4-25-2024

Finite State and Sequential Automata

Kriti Gulgulia
Western Michigan University

Finite-State and Sequential Automata

Kriti Gulgulia

April 25, 2024

Abstract

In this thesis, we introduce the concept of a Moore machine $M = (S, I, F, s_0, T)$ and construct a quotient machine $\bar{M} = (\bar{S}, I, \bar{F}, [s_0], \bar{T})$. We explore the notion of a machine's language $L(M)$ and examine its properties such that $L(M) = L(\bar{M})$. In this context, we demonstrate the technique of constructing simplified Moore Machines. Furthermore, we look at the notion of Mealy machines $M = (S, I, \theta, F, G, s_0, T)$ in relation to output function and output symbol. Within the context of this thesis, we present a demonstration of the machines in arrangements. Furthermore, we examine linear finite state machines.

Contents

1 Finite-State Automata and Languages

1.1 Introduction

The Theory of Automata is a fundamental and enduring mathematical theory that focuses on the study of abstract machines, known as automata and their computational capacities. It was first developed to model the capabilities of computer systems. Currently, automata are widely used in many activities such as the creation and validation of hardware and software systems, as well as in text and speech recognition and digital picture processing. This thesis will primarily examine two forms of finite state machines: the Moore machine and the Mealy machine. This chapter focuses on the concept of a Moore machine.

We think of machine as system that can accept input, possibly produce output and have some sort of internal memory that can keep track of certain information about previous inputs. The complete Internal condition condition of the machine and all of it's memory at any particular time is said to constitute state of the machine at that time. The state in which a machine finds itself at any instant summarizes it's memory of past inputs and determines how it will react to subsequent input. When more input arrives, the given state of machine determines the next state to be occupied and any output that may be produced.

Definition 1.1.1. If the number of states is finite the machine is finite state machine. Suppose that we have a finite set S = $\{s_0, s_1, ..., s_n\}$, a finite set I and for each $x \in I$, a function $f_x : S \mapsto S$. Let $F = \{f_x \mid x \in I\}$. The triple (S,I,F) is called a *finite state machine* where S is called state set of the machine and it's elements are called states. The set I is called the input set of machine and the function f_x which describes the effect that occurs on states of machine for any input is called state transition function.

So if a machine is in state s_i and input x occurs the next state of the machine will be $f_x(s_i)$. The following illustrates an example of a finite state machine.

Example 1.1.2: Let $S = \{s_0, s_1\}$ and $I = \{0, 1\}$, so we define f_0 and f_1 as follows: $f_0(s_0) = s_0, f_1(s_0) = s_1, f_0(s_1) = s_1$ and $f_1(s_1) = s_0$. This shows that the given finite-state machine has two states s_0 and s_1 accepts two possible inputs 0 and 1. We observe that the input 0 leaves each state fixed whereas input 1 reverses each state. The state-transition table the digraph of the machine is shown below.

	0	1
s_0	s_0	s_1
s_1	s_1	s_0

Table 1. State-transition table

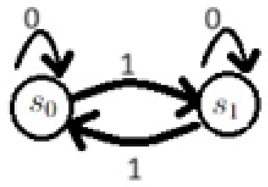

Figure 1: Digraph of Moore machine M

Now we can think of above machine as a model for a circuit device and visualize such a device. The output signals will, at any given time, consists of two voltages, one higher line than the other. Either line 1 will be at the higher voltage and line 2 at the lower, or the reverse. The first set of output conditions will be denoted as s_0 and the second will be denoted as s_1. An input pulse, represented by the symbol 1, will reverse output voltages. The symbol 0 represents the absence of an input pulse and so results in no change of output. This device is often called a T flip-flop and is a concrete realization of the machine. We summarize this machine in Table 1. The table shown there lists the states down the side and input across the top. The column under each input gives the value of the function corresponding to that input at each state shown on the left. The table given for summarizing the effects of inputs on states is called state transition table of the finite state machine. It can be used with any machine of reasonable size and is a convenient method of specifying the machine. Let us look at another example of Finite State Machine with digraph given below.

Example 1.1.3: Let us now consider the machine M whose table is given below.

	a	b	c
s_0	s_0	s_0	s_0
s_1	s_2	s_3	s_2
s_2	s_1	s_0	s_3
s_3	s_3	s_2	s_3

Table 2. State-transition table

Then the digraph of the machine is

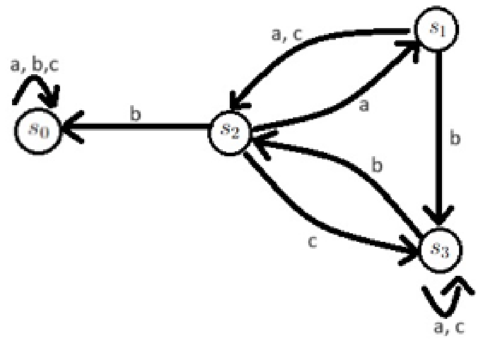

Figure 2: Digraph of Moore machine M

Note that an edge may be labeled by more than one input, since several inputs may cause the same change of state. The reader will observe that every input must be part of the label of exactly out of each state. This is a general property that holds for the labeled diagraph of all finite-state machines. For brevity, we will refer to the labeled diagraph of a machine M simply as the digraph of M. It is possible to add variety of extra features to a finite-state machine in order to increase the utility of the concept. A simple, yet very useful extension results in what is often called Moore machine or recognition machine. The Moore machine is named after Edward F. Moore, an American mathematician and computer scientist who lived from 1925 to 2003. He was an early pioneer of artificial life and introduced the notion in a 1956 paper on finite state Machines.

Definition 1.1.4. Moore machine is defined as $M = (S, I, F, s_0, T)$ where (S, I, F) constitutes a finite-state machine where $s_0 \in S$ and $T \subseteq S$. The state s_0 is called the starting state of M and it will be used to represent the condition of the machine before it receives any input. The set T is called the set of acceptances states of M.

When the diagraph of Moore machine is drawn, the acceptances state are indicated with two concentric circles, instead of one.

Definition 1.1.5. Let M = (S,I,F) be a finite-state machine and suppose that R is an equivalence relation on S. We say that R is a Machine congruence on M if for any $s, t \in S$, s R t implies that $f_x(s)$ R $f_x(t)$ for all $x \in I$.

In other words, R is a machine congruence if R-equivalent pairs of states are always taken into R-equivalent pairs of states by every input in I. If R is a machine congruence on M = (S,I,F) we let $\bar{S} = S/R$ be the partition of S corresponding to R. Then $\bar{S} = \{[s] \mid s \in S\}$. If $M = (S, I, F, s_0, T)$ is a Moore machine and R is a machine congruence on M, then we may let $\bar{T} = \{[t] \mid t \in T\}$. Here, the sequence $\bar{M} = (\bar{S}, I, \bar{F}, [s_0], \bar{T})$ is a Moore machine. In other words, we compute the usual quotient machine M/R; then we designate $[s_0]$ as a starting state, and let $\bar{T}$ be the set of equivalence classes of acceptance states. The resulting Moore machine $\bar{M}$ constructed this way will be called the *quotient Moore machine* of M.

1.2 Machines and Regular Languages

In this section we will be looking at language of a machine.

Let $M = (S, I, F)$ be a finite state machine with the state set $S = \{s_0, s_1, s_2, ..., s_n\}$, input set I and state transition function $F = \{f_x, \mid x \in I\}$. We will associate with M two monoids.

First, there is the free monoid I^* on the input set I. This monoid consists of all finite sequences from I, with catenation as it's binary operation. The identity is the empty string Λ. Second, we have the monoid S^S, which consists of all functions from S to S and which has the function composition as it's binary operation. The identity in S^S is the function 1_S defined by $1_s(s) = s$ for all s in S.

Definition 1.2.1. If $w = x_1x_2.....x_n \in I^*$, we let $f_w = f_x \cdot f_y \cdot .... \cdot f_v$ the composition of functions. Also we define f_Λ to be 1_s. In this way we assign an element f_w of S^S to each element w of I^*. If we think of each f_x as the effect of the input x on the states of machine M then f_w represents the combined effect of all input letters in the word w received in the sequence specified by w. So we call f_w as the **state transition function corresponding to w.**

This method of interpreting word transition functions such as f_w and f'_w is useful in designing machines that have word transitions possessing certain desired properties.This is a crucial step in the practical application of the theory and we will consider it in the next section.

Let $M = (S, I, F)$ be a finite state machine. We define a function T from I^* to S^S. If w is a string in I^* let $T(w) = f_w$ as defined previously. Then we have the following result.

Theorem 1.2.2. If $w_1, w_2, ..., w_n \in I^*$ then

$$T(w_1 \cdot w_2 \cdot .....w_n) = T(w_n) \cdot T(w_{n-1}) \cdot ......T(w_2) \cdot T(w_1).$$

Proof. Now let $w_1 = a_1a_2.....a_n$, $w_2 = b_1b_2....b_n$,....., $w_n = z_1z_2....z_n$ be strings in I^*. Then we have
$T(w_1 \cdot w_2 \cdot ...... \cdot w_n) = T(a_1a_2.....a_nb_1b_2....b_nz_1z_2....z_n)$
$= (f_{zn} \cdot f_{zn-1} \cdot ......f_{z1} \cdot .... \cdot (f_{bn} \cdot f_{bn-1} \cdot ......f_{b1}) \cdot .... \cdot (f_{an} \cdot f_{an-1} \cdot ......f_{a1})$

$= T(w_n) \cdot T(w_{n-1}) \cdot ..... \cdot T(w_2) \cdot T(w_1)$. □

Example 1.2.3: Let $M = (S, I, F)$, where $S = \{s_0, s_1, s_2\}$, $I = \{0, 1\}$ and F is given by the following state transition table.

	0	1
s_0	s_0	s_1
s_1	s_2	s_2
s_2	s_1	s_0

Table 3. State-transition table

Let $w = 011 \in I^*$. Then,

$$f_w(s_0) = (f_1 \circ f_1 \circ f_0)(s_0) = (f_1(f_1(f_0(s_0))) = (f_1(f_1(s_0)) = (f_1(s_0)) = s_2.$$
$$f_w(s_1) = (f_1 \circ f_1 \circ f_0)(s_1) = (f_1(f_1(f_0(s_1))) = (f_1(f_1(s_2)) = (f_1(s_0)) = s_1$$
$$f_w(s_2) = (f_1 \circ f_1 \circ f_0)(s_2) = (f_1(f_1(f_0(s_2))) = (f_1(f_1(s_1)) = (f_1(s_2)) = s_0.$$

Theorem 1.2.4: If $M = T(I^*)$, then M is a submonoid of S^S.

Proof. From theorem 1.2.2 we know that if f and g are in M then $f \cdot g$ and $g \cdot f$ are in M. Thus M is a subsemigroup of S^s. Since $1_s = T(\Lambda), 1_s \in M$. Thus M is a submonoid of S^s. The monoid M is called the monoid of the machine M. □

Example 1.2.5 Let $M = (S, I, F)$, where $S = \{s_0, s_1, s_2\}$, $I = \{0, 1\}$ and F is given by the following state transition table. We will show that $f_w(s_0) = s_0$ if and only if w has *3n* 1's for some $n \geq 0$.

	0	1
s_0	s_0	s_1
s_1	s_1	s_2
s_2	s_2	s_0

Table 4. State-transition table

We first construct the digraph of machine M as shown in Figure 3 below.

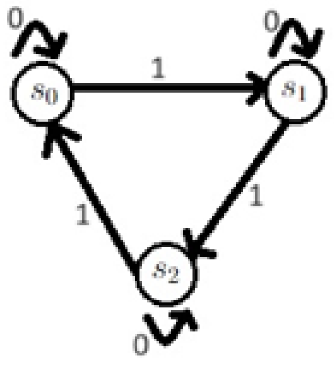

Figure 3: Digraph of Moore machine M

Observe that $f_0 = 1_s$, so the 0's in a string $w \in I^*$ have no effect on f_w. Thus, if $\bar{w}$ is w with all 0's removed, then $f_w = f_{\bar{w}}$.

Let l(w) denote the length of w, that is, the number of digits in w. Then $l(\bar{w})$ is the number of 1's in w, for all $w \in I^*$.

For each $n \geq 0$, consider the statement

P(n): *Let $w \in I^*$ and let $l(\bar{w}) = m$.*

(a) If $m = 3n$, then $f_w(s_0) = s_0$.
(b) If $m = 3n + 1$, then $f_w(s_0) = s_1$.
(c) If $m = 3n + 2$, then $f_w(s_0) = s_2$.

We prove by mathematical induction that P(n) is true for all $n \geq 0$. Suppose that n = 0. Then we consider the three cases as follows.

Case (a): m = 0, therefore, w has no 1's and $f_w(s_0) = 1_s(s_0) = s_0$.
Case (b): m = 1, so $\bar{w} = 1$ and $f_w(s_0) = f_{\bar{w}}(s_0) = f_1(s_0) = s_1$
Case (c): m = 2, so $\bar{w} = 11$, and $f_w(s_0) = f_{\bar{w}}(s_0) = f_{11}(s_0) = f_1(s_1) = s_2$.

Next, we must use P(k) to show $P(k+1)$. Let $w \in I^*$, and denote $l(\bar{w})$ by m. Let us first consider case (a). Here, $m = 3(k+1) = 3k+3$. This implies $\bar{w} = w'.111$, where $l(w') = 3k$. It follows that $f_{\bar{w}}(s_0) = f_{w'}(f_{111}(s_0)) = f_{w'}(s_0) = s_0$. Similarly, we solve for case (b) and (c). it follows that $P(k+1)$ is true.

Hence, by the principle of mathematical induction, P(n) is true for all $n \geq 0$, so $f_w(s_0) = s_0$ if and only if the number of 1's in w is a multiple of 3.

1.3 Simplification of Machines

Let's start this section by discussing the simplification of Moore machines. First we examine a fundamental theorem regarding the relation on Moore machines.

Let $M = (S, I, F, s_0, T)$ be a Moore machine. We define a relation R on S as follows: For any $s, t \in S$ and $x \in I^*$, we say that s and t are ***w-compatible*** if $f_w(s)$ and $f_w(t)$ both belong to T, or neither does. Also, let sRt imply that s and t are *w-compatible* for all $w \in I^*$.

Theorem 1.3.1: *Let $M = (S, I, F, s_0, T)$ be a Moore machine and let R be the relation defined previously then:*
(i) R is an equivalence relation on S
(ii) R is a machine congruence

Proof. (i) R is clearly symmetric and reflexive. Suppose now that s R t and t R u for s,t and u in s and let $w \in I^*$. Then s and t are w compatible as are t and u, so if we consider $f_w(s)$, $f_w(t)$ and $f_w(u)$ it follows that either all belong to T or all belong to $\bar{T}$, the complement of T. Thus s and u are w-compatible and so R is transitive. Hence R is an equivalence relation.

(ii) We must show that if s and t are in S and $x \in I$ then s R t implies that $f_x(s)$ R $f_x(t)$. To show this let $w \in I^*$ and let $w^{'} = x \cdot w$ where the dot represents the operation of catenation. Since s R t, $f^{'}_w(s)$ and $f^{'}_w(t)$ are both in T or both in $\bar{T}$. But $f^{'}_w(s) = f_{x \cdot w}(s) = f_w(f_x(s))$ and $f^{'}_w(t) = f_{x \cdot w}(t) = f_w(f_x(t))$ so this clearly implies that $f_x(s)$ and $f_x(t)$ are w- compatible. Since w is arbitrary in I^*, $f_x(s)$ R $f_x(t)$. □

Example 1.3.2. Let $M = (S, I, F, s_0, T)$ be a Moore machine where, $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, and $T = \{s_2, s_3\}$. We will find the quotient machine $\bar{M}$.
First, we see that $s_0 R s_1$. Note that $f_w(s_0) \in T$ and $f_w(s_1) \in T$ if and only if w contains at least one 1. Thus, s_0 and s_1 are w-compatible for all $w \in I^*$. Now s_2 is not related to s_0 and s_3 is not related to s_0, since $f_0(s_2) \in T$, $f_0(s_3) \in T$ but $f_w(s_0)$ does not belong to T. This implies that $\{s_0, s_1\}$ is one R-equivalence class. Also, $s_2 R s_3$, since $f_w(s_2) \in T$ and $f_w(s_3) \in T$ for all $w \in I^*$. This proves that $S/R = ((s_0, s_1), (s_2, s_3)) = ([s_0], [s_2])$. Also note that $T/R = ([s_2])$. The

resulting quotient Moore machine $\bar{M}$ is equivalent to M and its diagraph is shown in the Figure below.

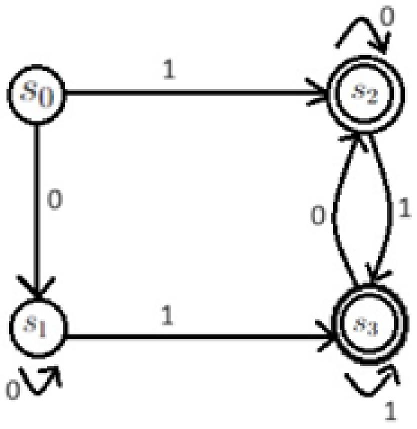

Figure 4: Moore Machine M

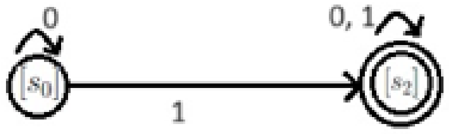

Figure 5: Quotient Moore machine $\bar{M}$

Given that R is a machine congruence, the quotient Moore machine can be constructed as $\bar{M} = (S/R, I, \bar{F}, s_0, T/R)$. Here the Machine $\bar{M}$ is the efficient version of M and we shall demonstrate, using the subsequent theorem that $L(\bar{M}) = L(M)$

Theorem 1.3.3. Let $M = (S, I, F, s_0, T)$ be a Moore machine, let R be the equivalence relation defined previously and let $\bar{M} = (S/R, I, \bar{F}, s_0, T/R)$ be the corresponding quotient machine. Then $L(\bar{M}) = L(M)$.

Proof. Let us suppose that W is accepted by M so that $f_w(s_0) \in T$. Then $f_w[\bar{(s_0)}] = [f_w(s_0)] \in T/R$ which implies $\bar{M}$ also accepts W. Now let us conversely suppose that $\bar{M}$ accepts W so that $f_w[\bar(s_0)] = [f_w(s_0)]$ is in T/R. This means that t R $f_w(s_0)$ for some element of t in T. By definition we know that t and $f_w(s_0)$ are $w^{'}$ compatible for every $w^{'} \in I^*$. When $w^{'}$ is Λ, the empty string, then $f^{'}_w = 1_s$, so $t = f^{'}_w(t)$ and $f_w(s_0) = f^{'}_w(f_w(s_0))$ are both in T or both in $\bar{T}$. Since $t \in T$, we must have $f_w(s_0) \in T$, so M accepts w. □

Theorem 1.3.4: (a) $R_{k+1} \subseteq R_k$ for all $k \geq 0$.
(b) $R \subseteq R_k$ for all $k \geq 0$.

Proof. If s,t $\in S$ and s and t are w-compatible for all $w \in I^*$ or for all w with $l(w) \leq k+1$, then in either case s and t are compatible for all w with $l(w) \leq k$. This proves both part (a) and (b). □

The subsequent two theorems aid us in formulating an algorithm for calculating relation R.

Theorem 1.3.5: *(a) $S/R_0 = \{T, \bar{T}\}$ where $\bar{T}$ is the complement of T. (b) Let K be a non negative integer and $s, t \in S$. Then s R_{k+1} t if and only if:*
(1) s R_k t
(2) $f_x(s)$ R_k $f_x(t)$ for all $x \in I$.

Proof. (a) Since only Λ has length 0 it follows that sR_0t if and only if both s and t re in T or both are in T^-. This proves that $S/R_0 = T, \bar{T}$.

(b) Let $w \in I^*$ be such that $l(w) \leq k+1$. Then $w = x \cdot w^{'}$ for some $x \in I$ and for some $w^{'} \in I^*$ with $l(w) \leq k$. Conversely if any $x \in I$ and $w^{'} \in I^*$ with $l(w') \leq k$ are chosen, the resulting string $w = x \cdot w^{'}$ has length less than or equal to $k+1$. Now $f_w(s) = f_{x.w'}(s) = f_{w'}(f_x(s))$ and $f_w(t) = f_{x.w'}(t) = f_{w'}(f_x(t))$ for any $s, t \in S$. This shows that s and t are w-compatible for any $w \in I^*$ with $l(w) \leq k+1$ if and only if $f_x(s)$ and $f_x(t)$ are, for all $x \in I$, w'-compatible, for any w' with $l(w') \leq k$. That is, $sR_{k+1}t$ if and only if $f_x(s)$ R_k $f_x(t)$ for all $x \in I$.

This result says that we may find the partition P_k, corresponding to the relations R_k by the following recursive method:

Step 1: Begin with $P_0 = \{T, \bar{T}\}$.
Step 2: Having reached partition $P_k = \{A_1, A_2, ...A_m\}$, examine each

equivalence class A_i and break it into pieces where two elements s and t of A_i fall into the same pieces if all inputs x take both s and t into same subset A_j depending on x.
Step 3: The new partition of S, obtained by taking all pieces of all the A_i, will be P_{k+1}.

Now, either of these equivalent conditions implies that sR_kt, since $R_{k+1} \subseteq R_k$, as desired. $\Box$

Theorem 1.3.6: *If $R_k = R_{k+1}$ for any non negative k, then $R_k = R$.*

Proof. Suppose that $R_k = R_{k+1}$, then by above theorem $s\ R_{k+2}\ t$ if and only if $f_x(s)\ R_k\ f_x(t)$ for all $x \in I$. This happens if and only if
$s\ R_{k+1}\ t$. Thus $R_{k+2} = R_{k+1} = R_k$. By induction it follows that $R_k = R_n$ for all n. Now it is easy to see that $R = \bigcap R_n$, since every string w in I^* must have some finite length. Since $R_{k+1} \subseteq R_k \subseteq R_{k-1} \subseteq .....R_2 \subseteq R_1$. Hence this implies that the intersection of R_n's is exactly R_k so $R = R_k$.$\Box$

A procedure for reducing a given Moore machine to an equivalent machine is as follows:

Step 1: Start with partition $P_0 = \{T, \bar{T}\}$.
Step 2: Construct successive partitions $P_1, P_2, ...$ corresponding to the equivalence relation $R_1, R_2, ...$ by using the method outlined in Theorem 1.3.7.
Step 3: Whenever $P_k = P_{k+1}$ we need to stop. The resulting partition $P = P_k$ corresponds to relation R.
Step 4: The resulting quotient machine is equivalent to the given Moore machine.

Example 1.3.7. Let us now reconsider machine M from Example 1.3.2.

First, the partition $P_0 = \{\{s_0, s_1\}, \{s_2, s_3\}\}$. To find P_1, let us decompose the P_0. Consider the set (s_0, s_1). Input 0 takes both the states into $\{s_0, s_1\}$ and input 1 takes them into $\{s_2, s_3\}$. This implies that the equivalence class $\{s_0, s_1\}$ does not decompose in passing to P_1. Next, Consider the set $\{s_2, s_3\}$. We see that both input 0 and 1 takes s_2 and s_3 into $\{s_2, s_3\}$. So, this implies that the equivalence class $\{s_2, s_3\}$ does not decompose in passing to P_1. It follows that $P_1 = P_0$ so P_0 corresponds to the congruence R.

2 Finite-state Automata with Output

2.1 Introduction to Mealy Machines

Now, let us examine another variant of finite state machine known as a Mealy machine. The Mealy machine is named after George H. Mealy, who introduced the concept in a paper published in 1955. This machine is a seven-tuple machine, as opposed to a Moore machine, which typically consists of five tuples. We define the Mealy machine as follows:

Definition 2.1.1 A finite state Mealy machine is a seven tuple $M = (S, I, \theta, F, G, s_o, T)$ where

(i) S is the finite nonempty set of states
(ii) I is the finite nonempty set of input symbols
(iii) θ is the finite nonempty set of output symbols
(iv) F is the state-transition function whose mapping is $F : S \times I \to S$
(v) G is the output function whose mapping is G: $S \times I \to \theta$

Now, let's examine the important characteristics associated with state transition and output function which are provided proof free.

Definition 2.1.2. Let $M = (S, I, \theta, F, G, s_o, T)$ be a finite state Mealy machine. For each $s \in S$, $x \in I^*$ and $a \in I$,

$F(s, \Lambda) = s$
$F(s, xa) = F(F(s, x), a)$
$G(s, \Lambda) = s$
$G(s, xa) = F(s, x)G(F(s, x), a)$ and
$\hat{G}(s, xa) = G(F(s, x), a)$

Mealy machines are utilized for a variety of purposes, including: a rudimentary mathematical paradigm for cipher machines is provided by them. When the input and output alphabets are considered, such as the Latin alphabet, one can design a Mealy machine capable of converting a series of letters (an input sequence) into a ciphered output sequence. Despite the fact that it could be employed to depict the Enigma, the intricacy of the state diagram would render it impracticable to devise practical methods for developing intricate ciphering devices. It is possible to model straightforward software systems, especially those that are re-presentable via regular expressions such as (i) Vending machines, (ii) Bar code scanner, (iii) watches with timer, etc.

Let us now understand the following theorem which highlights that when two machines are in the same state and receive the exact same input, they will consistently produce the same output and transition to the identical state.

Theorem 2.1.3. *If two Mealy machine M_1 and M_2 have same state and input then it results in*
(i) Same output
(ii) Same state.

Proof. (i) Let us assume to the the contrary that M_1 and M_2 have different outputs. So, $g_1(s,x) = \theta_1$ and $g_2(s,x) = \theta_2$ but we know that $g : s \times x \to \theta$. It follows that $g_1(s,x) = s \times x = \theta = s \times x = g_2(s,x)$ implies $\theta_1 = \theta = \theta_2$ which is a contradiction.
(ii) Let us assume to the contrary that M_1 and M_2 have different states. So, $F_1(s,x) = s_1$ and $F_2(s,x) = s_2$ but we know that $F : s \times x \to s'$. It follows that $F_1(s,x) = s \times x = s' = s \times x = F_2(s,x)$ implies $s_1 = s' = s_2$ which is a contradiction. $\square$

Example 2.1.4. Let $M =< S, I, \theta, F, G, s_0, T >$ where $S = \{s_0, s_1, s_2\}$ and $I = \theta = \{0,1\}$. The following two tables showcase the F- function and G- function.

	0	1
s_0	s_0	s_0
s_1	s_1	s_1
s_2	s_1	s_0

Table 5. F-function for Machine M

	0	1
s_0	0	1
s_1	0	0
s_2	0	1

Table 6. G-function for Machine M

Let us now begin by defining input-output relations of a machine using an example. Following that, we will demonstrate under certain conditions how state transition functions of two machines are equivalent.

Definition 2.1.5. Let $M =< S, I, \theta, F, G, s_0, T >$ be a finite state machine. Define $F(M) = G_s \mid s \in S$ and $R(M) = (x, G(s, x)) \mid x \in I^*, s \in S$ F(M) is set of functions of m, while R(M) is input-output relation of M.

Definition 2.1.6. Let $M_i =< S, I, \theta, F, G, s_0, T >$ be finite state machines. States $s_i \in S_i$, $i = 1, 2$ are said to be equivalent if $G_1(s_1, x) = G_2(s_2, x)$ for each $x \in I^*$. We write $s_1 \equiv s_2$.

Theorem 2.1.7. Let $M_i =< S, I, \theta, F, G, s_0, T >$ for $i = 1, 2$ be finite state machines. If $s_i \in S_i$, $i = 1, 2$ and $s_1 \equiv s_2$ then $F_1(s_1, x) \equiv F_2(s_2, x)$ for each $a \in I$.

Proof. Let $x \in I^*$, then

$$G_1(F_1(s_1, a), x) = \hat{G}_1(s_1, ax) = \hat{G}_2(s_2, ax) = G_2(F_2(s_2, a), x).$$

Therefore, $F_1(s_1, x) \equiv F_2(s_2, x)$. $\square$

2.2 Structure of Finite State Machines

This section will examine the structure of finite state machines. To start, we will commence by defining a minimal machine and examining its significant results with illustrations. Subsequently, we will delve into the concept of machines in arrangement.

Definition 2.2.1. Let $M =< S, I, \theta, F, G, s_0, T >$ be a finite state machine. M is said to be minimal if $s_1 \equiv s_2$ implies $s_1 = s_2$ for each $s_1, s_2 \in S$.

Definition 2.2.2. Let $M_i =< S, I, \theta, F, G, s_0, T >$ be finite state machines. K is said to be homomorphism from M_1 into M_2 if K is a map from S_1 onto S_2 such that
$K(F_1(s, a) = F_2(Ks, a)$ and $G_1(s, a) = G_2(Ks, a)$ for each $(s, a) \in S_1 * I$.

Theorem 2.2.3. For each finite state machine $M_1 =< S_1, I, \theta, F_1, G_1, s_0, T_1 >$ there is a minimal machine M such that $M_1 \equiv M$.

Proof. Let $M =< S, I, \theta, F, G, s_0, T >$ where M is constructed from M_1 by taking $S = \{[s] \mid s \in S_1\}$ where [s] is the equivalence classes of the relation $\equiv$ which contains S. It follows that $F([s], a) = [F_1(s, a)]$ and $G([s], a) = [G_1(s, a)]$. M which is really $M_1/\equiv$ is well defined. For $S_1 \equiv S_2$ implies $F_1(s_1, a) \equiv F_2(s_2, a)$ so that $F_1([s_1], a) = F_2([s_2], a)$. So, $G([s_1], a) = G([s_2], a)$. Clearly, M is minimal if $[s] \equiv [s']$ then $G([s], x) = G([s'], x)$ implies $G_1(s, x) = G_1(s', x)$. So, $s \equiv s'$ or $[s] = [s']$. Since M is clearly equivalent to M_1, implies there is a minimal machine M such that $M_1 \equiv M$. $\square$

Now, let us define a map K called an assignment and then we will understand it with Example 2.2.5.

Definition 2.2.4. Let $M_i =< S, I, \theta, F, G, s_0, T >$ for $i = 1, 2$ be finite state machines. M_2 is a state realization of M_1 if there is a one to one map K from Q_1 into Q_2 such that $K(F_1(s, a) = F_2(Ks, a)$ for each $(s, a) \in S_1 * I$. K is called an assignment.

Example 2.2.5. Let M_1 and M_2 be defined below.

	0	0	1	1
s_0	s_2	0	s_1	0
s_1	s_0	1	s_1	1
s_2	s_2	1	s_0	1

Table 7. Machine M_1

	0	0	1	1
s_0	s_2	0	s_1	0
s_1	s_0	1	s_1	1
s_2	s_2	1	s_0	0

Table 8. Machine M_2

s	Ks
s_0	$(0, 0)$
s_1	$(1, 1)$
s_2	$(0, 1)$

Table 9. Assignment K

	0	0	1	1
s_0	$(0,1)$	0	$(1,1)$	0
s_1	$(0,0)$	1	$(1,1)$	1
s_2	$(0,1)$	1	$(0,0)$	0

Table 10. Machine M_2

It is possible to establish a network of interconnected machines in a configuration that exhibits intriguing properties. Let us begin by first defining the arrangement of machines coupled in a parallel composition and then in serial composition.

Definition 2.2.6. Let $M_i =< S, I, \theta, F, G, s_0, T >$ for $i = 1, 2$ be finite state machines. The parallel connection of M_1 and M_2 is defined to be $M_1 \mid\mid M_2 =< S_1 * S_2, I, \theta_1 * \theta_2, F, G >$ where $F((s_1, s_2), a) = (F_1(s_1, a), F_2(s_2, a))$ and $G((s_1, s_2), a) = (G_1(s_1, a), G_2(s_2, a))$ for each $(s_1, s_2, a) \in S_1 * S_2 * I$.

Now, in the figure shown below, we observe that two machines M_1 and M_2 are arranged in parallel combination. Machine M_1 is in state s_1 and machine M_2 is in state s_2. They receive input a and so their state transition function is given by $F((s_1, s_2), a) = (F_1(s_1, a), F_2(s_2, a))$. Similarly, the output function is $G((s_1, s_2), a) = (G_1(s_1, a), G_2(s_2, a))$ for each $(s_1, s_2, a) \in S_1 * S_2 * I$.

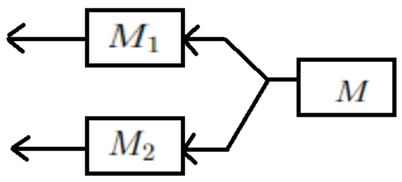

Figure 6: Machines in Parallel Connection

Definition 2.2.7. Let $M_i =< S, I, \theta, F, G, s_0, T >$ for $i = 1, 2$ be finite state machines. The serial connection is $M_2 \leftarrow M_1 =< S_2 * S_1, I_1, \theta_2, F, G >$ where $F((s', s), a) = (F_2(s', G_1(s, a)), F_1(s, a))$ and $G((s', s), a) = G_2(s', G_1(s, a))$.

Now, in the figure shown below, we observe that two machines M_1 and M_2 are arranged in serial combination. Machine M_1 is in state s_1 and machine M_2 is in state s_2. Here, we note that the output from machine M_1 is actually the input for machine M_2. So, under an initial input a their state transition function is given by $F((s', s), a) = (F_2(s', G_1(s, a)), F_1(s, a))$. Similarly, the output function is $G((s', s), a) = G_2(s', G_1(s, a))$.

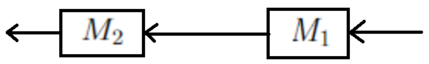

Figure 7: Machines in Serial Connection

3 Linear Finite State Machines

3.1 Monoid of Finite State Machines

In this section, we will first provide a description of the submachine $M^{'}$ that is part of a given machine M. We will next examine the monoid of M and discuss significant findings regarding the relationship between isomorphism and the monoid of a machine.

Definition 3.1.1. Let M = $< S, I, \theta, F, G, s_0, T >$ be a finite state machine. Let $M^{'} =< S^{'}, I, \theta, F^{'}, G^{'}, s_0^{'}, T^{'} >$ is a submachine of M if :
(i) $\varnothing \neq S^{'} \subseteq S$
(ii) For each $a \in I$, $F(S^{'}, a) = \{F(s^{'}, a) \mid s^{'} \in S'\} \subseteq S^{'}$
(iii) $F^{'} = F \cap (S^{'} \times I \times S^{'})$ and $G^{'} = G \cap (S^{'} \times I \times \Theta)$

Example 3.1.2. Let M be a finite state machine having $S = \{s_0, s_1\}$, $I = \{0\}$, $\theta = \{0, 1\}$. Then, $M^{'}$ is a submachine of M.

	0	0
s_0	s_0	0
s_1	s_0	1

Table 11. Machine M

	0	0
s_0	s_0	0

Table 12. Machine $M^{'}$

Let us commence by providing a precise definition of a monoid. We will first define it in terms of algebraic structures and then consider its interpretation from a machine perspective.

Definition 3.1.3. A monoid is a triple $< S, *, e >$ where S is a non void set and * is a binary operation on S which is associative i.e., a(bc) = (ab)c for all a,b,c ∈ S. Where e ∈ S is the identity so $ae = ea = a$ for all $a \in S$. As usual we have written ab for $a * b$. Let $M =< S, I, \theta, F, G, s_0, T >$ be a finite state machine. For each $x \in I^*$, we write x(M) as the mapping which takes s into F(s,x).

Definition 3.1.4. For each finite state machine M, the monoid of M, usually denoted by $G(M)$ is $G(M) = \{x(M) \mid x \in I^*\}$. We note that the monoid operation is composition of functions and that $\Lambda(M)$ is the identity mapping on S. Moreover $xy(M) = x(M)y(M)$ for all x,y $\in I^*$.

Example 3.1.5. Let M be the finite state machine having $S = \{s_0, s_1, s_2\}$, and $I = \{0, 1\}$.

	0	1
s_0	s_1	s_2
s_1	s_2	s_0
s_2	s_0	s_1

Table 13. Machine M

	$\Lambda(M)$	0(M)	1(M)
s_0	s_0	s_1	s_2
s_1	s_1	s_2	s_0
s_2	s_2	s_0	s_1

Table 14. G(M)

Definition 3.1.6. Let M and $M^{'}$ be finite state machines. M is said to be state isomorphic to $M^{'}$, if $M^{'}$ state realizes M via an assignment k which is also onto.

Let us now examine the following theorem, which will help us comprehend how state realization and submachine relate to each other.

Theorem 3.1.7. If $M_1 = < S_1, I, \theta, F_1, G_1, s_0, T_1 >$ and $M_2 = < S_2, I, \theta, F_2, G_2, s_0, T_2 >$ be finite state machines and M_2 is a state realization of M_1, then M_1 is state isomorphic to some submachine $M^{'}$ of M_2.

Proof. Since M_2 is a state realization of M_1, there is a one to one map k from S_1 into S_2 such that $kF_1(s, a) = F_2(ks, a)$. Let $S^{'} = kS_1 \subseteq S_2$, so let us consider $M^{'} = < S^{'}, I, \theta, F^{'}, G^{'}, s_0, T^{'} >$ where $F^{'} = F_2 \cap (S^{'} \times I \times S^{'})$. Clearly $M^{'}$ is a submachine of M_2. The map k is restricted to $S^{'}$ is still one to one and is now onto. Hence it is a state isomorphism. $\square$

The next two theorems will elucidate the relationship between state isomorphism and the monoid of a machine. The initial one is presented here without proof, however it will assist us in demonstrating theorem 2.3.9.

Theorem 3.1.8. If $M_1 = < S_1, I, \theta, F_1, G_1, s_0, T_1 >$ and $M_2 = < S_2, I, \theta, F_2, G_2, s_0, T_2 >$ be finite state machines. If $M^{'}$ is a submachine of M_2 and M_1 is state isomorphic to $M^{'}$, then $G(M_1)$ is isomorphic to submonoid of $G(M_2)$.

Theorem 3.1.9. Let $M_1 = < S_1, I, \theta, F_1, G_1, s_0, T_1 >$ and $M_2 = < S_2, I, \theta, F_2, G_2, s_0, T_2 >$ be finite state machines. If M_2 is a state realization of M_1 then $G(M_1)$ is isomorphic to submonoid of $G(M_2)$.

Proof. Since M_2 is a state realization of M_1, M_1 is state isomorphic to a submachine $M^{'}$ of M_2 by theorem 3.1.7. By the theorem 3.1.8 $G(M_1)$ is isomorphic to a submonoid of $G(M_2)$. $\square$

3.2 Properties of Linear Finite State Machines

This section is an introduction to a linear finite state machine, along with a discussion of key properties associated with them. In the language of system theory they are linear, finite-dimensional, time invariant systems From an intuitive perspective, these machines can be understood as finite state machines in which their state transition and output functions can be represented as linear functions. The following is the definition of a linear finite state machine:

Definition 3.2.1. A linear finite state machine (LSM for short) is a finite state machine $M = (S, I, \theta, F, G, s_o, T)$ with the following special properties. There exists a field F and non-negative integers n, k and l such that $S = F_n, I = F_k$ and $\theta = F_l$. Furthermore there exists an $n \times n$ matrix **A** over F, an $l \times k$ matrix **B** over F, an $l \times n$ matrix **C** over F and an $l \times k$ matrix **D** over F such that for each $(s, a) \in S \times I$
F(s,a) = As + Ba
G(s,a) = Cs + Da.

Now that we have an understanding of what the state transition function and output function for a linear finite state machine is, we exhibit the following three important results.

Theorem 3.2.2. *Let M be an LSM. For each $s \in F_n$ and $x_0, ..., x_{t-1} \in F_k, t > 0$ then*

$$F(s, x_0, ..., x_{t-1}) = A^t s + \sum_{i=0}^{t-1} A^{t-1-i} B x_i.$$

Proof. Let p(t) $= F(s, x_0, ..., x_{t-1}) = A^t s + \sum_{i=0}^{t-1} A^{t-1-i} B x_i$. We induct on the length of $w = x_0, ..., x_{t-1}$. Since

$$F(s, x_0) = A^1 s + A^0 B x_0 = As + Bx_0$$

implies the statement $p(1)$ true. Now we assume that $p(t)$ is true for any arbitrary positive integer k such that

$$F(s, x_0, ..., x_{k-1}) = A^t s + \sum_{i=0}^{k-1} A^{k-1-i} B x_i.$$

Now, for $p(k+1)$ we have

$$F(s, x_0, ..., x_k) = F(F(s, x_0, ..., x_{k-1}), x_k) \implies$$
$$F(A^t s + \sum_{i=0}^{k-1} A^{k-1-i} B x_i), x_k) \implies$$
$$A(A^t s + \sum_{i=0}^{k-1} A^{k-1-i} B x_i) + B x_k \implies A^{k+1} s + \sum_{i=0}^{k-1} A^{k-1-i} A B x_i + B x_k.$$

Hence, by the principle of mathematical induction, $p(t)$ is true for any positive integer k. □

Theorem 3.2.3. *Let M be an LSM. For each $s \in F_n$ and $x_0, ..., x_{t-1} \in F_k$, then*

$$\hat{G}(s, x_0, ..., x_{t-1}) = CA^{t-1} s + \sum_{i=0}^{t-2} CA^{t-i-2} B x_i + D x_{t-1}.$$

Proof. Consider

$$p(t) = \hat{G}(s, x_0, ..., x_1) = CA^{2-1} s + CA^0 B x_0 + D x_1 = CAs + CBx_0 + Dx_1.$$

This implies that the statement p(t) is true for $t = 2$. Now, suppose $p(t)$ is true for any arbitrary $t \geq 2$ such that

$$\hat{G}(s, x_0, ..., x_{t-1}) = CA^{t-1} s + \sum_{i=0}^{t-2} CA^{t-i-2} B x_i + D x_{t-1}.$$

Then for $p(t+1)$ we have,

$$\hat{G}(s, x_0, ..., x_t) = CA^t s + \sum_{i=0}^{t-1} CA^{t-i-1} B x_i + D x_t.$$

Hence by the principle of mathematical induction, $p(t)$ is true for all $t \geq 2$. □

Furthermore, we may apply the aforementioned outcome to a number of linear finite state machines arranged in series.

Theorem 3.2.4. *Let $M_1, M_2, ..., M_n$ be linear finite state machines arranged in a serial connection (as shown in Figure below). Then*

$$G(S_n, I_n) = CS_n + \textstyle\sum_{i=1}^{n-1} D^i CS_{n-i} + D^n I_1.$$

Proof. We proceed by using mathematical induction. For $n = 2$ we get,

$$G(S_2, I_2) = G(S_2, \theta_1) = G(S_2, CS_1 + DI_1) = CS_2 + D(CS_1 + DI_1) = \\ CS_2 + DCS_1 + D^2 I_1.$$

Hence we find that the statement holds true for n = 2. Let us assume that for $n = k$ it holds true such that

$$G(S_k, I_k) = CS_k + \textstyle\sum_{i=1}^{k-1} D^i CS_{k-i} + D^k I_1.$$

Then for $n = k + 1$ we have,

$$G(S_{k+1}, I_{k+1}) = G(S_{k+1}, \theta_k) = G(S_{k+1}, CS_k + \textstyle\sum_{i=1}^{k-1} D^i CS_{k-i} + D^k I_1) =$$

$CS_{k+1} + DCS_k + D \sum_{i=1}^{k-1} D^i CS_{k-i} + D^k I_1 =$
$CS_{k+1} + D^k I_1 + DCS_k + [D^2 CS_{k-1} + D^3 CS_{k-2} + ... + D^k CS_1] =$
$CS_{k+1} + D^k I_1 + \sum_{i=1}^{k} D^i CS_{k+1-i}.$
Hence by the principle of mathematical induction, the formula holds true for all $n \geq 2$. $\square$

4 References

[1] B. Kolman, R. Busby, S. Ross, Discrete Mathematical Structures. Pearson Prentice Hall, N.J. (2004).

[2] M. Harrison, Lectures on Linear finite state Machines, Academic Press, New York and London, (1962).

[3] H. Gallaire, M. Harrison, Decomposition of linear finite state machines. Math. Systems Theory, Vol. 3 (1969).

[4] A. Gill, Introduction to the Theory of Finite State Machines. McGraw-Hill, New York, (1962).

[5] A. Ginzburg, Algebraic Theory of Automata. Academic press, New York, (1968).

[6] M. Harrison, Introduction to Switching and Automata Theory, McGraw-Hill, New York, (1962).