next state for Input 0	Next State of Input 1
→q0	q1	ε
q1	ε	q2
*q2	q2	q2

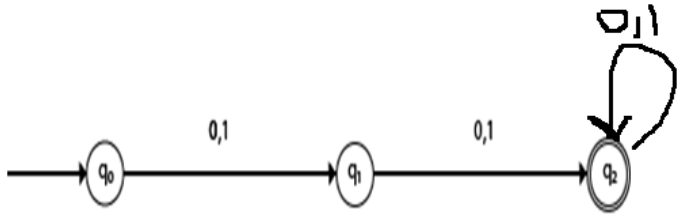
NFA

- Construct an NFA with Σ = {0, 1} which accepts all string ending with 01.



NFA

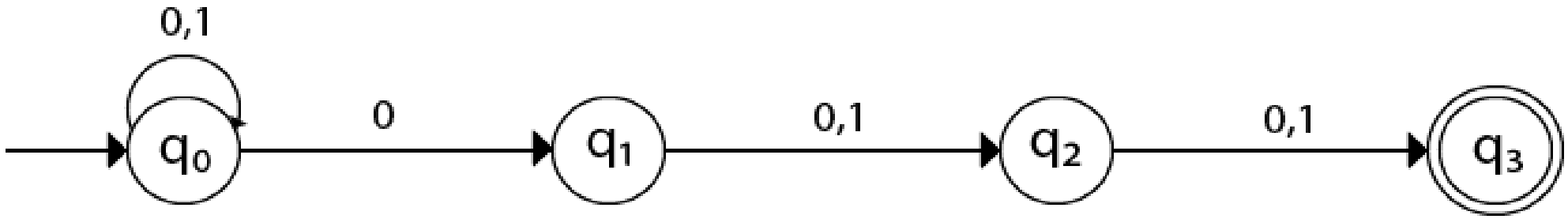
- Example 3:NFA with ∑ = {0, 1} and accept all string of length at least 2.



Present State	Next state for Input 0	Next State of Input 1
→q0	q1	q1
q1	q2	q2
*q2	ε	ε

NFA

- Example 4:Design an NFA with ∑ = {0, 1} accepts all string in which the third symbol from the right end is always 0



Question Hour

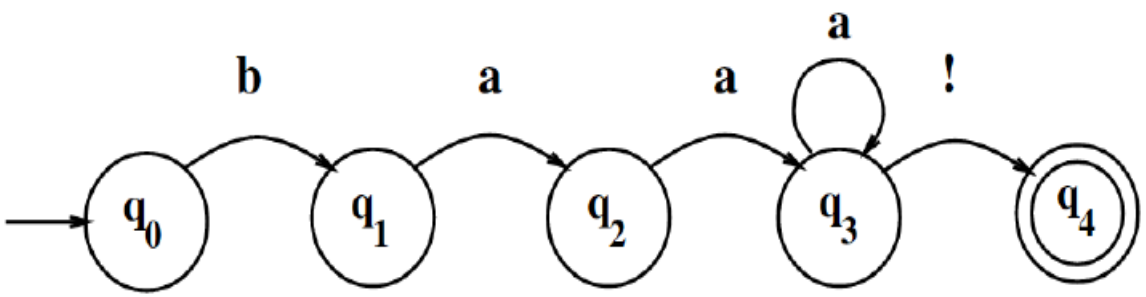
Sheep Language can be any string from the following language L={baa!,baaa!,baaaa!,……}

- Specify the regular expression
- Draw DFA and NFA for L

Question Hour

Regular expression can be specified in two ways for the sheep language: ***/baaa*!/*** or ***/baa+!/***

DFA



Question Hour

NFA