Finite Automata

Let us begin by removing almost all of the Turing machine's power! Maybe then we shall have solvable decision problems and still be able to accomplish *some* computational tasks. Also, we might be able to gain insight into the nature of computation by examining what computational losses we incur with this loss of power.

If we do not allow writing or two-way operation of the tape head, we have what has been traditionally called a *finite automaton.* This machine is only allowed to read its input tape and then, on the basis of what it has read and processed, accept or reject the input. This restricted machine operates by:

a) Reading a symbol,
b) Transferring to a new instruction, and
c) Advancing the tape head one square to the right.

When it arrives at the end of its input it then accepts or rejects depending upon what instruction is being executed.

This sounds very simple. It is merely a one-way, semi-literate Turing machine that just decides membership problems for a living! Let us examine one. In order to depict one, all we need to do is jot down Turing machine instructions in one large table, leave out the write part (that was not a pun!), and add a note which indicates whether the machine should accept. Here is an example:

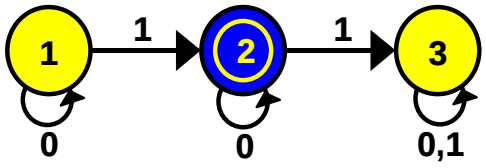
Instruction	*Read*	*Goto*	*Accept?*
1	0 1	same next	no
2	0 1	same next	yes
3	0 1	same same	no

Look closely at this machine. It stays on instruction one until it reads a one. Then it goes to instruction two and accepts any input unless another one arrives (the symbol 1 - not another input). If two or more ones appear in the input, then it ends up executing instruction three and does not accept when the input is over. And if no ones appear in the input the machine remains on instruction one and does not accept. So, this machine accepts only inputs that contain *exactly one 1.*

(**N.B.** Endmarkers are not needed since the machine just moves to the right. Accepting happens when the machine finishes the input while executing an instruction that calls for acceptance. Try this machine out on the inputs 000, 0100, 1000100, etc.)

Traditionally these machines have not had instructions but *states.* (Recall A. M. Turing's *states of mind*.) Another way to represent this same machine is to put the *next instruction* or *next state* or *goto* portion under the input in a table like that in figure 1.

There is another traditional method to describe finite automata which is extremely intuitive. It is a picture called a *state graph.* The states of the finite automaton appear as vertices of the graph while the transitions from state to state under inputs are the graph edges. The state graph for the same machine also appears in figure 1.



State	Input		Accept?
	0	1	
1	1	2	no
2	2	3	yes
3	3	3	no

Figure 1 - Finite Automaton Representations

(Note that the two circles that surround state two mean acceptance.)

Before continuing let's examine the computation of a finite automaton. Our first example begins in state one and reads the input symbols in turn changing states as necessary. Thus a computation can be characterized by a sequence of states. (Recall that Turing machine configurations needed the state plus the tape content. Since a finite automaton never writes, we always know what is on the tape and need only look at a state as a configuration.) Here is the sequence for the input 0001001.

Input Read:	**0**		**0**		**0**		**1**		**0**		**0**		**1**		
States:	**1**	→	**1**	→	**1**	→	**1**	→	**2**	→	**2**	→	**2**	→	**3**

Our next example is an elevator controller. Let's imagine an elevator that serves two floors. Inputs are calls to a floor either from inside the elevator or from the floor itself. This makes three distinct inputs possible, namely:

0 - no calls
1 - call to floor one
2 - call to floor two

The elevator itself can be going up, going down, or halted at a floor. If it is on a floor it could be waiting for a call or about to go to the other floor. This provides us with the six states shown in figure 2 along with the state graph for the elevator controller.

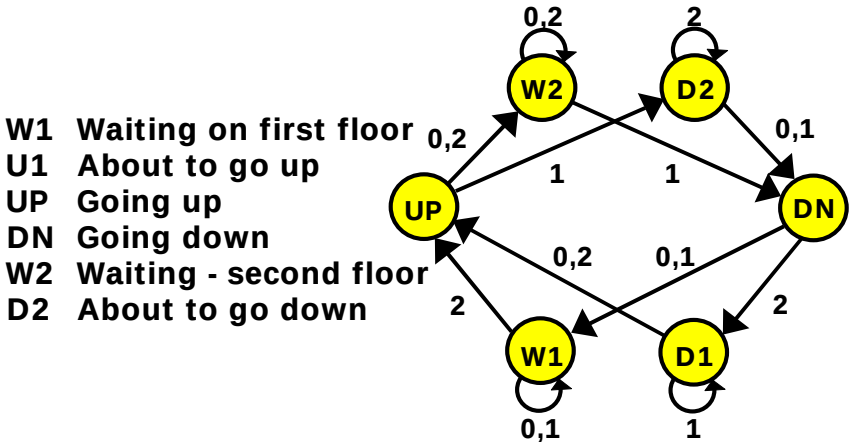


Figure 2 - Elevator Control

A state table for the elevator is provided below as table 1.

State	*Input*		
	none	call to 1	call to 2
W1 (wait on 1)	W1	W1	UP
U1 (start up)	UP	U1	UP
UP	W2	D2	W2
DN	W1	W1	U1
W2 (wait on 2)	W2	DN	W2
D2 (start down)	DN	DN	D2

Table 1 - Elevator Control

Accepting and rejecting states are not included in the elevator design because acceptance is not an issue. If we were to design a more sophisticated elevator, it might have states that indicated:

a) power failure,
b) overloading, or
c) breakdown

In this case acceptance and rejection might make sense.

Let us make a few small notes about the design. If the elevator is about to move (i.e. in state U1 or D2) and it is called to the floor it is presently on it will stay. (This may be good - try it next time you are in an elevator.) And if it is moving (up or down) and gets called back the other way, it remembers the call by going to the U1 or D2 state upon arrival on the next floor. Of course the elevator does not do things like open and close doors (these could be states too) since that would have added complexity to the design. Speaking of complexity, imagine having 100 floors.

That is our levity for this section. Now that we know what a finite automaton is, we must (as usual) define it precisely.

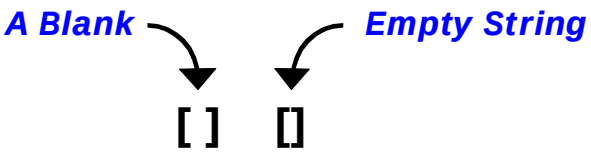
Definition. *A **finite automaton** M is a quintuple $M = (S,I,\delta,s_0,F)$ where:*
S is a finite set (of states)
I is a finite alphabet (of input symbols)
δ: $S \times I \rightarrow S$ (next state function)
$s_0 \in S$ (the starting state)
$F \subseteq S$ (the accepting states).

We also need some additional notation. The next state function is called the *transition function* and the accepting states are often called *final states.* The entire machine is usually defined by presenting a state table or a state graph. In this way, the states, alphabet, transition function, and final states are constructively defined. The starting state is usually the lowest numbered state. Our first example of a finite automaton is:

$$M = (\{s_1 , s_2 , s_3\}, \{0,1\}, \delta, s_1, \{s_2\})$$

where the transition function δ, is defined explicitly by either a state table or a state graph.

At this point we must make a slight detour and examine a very important yet seemingly insignificant input string called the *empty* string. It is a string without any symbols in it and is denoted as ε. It is *not* a string of blanks. An example might make this clear. Look between the brackets in the picture below.



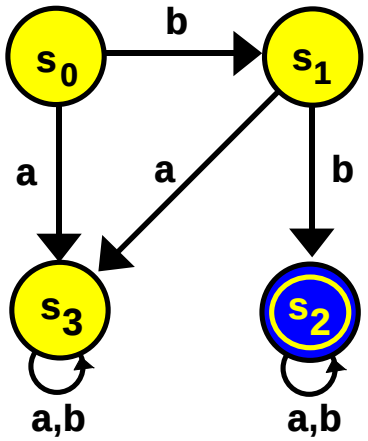
Let's look again at a computation by our first finite automaton. For the input 010, our machine begins in s_1, reads a 0 and goes to $\delta(s_1,0) = s_1$, then reads a 1 and goes to $\delta(s_1,1) = s_2$, and ends up in $\delta(s_2,0) = s_2$ after reading the final 0. All of that can be put together as:

$$\delta(\delta(\delta(s_1,0),1),0) = s_2$$

We call this transition on strings δ^* and define it as follows.

> **Definition.** *Let $M = (S,I,\delta,s_0,F)$. For any input string x, input symbol a, and state s_i, the* ***transition function on strings*** *δ^* takes the values:*
>
> $\delta^*(s_i,*e) = s_i$
>
> $\delta^*(s_i,a) = \delta(s_i,a)$
>
> $\delta^*(s_i,xa) = \delta(\delta^*(s_i,x),a).$

That certainly was terse. But, δ^* is really just what one expects it to be. It merely applies the transition function to the symbols in the string. Let's look at this for the example in figure 3.



This machine has a set of states = $\{s_0, s_1, s_2, s_3\}$ and operates over the input alphabet {a, b}. Its starting state is s_0 and its set of final or accepting states, $F = \{s_2\}$. The transition function is fully described twice in figure 3; once in figure 3a as a state table and once in figure 3b as a state graph.

State	Input		Accept?
	a	b	
0	3	1	no
1	3	2	no
2	2	2	yes
3	3	3	no

Figure 3 - Finite Automaton

If the machine receives the input bbaa it goes through the sequence of states:

$$s_0, s_1, s_2, s_2, s_2$$

while when it gets an input such as abab it goes through the state transition:

$$s_0, s_3, s_3, s_3, s_3$$

Now we shall become a bit more abstract. When a finite automaton receives an input string such as:

$$x = x_1 x_2 \ldots x_n$$

where the x_i are symbols from its input alphabet, it progresses through the sequence:

$$s_{k_1}, s_{k_2}, \ldots, s_{k_{n+1}}$$

where the states in the sequence are defined as:

$$\begin{aligned} s_{k_1} &= s_0 \\ s_{k_2} &= \delta(s_{k_1}, x_1) = \delta(s_0, x_1) \\ s_{k_3} &= \delta(s_{k_2}, x_2) = \delta^*(s_0, x_1 x_2) \\ &\vdots \\ s_{k_{n+1}} &= \delta(s_{k_n}, x_n) = \delta^*(s_0, x_1 x_2 \ldots x_n) \end{aligned}$$

Getting back to a more intuitive reality, the following table provides an assignment of values to the symbols used above for an input of bbaba to the finite automaton of figure 3.

i	**1**	**2**	**3**	**4**	**5**	**6**
x_i	**b**	**b**	**a**	**b**	**a**	
s_{k_i}	**s_0**	**s_1**	**s_2**	**s_2**	**s_2**	**s_2**

We have mentioned acceptance and rejection but have not talked too much about it. This can be made precise also.

> **Definition.** *The **set** (of strings) **accepted** by the finite automaton* $M = (S,I,\delta,s_0,F)$ *is:* $T(M) = \{ x \mid \delta^*(s_0,x) \in F \}$

This set of accepted strings (named T(M) to mean T*apes of M*) is merely all of the strings for which M ended up in a final or accepting state after processing the string. For our first example (figure 1) this was all strings of 0's and 1's that contain *exactly* one 1. Our last example (figure 3.1.3) accepted the set of strings over the alphabet {a, b} which began with *exactly* two b's.