Automata ,Languages and Computation

Introduction

Automata theory : the study of abstract computing devices, or ”machines”.

Abstract Machine is a model of a computer system(either as hardware or software) constructed to allow a detailed and precise analysis of how the computer system works. Such a model usually consists of input, output and operations that can be performed .eg Turing machines.

Before computers (1930), A. Turing studied an abstract machine (Turing machine) that had all the capabilities of today’s computers (concerning what they could compute). His goal was to describe precisely the boundary between what a computing machine could do and what it could not do.

Abstract machines that model software are usually thought of as having very high level operations Example: An Abstract machine that models a banking system can have operations like “deposit”, “Withdraw”, “Transfer” etc.

Simpler kinds of machines (finite automata) were studied by a number of researchers and useful for a variety of purposes.

Theoretical developments bear directly on what computer scientists do today

- Finite automata, formal grammars: design/ construction of software
- Turing machines: help us understand what we can expect from a software
- Theory of intractable problems: are we likely to be able to write a program to solve a given problem? Or we should try an approximation, a heuristic...

Finite automata are a useful model for many important kinds of software and hardware:

1. Software for designing and checking the behaviour of digital circuits

2. The lexical analyser of a typical compiler, that is, the compiler component that breaks the input text into logical units

3. Software for scanning large bodies of text, such as collections of Web pages, to find occurrences of words, phrases or other patterns

4. Software for verifying systems of all types that have a finite number of distinct states, such as communications protocols of protocols for secure exchange information.

The Central Concepts of Automata Theory

Alphabet

- A finite, nonempty set of symbols.
- Symbol: Σ
- Examples:
  - ✓ The binary alphabet: Σ = {0, 1}
  - ✓ The set of all lower-case letters: Σ = {a, b, . . . , z}
  - ✓ The set of all ASCII characters

Strings

- A string (or sometimes a word) is a finite sequence of symbols chosen from some alphabet.
- Example: 01101 and 111 are strings from the binary alphabet Σ = {0, 1}
- Empty string: the string with zero occurrences of symbols This string is denoted by ε and may be chosen from any alphabet whatsoever.
- Length of a string: the number of positions for symbols in the string Example: 01101 has length 5
  • There are only two symbols (0 and 1) in the string 01101, but 5 positions for symbols.
- Notation of length of w: |w| Example: |011| = 3 and |ε| = 0

Powers of an alphabet (1)

If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using the exponential notation:

- Σ^k: the set of strings of length k, each of whose is in Σ
- Examples:
  - Σ^0 : { ε }, regardless of what alphabet Σ is. That is ε is the only string of length 0
  - If Σ = { 0, 1 }, then:
    1. Σ^1 = { 0, 1 }
    2. Σ^2 = {00, 01, 10, 11 }
    3. Σ^3 = {000, 001, 010, 011, 100, 101, 110, 111 }

Note: confusion between Σ and Σ^1 :

1. Σ is an alphabet; its members 0 and 1 are symbols
2. Σ^1 is a set of strings; its members are strings (each one of length 1)

Kleen star

- $\Sigma*$: The set of all strings over an alphabet Σ
  - $\{0, 1\}* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$
  - $\Sigma* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- The symbol $*$ is called Kleene star and is named after the mathematician and logician Stephen Cole Kleene.
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$ Thus: $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$
  (Kleen Closure)

Concatenation

- Define the binary operation . called concatenation on $\Sigma*$ as follows: If a1a2a3 . . . an and b1b2 . . . bm are in $\Sigma*$, then a1a2a3 . . . an.b1b2 . . . bm = a1a2a3 . . . anb1b2 . . . bm
- Thus, strings can be concatenated yielding another string: If x are y be strings then x.y denotes the concatenation of x and y, that is, the string formed by making a copy of x and following it by a copy of y
- Examples:
  1. x = 01101 and y = 110 Then xy = 01101110 and yx = 11001101
  2. For any string w, the equations w = wε = w hold. That is, ε is the identity for concatenation (when concatenated with any string it yields the other string as a result)
- If S and T are subsets of $\Sigma*$, then $S.T = \{s.t \mid s \in S, t \in T\}$

Languages

- If Σ is an alphabet, and $L \subseteq \Sigma*$, then L is a (formal) language over Σ.
- Language: A (possibly infinite) set of strings all of which are chosen from some $\Sigma*$
- A language over Σ need not include strings with all symbols of Σ Thus, a language over Σ is also a language over any alphabet that is a superset of Σ
- Examples:
  - Programming language C
    Legal programs are a subset of the possible strings that can be formed from the alphabet of the language (a subset of ASCII characters).
  - English or French

Other language examples

1. The language of all strings consisting of n 0s followed by n 1s ($n \geq 0$): $\{\varepsilon, 01, 0011, 000111, \ldots\}$

2. The set of strings of 0s and 1s with an equal number of each: $\{\varepsilon, 01, 10, 0011, 0101, 1001, \ldots\}$

3. Σ^* is a language for any alphabet Σ

4. Ø, the empty language, is a language over any alphabet

5. { ε}, the language consisting of only the empty string, is also a language over any alphabet

NOTE: Ø ≠ { ε } since Ø has no strings and {ε} has one

6. { w | w consists of an equal number of 0 and 1 }

7. { $0^n 1^n$ | n ≥ 1 }

8. { $0^i 1^j$ | 0 ≤ i ≤ j }

Automata Theory is a branch of computer science that deals with designing abstract self propelled computing devices that follow a predetermined sequence of operations automatically. An automaton with a finite number of states is called a **Finite Automaton**(FA) **or Finite State Machine** (FSM).

The term "Automata" is derived from the Greek word "αὐτόματα" which means "self-acting".

Formal definition of a Finite Automaton

An automaton can be represented by a 5-tuple (Q, ∑, δ, q_0, F), where −

- **Q** is a finite set of states.
- **∑** is a finite set of symbols, called the **alphabet** of the automaton.
- **δ** is the transition function.
- **q_0** is the initial state from where any input is processed (q_0 ∈ Q).
- **F** is a set of final state/states of Q (F ⊆ Q).

Finite Automaton can be classified into two types −

- Deterministic Finite Automaton (DFA)
- Non-deterministic Finite Automaton (NDFA / NFA)

Deterministic Finite Automaton (DFA)

In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**. As it has a finite number of states, the machine is called **Deterministic Finite Machine** or **Deterministic Finite Automaton.**

Formal Definition of a DFA

A DFA can be represented by a 5-tuple (Q, ∑, δ, q_0, F) where −

- **Q** is a finite set of states.
- **∑** is a finite set of symbols called the alphabet.
- **δ** is the transition function where δ: Q × ∑ → Q. The transition function takes as arguments a state and an input symbol and returns a state.
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$). It is one of the states in Q.
- **F** is a set of final or accepting state/states of Q ($F \subseteq Q$).

Graphical Representation of a DFA(Transition Diagrams)

A transition diagram for a DFA A=(Q, ∑, δ, q_0, F) is a graph defined as follows:

a) For each state in Q there is a node.(The vertices of graph represent the states.)
b) For each state q in Q and each input symbol a in ∑.Let δ(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 states (those in F) are marked by double circle. States not in F have single circle.

Transition table

It is the tabular representation of a function δ , that takes two arguments and returns a value. The rows of the table correspond to the states , and the columns correspond to the inputs. The entry for the row corresponding to state q and the column corresponding to input a is the state δ(q,a). The start state is marked with arrow ,and the accepting states are marked with a star.

Example(1) :

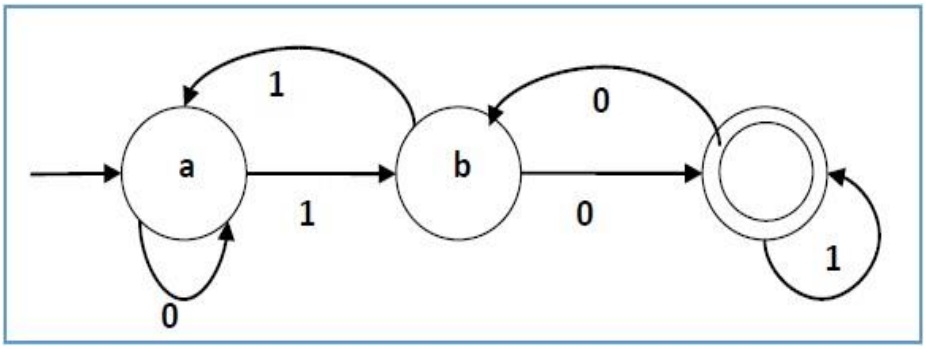
Let a deterministic finite automaton be →

- Q = {a, b, c},
- ∑ = {0, 1},
- q_0 = {a},
- F = {c}, and

Transition function δ as shown by the following table −

Present State	Next State for Input 0	Next State for Input 1
a	a	b
b	c	a
c	b	c

Its graphical representation would be as follows:



Example 2:

Construct a DFA that accepts all and only the strings of 0's and 1's that have the sequence 01 somewhere in the string.

Solution:

Language L is written as

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

Or

L= { x01y| x and y are any strings of 0's and 1's }

The strings in this language are 01,11010,100011 ….

The strings not in this language are ε,0,111000….

The DFA A = (Q, ∑, δ, q_0, F)

Where Q= {q_0,q_1,q_2}

∑ = {0,1}

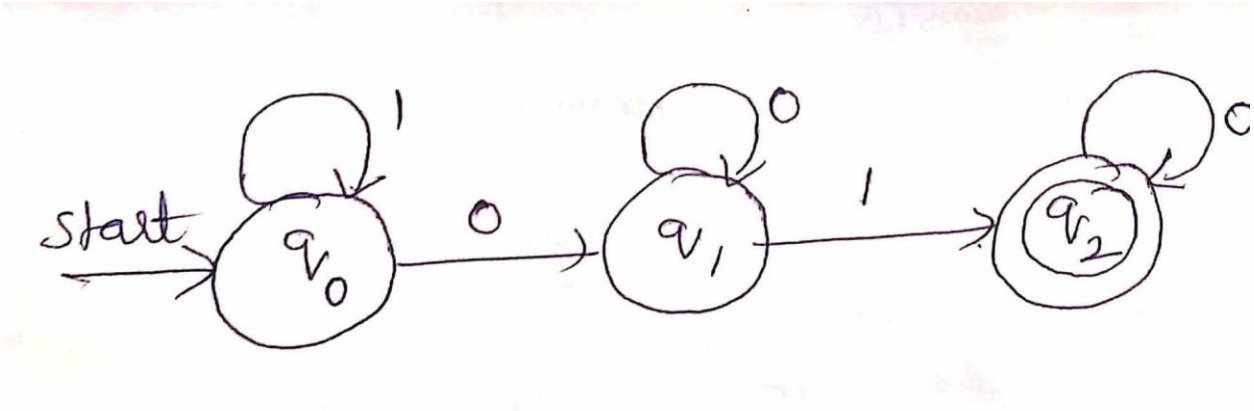
q_0 - Initial State

F = {q_2}

The transition function δ is defined as

	0	1
→q_0	q1	q0
q1	q1	q2
q2	q2	q2

The transition diagram is



Extending the Transition Function to Strings

- The DFA define a language: the set of all strings that result in a sequence of state transitions from the start state to an accepting state.
- Extended transition function
  - Describes what happens when we start in any state and follow any sequence of inputs.
  - If δ is our transition function, then the extended transition function is denoted by ˆδ
  - The extended transition function is a function that takes a state q and a string w and returns a state p (the state that the automaton reaches when starting in state q and processing the sequence of inputs w).

Formal definition of the extended transition function

Definition by induction on the length of the input string

Basis: δˆ(q, Ǫ) = q 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 string consisting of all but the last symbol

Then: ˆδ(q, w) = δ(ˆδ(q, x), a)

- To compute δˆ(q, w), first compute δˆ(q, x), the state that the automaton is in after processing all but the last symbol of w
- Suppose this state is p, i.e., δˆ(q, x) = p
- Then ˆδ(q, w) is what we get by making a transition from state p on input a - the last symbol of w.

Example: Design a DFA to accept the language

L = {w | w is of even length and begins with 01}

Solution: The automaton needs to remember whether the string seen so far started with 01. It also needs to keep track of the length of the string. Hence it contains five states and they are:

q_0: The initial state.
q_1: The state entered on reading 0 in state q_0.
q_2: The state entered on reading 01 initially. The automation subsequently returns to this state whenever the substring seen so far starts with 01 and is of even length.
q_3: The DFA enters this state whenever the substring seen so far starts with 01 and is of odd length.
q_4: This state is encountered whenever a 1 is encountered in state q_0 or a 0 is encountered in state q_1.

q_2 is the only accepting state. The DFA can be given as

M= ({q_0, q_1, q_2, q_3, q_4},{0,1}, δ , q_0 , {q_2}) .where δ the transition function is given by the transition diagram (see above figure).

Representation of this DFA in the form of transition diagram.

δ	0	1
→q_0	q_1	q_4
q_1	q_4	q_2
*q_2	q_3	q_3
q_3	q_2	q_2
q_4	q_4	q_4

Check whether the string 011101 is accepted by DFA or not.

Since this string starts with 01 and is of even length it is in the language.

Thus, we expect that δ^(q_0 , 011101) = q_2 . Since q_2 is the only accepting state.

The check involves computing δ^(q_0,w) for each prefix w of 011101,starting at ε and going in increasing size.

δ^(q_0,ε) = q_0.

δ^(q_0,0)=δ(δ^(q_0,ε),0)=δ(q_0,0)=q_1.

δ^(q_0,01)=δ(δ^(q_0,0),1)=δ(q_1,1)=q_2.

δ^(q_0,011)=δ(δ^(q_0,01),1)=δ(q_2,1)=q_3.

δ^(q_0,0111)=δ(δ^(q_0,011),1)=δ(q_3,1)=q_2.

δ^(q_0,01110)=δ(δ^(q_0,0111),0)=δ(q_2,0)=q_3.

δ^(q_0,011101)=δ(δ^(q_0,01110),1)=δ(q_3,1)=q_2.

The Language of a DFA

The language of a DFA A = (Q,∑,δ,q_0,F). This language is denoted by L(A), and is defined by

L(A) = { w| δ^(q_0 ,w) is in F}

That is, the language of A is the set of strings w that take the start state q_0 to one of the accepting states. If L is L(A) for some DFA A, then L is a regular language.

Exercise - I

Give DFA's accepting the following strings over the alphabet ∑ = {0,1}

1) The set of all the strings beginning with 101.
2) The set of all the strings containing 1101 as a substring.

3) The set of all the strings with exactly three consecutive 0's.
4) The set of all strings such that the number os 1's is even and the number of 0's is multiple of 3.
5) The set of all the strings not containing 110.
6) The set of all the strings that begin with 01 and end with 11.
7) The set of all strings which when interpreted as a binary integer is a multiple of 3.
8) The set of all the strings beginning with a 1 that , when interpreted as a binary integer , is a multiple of 5. Eg., strings 101,1010 and 1111 are in the language; 0,100, and 111 are not.

Give DFA's accepting the following strings over the alphabet ∑ = {a,b}

9) Construct a DFA for a string of length exactly 2.
10) Construct a DFA , for string length >= 2 .
11) Construct DFA for string length at most 2 i.e., |w|<=2.
12) Construct a minimal DFA which accepts all the strings where |w| % 2=0 .
13) Construct a DFA , for all strings |w| mod 3 = 0 .
14) Construct DFA for all strings if |w| ≈ 1 mod 3 o.
15) Construct a minimal DFA where n_a (w) = 2 .
16) Construct a minimal DFA where n_a (w) mod 2 = 0.

Nondeterministic Finite Automata (NFA) The FA which allows 0 or 1 or more states upon receiving the input symbol from ∑ is called NFA.

- A 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.
- Each NFA accepts a language that is also accepted by some DFA .
- NFA are often more succinct and easier than DFAs.
- We can always convert an NFA to a DFA, but the latter may have exponentially more states than the NFA (a rare case).
- The difference between the DFA and the NFA is the type of transition function δ .
  - For a NFA δ is a function that takes a state and input symbol as arguments (like the DFA transition function), but returns a set of zero or more states (rather than returning exactly one state, as the DFA must)

Example: An NFA accepting strings that end in 01

A = ({ q 0, q 1, q 2 }, { 0, 1 }, δ, q 0, { q 2 }) where the transition function δ is given by the table

	0	1
→ q_0	{ q_0,q_1 }	{ q_0}
q_1	Ø	{ q_2 }
⋆ q_2	Ø	Ø

Fig :

NFA: Formal definition

A nondeterministic finite automaton (NFA) is a tuple A = (Q, Σ, δ, q 0, F) where:

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

The only difference between a NFA and a DFA is in the type of value that δ returns

The Extended Transition Functions

Basis: $\delta\hat{}(q, \varepsilon) = \{ q \}$
Without reading any input symbols, we are only in the state we began in.

Induction:
Suppose w is a string of the form xa; that is a is the last symbol of w, and x is the string consisting of all but the last symbol.
Also suppose that $\delta\hat{}(q, x) = \{ p1, p2, \ldots pk, \}$. Let

$$\bigcup_{i=1}^{k} \delta(pi, a) = \{ r1, r2, \ldots, rm\}$$

Then: $\hat{}\delta(q, w) = \{ r1, r2, \ldots, rm \}$ We compute $\delta\hat{}(q, w)$ by first computing $\delta\hat{}(q, x)$ and by then following any transition from any of these states that is labeled a.

Example: An NFA accepting strings that end in 01.

Processing w = 00101

1. $\delta\hat{}(q_0, \varepsilon) = \{ q_0 \}$
2. $\delta\hat{}(q_0, 0) = \delta(q_0, 0) = \{ q_0, q_1 \}$
3. $\delta\hat{}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{ q_0, q_1\} \cup \emptyset = \{ q_0, q_1 \}$
4. $\delta\hat{}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{ q_0\} \cup \{ q_2 \} = \{ q_0, q_2 \}$
5. $\delta\hat{}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{ q_0, q_1\} \cup \emptyset = \{ q_0, q_1 \}$
6. $\delta\hat{}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{ q_0\} \cup \{ q_2 \} = \{ q_0, q_2 \}$

The Language of a NFA

The language of a NFA A = (Q, Σ, δ, q 0, F), denoted L(A) is defined by

$$L(A) = \{ w \mid \delta\hat{}(q0, w) \cap F \neq \emptyset\}$$

The language of A is the set of strings $w \in \Sigma *$ such that $\delta\hat{}(q_0, w)$ contains at least one accepting state.
The fact that choosing using the input symbols of w lead to a non-accepting state, or do not lead to any state at all, does not prevent w from being accepted by a NFA as a whole.

Note:

1. The capabilities of NFA's and DFA's are same. i.e., both NFA and DFA accepts the same set of strings.

   L(NFA) = L(DFA)
2. NFA is more powerful tool than DFA.
3. The construction of NFA is easier than DFA and understanding the business logic in NFA is also easy.
4. No, concept of dead state in NFA. i.e., in NFA we take care of only valid constructions.
5. NFA is like a parallel computing where we can run multiple threads concurrently.
6. Every DFA is an NFA.
7. Every NFA can be converted into DFA.

Exercise -2:

1. Design a NFA , which accepts exactly those strings that have a symbol 1 in the second last position.
2. Construct the NFA that accepts all the strings of 0's and 1's where every string starts with 0.
3. Construct the NFA that accepts all the strings of 0's and 1's where every string starts with 10.
4. Construct the NFA that accepts all the strings of a's and b's where every string ends with aa.
5. Construct the NFA that accepts all the strings of a's and b's where every string contains substring ab.
6. Construct the NFA that accepts all the strings of a's and b's where each string contains exactly 2 a's.
7. Construct the NFA that accepts all the strings of a's and b's where each string starts with a and ends with b.
8. Construct the NFA that accepts all the strings of a's and b's where each string starts and end's with different symbol.
9. Construct the NFA that accepts all the strings of a's and b's where every string starts and ends with same symbol.
10. Construct NFA where the third symbol from LHS is always b.
11. Construct the NFA , where the third symbol from RHS is always b.
12. Construct the NFA that accepts all the strings of a's and b's where the length of the string is exactly 3.
13. Construct the NFA where the length of the string is at least 3.
14. Construct NFA that accepts all the strings of a's and b's where the length of the string is atmost 3.
15. Construct NFA where the length of the string is divisible by 3.
16. Construct NFA where no.of a's in a string is congruent to 1(mod 3).
17. Construct NFA where the no.of b's is odd.

NOTE:

Sl.no	languages	Number of States	
		DFA	NFA
1	Starts and ends with same symbol	5	5
2	Starts and ends with different symbol	5	4
3	n^{th} symbol from LHS	n+2	n+1
4	n^{th} symbol from RHS	2^n	n+1
5	|w| = n	n+2	n+1
6	|w|<=n	n+2	n+1
7	|w|>=n	n+1	n+1
8	|w|≈m(mod n)	n	n
9	no.of a's is divisible by 5	n	n
10	no.of b's is ≈ m(mod n)	n	n

Equivalence 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 state).

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 stages of NFA.

From NFA to DFA

- We have a NFA $N = (Q_N , \Sigma, \delta_N , q_0, F_N)$
- The goal is the construction of a DFA $D = (Q_D, \Sigma, \delta_D, \{ q_0 \}, F_D)$ such that L(D) = L(N).

Subset Construction

1. Input alphabets are the same.
2. The start set in D is the set containing only the start state of N.
3. 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.
4. F_D is the set of subsets S of Q_N such that $S \cap F_N \neq \emptyset$. That is, F_D is all sets of N's states that include at least one accepting state of N.

5. 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 N goes from p on input a, and take the union of all those states.