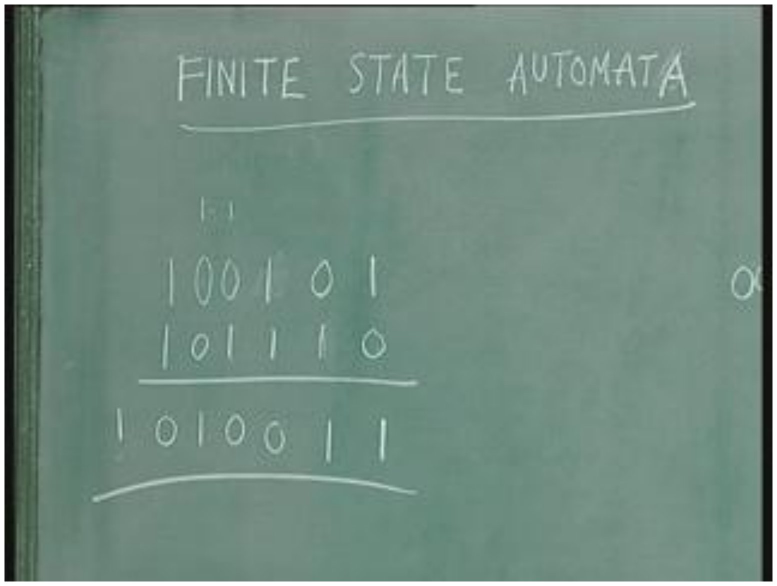
Discrete Mathematical Structures
Dr. Kamala Krithivasan
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Lecture - 38
Finite State Automaton

Today we shall consider the topic finite state automata. This is a very useful topic it has got lot of applications. It finds use in the lexical analysis part of a compiler. You know that a compiler is a very big program which translates a high level language into a machine language and it has got several parts. And in the lexical analysis part of a compiler the idea of finite state automaton has found lot of use. And it is also used in text editing and in computer networks and in image compression and so on. There are a lot of applications for that. Though we may not study all of them I will indicate you the application of finite state automaton in the lexical analysis part.

Actually, if you know about sequential circuits the finite state automaton is an abstract model of a synchronous sequential circuit. So let us start with a serial adder and go into the concept of a finite state automaton. Consider two binary numbers and consider the addition of these two numbers 1 0 gives you 1 0 1 also gives you 1 1 1 gives you 0 with a carry 1 and that carry along with this 1 gives you 0. And there is a carry and the carry with 0 0 gives you 1. Now, there is no carry, this 1 1 also gives you a 0 and there is a carry and that is given here.

(Refer Slide Time: 02:45)



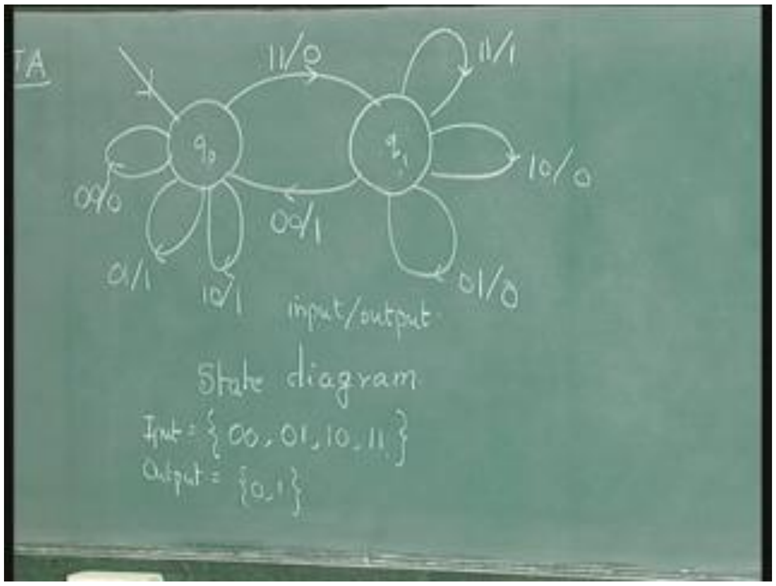
Now, if you want to represent it as a state diagram like this, this is called a state diagram, first you get 10, the input is 10 then the input is 01 then the input is 11 then 01 then 00 11, the output is 1 1 0 0 1 0 1. Let us see how you get this from this diagram.

First you are starting in the initial state q_0 which is marked like this then in this you get 10, the 10 gives you a output 1 and you go here that is what you get.

Then you get an input 01 then also the output is 1 and you go here then you get a input 11 so the output is 0 and you go here and then you get an input 01 so you get 0 as the output and you go here then you get the input 00 so you go here with a output 1. And from here you get the input 11 so the output is 0 and go here and from here you get 00 that is there is no output, the last output, for this there is no input but without loss you can assume it as 00 so from here you get 00 and you output 1.

Now, here the input is a pair of binary digits 00, input is either 00 or 01 or 10 or 11 two digits, and the output is 0 or 1. In the diagram the input is written first with a slash and then the output is written.

(Refer Slide Time: 05:10)

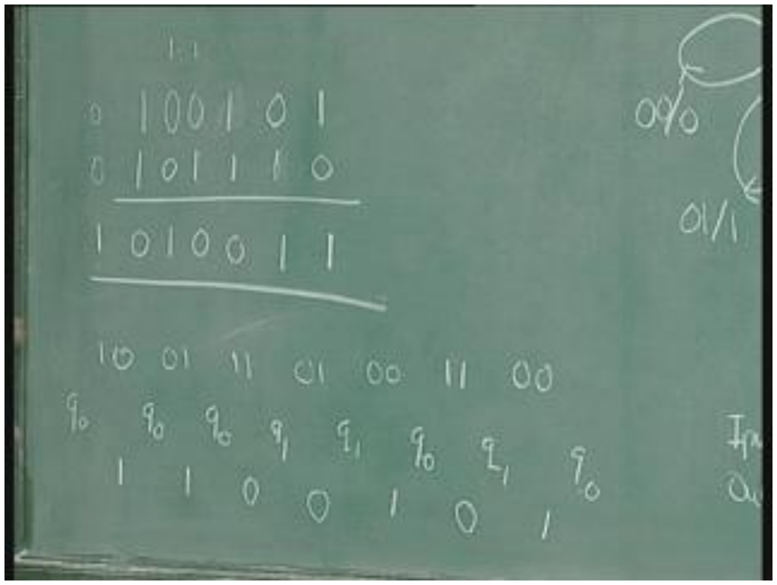


So let us write the inputs like this, this is the 1 which I am going to get first, then this then this and so on. So, first you get 10 then 01 then 11 then 01 00 11 and 00.

So initially you are starting in state q_0, after getting 10 you are again in state q_0 and at that time the output will be q_0 10 gives you an output 1, then next you are in state q_0 and you get 01. So you go to q_0 again and the output is 1 these two are called states. So from q_0 if you get 1 1 you go to q_1 and the output is 0 now.

From q_1 if you get 01 you go to q_1 again and the output is 0, from q_1 if you get a 00 you go to q_0 and the output is 1, from q_0 if you get 11 you go to q_1 and the output is 0, from q_1 if you get 00 you go to q_0 and the output is 1. So when the inputs are 10 01 11 01 00 11 00 that is this sequence, the sequence of states will be q_0 q_0 q_0 q_1 q_1 q_0 q_1 q_0 and the output will be 1 1 0 0 1 0 1 that is 1 1 0 0 1 0 1. This is what you get and this you get from this diagram.

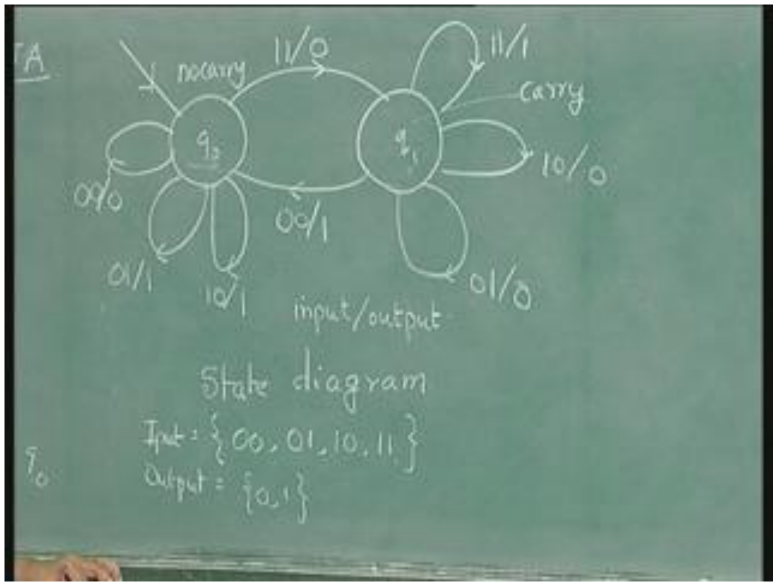
(Refer Slide Time: 07:24)



Now when you look at this diagram carefully you see that this state q_0 corresponds to a no carry state. When there is no carry the machine is in q_0 and when there is a carry this corresponds to a carry state. When you read two binary numbers digit by digit that is pairs of digits you read then after reading a particular portion either there will be a carry from the previous digits or there will be no carry. So when you see that initially when you start there is no carry. So when there is no carry if you add a 0 and 1 you get 1.

Again there is no carry without a carry when you add 0 and 1 you get 1 and now also there is no carry but when you add 1 and 1 you get the output 0 but there is a carry. So you have to distinguish between two types of states one is when there is a carry and the other is when there is no carry. The whole thing can be represented by a diagram like this, this is called a state diagram. And this is an abstract model of a synchronous sequential circuit. You know that a serial adder is a synchronous sequential circuit and abstract model is represented like this.

(Refer Slide Time: 09:03)



Now, let us consider one more example, consider this example q_0 and q_1 when you get 0 output is 0 when you get 1 the output is 1 when you get 0 here the output is 1 when you get 1 the output is 0 here the input is the symbols 0 and 1 that is also the output. Q_0 is an initial state the initial state is marked by an arrow like this so suppose I get the input 0 1 1 0 1 0 0 1 suppose I get this input what will be the state sequence? Initially the machine is in state q_0 when you get a 0 it goes to q0 again, what will be the output? Output will be 0 here.

In q_0 if you get 1 you go to state q_1, here again the input and output are written with a slash. The first symbol is input then slash then the output. This is the convention which we write. In q_0 when you get 1 you go to q_1 and then the output is 1. In q_1 when you get a 1 you go to q_0 and output is 0. In q_0 when you get 0 you are in q_0 and output is 0. In q_0 when you get 1 you go to q_1 and output is 1. In q_1 when you get 0 you remain in q_1 and output is 1. In q_1 again when you get 0 you remain in q_1 and output is 1. In q_1 when you get 1 you go to q_0 and output is 0.

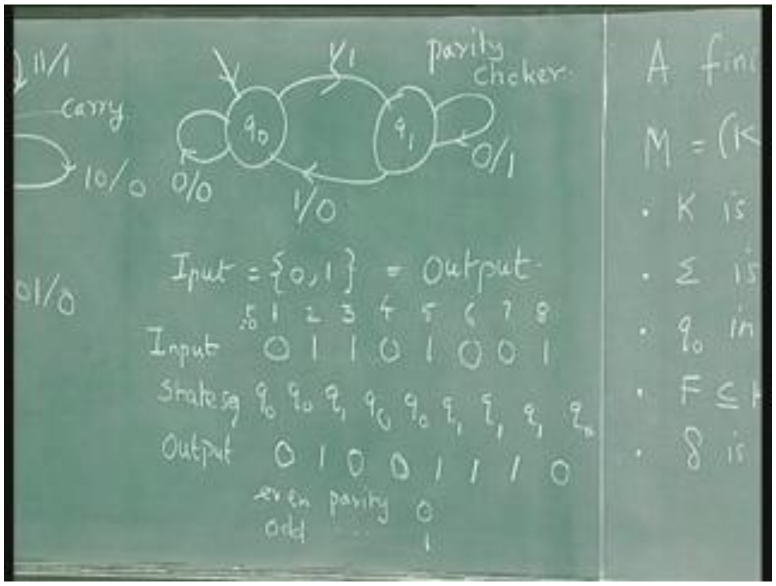
Now, if you look at this very carefully this is the input 0 1 1 0 1 0 0 1 and this is the output 0 1 0 0 1 1 1 0. Now, this is the time with which we start time t is equal to 0 then after reading time one you read this, time instance two you read this, time instance four and so on.

What does this do now?
At time five you have read this portion of the input and what is the output at time five? It is 1. At time six you have read this portion and what is the output? It is 1. At time eight you have read this portion and what is the output? That is 0.

At any time instance the output denotes the number of 1s you have read so far whether it is an odd number of 1s or even number of 1s. So this machine is actually a parity checker, this is a parity checker. So at time instance 0 you are in q_0, then at time instance 1 you read 0 and output 0. That is so far you have not read any one so even parity.

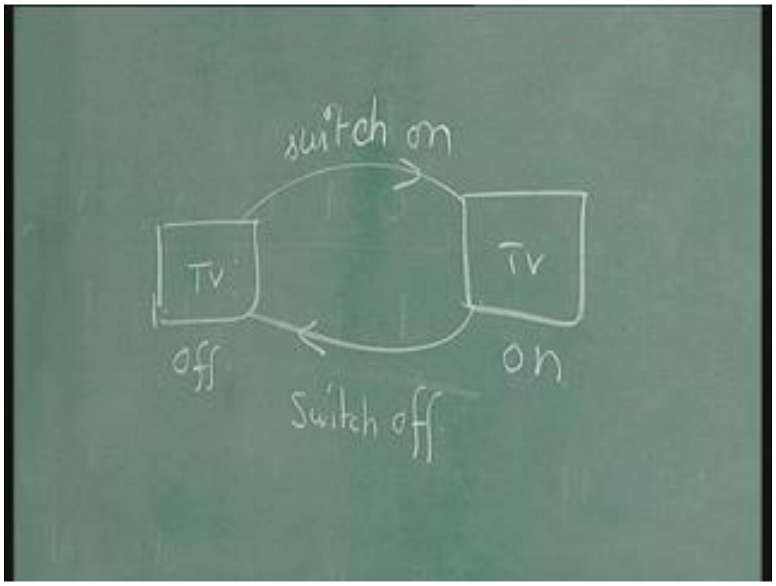
Next you read a 1 so you have read 11 after this point so the odd parity so you get 1. At next instance you have read two 1s so you have read even number of 1s so even parity you get 0. So, fourth instance also you get 0 so this tells you that you have read an even number of 0s so the output is 0. So whenever there is even parity the output is 0 so you get this even parity you get output 0 odd parity is denoted by the output 1 this is known as a parity checker. This is again a state diagram and there are two states q_0 and q_1 and input and output are written like this; input and then slash and then output, this is known as a finite state automaton. This is another example you know that parity checker can also be implemented by synchronous sequential circuit and this is an abstract model.

(Refer Slide Time: 15:04)



Now apart from this let us consider a simple one, suppose you are having a TV and when you switch on the TV is initially off then when you switch on it goes to the on state and when you switch off it goes to the off state, this is a small instance of a day-to-day life which you can represent it by means of a state diagram like this.

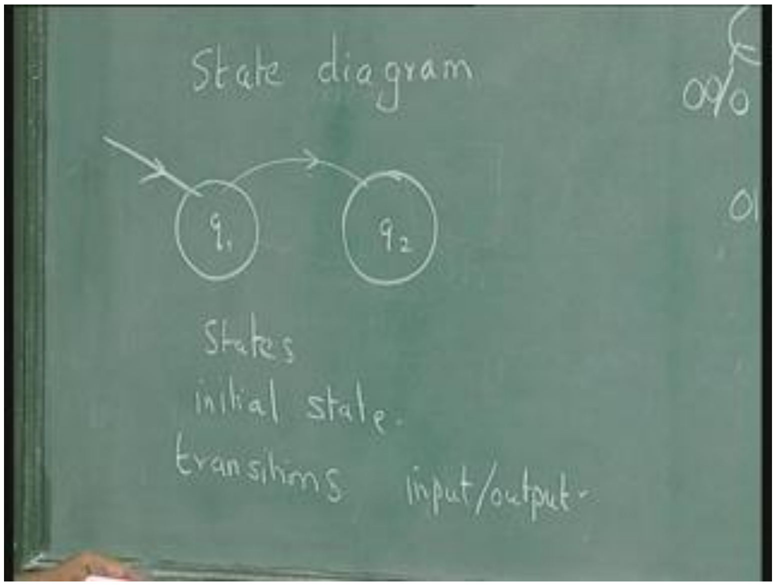
(Refer Slide Time: 16:00)



In general a state diagram consists of the following things; it has states written in the form of circles, states will be written like this, they are called states. And the initial state will be denoted like this; you have initial state marked then transitions are marked by arrows and on the transitions you have input slash output.

Later on we shall consider that you can look at it also as a recognition device in which case you will not talk about output but only input but some of the states will be designated as final states.

(Refer Slide Time: 17:13)



So let us consider one more example like this; this also has got two states and the input can be 01 which is also the output, in that case when you get 0 the output is 0 when you get 1 you go to q_1, in q_1 when you get 1 you remain there, again in q_1 when you get 0 you go here but the output will be 1. So this is another state diagram of a machine. Let us see what this machine does.

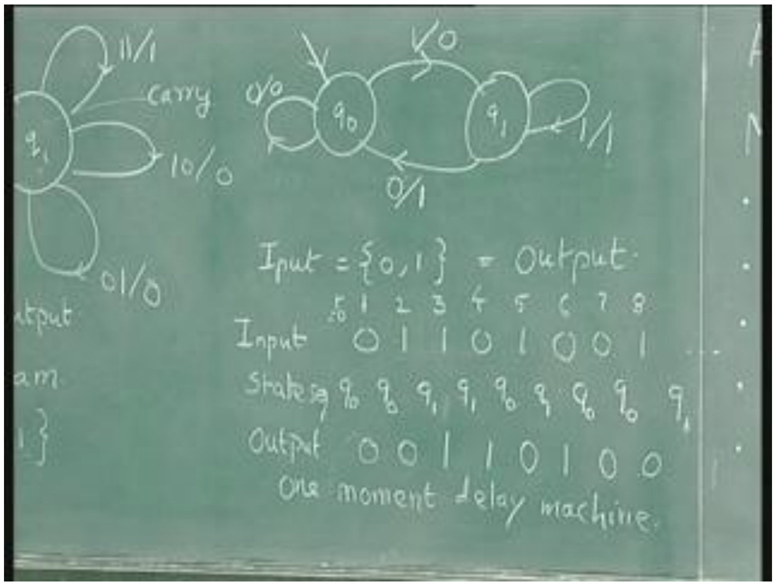
Let us consider the state sequences, initially you are in q_0 and when you get 0 you are in q_0, in q_0 when you get 1 you go to q_1, in q_1 when you get 1 you go to q_1, in q_1 when you get 0 you go to q_0 and in q_0 when you get 1 you go to q_1 and in q_1 when you get 0 you go to q_0 and in q_0 when you get 0 you go to q_0 and in q_0 when you get 1 you go to q_1.

Now, let us see what is the output, the output is q_0 0 is 0, q_0 1 is 0, q_1 1 output will be 1, q_1 0 you go to q_0 but output 1 and q_0 when you get 1 you output 0 and go to q_1, in q_1 when you get 0 you output 1 and go to q_0, in q_0 when you get 0 you go to q_0 and output 0, in q_0 when you get 1 you go to q_1 and output 0.

Now, you look at it carefully, what does the output represent? The output represents like this, leave a_l1 the first output the first output will be 10 here always, afterwards look at the input sequence and the output sequence 0 1 1 0 1 0 0 so the same input you are getting here except for the first one and the first one will always be 0. And you are in state q_1 now next whether you get 0 or 1 you are getting q_1 next instance you may get 1 or 0 but the output will be 1, next instance what ever may be the input the output will be only 1.

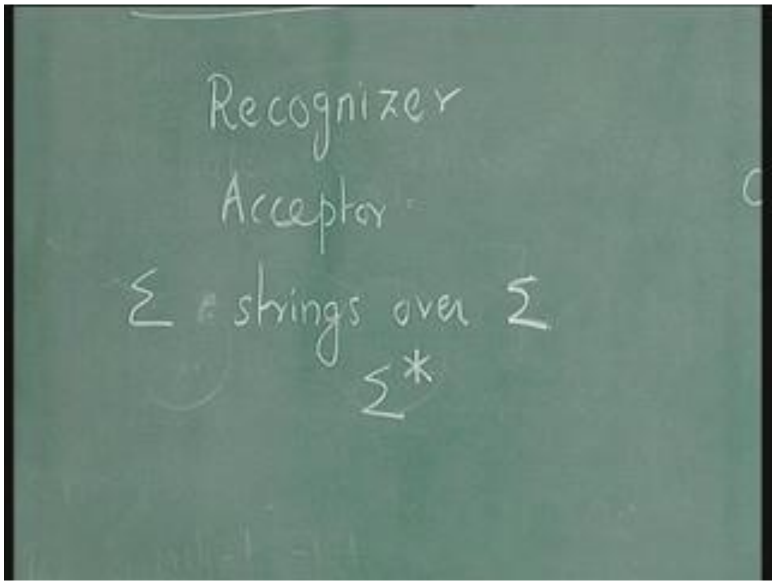
So if you forget about the first output the same input sequence you are getting here and this machine is called a one moment delay machine. The first output is 0 afterwards this input is output one instance later, second instance you get 1 but you output what you got at the first instance, third instance the input is 1 but you are outputting what you got at the second instance and so on. That is why this machine is called a one moment delay machine, this is again another example of a finite state machine. Like that we can consider several examples. In all these examples we have input we have output but we did not say anything about final states and so on.

(Refer Slide Time: 22:13)



You can also look at the finite state automaton as a Recognizer or an Acceptor. It is accepting strings, you consider an input alphabet sigma and consider strings over sigma, this is denoted as sigma star this we know, strings over sigma are denoted by sigma star. A finite state machine can be designed such that it accepts a subset of sigma star.

(Refer Slide Time: 23:00)


Recognizer
Acceptor
Σ strings over Σ
Σ*

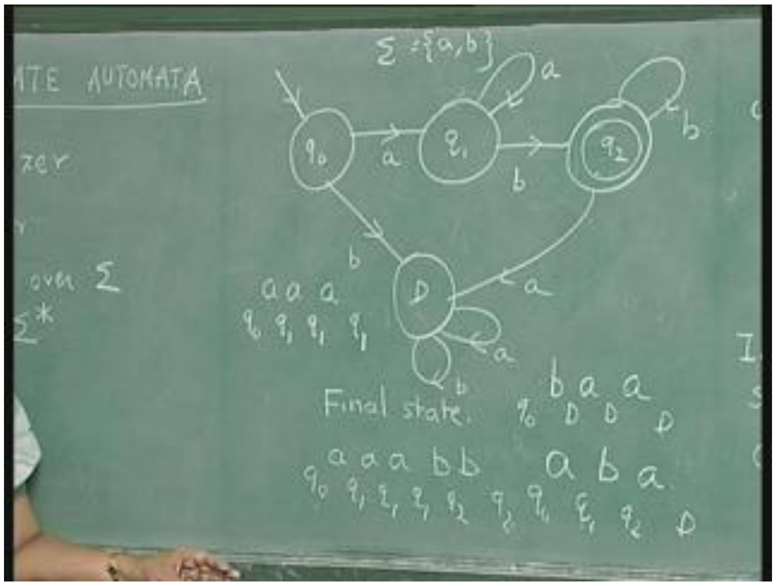
Let me consider one example; look at this machine the alphabet is a, b. Let us consider this diagram, there are four states q_0 q_1 q_2 and D, now you see that you are not marking any of them like input slash output the transitions but it has only one symbol which denotes the input. Now, we are bothered about only the input. And you see that this state is marked with a double circle that is called a final state and this is the initial state.

Now, if you traverse a path along for a string for example consider a string a a a b b this is the input, what will be the state sequence for this? q_0 a will give you q_1, q_1 a will give you q_1, q_1 a will give you q_1, q_1 b will give you q_2, q_2 b will give you q_2. Hence, for the sequence a a a b b and after reading the string a a a b b you reach the state q_2 which is a final state. So, given a string you start with q_0 and after reading the whole string if you reach a state which is a final state that string is said to be accepted by this final state automaton.

Consider this string b a a or consider the string a b a what will be the state sequence for this? This is q_0, q_0 b will be D why I am calling it as D and not q_3 will be clear in a moment, D a is D, D a is D. Similarly, start from here q_0 a is q_1, q_1 b is q_2, q_2 a is D. After reading these strings you are in the state D which is not a final state. So these two strings b a a and a b a are not accepted by the machine look at this string also a a a start from q_0 after reading a you will be in q_1 after reading a again you will be in q_1 after reading a you will be in q_1 q_1 is not a final state so this string also cannot be accepted by the final state automaton. So starting from the initial state after reading the string if you reach a final state the string will be accepted. So you look at this, this string will be accepted whereas this string or this string or this string will not be accepted by this machine.

So, if you look at the diagram very carefully you will realize that any string where you have a sequence of a's followed by a sequence of b's will be accepted but there should be at least 1 a and 1 b. Hence, a sequence of a's followed by a sequence of b's will be accepted by this machine with a condition that there should be at least 1 a and 1 b any other string will not be accepted. This is the idea of a finite state automaton as a recognizer.

(Refer Slide Time: 28:20)



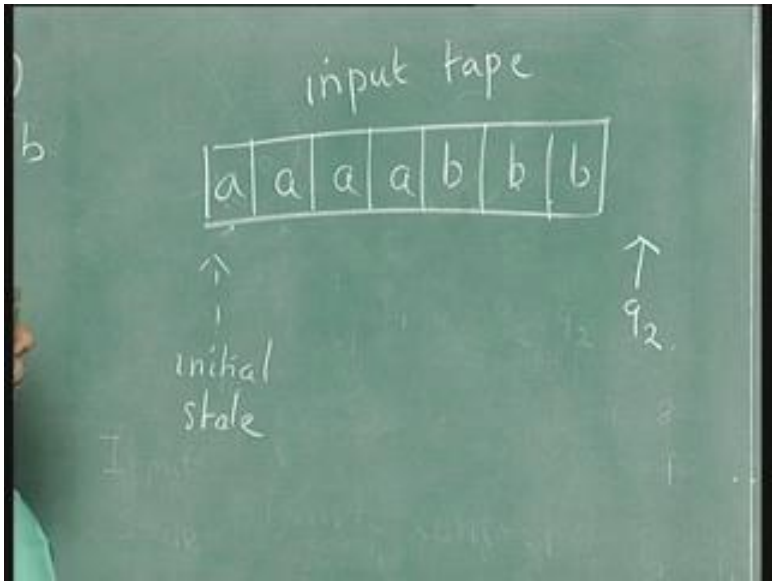
Let us see the formal definition. Now, before going into that the idea is represented like this; you have an input tape in which you have the symbols say for example here you have the symbols. There is a finite control this is called finite control which represents the state of the machine. And there will be an input head, initially the input head will be pointing to the left most symbol of the input tape, this is the input tape. So now the input is a a a a b b b and initially the machine will be in state q0 which is the initial state. Then at any particular instance depending upon the state and the symbol you go to the next state and the input pointer will be moved one point to the right.

Let us consider this example, so initially the machine is in q_0 reading the symbol reading a, what is q_0 a? q_0 a is q_1 so the next instance it will go here, the input pointer will be moved and you will be in state q_1. In q_1 it is reading a a so depending upon the state and the symbol it will move the pointer to the next cell and the state will be changed.

What is q_1 a?
From this diagram you can see that q_1 a is q_1 so here you will get this. So state q_1 is reading again q_1, a is again q_1 so it will move its pointer here and it will be reading this in q_1. Again q_1 a is q_1 so the input pointer will be moved and you will go to the next cell.

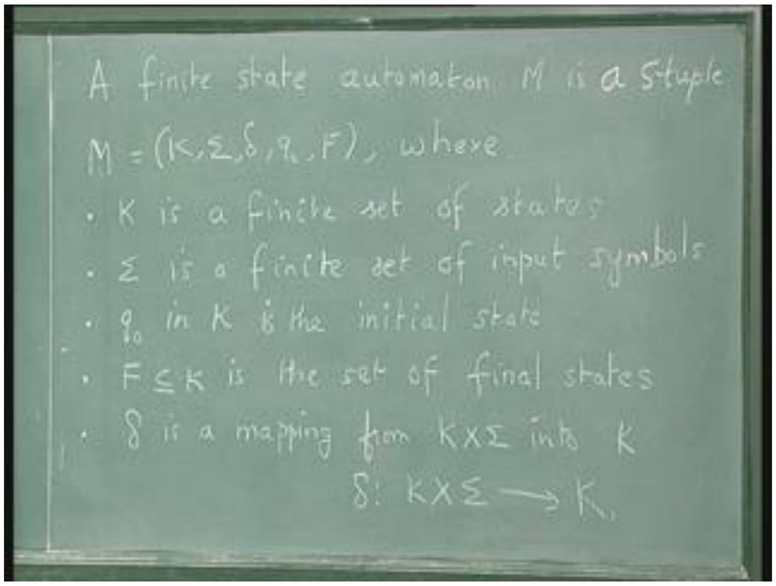
What is q_1 b in the diagram?
q_1 b will be q_2 so from q_1 b the machine will go to q_2 and the input pointer will be moved next. In q_2 if you read a b you will get only q_2 the next state will be q_2. So now the input pointer will be moved and you will go to this cell in state q_2. In q_2 again you are reading a b so the next state is q_2. So after reading the whole input you are in state q_2 which is a final state. So initially you start reading the left most symbol in the initial state and after reading the whole input if you reach a final state the string will be accepted by the automaton. If you reach a non final state then the string will not be accepted by the automaton. So let us go to the formal definition.

(Refer Slide Time: 31:42)



A finite state automaton m is a five tuple, m is denoted formally like this m is equal to k sigma delta q_0 F where K is a finite set of states, sigma is a finite set of input symbols q_0 in k is the initial state and one of the states is designated as the initial state. A subset of the set of states is designated as final states. F contained in K is the set of final states. Delta is a mapping from K into sigma into K or you denote delta as K into sigma into K.

(Refer Slide Time: 32:42)



A finite state automaton M is a 5tuple

$M = (K, \Sigma, \delta, q_0, F)$, where

- K is a finite set of states
- Σ is a finite set of input symbols
- q_0 in K is the initial state
- $F \subseteq K$ is the set of final states
- δ is a mapping from $K \times \Sigma$ into K

$\delta : K \times \Sigma \rightarrow K$.

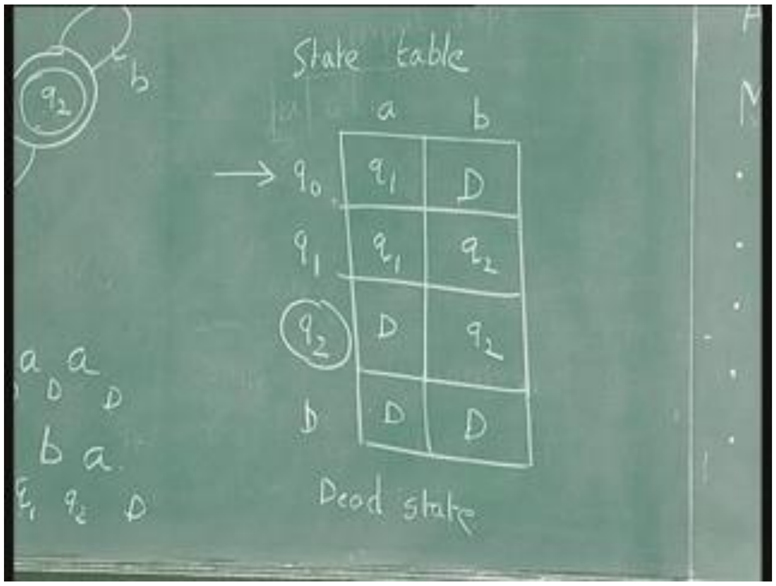
And the whole thing is denoted by a state diagram like this. The states are marked with circles, the initial state is marked with this, the transitions delta mappings are denoted like this. The whole thing can be represented by what is known as a state table also.

There will be one column corresponding to each input symbol and there will one row corresponding to each state q_0 q_1 q_2 and D in this particular example. Let us consider this example, the initial state is marked like this q_0 is the initial state so it is marked like this and q_2 is the final state which is marked like this.
The transitions are marked in the state table like this.

What is q_0 a?
q_0 a is q_1, q_0 b is b. so q_0 a is q_1 q_0 b is D.

What is q_1 a?
q_1 a is q_1 and q_1 b is q_2, q_2 a is D q_2 b is q_2. So this is d this is q_2, D a is D and D b is also D. So the same diagram you can represent as a table like this, you see that each cell contains a single state and so on.

(Refer Slide Time: 35:11)



State table

	a	b
$\rightarrow q_0$	q_1	D
q_1	q_1	q_2
q_2	D	q_2
D	D	D

Dead state

Why is this state D?
I am denoting as D. Once you reach this state afterwards you cannot go to any final state so the string has to be rejected only, so such a state is known as a dead state. D corresponds to a dead state here that is why we are calling it as a D. So we have seen that a finite state automaton is a five tuple and it has got five components K sigma q0 F, we have seen what is that, delta is mapping from K into sigma into K. And this transition, this is the called transition mapping and that can be represented by a state diagram like this or by a state table like this.

Now, let us define the language accepted in a formal manner. Now extend delta to delta bar where this is from K into sigma star to K, delta is from K into sigma now you are extending into K into sigma star into K. You define like this; delta cap q epsilon the empty word equal to q for all q in K. And if x is a string and a is a symbol then you define delta cap as of course q belongs to K.

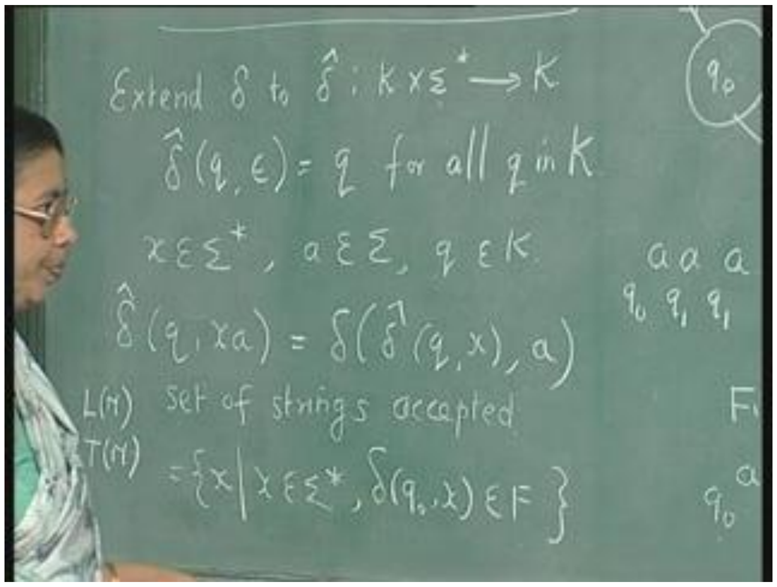
What can you say about delta cap q x a?
That is equal to delta of delta cap q of x, a. In a sense it means like this you have a string x the last symbol of which a you start in the beginning with a state q, what is the state to which you go after reading this x a that is denoted by delta cap q(x a). That is after reading q x you will be in a particular state and from that state say p then from that state you read a symbol a and go to a state. Therefore, after reading x you are in a state and from that state you read a and go to this state say r this is what is meant by this.

Now, if you look at this carefully for a single symbol delta cap q a is the same a delta q a. So, we need not have to write delta cap and delta. Actually delta cap is when you have a string and delta is when you have a single symbol because they are the same when you take a single symbol you need not have to distinguish and write two different symbols you can just use the symbol delta itself.

So what is the language accepted?
The language accepted is denoted by this, sometimes it is denoted as L(M), sometimes it is denoted as T(M) where M denotes the finite state automaton, that is the set accepted, set of strings accepted and that is denoted by set of strings of the form x belongs to sigma star delta of q_0 x belongs to F.

So the set of strings which takes you from the initial state to a final state or when you start reading a string starting with the initial state and if you reach a final state such a string will be accepted by the machine. That is what we have seen in this example also. So the language accepted is denoted like this.

(Refer Slide Time: 39:11)



Extend δ to $\hat{\delta} : K \times \Sigma^* \to K$

$\hat{\delta}(q, \epsilon) = q$ for all q in K

$x \in \Sigma^*$, $a \in \Sigma$, $q \in K$

$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$

L(M) set of strings accepted

T(M) $= \{x \mid x \in \Sigma^*, \delta(q_0, x) \in F\}$

q_0
a a a
q_0 q_1 q_1
F
q_0 a

Let us consider a few more examples. Let us take the alphabet sigma is equal to a, b input alphabet is this, look at this diagram there is only one state and you have this diagram.

What is the set of strings accepted?
Any string if u take consisting of a’s and b after reading that starting from q_0 you will be in q_0 only there is only one state so the set accepted or the language accepted is just sigma star in this case any string of a’s and b’s will be accepted by this machine.

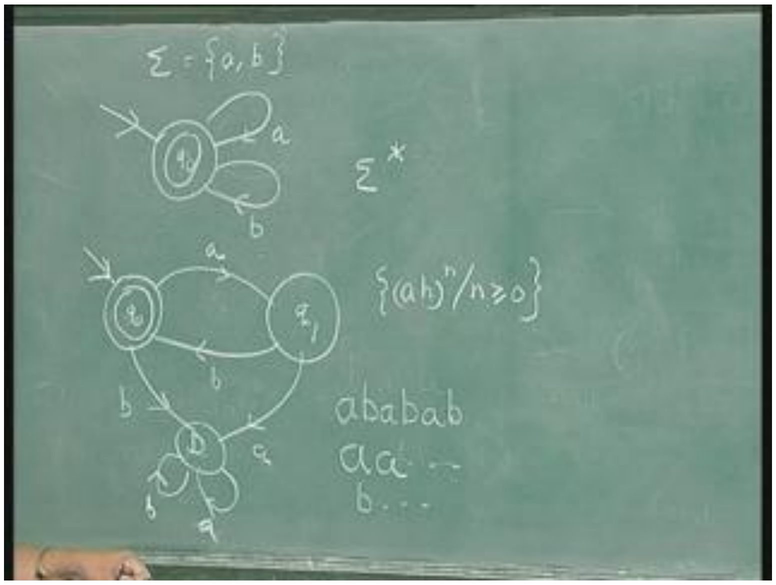
Let us consider this example; q_0 is the initial state as well as the final state there is a state q_1, if you read a you go here if you read a b you go here, then there is one more state this one.

What is the set of strings accepted?
If you get a string of the form ababab start from here a b a b a b you will reach a final state q_0. So any string of the form a b a b a b a b like that will be accepted by the machine. But if you have a string of the form a a something like that or a string beginning with b what will happen a a you will go to this afterwards you will remain in state D only.

Similarly, if a string begins with b you will go here and afterwards you will be in the dead state only.

Once you reach a dead state you cannot go back to other states so string cannot be accepted. So the language accepted is the set of strings of the form a b power n n greater than or equal to 0. What do I mean by n greater than or equal to 0 is epsilon the empty string will also be accepted by this machine. This machine accepts the empty string also.

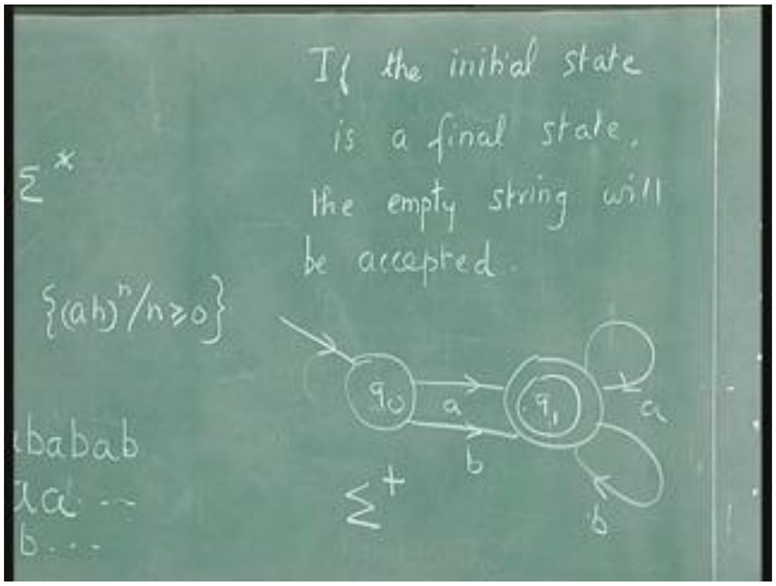
(Refer Slide Time: 42:28)



When does epsilon gets accepted?
If the initial state is a final state the empty string will be accepted.

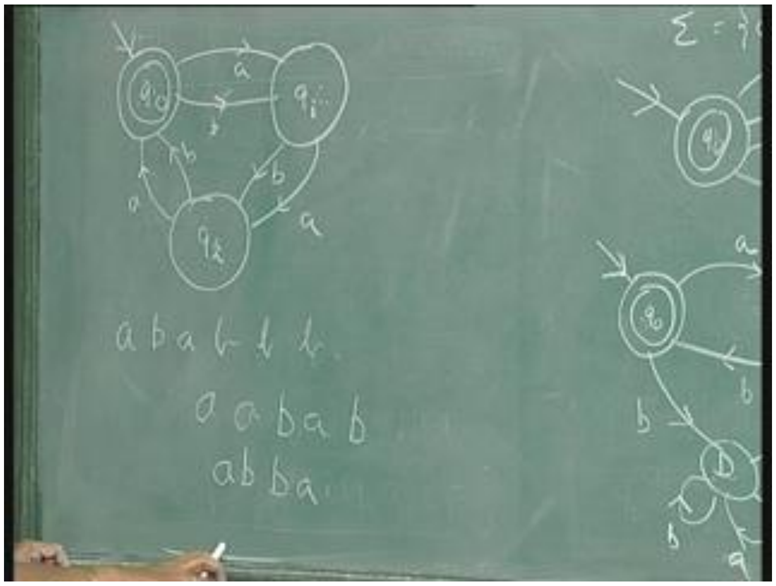
Look at this diagram, what sort of strings will be accepted by this machine. Here we are having two states q_0 and q_1 and q_0 is the initial state and q_1 is the final state. Here also any string of a's and b's will be accepted. First you have a then you can have any string, first you have b then you can have any string so any strings of a's and b's will be accepted but q_0 is not the final state so at least one symbol should be there in the string to get it accepted so epsilon will not be accepted by this machine. This machine does not accept the empty string but any other string of a's and b's will be accepted. So the language accepted is sigma plus any string of a's and b's will be accepted but empty string will not be accepted by this machine so the language accepted by this is sigma plus.

(Refer Slide Time 44:23)


If the initial state is a final state, the empty string will be accepted.
Σ^*
$\{(ab)^n / n \geq 0\}$
babab
aa --
b . . .
q_0
q_1
a
b
Σ^+

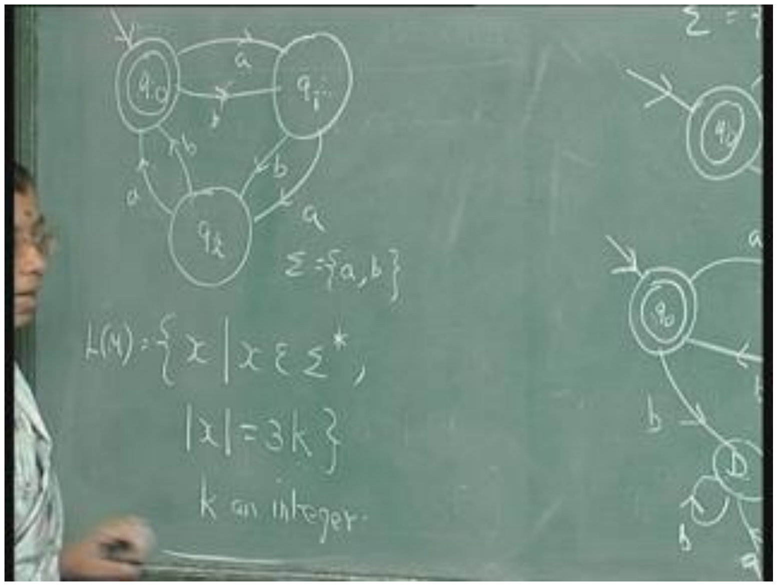
Let us consider a few more examples so this idea is clear. Look at this machine q_0 q_1 q_2 a b a b like the initial state of the final state so epsilon will also be accepted. What sort of strings will be accepted by this machine? Here again you see that if you take any string a b a b b b something like that it will be accepted a b a b b b will be accepted. But the condition is the length of the string should be divisible by 3. So if you just take a alone starting from here you will go here so it will not be accepted. Or if you take five letters a a b a b you will reach this state and it will not be accepted. If you take a four letter one a b b a starting from here a b b a you go to q_1 it will not be accepted.

(Refer Slide Time 46:00)



Any string of a's and b's will be accepted provided the length is divisible by 3. So the language accepted are L(M) T(M) is the set of strings x x belongs to sigma star of course sigma is a b here a, b and the length of the string is of the form 3K, K is an integer, this is another example.

(Refer Slide Time: 46:42)

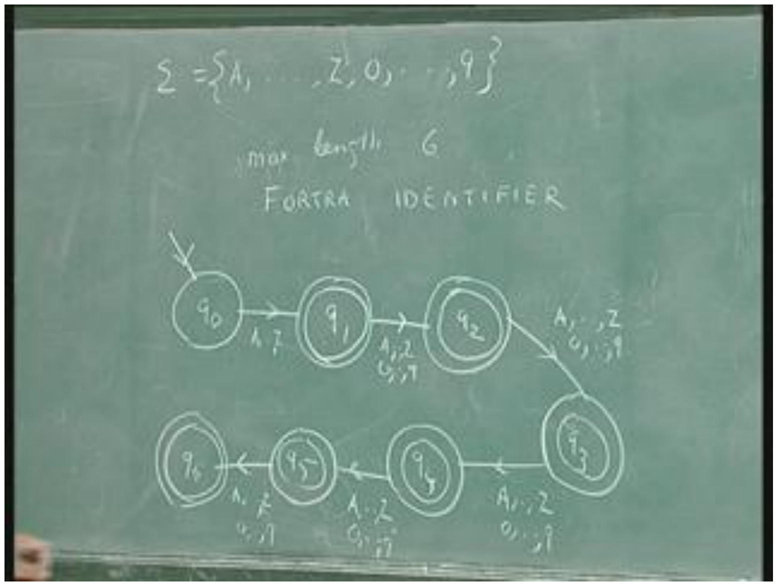


Like that we can consider several examples. Now, let us see how an integer can be represented by this and a Fortran identifier or a Pascal identifier will be accepted by this. Let us take the alphabet sigma to be, I shall use only capital letters A to Z and 0 to 9. I am just taking only capital letters small letters also can be taken in some other example. But here the alphabet consists of the symbols A to Z and the digits 0 to 9. Now, a Fortran identifier has a maximum length of 6 Fortran identifier and it begins with the letter A to Z and afterwards you can have a letter or a digit.

So that can be represented by a state diagram like this, we start with q_0 the initial state then q_1 q_2 q_3 q_4 q_5 q_6. So it begins with the letter A to Z starting from q_0 if you get A or Z A B C D any Z you go to q_1. And it can be just that you may end there also so this also happens to be a final state but you can have two letter identifiers so the second symbol can be A to Z or 0 to 9 then you go to q_2 again you can end up here or continue further so this also happens to be a final state.

Then we have three letters, an identifier consisting of three symbols the first one of course is the letter, second one is a digit, third one can be a digit or a letter so you go here, the fourth one again could be A to Z or 0 to 9 then this can again be a final state you can have a fifth letter which is A to Z or 0 to 9 again this can be a final state this also can be a final state sixth letter can be letter from A to Z or 0 to 9.

(Refer Slide Time: 50:09)

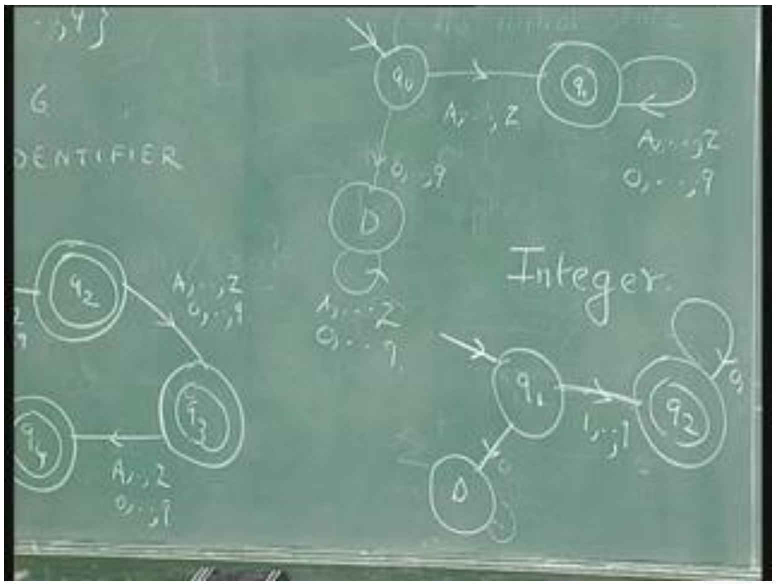


In Pascal you do not have the limit on the length of the thing so suppose I use only capital letters then you can have like this, any number of symbols you can have the first letter has to be A to Z then you may get q_1 A to Z or 0 to 9.

Of course I should also have something like this; the identifier cannot begin with a digit so if you get 0 to 9 I should go to a dead state and like this A to Z 0 to 9. So a Pascal identifier can be given by this, a Fortran identifier can be given by this, an integer should not begin with a 0, how do you represent an integer using a finite state diagram?

Initially you have q_1 the first symbol you do not want to be any integer decimal integer so 1 to 9 if you get you go to q_2 then it can be followed by any digits 0 to 9. So this diagram represents an integer. Of course when you get a 0 you do not expect it to begin with a 0 you go to a dead state and so on.

(Refer Slide Time: 51:42)



So, you see that you can represent identifiers and integers using a state diagram. You can also represent something like 10172.673 like this a decimal number like this using a state diagram.

Initially you will have an integer q_0 q_1 then it can be followed by any digit then a decimal point so here I am just writing those symbols which will lead you to find states and here the alphabet will also consist of a dot. So when you do not get a proper thing you have to go to a dead state that portion of it I am not marking on the diagram q_3. And in q_3 you may get any digit 0 to 9. So this represents a decimal number. In lexical analysis parts of the compiler you have one finite state automaton to represent the identifier one to represent integer and one to represent keywords and so on.

So this entire finite state automaton will work together on the input. the program is taken as a sequence of symbols it will work on this and whenever you recognize an identifier or an input or a special symbol, logical operators like that you will take an you will take it as token and that will be entered in a proper table a symbol table or a constant table and you will put it there and corresponding action will take place then the parsing will start taking place. So that recognizing each one as a token is called the lexical analysis and in this part the finite state automaton plays a very useful part. not only that in text editing suppose I want to replace a particular portion of a text by some other thing that particular text alone can be represented by a finite state automaton and in that you will remove that and replace it and so on.

The same idea is useful in text editing also. This is one of the very important uses of finite state automaton. And one of the things you must notice here is that when you are in a particular state and when you get a next symbol the next state is uniquely determined.

So whatever we have considered today is called a deterministic finite state automaton. In contrast to this you may also have non deterministic finite state automaton about which we shall study in the next lecture. In the deterministic finite state automaton the mapping is from K into sigma into K whereas in the non deterministic finite state automaton it will be from k into sigma into finite subsets of k.

We shall consider some examples and see whether the power is really increased or not. The power will not be increased, non deterministic automaton also has the same power. One more thing is the language accepted by a finite state automaton is called a regular set.

In the next class we shall see more about finite state automaton, in fact non deterministic finite state automaton and also some minimization procedures.