1. Introduction

Q.4.5.1.1 What is automata theory?
Answer: *Automata theory* is a theoretical branch of computer science. The word automata is plural form of automaton. The word "automaton" is closely related to the word "automation", denotes automatic processes carrying out the production of an item. Automata theory deals with the logic of computation with respect to simple machines, referred to as automata. In automata theory, computer scientists are able to understand how machines compute functions and solve problems. Here we answer the questions like
i) what it means for a function to be defined as computable?
ii) what it means for a question to be described as decidable?
Automatons are abstract models of machines that perform computations on an input by moving through a series of states or configurations. At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration. As a result, once the computation reaches to an accepting configuration, it accepts that input. The most general and powerful automata is the Turing machine.
The major objective of automata theory is to develop methods by which computer scientists can describe and analyze the dynamic behavior of discrete systems, in which signals are sampled periodically. The behavior of these discrete systems is determined by the way that the system is constructed from storage and combinational elements. Characteristics of an automata include the following.
Inputs: Sequences of symbols selected from a finite set
$I = \{x_1, x_2, x_3, \ldots, x_m\}$, where m is the number of inputs.
Outputs: Sequences of symbols selected from a finite set
$O = \{y_1, y_2, y_3, \ldots, y_n\}$, where n is the number of outputs.
States: Finite set Q. The definition of Q depends on the type of automaton.
There are four major types of automaton : finite-state machine, push-down automata, linear-bounded automata and Turing machine.
These families of automata are shown in Fig. 1.1 as a hierarchal form, where the finite-state machine is the simplest automata and the Turing machine is the most complex one.

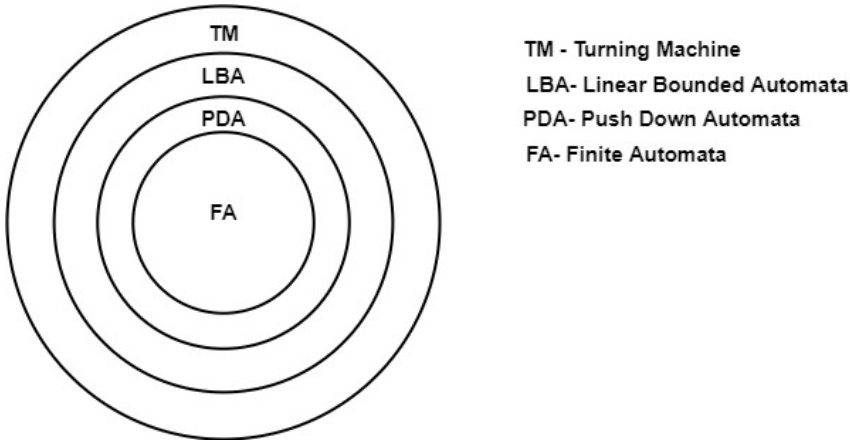


Figure 1.1: Chomsky hierarchy of automata

Q.4.5.1.2 What is computability theory?
Answer: It deals with the computer algorithm of a problem. There are many problems which do not possess any algorithm to solve them. For example, the problem of determining the truth value of mathematical statement. So, the objective of the computability theory is to classify problems by those that are solvable and those that are not.

Q.4.5.1.3 What are the objectives of complexity theory? Mention an area, where complexity theory plays an important role.
Answer: The objectives of complexity theory is to classify the problems as easy ones and hard ones. But the central idea of complexity theory is what makes a problem computationally hard and the other easy.
Generally, we prefer an easy solution of a hard problem. Cryptography is exceptional to that. Here we prefer a hard computational problem than an easy one. We try to protect valuable information by providing secret codes that are hard to break.

Q.4.5.1.4 What is multi-set? How does it differ from set?
Answer: A multi-set is a set that allows more than one occurrence of an element. For example, {1, 1, 1} and {1, 1} and {1} are different multi-sets, but they are identical sets.

Q.4.5.1.5 What is the difference between a sequence and a tuple?
Answer: A sequence is an ordered list of objects. For example, {4, 8, 12}

is a sequence. A finite sequence is a tuple. A sequence with m elements is an m-tuple.

Q.4.5.1.6 Find the error in the following proof.
Prove that $2 = 1$ (?)
Answer: Consider the equation $a = b$. (1.1)
Multiplying both-sides of (1.1) by a, we get
$a^2 = ab$
or, $a^2 - b^2 = ab - b^2$
or, $(a + b)(a - b) = b(a - b)$ (1.2)
or, $a + b = b$ (1.3)
Let $a = 1$ and $b = 1$.
From (1.3) we get, $2 = 1$.
Answer: The mistake in the above proof is that we cannot divide both side of (1.2) by $a - b$, since a and b are the same as assumed in (1.1).

Q.4.5.1.7 Prove that $(xy)^R = y^R x^R$ for any string over alphabet Σ, where z^R is the reverse of string z.
Answer: We shall prove it by induction on length of string y (i.e., $|y|$)
For $|y| = 0$, $y = \triangle$ (empty string).
In this case $(xy)^R = (x\triangle)^R = x^R = \triangle x^R = \triangle^R x^R = y^R x^R$
So, the result is true for $|y| = 0$. Assume that the result is true for $|y| \leq n$. So, $(xy)^R = y^R x^R$ for $|y| \leq n$.
Now, consider string y such that $|y| = n + 1$. Without loss of generality $y = wa$, for some $w \in \Sigma^*$ and $a \in \Sigma$, where $|w| = n$.
Now, $(xy)^R = (x(wa))^R = ((xw)a)^R = a(xw)^R$ [by definition of reversal]
$= aw^R x^R$ [by induction hypothesis as $|w| \leq n$]
$= (wa)^R x^R$ [by definition of reversal]
$= y^R x^R$

Q.4.5.1.8 Prove that, if $|A| > |B|$ then there is no one-to-one function from A to B.
Answer: We shall prove it by induction on the number of elements in B.
Let $|B| = 1$ and $b \in B$. Then $|A| > 1$ (given condition). Then, there exists at least two elements, $a_1, a_2 \in A$ and $f(a_1) = b, f(a_2) = b$. So, $f : x \to y$ is not one-to-one.
Let $|B| \leq n$ and $|A| > |B|$. Assume that $f : x \to y$ is not one-to-one.
Now, let $|B| = n + 1$ and $|A| > |B|$ (1.4)
We have to show that $f : x \to y$ is not one-to-one.
Let b be an arbitrary element in B. It is associated with more than one

2. Finite Automata

Q.4.5.2.1 State some features of finite automata.
Answer: Some important features of a finite automaton (FA) are given below.
(i) FA have no auxiliary memory.
(ii) They do not deliver any output, except an indication of whether the given input string is acceptable.
(iii) FA have a finite number of states.
(iv) They have a start state, but the number of final (acceptable) states could be more than one.

Q.4.5.2.2 Construct a deterministic finite automaton for language L over $\Sigma = \{0\}$ such that it contains all string of size divisible by 6.
Answer: Here, $L = \{\Lambda, 000000, 000000000000, \dots\}$
Deterministic finite automaton (DFA) is shown in Fig. 2.1.

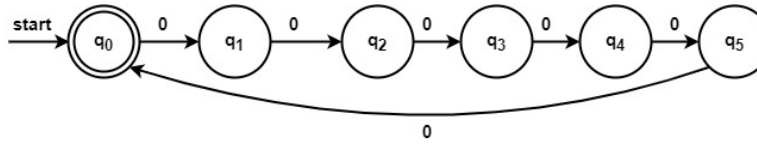


Figure 2.1: A FA that accepts string of length divisible by 6

In this case, the null string (Λ) is also acceptable by the automaton. Let s be an arbitrary string input to DFA. Then q_i represents the last state of DFA, when $|s|\ mod\ 6 = i$, where $|s|$ represents the length of string s. Obviously, q_0 becomes the only accepting state.

Q.4.5.2.3 Identify the sources of non-determinism in a non-deterministic finite automaton (NDFA).
Answer: There are two sources of non-determinism.
(i) For given input and current state, we permit NDFA to move one of the several states.
(ii) We also allow NDFA to move state without reading any input. This is also called the null transition or Λ-transition.

Q.4.5.2.4 Prove the following:
Let L_1, L_2 be two languages that are accepted by NDFAs M_1 and M_2 respectively. Then there exists NDFAs accepting the following languages.
(i) $L_1 \cup L_2$, (ii) $L_1 L_2$, (iii) L_1^*, (iv) $\Sigma^* - L_1$, (v) $L_1 \cap L_2$.
Answer: (i) We need to design a NDFA for $L_1 \cup L_2$, given that there there exist NDFAs for the languages L_1 and L_2.
Let $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$.

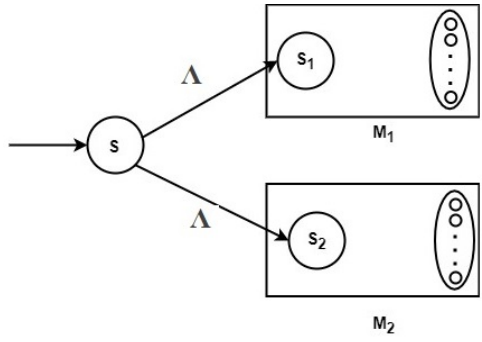


Figure 2.2: NDFA that accepts language $L_1 \cup L_2$

Let $M = (Q, \Sigma, \delta, s, F)$ be the NDFA that accepts language $L_1 \cup L_2$, where $Q = Q_1 \cup Q_2 \cup \{s\}, F = F_1 \cup F_2$, s is the start state of M and δ is defined as given in Fig. 2.2. Here, δ includes the transitions in δ_1, δ_2 and two more Λ transitions (null transitions).
(ii) We are given NDFAs for the languages L_1 and L_2. We need to find a NDFA for $L_1 L_2$, if it exists.
Let $M_1 = (Q, \Sigma, \delta, s, F)$ be the NDFA that accepts language $L_1 L_2$, where $Q = Q_1 \cup Q_2, F = F_2, s = s_1$ and δ as defined in Fig. 2.3. Transitions include δ_1, δ_2 and some other Λ transitions from the final states of M_1 to s_2, the start state of M_2.

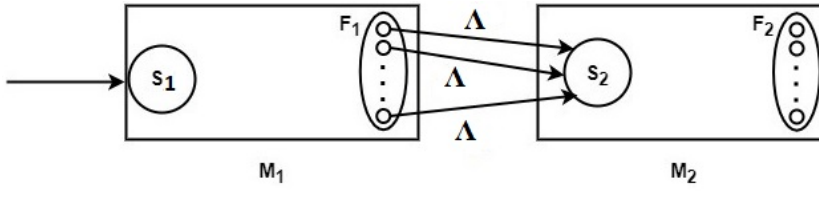


Figure 2.3: NDFA accepting the language $L_1 L_2$

(iii) There exists an NDFA for language L_1. Here we construct a NDFA for L_1^*.
New state s has been added and it is one of the final states. This ensure that the null string (Λ) also gets accepted by the NDFA $M = (Q, \Sigma, \delta, s, F)$, where $Q = \{s\} \cup Q_1, F = \{s\} \cup F_1$ with s as the new start state, and δ as defined by incorporating a set of Λ transitions from the final states of M_1 to s_1, so that a string can be repeated arbitrarily before getting it accepted (See Fig. 2.4).

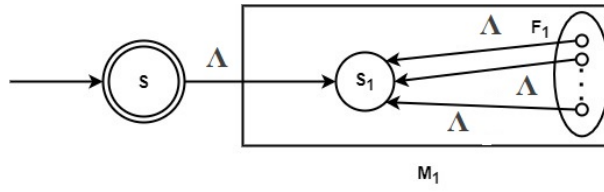
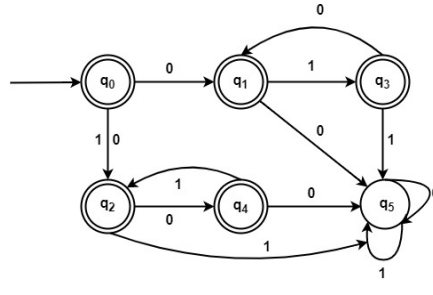


Figure 2.4: NDFA accepting language L_1^*

(iv) We give an idea of constructing a NDFA for $\Sigma^* - L_1 = \overline{L}_1$, given that M_1 is the NDFA for L_1. Let M be the NDFA for $\overline{L}_1$. Machine M is designed as follows:
$M = (Q, \Sigma, \delta, s, F)$, where $F = Q - F_1$, so that the set of rejected strings by M_1 becomes the set of accepting strings by M.
(v) We shall construct a NDFA for $L_1 \cap L_2$ with the help of previous results. By De Morgan's law, $L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}$. Applying results (i) and (iv), we can conclude that there exists a NDFA that accepts $L_1 \cap L_2$.

Q.4.5.2.5 Design a DFAs that accept
(i) $L_1 = \{w \in \{0,1\}^*$: neither 00 nor 11 is a substring of $w\}$
(ii) $L_2 = \{w \in \{0,1\}^*$: either 101 or 1001 is a substring of $w\}$
Answer: (i) The state diagram for the DFA accepting L_1



3. Regular Expressions

Q.4.5.3.1 Give definition of regular expression.
Answer: Let Σ be an alphabet for a given language, $|\Sigma| < \infty$. The following constants are regular expressions.
(i) Empty set $\emptyset$, denoting the set $\{\}$, is a regular expression.
(ii) Null string Λ, denoting the set $\{\Lambda\}$, is a regular expression.
(iii) $a \in \Sigma$ is a regular expression. It denotes the set $\{a\}$.
Consider that r and s are regular expressions over Σ. Then $(r + s)$, (rs) and (r^*) are regular expressions over Σ.
NOTE: An *empty* string is a string instance of zero length, whereas a *null* string has no value at all. Empty string is a string, but null string is not a string.

Q.4.5.3.2 $L = \{s \in \{0,1\}^* : s$ has an unequal number of 0s and 1s. Show that $L^* = \{0,1\}^*$.
Answer: $\{0,1\}^*$ is the collection of all strings of 0s and 1s.
So, $L \subseteq \{0,1\}^*$ (3.1)
Again, if $L_1 \subseteq L_2$ then $L_1^* \subseteq L_2^*$.
Now, $\{0,1\} \subseteq L$, since both strings 0 and 1 contain an unequal number of 0s and 1s.
Then, $\{0,1\}^* \subseteq L^*$ (3.2)
From (3.1) and (3.2), $L^* = \{0,1\}^*$.

Q.4.5.3.2 Prove that $\{a,b\}^* = a^*(ba^*)^*$.
Answer: $\{a,b\}^*$ represents set of all possible strings of a and b. For an arbitrary string, the symbol b will occur arbitrary number of times in that string, and symbol a occurs arbitrary number of times in between two bs.

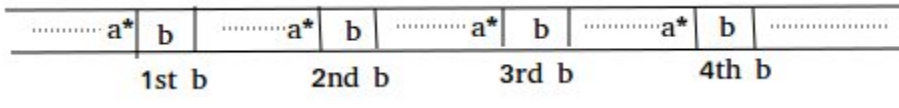


Figure 3.1: A string containing of a and b, where two successive bs contain an arbitrary number of as

Note that ba^* is repeating arbitrary number of times. See Fig 3.1. Thus, any arbitrary string of a and b can be constucted by the regular expres-

sion $a^*(ba^*)^*$.

Q.4.5.3.3 Write the regular expression of the set of all strings on $\{a, b\}$ containing exactly one occurrence of the substring aaa.
Answer: Regular expression containing zero occurrence of a is b^*. Regular expression containing one occurrence of a is b^*ab^*. Regular expression containing two occurrences of a is $b^*ab^*ab^*$. Regular expression containing zero occurrence of aaa is $b^* \bigcup b^*ab^* \bigcup b^*ab^*ab^*$.
Therefore, regular expression containing exactly one occurrence of the substring aaa is $(b^* \bigcup b^*ab^* \bigcup b^*ab^*ab^*)\, aaa\, (b^* \bigcup b^*ab^* \bigcup b^*ab^*ab^*)$.

Q.4.5.3.4 Find the regular expressions corresponding to the languages.
(i) $\{a^n b^m \mid n < 4, m \leq 3\}$
(ii) $\{w \in \{a, b\}^* : \ |w| \bmod \ 3 = 0\}$
(iii) $\{a^n b^m \mid n \geq 4, m \leq 3\}$
Answer: (i) $(\Lambda + a + aa + aaa)(\Lambda + b + bb + bbb)$, where, Λ stands for null string
(ii) $((a + b)(a + b)(a + b))^*$
(iii) $aaaaa^*(\Lambda + b + bb + bbb)$

Q.4.5.3.5 Describe the following sets using regular expressions:
(a) $\{abb, a, b, bba\}$
(b) {1, 11, 111, ...}
Answer: (a) The set $\{abb, a, b, bba\}$ is represented by $abb + a + b + bba$.
(b) {1, 11, 111, ...} is represented by $1(1)^*$.

Q.4.5.3.6 Draw the transition systems that recognise the following regular expressions.
(i) Λ (null string), (ii) $\emptyset$ (empty string), (iii) a ($\in \Sigma$)
Answer: Fig. 3.2(i), Fig. 3.2(ii) and Fig. 3.2(iii) show transition systems for regular expressions (i), (ii) and and (iii) respectively.

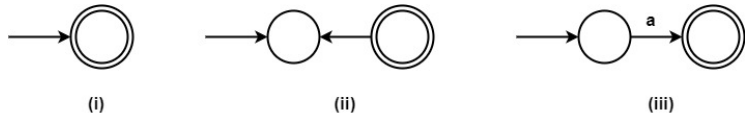


Figure 3.2: State diagrams of finite automata accepting languages Λ (null set), $\emptyset$ (empty set) and singleton set $\{a\}$

Note that the null set is not a set. But, empty set is a set containing no element.

Q.4.5.3.7 Remove the null-moves (i.e., Λ moves) from the given automaton (Fig. 3.3).

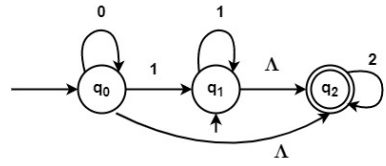


Figure 3.3: An NDFA

Answer: Each null-move is removed using a systematic process as given in steps 1 to 4. See Fig. 3.4.

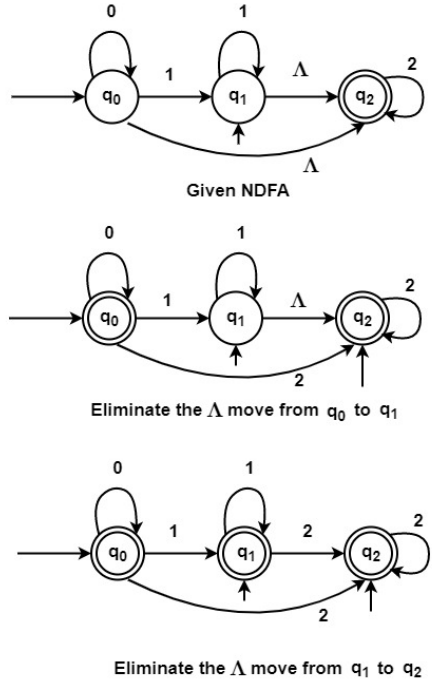


Figure 3.4: Removing null moves from the finite automaton given in Fig. 3.3

Let there exists a Λ move from state q_1 to state q_2. We want to replace this Λ move and the steps for removing a Λ move are given below.