FINITE AUTOMATA AND APPLICATIONS

by

NAWARA R.F ELFAKHAKHRE

Project Submitted in partial fulfillment

of the requirements for the degree

of Master of Sciences (Teaching of Mathematics)

JUNE 2010

ACKNOWLEDGEMENTS

I would like to thank the administration of The School of Mathematical Sciences, Universiti Sains Malaysia for all their help.

Of course, my advisor, Prof K.G. Subramanian, deserves great thanks. Without him I would not have discovered any things about finite automata and its applications, and without his guidance and advice. I would not have made it through graduate school.

Also I would like to thank Scholarship Department, Ministry of Education Libya for giving me Scholarship to upgrade my knowledge.

Finally, I would like to thank my husband and my parents and my parents in law for their support and encouragement.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

AUTOMATA TERHINGGA DAN APLIKASI

ABSTRAK

Automaton terhingga (finite automaton) boleh dianggap sebagai suatu model mesin teringkas, yang mempunyai memori terhingga. Saiz memori bergantung pada panjang input (input length). Automata terhingga merupakan model tertua daripada teori bahasa formal, iaitu sejak 1943. Ia digunakan oleh McCulloch dan Pitts dalam merintis konsep automaton sebagai suatu model bagi mengkaji sistem saraf. Kleene, kemudiannya menulis kertas kerja pertama tentang automata terhingga, yang akhirnya mewujudkan suatu teorem yang dikenali sebagai teorem Kleene. Pada tahun 1959, Michael Rabin dan Dana Scott menjalankan suatu kajian tentang teori automata terhingga "Finite Automata and Their Decision Problem", yang memperkenalkan idea tentang mesin yang tidak berketentuan (nondeterministic machine), yang terbukti suatu konsep yang penting. Dalam peringkat awal, teori automata terhingga telah dibangunkan sebagai suatu teori matematik bagi litar berjujukan (sequential circuits). Sesuatu litar berjujukan mampu mengekalkan keadaan semasa numbor terhingga (finite number). Logik litar (suatu kawalan keadaan terhingga) menentukan keadaan baru berdasarkan keadaan semasa litar dan simbol input yang diberikan. Apabila sesuatu simbol input diproses, litar tidak akan dapat membacanya lagi.

Aplikasi utama automata terhingga adalah pemprosesan teks. Dalam reka bentuk pengkompil (compiler design), automata terhingga digunakan untuk menentukan logik bagi analisis leksikal. Aplikasi automata terhingga yang lain antaranya: padanan rentetan (string matching), pemprosesan bahasa tabii (natural language processing), pemampatan teks (text compression), dll.

Dalam projek ini, kami memperkenalkan konsep daripada mana-mana automata terhingga dan meneliti definisi asas. Kami, kemudiannya memperkenalkan dua jenis automata terhingga iaitu automata terhingga berketentuan (deterministic finite automata, DFA) dan automata terhingga tidak berketentuan (nondeterministic finite automata, NFA). Kedua-dua jenis automata terhingga ini menerima keluarga bahasa yang sama, iaitu bahasa yang regular, yang akan diperkenalkan di sini. Kami juga menjelaskan kepentingan teorem Kleene's dalam kedua-dua bahagian.

Sebagai kesimpulan, kami juga turut memperkenalkan definisi asas bagi automata terhingga kabur (fuzzy finite automata) dan bahasa kabur regular (regular fuzzy languages) sebagaimana yang diutarakan oleh (Salomma et.al (1995), Martin Vide et.al (2004)).

ABSTRACT

A finite automaton can be considered as the simplest machine model in that the machine has a finite memory; that is, the memory size is independent of the input length. Finite automaton is the oldest model of formal language theory, since in 1943, in the pioneering work of McCulloch and Pitts, the concept of automaton appeared as a model for studying nervous systems. Then Kleene wrote the first paper on finite automata that proved a theorem which we now know as Kleene's theorem. In 1959, Michael Rabin and Dana Scott presented a study on the theory of finite automata, in their joint paper "Finite Automata and Their Decision Problem", which introduced the idea of nondeterministic machines, which has proved to be a very important concept. In the early stage, the theory of finite automata has been developed as a mathematical theory for sequential circuits. A sequential circuit maintains a current state from a finite number of possible states. The circuit logic (which is a finite state control) decides the new state based on the current state of the circuit and the given input symbol. Once an input symbol is processed, the circuit will not be able to read it again.

A main application of finite automata is text processing. In compiler design, finite automata are used to capture the logic of lexical analysis. Other applications include string matching, natural language processing, text compression, etc.

In this project, we first introduce the concepts of finite automata and review the basic definitions. Then we introduce the two types of finite automata deterministic finite automata (DFA), nondeterministic finite automata (NFA). The two types of finite automata really accept the same family of languages, the regular languages, that will be introduced here, and then we describe the important Kleene's theorem in two parts.

Finally, we also introduce the basic definitions for fuzzy finite automata and regular fuzzy languages as stated by (Salomma et.al (1995), Martin Vide et.al (2004)), which is application of finite automata such as lexical analyser.

CHAPTER 1

INTRODUCTION

In this chapter, the basic notions needed for the discussion in the subsequent chapters, are given based on (Linz 2006).

1.1 Mathematical Preliminaries and Notations

1.1.1 Sets

A set is a collection of objects. The objects in the set, called members or elements, are described by listing or by a property. For example, a finite set A = { 1, 2, 3,..., 10 } and an infinite set B = {1, 2, 3,...} can respectively be written as $A = \{ x: 1 \leq x \leq 10 \; and \; x \in N \}$, $B = \{x: x \geq 1 \; and \; x \in N\}$.

There are some special sets:

Empty set is a set which does not have any element and it is denoted by ∅ or { }. Universal set is the set of all objects or elements considered in a given problem. It is denoted by U. Also there is another set called Power set. The power set of X is the set of all subsets of X $defined \; as \; P(X) = \{ Y \mid Y \subseteq X \}$. This is also written as $P(X) = 2^X$.

For two sets A and B, if every element of set A is an element of set B, then A is

a subset of B. This is written as $A \subseteq B$. As an example, consider $X = \{5, 6, 7\}$,

$A = \{1, 2, 3, \ldots, 10\}$, *and* $Y = \{0, 1, 2\}$ *then* $X \subseteq A$, but $Y \nsubseteq A$ because $0 \notin A$.

The usual set operations are union (∪), intersection (∩), and difference (-). Another operation is the complement of a set. The complement of a set A with respect to U is denoted by $\bar{A}$. In mathematical symbol, all these are written as

$\bar{A} = \{x \mid x \in U \text{ and } x \notin A\}$

$A \cup B = \{x \mid x \in A \text{ or } x \in B \text{ or both}\}$

$A - B = \{x \mid x \in A \text{ and } x \notin B\}$

$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$.

Example:

Consider $U = \{u \mid u \in N\}$, $U_1 = \{7, 8, 9\}, U_2 = \{9, 10, 11, 12, 13\}$ *and*

$U_3 = \{121, 120\}$ Then $\bar{U}_1 = \{1,2,3,4,5,6,10,11,12, \ldots\}$,

$U_1 \cup U_3 = \{7,8,9,121,120\}$ and $U_1 \cap U_3 = \emptyset$ *and* $U_1 \cap U_2 = \{9\}$.

1.1.2 Functions and Relations

The Cartesian product of sets $X_1, X_2, X_3, \ldots\ldots, X_n$ is defined as:

$$X_1 \times X_2 \times \ldots \times X_n = \{(x_1, x_2, \ldots, x_n) \mid x_1 \in X_1, x_2 \in X_2, \ldots, x_n \in X_n\}$$

A relation on $X_1, X_2, X_3, \ldots, X_n$ is a subset of $X_1 \times X_2 \times \ldots \times X_n$. There are many types of relation, such as Binary relations, Ordering relations, and equivalence relations. A function is a binary relation between two sets X, Y with the

condition that for every element $x \in X$, there is a unique element y of Y, such that $f(x) = y$.

If $f: X \to Y$ is a function then the domain of the function f is the set of X, and we write $D_f = \{x: x \in X; f: X \to Y\}$. The co-domain of the function f is the set of elements of Y, and the range of the function f is the set of elements such that $y = f(x)$; y is then called the image of x, we write $R_f = \{y: y \in Y; f: X \to Y\ y = f(x)\}$.

Equivalently we can define a function $f: A \to B$ as a relation R where $R: A \to B$ that satisfies the following two conditions:

(i) $\forall a \in A, \exists b \in B\ such\ that\ (a, b) \in f.$

(ii) $\forall a \in A, \forall b, c \in B, (a, b) \in f and\ (a, c) \in f \Longrightarrow b = c.$

Example:

In Figure 1 below,

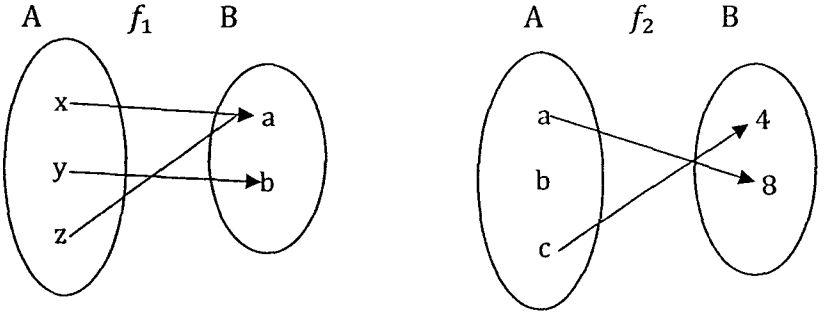

Figure 1 Mappings or functions f_1, f_2

observe that f_1 is a function since $f(x) = a,\ f(y) = b,\ f(z) = a,$ and the range is {a, b}, but $f_2$is not a function because $b \in A$, but b does not have any image in the co-domain B.

1.1.3 Graphs and Trees

A graph is a set of vertices with lines joining some of the vertices. The vertices are called nodes, and the lines are called edges. Each edge can be an arc consisting of an ordered pair of vertices and represents possible direction of motion that may occur between vertices. Suppose that the edge consists of a pair of nodes i and j then we write the edge as a pair (i, j). Usually, in the graph the node is denoted by a circle and the edge by a line as shown in the Figure 2.

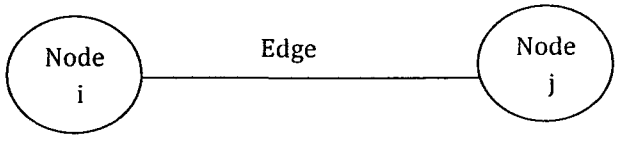

Figure 2 A Graph with two nodes and an edge

Thus we can denote the graph G as a pair (V,E) when V is a finite set of nodes and E is a finite set of edges which is simply a binary relation on V. We use the graph to represent information on problems so that it becomes easy to understand. A graph can be drawn in many ways. Also we can divide the graph into two types: directed graph and undirected graph. The difference between them is that the edges in a directed graph carry information about directions; For example consider the directed graph in Figure 3.

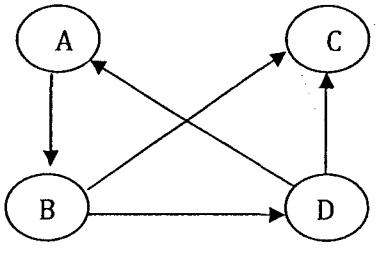

Figure 3 Directed graph

In the graph of Figure 3, we have four nodes and five edges, each edge possesses a sense of a direction such as the edge (A,B) represents leaving node A and entering node B. A description of the graph as formally is as follows:

$$G = (\{A, B, C, D\}, \{(A, B), (B, D), (D, C), (D, A), (B, C)\})$$

But in undirected graphs no sense of the direction of edges is considered. For example consider the same graph in Figure 3 but without a sense of direction. Then it will become like the graph in Figure 4.

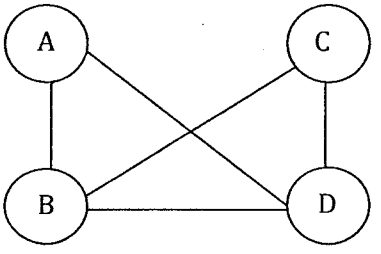

Figure 4 Undirected graph

The edge (A,B) represents either leaving A to enter B or leaving B to enter A. We can describe this formally as follows:

$G = (\{A, B, C, D\}, \{(A,B),(B,A),(A,D),(D,A),(B,C),(C,B),(B,D),(D,B),(D,C),(C,D)\})$.

The degree of the node corresponds to the number of edges at the node. In the Figure 4 the nodes A,C have degree 2 and the nodes B,D have degree 3. In-degree of a vertex is the number of edges entering it. The out-degree of a vertex is the number of edges leaving it. An edge is called loop when joining a vertex to itself. A loop at a vertex counts 2 to the degree because we count each end of the loop. A vertex is called isolated if the degree of the vertex zero .Two or more

edges joining the same pair of vertices are called multiple edges. A graph with no multiple edges and loops is called a simple graph. If G' is a sub graph of the graph G then the nodes of G' is subset of nodes of G and the edges of G' is subset of the edges of G, like that in Figure 5.

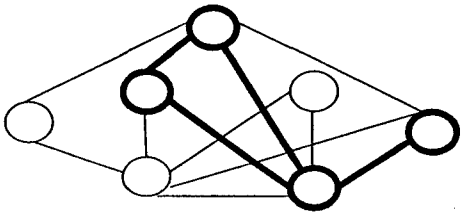

Figure 5 Graph G, and Sub graph G' (shown darker)

A path is a chain of nodes joined by edges. If every two nodes in the graph have a path between them then the graph is connected. A path is a cycle if it starts and ends in the same node. If the graph consists of at least three nodes and repeats only the first and the last nodes then it is a simple cycle.

A tree of graph is connected graph that has no cycles, and has the shape like a tree when the root of a tree is the node of degree 1 and the nodes which have more degree one are leaves of a tree.

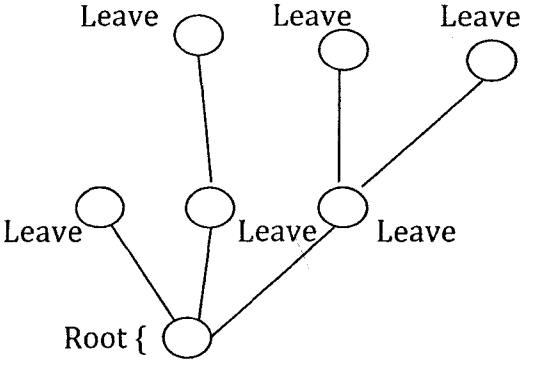

Figure 6 Tree of the graph

1.2 Three basic concepts

There are three basic concepts to define any finite automata which are as follows:

1.2.1 Languages

The concept of a language is introduced after introducing certain other concepts. First we will define symbols, alphabets then string to arrive in the end to define a language.

Symbols can be letters a,b,c,...,z or numbers 0,1,2,....,9 , or Bits 0,1 or English word like cat, mouse also syntactic components of a programming language such as for , begin and ; : =. Symbols are elements of an Alphabet where an Alphabet is a finite set of symbols denoted by $\sum$. For example $\sum_1 = \{0,1\}, \sum_2 = \{a, c, r, e, p, h, l, n, t\}$ are alphabets. A string or word over an alphabet $\sum$ is a finite sequence of symbols, for example

$w_1 = car\ and\ w_2 = elephent$ are two strings over$\sum_2$, and strings over $\sum_1$ are sequences of binary digits $, s_1 = 11000011000100101100111001 1$, $s_2 = 110011$. If a string v appears within another string w then v called a substring of w. 1, 11, 10, 100, 011 are substrings of s_2.

Every string w has a length, denoted by |w| which is the length of sequence of symbols such as $|w_2| = 8\ and\ |s_2| = 6$. The string which has length zero called the null string or empty string which means that the string does not have any symbols. It is denoted $\epsilon, \wedge\ or\ e$. If w and s are two strings, then the concatenation of w and s is denoted by w.s or just ws is the string obtained by

having the symbols of s to the right end of w such as the concatenation of $w_1$and s_2 is $w_1 s_2 = car110011$.

The identity string of concatenation operation is null string since every string w, satisfies $\lambda\, w = w = w\, \lambda$. String s is a substring of w if $w = xsy$ for some strings x and y. String s is a prefix of w if $w = sy$ for some strings y; String s is a suffix of w if $w = xs$ for some strings x. For example some of suffixes of w_2 are hent, ent, nt, and t; and while the set of all prefixes of the string w_2 is $\{\lambda, e, el, ele, elep, eleph, elephe, elephen, elephent\}$.

String w^R is the reversal of w if it is the sequence w in last-to-first order. String w is a palindrome, when $w = w^R$, like $s_2 = 110011 = 110011 = s_2^R$. If w is a string then the power of it is denoted w^n which means repeating w, n times. The string with power zero is empty string $w^0 = \lambda$. The Kleene Star (Closure) is a set of all finite strings over $\sum$ including the empty string λ, denoted to it by $\sum^*$ when $\sum$ is any finite alphabet. Always the set $\sum^*$ is an infinite set because there is no limit on the length of the strings which in the set. We define $\sum^+$ as $\sum^+ = \sum^* - \{\lambda\}$, it is the Kleene Star but does not include the empty string.

A language over $\sum$ is any subset of $\sum^*$, is denoted by L; this means $L \subseteq \sum^*$. Every element in L is called sentence or word or string. For example for alphabet $\sum_1 = \{0,1\}$ then $\sum_1^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, \ldots.\}$ the set $\{0,00,001\}$ is subset of $\sum_1^*$ thus it is a language on $\sum_1$. Note that the set $\{0,00,001\}$ has a finite number of sentences and so this language is called a finite language. But an infinite language which has infinite number of sentences such

as the set $L = \{0^n 1^n : n \geq 0\}$ which is a language on $\sum_1$. The strings 01,0011,000111 are in L but 011 is not in L.

For example the languages over $\sum = \{a, b, c\}$ are $\emptyset, \{aaab, aabb, abab, abbb, aacb, acab, accb, abcb, acbb\}$ *and* $\{w \in \sum^* | \, |w| \geq 7\}$. When the first represents to the empty language i.e language does not contain any string, the second is the set of strings of length 4 that begin with a and end with b, and the last one is the set of all strings of length at least 7.

As languages are sets of words, so the usual operations that apply to sets can be applied to languages. For the union(∪), intersection(∩) and diffrence (−) of two languages directly defined. But the complement of the language is defined with respect to $\sum^*$, denoted to it $\bar{L}$ defined as

$$\bar{L} = \sum^* - L .$$

The reverse of a language is the set of all string reversal, defined as

$$L^R = \{ w^R : w \in L \}.$$

Cross product of two languages $L_1 and\ L_2$ represented $L_1 \times L_2$ is a string of all pairs (x, y) when x is in L_1, y is in L_2

$$(L_1 \times L_2) = \{(x, y) | \, x \in L_1 and\ y \in L_2 \}$$

The concatenation of two languages L_1, L_2 which denoted $L_1 L_2$ defined as

$$L_1 L_2 = \{ xy : x \in L_1, y \in L_2 \}.$$

For every language satisfy two properties

1. $L\{\lambda\} = \{\lambda\}L = L.$
2. $L\emptyset = \emptyset L = \emptyset.$

Power i of a language L since $i \geq 1$ denoted L^i defined as, when $i = 0$ will be $L^0 = \{\lambda\}$ and for all $i \geq 1, L^i = L\,L^{i-1}$. The following some examples:

(i) Consider $\sum = \{a, b\}$ *then* $\sum^* = \{\lambda, a, b, aa, ab, ba, bb, \ldots..\}$ if $L_1 = \{ab, b, bb\}$ *and* $L_2 = \{ab, aa, a, abba\}$ then $L_1 \cap L_2 = \{ab\}$, $L_1 \cup L_2 = \{ab, b, bb, aa, a, abba\}$,$L_2 - L_1 = \{aa, a, abba\}$,

(ii) $\overline{L_1} = \{\lambda, a, aa, ba, aaa, \ldots..\}$ and $L_2{}^R = \{ba, aa, a, abba\}$.

(iii) If $L_1 = \{a, aba, cab, \lambda\}$ and $L_2 = \{ca, cb\}$ then $L_1L_2 = \{aca, acb, abaca, abacb, cabca, cabcb, ca, cb\}$

(iv) If $L_1 = \{\lambda, 0, 1\}$, $L_2 = \{00, 10\}$ then

$$L_1 \times L_2 = \{(\lambda, 00), (\lambda, 10), (0,00), (0,10), (1,00), (1,10)\}$$

(v) If $L = \{c, d\}$ then

(a) $L^0 = \lambda$

(b) $L^1 = \{c, d\}$

(c) $L^2 = LL = \{c, d\}\{c, d\} = \{cc, cd, dc, dd\}$

(d) $L^3 = LLL = LL^2 = \{c, d\}\{cc, cd, dc, dd\}$ thus

$$L^3 = \{ccc, ccd, cdc, cdd, dcc, dcd, ddc, ddd\}.$$

Finally, the Kleene closure or star-closure of a language L is

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots\ldots\ldots = \cup_{i=0}^{\infty} L^i$$

While the KLeene plus or positive closure of a language is

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots\ldots = \cup_{i=1}^{\infty} L^i$$

Example (1.2.1.1):

If $\sum = \{0,1\}$ and $L = \{1,01\}$ then L^* consists of the empty string λ and all the strings that can be formed using 1 and 01 with the property that every 0 is followed by 1. For L^+ it will be the same set but does not consist of λ.

$$L^* = \{\lambda, 1, 01, 11, 0101, 101, 011, \ldots\ldots\}$$

$$L^+ = \{1, 01, 11, 0101, 101, 011, \ldots\ldots\}$$

Example (1.2.1.2):

Let $L = \{ab, aa, baa\}$. Which of the following strings are in L^*: $abaabaaabaa, aaaabaaaa, baaaaabaaaab, baaaaabaa$? which strings are in L^4?

Notice that $abaabaaabaa$ can be split into strings ab, aa, baa, ab, aa each of which is in L. Also for $aaaabaaaa$, $baaaaabaa$ split into aa, aa, baa, aa ; and baa, aa, ab, aa respectively with each of them in L, the strings $aaaabaaaa$, $baaaaabaa$ are in L^4 because they contain strings of length 4 in L. Thus the strings $abaabaaabaa, aaaabaaaa$, $baaaaabaa$ are in L^*. But, the string $baaaaabaaaab$ is not in L^* because there is no possible to express it in terms of elements of L. In fact,

$L = \{ab, aa, baa\}$

$L^2 = \{ab, aa, baa\}\{ab, aa, baa\}$

$= \{abab, abaa, abbaa, aaab, aaaa, aabaa, baaab, baaaa, baabaa\}$

$L^3 = L\,L^2$

$= \{ab, aa, baa\}\{abab, abaa, abbaa, aaab, aaaa, aabaa, baaab, baaaa, baabaa\}$

$= \{ababab, ababaa, ababbaa, abaaab, abaaaa, abaabaa, abbaaab, abbaaaa,$
$abbaabaa, aaabab, aaabaa, aaabbaa, aaaaab, aaaaaa, aaaabaa, aabaaab,$
$aabaaaa, aabaabaa, baaabab, baaabaa, baaabbaa, baaaaab, baaaaaa,$
$baaaabaa, baabaaab, baabaaaa, baabaabaa\}$

$L^4 = LL^3 = \{abababab, \ldots, aaaabaaaa, \ldots\ldots, baaaaabaa, \ldots., baabaabaabaa\}$

1.2.2 Grammars

A grammar G is defined as a quadruple $G = (V, T, S, P)$ where

(i) V is a finite nonempty set of objects called variables.

(ii) T is a finite nonempty set disjoint from V of objects called terminal symbols.

(iii) $S \in V$ is a special symbols called start variable.

(iv) P is a finite set of productions.

The grammar transforms one string into another by production rules and the strings obtained are the elements of the language of G. All production rules can be formally expressed as $x \rightarrow y$ when x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. If there is a string w of the form $w = uxv$, then the production $x \rightarrow y$ means replace x with y in the new string which will yield $z = uyv$. Thus we write $w \Rightarrow z$ to represent that w derives z or that z is derived from w, which means that z is directly derivable from string w. A sequence of direct derivations like $w \Rightarrow u_1, u_1 \Rightarrow u_2, \ldots\ldots, u_n \Rightarrow z$ is a derivation of the string z

from the string w. Sometimes it is called parsing. The string z is derivable from the string w with respect to the grammar G if there is a derivation of z from w represented by $w \Rightarrow^* z$. The language generated by G, when G is the grammar, is the language $L(G) = \{w \in T^* : S \Rightarrow^* w\}$. If $w \in L(G)$, then the sequence $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$ is the derivation of the sentence w. The strings $S, w_1, w_2, \ldots\ldots, w_n$, which contain variables as well as terminals, are called sentential forms of the derivation.

Example (1.2.2.1):

Consider the grammar $G = (\{s\}, \{a, b\}, S, P)$ with P given by

$$S \rightarrow aA$$

$$A \rightarrow bS$$

$$S \rightarrow \lambda$$

Then $S \Longrightarrow aA \Longrightarrow abS \Longrightarrow abaA \Longrightarrow ababS \Longrightarrow abab$

So we can write $S \Longrightarrow^* abab$

The string $abab$ is a sentence in the language generated by G, while $ababS$ is a sentential form. It is easy to deduce that $L(G) = \{(ab)^n : n \geq o\}$, we will prove that by induction. Notice that the rule $S \rightarrow abs$ is repeated so we first show that all sentential forms must have the form $w_i = (ab)^i S$.

Suppose that $w_i = (ab)^i S$ hold for sentential forms w_i of length $2i + 1$ or less. We have to prove this for $i + 1$. By applying the productions $S \rightarrow aA$, $A \rightarrow bS$ to get that $(ab)^i S \Rightarrow (ab)^{i+1} S$ so that every sentential form of length $2i + 3$ is also

CHAPTER 2

FINITE AUTOMATA

2.1 Finite state systems

Finite automaton is the simplest model of computation. It is a mathematical model that consists of discrete outputs, discrete inputs and states.

It summarizes information concerning past inputs which need to determine the manner of the system on subsequent input. For example:

Theoretically, a computer and human brain can be viewed as finite state systems when in the computer the state of the system is a central processor, main memory, and auxiliary storage at any time is one of a very large but finite number of states, one cannot extend the memory indefinitely. And in human brain the states will be the cells which are evaluated as 2^{35}. Also finite state automaton is the simplest model of working of a computer program and used to model processes which involve the determination of whether a particular string is acceptable for specific purpose. In this chapter we discuss finite automata based on (Linz, P. (2006) & Martin, J. C. (2003)).

Definition (2.1.1):

Finite automata or finite state automata (FA) are composed of a finite number of states and transitions between those states that occur on input symbols chosen from an alphabet $\sum$. For the same input symbol there may be 0,1 or more transition from each state. The automaton starts at a state called initial state or start state, and some states are as final states or designated as accepting states. If there is exactly one transition from each state for each input symbol then the FA is said to be Deterministic finite automaton (DFA), otherwise the FA is called a nondeterministic finite automaton (NFA).

A finite automaton is a function from $\sum^*$ into {Yes, No} in the following way:

Given a string with the first symbol k the first value of $\delta(q_0, k) = q_1$ is estimated then continued until we come up to the final state. If the final estimation of δ leads to accepting state then the string is accepted by the automaton, the automaton having returned a value of yes. If the final estimation of δ does not lead to accepting state, the string is rejected, the automaton, having returned a value of No.

To describe the mathematical theories of finite automata it must be done in a manner devoid without linking it to any specific applications.

The following figure depicts a finite automaton called M_1:

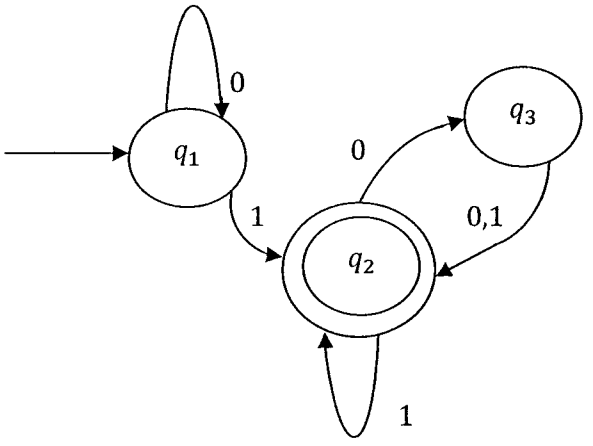

Figure 8 Diagram to a finite automaton called M_1

Observe that M_1 has three states, labeled $q_1, q_2, and\ q_3$ which as vertices of the graph where:

- q_1 called the start state which indicated by the arrow labeled "start".
- There is one accept state is q_2 which indicated by double circle.
- The arrows in the diagram represent to move from any state to another called transitions.
- Where δ is given by

$$\delta(q_1, 0) = q_1\,, \quad \delta(q_1, 1) = q_2$$

$$\delta(q_2, 0) = q_3\,, \quad \delta(q_2, 1) = q_2$$

$$\delta(q_3, 0) = q_2\,, \quad \delta(q_3, 1) = q_2$$

Assume, this automaton receive an inputs string such as 1011 to get the outputs either accept or reject will consider only this yes/no type of output. Processing begin in M_1's start state when it receives symbols of the input string and one after other from left to right. After reading every symbol M_1moves from one state to other, at read the last state M_1 produces its output.

These outputs are accept if M_1 in accept state when read the last symbol of inputs, and are reject if it is not.

For example when input the string 1011 to automaton M_1 which represents the above diagram, the processing proceeds as follows.

1. Start in state q_1.
2. Read 1, $\delta(q_1, 1) = q_2$.
3. Read 0, $\delta(q_2, 0) = q_3$.
4. Read 1, $\delta(q_3, 1) = q_2$.
5. Read 1, $\delta(q_2, 1) = q_2$.
6. Accept because N_1 in an accept state $q_2$at the end of the input.

When testing a large number of strings on this machine can be deduced the following:

- All the strings which end with 1,11,01,111101,01101010101, its outputs accept.
- All the strings which end with even number of zeros after 1 such as 100, 1000000, its outputs accept.
- All the strings which end with odd number of zeros such as 01000 its outputs reject.

2.2 Deterministic finite automata (DFA)

Deterministic finite automaton (DFA), is a model of with a finite amount of memory since when an input enter s and v then read, the DFA decides if the input is accepted or rejected. There is a uniquely way to determine the next

state by the current state and the current input, in a DFA. For example the M_1 described before is a DFA.

A deterministic finite accepter or DFA is mathematically defined by 5-tuple $(Q, \sum, q_0, A, \delta)$ where

- Q is a finite set of internal states.
- $\sum$ is a finite alphabet of input symbols.
- $q_0 \in Q$is the start state or initial state.
- $A \subseteq Q$ is a set of accepting states
- δ is a function from $Q \times \sum$ to Q which is the transition function.

Example (2.2.1):

We can describe M_1 described earlier formally by writing $M_1 = (Q, \sum, q_0, A, \delta)$ where

- $Q = \{q_1, q_2, q_3\}$
- $\sum = \{0,1\}$
- δ is called transition function and is given by the following table:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- q_1 is the start state
- $A = \{q_2\}$.

If all strings that machine M accepts in the set denoted by F then F is the language of machine M written as $L(M) = F$. we say that M recognizes F or that

M accepts A. A machine always recognizes only one language and many strings may be accepted by a machine. The empty language $\emptyset$ is a language which is recognized by a machine when that machine accepts no strings.

For example (2.2.1), let

$F = \{w: w \text{ consists at least } 1 \text{ and even number of } 0s \text{ follow the last } 1\}$

Then $L(M_1) = F$, or equivalently, M_1 recognizes A, where $A = \{q_2\}$.

The transition function $\delta: Q \times \sum \longrightarrow Q$ can be extended to δ^* which is defined as $\delta^*: Q \times \sum^* \longrightarrow Q$ when $\forall\, q \in Q, \delta^*(q,\lambda) = q$ and $\forall\, q \in Q,\ y \in \sum^* and\ a \in \sum\ \delta^*(q, ya) = \delta(\delta^*(q, y), a)$. For example if we have a finite automaton called M_2 as in Figure 9, then

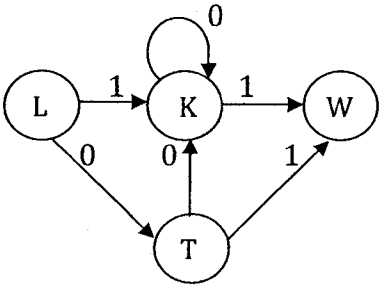

Figure 9 Diagram to a finite automaton is called M_2

to calculate $\delta^*(L, 0001)$:

$$\delta^*(L, 0001) = \delta(\delta^*(L, 000), 1)$$
$$= \delta(\delta(\delta^*(L, 00), 0),1)$$
$$= \delta(\delta(\delta(\delta^*(L, 0),0), 0),1)$$
$$= \delta(\delta(\delta(\delta^*(L,\lambda\ 0),0), 0),1)$$
$$= \delta(\delta(\delta(\delta(\delta^*(L,\lambda),0),0), 0),1)$$
$$= \delta(\delta(\delta(\delta(L, 0), 0), 0), 1)$$

$$= \delta(\delta(\delta(T,0),0),1)$$

$$= \delta(\delta(K,0),1)$$

$$= \delta(K,1) = w$$

Since DFA will process every string in $\sum^*$ and either is accepted or is not accepted by $M = (Q, \sum, q_0, A, \delta)$ and so in general the language accepted by M can be defined using extended function as

$$L(M) = \{w \in \sum^* : \delta^*(q_0, w) \in A\}$$

Therefore the complement of $L(M)$ will be all strings nor accepted by DFA M and is defined as

$$\overline{L(M)} = \{w \in \sum^* : \delta^*(q_0, w) \notin A\}$$

Example (2.2.2):

To design a DFA, the language recognized by the automaton being $L = \{b^n a : n \geq 0\}$. For the given language $L = \{b^n a : n \geq 0\}$, the strings could be $a, ba, b^2a, b^3a, \ldots\ldots$, so the DFA accepts all strings consisting of an arbitrary number of $b's$, followed by a single a, so that the DFA will be as in Figure 10.

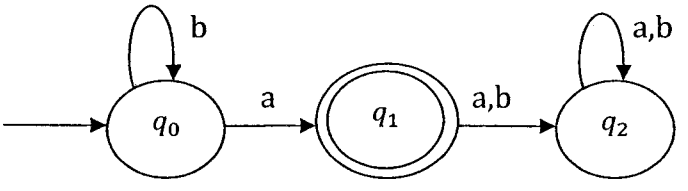

Figure 10 DFA that accepts the language $L = \{b^n a : n \geq 0\}$

Example (2.2.3):

Given $\sum = \{a, b\}$, to construct a DFA that shall recognize the language

$L = \{a^n ba^m : n, m > 0\}$.

The Given language $L = \{a^n ba^m : n, m > 0\}$ represents all strings with exactly one b which is neither the first not last letter of the string. This means there is one or more a's before or after a. The DFA is as in Figure11.

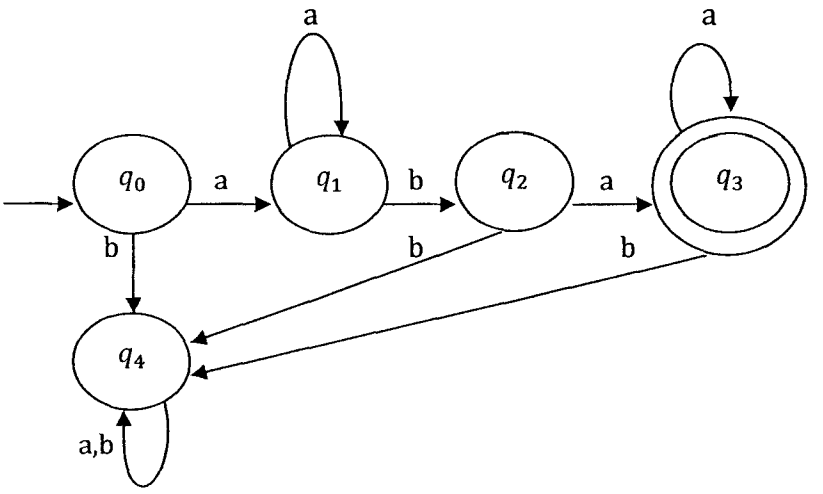

Figure 11 A DFA to recognize the language $L = \{a^n ba^m : n, m > 0\}$

Here the DFA $M = (Q, \Sigma, q_0, A, \delta)$ with, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$; q_0 is initial state and $A = \{q_3\}$ final state when δ is defined as per the language $L = \{q_4 \text{ is dead state}\}$.

Example (2.2.4):

To obtain the state table diagram and state transition diagram of the DFA

$M_2 = (Q, \Sigma, q_0, A, \delta)$ where $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$ and $A = \{q_0\}$ with transition defined by

$\delta(q_0, b) = q_2$,$\delta(q_3, b) = q_1, \delta(q_2, a) = q_3, \delta(q_1, b) = q_3, \delta(q_0, a) = q_1,$

$\delta(q_3, a) = q_2, \delta(q_2, b) = q_0, \delta(q_1, a) = q_0.$