4. INTRODUCTION TO FINITE STATE MACHINES

§4.1. Cyclops, The Simplest Possible Computer

We now begin our study of the Theory of Computation. Here we are not interested in the latest software or hardware, knowledge that changes by the month and which becomes obsolete after only a few years. Instead we are going to ask some rather deep questions about the nature of computation. While it is clear that this should be valuable background knowledge for the professional computer scientist, it should be pointed out that it also has relevance for the student of mathematics. After all, a large part of mathematics involves computation, and algorithms (procedures for computing something) have been around for thousands of years.

Questions about the fundamental nature of computation were asked, not by computer scientists, but by mathematicians and when such questions began to be asked (in the first half of the 20th century) there were no such people as computer scientists and no such machines as computers. Indeed it was the pioneering work of mathematicians such as Alan Turing that helped to give birth to the computer.

Now a modern computer is an exceedingly complicated machine. In order to understand its essential nature we must strip away all this complexity and consider the simplest possible computer. We shall call this simplest-of-all-possible-computers, "Cyclops". [Cyclops was the name given to a character from Greek mythology with only one eye, placed in the middle of his forehead. You will see shortly why this name is quite appropriate for this most primitive machine.]

There are four essential features of any computing device:

* a mechanism for input
* a mechanism for output
* memory
* programs.

Input can come through a variety of devices such as CD-ROMs or wireless modems, but the most familiar input device is the keyboard. We will provide Cyclops with a keyboard with just one key with the numeral "1" engraved on it. So a typical input to Cyclops can only be a string of 1's, such as 11111. This means that we cannot use binary notation for numbers. Instead, to input the number n we will have to press the "1" key n times.

Output can also come through a variety of channels, but probably the most familiar is the video display unit, or computer screen. Our simplest-of-all possible computers will have to have a monochrome display. On a typical LCD screen there are many thousands of tiny "pixels" – points that can be either on or off (lit or dark). Cyclops will have to be content with just a single pixel screen – or equivalently, a single light bulb which can either be ON or OFF.

In all computers a part of the RAM (random access memory) is devoted to providing the screen output. (In the early days of home computers when there was very limited memory

programmers were forced sometimes to make use of this memory for their calculations and so while the program was running all sorts of "garbage" would appear on the screen until the final output was displayed. Today memory is measured in gigabytes and each gigabyte consists of a very large number of tiny units of memory called "bits". One bit is the amount of memory contained in a light switch that "remembers" whether it is ON or OFF. In order to create the simplest-of-all-possible-computers we will insist that the single light bulb that Cyclops has for its display has to double up as a one bit memory.

Finally there is the program. Normally computers use part of their memory to store a program, but this is a bit much to ask of our single-bit midget! Instead we will "hard-wire" the program into Cyclops in much the same way that there is a program hard-wired into the micro-processor that controls an automatic washing machine. This means, of course, that unless we re-wire him, Cyclops will be a single program computer. So our minimal computer is a one bit, single program machine with just one key and a single light-bulb as the display!

At any given stage the light bulb is either on or off. As the keys are pressed, the program will determine whether the bulb should be on or off. Since our intention is to make Cyclops a deterministic machine, the decisions made by the program must be determined solely by two things – which key has been pressed and what state the machine is currently in. The decision rule can be set out in a table of the following form:

	1
OFF	
ON	

Each of these two cells needs to contain the name of the next state into which the machine goes, that is each is either an "ON" or an "OFF".

Since there are two possibilities for each cell there are 4 combinations altogether. Each of these can be thought of as a program that can be hard-wired into Cyclops. Here is one such program.

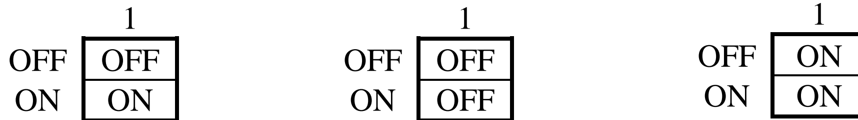
Example 1:

	1
OFF	ON
ON	OFF

When the light is OFF a key-press will turn it ON. When it is ON, pressing the key turns it OFF. This program will cause Cyclops to repeatedly switch the output bulb on and off every time the "1" key is pressed.

This very simple computer, running this very simple program, is in fact actually manufactured, though not as a computing device. Rather it's a toggle light switch such as you get on many desk lamps. Push the button, and if the light is OFF it then goes ON. Push it again and the light goes OFF. Many years ago such switches were connected to long cords from ceilings and each time you pulled on the cord the ceiling light would change its state.

Now you may never have thought of such a mechanism as a primitive computer, but it can in fact compute. We can use it to compute whether a given number n is odd or even by examining the effect of n pushes of the key. We first reset the machine by switching the light off. Then we push the button n times. If, when we've finished, the light is ON, then n is odd. If it ends up being OFF, the number is even. Altogether there are four different "programs" that Cyclops can be given. One was given in Example 1. The other three are:



These three are not nearly so useful as the light-switch model. In the first one the input has no effect whatsoever, just as if the key was a dummy one – not connected to anything. The other two can change the state, but the effect can't be reversed. The buttons on a lift work like this last one. Once they're ON they can't be switched off (until they are reset when you reach the required floor).

So we have learnt all there is to know about 1-input 2-state machines, which is not much! So let us generalise Cyclops to a machine with m input keys and n states.

§4.2. Finite State Machines

A **finite state machine** consists of the following:

(1) a set I called the **input alphabet**;
(2) a set S whose elements are called **states**;
(3) a function $T:S \times I \to S$ called the **transition function**;
(4) a particular element, $s_0 \in S$ called the **initial state**;

The functioning of the machine is as follows:
The machine starts in the initial state s_0. The input is a string of characters from the input alphabet which are read one at a time (from the left). At each stage the machine is in some state $s \in S$. If the machine is in state s, and the next input character is $c \in I$, the machine moves to state T(s, c) and awaits the next input character. The process continues in this way until all the input characters have been processed.

Example 2:
I = {1, 2}; S = {A, B, C, D}; s_0 = A.
T is given by the table:

	1	2
→A	B	C
B	C	D
C	D	A
D	A	C

Suppose the input to this machine is 1121221121. The successive transitions of the machine are:

$$A \xrightarrow{1} B \xrightarrow{1} C \xrightarrow{2} A \xrightarrow{1} B \xrightarrow{2} D \xrightarrow{2} C \xrightarrow{1} D \xrightarrow{1} A \xrightarrow{2} C \xrightarrow{1} D$$

Although we described it abstractly, this particular machine can have a very familiar interpretation. Imagine that you are the operator of an amusement park ride and you have to fill four-seater cars from a succession of couples and single customers.

Your instructions are to wait until the car is full before letting it go. However if a couple arrives, rather than splitting them, you let the car go with just 3 passengers and put the couple in the next one.

The input consists of the customers who arrive in 1's and 2's. The four states, A to D, correspond to the four possible situations you can have with the current car. State A is

where you have a completely empty car and states B, C and D are where there are one, two or three passengers waiting in the car. Examine the state table to see that you agree that it accurately describes the situation.

Both examples 1 and 2 serve some other purpose than computation. The fact that a light switch can distinguish between odd and even numbers is not the purpose for which it was made, and the operator of the ghost train ride certainly doesn't see himself as a cog in some giant, but rather feeble, computer.

In what follows we shall concentrate on finite state machines as devices to perform some sort of computation. The physical nature of the states won't concern us, only the abstract movement from state to state.

In practice, finite state machines are implemented electronically in one of two ways. We can build an electronic chip, incorporating the logic of the machine. This is done where the machine is part of a piece of equipment that is designed to react to external input in some definite way. Or we can incorporate the finite state machine within some computer program, where the computer simulates to machine.

In fact a computer itself is a finite state machine. The set of states is the set of all possible combinations of bits of internal memory. This is huge, but finite.

§4.3. Mealy Machines

To be of any use in computation a finite state machine must have some form of output. There are several ways we can provide that output. An obvious way is to have the machine print it (on a screen, or on paper, or perhaps as a sequence of tones). We need an output alphabet, O, that may or may not be the same as the input alphabet. Every time the machine reads a character from the input, it outputs a character as well as changing its state.

To describe the output of such a machine we can have an additional function that assigns to each combination of state and input character, an output character. In other words we have a function $P: S \times I \rightarrow O$. When the machine is in state s and reads the character c, the output is $P(s, c)$.

A machine of this type is known as a **Mealy Machine**. We can describe a Mealy machine by a pair of tables. The state table provides the new state, for each combination of state and character and the output table provides the character that is output at each stage. The initial state is indicated by putting a short arrow in front of that state in the left-hand column.

Example 3: The following Mealy machine echoes the input after a two step delay. The first two output characters are both 0's. Thereafter the output at each stage is the input from two steps earlier. The last two input characters are ignored.

$I = O = \{0, 1\}$; $S = \{A, B, C, D\}$ with $s_0 = A$.
The meaning of these states will be as follows:

A means the next two output characters are to be 0, 0
B means the next two output characters are to be 0, 1
C means the next two output characters are to be 1, 0
D means the next two output characters are to be 1, 1

Giving an interpretation to the states is not a part of the description of the machine. One can "operate" the machine mindlessly without such knowledge. However when designing finite state machines, and understanding their function, it is quite important to be able to attach a meaning to each state.

The transition and output tables for this machine are as follows (we indicate the fact that A is the initial state by putting an arrow in front of it):

	State Table		Output Table	
	0	**1**	**0**	**1**
→A	A	B	0	0
B	C	D	0	0
C	A	B	1	1
D	C	D	1	1

So if the input is 111001111 the output will be 001110011 as shown below:

input		1		1		1		0		0		1		1		1		1	
states	A	→	B	→	D	→	D	→	C	→	A	→	B	→	D	→	D	→	D
		↓		↓		↓		↓		↓		↓		↓		↓		↓	
output		0		0		1		1		1		0		0		1		1	

The effect is to print two initial 0's and then to repeat the input with a two-step delay. Of course because the output has the same length as the input the last two input characters will be ignored.

§4.4. Moore Machines

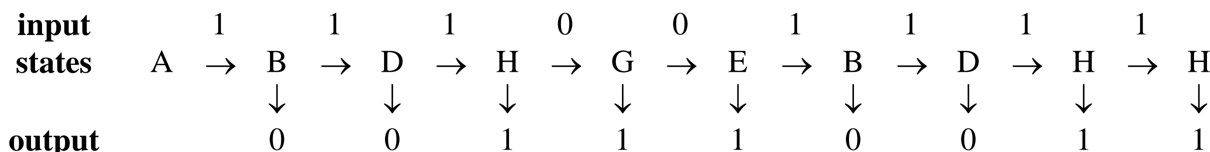
An alternative arrangement for output is to have the machine print a certain output character as it enters a state. These machines are called **Moore machines**. The output is attached to the states rather than to the transitions.

Formally, the output of a Moore machine is described by a function P: S → O, where as before, O is the output alphabet.

Example 4:
The following Moore machine is also a two-step delay machine, behaving equivalently to the Mealy machine in example 3.
I = O ={0, 1}; S = {A, B, C, D} with s_0 = A.

	T		P	INTERPRETATION
	0	**1**		**last 3 inputs**
→A	A	B	0	0 0 0
B	C	D	0	0 0 1
C	E	F	0	0 1 0
D	G	H	0	0 1 1
E	A	B	1	1 0 0
F	C	D	1	1 0 1
G	E	F	1	1 1 0
H	G	H	1	1 1 1

Using the same input string as before we get the same output. The only difference is that the output responds only to the state being entered.



It is a relatively routine task to convert a Mealy machine to a Moore machine and vice versa.

§4.5. Finite State Acceptors

Even the Moore machine is more than we need here. Our focus on finite state machines will be as **acceptors** for languages. Given a language, on a given alphabet, we would like to find a machine that decides whether or not a given string belongs to the language.

Here the output is a simple "YES" or "NO". We are not interested in partial results. We wait for the machine to read the entire input string and only then are we interested in the answer. In a **finite state acceptor (FSA)** the output is determined by selecting a subset of states as the **accepting states**. If the machine ends in an accepting state, after having read the entire string, we say that the string has been accepted by the machine. Otherwise it's rejected.

Every Moore machine with output set {0,1} can be considered as an FSA by taking the accepting states to be those for which the output is a "1". And every FSA can be considered as a Moore machine with the output alphabet {0, 1} simply by assuming that the machine prints a "1" whenever it enters an accepting state and "0" when it enters any other state.

The slight modification to the way we present it as a finite state acceptor is to replace the 1's in the final column by *'s and the 0's by blanks, so that the accepting states are those with an asterisk against them.

Example 5:
Writing the Moore machine in example 4 as a finite state acceptor we have:

	0	**1**	
→A	A	B	
B	C	D	
C	E	F	
D	G	H	
E	A	B	*
F	C	D	*
G	E	F	*
H	G	H	*

As we said earlier, our main focus will be on finite state acceptors. Now corresponding to every FSA there is a language, the set of all input strings that are accepted by the machine. We say that an FSA **accepts** a language, L, if it accepts every string in L and rejects all others.

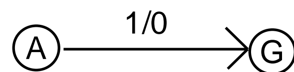
An important question is "can every language be accepted by an FSA?" The answer is "no". Only for certain languages does there exist a corresponding FSA. Which ones? We shall have to wait till a later chapter for an answer.

Example 6:
The language accepted by the above FSA is the set of all strings whose third last symbol is a "1". This language can be described by the regular expression (0+1)*1(00+01+10+11).

§4.6. State Diagrams

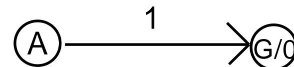
It is often convenient to depict a finite state machine pictorially. This is done by drawing a small circle for each state, with the name of the state inside, and drawing arrows connecting the states to depict the transitions. The input character that gives rise to a transition is written beside the corresponding arrow. We indicate the initial state by drawing a short arrow pointing to it. Output is depicted in the following ways:

Mealy Machines:
The output is written beside the input character for each transition as follows:



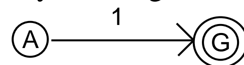
This indicates that if the machine is in state A and receives input "1" then it outputs a "0" and moves to state G.

Moore Machines:
The output is attached to the name of the state as follows:



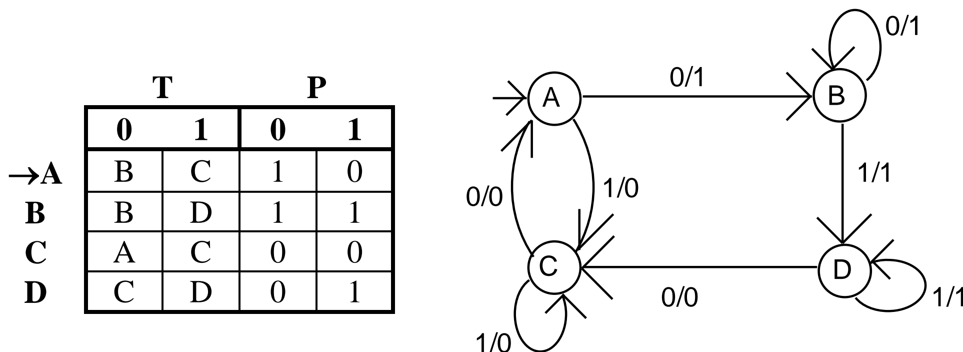
This indicates that if the machine is in state A and receives input 1, it moves to state G and outputs a 0.

FSA's:
The accepting states are distinguished by drawing a double ring around them as follows:



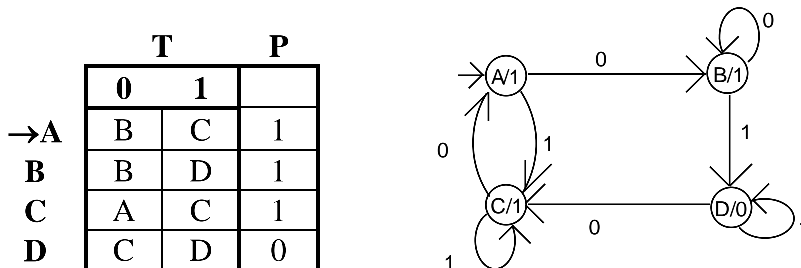
This indicates that if the machine is in state A and receives input 1, it moves to state G. If that was the last character of the input string, the string would be accepted.

Example 7: The following table and diagram represent the same Mealy machine.



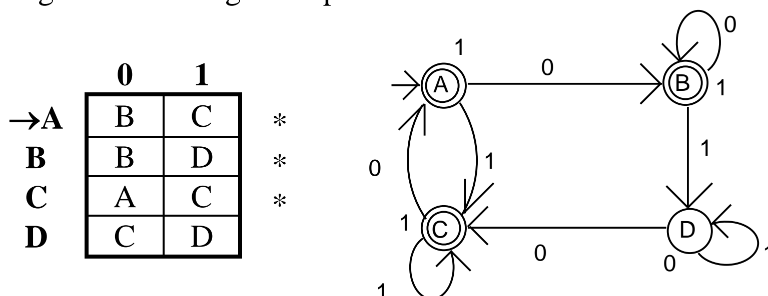
	T		**P**	
	0	**1**	**0**	**1**
→A	B	C	1	0
B	B	D	1	1
C	A	C	0	0
D	C	D	0	1

Example 8:
The following table and diagram represent the same Moore machine.



	T		P
	0	**1**	
→A	B	C	1
B	B	D	1
C	A	C	1
D	C	D	0

Example 9:
The following table and diagram represent the same FSA.

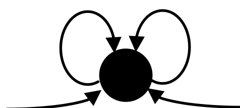


	0	**1**	
→A	B	C	*
B	B	D	*
C	A	C	*
D	C	D	

§4.7. Black Holes

With an FSA we often arrive at a situation where we can reject the string without needing to see the rest of it. For example if we have an acceptor for the language (10)* we can reject the string 101101010101 as soon as we read the fourth character. Beyond this point, we don't care what we read. The fate of the string is sealed!

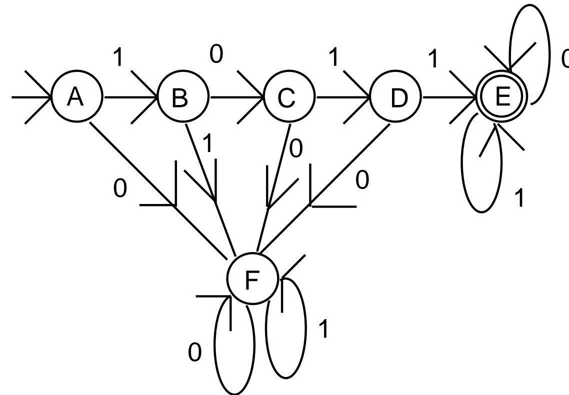
The most convenient way of dealing with this is to have a special state that represents rejection before the input is complete. The transitions coming out of such a state should lead right back again. Therefore, once the machine gets into this state it can never get out. For this reason it is sometimes known as a "black hole", by analogy with those regions of outer space with such a high concentration of gravitational force that nothing that enters can ever escape.



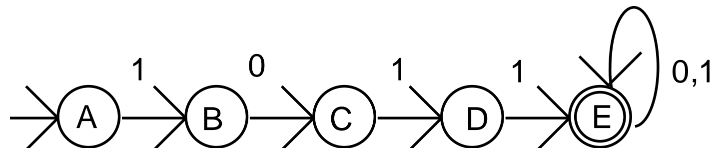
A typical FSA will have a black hole (only one is ever necessary) with many transitions into it. To include the black hole in a state diagram with all its entering arrows, would make the diagram look excessively cluttered. For this reason we generally omit the black hole from state diagrams. Therefore, whenever there is no arrow emerging from a state for a particular input character, it's assumed that there is an invisible arrow leading to the black hole. And of course the black hole is never an accepting state.

Example 10:
Design an FSA to accept the language consisting of all binary strings that begin with 1011. In other words design an FSA for the language given by the regular expression 1011(0+1)*.

Solution:



The state F is clearly acting as a black hole. Therefore we can simplify the diagram by deleting it. Any transitions that are not shown are assumed to lead to this black hole. Another simplification is to combine arrows that start and finish with the same state.



NOTES:

(1) Do not confuse the 0,1 notation with the 0/1 notation that we used for Mealy machines. Some books use 0,1 in such situations so there is some danger of confusion. For us, 0/1 means "read 0 and write 1" while 0,1 means "read either 0 or 1".

(2) Note that E is not a black hole because it is an accepting state.

(3) We omit a black hole in a state diagrams but it must be included in a state table.

EXERCISES FOR CHAPTER 4

EXERCISES 4A (Operating FSMs)

Ex 4A1: Find the output of the following Mealy Finite State Machine if the input is WOOLOOMOOLOO.
Input alphabet = {W, L, M, O}; Output alphabet = {a, n, t};

	TRANSITION				**OUTPUT**			
	W	L	M	O	W	L	M	O
→0	0	2	1	2	t	a	a	a
1	2	0	2	1	a	t	t	a
2	1	0	2	1	n	n	n	n

Ex 4A2: Operate the following binary FSA with the following input strings:
$\alpha =$ 11100; $\beta =$ 011010111; $\gamma =$ 1100111010011. Which strings are accepted?

	0	1	
→A	B	C	
B	C	B	*
C	A	C	

Ex 4A3: Operate the following Moore machine with input 1101100. What is the output?

	0	1	
→A	B	C	X
B	C	B	X
C	A	C	Y

Ex 4A4: Operate the following Mealy machine with input 001001001. What is the output?

	T		P	
	0	1	0	1
→A	B	C	0	1
B	C	B	1	1
C	A	C	1	0

Ex 4A5: What is the output if the string 111001 is input to the following Mealy Finite State Machine?

	Transition		Output	
	0	1	0	1
→A	B	C	0	0
B	A	B	1	0
C	B	C	0	1

Ex 4A6: What is the output if the string 111001 is input to the following Mealy Finite State Machine?

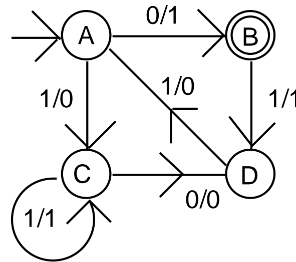
	Transition		Output	
	0	1	0	1
→A	B	A	1	0
B	C	A	1	1
C	B	C	0	1

Exercises 4B (State Diagrams)

Ex 4B1: Draw the state diagram for the following FSA:

	0	1	
→A	B	C	
B	C	E	*
C	A	C	
D	A	B	*
E	D	E	

Ex 4B2: Construct the state and output tables for the following Mealy machine:



Ex 4B3: Draw the state diagram for the following Finite State Acceptor:

	0	1	
→A	D	F	
B	C	E	
C	B	A	*
D	C	B	
E	A	E	*
F	F	A	

Ex 4B4: Draw a state diagram for the following Finite State Acceptor.

	0	1	
→A	B	C	*
B	E	A	
C	C	E	
D	B	D	*
E	A	A	

Ex 4B5: Draw a state diagram for the following Finite State Acceptor.

	0	1	
→A	A	D	*
B	C	E	
C	C	A	
D	D	C	
E	B	E	*

Ex 4B6: Draw a state diagram for the following Mealy machine and give the output string for the input 10110.

	Transitions		**Output**	
	0	**1**	**0**	**1**
→A	C	D	0	0
B	C	B	1	0
C	A	D	0	1
D	D	B	1	1

Exercises 4C (Designing FSAs)

Ex 4C1: Design an FSA, on the alphabet {0,1}, that will accept all strings that start with 11 and contain the substring 00.

Ex 4C2: Design an FSA that accepts the language 110 + 11.

Ex 4C3: Design a FSA that accepts the language λ + 110.

Ex 4C4: Design an FSA, on the alphabet {A, F, S}, that will accept all strings that contain the substring FSA.

Ex 4C5: Design an FSA with 3 states that accepts the language (110 + 11)*.

Ex 4C6: Design an FSA to accept the language of all binary strings whose length is at least 2.

Ex 4C7: Design a Finite State Acceptor to accept the language on {0, 1} consisting of all strings of length at least 2 that start and finish with the same character.
[HINT: Use 7 states A to G. States A, B and C correspond to the three strings of length less than 2 and states D, E, F, G correspond to the 4 possible combinations of first and last characters read.]

Ex 4C8: (a) Design an FSA to accept the language of all binary strings in which all the 0's precede all the 1's.
(b) Design an FSA to accept the language $\{\alpha \mid \exists n[(|\alpha| = 2n) \wedge \exists\beta[\alpha = 110\beta]\}$.
The universe for α, β is the set of all binary strings and for n it is the set of non-negative integers.

Ex 4C9: Design a Finite State Acceptor to accept the language of all binary strings that do not include the substring 1011.

Ex 4C10: Design a Finite State Acceptor to accept the language of all binary strings that start with 10 but do not include the substring 1011. Express your answer as a table.

Ex 4C11: (This assumes you know how tennis is scored.)
Design a Mealy machine to keep the score in a tennis match as follows. The input characters are S = "server wins the point" and R = "receiver wins the point". The output characters are S = "server wins the game" and R = "receiver wins the game". The states are the possible scores within a game. (Don't bother with keeping track of the game scores.) Give your answer as a state table. Code the states (scores) as follows:
A = "love all", B = "love-15", C = "love-30", D = "love-40",
E = "15-love", F = "15 all", G = "15-30", H = "15-40",
I = "30-love", J = "30-15", K = "30 all", L = "30-40",
M = "40 love", N = "40-15", O = "40-30",
P = "deuce", Q = "advantage server", R = "advantage receiver".

Exercises 4D (Analysis of FSAs)

Ex 4D1: (a) Show that there are 64 binary FSA's with two states A, B where A is the initial state.
(b) One of these FSA's accepts 000 and 10 but rejects 00 and 0001. Which one? Express your answer both as a table and as a state diagram.

Ex 4D2: Let F be the set of all 3-state binary FSA's with states {A, B, C} where A is the initial state.
(a) How many FSA's are there altogether in F?
(b) How many of these accept the string 11?
(c) How many FSA's in F reject 1111?
(d) How many FSA's in F accept 11 but reject 1111?

SOLUTIONS FOR CHAPTER 4

Ex 4A1:

$$\longrightarrow 0 \xrightarrow[t]{W} 0 \xrightarrow[a]{O} 2 \xrightarrow[n]{O} 1 \xrightarrow[t]{L} 0 \xrightarrow[a]{O} 2 \xrightarrow[n]{O} 1 \xrightarrow[t]{M} 2 \xrightarrow[n]{O} 1 \xrightarrow[a]{O} 1 \xrightarrow[t]{L} 0 \xrightarrow[a]{O} 2 \xrightarrow[n]{O} 1$$

Ex 4A2: α and γ are accepted.

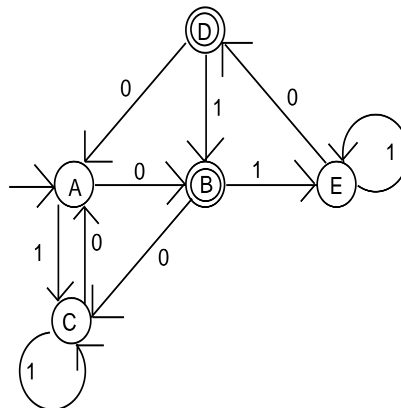
Ex **4A3:** YYXYYXX.

Ex 4A4: 010101111

Ex 4A5: 011010

Ex 4A6: 000111

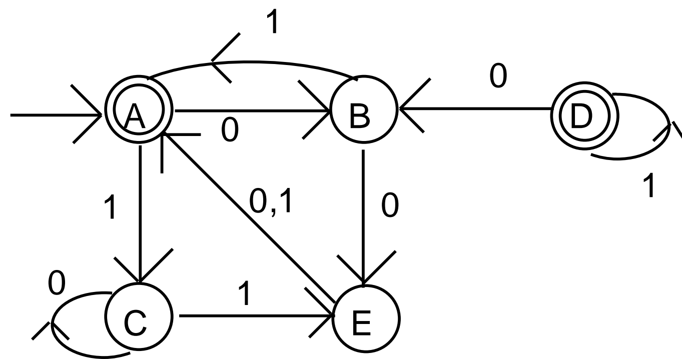
Ex 4B1:



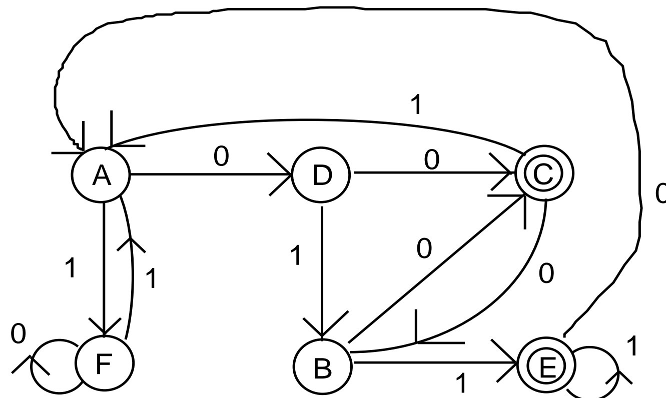
Ex 4B2:

	T		P	
	0	**1**	**0**	**1**
→A	B	C	1	0
B		D		1
C	D	C	0	1
D		A		0

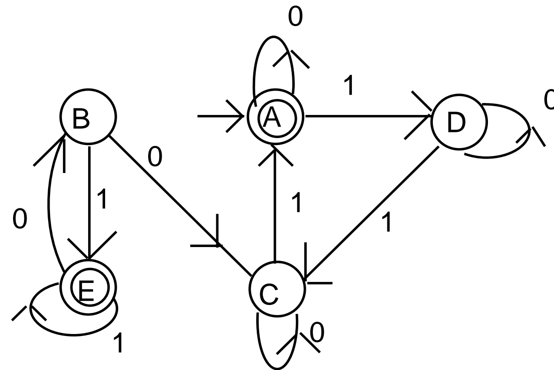
Ex 4B3:



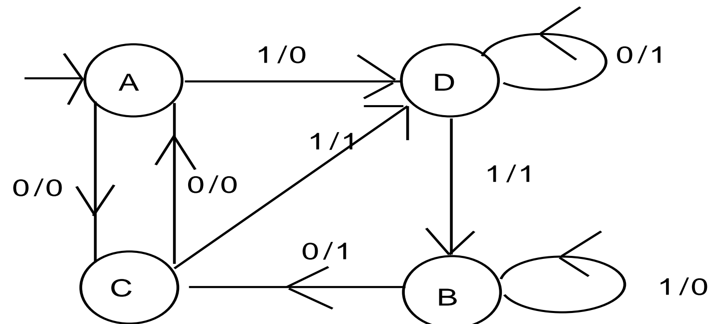
Ex 4B4:



Ex 4B5:

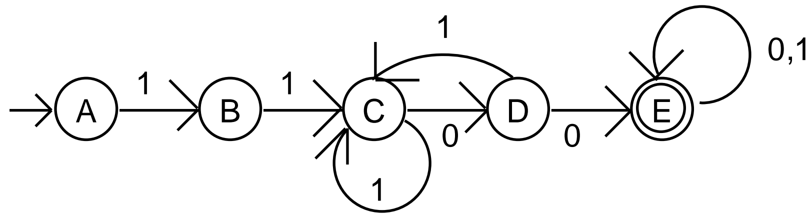


Ex 4B6:

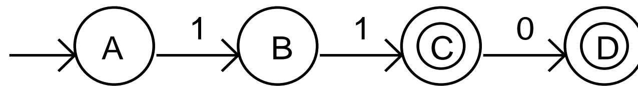


01101

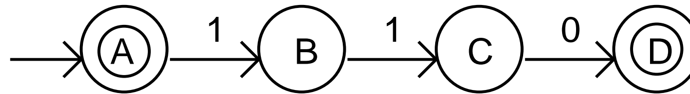
Ex 4C1:



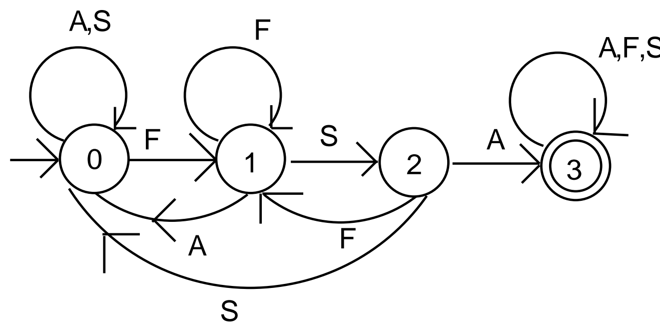
Ex 4C2:



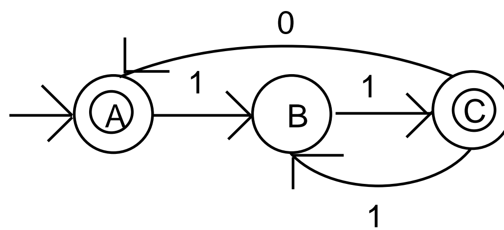
Ex 4C3:



Ex4C4:



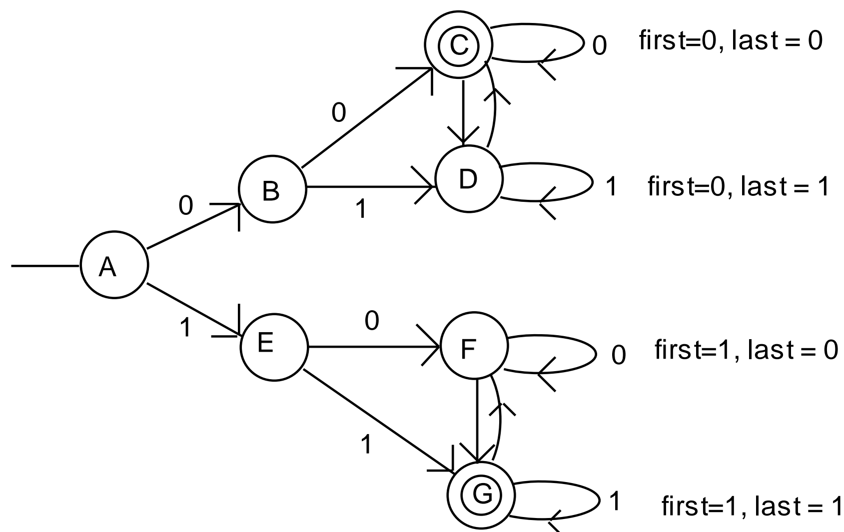
Ex 4C5:



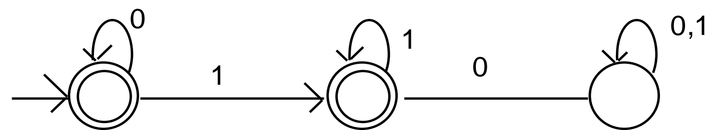
Ex 4C6:

	0	**1**		
→A	B	B		start
B	C	C		1 character
C	C	C	*	2 characters

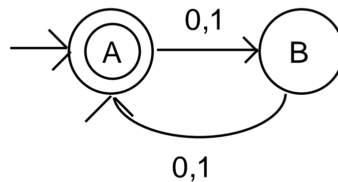
Ex 4C7:



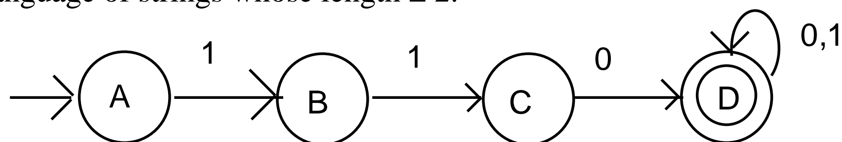
Ex 4C8: (a) This is the same as the language of all binary strings which do not contain 10. Hence a FSA is:



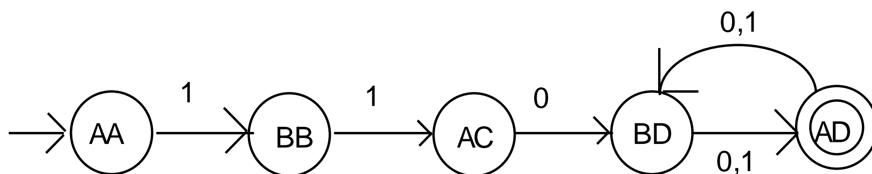
(b) The language is the set of all binary strings α of even length which start with 110. We first design separate machines for each of these conditions.



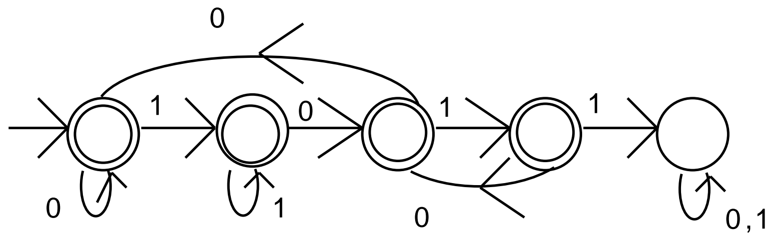
accepts the language of strings whose length ≥ 2.



accepts the language of all strings starting with 110.
We therefore construct an FSA with 8 states: AA, AB, AC, AD, ... , BD. Actually we only need 5 states since the other 3 are inaccessible.



Ex 4C9:



Ex 4C10:

	0	1	
→A	G	B	
B	C	G	
C	C	D	*
D	E	D	*
E	C	F	*
F	E	G	*
G	G	G	

Ex 4C11:

	T		P	
	S	R	S	R
→A	E	B		
B	F	C		
C	G	D		
D	H	A		R
E	I	F		
F	J	G		
G	K	H		
H	L	A		R
I	M	J		
J	N	K		
K	O	L		
L	P	A		R
M	A	N	S	
N	A	O	S	
O	A	P	S	
P	Q	R		
Q	A	P	S	
R	P	A		R

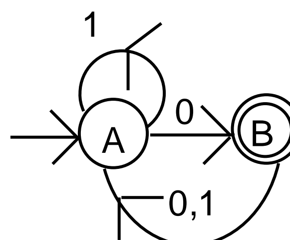
Ex 4D1: Each binary FSA with two states will correspond to a table:

	0	1
→A		
B		

Each entry in the body of the table will be either A or B, giving 2^4 possibilities. For each such table there are 4 possibilities for the set of accepting states: none, just A, just B or both. Thus there are $4 \times 2^4 = 64$ possible FSA's altogether.

(b) With a little bit of detective work we can see that the only FSA which accepts 000 and 10 but rejects 00 and 0001 is:

	0	1	
→**A**	B	A	
B	A	A	*



[For example if the (A, 0) entry was A, the FSA would accept all strings of 0's or rejected them all. That is why there is a B there. And if the entry below it had been a B the FSA would have dealt equally with 00 and 000. That is why there is an A. Since 00 will return us to state A this must be a rejecting state, and by a similar argument for 000, B must be accepting.]

Ex 4D2: (a) **5832**. There are 6 cells in the 3×2 state table, each of which is one of the 3 states, so there are 3^6 ways of filling out the table. There are 2^3 possibilities for the set of accepting states and so $3^6 \times 2^3$ FSA's in F.
(b) **2916**: For each of the 3^6 ways of filling out the state table the input string 11 will cause the FSA to terminate in one specific state. So for the FSA to accept 11 this state must be made accepting. There are 2^2 ways of assigning the remaining two states as accepting or rejecting. Hence there are $3^6 \times 2^2$ FSA's in F which accept 11.
(c) **2916**: by a similar argument.
(d) **108**: Let the state table for a typical FSA in F be:

	0	1
A		X
B		Y
C		Z

where X, Y, Z are states. If X = A, the FSA will end in A for both 11 and 1111. Yet one is accepted and the other rejected. So X = B or C.
Case 1: X = B. If Y = A or B, the FSA will end in the same state for the two strings 11 and 1111, a contradiction. Hence Y = C. If Z = B or C, the FSA will treat 11 and 1111 identically. So we must have Z = A. So in this case the state table must be:

	0	1
A		B
B		C
C		A

Clearly C must be accepting and B non-accepting. State A could be either. There are 3^3 ways of filling out the first column and for each, two choices for A's acceptance or non-acceptance. Hence there are 54 FSA's in this case.
Case 2: X = C. This is similar (with the roles of B, C being reversed) and so there are 54 FSA's in this case, giving a total of 108 FSA's which accept 11 and reject 1111.