

DISCRETE MATHEMATICS

W W L CHEN

© W W L Chen, 1991, 2008.

This chapter is available free to all individuals, on the understanding that it is not to be used for financial gain, and may be downloaded and/or photocopied, with or without permission from the author.

However, this document may not be kept on any information storage and retrieval system without permission from the author, unless such system is not accessible to any individuals other than its owners.

Chapter 6

FINITE STATE MACHINES

6.1. Introduction

EXAMPLE 6.1.1. Imagine that we have a very simple machine which will perform addition and multiplication of numbers from the set $\{1, 2\}$ and give us the answer, and suppose that we ask the machine to calculate $1 + 2$. The table below gives an indication of the order of events:

TIME	t_1	t_2	t_3	t_4
STATE	(1) s_1	(4) s_2 [1]	(7) s_3 [1, 2]	(10) s_1
INPUT	(2) 1	(5) 2	(8) +	
OUTPUT	(3) nothing	(6) nothing	(9) 3	

Here $t_1 < t_2 < t_3 < t_4$ denotes time. We explain the table a little further:

- (1) The machine is in a state s_1 of readiness.
- (2) We input the number 1.
- (3) There is as yet no output.
- (4) The machine is in a state s_2 (it has stored the number 1).
- (5) We input the number 2.
- (6) There is as yet no output.
- (7) The machine is in a state s_3 (it has stored the numbers 1 and 2).
- (8) We input the operation +.
- (9) The machine gives the answer 3.
- (10) The machine returns to the state s_1 of readiness.

Note that this machine has only a finite number of possible states. It is either in the state s_1 of readiness, or in a state where it has some, but not all, of the necessary information for it to perform its task. On the other hand, there are only a finite number of possible inputs, namely the numbers 1, 2 and the operations + and \times . Also, there are only a finite number of outputs, namely nothing, “junk” (when

incorrect input is made) or the right answers. If we investigate this simple machine a little further, then we will note that there are two little tasks that it performs every time we input some information: Depending on the state of the machine and the input, the machine gives an output. Depending on the state of the machine and the input, the machine moves to the next state.

DEFINITION. A finite state machine is a 6-tuple $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$, where

- (a) \mathcal{S} is the finite set of states for M ;
- (b) \mathcal{I} is the finite input alphabet for M ;
- (c) \mathcal{O} is the finite output alphabet for M ;
- (d) $\omega : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{O}$ is the output function;
- (e) $\nu : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{S}$ is the next-state function; and
- (f) $s_1 \in \mathcal{S}$ is the starting state.

EXAMPLE 6.1.2. Consider again our simple finite state machine described earlier. Suppose that when the machine recognizes incorrect input (*e.g.* three numbers or two operations), it gives the output “junk” and then returns to the initial state s_1 . It is clear that $\mathcal{I} = \{1, 2, +, \times\}$ and $\mathcal{O} = \{j, n, 1, 2, 3, 4\}$, where j denotes the output “junk” and n denotes no output. We can see that

$$\mathcal{S} = \{s_1, [1], [2], [+], [\times], [1, 1], [1, 2], [1, +], [1, \times], [2, 2], [2, +], [2, \times]\},$$

where s_1 denotes the initial state of readiness and the entries between the signs [and] denote the information in memory. The output function ω and the next-state function ν can be represented by the transition table below:

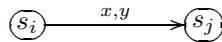
	output ω				next state ν			
	1	2	+	\times	1	2	+	\times
s_1	n	n	n	n	[1]	[2]	[+]	[\times]
[1]	n	n	n	n	[1, 1]	[1, 2]	[1, +]	[1, \times]
[2]	n	n	n	n	[1, 2]	[2, 2]	[2, +]	[2, \times]
[+]	n	n	j	j	[1, +]	[2, +]	s_1	s_1
[\times]	n	n	j	j	[1, \times]	[2, \times]	s_1	s_1
[1, 1]	j	j	2	1	s_1	s_1	s_1	s_1
[1, 2]	j	j	3	2	s_1	s_1	s_1	s_1
[1, +]	2	3	j	j	s_1	s_1	s_1	s_1
[1, \times]	1	2	j	j	s_1	s_1	s_1	s_1
[2, 2]	j	j	4	4	s_1	s_1	s_1	s_1
[2, +]	3	4	j	j	s_1	s_1	s_1	s_1
[2, \times]	2	4	j	j	s_1	s_1	s_1	s_1

Another way to describe a finite state machine is by means of a state diagram instead of a transition table. However, state diagrams can get very complicated very easily, so we shall illustrate this by a very simple example.

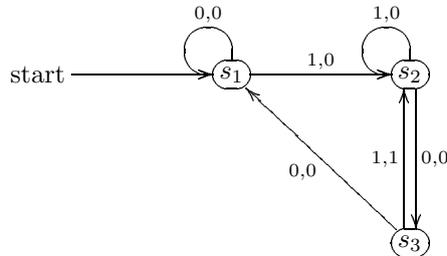
EXAMPLE 6.1.3. Consider a simple finite state machine $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$ with $\mathcal{S} = \{s_1, s_2, s_3\}$, $\mathcal{I} = \{0, 1\}$, $\mathcal{O} = \{0, 1\}$ and where the functions $\omega : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{O}$ and $\nu : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{S}$ are described by the transition table below:

	ω		ν	
	0	1	0	1
$+s_1+$	0	0	s_1	s_2
s_2	0	0	s_3	s_2
s_3	0	1	s_1	s_2

Here we have indicated that s_1 is the starting state. If we use the diagram



to describe $\omega(s_i, x) = y$ and $\nu(s_i, x) = s_j$, then we can draw the state diagram below:



Suppose that we have an input string 00110101, *i.e.* we input 0, 0, 1, 1, 0, 1, 0, 1 successively, while starting at state s_1 . After the first input 0, we get output 0 and return to state s_1 . After the second input 0, we again get output 0 and return to state s_1 . After the third input 1, we get output 0 and move to state s_2 . And so on. It is not difficult to see that we end up with the output string 00000101 and in state s_2 . Note that the input string 00110101 is in \mathcal{I}^* , the Kleene closure of \mathcal{I} , and that the output string 00000101 is in \mathcal{O}^* , the Kleene closure of \mathcal{O} .

We conclude this section by going ever so slightly upmarket.

EXAMPLE 6.1.4. Let us attempt to add two numbers in binary notation together, for example $a = 01011$ and $b = 00101$. Note that the extra 0's are there to make sure that the two numbers have the same number of digits and that the sum of them has no extra digits. This would normally be done in the following way:

$$\begin{array}{r}
 a \qquad \qquad \qquad 0 \ 1 \ 0 \ 1 \ 1 \\
 + \ b \qquad \qquad + \ 0 \ 0 \ 1 \ 0 \ 1 \\
 \hline
 c \qquad \qquad \qquad 1 \ 0 \ 0 \ 0 \ 0
 \end{array}$$

Let us write $a = a_5a_4a_3a_2a_1$ and $b = b_5b_4b_3b_2b_1$, and write $x_j = (a_j, b_j)$ for every $j = 1, 2, 3, 4, 5$. We can now think of the input string as $x_1x_2x_3x_4x_5$ and the output string as $c_1c_2c_3c_4c_5$, where $c = c_5c_4c_3c_2c_1$. Note, however, that when we perform each addition, we have to see whether there is a carry from the previous addition, so that are two possible states at the start of each addition. Let us call them s_0 (no carry) and s_1 (carry). Then $\mathcal{S} = \{s_0, s_1\}$, $\mathcal{I} = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ and $\mathcal{O} = \{0, 1\}$. The functions $\omega : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{O}$ and $\nu : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{S}$ can be represented by the following transition table:

	output ω				next state ν			
	(0, 0)	(0, 1)	(1, 0)	(1, 1)	(0, 0)	(0, 1)	(1, 0)	(1, 1)
$+s_0+$	0	1	1	0	s_0	s_0	s_0	s_1
s_1	1	0	0	1	s_0	s_1	s_1	s_1

Clearly s_0 is the starting state.

6.2. Pattern Recognition Machines

In this section, we study examples of finite state machines that are designed to recognize given patterns. As there is essentially no standard way of constructing such machines, we shall illustrate the underlying ideas by examples. The questions in this section will get progressively harder.

REMARK. In the examples that follow, we shall use words like “passive” and “excited” to describe the various states of the finite state machines that we are attempting to construct. These words play no role in serious mathematics, and should not be used in exercises or in any examination. It is hoped that by using such words here, the exposition will be a little more light-hearted and, hopefully, a little clearer.

EXAMPLE 6.2.1. Suppose that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 11 in the input string $x \in \mathcal{I}^*$. An example of an input string $x \in \mathcal{I}^*$ and its corresponding output string $y \in \mathcal{O}^*$ is shown below:

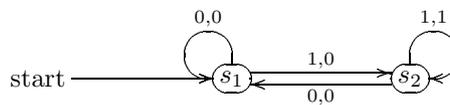
$$x = 10111010101111110101$$

$$y = 000110000000111110000$$

Note that the output digit is 0 when the sequence pattern 11 is not detected and 1 when the sequence pattern 11 is detected. In order to achieve this, we must ensure that the finite state machine has at least two states, a “passive” state when the previous entry is 0 (or when no entry has yet been made), and an “excited” state when the previous entry is 1. Furthermore, the finite state machine has to observe the following and take the corresponding actions:

- (1) If it is in its “passive” state and the next entry is 0, it gives an output 0 and remains in its “passive” state.
- (2) If it is in its “passive” state and the next entry is 1, it gives an output 0 and switches to its “excited” state.
- (3) If it is in its “excited” state and the next entry is 0, it gives an output 0 and switches to its “passive” state.
- (4) If it is in its “excited” state and the next entry is 1, it gives an output 1 and remains in its “excited” state.

It follows that if we denote by s_1 the “passive” state and by s_2 the “excited” state, then we have the state diagram below:



We then have the corresponding transition table:

	ω		ν	
	0	1	0	1
$+s_1+$	0	0	s_1	s_2
s_2	0	1	s_1	s_2

EXAMPLE 6.2.2. Suppose again that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 111 in the input string $x \in \mathcal{I}^*$. An example of the same input string $x \in \mathcal{I}^*$ and its corresponding output string $y \in \mathcal{O}^*$ is shown below:

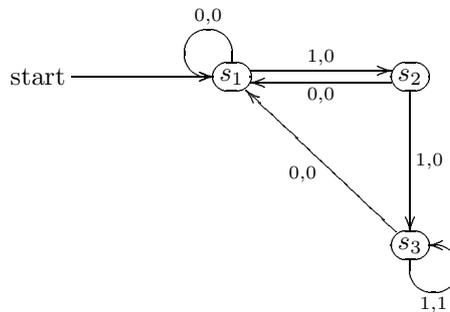
$$x = 10111010101111110101$$

$$y = 00001000000011110000$$

In order to achieve this, the finite state machine must now have at least three states, a “passive” state when the previous entry is 0 (or when no entry has yet been made), an “expectant” state when the previous two entries are 01 (or when only one entry has so far been made and it is 1), and an “excited” state when the previous two entries are 11. Furthermore, the finite state machine has to observe the following and take the corresponding actions:

- (1) If it is in its “passive” state and the next entry is 0, it gives an output 0 and remains in its “passive” state.
- (2) If it is in its “passive” state and the next entry is 1, it gives an output 0 and switches to its “expectant” state.
- (3) If it is in its “expectant” state and the next entry is 0, it gives an output 0 and switches to its “passive” state.
- (4) If it is in its “expectant” state and the next entry is 1, it gives an output 0 and switches to its “excited” state.
- (5) If it is in its “excited” state and the next entry is 0, it gives an output 0 and switches to its “passive” state.
- (6) If it is in its “excited” state and the next entry is 1, it gives an output 1 and remains in its “excited” state.

If we now denote by s_1 the “passive” state, by s_2 the “expectant” state and by s_3 the “excited” state, then we have the state diagram below:



We then have the corresponding transition table:

	ω		ν	
	0	1	0	1
+s ₁ +	0	0	s ₁	s ₂
s ₂	0	0	s ₁	s ₃
s ₃	0	1	s ₁	s ₃

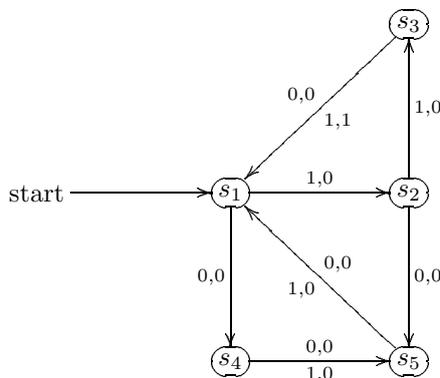
EXAMPLE 6.2.3. Suppose again that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 111 in the input string $x \in \mathcal{I}^*$, but only when the third 1 in the sequence pattern 111 occurs at a position that is a multiple of 3. An example of the same input string $x \in \mathcal{I}^*$ and its corresponding output string $y \in \mathcal{O}^*$ is shown below:

$$x = 1011101010101111110101$$

$$y = 00000000000000100000$$

In order to achieve this, the finite state machine must as before have at least three states, a “passive” state when the previous entry is 0 (or when no entry has yet been made), an “expectant” state when the previous two entries are 01 (or when only one entry has so far been made and it is 1), and an “excited” state when the previous two entries are 11. However, this is not enough, as we also need to keep track of the entries to ensure that the position of the first 1 in the sequence pattern 111 occurs at a position immediately after a multiple of 3. It follows that if we permit the machine to be at its “passive” state only either at the start or after $3k$ entries, where $k \in \mathbb{N}$, then it is necessary to have two further states to cater for the possibilities of 0 in at least one of the two entries after a “passive” state and to then

delay the return to this “passive” state. We can have the state diagram below:



We then have the corresponding transition table:

	ω		ν	
	0	1	0	1
$+s_1+$	0	0	s_4	s_2
s_2	0	0	s_5	s_3
s_3	0	1	s_1	s_1
s_4	0	0	s_5	s_5
s_5	0	0	s_1	s_1

EXAMPLE 6.2.4. Suppose again that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 0101 in the input string $x \in \mathcal{I}^*$. An example of the same input string $x \in \mathcal{I}^*$ and its corresponding output string $y \in \mathcal{O}^*$ is shown below:

$$x = 10111010101111110101$$

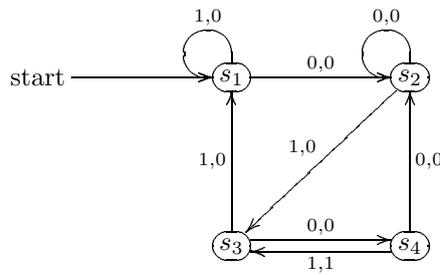
$$y = 00000000101000000001$$

In order to achieve this, the finite state machine must have at least four states, a “passive” state when the previous two entries are 11 (or when no entry has yet been made), an “expectant” state when the previous three entries are 000 or 100 or 110 (or when only one entry has so far been made and it is 0), an “excited” state when the previous two entries are 01, and a “cannot-wait” state when the previous three entries are 010. Furthermore, the finite state machine has to observe the following and take the corresponding actions:

- (1) If it is in its “passive” state and the next entry is 0, it gives an output 0 and switches to its “expectant” state.
- (2) If it is in its “passive” state and the next entry is 1, it gives an output 0 and remains in its “passive” state.
- (3) If it is in its “expectant” state and the next entry is 0, it gives an output 0 and remains in its “expectant” state.
- (4) If it is in its “expectant” state and the next entry is 1, it gives an output 0 and switches to its “excited” state.
- (5) If it is in its “excited” state and the next entry is 0, it gives an output 0 and switches to its “cannot-wait” state.
- (6) If it is in its “excited” state and the next entry is 1, it gives an output 0 and switches to its “passive” state.

- (7) If it is in its “cannot-wait” state and the next entry is 0, it gives an output 0 and switches to its “expectant” state.
- (8) If it is in its “cannot-wait” state and the next entry is 1, it gives an output 1 and switches to its “excited” state.

It follows that if we denote by s_1 the “passive” state, by s_2 the “expectant” state, by s_3 the “excited” state and by s_4 the “cannot-wait” state, then we have the state diagram below:



We then have the corresponding transition table:

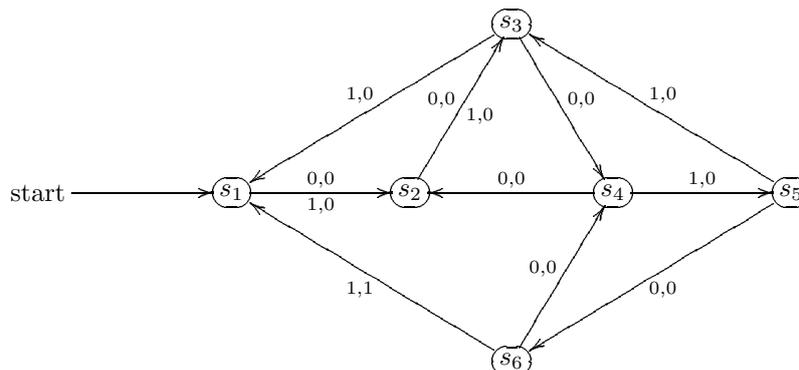
	ω		ν	
	0	1	0	1
$+s_1+$	0	0	s_2	s_1
s_2	0	0	s_2	s_3
s_3	0	0	s_4	s_1
s_4	0	1	s_2	s_3

EXAMPLE 6.2.5. Suppose again that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 0101 in the input string $x \in \mathcal{I}^*$, but only when the last 1 in the sequence pattern 0101 occurs at a position that is a multiple of 3. An example of the same input string $x \in \mathcal{I}^*$ and its corresponding output string $y \in \mathcal{O}^*$ is shown below:

$$x = 10111010101111110101$$

$$y = 00000000100000000000$$

In order to achieve this, the finite state machine must have at least four states s_3, s_4, s_5 and s_6 , corresponding to the four states before. We need two further states for delaying purposes. If we denote these two states by s_1 and s_2 , then we can have the state diagram as below:



We then have the corresponding transition table:

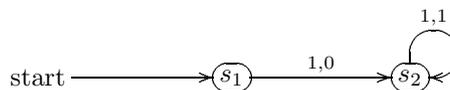
	ω		ν	
	0	1	0	1
$+s_1+$	0	0	s_2	s_2
s_2	0	0	s_3	s_3
s_3	0	0	s_4	s_1
s_4	0	0	s_2	s_5
s_5	0	0	s_6	s_3
s_6	0	1	s_4	s_1

6.3. An Optimistic Approach

We now describe an approach which helps greatly in the construction of pattern recognition machines. The idea is extremely simple. We construct first of all the part of the machine to take care of the situation when the required pattern occurs repeatedly and without interruption. We then complete the machine by studying the situation when the “wrong” input is made at each state.

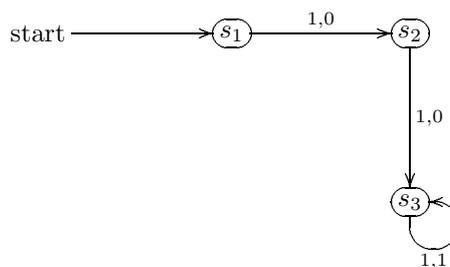
We shall illustrate this approach by looking the same five examples in the previous section.

EXAMPLE 6.3.1. Suppose that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 11 in the input string $x \in \mathcal{I}^*$. Consider first of all the situation when the required pattern occurs repeatedly and without interruption. In other words, consider the situation when the input string is 11111... To describe this situation, we have the following incomplete state diagram:



It now remains to study the situation when we have the “wrong” input at each state. Naturally, with a “wrong” input, the output is always 0, so the only unresolved question is to determine the next state. Note that whenever we get an input 0, the process starts all over again; in other words, we must return to state s_1 . We therefore obtain the state diagram as in Example 6.2.1.

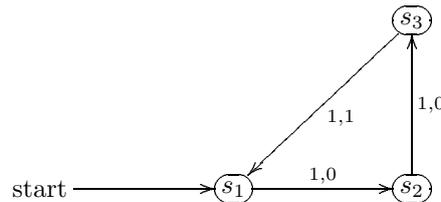
EXAMPLE 6.3.2. Suppose again that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 111 in the input string $x \in \mathcal{I}^*$. Consider first of all the situation when the required pattern occurs repeatedly and without interruption. In other words, consider the situation when the input string is 11111... To describe this situation, we have the following incomplete state diagram:



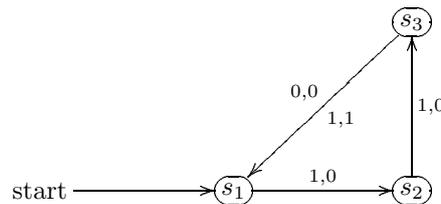
It now remains to study the situation when we have the “wrong” input at each state. As before, with a “wrong” input, the output is always 0, so the only unresolved question is to determine the next state.

Note that whenever we get an input 0, the process starts all over again; in other words, we must return to state s_1 . We therefore obtain the state diagram as Example 6.2.2.

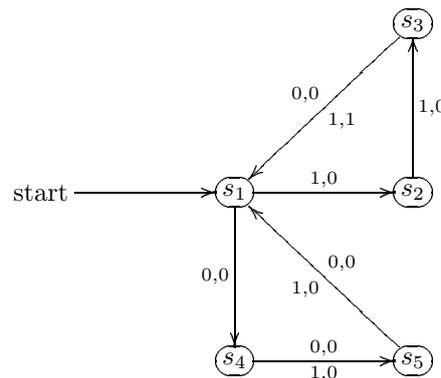
EXAMPLE 6.3.3. Suppose again that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 111 in the input string $x \in \mathcal{I}^*$, but only when the third 1 in the sequence pattern 111 occurs at a position that is a multiple of 3. Consider first of all the situation when the required pattern occurs repeatedly and without interruption. In other words, consider the situation when the input string is 111111... To describe this situation, we have the following incomplete state diagram:



It now remains to study the situation when we have the “wrong” input at each state. As before, with a “wrong” input, the output is always 0, so the only unresolved question is to determine the next state. Suppose that the machine is at state s_3 , and the wrong input 0 is made. We note that if the next three entries are 111, then that pattern should be recognized. It follows that $\nu(s_3, 0) = s_1$. We now have the following incomplete state diagram:



Suppose that the machine is at state s_1 , and the wrong input 0 is made. We note that the next two inputs cannot contribute to any pattern, so we need to delay by two steps before returning to s_1 . We now have the following incomplete state diagram:

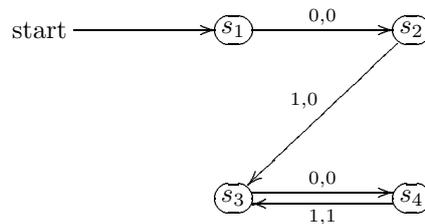


Finally, it remains to examine $\nu(s_2, 0)$. We therefore obtain the state diagram as Example 6.2.3.

Note that in Example 6.3.3, in dealing with the input 0 at state s_1 , we have actually introduced the extra states s_4 and s_5 . These extra states are not actually involved with positive identification of the desired pattern, but are essential in delaying the return to one of the states already present. It follows that at these extra states, the output should always be 0. However, we have to investigate the situation for input 0 and 1.

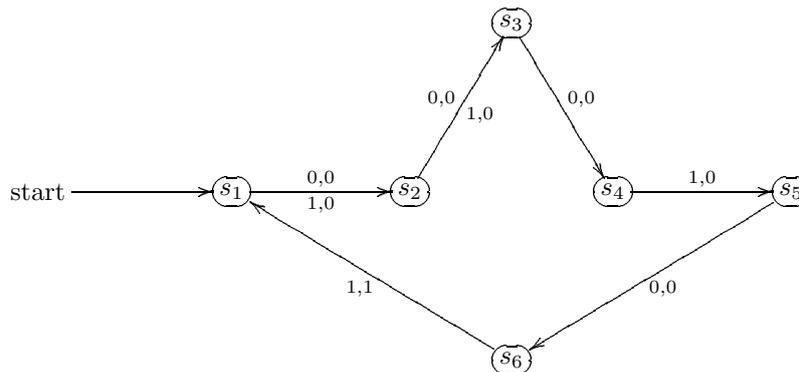
In the next two examples, we do not include all the details. Readers are advised to look carefully at the argument and ensure that they understand the underlying arguments. The main question is to recognize that with a “wrong” input, we may still be able to recover part of a pattern. For example, if the pattern we want is 01011 but we got 01010, where the fifth input is incorrect, then the last three inputs in 01010 can still possibly form the first three inputs of the pattern 01011. If one is optimistic, one might hope that the first seven inputs are 0101011.

EXAMPLE 6.3.4. Suppose again that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 0101 in the input string $x \in \mathcal{I}^*$. Consider first of all the situation when the required pattern occurs repeatedly and without interruption. In other words, consider the situation when the input string is 010101 To describe this situation, we have the following incomplete state diagram:



It now remains to study the situation when we have the “wrong” input at each state. As before, with a “wrong” input, the output is always 0, so the only unresolved question is to determine the next state. Recognizing that $\nu(s_1, 1) = s_1$, $\nu(s_2, 0) = s_2$, $\nu(s_3, 1) = s_1$ and $\nu(s_4, 0) = s_2$, we therefore obtain the state diagram as Example 6.2.4.

EXAMPLE 6.3.5. Suppose again that $\mathcal{I} = \mathcal{O} = \{0, 1\}$, and that we want to design a finite state machine that recognizes the sequence pattern 0101 in the input string $x \in \mathcal{I}^*$, but only when the last 1 in the sequence pattern 0101 occurs at a position that is a multiple of 3. Consider first of all the situation when the required pattern occurs repeatedly and without interruption. Note that the first two inputs do not contribute to any pattern, so we consider the situation when the input string is $x_1x_20101 . . .$, where $x_1, x_2 \in \{0, 1\}$. To describe this situation, we have the following incomplete state diagram:



It now remains to study the situation when we have the “wrong” input at each state. As before, with a “wrong” input, the output is always 0, so the only unresolved question is to determine the next state. Recognizing that $\nu(s_3, 1) = s_1$, $\nu(s_4, 0) = s_2$, $\nu(s_5, 1) = s_3$ and $\nu(s_6, 0) = s_4$, we therefore obtain the state diagram as Example 6.2.5.

6.4. Delay Machines

We now consider a type of finite state machines that is useful in the design of digital devices. These are called k -unit delay machines, where $k \in \mathbb{N}$. Corresponding to the input $x = x_1x_2x_3 . . .$, the machine will

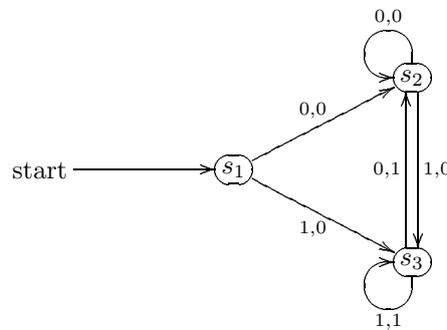
give the output

$$y = \underbrace{0 \dots 0}_k x_1 x_2 x_3 \dots;$$

in other words, it adds k 0's at the front of the string.

An important point to note is that if we want to add k 0's at the front of the string, then the same result can be obtained by feeding the string into a 1-unit delay machine, then feeding the output obtained into the same machine, and repeating this process. It follows that we need only investigate 1-unit delay machines.

EXAMPLE 6.4.1. Suppose that $\mathcal{I} = \{0, 1\}$. Then a 1-unit delay machine can be described by the state diagram below:



Note that $\nu(s_i, 0) = s_2$ and $\omega(s_2, x) = 0$; also $\nu(s_i, 1) = s_3$ and $\omega(s_3, x) = 1$. It follows that at both states s_2 and s_3 , the output is always the previous input.

In general, \mathcal{I} may contain more than 2 elements. Any attempt to construct a 1-unit delay machine by state diagram is likely to end up in a mess. We therefore seek to understand the ideas behind such a machine, and hope that it will help us in our attempt to construct it.

EXAMPLE 6.4.2. Suppose that $\mathcal{I} = \{0, 1, 2, 3, 4, 5\}$. Necessarily, there must be a starting state s_1 which will add the 0 to the front of the string. It follows that $\omega(s_1, x) = 0$ for every $x \in \mathcal{I}$. The next question is to study $\nu(s_1, x)$ for every $x \in \mathcal{I}$. However, we shall postpone this until towards the end of our argument. At this point, we are not interested in designing a machine with the least possible number of states. We simply want a 1-unit delay machine, and we are not concerned if the number of states is more than this minimum. We now let s_2 be a state that will “delay” the input 0. To be precise, we want the state s_2 to satisfy the following requirements:

- (1) For every state $s \in \mathcal{S}$, we have $\nu(s, 0) = s_2$.
- (2) For every state $s \in \mathcal{S}$ and every input $x \in \mathcal{I} \setminus \{0\}$, we have $\nu(s, x) \neq s_2$.
- (3) For every input $x \in \mathcal{I}$, we have $\omega(s_2, x) = 0$.

Next, we let s_3 be a state that will “delay” the input 1. To be precise, we want the state s_3 to satisfy the following requirements:

- (1) For every state $s \in \mathcal{S}$, we have $\nu(s, 1) = s_3$.
- (2) For every state $s \in \mathcal{S}$ and every input $x \in \mathcal{I} \setminus \{1\}$, we have $\nu(s, x) \neq s_3$.
- (3) For every input $x \in \mathcal{I}$, we have $\omega(s_3, x) = 1$.

Similarly, we let s_4, s_5, s_6 and s_7 be the states that will “delay” the inputs 2, 3, 4 and 5 respectively. Let us look more closely at the state s_2 . By requirements (1) and (2) above, the machine arrives at state s_2 precisely when the previous input is 0; by requirement (3), the next output is 0. This means that “the previous input must be the same as the next output”, which is precisely what we are looking for.

Now all these requirements are summarized in the table below:

	ω						ν					
	0	1	2	3	4	5	0	1	2	3	4	5
$+s_1+$	0	0	0	0	0	0						
s_2	0	0	0	0	0	0	s_2	s_3	s_4	s_5	s_6	s_7
s_3	1	1	1	1	1	1	s_2	s_3	s_4	s_5	s_6	s_7
s_4	2	2	2	2	2	2	s_2	s_3	s_4	s_5	s_6	s_7
s_5	3	3	3	3	3	3	s_2	s_3	s_4	s_5	s_6	s_7
s_6	4	4	4	4	4	4	s_2	s_3	s_4	s_5	s_6	s_7
s_7	5	5	5	5	5	5	s_2	s_3	s_4	s_5	s_6	s_7

Finally, we study $\nu(s_1, x)$ for every $x \in \mathcal{I}$. It is clear that if the first input is 0, then the next state must “delay” this 0. It follows that $\nu(s_1, 0) = s_2$. Similarly, $\nu(s_1, 1) = s_3$, and so on. We therefore have the following transition table:

	ω						ν					
	0	1	2	3	4	5	0	1	2	3	4	5
$+s_1+$	0	0	0	0	0	0	s_2	s_3	s_4	s_5	s_6	s_7
s_2	0	0	0	0	0	0	s_2	s_3	s_4	s_5	s_6	s_7
s_3	1	1	1	1	1	1	s_2	s_3	s_4	s_5	s_6	s_7
s_4	2	2	2	2	2	2	s_2	s_3	s_4	s_5	s_6	s_7
s_5	3	3	3	3	3	3	s_2	s_3	s_4	s_5	s_6	s_7
s_6	4	4	4	4	4	4	s_2	s_3	s_4	s_5	s_6	s_7
s_7	5	5	5	5	5	5	s_2	s_3	s_4	s_5	s_6	s_7

6.5. Equivalence of States

Occasionally, we may find that two finite state machines perform the same tasks, *i.e.* give the same output strings for the same input string, but contain different numbers of states. The question arises that some of the states are “essentially redundant”. We may wish to find a process to reduce the number of states. In other words, we would like to simplify the finite state machine.

The answer to this question is given by the Minimization process. However, to describe this process, we must first study the equivalence of states in a finite state machine.

Suppose that $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$ is a finite state machine.

DEFINITION. We say that two states $s_i, s_j \in \mathcal{S}$ are 1-equivalent, denoted by $s_i \cong_1 s_j$, if $\omega(s_i, x) = \omega(s_j, x)$ for every $x \in \mathcal{I}$. Similarly, for every $k \in \mathbb{N}$, we say that two states $s_i, s_j \in \mathcal{S}$ are k -equivalent, denoted by $s_i \cong_k s_j$, if $\omega(s_i, x) = \omega(s_j, x)$ for every $x \in \mathcal{I}^k$ (here $\omega(s_i, x_1 \dots x_k)$ denotes the output string corresponding to the input string $x_1 \dots x_k$, starting at state s_i). Furthermore, we say that two states $s_i, s_j \in \mathcal{S}$ are equivalent, denoted by $s_i \cong s_j$, if $s_i \cong_k s_j$ for every $k \in \mathbb{N}$.

The ideas that underpin the Minimization process are summarized in the following result.

PROPOSITION 6A.

- (a) For every $k \in \mathbb{N}$, the relation \cong_k is an equivalence relation on \mathcal{S} .
- (b) Suppose that $s_i, s_j \in \mathcal{S}$, $k \in \mathbb{N}$ and $k \geq 2$, and that $s_i \cong_k s_j$. Then $s_i \cong_{k-1} s_j$. In other words, k -equivalence implies $(k - 1)$ -equivalence.
- (c) Suppose that $s_i, s_j \in \mathcal{S}$ and $k \in \mathbb{N}$. Then $s_i \cong_{k+1} s_j$ if and only if $s_i \cong_k s_j$ and $\nu(s_i, x) \cong_k \nu(s_j, x)$ for every $x \in \mathcal{I}$.

PROOF. (a) The proof is simple and is left as an exercise.

(b) Suppose that $s_i \not\cong_{k-1} s_j$. Then there exists $x = x_1x_2 \dots x_{k-1} \in \mathcal{I}^{k-1}$ such that, writing $\omega(s_i, x) = y'_1y'_2 \dots y'_{k-1}$ and $\omega(s_j, x) = y''_1y''_2 \dots y''_{k-1}$, we have that $y'_1y'_2 \dots y'_{k-1} \neq y''_1y''_2 \dots y''_{k-1}$. Suppose now that $u \in \mathcal{I}$. Then clearly there exist $z', z'' \in \mathcal{O}$ such that

$$\omega(s_i, xu) = y'_1y'_2 \dots y'_{k-1}z' \neq y''_1y''_2 \dots y''_{k-1}z'' = \omega(s_j, xu).$$

Note now that $xu \in \mathcal{I}^k$. It follows that $s_i \not\cong_k s_j$.

(c) Suppose that $s_i \cong_k s_j$ and $\nu(s_i, x) \cong_k \nu(s_j, x)$ for every $x \in \mathcal{I}$. Let $x_1x_2 \dots x_{k+1} \in \mathcal{I}^{k+1}$. Clearly

$$\omega(s_i, x_1x_2 \dots x_{k+1}) = \omega(s_i, x_1)\omega(\nu(s_i, x_1), x_2 \dots x_{k+1}). \quad (1)$$

Similarly

$$\omega(s_j, x_1x_2 \dots x_{k+1}) = \omega(s_j, x_1)\omega(\nu(s_j, x_1), x_2 \dots x_{k+1}). \quad (2)$$

Since $s_i \cong_k s_j$, it follows from (b) that

$$\omega(s_i, x_1) = \omega(s_j, x_1). \quad (3)$$

On the other hand, since $\nu(s_i, x_1) \cong_k \nu(s_j, x_1)$, it follows that

$$\omega(\nu(s_i, x_1), x_2 \dots x_{k+1}) = \omega(\nu(s_j, x_1), x_2 \dots x_{k+1}). \quad (4)$$

On combining (1)–(4), we obtain

$$\omega(s_i, x_1x_2 \dots x_{k+1}) = \omega(s_j, x_1x_2 \dots x_{k+1}). \quad (5)$$

Note now that (5) holds for every $x_1x_2 \dots x_{k+1} \in \mathcal{I}^{k+1}$. Hence $s_i \cong_{k+1} s_j$. The proof of the converse is left as an exercise. \circ

6.6. The Minimization Process

In our earlier discussion concerning the design of finite state machines, we have not taken any care to ensure that the finite state machine that we obtain has the smallest number of states possible. Indeed, it is not necessary to design such a finite state machine with the minimal number of states, as any other finite state machine that performs the same task can be minimized.

THE MINIMIZATION PROCESS.

- (1) Start with $k = 1$. By examining the rows of the transition table, determine 1-equivalent states. Denote by P_1 the set of 1-equivalence classes of \mathcal{S} (induced by \cong_1).
- (2) Let P_k denote the set of k -equivalence classes of \mathcal{S} (induced by \cong_k). In view of Proposition 6A(b), we now examine all the states in each k -equivalence class of \mathcal{S} , and use Proposition 6A(c) to determine P_{k+1} , the set of all $(k+1)$ -equivalence classes of \mathcal{S} (induced by \cong_{k+1}).
- (3) If $P_{k+1} \neq P_k$, then increase k by 1 and repeat (2).
- (4) If $P_{k+1} = P_k$, then the process is complete. We select one state from each equivalence class.

EXAMPLE 6.6.1. Suppose that $\mathcal{I} = \mathcal{O} = \{0, 1\}$. Consider a finite state machine with the transition table below:

	ω		ν	
	0	1	0	1
s_1	0	1	s_4	s_3
s_2	1	0	s_5	s_2
s_3	0	0	s_2	s_4
s_4	0	0	s_5	s_3
s_5	1	0	s_2	s_5
s_6	1	0	s_1	s_6

For the time being, we do not indicate which state is the starting state. Clearly

$$P_1 = \{\{s_1\}, \{s_2, s_5, s_6\}, \{s_3, s_4\}\}.$$

We now examine the 1-equivalence classes $\{s_2, s_5, s_6\}$ and $\{s_3, s_4\}$ separately to determine the possibility of 2-equivalence. Note at this point that two states from different 1-equivalence classes cannot be 2-equivalent, in view of Proposition 6A(b). Consider first $\{s_2, s_5, s_6\}$. Note that

$$\nu(s_2, 0) = s_5 \cong_1 s_2 = \nu(s_5, 0) \quad \text{and} \quad \nu(s_2, 1) = s_2 \cong_1 s_5 = \nu(s_5, 1).$$

It follows from Proposition 6A(c) that $s_2 \cong_2 s_5$. On the other hand,

$$\nu(s_2, 0) = s_5 \not\cong_1 s_1 = \nu(s_6, 0).$$

It follows that $s_2 \not\cong_2 s_6$, and hence $s_5 \not\cong_2 s_6$ also. Consider now $\{s_3, s_4\}$. Note that

$$\nu(s_3, 0) = s_2 \cong_1 s_5 = \nu(s_4, 0) \quad \text{and} \quad \nu(s_3, 1) = s_4 \cong_1 s_3 = \nu(s_4, 1).$$

It follows that $s_3 \cong_2 s_4$. Hence

$$P_2 = \{\{s_1\}, \{s_2, s_5\}, \{s_3, s_4\}, \{s_6\}\}.$$

We now examine the 2-equivalence classes $\{s_2, s_5\}$ and $\{s_3, s_4\}$ separately to determine the possibility of 3-equivalence. Consider first $\{s_2, s_5\}$. Note that

$$\nu(s_2, 0) = s_5 \cong_2 s_2 = \nu(s_5, 0) \quad \text{and} \quad \nu(s_2, 1) = s_2 \cong_2 s_5 = \nu(s_5, 1).$$

It follows from Proposition 6A(c) that $s_2 \cong_3 s_5$. On the other hand,

$$\nu(s_3, 0) = s_2 \cong_2 s_5 = \nu(s_4, 0) \quad \text{and} \quad \nu(s_3, 1) = s_4 \cong_2 s_3 = \nu(s_4, 1).$$

It follows that $s_3 \cong_3 s_4$. Hence

$$P_3 = \{\{s_1\}, \{s_2, s_5\}, \{s_3, s_4\}, \{s_6\}\}.$$

Clearly $P_2 = P_3$. It follows that the process ends. We now choose one state from each equivalence class to obtain the following simplified transition table:

	ω		ν	
	0	1	0	1
s_1	0	1	s_3	s_3
s_2	1	0	s_2	s_2
s_3	0	0	s_2	s_3
s_6	1	0	s_1	s_6

Next, we repeat the entire process in a more efficient form. We observe easily from the transition table that

$$P_1 = \{\{s_1\}, \{s_2, s_5, s_6\}, \{s_3, s_4\}\}.$$

Let us label these three 1-equivalence classes by A, B, C respectively. We now attach an extra column to the right hand side of the transition table where each state has been replaced by the 1-equivalence class it belongs to.

	ω		ν		\cong_1
	0	1	0	1	
s_1	0	1	s_4	s_3	A
s_2	1	0	s_5	s_2	B
s_3	0	0	s_2	s_4	C
s_4	0	0	s_5	s_3	C
s_5	1	0	s_2	s_5	B
s_6	1	0	s_1	s_6	B

Next, we repeat the information on the next-state function, again where each state has been replaced by the 1-equivalence class it belongs to.

	ω		ν		\cong_1	ν	
	0	1	0	1		0	1
s_1	0	1	s_4	s_3	A	C	C
s_2	1	0	s_5	s_2	B	B	B
s_3	0	0	s_2	s_4	C	B	C
s_4	0	0	s_5	s_3	C	B	C
s_5	1	0	s_2	s_5	B	B	B
s_6	1	0	s_1	s_6	B	A	B

Recall that to get 2-equivalence, two states must be 1-equivalent, and their next states must also be 1-equivalent. In other words, if we inspect the new columns of our table, two states are 2-equivalent precisely when the patterns are the same. Indeed,

$$P_2 = \{\{s_1\}, \{s_2, s_5\}, \{s_3, s_4\}, \{s_6\}\}.$$

Let us label these four 2-equivalence classes by A, B, C, D respectively. We now attach an extra column to the right hand side of the transition table where each state has been replaced by the 2-equivalence class it belongs to.

	ω		ν		\cong_1	ν		\cong_2
	0	1	0	1		0	1	
s_1	0	1	s_4	s_3	A	C	C	A
s_2	1	0	s_5	s_2	B	B	B	B
s_3	0	0	s_2	s_4	C	B	C	C
s_4	0	0	s_5	s_3	C	B	C	C
s_5	1	0	s_2	s_5	B	B	B	B
s_6	1	0	s_1	s_6	B	A	B	D

Next, we repeat the information on the next-state function, again where each state has been replaced

by the 2-equivalence class it belongs to.

	ω		ν		\cong_1	ν		\cong_2	ν	
	0	1	0	1		0	1		0	1
s_1	0	1	s_4	s_3	A	C	C	A	C	C
s_2	1	0	s_5	s_2	B	B	B	B	B	B
s_3	0	0	s_2	s_4	C	B	C	C	B	C
s_4	0	0	s_5	s_3	C	B	C	C	B	C
s_5	1	0	s_2	s_5	B	B	B	B	B	B
s_6	1	0	s_1	s_6	B	A	B	D	A	D

Recall that to get 3-equivalence, two states must be 2-equivalent, and their next states must also be 2-equivalent. In other words, if we inspect the new columns of our table, two states are 3-equivalent precisely when the patterns are the same. Indeed,

$$P_3 = \{\{s_1\}, \{s_2, s_5\}, \{s_3, s_4\}, \{s_6\}\}.$$

Let us label these four 3-equivalence classes by A, B, C, D respectively. We now attach an extra column to the right hand side of the transition table where each state has been replaced by the 3-equivalence class it belongs to.

	ω		ν		\cong_1	ν		\cong_2	ν		\cong_3
	0	1	0	1		0	1		0	1	
s_1	0	1	s_4	s_3	A	C	C	A	C	C	A
s_2	1	0	s_5	s_2	B	B	B	B	B	B	B
s_3	0	0	s_2	s_4	C	B	C	C	B	C	C
s_4	0	0	s_5	s_3	C	B	C	C	B	C	C
s_5	1	0	s_2	s_5	B	B	B	B	B	B	B
s_6	1	0	s_1	s_6	B	A	B	D	A	D	D

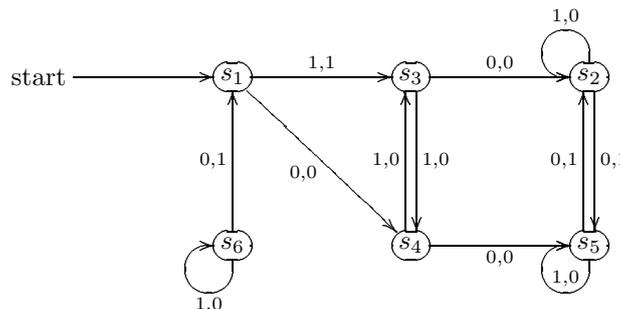
It follows that the process ends. We now choose one state from each equivalence class to obtain the simplified transition table as earlier.

6.7. Unreachable States

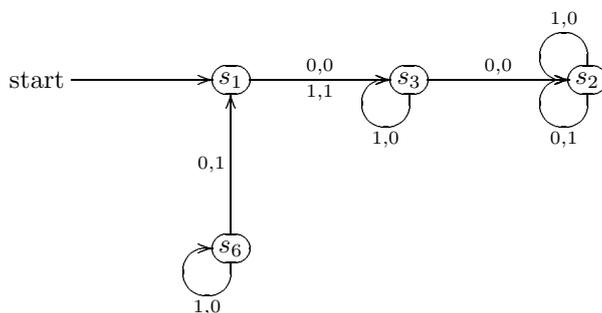
There may be a further step following minimization, as it may not be possible to reach some of the states of a finite state machine. Such states can be eliminated without affecting the finite state machine.

We shall illustrate this point by considering an example.

EXAMPLE 6.7.1. Note that in Example 6.6.1, we have not included any information on the starting state. If s_1 is the starting state, then the original finite state machine can be described by the following state diagram:



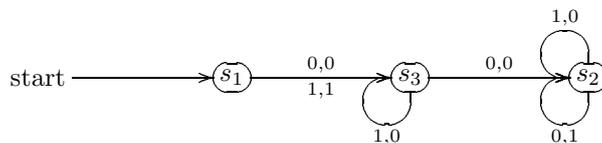
After minimization, the finite state machine can be described by the following state diagram:



It is clear that in both cases, if we start at state s_1 , then it is impossible to reach state s_6 . It follows that state s_6 is essentially redundant, and we may omit it. As a result, we obtain a finite state machine described by the transition table

	ω		ν	
	0	1	0	1
$+s_1+$	0	1	s_3	s_3
s_2	1	0	s_2	s_2
s_3	0	0	s_2	s_3

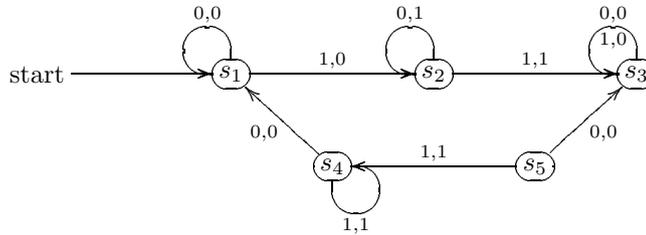
or the following state diagram:



States such as s_6 in Example 6.7.1 are said to be unreachable from the starting state, and can therefore be removed without affecting the finite state machine, provided that we do not alter the starting state. Such unreachable states may be removed prior to the Minimization process, or afterwards if they still remain at the end of the process. However, if we design our finite state machines with care, then such states should normally not arise.

PROBLEMS FOR CHAPTER 6

1. Consider the finite state machine $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$, where $\mathcal{I} = \mathcal{O} = \{0, 1\}$, described by the state diagram below:

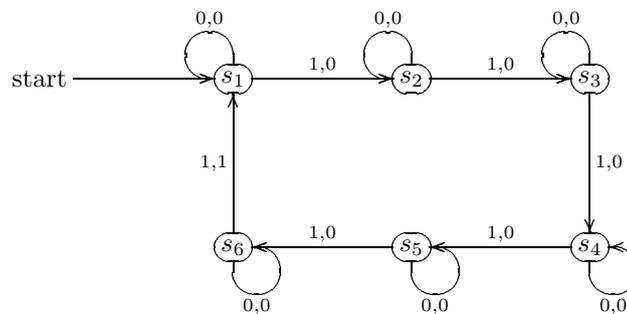


- a) What is the output string for the input string 110111? What is the last transition state?
 - b) Repeat part (a) by changing the starting state to s_2, s_3, s_4 and s_5 .
 - c) Construct the transition table for this machine.
 - d) Find an input string $x \in \mathcal{I}^*$ of minimal length and which satisfies $\nu(s_5, x) = s_2$.
2. Suppose that $|\mathcal{S}| = 3, |\mathcal{I}| = 5$ and $|\mathcal{O}| = 2$.
- a) How many elements does the set $\mathcal{S} \times \mathcal{I}$ have?
 - b) How many different functions $\omega : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{O}$ can one define?
 - c) How many different functions $\nu : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{S}$ can one define?
 - d) How many different finite state machines $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$ can one construct?

3. Consider the finite state machine $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$, where $\mathcal{I} = \mathcal{O} = \{0, 1\}$, described by the transition table below:

	ω		ν	
	0	1	0	1
s_1	0	0	s_1	s_2
s_2	1	1	s_1	s_2

- a) Construct the state diagram for this machine.
 - b) Determine the output strings for the input strings 111, 1010 and 00011.
4. Consider the finite state machine $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$, where $\mathcal{I} = \mathcal{O} = \{0, 1\}$, described by the state diagram below:



- a) Find the output string for the input string 0110111011010111101.
- b) Construct the transition table for this machine.
- c) Determine all possible input strings which give the output string 0000001.
- d) Describe in words what this machine does.

5. Suppose that $\mathcal{I} = \mathcal{O} = \{0, 1\}$. Apply the Minimization process to each of the machines described by the following transition tables:

	ω		ν	
	0	1	0	1
s_1	0	0	s_6	s_3
s_2	1	0	s_2	s_3
s_3	1	1	s_6	s_4
s_4	0	1	s_5	s_2
s_5	0	1	s_4	s_2
s_6	0	0	s_2	s_6
s_7	0	1	s_3	s_8
s_8	0	0	s_2	s_7

	ω		ν	
	0	1	0	1
s_1	0	0	s_6	s_3
s_2	0	0	s_3	s_1
s_3	0	0	s_2	s_4
s_4	0	0	s_7	s_4
s_5	0	0	s_6	s_7
s_6	1	0	s_5	s_2
s_7	1	1	s_4	s_1
s_8	0	0	s_2	s_4

6. Consider the finite state machine $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$, where $\mathcal{I} = \mathcal{O} = \{0, 1\}$, described by the transition table below:

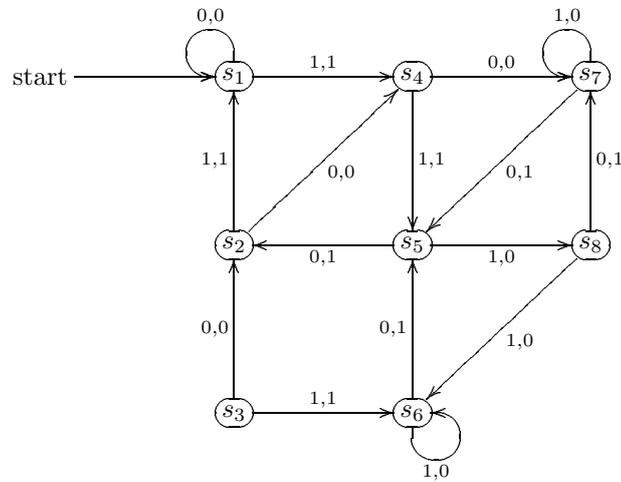
	ω		ν	
	0	1	0	1
s_1	1	1	s_2	s_1
s_2	0	1	s_4	s_6
s_3	0	1	s_3	s_5
s_4	0	0	s_1	s_2
s_5	0	1	s_5	s_3
s_6	0	0	s_1	s_2

- Find the output string corresponding to the input string 010011000111.
 - Construct the state diagram corresponding to the machine.
 - Use the Minimization process to simplify the machine. Explain each step carefully, and describe your result in the form of a transition table.
7. Consider the finite state machine $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$, where $\mathcal{I} = \mathcal{O} = \{0, 1\}$, described by the transition table below:

	ω		ν	
	0	1	0	1
s_1	1	1	s_2	s_1
s_2	0	1	s_4	s_6
s_3	0	1	s_3	s_5
s_4	0	0	s_1	s_2
s_5	0	1	s_5	s_3
s_6	0	0	s_1	s_2
s_7	0	0	s_1	s_2
s_8	0	1	s_5	s_3
s_9	0	0	s_1	s_2

- Find the output string corresponding to the input string 011011001011.
- Construct the state diagram corresponding to the machine.
- Use the Minimization process to simplify the machine. Explain each step carefully, and describe your result in the form of a transition table.

8. Consider the finite state machine $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, \omega, \nu, s_1)$, where $\mathcal{I} = \mathcal{O} = \{0, 1\}$, described by the state diagram below:



- Write down the output string corresponding to the input string 001100110011.
 - Write down the transition table for the machine.
 - Apply the Minimization process to the machine.
 - Can we further reduce the number of states of the machine? If so, which states can we remove?
9. Suppose that $\mathcal{I} = \mathcal{O} = \{0, 1\}$.
- Design a finite state machine that will recognize the 2-nd, 4-th, 6-th, etc, occurrences of 1's in the input string (e.g. 0101101110).
 - Design a finite state machine that will recognize the first 1 in the input string, as well as the 2-nd, 4-th, 6-th, etc, occurrences of 1's in the input string (e.g. 0101101110).
 - Design a finite state machine that will recognize the pattern 011 in the input string.
 - Design a finite state machine that will recognize the first two occurrences of the pattern 011 in the input string.
 - Design a finite state machine that will recognize the first two occurrences of the pattern 101 in the input string, with the convention that 10101 counts as two occurrences.
 - Design a finite state machine that will recognize the first two occurrences of the pattern 0100 in the input string, with the convention that 0100100 counts as two occurrences.
10. Suppose that $\mathcal{I} = \mathcal{O} = \{0, 1\}$.
- Design a finite state machine that will recognize the pattern 10010.
 - Design a finite state machine that will recognize the pattern 10010, but only when the last 0 in the sequence pattern occurs at a position that is a multiple of 5.
 - Design a finite state machine that will recognize the pattern 10010, but only when the last 0 in the sequence pattern occurs at a position that is a multiple of 7.
11. Suppose that $\mathcal{I} = \{0, 1, 2\}$ and $\mathcal{O} = \{0, 1\}$.
- Design a finite state machine that will recognize the 2-nd, 4-th, 6-th, etc, occurrences of 1's in the input string.
 - Design a finite state machine that will recognize the pattern 1021.
12. Suppose that $\mathcal{I} = \mathcal{O} = \{0, 1, 2\}$.
- Design a finite state machine which gives the output string $x_1x_1x_2x_3 \dots x_{k-1}$ corresponding to every input string $x_1 \dots x_k$ in \mathcal{I}^k . Describe your result in the form of a state diagram.
 - Design a finite state machine which gives the output string $x_1x_2x_2x_3 \dots x_{k-1}$ corresponding to every input string $x_1 \dots x_k$ in \mathcal{I}^k . Describe your result in the form of a state diagram.

13. Suppose that $\mathcal{I} = \mathcal{O} = \{0, 1, 2, 3, 4, 5\}$.
- a) Design a finite state machine which inserts the digit 0 at the beginning of any string beginning with 0, 2 or 4, and which inserts the digit 1 at the beginning of any string beginning with 1, 3 or 5. Describe your result in the form of a transition table.
 - b) Design a finite state machine which replaces the first digit of any input string beginning with 0, 2 or 4 by the digit 3. Describe your result in the form of a transition table.