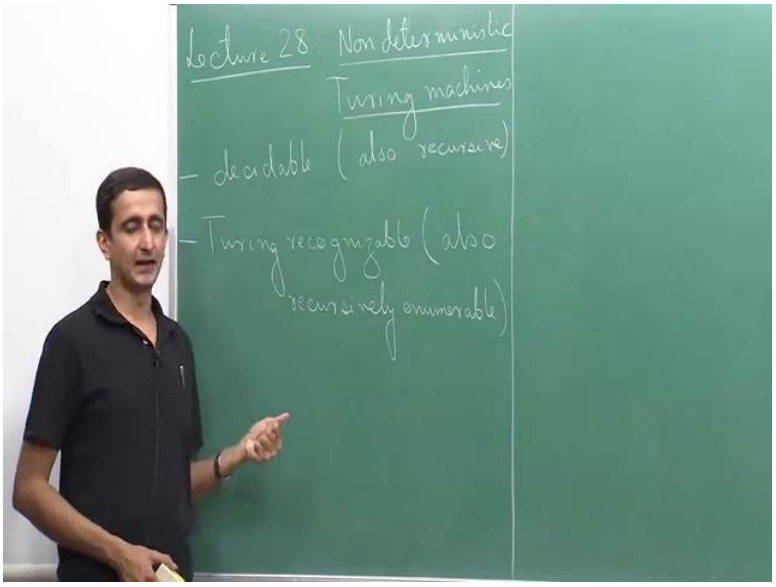
Theory of Computation
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 28
Non deterministic Turing Machine Edit Lesson

Welcome to the 28th lecture of this course. Till so far we have studied several models of computation and the last model of computation that we were looking at was that of Turing machines. So, Turing machines are a model of computation, where we have a finite control and together with the finite control, we have a tape which is unbounded and any cell of the tape can be accessed at any point during the computation.

(Refer Slide Time: 00:48)

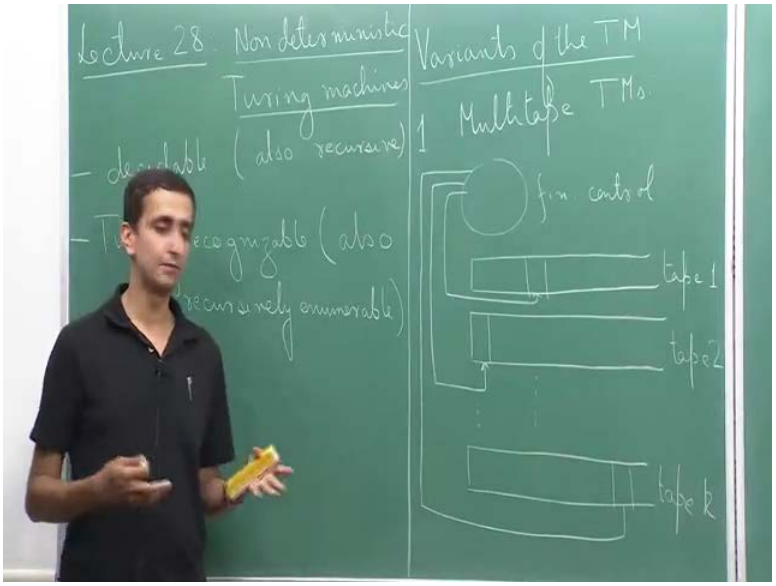


So, just to recall let me state some of the classes related to Turing machine. We have the class of all decidable languages. These are as I said also known as recursive languages. So, decidable languages are those languages for which there is a halting Turing machine that accepts that language. We saw the definition last time, and a super-class of decidable languages is what is known as Turing recognizable languages. This is also known as recursively enumerable languages. For Turing recognizable languages, we say that are

languages Turing recognizable language, if there is a Turing machine that accepts that language.

In other words, for strings which are in the language, a Turing machine halts and accepts; for strings that are outside the language, the Turing machine can either halt or reject or it can go into an infinite loop. So, that is the distinction between the two. Break in video that decidable languages are a subset of Turing recognizable languages. So, what we are going to see today is, we are going to look at other variants of Turing machine, ok.

(Refer Slide Time: 02:52)

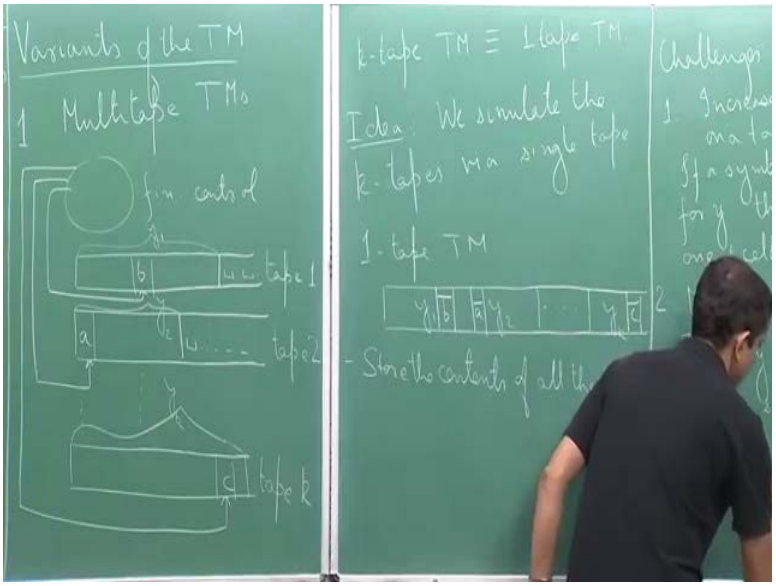


The first variant that we are going to study is, what are known as multi-tape Turing machines. When we defined Turing machine, we said that, it is a computational model which has a finite state and 1 tape. But, what if instead of 1 tape, I have 2 tapes or 3 tapes or some k number of tapes? So, how is it any extra powerful or is it the same as that of Turing machine? We have finite control. So, this is the finite control; and there are several tapes. So, this is tape 1 and then let us says we have a second tape and so on. Let us say, there are k tapes.

And because we have k many tapes, there are k many tape heads. So, every tape has its own input head, yes. So, for this tape, may be the tape head is pointing to this cell; for

the next tape, maybe it is pointing somewhere here; may be the first cell; and for the k th tape, maybe it is pointing some cell here. There are other tape heads also. So, each tape has a different tape head and each tape head is capable of pointing to any cell of that respective tape.

(Refer Slide Time: 05:08)



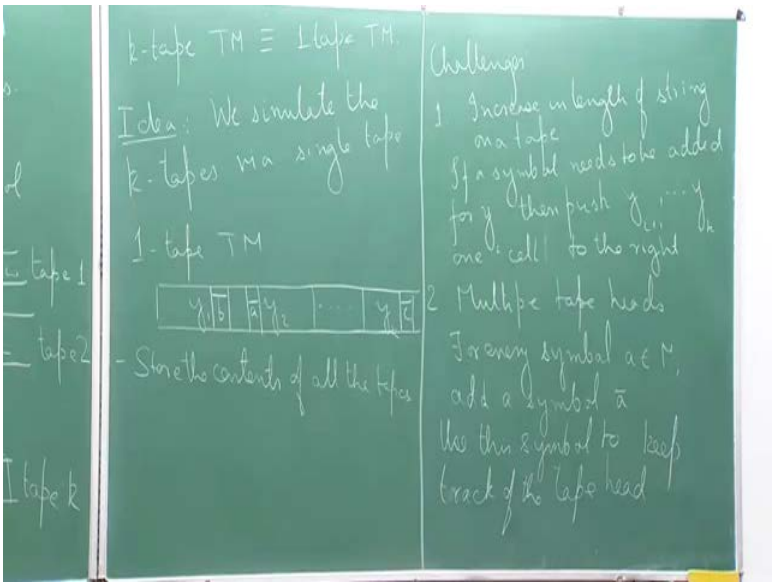
So, what we claim is that a k-tape Turing machine is equivalent to a 1 tape Turing machine. So, what is the reason? The idea is, we simulate the k tapes via a single tape. And how do we do that? Suppose, tape 1 contains the string, let us say some w 1. This is the non-blank portion of tape 1. So, beyond this it is all blanks; and maybe tape 2 has non-blank portion from here to here and maybe the k th tape has some non-blank portion up to here. Let us give some names to these strings. At any step let us say that, this string is some maybe y 1, this string is y 2 and this is some y k. So, each of the tapes whatever string is there on that tape, I will call it y i for the i th tape.

Now, what I do is that, here is the idea of my 1 tape. Here is the 1 tape Turing machine. So, what the 1 tape Turing machine will store is, it will store a concatenation of the y is. It will store the contents of all the tapes. Basically, it will have y 1, then, it will have y 2 and then, it will have y 3 and so on. Finally, it will have y k basically, whatever are the contents of each of these tapes, I will just concatenate them and store in my 1 tape Turing

machine. So, this is essentially the idea. Now, the idea is to implement this. How do I implement this? So, suppose, if there are any operations that is taking place in tape 1, I will perform that operation over here; if there is any operation taking place in tape k, I will essentially perform that in this portion and so on.

So, everything is fine but if you actually think about it a little more, if you try to implement this there, we run into 2 problems. The first problem is, what if the string that is there in tape 1 increases in length? So far, I have a string which is, let us say occupying cells from here. What if I add 1 more symbol? It can very well happen that, it replaces the blank symbol with a new symbol. So, what if we, that string increase in length? Or for example tape 2, whatever string is there in tape 2, increases in length? Because, if you look here there is no more place for any increase.

(Refer Slide Time: 09:11)



So, let me call this as challenges number one is increase in length of string on a tape. This is problem one. The second problem is now, for every tape we had a different tape head, but in a single tape Turing machine I cannot have k different heads. This only contains 1 head. If that head points to the cell of tape 1, I have lost the information about where the head of tape 2 was pointing or tape 3 or tape k; because, this is only 1 tape

head. So, multiple tapes head. These are 2 challenges. How do we take care of the challenges? So, let us look at the first challenge.

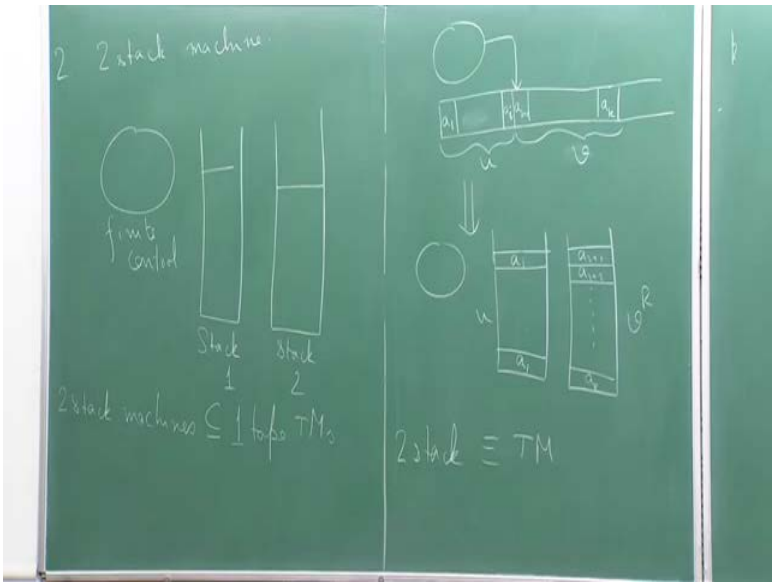
So, to take care of the first challenge, what we do is that, if there is an increase in the length of y 1, I just push all the contents from y 2 onwards, up to y k, by 1 cell. If, let us say, I add an extra symbol over here, I push y 2, y k, every y 2, y 3, all the way up to y k, one step to the right to create an extra cell for y 1. If a symbol needs to be added for y i, then, push y i plus 1 all the way up to y k, 1 cell to the right; you push everything 1 cell to the right. So, that takes care of the first challenge.

What about the second challenge? How do we handle multiple tape heads? So, what we do is that, for every symbol a belonging to gamma add a symbol, let us call it a bar. So what we do is that, to represent what is the position of the tape head in a particular tape, we will replace the symbol of that cell with the corresponding bar symbol. For example, let us say that here, I had the tape head pointing to this cell and this contained. Let us say, a symbol b. In this case, let us say, this contains a symbol may be a and here. Let us say, it contains the symbol, let us say c.

In the case of y 1, I have my b over here; in the case of y 2, I have a over here; and in the case of y k, let us say, I have c over here. Of course, there are other symbols to the left and right so what we do is that, here we replace b with b bar, we replace a with a bar and we replace c with c bar. So, this will tell us, what the position of the tape head is. Now, suppose, if the tape head in tape 1 moves from this cell to the cell left of it, we will change this back to b and replace the symbol that is to the left of this cell with its corresponding bar symbol. So, use this symbol to keep track of the tape head. So that is all.

This settles the complexity of multi-tape Turing machines. So, multi-tape Turing machines are equivalent in power to 1 tape Turing machines, in terms of the class of languages that they accept.

(Refer Slide Time: 14:56)



Now, let us look at the following. So we know that, push down automaton is basically a finite control with 1 stack; what if we have a 2 stack machine? Basically, we have a finite control and you can assume, this is deterministic, it does not matter; and together with it, we have 2 stacks. So, this is stack 1 and stack 2. What is the complexity of this model of computation? First of all, observe that, if I have 2 stacks, I can, of course, think of a stack as a tape. It is just that, it is a special type of tape, where addition and deletion happens only at one end it cannot happen at the middle; I can think of it as a tape it is, of course a subclass of 2 tape Turing machine. And because we showed that k tape is no more powerful than 1 tape, therefore 2 stack machines are a subclass of 1 tape Turing machines.

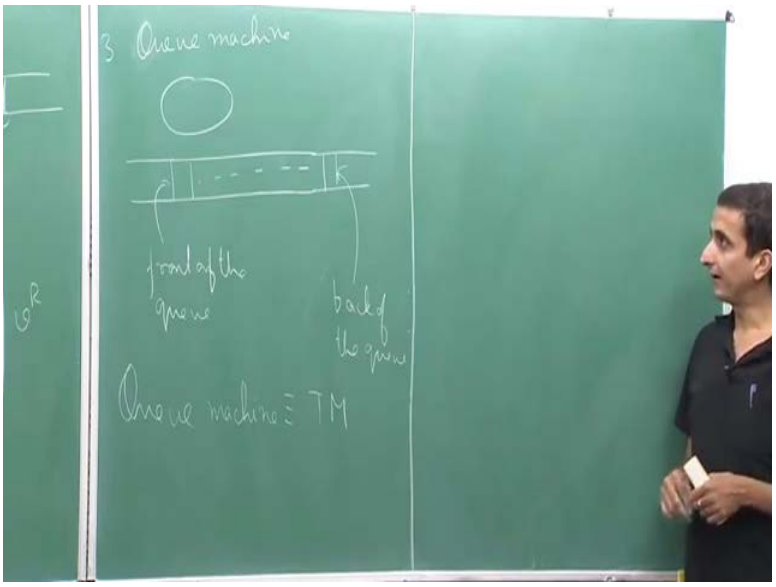
But, what about other direction? Whatever I can do with 1 tape, can I do with two stacks? So, the answer is, yes. This shows that they are equivalent. So, to look at the other direction considers the following. Let us say, I have a 1 tape Turing machine. I have my control finite control and let us say, this is the state of the tape and maybe this is the cell to which the input points to. Let us say that this portion of my string is u, from here to here and this portion is v, this portion is v. Suppose, I have a 1 tape Turing machine like this, what I will do is that, I will simulate this using a 2 stack, 2 stack

machine, in the following manner. Suppose, this is u, I store u on the stack such that this symbol.

So, this is the symbol; let us call it some a i. So, a i is at the top of the stack. Let us say, this is a 1, this is a i and may be this is a i plus 1 up to a k. I stored u over here, a i and a 1 in stack 1 and in the second stack, I store v, such that, a i plus 1 is stored at the top; then, I have a i plus 2 and so on, till a k. This is basically u and this is basically v, reverse. Now, what I have is that basically, whatever is there to the left of the tape, left of the head is in one stack and whatever is to the right of the input head is in the other stack. Suppose, if I want to replace this symbol with some other symbol and move left or right, I will just do accordingly over here. If I want to move left, then, basically, I just pick up, I, first of all, I replace a i plus 1 with whatever symbol that is necessary; I pop out a i from stack 1 and push it on to stack 2.

Similarly, if I want to replace a i 1, a i plus 1 with something and move right, what I do is that, I pop out a i plus 1 and push whatever I want to replace a i plus 1 with, on the first stack. Basically, I always maintain the invariant that the cell whatever is the content of the cell to which the tape head points to, is the top most element of the second stack; that is the invariant that I always maintain. And this will allow me to simulate a Turing machine using 2 stacks. This shows that, 2 stack machine is equivalent to a Turing machine.

(Refer Slide Time: 20:37)



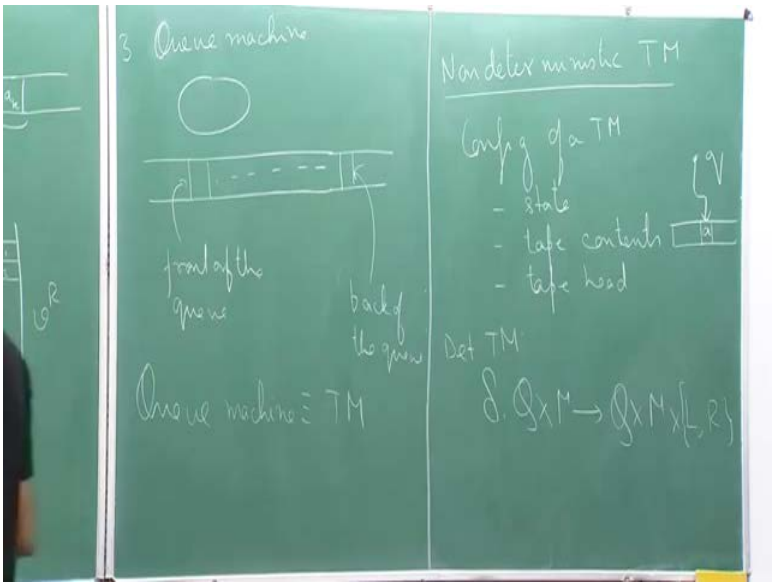
Let us look at one more model. This is the Queue model. I have a finite control and together with the finite control, I have a queue instead of a. This is the front of the queue and this is the back of the queue. And I can add an element only to the back of the queue that is known as the enqueue operation. I am not going to go through this. So, what is known is that, once again that queue machine is equivalent to a Turing machine and the idea is that, see what you can easily show is that, a queue machine is actually or one can simulate a queue using 2 stacks.

Basically, whenever I have a queue, I can simulate it using the help of 2 stacks. Suppose, if I want to. Suppose, the back of the queue is where I push, if I store the queue on 1 stack, if I want to remove something from the queue, which is at the variant, what I do is that, I pop out everything from the queue and use the second stack to store it. I pop out everything and store it over here, in reverse order remove the last element and then, push everything back to the first, first stack.

It is like, if you have a pile of dishes and if you want to remove the first, that of the bottom-most dish, what you do is that, you keep on removing the dish one after the other and create a second pile of dishes, until only the last dish remains. Now, you remove the last dish and again, replace the pile back. So, you again replace it, back where it was. So,

that is essentially the idea. Now, let us look at another very important model in terms of Turing machine, that is the subject of today's lecture, that is Non-deterministic Turing machine.

(Refer Slide Time: 23:37)



So, think of a configuration of a Turing machine. So, what do have in a configuration of a Turing machine? If you look at the configuration of a Turing machine, It has 3 things. It has a current state; it has the current tape contents and it has the position of the tape head. We have a state q and in the tape basically I am pointing to a particular cell. If you have a deterministic Turing machine, the transition of a deterministic Turing machine is as follows.

So, the transition was, you, from a state comma symbol pair. If you are at a particular state and if you are pointing to some symbol, let us give it a name a, then, it goes to a unique tuple. It replaces q with, you go from q to a unique state; you replace with a unique symbol and you go either left or right. So, this is a deterministic. In a Non-deterministic Turing machine. So, let me write it here.

(Refer Slide Time: 25:22)

In a Non-deterministic Turing machine, one can have multiple transitions from a given configuration. Basically, the transition function will be of the following form. So, here is the transition function. It will be a function which, from a Q comma symbol pair… So, from a state comma symbol pair, it goes to the power set of Q cross gamma cross L comma R. So, there are multiple transitions. For example, this is a configuration. Let us, we will actually talk more about configurations in our next lecture, but it may give you an idea. So, let us say, this is a configuration.

From here, it can go to several other configurations. In this case, it is going to 3 different configurations. So, let us call it C 2, may be C 3 and C 4. So, so, this is what is known as a non-deterministic Turing machine. Basically, in every step of the computation, the machine can non-deterministically decide to do zero or more different transitions. It can do several things. Let us look at an example and we will stop after we see the example. Here is a simple problem. We are given a Turing machine, we are given a, we want to solve the problem; so, the following language which consists of strings of the form zero to the power t, where t is a composite number. So, how do we solve this? Of course, we can solve this deterministically also. I just check, whether a number is prime, but let us try to do this in a non-deterministic manner.

First, what we do is that, we first non-deterministically write down 2 things. We write down, write down a, let us say, n 1 number of zeroes and n 2 numbers of 1s. Now, check if n 1 time n 2 is equal to t; if yes then accept else reject. Observe that, a number is composite if it has 2 factors. So, when I say n 1 and n 2, let us assume that, both n 1 and n 2 are greater than or equal to 2 and strictly less than t, because, they should not, 1 and t should not be factors. So, we have these 2 numbers. If I have a number which is a product of 2 such numbers, then, we say it is composite; hence, we accept it. and this check can be easily done using a Turing machine.

We use non-determinism to guess these 2. So, write down. So, n 1 and n 2, I just use non-determinism to find out how many zeros and how many ones should I write, ok. So, here, I will just say, non-deterministically. And as it is the case with all non-deterministic algorithms, if I have a non-deterministic algorithm, I will accept an input, if there is some accepting path and I will reject the input if all paths lead to reject. If a number is composite, it will have some 2 factors. Hence, there will be some accepting path but, if the number is not composite that if it is prime it does not, it should not have any factors, any proper factors; hence, no matter what n 1 and n 2 I guess, it will always lead to reject. Hence, I will not have any accepting computation. So, this is an example. I will stop here.

Thank you.