UNIT-I

Strings, Alphabets, Language and Operations

- Strings of characters are fundamental building blocks in computer science.
- **Alphabet** is defined as a non empty finite set or nonempty set of symbols.
- The members of alphabet are the symbols of the alphabet.
- We use capital Greek letter Σ to designate the alphabets and Γ (pronounced as gamma) to designate the typewriter font for symbols.
  - ***Examples:***
  - $\Sigma_1 = \{0, 1\}$
  - $\Sigma_2 = \{a, b, \ldots, z\}$, the set of all lower case letters
  - The set of all ASCII characters or the set of all printable ASCII characters
  - $\Gamma = \{0, 1, x, y, z\}$
- ✓ **String** over an alphabet is a finite sequence of symbols from the alphabet, usually written next to one another and not separated by commas.
- ✓ If $\Sigma_1 = \{0, 1\}$ then ***01001*** is a string over the alphabet Σ_1.
- ✓ If $\Sigma_2 = \{a, b, c, \ldots, z\}$ then ***abracadabra*** is a string over Σ_2.
- ✓ If **w** is a string over Σ then the length of w is written as **|w|** which is the number of symbols that it contains.
- ✓ The string of length zero is called the **empty string.** It is written as **ε**.
- ✓ The ***empty string*** plays the role of 0 in a number system. If **w** has length **n** then we can write $w = w_1, w_2, \ldots, w_s$ where each $w_i \in \Sigma$.
- ✓ The reverse of **w** written as $\mathbf{w}^R$ is the string obtained by writing b in the opposite order (i.e. $w_n, w_{n-1}, \ldots, w_1$).
- ✓ String **z** is a **substring** of **w** if **z** appears consecutively within **w**. For example ***cad*** is a substring of ***abracadabra***.
- ✓ If we have string **x** of length **m** and string **y** of length **n** , the **concatenation** of **x** and **y** written as **xy** is the string obtained by appending y to the end of x as in $x_1 \ldots x_m y_1 \ldots y_n$. to concatenate the string with itself many times we use the notation below:

$$\overbrace{xx\ldots x}^{k} = x^k$$

- ✓ The **lexicographic ordering** of strings is the same as the familiar dictionary ordering except that shorter strings precede longer strings.
- ✓ For example lexicographic ordering of all strings over the alphabet {0, 1} is (ε, 0, 1, 00, 01, 10, 11, 000, 0001, …).
- ✓ **Powers of an alphabet:** If Σ is an alphabet, then
  - $\mathbf{\Sigma^0 = \varepsilon}$**,** no matter what the alphabet Σ is. In other words ε is the only string of length 0.

- If $\Sigma = \{a, b, c\}$ then $\Sigma^1 = \{a, b, c\}$, $\Sigma^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$, $\Sigma^3 = \{aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa, bab, bac, bba, bbb, bbc, bca, bcb, bcc, caa, cab, cac, cba, cbb, cbc, cca, ccb, ccc\}$
- The set of all strings over an alphabet Σ is conventionally denoted by **Σ*.** For instance $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. Another way is

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

- The set of nonempty strings of an alphabet Σ is denoted as Σ^+. And the two appropriate equivalences are :
  - $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$
  - $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

✓ A **language** is a set of strings.

✓ A set of strings all of which are chosen from Σ*, where Σ is a particular alphabet, is called language.

- **Examples:**
- The language of all strings consisting of n 0's followed by n 1's for some n≥0; { ε, 01, 0011, 000111, . . . }.
- The set of strings of 0's and 1's with an equal number of each :

  {ε, 01, 10, 0011, 0101, 1001, . . .}
- The set of binary values whose value is prime: {10, 11, 101, 111, 1011,. . .}
- Σ* is a language for any alphabet Σ.
- ϕ the empty language is a language over any alphabet.
- {ε} the language consisting of only the empty string, is also a language over any alphabet.

  Note: ϕ ≠ {ε}; the former has no strings and the later has one string.

- **Operations :**
- **Boolean logic** is a mathematical symbol built around the two values TRUE and FALSE called as the Boolean values and are often represented by values 1 and 0.
- We manipulate Boolean values with specially designed operations called **Boolean operations**. The simplest such operation is the **negation** or **NOT** operation, designated with the symbol ¬.
- The negation of a Boolean value is the opposite value. Thus ¬0 = 1 and ¬1 = 0.
- The **conjunction** or **AND** operation is designated with the symbol ∧. The conjunction of two Boolean values is 1 if both of those values are 1.
- The **disjunction** or **OR** operation is designated with the symbol v. The disjunction of two Boolean values is 1 if either of those values is 1. We summarize this a follows:

$0 \wedge 0 = 0$	$0 \vee 0 = 0$
$0 \wedge 1 = 0$	$0 \vee 1 = 1$
$1 \wedge 0 = 0$	$1 \vee 0 = 1$
$1 \wedge 1 = 1$	$1 \vee 1 = 1$

- We use Boolean operations for combining simple statements in to more complex Boolean expressions just as we use arithmetic operations + and x to construct complex arithmetic expressions.
- Several other Boolean operations appear occasionally like **exclusive or** or **XOR** operation, it is designated by the symbol $\oplus$ and is 1 if either but not both of its two operands are 1.
- The **equality** operation written with the symbol $\leftrightarrow$ is 1 if both of its operands have the same value.
- Finally **implication** operation is designated by the symbol $\rightarrow$ and is 0 if first operand is 1 and its second operand is 0 other wise $\rightarrow$ is 1.
- In addition to standard set operations we can also use operations on strings like concatenation to generate new languages. If x and y are strings over Σ then the concatenation of x and y denoted by xy is a new string formed by writing the symbols of x followed by the symbols of y. So if x = aa and y = bbb then xy = aabbb.
- If L_1 and L_2 are two languages then we can generate a new language L_1L_2 which is defined as follows:

$$L_1L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$$

FINITE STATE MACHINE

- A finite-state machine (FSM) or simply a state machine is a mathematical model of computation used to design both computer programs and sequential logic circuits.
- It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state.
- It can change from one state to another when initiated by a triggering event or condition, this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.
- Finite State machine is also termed as finite automaton.
- Finite automata are good models for computers with an extremely limited amount of memory. A computer can do a lot of useful things with such a small memory.
- The controller for an automatic door is one example of such a device which is often found at supermarket entrances and exits.
- Also automatic doors swing open when sensing that a person is approaching. It has a pad in front to detect the presence of a person about to walk through the doorway.
- Another pad is located to the rear of the doorway so that the controller can hold the door open long enough for the person to pass all the way through and also so that the door does not strike someone standing behind it as it opens.
- This configuring is shown as below:

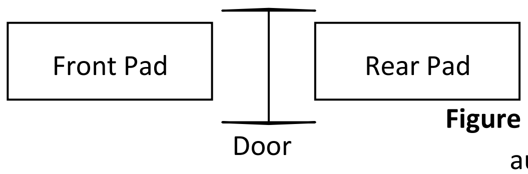

Figure 1.1: Top View of an automatic door

- The controller is in either of two states "OPEN" and "CLSOED", representing the corresponding condition of the door.
- As shown in the following figures, there are four possible input conditions "FRONT" (meaning that a person is standing on the pad in front of the doorway), "REAR" (meaning that a person is standing on the pad to the rear of the doorway), "BOTH" (meaning that a person is standing on both pads), and "NEITHER" (meaning that no one is standing on either pad).

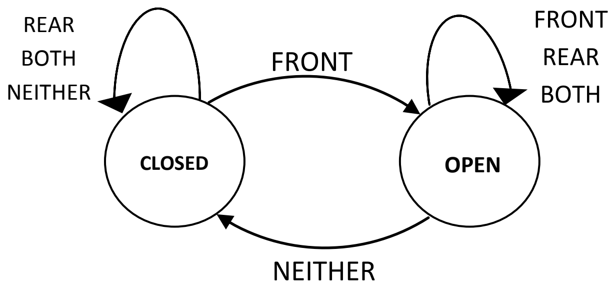

Figure 1.2: State Diagram for automatic door controller

Input Signal

state	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

Figure 1.3 State transition table for automatic door controller

- The controller moves from state to state, depending on the input it receives. When in the CLOSED state and receiving input NEITHER or REAR it remains in CLOSED state.
- In addition if the input BOTH is received it stays in CLOSED because opening the door risks knocking someone over on the rear pad.
- But if the input FRONT arrives, it moves to the OPEN state. In the OPEN state, if the input FRONT, REAR, or BOTH is received, it remains in OPEN. If the input NEITHER arrives, it returns to CLOSED.
- For example a controller might start in state CLOSED and receive the series of input signals FRONT, REAR, NEITHER, FRONT, BOTH, NEITHER, REAR, and NEITHER. It then would go through the series of states CLOSED (starting), OPEN, OPEN, CLOSED, OPEN, OPEN, CLOSED, CLOSED, and CLOSED.
- This controller is a computer that has just a single bit of memory, capable of recording which of the two states the controller is in.
- Other common devices have controllers with somewhat larger memories. In an elevator controller a state may represent the floor the elevator is on and the inputs might be the signals received the buttons.
- This computer might need several bits to keep track of the information. Controllers for various household appliances like dishwashers, electronic thermostats, as well as parts of digital watches and calculators are additional examples of computers with limited memories.

The design of this kind of devices requires keeping the methodology and terminology of finite automata in mind.

- Finite automata and their probabilistic counterpart Markov Chains are useful tools when we are attempting to recognize patterns in data. These devices are used in speech processing and in optical character recognition.

Definitions

- In the beginning to describe the mathematical theory of finite automata, we do so in the abstract, without reference to any particular application.
- The following figure depicts a finite automaton called M_1.

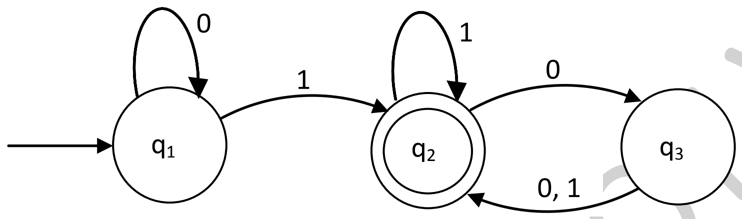

Figure 1.4: A finite automaton called M_1 that has three states

- Figure 1.4 is called as the **state diagram** of M_1. It has three states labeled q_1, q_2, and q_3.
    - The start state, q_1 is indicated by the arrow pointing at it from nowhere.
    - The accept state, q_2 is the one indicated with a double circle. The arrow going from one state to another are called **transitions**.
    - When this automaton receives the input string like 1101, it processes that string and produces an output. The output is either **accept** or **reject**.
    - The processing proceeds as follows:
        - Start in state q_1.
        - Read 1, follow transition from q_1 to q_2.
        - Read 1, follow transition from q_2 to q_2.
        - Read 0, follow transition from q_2 to q_3.
        - Read 1, follow transition from q_3 to q_2.
        - *Accept* because M_1 is in an accept state q_2 at the end of the input.
- Experimenting with this machine on a variety of input strings reveals that it accepts the strings 1, 01, 11 and 0101010101.
- In fact M_1 accepts any string that ends with a 1 as it goes to its accept state q_2 whenever it reads the symbol 1.
- In addition it accepts strings 100, 0100, 110000 and 0101000000 and any string that ends with an even number of 0's following the last 1.
- It rejects other strings such as 0, 10, and 101000.

Automaton is defined as a system where energy, materials and information are transformed, transmitted and used for performing some functions without direct participation of man.

FINITE AUTOMATON MODEL

- A finite automaton has several parts. It has the set of states and rules for going from one state to another depending on the input symbol.
- It has an input alphabet that indicates the allowed input symbols.
- It has the start state and a set of accept states.
- The formal definition says that a finite automaton is a list of those five objects: *set of states, input alphabet, rules for moving, start state,* and *accept states.*
- In mathematical language the list of five elements is often called as 5-tuple.
- We use something called ***transition function*** frequently denoted by δ to define the rules for moving.
- If the finite automaton has an arrow from state **x** to sate **y** labeled with the input symbol **1**, that means that, if the automaton is in state **x** when it reads a **1**, it moves to state **y**. we indicate this with transition function by saying that δ(x,1)=y.
- The block diagram of Finite automaton is depicted in the diagram below:

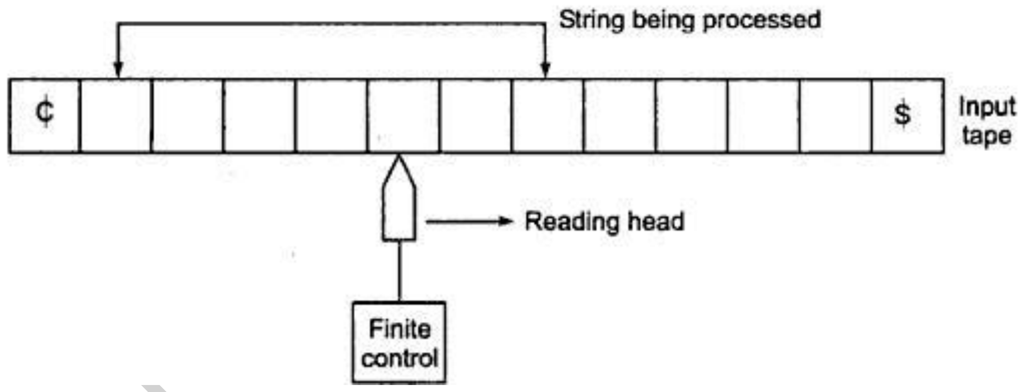

Figure A: Block Diagram of Finite Automaton

- There are various components in finite automaton
  1. **Input Tape:** The input tape is divided into squares, each square containing a single symbol from the input alphabet Σ. The end squares of the tape contain end markers ₵ at the left end and $ at the right end. Absence of the end markers indicates that the tape is of infinite length. The left to right sequence of symbols between the end markers is the input string to be processed.
  2. **Reading head:** The head examines only one square at a time and can move one square either to the left or to the right.

3. **Finite Control:** The input to the finite control will be usually: symbol under the R-head say a, or the present state of the machine say q to give the following outputs: (a) A motion of R-head along the tape to the next square. (b) The next state of the finite state machine given by δ(q, a).

FORMAL DEFINITION OF A FINITE AUTOMATON

- A ***finite automaton*** is defined as a 5 – tuple (Q, Σ, δ, q_0, F) where
  1. Q is the finite set called the ***states***,
  2. Σ is the finite set called the ***alphabet***,
  3. δ: Q x Σ → Q is the ***transition function***,
  4. q_0 Є Q is the ***start state***, and
  5. F ⊆ Q is the ***set of accept states***.
- By setting F to be the empty set ϕ yields 0 accepts states which is allowable.
- Consider the finite automaton M_1 in figure 1.4, where M_1 is formally written as M_1 = (Q, Σ, δ, q_1,F), where
  1. Q = {q_1, q_2, q_3}
  2. Σ = {0, 1}
  3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

  4. q_1 is the start state and
  5. F = {q_2}

ACCEPTANCE OF STRINGS AND LANGUAGES

ACCEPTABILITY OF STRING BY FINITE AUTOMATON

- Def: A string x is accepted by a finite automaton M = (Q, Σ, δ, q_0, F) if δ(q_0, x) =q for some q Є F. This is basically the acceptability of a string by the final state.
  Note: A final state is also called an accepting state.
- If A is the set of all strings that machine M accepts then we say that A is the language of machine M and written as L(M) = A. we say that M recognizes A or M accepts A.
- Because the term accepts has different meanings when we refer to machine accepting strings and machines accepting languages, we prefer the term recognize for languages to avoid confusion.
- A machine may accept several strings but it always recognizes only one language.
- If the machine accepts no strings, it still recognizes one language namely the empty language ϕ.

- In our example let A = {**w** | **w** contains at least one 1 and an even number of 0's follow the last 1}. Then $L(M_1)$ = A or equivalently M_1 recognizes A.
- Let $L(M_1)$ = {**w** | $\delta(q_0,$**w**$)$ is in F}. That is the language of M_1 is the set of strings **w** that take the start state q_0 to one of the accepting states.
- **Examples:**

1) Consider the state diagram of the finite automaton M_2.

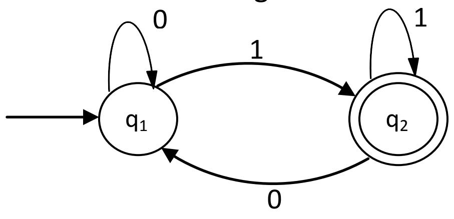

Figure 1.5: State diagram of the two state finite automaton M_2

In the formal description $M_2 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1 \{q_2\})$. The transition function δ is

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

2) Consider automaton M_3.

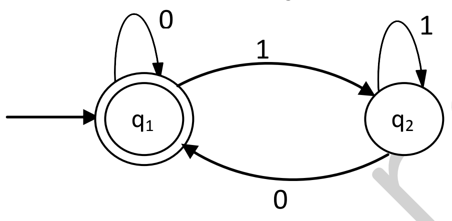

Figure 1.6: State diagram of the two state finite automaton M_3

- Machine M_3 is similar to M_2 except for the location of the accept state. As usual machine accepts all strings that leave it in an accept state when it has finished reading.
- Note that because the start state is also an accept state, M_3 accepts the empty string ε.
- In addition to the empty string this machine accepts any string ending with a 0.
- Here

  $L(M_3)$ = {w | w is the empty string ε or ends in a 0}.

DETERMINISTIC FINITE AUTOMATA

- By introducing the formalism of a deterministic finite automaton, one that is in a single state after reading any sequence of inputs.
- The term deterministic refers to the fact that on each input there is one and only one state to which the automaton can transition from its current state.

DEFINITION OF DETERMINISTIC FINITE AUTOMATON

- A deterministic finite automaton consists of
  - A finite set of states often denoted Q.

  - A finite set of input symbols often denoted Σ.
  - A transition function that takes as arguments a state and an input symbol and returns a state. The transition function will commonly be denoted by δ.
  - A start state, one of the states in Q.
  - A set of final or accepting states F. The set F is a subset of Q.
- A deterministic finite automaton will often be referred to by its acronym: DFA
- DFA in 5-tuple notation is given as below:

$$A = (Q, \Sigma, \delta, q_0, F)$$

Where A is the name of the DFA, Q is its set of states, Σ its input symbols, δ its transition function, q_0 its start state and F its set of accepting states.

How DFA processes strings?

- The first thing we need to understand about is
  - How the DFA decides whether or not to “accept” a sequence of input symbols.
  - The “language” of the DFA is the set of all strings that the DFA accepts.
- Let $a_1a_2.....a_n$ is a sequence of input symbols. We start out with the DFA in its start state q_0.
- We consult the transition function δ, say $\delta(q_0,a_1) = q_1$ to find the state that the DFA A enters after processing the first input symbol a_1.
- We process the next input symbol a_2, by evaluating $\delta(q_1,a_2)$; let us suppose this state is q_2.
- We continue in this manner finding the states $q_3, q_4, . . . , q_n$ such that $\delta(q_{i-1},a_i) = qi$ for each i.
- If q_n is a member of F then the input $a_1a_2. . . a_n$ is accepted and if not it is “rejected”.

Example: (B)

Let us formally specify DFA that accepts all and only the strings of 0’s and 1’s that have the sequence 01 somewhere in the string. We can write this language L as:

{w | w is of the form x01y for some strings x and y consisting of 0’s and 1’s only}

Another equivalent description using parameters x and y to the left of the vertical bar is:

{x01y | x and y are any strings of 0’s and 1’s}

- Examples of strings in the language include 01, 11010, and 100011.
- Examples of strings not in the language include ε, 0, and 111000.
- If the automaton accepts the language L then its input alphabet is Σ = {0, 1}
- To decide whether 01 is a substring of the input, A needs to remember:
  1. Has it already seen 01? If so it accepts every sequence of further inputs, i.e. it will only be in accepting states from now on.
  2. Has it never seen 01, but its most recent input was 0, so if it now sees a 1, it will have seen 01 and can accept everything it sees from here on?
  3. Has it never seen 01, but its last input was either nonexistent (it just started) or it last saw a 1? In this case, A cannot accept until it first sees a 0 and then sees a 1 immediately after.

- These three can each be represented by a state. Condition (3) is represented by start state q_0. Surely when just starting we need to see a 0 and then a 1.
- But if in state q_0 we next see a 1, then we are no closer to seeing 01 and so we must stay in state q_0. i.e. $\delta(q_0, 1) = q_0$.
- However if we are in state q_0 and we next see a 0, we are in condition (2) i.e. we have never seen 01 but we have our 0. Thus let us use q_2 to represent condition (2).
- Our transition from q_0 on input 0 is $\delta(q_0,0) = q_2$.
- If we are in state q_2 and we see a 1 input. We now know there is a 0 followed by a 1. We can go an accepting state which we shall call q_1 and which corresponds to condition (1) above. i.e. $\delta(q_2, 1) = q_1$.
- Finally we must design the transitions for state q_1. In this state we have already seen a 01 sequence i.e. $\delta(q_1, 0) = \delta(q_1, 1) = q_1$.
- Thus $Q = \{q_0, q_1, q_2\}$. As we said q_0 is the start state and the only accepting state is q_1. i.e. $F= \{q_1\}$.
- The complete specification of the automaton A that accepts the language L of strings that have a 01 substring is

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$$

- ***Simpler Notations for DFA***
- There are two preferred notations for describing automata
  1) A transition diagram which is a graph
  2) A transition table which is a tabular listing of the δ function

Transition Diagram:

- A transition diagram for DFA $A = (Q, \Sigma, \delta, q_0, F)$ is a graph defined as follows:
  a) For each state in Q there is a node
  b) For each state *q* in Q and each input symbol *a* in Σ, let $\delta(q, a) = p$. then the transition diagram has an arc from node q to node p labeled a. If there are several input symbols that cause transitions from q to p then the transition diagram can have one arc labeled by the list of these symbols
  c) There is an arrow into the start state q_0, labeled *start*. This arrow does not originate at any node.
  d) Nodes corresponding to accepting state (those in F) are marked by a double circle. States not in F have a single circle.

Example:

- The following figure shows the transition diagram for the DFA that we designed in three states. There is a *start* arrow entering the start state q_0, and the one accepting state q_1, is represented by double circle.

- Out of each state is one arc labeled 0 and one arc labeled 1, although the two arcs are combined into the one with a double label in the case q_1.

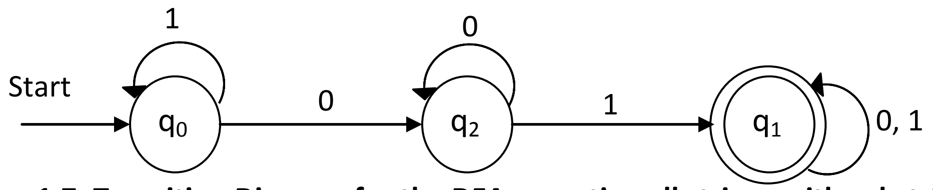

Figure 1.7: Transition Diagram for the DFA accepting all strings with substring 01

Transition Tables:

- A transition table is a conventional tabular representation of a function like δ that takes two arguments and returns a value.
- The rows of the table correspond to the states and columns correspond to the inputs. The entry for the row corresponding to state q and the column corresponding the input *a* is the state δ(q, a).

Example:

- The following figure shows the transition table, where the start state is marked with an arrow, and the accepting states are marked with a star.
- Since we can deduce the sets of states and input symbols by looking at row and column heads, we can now read from the transition table all the information we need to specify the finite automaton uniquely.

	0	1
→ q_0	q_2	q_0
$^*q_1$	q_1	q_1
q_2	q_2	q_1

Figure 1.8: Transition Table for the DFA of example B

NONDETERMINISTIC FINITE AUTOMATA

- A nondeterministic finite automaton (NFA) has the power to be in several states at once.
- This ability is often expressed as an ability to guess something about its input.

An informal view of Nondeterministic Finite Automata

- Like DFA, an NFA has a finite set of states, a finite set of input symbols, one start state and a set of accepting states.
- It also has transition function which we shall commonly call Δ. The difference between DFA and NFA is in the type of Δ.
- For NFA, Δ is a function that takes a state and input symbol as arguments but returns a set of zero, one or more states.

Example:

- The following figure shows a finite automaton which accepts exactly those strings that have the symbol 1 in the second last position.
- State q_0 is the initial state from where it moves when it sees a 1 and guesses that there is only one more symbol to follow. Since it is possible that there is more than one symbol to be examined, there are transitions from q_0 to itself on reading either 0 or 1.

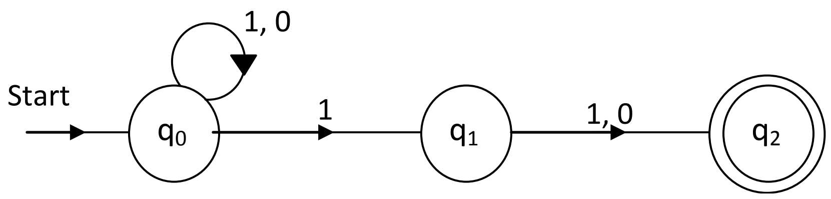

Figure 1.9: An NFA accepting the set of all strings whose second last symbol is 1

- Note that now there are two possible transitions labeled 1 out of q_0 and hence NFA has two options – it can move to either q_0 or q_1.

DEFINITION OF NONDETERMINISTIC FINITE AUTOMATA

- An NFA is represented essentially like a DFA:

$$A = (Q, \Sigma, \Delta, q_0, F)$$

Where:

1. Q is finite set of *states*.
2. Σ is finite set of *input symbols*.
3. q_0 a member of Q is the *start* state
4. F is a subset of Q is the set of the *final* (or *accepting*) states.
5. 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. The difference between an NFA and DFA is in the type of value that Δ returns a set of states in the case of an NFA and a single state in the case of DFA. ($\Delta: Q \times \Sigma \rightarrow P(Q)$)

Example

- The NFA of figure 1.9 can be specified formally as ($\{q_0, q_1, q_2\}, \{0, 1\}, \Delta, q_0, \{q_2\}$) where the transition function Δ is given by the table of figure 1.10.

	0	1
→q_0	q_0	q_0,q_1
q_1	q_2	q_2
*q_2	ϕ	ϕ

Figure 1.10: Transition Table for an NFA that accepts all strings ending in 01

Language Recognizer:

- A device that accepts valid strings is called the Language recognizer. For example finite automata are language recognizer.
- Recognizers are Machines. The Machines take a string as input. The Machines will accept the input if when run, the Machine stops at an accept state. Otherwise the input is rejected.
- If a Machine M recognizes all strings in Language L, and accepts input provided by a given string S, M is said to accept S. Otherwise M is said to reject S. S is in L if and only if M accepts S.
- Another example for language recognizer are PDA's (Push Down Automaton)