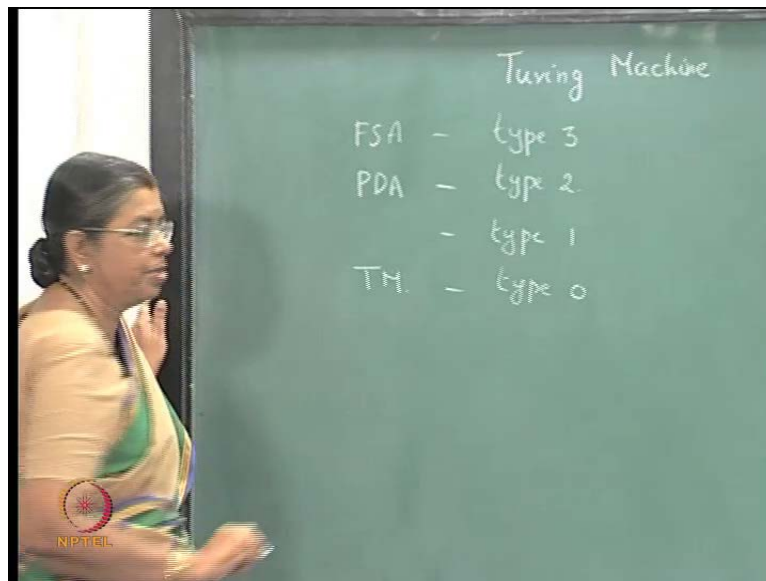


Theory of Computation
Prof. Kamala Krithivasan
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture No. #26

Turing Machines

(Refer Slide Time: 00:19)

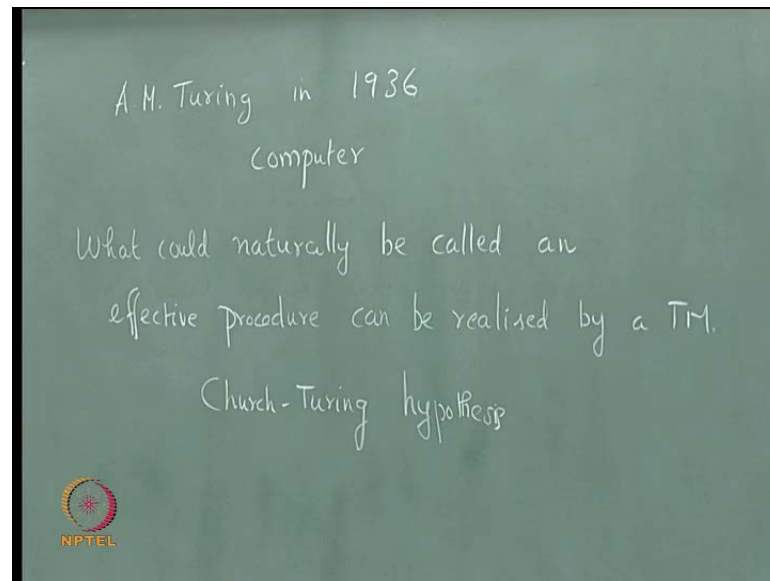


Today we shall consider the most general type of machine Turing machine.

(No audio from 00:16 to 00:30)

We have already consider finite state automata, which correspondent to type 3 grammars. And also, we have consider the pushdown automata which was equivalent to type 2 and we have studied four types of grammar in the Chomsky and hierarchy type 0 and type 1. The turing machine corresponds to type 0 grammar, the equivalence we shall study later, is a machine model.

(Refer Slide Time: 01:09)



The machine model was defined by A M Turing in 1936 and he called the basic model as a Computer he just called it as a Computer. And what he stated is this, what could naturally called an effective procedure (No audio from 01:47 to 01:57) can be realized by a turing machine.(No audio from 02:09 to 02:15) This is known as Church Turing hypothesis.

(No audio from 02:18 to 02:38)

(Refer Slide Time: 02:47)



Now, what he did is he constructed an abstract a model, And that abstract model had an infinite tape. Turing machine has an infinite tape, which is divided into cells. It is infinite you can take it infinite in both direction or in one direction does not matter. Later on, we will see the equivalence between these two things. Except for some portion, some a_1 a_2 a_n , the rest of them will be blanks. I will denote the blank portion with a b and the slash. Except for a finite portion, the rest of them will be blanks. The tape is infinite, but the non blank portion is only finite. Now, you can look at a Turing machine as an input output device or an accepting device. When you look at it as an accepting device, it accepts type 0 languages.

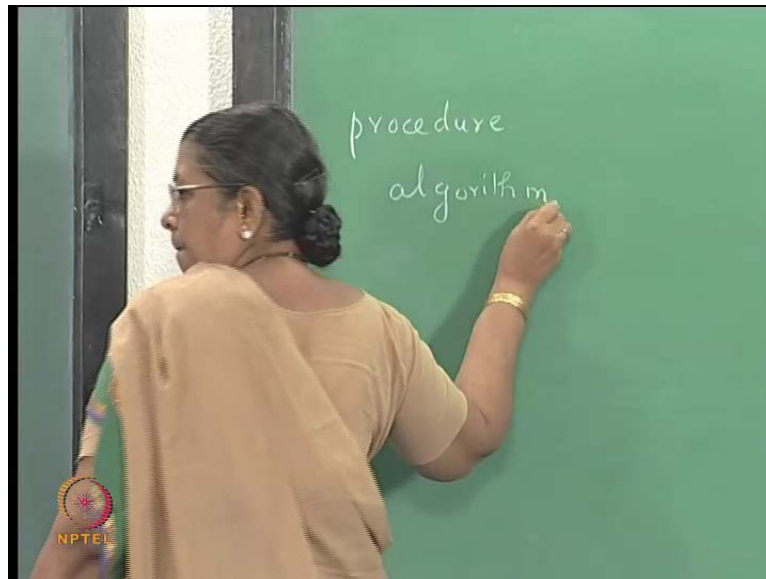
First, we shall look at it as a input output device. And what is the input, the input is the non blank portion when you start the machine. How does a machine work? Depending on a state and a symbol, the current state is q and the symbol read is a . The next instance, the state can change to p , and this symbol will be read and you can rewrite over that symbol something else will be written on that cell. And then the tape head can move right or left, it can move right or it can move left. There are some symbols which can be written on the tape they are called tape symbols, Γ if you denote as tape symbols. The mappings will be written like this $\delta(q, a) = (p, A, R)$ something like this, formal definition, I will write in a moment.

This means that in state q if you are reading A each cell contains one symbol. So, if you are writing, if you are reading a from q , you will go to state p , instead of small a you print; A capital A and move right. And, what he proved that you can compute anything at the end, it will go to halting state and halt. And when the machine halts, whatever is a non blank portion, that is the output. Whenever you start the machine, the non blank portion is the input and after some move, the machine will go to a state called halting state. It will start, some of the states, it will just halt. So, when it goes to halting state, whatever is the non blank portion that is the output. So, you can look at it as a input output device.

Now, what Turing said in 1936, was, that whatever could naturally be called an effective procedure can be realized by a Turing machine. What is an effective procedure? An effective procedure is like a program. It tells you what is the next step to be performed. If it is not some line of the code is there, next line it will be executive next line. And if there is a transfer of control, you can use a go to or here it is leave, then else you may use some portions where the code main a control will be transferred. In a sense, it tells you what is the next, what you have to do in the next step. That is an effective procedure.

Usually, the words procedure and algorithm, they are used with the same meaning or the something like that, but there is a slight difference between them, a procedure and an algorithm.

(Refer Slide Time: 07:35)



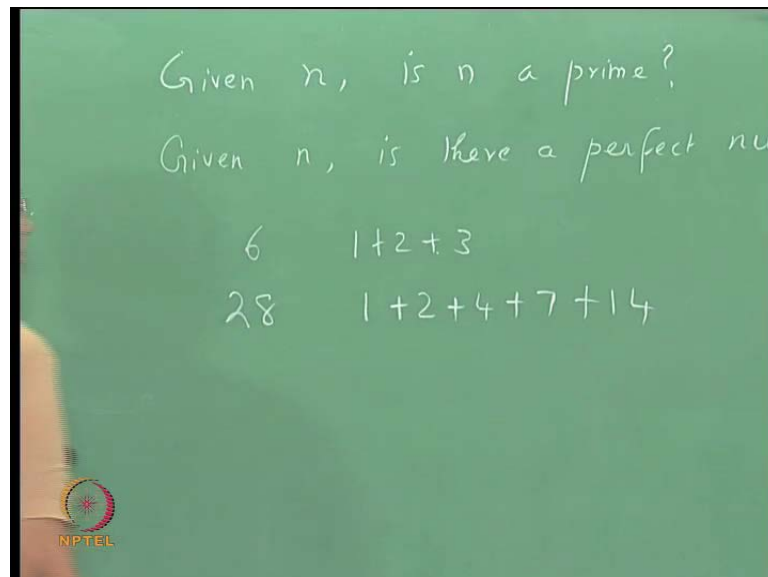
What is a procedure and what is an algorithm? (No audio from 07:29 to 07:43) Essentially, they are programs, a procedure is like a program, it tells you what is a next step to be done. Step by step doing something in step by step, but what is the difference between algorithm is also like that is not it step by step. You know what to do, solve some problem. But, what is the difference between them? The difference between them is that a procedure can halt or need not halt. An algorithm is suppose to always halt and give you the answer. In algorithm at the end, you are suppose to get the answer for your problem.

In a procedure, if the answer is yes, you may get, If the answer is no, you may not get the answer. I will tell you with an example which will illustrate you better manner. The input is a natural number. The problem is, is it the prime? You are going to test given input is a natural number. And you have to test whether it is the prime or not. If we will let us not to worry about effective, I mean time and all that they are efficiency of the algorithm. Just look at something which could be solved. Then how will you find out whether number is prime or not? Try to divide it by 2, the remainder is 0, it is not a prime.

If the remainder is not, if the remainder is 0, it is not a prime. If the remainder is 0, try with 3 and so on. Is not it? That is the way you proceed up to $p - 1$. If the input is p , up to

p minus 1 you will try. In fact, you can try up to root p , that is different thing. But at the end you will be able to say whether the given number is a prime or not; definitely there is an answer is it the prime or not? It is a prime means you will say yes, it is not a prime no.

(Refer Slide Time: 10:06)



So, that is an algorithm, you will definitely have an answer. Whereas, look at another problem. Give first problem which we considered is given n is n a prime; this one. Given n , is there a perfect number, given a number n , is there a perfect number greater than n ? What is the perfect number? The some of the divisors.

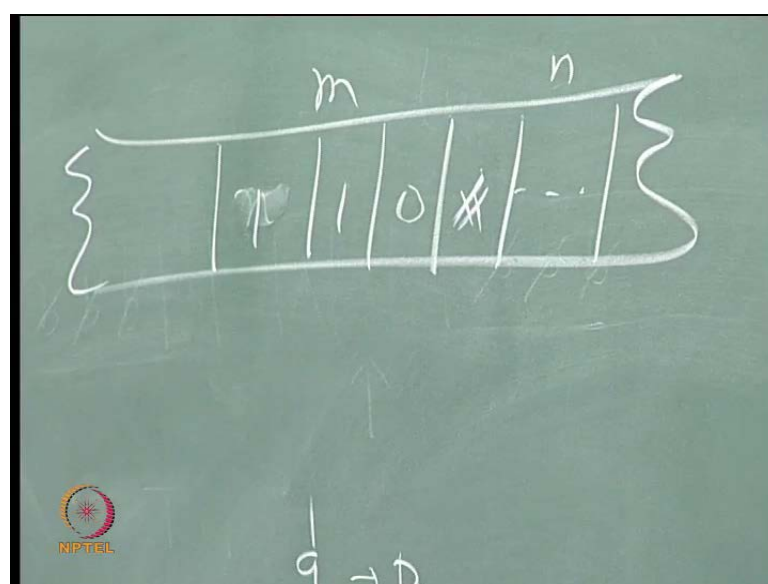
Excluding **(())**

Yes, some of the divisors is equal to for example, if you take a 6 1 plus 2 plus 3 will give you 6; the divisors are like that. If you take 28, the divisors are 1 you have to take 1 2 4 7; 28 is a perfect number. But given a number n is there a perfect number greater than? We can always just give, see given a number n you can test whether it is a perfect number or not. That see you can find all the divisors then add them up is it equal to the number. So, given a number is it a perfect number or not? You can test. Given a number n is there a perfect number greater than n that is the question.

Now, what will you do you can write a step by step procedure like this take n plus 1 find out whether it is a perfect number or not. If it is a perfect number yes otherwise you will take n plus 2. Try for the n plus 2. If it is not a perfect number, take n plus 3. If it is not, take n plus 4. If it is not, n plus 5. So, is the answer is yes ultimately, it will come out with the yes answer, but if the answer is no, you will keep on trying the next number. You will not know. But this also you can write step by step a program for this; I is equal to I plus 1, just whether I is the prime, I m sorry, I is the perfect number. If not, again go back I is equal to I plus 1. Test whether it is a perfect number. You can write the step by step procedure.

But if the answer is yes, it halts the answer is no. It goes on trying that is also an effective procedure. But it may halt, that is the slight difference between the procedure and algorithm, which may not be mentioned very clearly in many books. And many books may use the procedure and algorithm the same manner, but this is the difference as a Turing machine. And when you study halting problem and all that this difference will come out. So, the number you can look at it as input output device. What is given initially is a input, the non blank portion. And when you halt what is given is, the output. If it is not going to halt, it is going on for ever in a loop.

(Refer Slide Time: 13:43)

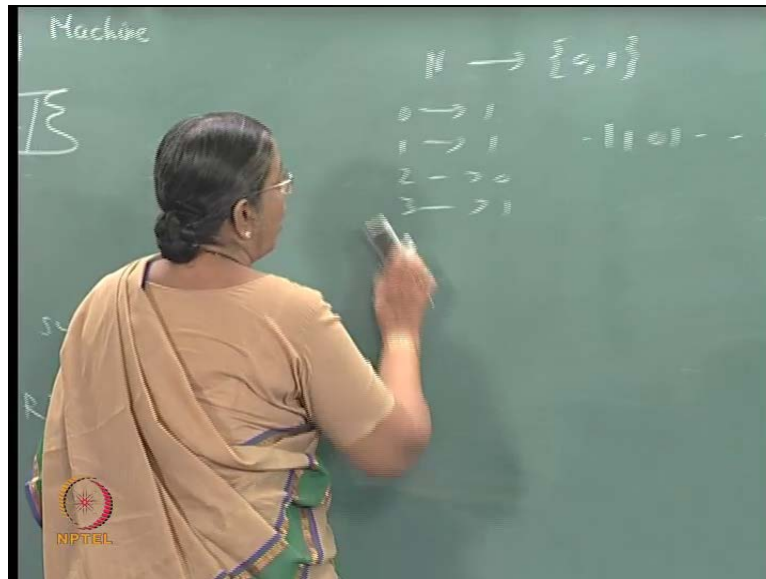


Now, initially you may have a some number and a ultimately you may end up with another number. For example, you may have something like this. You may have a number n , it is not in A 1 cell some number say, if I binary or a decimal you can have does not matter. So, it is another number m and n and you may write the Turing machine. So, that ultimately, you come up with m plus n or multiplication m into n and so on. So, it is computing a function from the set of non negative integers, to non negative integers. We can also look at like this.

So, given N , initially some number, here there are two numbers. So, it actually it will be N cross N from N cross N to N cross N . Anyway from natural numbers depending upon the arguments, it is computing a function. Now, can you compute anything with this. You cannot compute there are things which cannot be computed by the computer or current day computer or with the Turing machine. The set of computable functions there are functions which are not computable. Why, because the set of, it is set of functions itself from natural numbers to natural numbers.

If you take that itself is uncountable, whereas the number of Turing machines are the number of a effective procedures. If you make A 1 Turing machine correspond with A 1 effective procedure, that is countable. One will correspond to something like a real number another will be natural numbers. And you know that they cannot be put to one to one correspondence. That is what that shows the existence of things which cannot be computed.

(Refer Slide Time: 16:32)



So, for example, take like this how would you know that there are functions which cannot be computed un computable functions. For example, just simple take from all functions from n to $0, 1$ set of non negative integers to $0, 1$. I can represent it as a real number for example, a 0 goes to say 1 one goes to a 1 2 goes to 0 3 goes to 1 like that something one function. This you can represent as say for example, point 1 one 0 1 like this it is a real number. So, if you look at all functions from n to $0, 1$ you can make them correspond to say real numbers. And you know that the set of real numbers is uncountable, but the number of Turing machines or the numbered of effective.

Any Turing machine you can describe in a proper way. Like six tuple will use $k, \sigma, \gamma, \delta, q, \text{naught}$ a for whatever it is, that you can write as a string over some alphabet and strings, over an alphabet. In the standard representation, you have lexicographic ordering and standard ordering which we studied lexicographic ordering. You I mean, it is not a well in the sense that you that may be elements, which comes (()) that and we cannot define. But standard ordering you can enumerate these strings, any alphabet you can enumerate these strings over that alphabet. This argument we are going to study again and again.

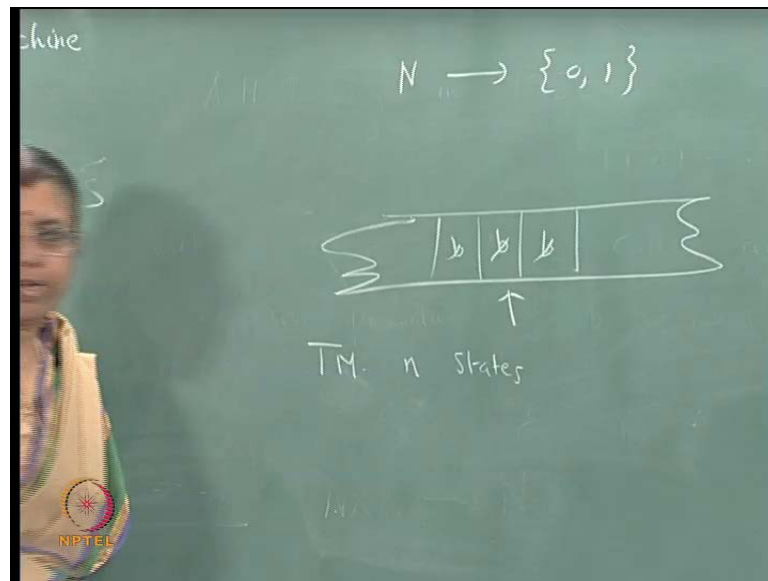
That is why, I always keep mentioning that you should not miss the class because

whatever we may say in one class, it will be carried on to the next class. And if you miss the classes, you may not be able to follow what is that in the next class. So, what happens is this functions you can make them correspond to the set of real numbers. Whereas, the procedures, effective procedures you can described as a Turing machine and a Turing machine can be represented as a string. Some string representation you can give over some particular alphabet. And any over particular alphabet the strings are enumerable you can enumerate them. So, they correspond to when something is countable it correspond to the set of natural numbers.

So, you cannot have a one to one correspondence or some correspondence between real numbers and natural numbers. So, there are functions which could not be computed by some effective procedures. So, there are un computable functions. Now, I use to say in the for a example, can you do anything with a computer? There are certain things which you cannot do with a computer. For example, if I say you want to list all the simple sentences in English, meaningful single simple sentences in English, you cannot do that. Because that meaningful is that you cannot write a program to do that.

What the computer can do only if you can write a program for that. If you are not able to write the program, you cannot do that. So, another function which is not computable I will give an example, I if you are not able to a follow this at this movement, do not worry. I will come to that again after two three classes.

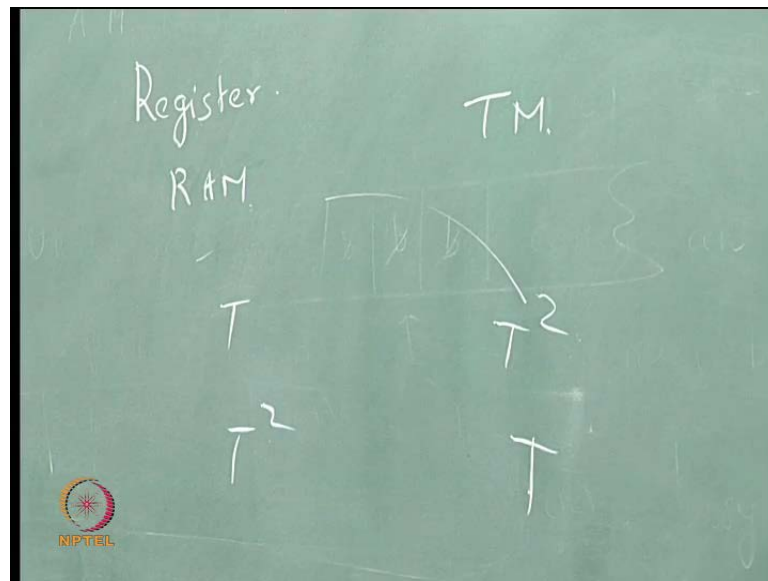
(Refer Slide Time: 20:20)



You start with the infinite tape, a Turing machine with blank symbols, whole tape is blank. And you have a Turing machine with n states can have n states. You start the machine anywhere you want. It should print ones and ultimately halt. There may be several Turing machines like that with n states. It should print some ones on the tape and then ultimately halt please remember. That you can have a Turing machine which keeps on printing 1 1 1 1 and going for infinity without stopping. But ultimately this machine has to halt. So, with n states I can have several machines. So, what is the maximum number of ones which could be printed on a tape, when you start on a blank tape with a Turing machine with n states.

There is the question. Suppose I denoted by some f of n , this is called the busy beaver function. (No audio from 21:44 to 21:56) It is an uncomputable function. Why it is not uncomputable in during the course? In the next set, we are four lectures after 3 or 4 lectures, we will see why this is not computable. So, they are exist functions which cannot be computed. Now, what Turing said is whatever could naturally be described as an effective procedure can be realized by a Turing machine. So, if you can write something as a program or step by step you can implement it on a Turing machine. The difference between a Turing machine and the current computer is Turing machine is not random access whereas, the current machine is current thing is random access.

(Refer Slide Time: 22:52)



So, you talk about register machine or random access machine. This the again I will deal within at later stage in detail register machine or random access machine. And in another direction, we have Turing this is exactly go and fetch some information. That is what you do in a computer, is not it? In Turing machine the same thing you can do, but it is divided into cells and if I want some information from here and I have to get it here, I have to travels like this go and get. So, that is the only difference and it has been proved that you can simulate one with the other time will be only at the most quadratic. If something you can do it in time T , you can do it in time T squared and something if you can do in time T , here you can do with t squared maximum.

So, if something can be done in polynomial time and the other one model also it can be done in polynomial time. The non polynomial or polynomial time is not going to be a affected which model you considered. So, generally for all computational models people have taken Turing machine. For all proofs, I will and there are other model. This model of Turing machine even though, it was stated in 1936, all these yes it has stood the test of time. And any other model which you define like a partial recursive functions or post canonical systems other things types 0 grammars. Any other model people have divined they are shown to be equivalent to that of Turing machine.

Any other model of computation it will have defined the it has been shown that it is equivalent to Turing machine. These are known as conventional models of computation. Turing machine registry machine canonical system type 0 grammar partial recursive function. Currently the new trend is you have unconventional models of computing. What are they? D N A computing, membrane computing, molecular peptide computing, quantum computing these are called unconventional models. Then when in define a new model and try to do something the experiments also a people try to do with quantum computing d n a computing and so on.

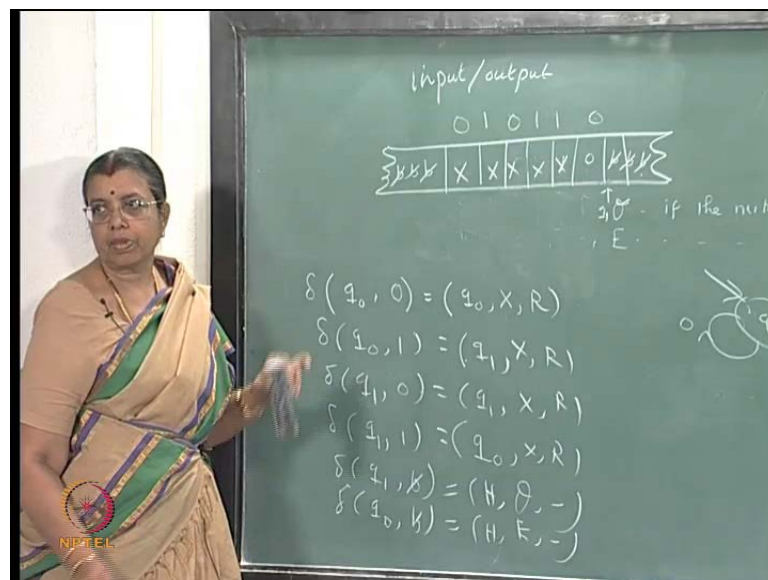
But experimentally small things they are trying now, but when you go to a new model you have to prove that whatever is possible with the original model which is possible. So, you have to show that it is universal, universal computation is possible. Or what you mean by that is whatever you can do with a ordinary computer or Turing machine, it is possible. So, generally people try to simulate a Turing machine or in a sense type 0 grammar or something like that. So, there is a trend when you define a new model, you have to show that it has the capability of that of a Turing machine. Before you proceed further, it is a full model of computation means you should do that.

So, this is and a slight deviation two things in the first half of 20th century where break through results. One was gordons incompleteness theorem, the second is halting problem of the Turing machine. These are two break through results. Because of that, people were able to show that certain things people were trying where you cannot write algorithm for this.

For example, given a context free grammar, is it ambiguous or not? It is undecidable problem. And earlier Hilbert post several problems. One of the problem this called Hilbert strength problem, that is given a polynomial with integer coefficient. That is it have a integer roots. It may have or it may not have, can you write an algorithm which will take as input? A polynomial with integer coefficient tell you that the roots are integer or not. And people try to write an algorithm for that they were not successful, but after Turing stated his un solvability result. It became known that you cannot write an algorithm, told you it is a very strong statement to say that you cannot write an algorithm that has to be proved. So, we will deal with those things after the 4, 5 classes.

Now, coming back and another deviation again to tell you this also. I do not know whether I have told you A M Turing is the British person. Even today there is an award being given in his name. And he has been, I mean Turing award a lecture people give a person ((C)) very good work in computer science is being awarded with that. He had his earlier education in south India, Bangalore, ooty and places like that. So, I always tell the class that you I I T people, you also have your young earlier education in India. And you go abroad and do very well. So, the school education or something like that may be India has much, much better than what is given in U S.

(Refer Slide Time: 28:33)



So, let us go to the formal model and consider one or two example. So, the it has an initially I will take it as an input output device (No audio from 28:36 to 28:48). You are having binary number a binary string something like that the rest of it is padded the tape is padded with blanks. (No audio from 29:03 to 29:10) Initially, if the Turing machine has a tape and a finite control, it is just similar to finite state automaton. There are two differences; one is the tape is infinite here, second is you can rewrite a symbol over a symbol red one the tape. So, this machine it reads this whole thing and in the first blank symbol it writes a 0 if the number of ones, is solved. If the I am I o and 0 to differentiate I am writing it like this the number of ones, is odd.

And it writes a E if the number of 1's is even. We had a finite state automaton for this. If you remember now, it is going to read and do something here, print something here O or E. How will it do finite control is again state. So, in state q_0 if it reads a 0, it remains in q_0 and prints X and moves right; the mappings will be like this. We are considering deterministic Turing machines now. Next, you move is uniquely determined non determinist Turing machines are also there we will come to that later. So, what we does is in state q_{naught} if it sees A 0 it changes that 0 into A X and moves right. In state q_0 right in state q_0 if it sees A 1 it goes to state q_1 . And prints A X and moves right. I have here is the string let me keep the original right the original string.

Because you know that ultimately you should get odd this was the original s tring. Now, in state q_1 it reads A 0 in state q_1 it reads A 0 it remains in state q_1 and moves right. Each time it reads it is replacing it by X you can also retain it you can also retain the input no problem slight variation you have to do here. Now in q_1 it sees a 1 when it sees a 1 in q_1 state it goes to q_{naught} it goes to q_0 changing the symbol by X. And moving, essentially like a finite state automaton. (No audio from 33:12 to 33:27) This is the a sense of it, but you are rewriting that as X and then also move. So, ultimately if the machine in is in q_0 what does that mean it has seen. If it is in q_0 t what does that mean (()). It has seen even number of (()) if it is in q_1 . (()).

It has seen odd number of it is. So, here again in q_0 it sees a one. So, in q_1 it will move right in q_1 if it sees a 0 in q_1 it will move right. So, the first blank symbol after the input in q_1 when it sees a blank symbol; that means, it has come across odd number of ones. So, it will halt printing A o it will go to the halting state and print A o. In q_1 if it sees in q_{naught} if it sees a blank; that means, it has seen even number of ones. So, it will halt printing A E. This could be done with the finite state automaton also. So, as a Turing machine how the machine works this is what we want to know and this can be done. You can also have a set of a state diagram for this slightly different from this it is almost like this, but our some difference will be there this I will consider later.


(Refer Slide Time: 35:23)

Parity Checker

States $\{q_0, q_1\}$

Input alphabet $\{0,1\}$


Tape alphabet $\{0,1,\not{0},X\}$



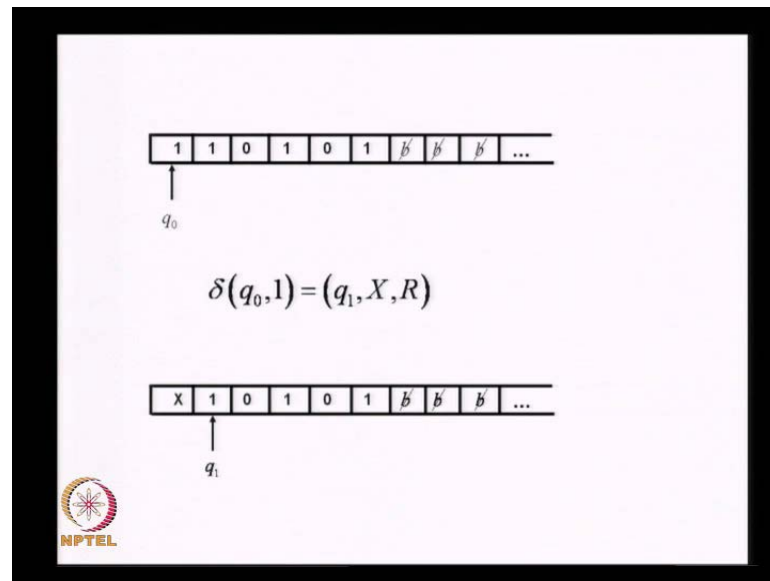
So, let us consider one more string which will check the parity of number. We have already seen that the parity checker has two states the input alphabet 0 1 and the tape alphabet 0 1 blank and X. The mappings are given as follows this also we have already seen.

(Refer Slide Time: 35:45)

Mappings

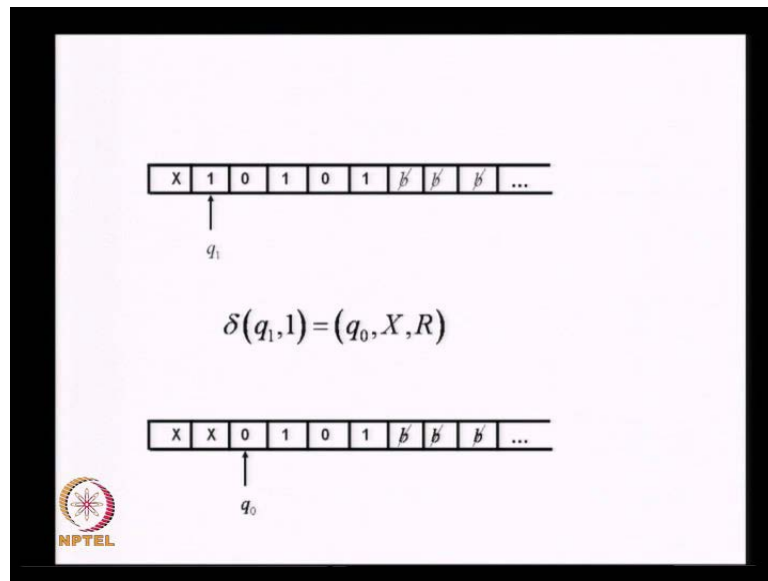
$$\delta(q_0, 0) = (q_0, X, R)$$
$$\delta(q_0, 1) = (q_1, X, R)$$
$$\delta(q_1, 0) = (q_1, X, R)$$
$$\delta(q_1, 1) = (q_0, X, R)$$


(Refer Slide Time: 35:52)



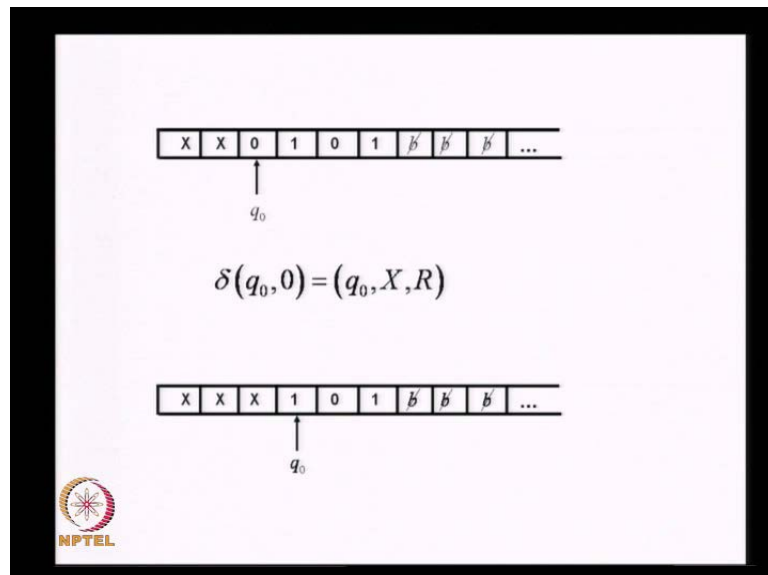
Let us consider the string 1 1 0 1 0 1 it has got even number of 1s. So, after reading this string it should print A E. Let us see how the machine works on this string initially it is reading the symbol 1 in state q_0 . And we use the mapping $\delta(q_0, 1) = (q_1, X, R)$. So, the next id becomes like this. So, it moves 1 cell to the prints A X over the 1 and moves in state q_1 . So, whenever its reads a 1 it changes state from q_0 to q_1 or q_1 to q_0 , but when it reads A 0 it does not change the state.

(Refer Slide Time: 36:39)



So, from this id when it reads A 1 in state q 1 it uses the mapping delta of q 1 1 is equal to q naught X R. And the next instance it goes to the next cell reading A 0 in state q 0.

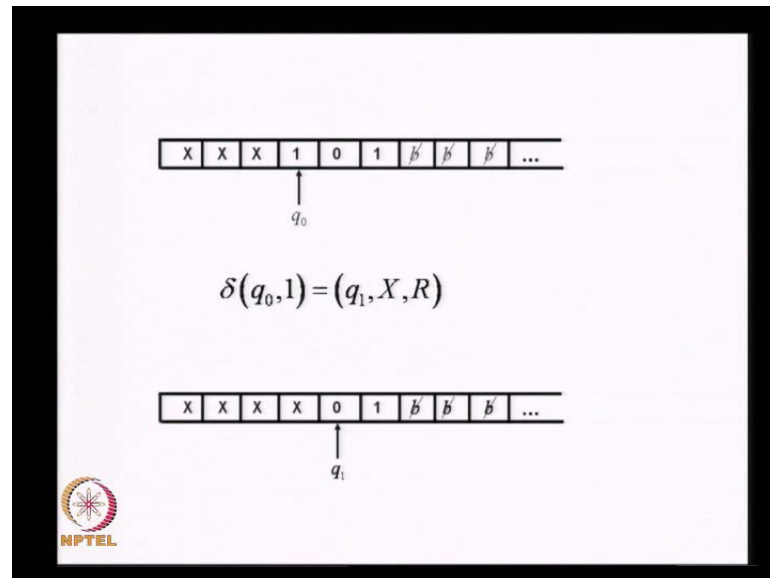
(Refer Slide Time: 36:54)



Next when it reads 0 in q naught it uses this mapping delta of q naught 0 is equal to q naught X R. And the next I d becomes this. It moves one cell to the right and goes to the

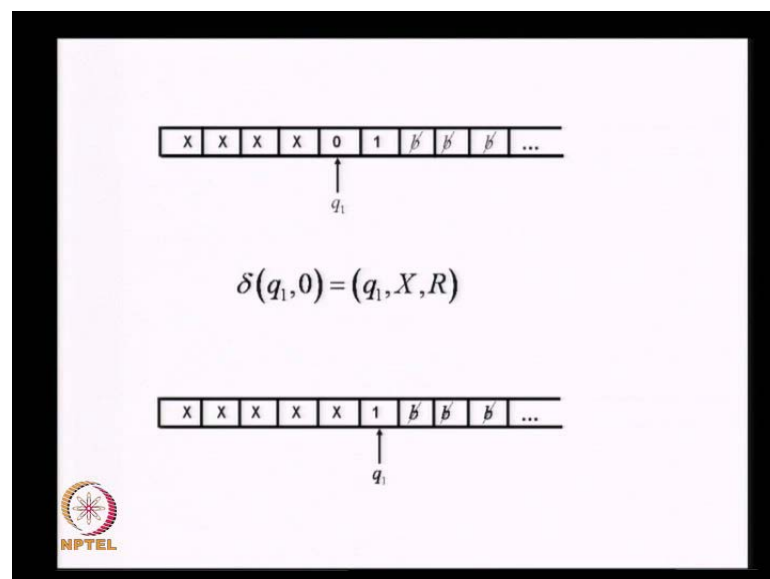
next cell in state q naught because it is reading A 0 does not change the state.

(Refer Slide Time: 37:16)

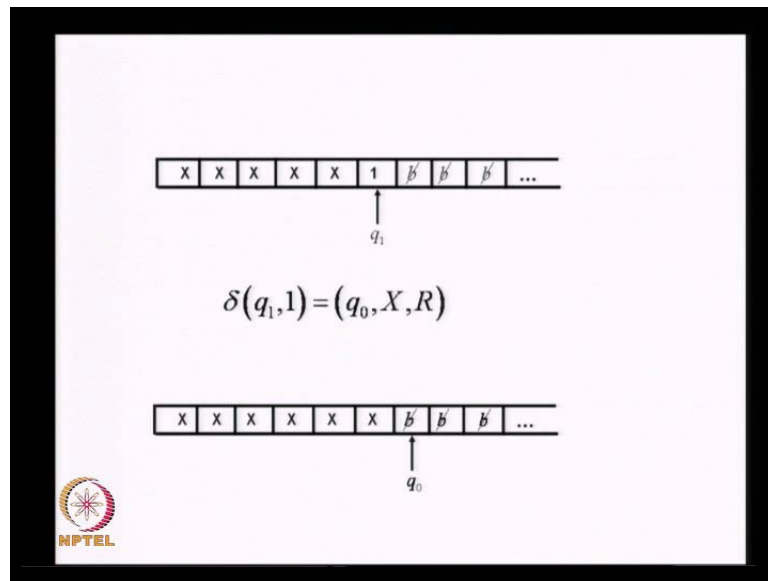


Now, in q naught it reads a 1 and uses the mapping delta of $q_0 1$ is equal to $q_1 X R$. And the next id becomes this proceeding like this in the next instance it reads A 0 in q_1 uses this mapping goes to the next cell.

(Refer Slide Time: 37:30)

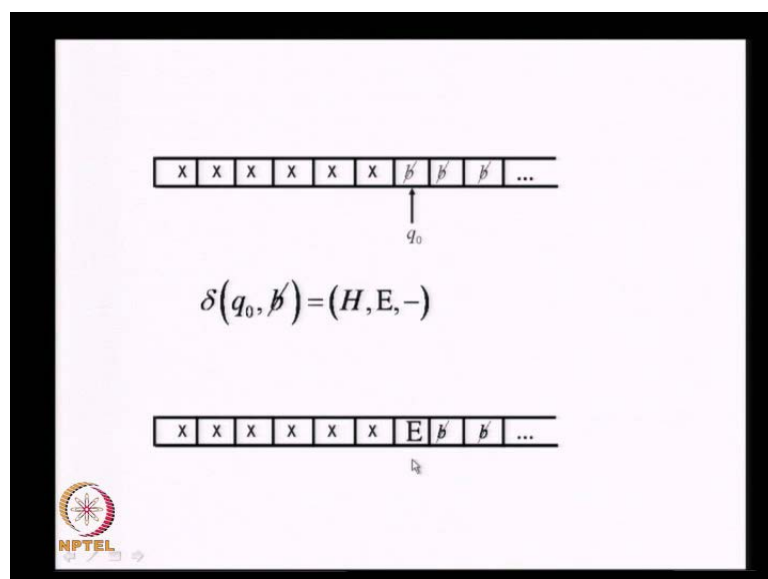


(Refer Slide Time: 37:39)



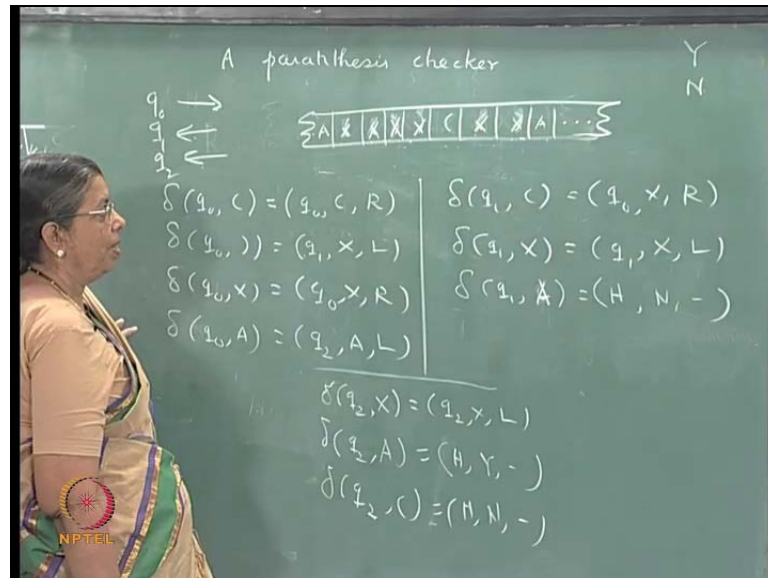
And then reads the one in the next instance changes the state and goes to the next right cell. So, it reaches the first blank symbol in state q naught if it reaches the first blank symbol in state q 0; that means, it has encountered an even number of ones. If it has reach this cell in state q 1; that means, it has encountered an odd number of ones.

(Refer Slide Time: 38:06)



So, depending upon that it has to print E or A O. So, in q naught it sees a blank and uses this mapping. And prints a E over this cell that is over this blank cell it prints A E which denotes that it has encountered an even number of ones.

(Refer Slide Time: 38:29)



That I will consider one more example parenthesis checker (No audio from 38:30 to 38:40) given a string of parenthesis is it well formed or not. If it is well formed it will stop printing A y yes it will stop printing a y if it is not well formed it will stop printing a no it is not well formed. Let us design the machine for this. So, let me first look at it the way the machine works can this we done by a finite state automaton. Can this we done by a finite state automaton can this we done by a pushdown automaton yes it can be accepted by a pushdown automaton.

Now, as a Turing machine how it works? Suppose I have something like this is this well formed or not is this well formed, it is well formed. So, I also want to have some end marker this A and A is end marker. Between the end marker sometimes for convenience I want to have an end marker. So, between the end markers and these are all blanks the string is given and I have to test whether it is a well formed string or not. Again the machine starts in state q naught reading this symbol the idea is it moves right and replaces the first right parenthesis by I X.

And moves left converting the matching left parenthesis into A X. It does it again and again ultimately it when it reaches there no more right parenthesis are there and it comes back and checks whether there are no more left parenthesis. So, this is what it does. So, the mappings will be like this delta of q 0 left parenthesis is q 0 left parenthesis and right. So, it just moves right in q 0 again the same mapping will be there. Now, it comes here in state q 0. So, q 0 is a right moving state q naught is right moving state. Now, when it sees right parenthesis in state q naught what will be the mapping it changes that into a X and starts moving left. So, it starts moving left in state q 1.

Now, in q 1 it sees a left parenthesis the matching right parenthesis matching left parenthesis for the right parenthesis it has converted into A X. So, it changes this into X and moves right in q naught change again to q naught. So, q 1 actually it moves left it is a left moving state. (No audio from 42:22 to 42:30) So, that is q 1 I will write for q 1 here delta of q 1 left parenthesis is q naught X right. Now, in q naught it sees a X it has to just travels over the X. So, delta of q 0 X is equals to q naught X R it just moves right. So, it moves right and over this left parenthesis also it moves right. And here it will change this into A X and starts moving left.

Now, in q 1 it will encountered this left parenthesis. So, it will change that into X and move right in q 0. So, in q 0 it moves right when it comes here sees a right parenthesis. So, it changes that into X and moves in q 1 now q 1 it should move left searching for the matching left parenthesis. So, it go through all the Xs. So, delta of q 1 X is q 1 X R, then in q 1 when it sees this left parenthesis changes that into X and starts moving in q naught. So, in q naught it will move and changes into A X. And it will move and in q 1 it will change this into X and again move in q 0. So, in q 0 when it is moving it is suppose to see a right parenthesis as long as they are the. But now everything is over.

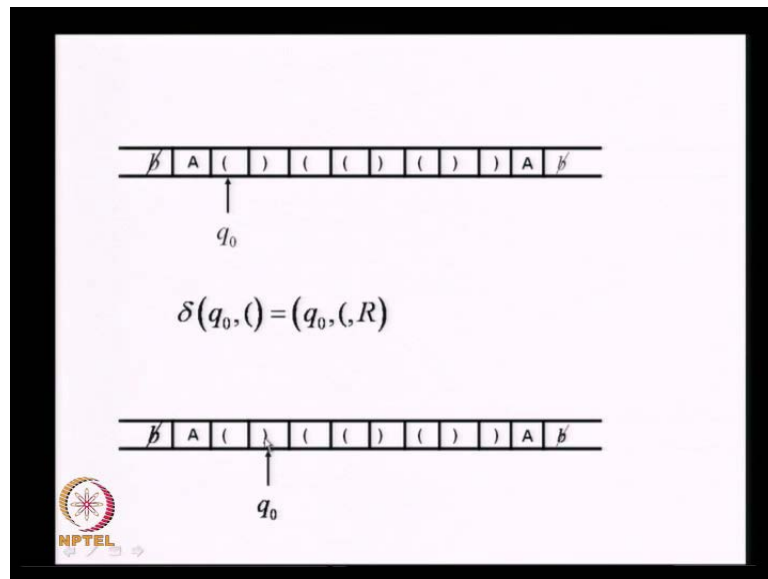
So, in q 0 it will see a A what does that mean all right parenthesis are exhausted. So, in q 0 if it sees a A it goes to another state q 2 it does not do anything with the A, but it starts moving left. Now, it should come back here and find out that no more left parenthesis are there. So, in q 2 what it does this is for q 1 in delta of over the Xs it has to move left. Q 2 is also a left moving state q 2 X is q 2 X 1 and in q 2 if it sees A; that means, there are no more left parenthesis (No audio from 45:46 to 45:52) it halts saying A yes. Now, if the

string is not well formed it has to say no and halt. When there are more right parenthesis also. So, suppose I had one more right parenthesis.

So, the machine will be moving here the Turing machine will be moving here and in this state it will print an X and start moving right in state q 1. In q 1 when it is moving left it is suppose to see a left parenthesis, but instead if it sees a; that means, the number of left parenthesis is less. So, in q 1 if it sees a left parenthesis then it halts saying a no I am sorry if it sees a A that means, number of right parenthesis is more. Now, what will happen if you have more left parenthesis suppose I take an input like this I have something (No audio from 47:36 to 42:52) like this how will the machine work it will change this into a X move left changes into X move right changes into X move left changes into A X. Move right changes into A X move left changes into X and move right..

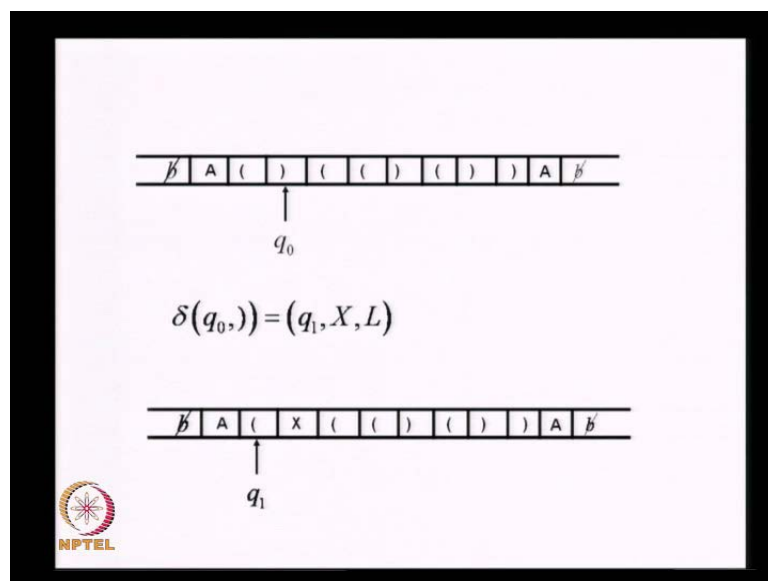
When it sees a A that means, there are no more right parenthesis. So, it will go to q 2 and in q 2 it will starts moving left. When in q 2 it sees a left parenthesis means there are more left parenthesis. So, delta of q 2 left parenthesis is halt. So, this is the mapping for the Turing machine. So, this Turing machine finds out whether a given string of parenthesis is well formed or not. If it is well formed it prints A yes if it is not well formed it prints A no. But where is that N, printed yes will be printed here right if there are more right parenthesis no will be printed here, if there are more left parenthesis no is printed here. It is not printed here it is printed here we can slightly modify the machine and make it printed there does not matter just make it move and finally, printed.

(Refer Slide Time: 49:35)



Let us consider one more string and see how the parenthesis checker works on this string this is the given string. And it starts on this symbol in state q_0 on the left parenthesis and it uses this mapping and moves in state q_0 .

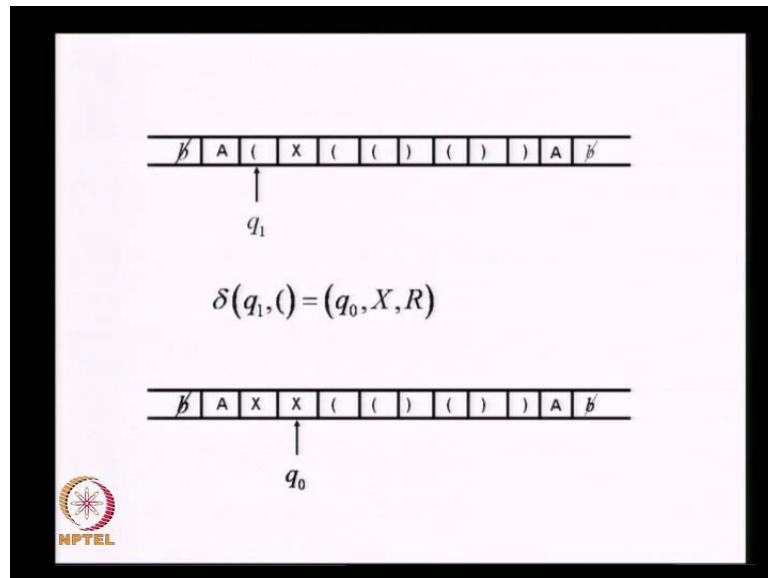
(Refer Slide Time: 50:04)



So, the next id becomes this in state q_0 it sees a right parenthesis and changes that into A

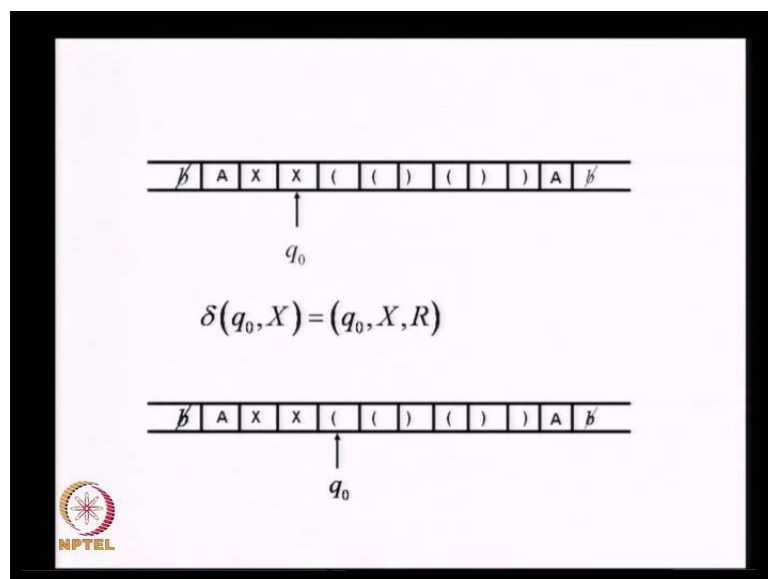
X and goes to state q 1 and moves left it uses this mapping and. So, it prints A X in this place and moves left in state q 1.

(Refer Slide Time: 50:17)



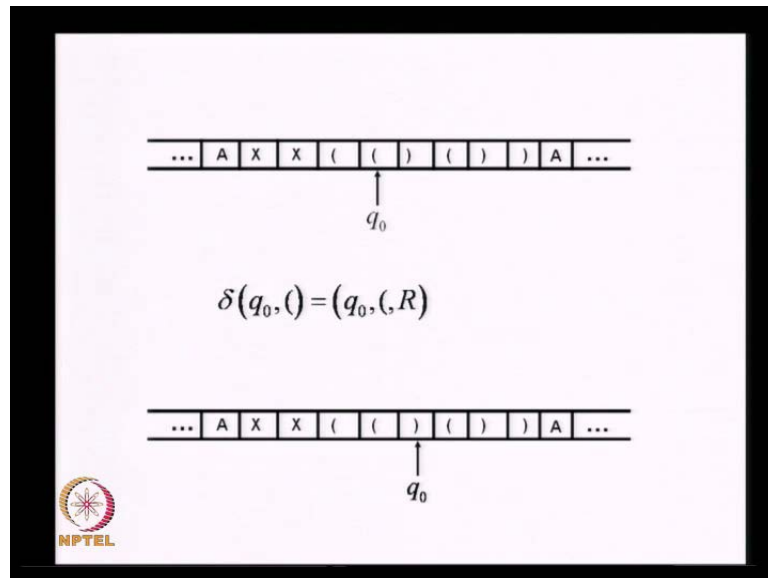
So, this is the next id and in q 1 it sees a left parenthesis it changes that into A X using this mapping and starts moving right in state q 0.

(Refer Slide Time: 50:35)



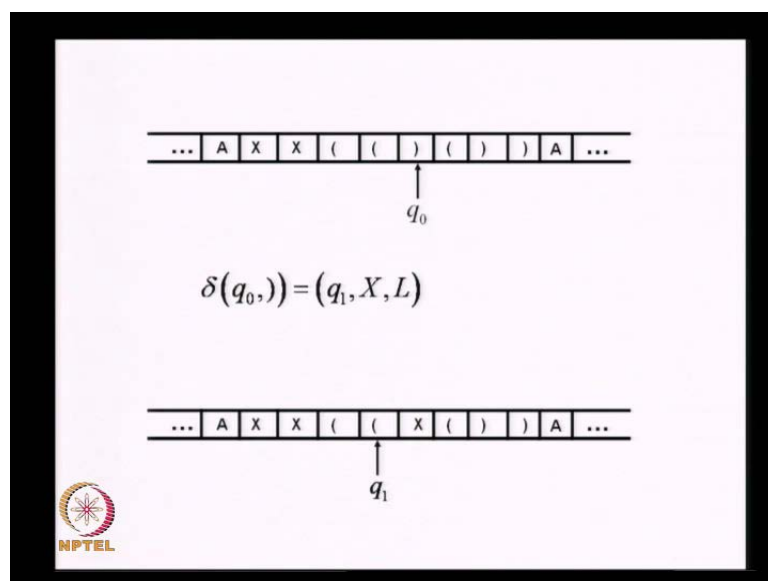
So, this becomes the next id and from this id it go it reads the X and just moves right.

(Refer Slide Time: 50:44)



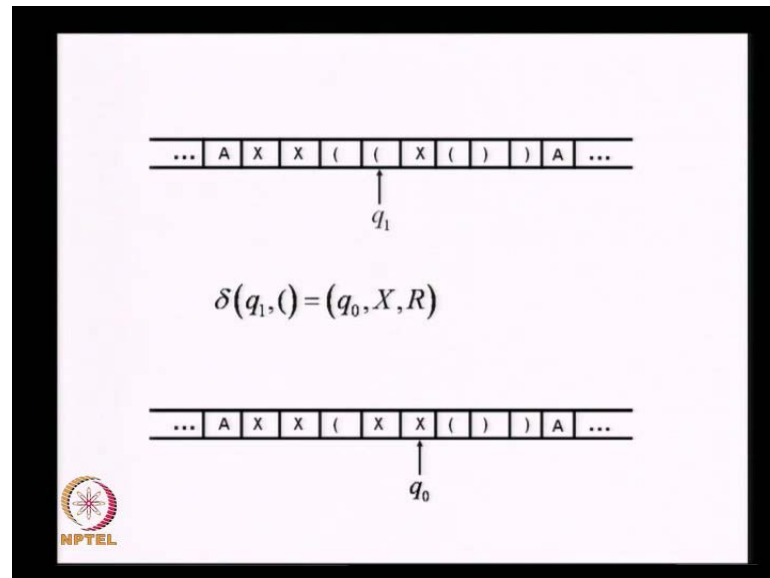
And it moves right in state q naught looking for the next right parenthesis. These all just right moving and here it just moves right using this mapping.

(Refer Slide Time: 51:00)



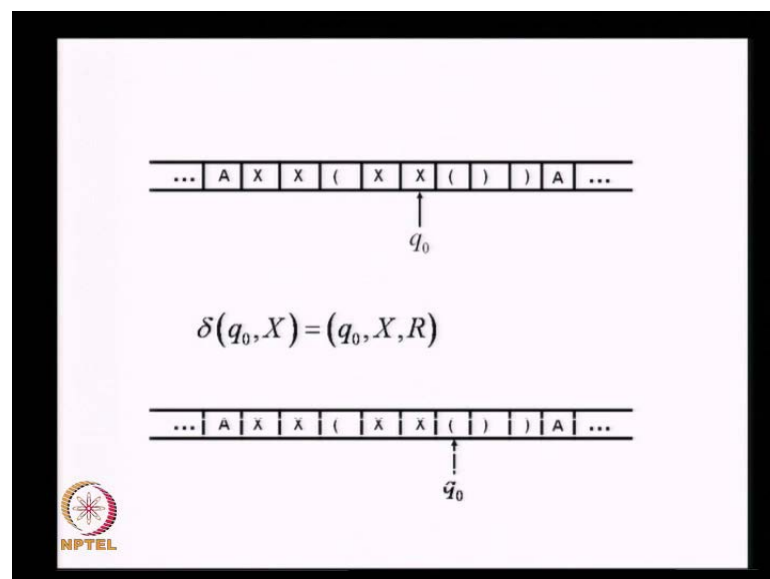
And this is the next id and from this id it changes the right parenthesis into A X and starts moving left. The mapping used is this and it moves left in state q 1.

(Refer Slide Time: 51:12)



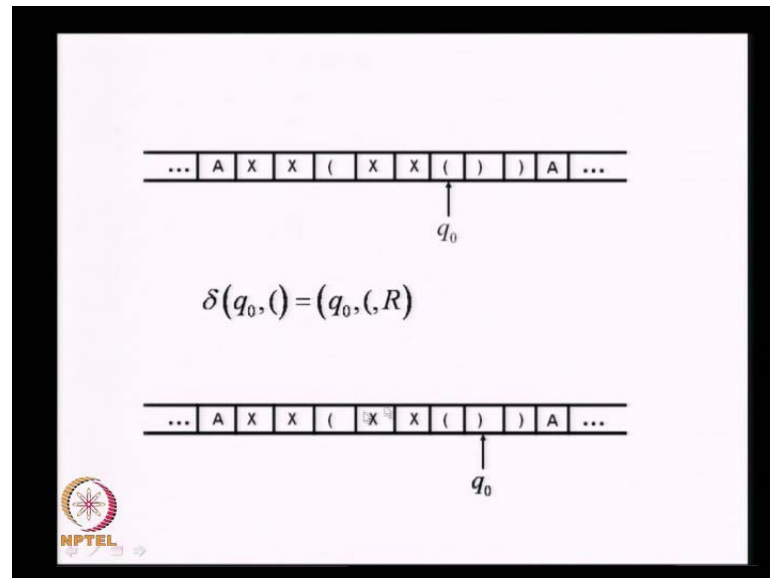
Now, in state q 1 it sees a left parenthesis using this mapping it changes that into A X and starts moving right.

(Refer Slide Time: 51:23)



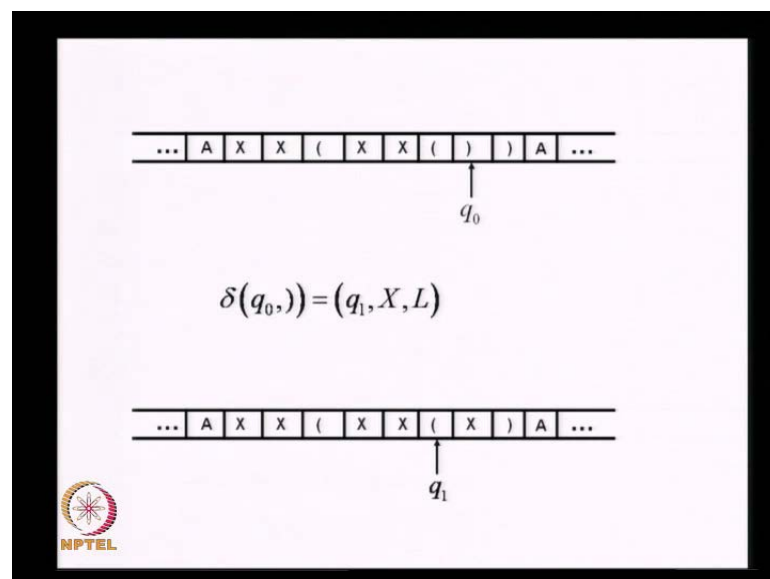
And it starts moving right till it sees the next right parenthesis.

(Refer Slide Time: 51:33)



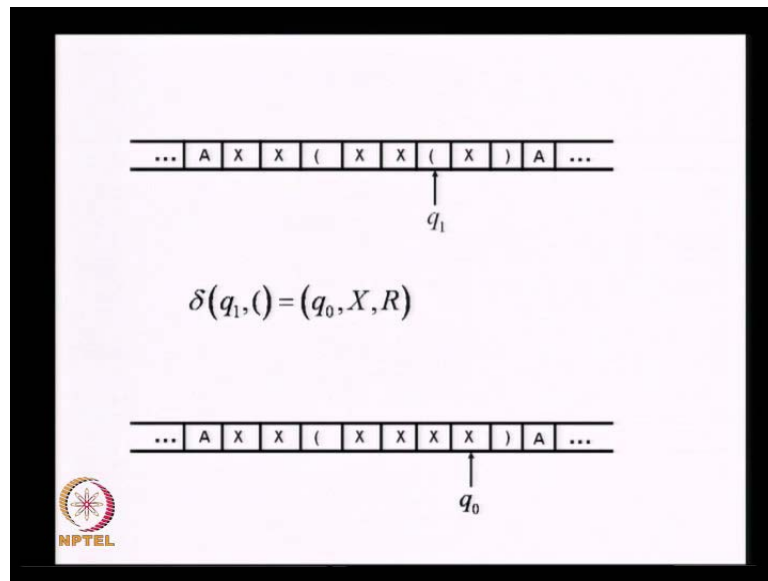
So, now, in state q_0 it sees a right parenthesis.

(Refer Slide Time: 51:39)



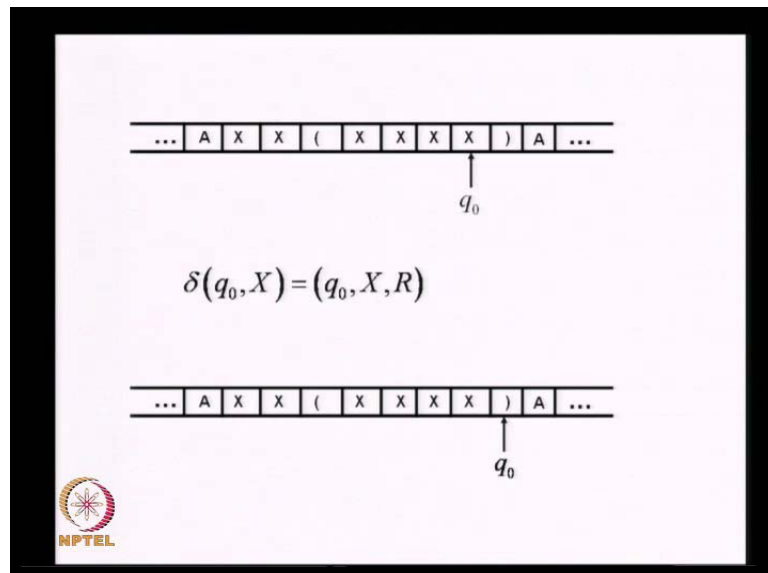
It changes that into AX using this mapping and moves left in state q_1 .

(Refer Slide Time: 51:46)



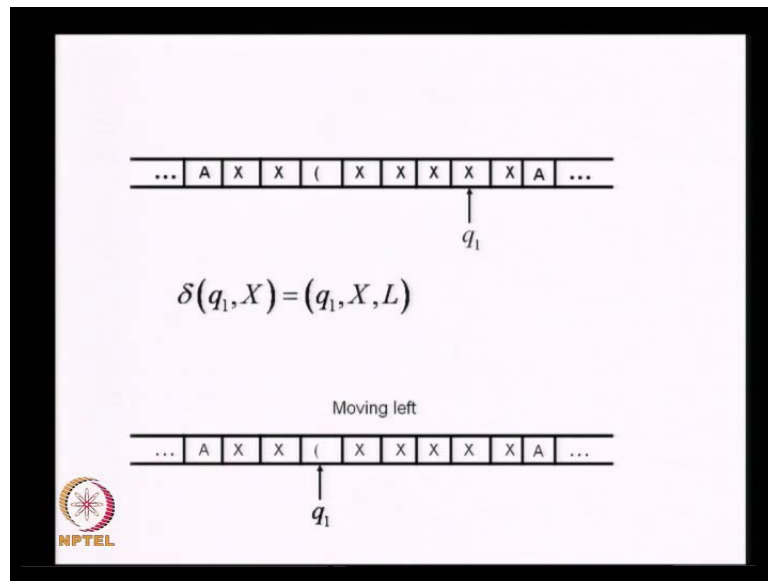
And here it reads the left parenthesis and makes use of this move and changes that into A X. Again it moves right looking for the next right parenthesis.

(Refer Slide Time: 51:57)



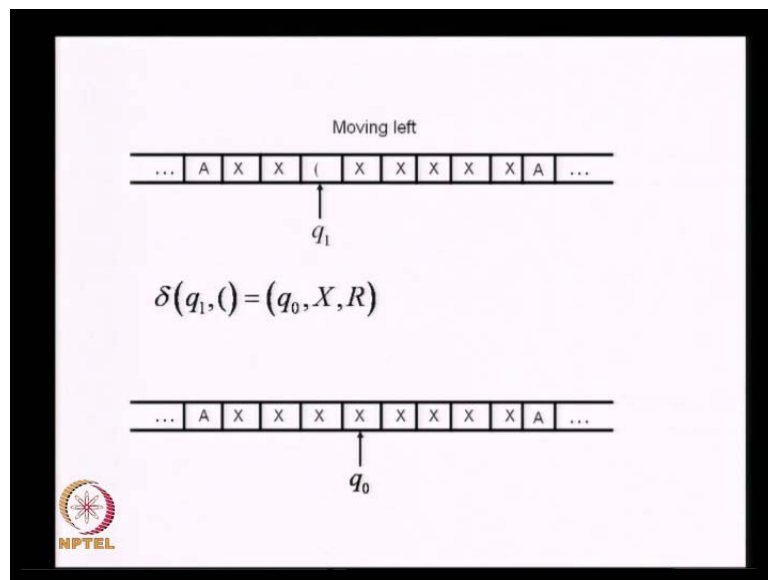
And changes that into A X using this move starts moving left looking for the next matching left parenthesis.

(Refer Slide Time: 52:10)



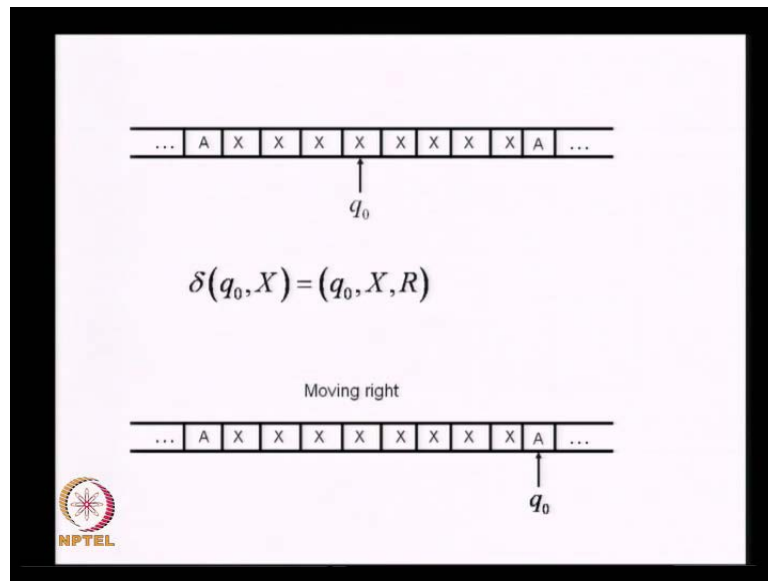
So, these are left moving moves and it moves left until sees the matching left parenthesis.

(Refer Slide Time: 52:22)



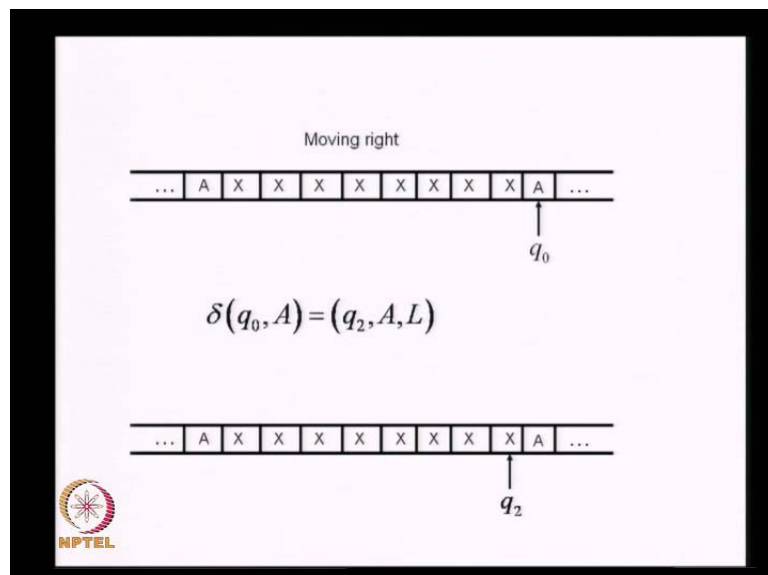
Now here it sees the left parenthesis is state q_1 changes that into A X using this move, then starts moving right.

(Refer Slide Time: 52:36)



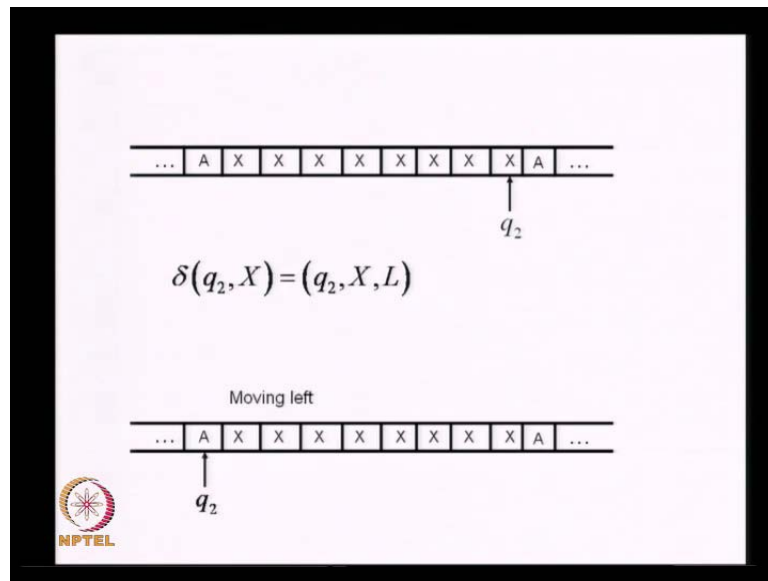
It keeps on moving right in state q_0 until it sees the end marker A. And it encounters A in state q_0 .

(Refer Slide Time: 52:48)



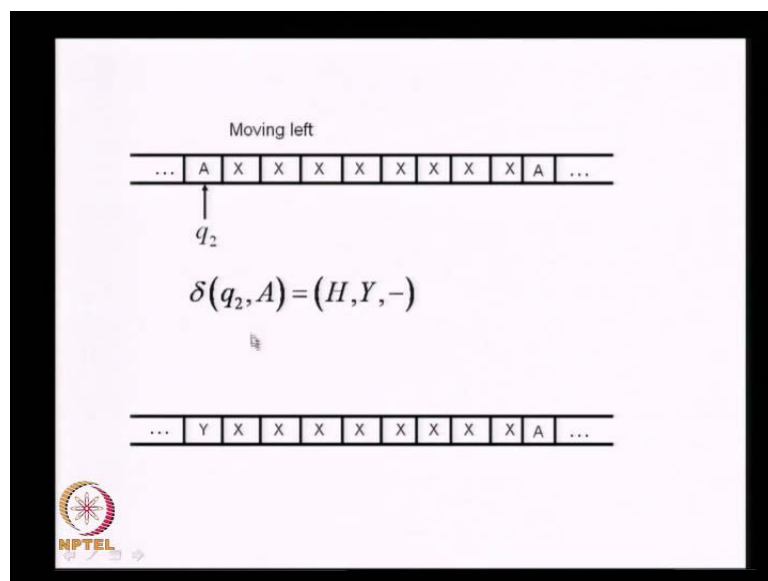
Now, when it encounters A in q_0 that means, it has seen all right parenthesis there are no more right parenthesis left.

(Refer Slide Time: 53:08)



So, it has to check whether there are no more left parenthesis it goes to q_2 . And keeps on moving left looking for A left parenthesis if there is no left parenthesis, and it reaches the end marker A in state q_2 .

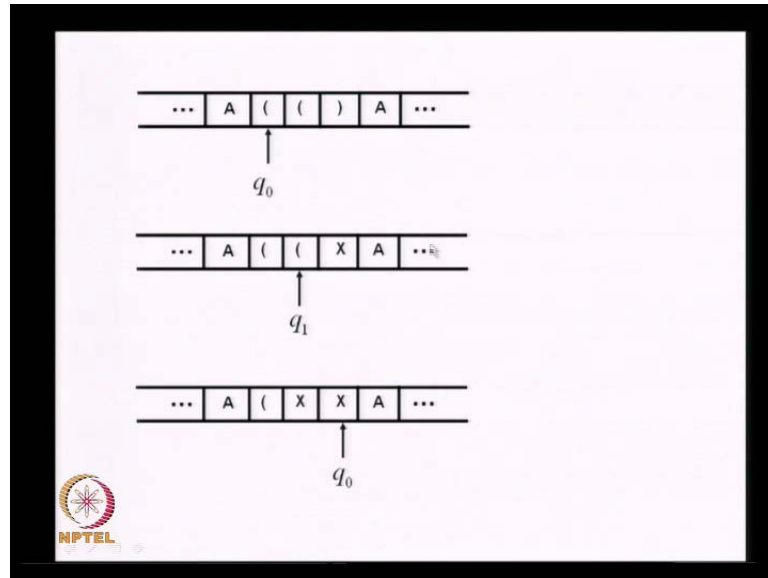
(Refer Slide Time: 53:20)



That means, it has matched all the right and the left parenthesis and. So, it prints A yes at

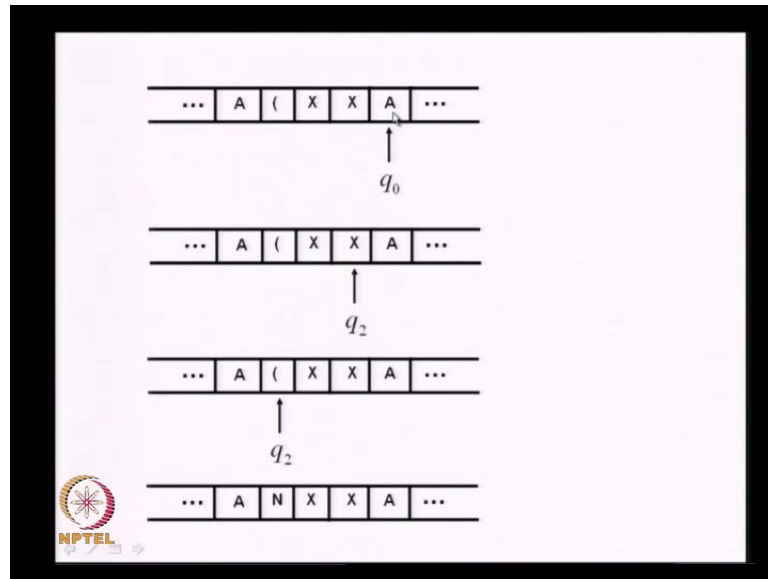
this stage it prints A yes here telling us that the given string is a well formed string of parenthesis.

(Refer Slide Time: 53:51)



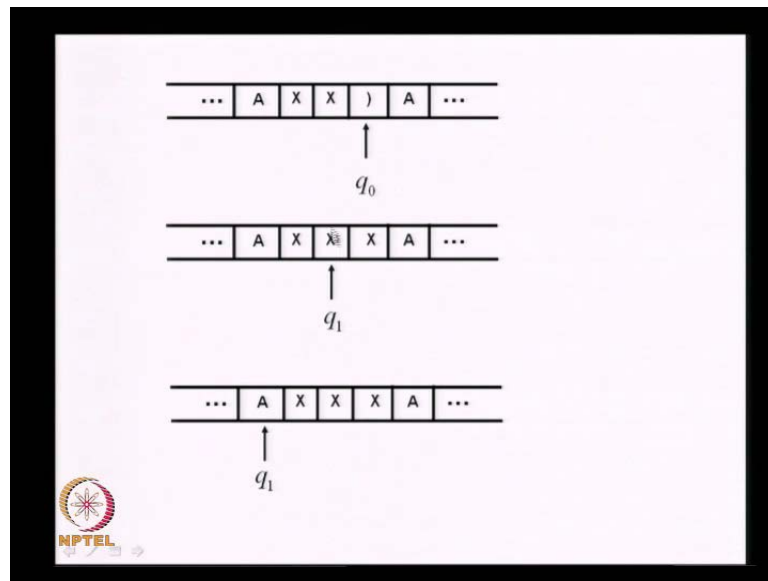
Now, what happens when you have more left parenthesis or when you have more right parenthesis. Look at this case where you have two left parenthesis and one right parenthesis you have one more left parenthesis. So, how does the machine work it moves right looking for the right parenthesis. And changes that into A X moves left looking for the matching left parenthesis changes into A X and moves right.

(Refer Slide Time: 54:16)



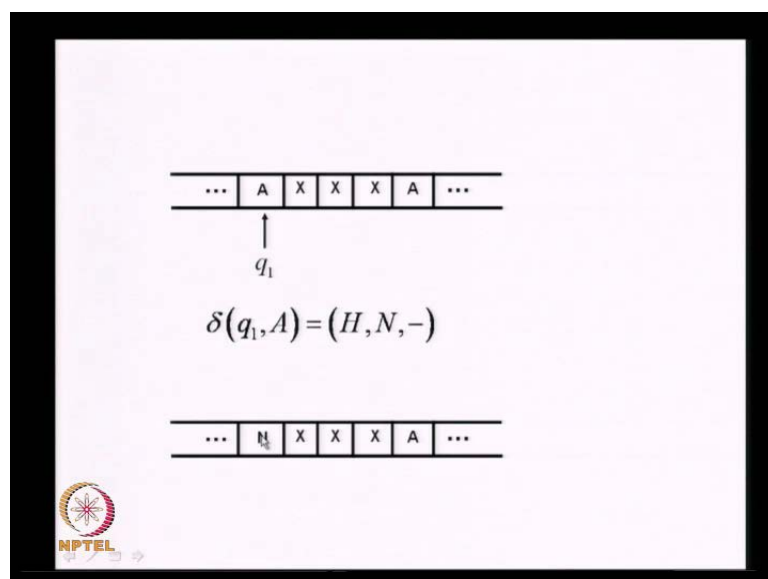
Now, in q_0 it sees a A that means, it has exhausted all the right parenthesis it moves left in state q_2 looking for the A. It has to reach the A in state q_2 where as it encounters a left parenthesis in state q_2 that means, there are more left parenthesis and. So, it prints a no telling us that the given string is not a proper string. Now, this no is printed in the place where you have more left parenthesis where as the yes was printed at this position. Now, what happens when you have more right parenthesis. So, this is the given string where you have more right parenthesis let us see how the machine works on this. It starts moving right looking for the first right parenthesis changes that into A X moves left.

(Refer Slide Time: 55:14)



And then changes the left parenthesis into X moves right. And then changes this into X and moves left in state q_1 in state q_1 when it is moving left it is looking for the matching left parenthesis. But it does not find one it reaches the A in state q_1 that means, there are, no matching left parenthesis for the right parenthesis changed last.

(Refer Slide Time: 55:46)



So, it prints a no at this position making use of this move it prints a no here. Note that when there are more right parenthesis the no is printed at this position. When there are more left parenthesis the no is printed wherever you find more left parenthesis. And when the given string is a proper string yes is printed in this position. So, now you know how to construct Turing machine for certain purposes. So, we can consider some more example as a homework you can take this please try this. And come for the next class which is given a Turing machine it contains the see the input has to be like this. Input is this is all blank blank, blank some unary number is given. Unary to binary converter the unary number is given here and ultimately the answer has to be given here.

The binary number what is this eight no. So, you should have 1 0 0 0, but a in here you can print it as b a a a something like print here. The a corresponds to 0 b corresponds to 1 given a unary number here this is all blank sorry ultimately the binary number corresponding to that will be printed here. But a instead of using 1 and 0 you will use b and a to distinguish between this one and other one you can use 1 0 1 also there is a matter. But convenience sake you can do that try to do this, and the reverse direction also you can do given a binary number as. The input design a Turing machine which is ultimately end up with a unary number for that. So, these two you try and we shall discuss them in the next lecture.