Theory of Finite Automata

According to the Oxford English Dictionary, an automaton (plural is "automata") is a "machine whose responses to all possible inputs are specified in advance". For this course, the term "finite automaton" can be viewed as a synonym for "finite state machine". The term "FSM" or "Finite State Machine" tends to be the preferred term when one is studying computer organization and architecture, while the term "Finite Automaton" tends to be used more by those studying topics in the theory of computation, such as language recognition.

The previous chapter approached finite automata with a basic descriptive style. Two styles of finite automata were introduced by example and discussed. This chapter returns to the subject and reinterprets some of the earlier results in a more theoretical light. This chapter could have been entitled "Theory of Finite State Machines", but this author has elected to use the fancier language to indicate a shift of focus to a more theoretical approach.

What is a Computer?

We might as well begin with the basic motivation for theoretical computer scientists to study finite automata: the question "What is a computer?". At one level, this is a silly question, because we all know a computer when we see it. Put this way, a computer is defined quite simply by the statement "A computer is what it is.". Lest we think that such a definition is trivial, the reader is invited to think of the number of items we all accept as defined by example; we define it by the statement "Look – this is one".

All definitions are circular and ultimately depend on the use of examples. The reader is invited to examine the definition of any word in a dictionary and follow the trail of words used in the definition; eventually it will return to the original word. This author remembers having been required, as a part of an introductory course in Physics, to memorize the definition of the word "mass", along with Newton's three laws of motion. Basically the definition was "mass is a property that objects with mass have" – quite circular.

Such obvious and intuitive definitions of a computer do not suffice for the theoretician who wants everything to be precise and logical. Such a person wants to ask questions such as "What is computable?" and is not satisfied with the answer "It is something that can be computed by this specific computer.". Theoreticians prefer to study **models of computation**, which are formal models used in thinking about the processes that our modern computers have automated. Many of the models of computation, such as finite automata, do resemble actual physical computers and draw their inspiration from observation of such machines. Nevertheless they are **formal mathematical models** of computation, with definitions arising logically from the basic principles of mathematics, including set theory and Boolean algebra. Formal models are elegant and quite interesting (to some of us), but suffer the downfall of existing apart from the real world and possibly not connecting to any real object or problem.

Fortunately, the formal models of computation have found application to many important problems found in the real world, one example being the modern compiler used to process high-level languages such as C++. It is for this reason that we should study these models.

An Example Automaton: the 11011 Sequence Detector

We begin our formal discussion of finite automata with an example discussed in the previous chapter. This example is the 11011 sequence detector, which we now discuss using the terms and state diagrams of finite automata theory.

Here is the state diagram of the 11011 sequence detector drawn as a finite automaton.

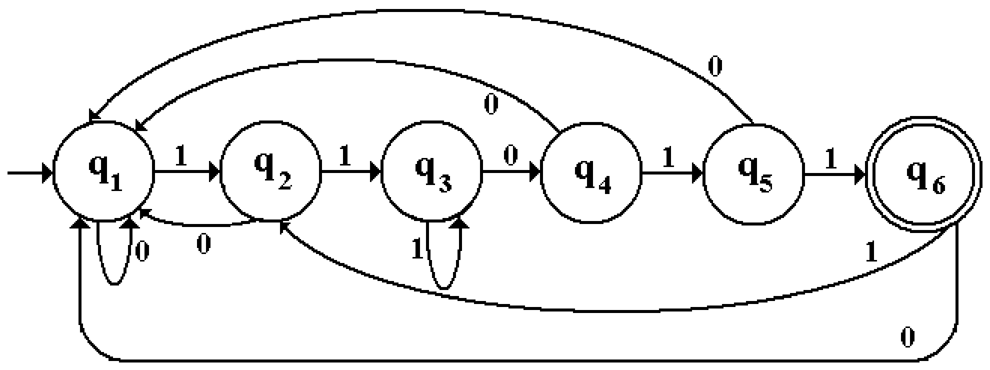

Figure: Finite Automaton for Detecting 11011 Sequence with no Overlap

The reader will note that the figure is quite different from that previously drawn for the sequence detector. We comment on a few of the more important features of this FA.

1) The states are labeled as q_1, q_2, q_3, q_4, q_5, and q_6. Theoreticians who work with finite automata prefer to label the states with the lower-case "q", beginning at 1 and running to the number of states.
2) The output of this automaton is not associated with the transitions, but with the states. In this, the automaton more resembles the counters we discussed earlier, than it does the sequence detector. We shall later note that some finite automata models do have output associated with the transitions, but they are not so common.
3) There are six states associated with the automaton, not the five as developed earlier. This is a result of the output being associated with the states.
4) The state q_6 has a double circle around it. This marks it as an **accept state**, also called a **final state**. Finite automata may have any number of accept states, including zero (for no accept states).
5) The FA has a **start state**, q_1, that is indicated by having an arrow pointing at it from nowhere. The FA starts here.
6) The FA takes a string, one character at a time, as input and outputs either "**accept**" or "**reject**". If the FA ends in an accept state at the end of processing the string, the FA accepts the string; otherwise it rejects it. The term "detector" seems not to be used when discussing finite automata.
7) There is no concept of a real clock, such as we used in our counters and sequence detectors in the previous chapter. These FA are mathematical models.
8) This finite automata is said to accept any string terminating in 11011, rather than detecting the sequence 11011. We now call this an "**11011 acceptor**".

Terms Used in Discussing Finite Automata

Formally, a finite automaton is described by a five elements, described as a 5-tuple. Before presenting the definition in formal mathematical fashion, we shall discuss each of the elements of a finite automaton as seen in the 11011 acceptor.

Q denotes a finite set of states.
In the 11011 acceptor, as defined above, we see that $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$.

Σ denotes a finite set, called the alphabet, of symbols that the FA is expected to read.
For this FA, we have $\Sigma = \{0, 1\}$, as the strings presented to the FA will consist only of those symbols. For a recognizer of English words, Σ would be the alphabet as we know it.

δ denotes the transition function.
Previously we have used either a state table or a transition table to represent the transition function. Before discussing other options, we present the state table.

δ	0	1
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_4	q_3
q_4	q_1	q_5
q_5	q_1	q_6
q_6	q_1	q_2

The same definition for the function can be presented in another way, in which the function δ is seen as taking two arguments, one a member of the set Q and one a member of the set Σ. Thus we have the function as specified by the following list.

$$\begin{array}{ll}
\delta(q_1, 0) = q_1 & \delta(q_1, 1) = q_2 \\
\delta(q_2, 0) = q_1 & \delta(q_2, 1) = q_3 \\
\delta(q_3, 0) = q_4 & \delta(q_3, 1) = q_3 \\
\delta(q_4, 0) = q_1 & \delta(q_4, 1) = q_5 \\
\delta(q_5, 0) = q_1 & \delta(q_5, 1) = q_6 \\
\delta(q_6, 0) = q_1 & \delta(q_1, 1) = q_2
\end{array}$$

We present this listing of the function in order to clarify notation commonly used when describing the transition function; mathematicians would say $\delta: Q \times \Sigma \rightarrow Q$.

The set of all possible inputs to a function is called the **domain**, often denoted as “D”. The outputs of a function come from a set, called the **range** and often denoted as “R”. Thus, we say that $\delta: D \rightarrow R$, with domain $D = Q \times \Sigma$ and range $R = Q$.

The statement that the range of the function δ is R = Q should be easy to understand. What we are saying is that the output of the function is one of the elements of the set Q.

We now turn our attention to the statement that the domain of the function δ is the set $Q \times \Sigma$. This is the **Cartesian product** of the two sets Q and Σ. For any two sets A and B, the Cartesian product of A and B, denoted $A \times B$, is the set of all pairs in which the first element comes from set A and the second element comes from set B.

Thus, we have $Q \times \Sigma = \{ (q, s) \mid q \in Q \text{ and } s \in \Sigma\}$. For the above example, we have.

$Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$
$\Sigma = \{0, 1\}$

$$Q \times \Sigma = \{ \begin{array}{l} (q_1, 0), (q_2, 0), (q_3, 0), (q_4, 0), (q_5, 0), (q_6, 0), \\ (q_1, 1), (q_2, 1), (q_3, 1), (q_4, 1), (q_5, 1), (q_6, 1) \} \end{array}$$

Since $|Q| = 6$ and $|\Sigma| = 2$, we have $|Q \times \Sigma| = 12$.

$q_1 \in Q$ denotes the start state.
Not much more to say here. In our example, we use q_1 as the start state.

$F \subseteq Q$ is the set of accept states or final states.
For our example, we have $F = \{q_6\}$, the set with one element, which is the only accept state.

Formal Definition of a Finite Automaton

A **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_1, F)$, where
1. Q is a finite non-empty set called the set of **states**,
2. Σ is a finite set called the **input alphabet**,
3. $\delta: Q \times \Sigma \to Q$ is the **transition function**,
4. $q_1 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the set of **accept states** or **final states.**

When a finite automaton receives an input string, it processes that string one symbol at a time (reading left to right) and produces an output. The automaton begins in the start state and moves through the states in response to the input characters as dictated by the transition function. When the last symbol of the string is read, the finite automaton produces its output: if the FA is in an accept state, the output is “accept”, otherwise the output is “reject”.

If L is the set of all strings that the FA accepts, we say that L is the **language recognized** by the FA, alternately saying that the FA **recognizes the language** L. Occasionally it is said that the FA accepts the language, but this is considered confusing. We prefer to say that a finite automaton **accepts a string** and **recognizes a language**.

Finite Automaton M_1

We now consider the following example of a finite automaton, analyzing it to determine what it does – what strings does it accept or, equivalently, what language does it recognize.

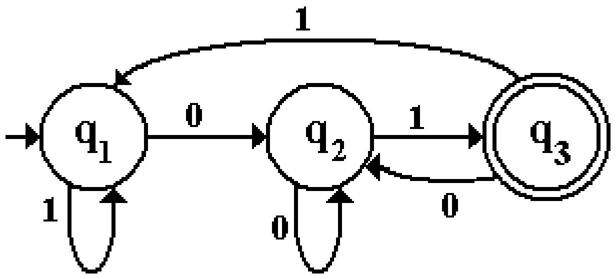

Figure: Finite Automaton M_1 for Analysis

At this point, we should make a disclaimer. The above example is the second finite automaton that has been seen in this chapter, and both FA have been drawn in a straight line with the one and only accept state being at the right of the figure. The next few FA to be considered will show that neither property is essential to the definition. We now analyze the circuit by asking and answering a number of questions.

1. **How many states does M_1 have and what are they?**
The finite automaton M_1 has three states, labeled q_1, q_2, and q_3; since these are the labels found in the three circles. Thus, $Q = \{ q_1, q_2, q_3 \}$

2. **What is the alphabet (set of input symbols) for M_1?**
The input symbols for M_1 are 0 and 1, as these are the only labels found on the arrows that represent the transitions. Thus, $\Sigma = \{ 0, 1 \}$.

At this point, a purist might note that all we have shown is that $\{ q_1, q_2, q_3 \} \subseteq Q$ and that $\{ 0, 1 \} \subseteq \Sigma$. Nevertheless, we follow the rule "If you can see it, it is there; if you can't, it ain't", and stipulate that we have shown the entire FA, with Q and Σ complete.

3. **What is the start state for M_1?**
The start state is q_1, the state shown with an unlabeled input arrow coming from nowhere. The reader will note that state q_1 is drawn leftmost on the diagram, as is often the case, but this is just conventional and does not imply anything about q_1.

4. **What is the set of accepting (final) states for M_1?**
The only state marked with the double circle is q_3, so $F = \{q_3\}$.

5. **Describe the transition function.**

$\delta(q_1, 0) = q_2$	$\delta(q_1, 1) = q_1$
$\delta(q_2, 0) = q_2$	$\delta(q_2, 1) = q_3$
$\delta(q_3, 0) = q_2$	$\delta(q_3, 1) = q_1$

6. Describe the language recognized by M_1.

First note that the two symbol string "01" is accepted, as we start in state q_1 and have both $\delta(q_1, 0) = q_2$ and $\delta(q_2, 1) = q_3$, leading to q_3 – an accept state. To show that any string ending with the two bits "01" is accepted, we must show that applying "01" is input to any state will result in the FA being in the accept state.

At any given time, the FA must be in one of the three states: q_1, q_2, or q_3. We have already shown that an input of "01" will take the FA from state q_1 to q_3. We say it again.

Start at q_1: $\delta(q_1, 0) = q_2$ and $\delta(q_2, 1) = q_3$, so the FA ends in state q_3 – an accept state.
Start at q_2: $\delta(q_2, 0) = q_2$ and $\delta(q_2, 1) = q_3$, so the FA ends in state q_3 – an accept state.
Start at q_3: $\delta(q_3, 0) = q_2$ and $\delta(q_2, 1) = q_3$, so the FA ends in state q_3 – an accept state.

Having shown that the FA accepts strings ending in "01", we now show that it accepts only those strings. The state q_3 is the only accept state. The only transition that ends at state q_3 is the transition $\delta(q_2, 1) = q_3$, so any string that is accepted must end with a "1". In order for a string of length $N \geq 2$ to be accepted, the last symbol must be a "1" and the first $(N – 1)$ symbols must have taken the FA from state q_1 to state q_2. The only way for the FA to end in state q_2 is for the last symbol input to have been a "0", hence the last two symbols of any string to be accepted must have been "01".

The language recognized by the finite automaton M_1 is the set of strings ending in "01".

Finite Automaton M_2

Here is the state diagram of another finite automaton. What can we say about M_2?

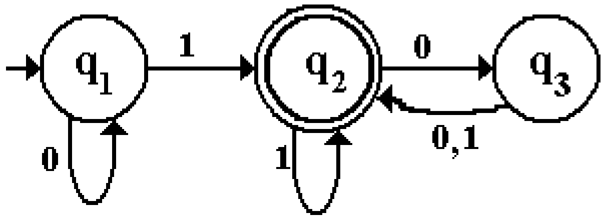

Figure: Finite Automaton M_2 for Analysis

Here we first remark that the accept state is now q_2 and, as promised, is not drawn as the rightmost state. We should also note that the arrow leaving state q_3 corresponds to two transitions; in state q_3 either a "0" or a "1" will cause a transition to state q_2.

1. How many states does M_2 have and what are they?

The finite automaton M_1 has three states, labeled q_1, q_2, and q_3; since these are the labels found in the three circles. Thus, $Q = \{ q_1, q_2, q_3 \}$

2. What is the alphabet (set of input symbols) for M_2?

The input symbols for M_1 are 0 and 1, as these are the only labels found on the arrows that represent the transitions. Thus, $\Sigma = \{ 0, 1 \}$.

3. What is the start state for M_2?

The start state is q_1, the state shown with an unlabeled input arrow coming from nowhere. The reader will note that state q_1 is drawn leftmost on the diagram, as is often the case, but this is just conventional and does not imply anything about q_1.

4. What is the set of accepting (final) states for M_1?

The only state marked with the double circle is q_2, so $F = \{q_2\}$.

5. Describe the transition function.

$\delta(q_1, 0) = q_1$	$\delta(q_1, 1) = q_2$
$\delta(q_2, 0) = q_3$	$\delta(q_2, 1) = q_2$
$\delta(q_3, 0) = q_2$	$\delta(q_3, 1) = q_2$

6. Describe the language recognized by M_2.

First note that the one symbol string "1" is accepted, as we start in state q_1 and have $\delta(q_1, 1) = q_2$, leading to an accept state. It should be immediately obvious that any string ending in a "1" will be accepted. To show this, we again note that having processed (N – 1) of the N bits of a string, the finite automaton must be in one of the three states q_1, q_2, or q_3.

Start at q_1:	$\delta(q_1, 1) = q_2$, so the FA ends in state q_2 – an accept state.
Start at q_2:	$\delta(q_2, 1) = q_2$, so the FA ends in state q_2 – an accept state.
Start at q_3:	$\delta(q_3, 1) = q_2$, so the FA ends in state q_2 – an accept state.

Unlike the previous finite automaton, this FA seems to accept more than one type of string. Consider the string "100". Following the transitions, we have

$\delta(q_1, 1) = q_2$	-- the accept state
$\delta(q_2, 0) = q_3$	
$\delta(q_3, 0) = q_2$	-- again the accept state and the string is accepted.

Thus, it appears that any string ending in two zeroes following a single one is accepted. This is not precisely the case, so we must refine our description a bit. Begin with a study of the start state, q_1. The FA stays in the start state until the first "1" is seen in the input string, after which it never returns to that state. Thus, we are correct in our statement that a string must contain at least a single "1" to be accepted and that the string "1" by itself is accepted.

Before considering the next state, we must define a new term: the **empty string**. This string, denoted by the symbol ε, has length 0 – thus contains no symbols. See below for its use.

Now consider the accept state q_2. We ask what strings will cause the FA to move to an accepting state if it begins in state q_2. The set of strings of length two or less that will take the FA from q_2 to q_2 is { **ε**, "1", "00", "01", "11"}. Note that we include the empty string in this list because the FA is state q_2 needs no further input to be in a final state.

We finally consider state q_3. The only way for the FA to move to that state is for it to process a "0" as input. The next input, whether a "0" or a "1" will take the FA back to state q_2 from state q_3. Starting in state q_3, the following strings of length 3 or less lead the FA to be in state q_2, the accepting state: { "0", "1", "01", "11", "000", "001", "100", "101" }. Noting that each of these strings must have been prefixed by an additional "0" for the FA to have been in state q_3 in the first place, we divide the accepted strings into two sets.

1) Those strings ending in a "1".
2) Those strings ending in an even number of "0" following the last "1".

Thus, the language recognized by machine M_2 is given by the following set expression.

L(M_2) = { *w* | *w* ends in a "1 } ∪ { *w* | *w* ends in an even number of "0" }.

Finite Automaton M_3

Here is the state diagram of another finite automaton. What can we say about M_3?

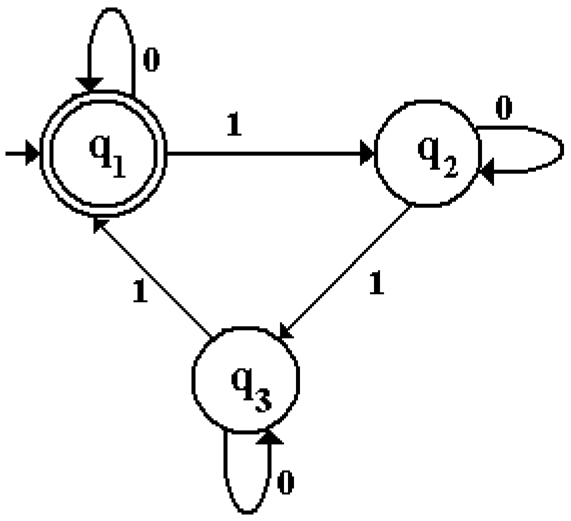

Figure: Finite Automaton M_3 for Analysis

The reader is invited to show that this finite automaton accepts all strings of 0's and 1's for which the number of 1's is divisible by 3. Recalling that the number 0 itself is divisible by 3, the reader will note that the FA accepts strings with no 1's as well as the empty string.

This figure was presented for two reasons: first, to show that the start state can itself be a final state, and secondly to show that we are not required to draw the states in a straight line.

Up to this point, each of the finite automata has had only a single accept state. We now discuss a FA with two accept states, just to show that we are not restricted to one such state.

Finite Automaton M_4

Here is the state diagram of another finite automaton. What can we say about M_4?

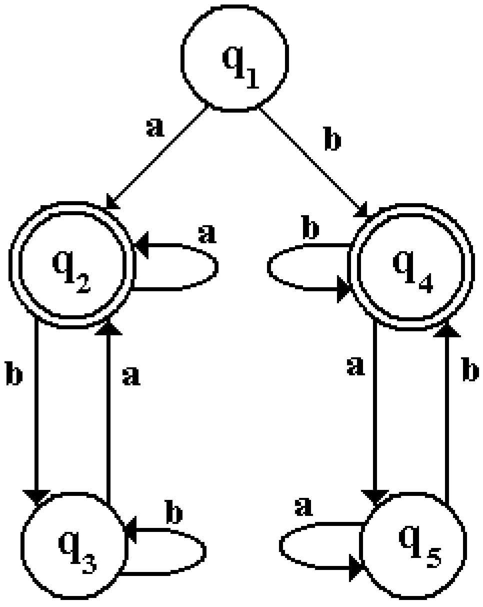

Figure: Finite Automaton M_4 for Analysis

Let's begin with a few abbreviated observations.

1. $Q = \{ q_1, q_2, q_3, q_4, q_5 \}$
2. $\Sigma = \{ a, b\}$ -- a departure from "0" and "1"
3. The start state is q_1.
4. $F = \{ q_2, q_4 \}$ -- we now have two accept states

M_4 accepts all strings that either begin and end with "a" or begin and end with "b". In part, $L(M_4)$ = { "a", "b", "aa", "bb", "aaa", "aba", "bab", "bbb", … }.

Problem – Modulo-8 Detector

Design a finite state machine that accepts strings representing binary numbers, with the most significant bit being read first and least significant bit last, accepting those equal to 5, mod 8.

Solution: The number $N \equiv 5$, modulo 8 if there exists an integer M such that $N = 8\bullet M + 5$. Suppose that N and M are so related and the binary representation of M is $M_K M_{K-1} \ldots M_1 M_0$, where each of the M_J is either 0 or 1. Then the binary representation of the integer N is given by $M_K M_{K-1} \ldots M_1 M_0 101$, obtained by appending "101" to the binary representation for M. Thus, we are looking for a finite automaton to accept strings ending in "101".

Note that another way to look at this problem is to recall that multiplication by 8 in binary is equivalent to adding three 0's to the right end, so that if $M_K M_{K-1} \ldots M_1 M_0$ is binary for M, then $M_K M_{K-1} \ldots M_1 M_0 000$ is the binary representation of $8\bullet M$, and $N = 8\bullet M + 5$ is represented by $M_K M_{K-1} \ldots M_1 M_0 000 + 101 = M_K M_{K-1} \ldots M_1 M_0 101$.

We begin solution of this problem by designing a four-state FSM to detect "101". At this point, we do not care about handling any other strings.

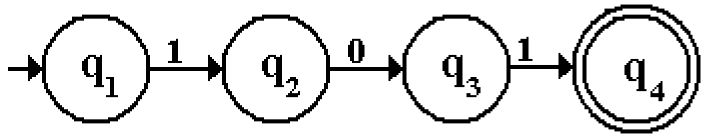

Figure: Part of a 101 Acceptor

Having drawn the obvious transitions, we now begin to handle the other five: one each from q_1, q_2, and q_3; and two from q_4. The other transitions from each of q_1 and q_2 are quite obvious – if q_1 gets a 0 the FA stays in q_1, and if q_2 gets a 1 the FA stays in q_2.

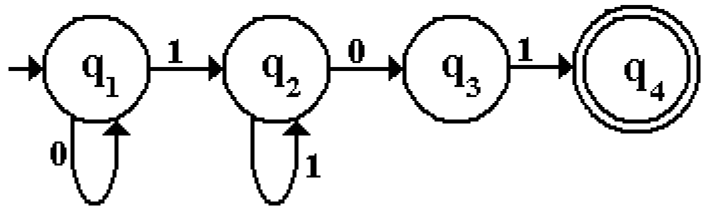

Figure: More of the 101 Acceptor

Now consider state q_3. If the FA is in state q_3 and the input is 0, then the last three symbols input were "100". The only way to convert this into a sequence ending in "101" is to append the entire sequence "101" to the "100"; so $\delta(q_3, 0) = q_1$. Now consider state q_4. If the FA is in state q_4, then the last three symbols input were "101". Suppose a 0 is input, giving the sequence "1010". The way to convert this to a sequence ending in "101" is to append another 1 to get "10101" – the binary representation of $21 \equiv 5 \bmod 8$. Thus $\delta(q_4, 0) = q_3$. Suppose that the FA is in state q_4 and a 1 is input, giving the sequence "1011". The way to convert this to a sequence ending in "101" is to append a "01" to it, thus $\delta(q_4, 1) = q_2$.

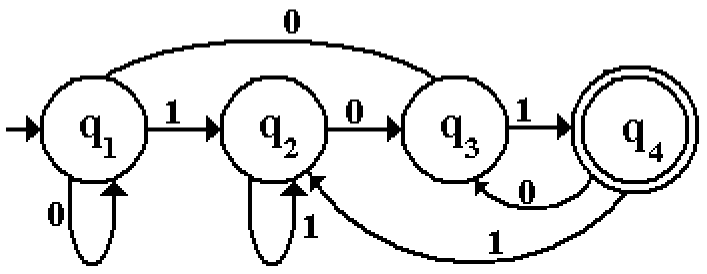

Figure: The State Diagram of the 101 Acceptor

The reader should take some time and trace the execution of this finite automaton for various input strings of size six or less. Note that the state q_1 would be an accepting state only if we were scanning for $N \equiv 0 \bmod 8$.

Algorithmic Simulation of a Finite Automaton

We now present a simple algorithmic simulator of a FA with $\Sigma = \{0, 1\}$, assuming a maximum of NMAX states, with the states labeled by integers in the range 1 through NMAX. The algorithm is written is a pseudocode that resembles BASIC. We begin assuming two matrices and an input function. The two matrices are

Next [1 to NMAX, 0 to 1]	–	the specification of the transition function. For all I and J, $1 \leq \text{Next}[I, J] \leq \text{NMAX}$.
Accept[1 to NMAX]	–	A Boolean array indicating an accept state.

We also assume the following subroutine to read input.
GetSymbol (ByRef NextSymbol : Integer, ByRef HaveAnother : Boolean)

The specification on the subroutine is that if the Boolean variable is TRUE, then the variable NextSymbol contains the next symbol to process. If the value of HaveAnother is FALSE, then there are no more symbols to process and the value of NextSymbol is not determined.

Here is a fragment of the pseudocode for the algorithm.

```
   Integer:    QP, QN, SN
‘
‘  At this point, the arrays Next and Accept must have
’  been defined.  These definitions specify the FA.
‘
   QN = 1        ‘ the start state is q1.
   GetSymbol (SN, HaveAnother)
   While (HaveAnother)
      QP = QN
      QN = Next[QP, SN]
      GetSymbol (SN, HaveAnother)
   End Loop
   Return Accept[QN]
```

Moore Machines and Mealy Machines

In the last two chapters of these notes we have studied two types of finite state machines, those with output associated with the transitions and those with output associated only with the states. It is now time that we gave these automata their formal names.

Automata such as the modulo-N counters in which the output is associated only with the state are called **Moore machines**, named after Edward Forrest Moore who first studied them in 1956. Automata such as the sequence detectors in which the output is associated with transitions between states are called **Mealy machines**, named after George H. Mealy who first studied them in 1955. The finite automata formally defined in this chapter are Moore machines. We shall shortly give a formal definition of Mealy machines.

Formal Definition of a Moore Machine

Moore machines are commonly used as language recognizers in the study of compiler theory and also as modulo-N counters. By definition, Moore machines do not have output that is associated with the transitions, but only with the state. Here is the formal definition.

A **Moore machine** is a 5-tuple $(Q, \Sigma, \delta, q_1, F)$, where

1. Q is a finite non-empty set called the set of **states**,
2. Σ is a finite set called the **input alphabet**,
3. $\delta: Q \times \Sigma \to Q$ is the **transition function**,
4. $q_1 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the set of **accept states** or **final states.**

11011 Sequence Detector

Here is the Moore machine version of the 11011 sequence detector.

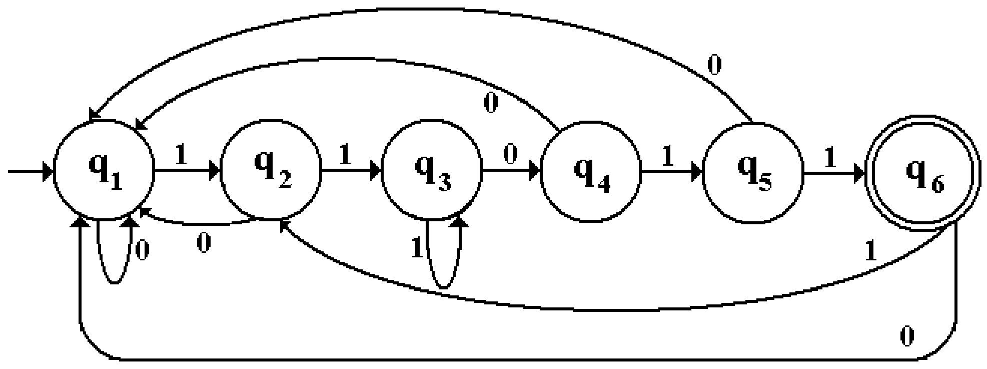

Figure: The 11011 Sequence Detector (String Acceptor) as a Moore Machine

For this definition, we have $Q = \{ q_1, q_2, q_3, q_4, q_5, q_6 \}$ and $\Sigma = \{ 0, 1 \}$. We have the transition function $\delta: Q \times \Sigma \to Q$ defined as follows.

$$\delta(q_1, 0) = q_1 \quad \delta(q_1, 1) = q_2$$
$$\delta(q_2, 0) = q_1 \quad \delta(q_2, 1) = q_3$$
$$\delta(q_3, 0) = q_4 \quad \delta(q_3, 1) = q_3$$
$$\delta(q_4, 0) = q_1 \quad \delta(q_4, 1) = q_5$$
$$\delta(q_5, 0) = q_1 \quad \delta(q_5, 1) = q_6$$
$$\delta(q_6, 0) = q_1 \quad \delta(q_1, 1) = q_2$$

The start state for this Moore machine is q_1 and the set of accepting states is $F = \{q_6\}$.

Formal Definition of a Mealy Machine

While Mealy machines can be used for language recognition, they are more often used to model certain electronic devices and other things we use, such as vending machines.

A **Mealy machine** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, \gamma, q_1)$, where

1. Q is a finite non-empty set called the set of **states**,
2. Σ is a finite set called the **input alphabet**,
3. Γ is a finite non-empty set of symbols, called the **output alphabet**,
4. $\delta: Q \times \Sigma \rightarrow Q$ is the **transition function**,
5. $\gamma: Q \times \Sigma \rightarrow \Gamma$ is the **output function**,
6. $q_1 \in Q$ is the **start state**.

There is no concept here of a set of final or accepting states.

11011 Sequence Detector

Here is the Mealy machine version of the 11011 sequence detector.

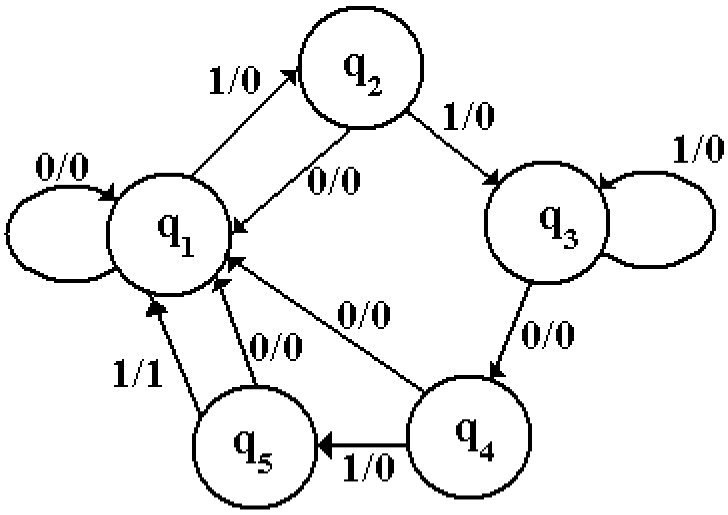

Figure: The 11011 Sequence Detector (String Acceptor) as a Mealy Machine

For this definition, we have $Q = \{ q_1, q_2, q_3, q_4, q_5\}$ (one fewer state) and $\Sigma = \{ 0, 1 \}$. We have the output alphabet (unique to Mealy machines) also as $\Gamma = \{ 0, 1 \}$. We should mention that this equality, $\Gamma = \Sigma$, is just an artifact of this particular problem.

The transition function is given by the following.

$\delta(q_1, 0) = q_1$, $\delta(q_2, 0) = q_1$, $\delta(q_3, 0) = q_4$, $\delta(q_4, 0) = q_1$, $\delta(q_5, 0) = q_1$,

$\delta(q_1, 1) = q_2$, $\delta(q_2, 1) = q_3$, $\delta(q_3, 1) = q_3$, $\delta(q_4, 1) = q_5$, $\delta(q_5, 1) = q_1$.

The output function is given by the following.

$\gamma(q_1, 0) = 0$, $\gamma(q_2, 0) = 0$, $\gamma(q_3, 0) = 0$, $\gamma(q_4, 0) = 0$, $\gamma(q_5, 0) = 0$,

$\gamma(q_1, 1) = 0$, $\gamma(q_2, 1) = 0$, $\gamma(q_3, 1) = 0$, $\gamma(q_4, 1) = 0$, $\gamma(q_5, 1) = 1$.

A Vending Machine Example of Mealy Machines

When this author was in college, his fraternity had an old soft drink machine. The machine was designed to sell 10 ounce Coca-Colas for 25 cents. It took nickels, dimes, and quarters as input, gave change in nickels, and produced a bottle automatically when enough money was put into the machine. What the Dean of Men did not know was that the machine gave bottles of beer, as we stocked the machine which cheap beer – you can buy cokes anywhere.

The states of the Mealy machine model of this vending machine correspond to the amount of money to be applied to the next purchase: 0, 5, 10, 15, and 20 cents. Note that the FA would not have a state labeled as 25 for the reason that if one put a quarter in the vending machine, he would get a beer now and there would be no money applied to the next purchase. The model of the machine must allow for giving change, so that inserting three dimes would result in getting a beer and a nickel in change. In extreme cases, a person would put in two dimes followed by a quarter, the expected result being a beer and four nickels in change.

The transitions for this FA are labeled Input/Output, with the following interpretation.

-- /	No input, transition is automatic	/ --	no output
N /	Nickel input	/ B	beer is output
D /	Dime input	/ BN	output a beer and a nickel
Q /	Quarter input	/ N	output a nickel in change.

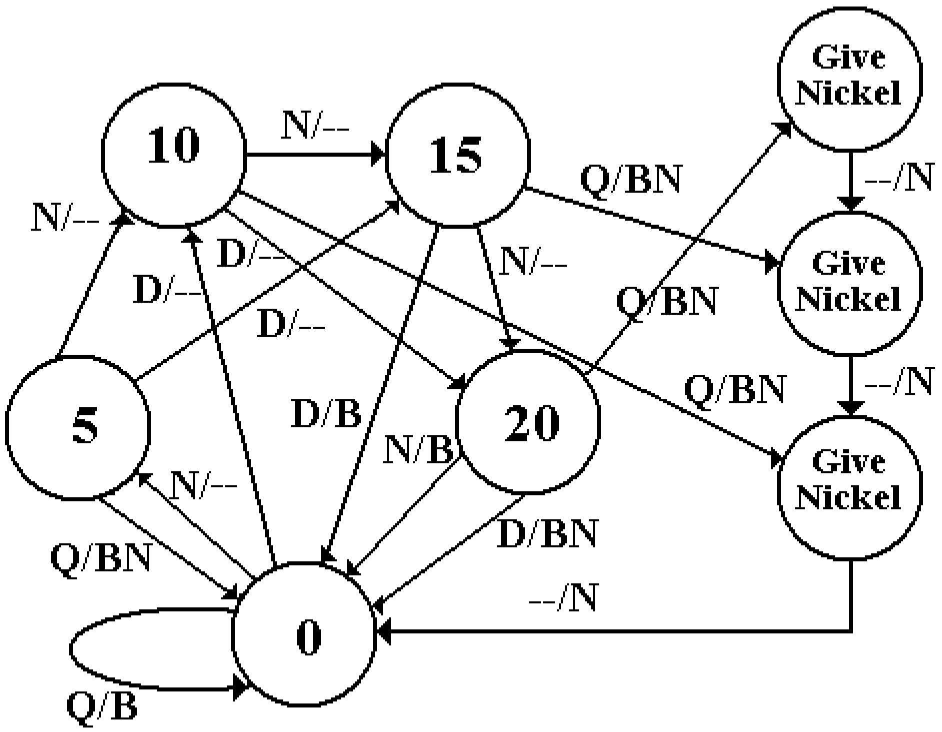

Figure: The Twenty-Five Cent Beer Machine as a Mealy Machine

The only thing notable about this diagram is the three states labeled “Give Nickel”. Each of these states automatically transitions to the next state with no input, and in that transition outputs another nickel in change.

WARNING: Finite Automata Theory Can Rot Your Mind

More precisely, the theory of finite automata is a tool with may profitable uses and a few notable misuses. Put another way, any model of computation should be understood along with the constraints on its proper use, and not used outside that area of application.

We close this section with a misapplication of parsing theory, a subject central to the understanding of language compilers. A parser is that front end of a compiler that reads all of the characters that make up a program and groups the characters into **tokens**, which might be considered the "words" of the programming language. For example, the code fragment

```
if (statecount == 0) then
```

contains 25 characters (without the CR at the line's end). A parser would create the following 7 tokens from this string: "if", "(", "statecount", "==", "0", ")", and "then".

We measure the time complexity of a parser in terms of N = the number of characters in the text of the program. This count included comments, because every comment must be read by the parser, even if it is immediately discarded. Most simplistic parsing algorithms are $O(N^2)$, meaning that the number of basic steps in the algorithm varies as the square of the size of the input. Parsers built on finite automata theory are O(N), meaning that the number of basic steps in the algorithm is a linear function of the input size.

Consider two parsers, one with $T_1(N) = 2\bullet N$ and with $T_2(N) = 0.5\bullet N^2$ steps. Imagine a fairly large size code module with N = 30,000. We have the following results.

$T_1(30000) = 2\bullet 30000 = 60000$

$T_2(30000) = 0.5\bullet 30000^2 = 0.5\bullet 900{,}000{,}000 = 450{,}000{,}000$, a factor of 7500 larger than T_1.

The time advantage for parsers built on finite automata theory is that they scan the input one time, left to right, with very little rescanning to specify the tokens. More simplistic parsers will scan the input several times and thus are much slower.

The misapplication of modern parsers occurs when the size of the input is guaranteed to be small and where the strings to be parsed have an unusual structure. We illustrate this point with a case study taken from this author's experience as an industrial programmer.

Consider the following problem: we are to put page numbers into the order that the pages appear in a document. The rules for paginating the document are as follows.

1) First come the pages for the front matter, these are numbered with Roman numerals.
2) Next come the pages that form the body of the document. These are numbered with standard numbers, except when correcting pages have been added. Correcting pages are of the form of a number followed by a letter, so that page 23 might be followed by page 23-A, 23-B, and 23-C before page 24.
3) Last come the pages for the appendices of the document. Page numbers for these pages are prefixed by a letter followed by a number; e.g. "B-17".

The standard parser approach is to read each string exactly once. This is theoretically a more efficient approach, but gives rise to considerable complexity. For example, suppose the first character read is a "C". Is this a Roman numeral or a designation for an appendix?

Sample strings to be parsed are shown below in their correct order.

```
III
XXI
CI
13
13-A
13-C
A-32
C-2
```

The more prudent programmer will realize that little time or efficiency is lost in reading each input string at least twice. The first pass on such a string will answer these questions.

1. Does the string begin with a letter or a numeral?
2. Does the string contain a dash?
3. Does the string contain a letter?
4. Does the string contain a letter that might be a Roman numeral, specifically one in the set { I, V, X, L, C, D, M }?
5. Does the string contain a digit.

Having scanned the string the first time, one can write an algorithm for the second pass that is tailored to the type of string found on the first pass.

The bottom line is that everything a student is taught is to be considered a tool for solving a problem; the student is responsible for selecting the most appropriate tool for the task.