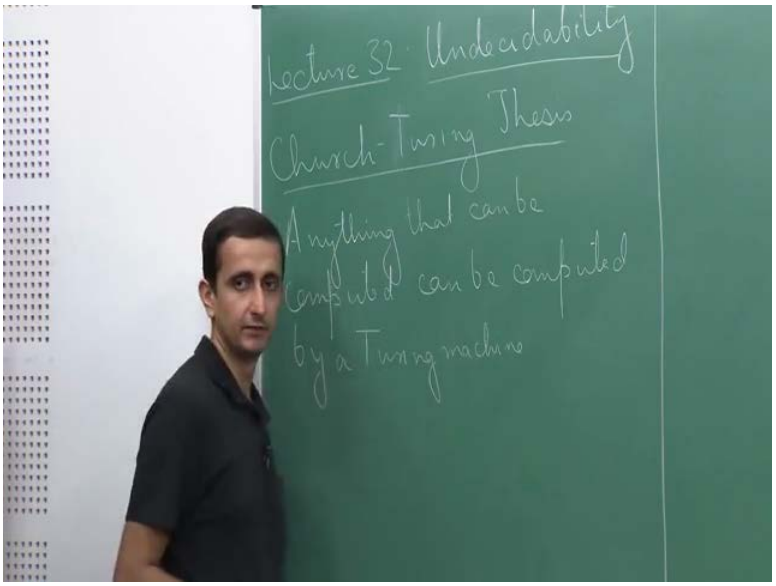
**Theory of Computation**
**Prof. Raghunath Tewari**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Kanpur**

**Lecture – 32**
**Undecidability**

Welcome to the 32 lecture of this course. Today, we are going to talk about another very important topic in theory of computation namely that of undecidability.

The question that we are going to address today and in the next couple of lectures is, are there problems that cannot be computed using a Turing machine. We will show that there are problems that cannot be computed using a Turing machine. So, they are called undecidable problems. And we will show how to show that certain problems are undecidable and so on.

(Refer Slide Time: 00:58)



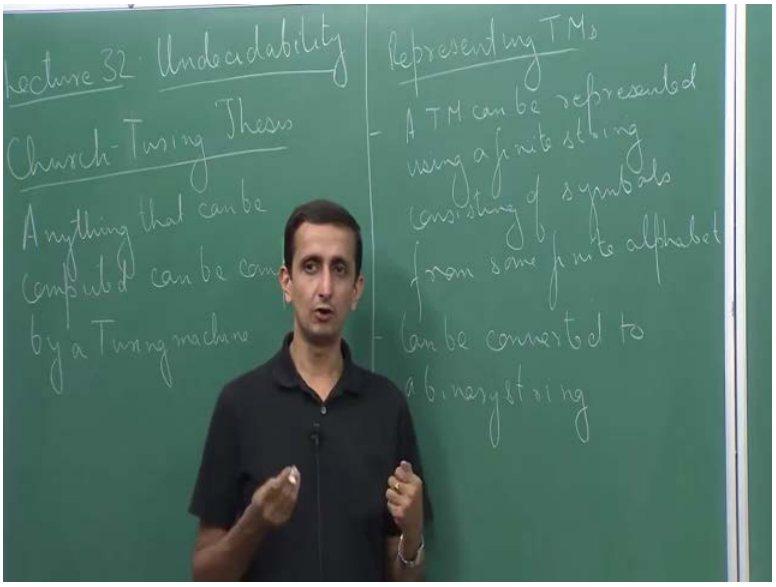

But before I go into the topic of undecidability, let me mention hypothesis that was laid down by Alonzo Church and Alan Turing. So, it is called the Church Turing thesis. So, essentially what they said is that anything that can be computed by a Turing machine. So, in other words, what this hypothesis says is that Turing machines are a Turing machines can be thought of as a universal model of competition. So, any problem any computational problem, which can be computed by any computational device for that

problem there also exist a Turing machine that can decide that problem, if we represent it suitably in a language format.

Basically, therefore then in the context of undecidability, what this means is that suppose if I show that a problem is undecidable, in other words, if I show that a problem cannot be decided by a Turing machine, it essentially means that it cannot be computed by any computing model or any computing device that is available at hand, if we believe the Church-Turing thesis.

So, this is actually to show some problems are undecidable is quite a big question I would say. Because if you imagine I mean with so much computing power that is available to us today to say that there are languages, which cannot be computed is something that is not immediately obvious. So, before I show the first undecidable problem, let me talk about equivalence of Turing machines and so representing Turing machines by integers.

(Refer Slide Time: 03:45)

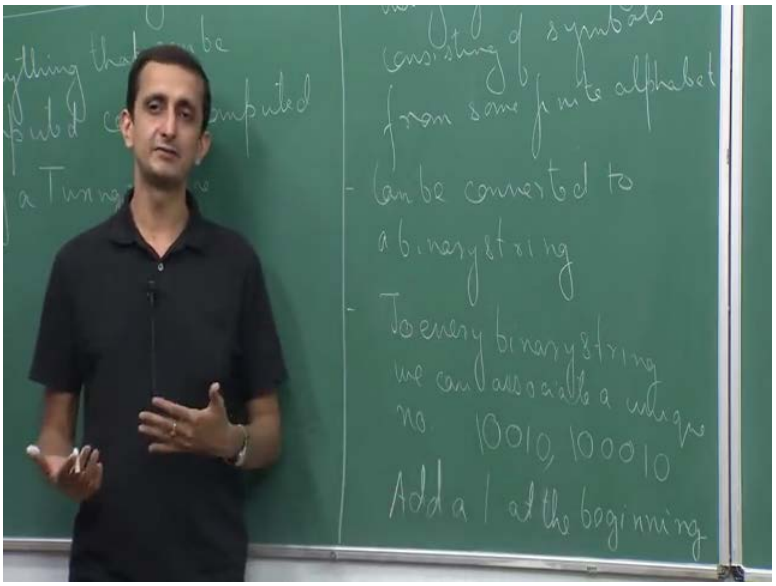



In our last lecture, I said that any Turing machine or for that matter DFA PDA you can represent it using a finite string. So, a Turing machine also can be represented using a finite string consisting of symbols from some finite alphabet. For example, if I just represent the description of a Turing machine that is a transition function with the states accepts state reject state; it is basically a finite string over some finite sequence of alphabet. Now I can convert this, this to a binary string. So, what I mean by this is that I

can take the string which represents a Turing machine, and I can convert it into a string over zero, ones.

The way to do that is suppose I have a string over an alphabet, which has size k. So, there are k symbols in the alphabet k can be any constant. So, therefore, to represent, I use log k bits I use log k bits over 0 1 to represent each symbol of the finite alphabet. For example, if there are h 10 symbols in that alphabet, I will use log 10 I will use the ceiling of log 10, which is basically 5. So, I will use four symbols each symbol of the alphabet will be represented by a binary string of length 4. Now, therefore, I take a string over the first alphabet, and convert it into a string over 0 and 1 by replacing every symbol with its corresponding binary string.

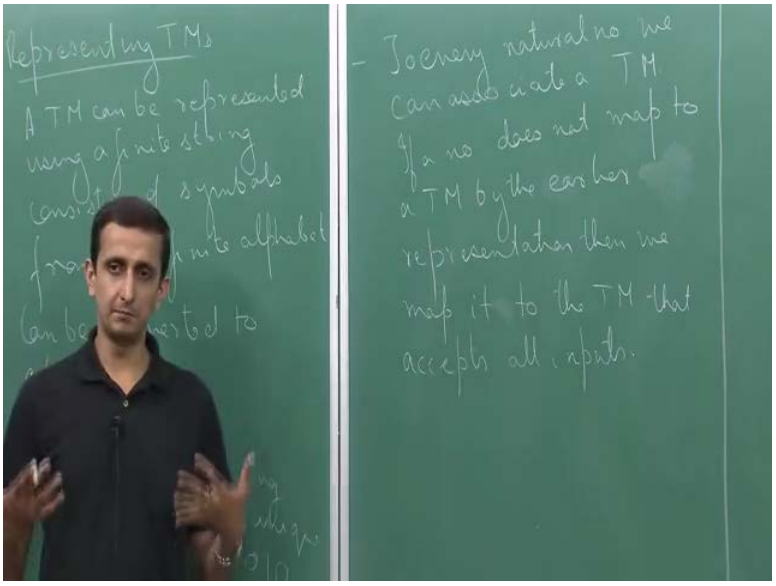
(Refer Slide Time: 06:32)




Now, to every binary string, we can associate a unique number. So, every binary string, so suppose if the string does not begin with a 0, then of course, every binary string can be converted into its corresponding decimal string and that is a natural number. But suppose if I have preceding 0s, so for example, if I have a string let say 0 0 1 and another string 0 0 0 1, so both these strings when converted to decimal corresponds to the number or let us take a little bit more. Let us have a string 0 0 1 0, and I have another string triple 0 1 0. So, when I convert both these strings into a decimal it corresponds to the number 2.

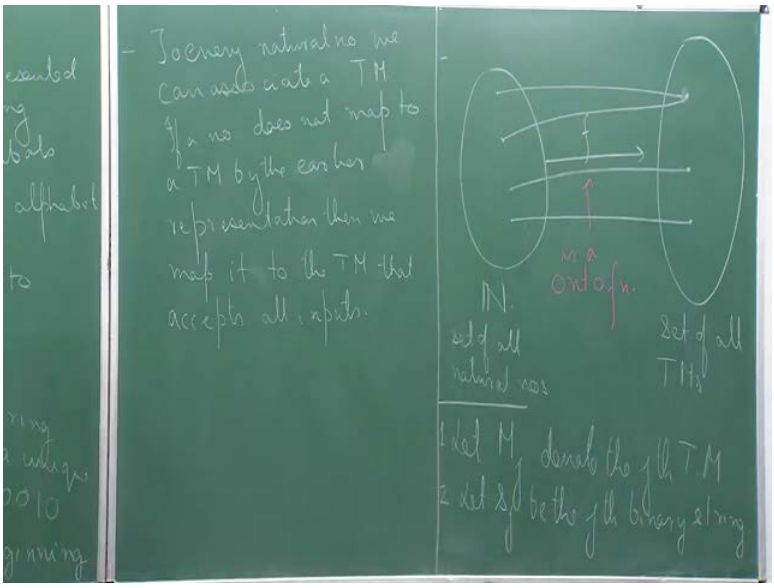
So, to avoid this kind of a situation where two binary strings can map to the same natural number, what I do is that I add a 1 at the beginning. Now, if I for example, add a 1 to the beginning of every string, so I add a 1 here, and I add a 1 here. So, this string now corresponds to some number. So, in this case, it corresponds to the number 18 and I think so, so this will correspond to 18 and this will correspond to 34. So, they correspond to two different natural numbers. So, basically if I add as 1 to the beginning of every binary string, then every binary string corresponds to a unique natural number.

(Refer Slide Time: 09:09)




Now, to every natural number, we can associate a Turing machine. So, by this mapping, here for every Turing machine I have a binary string, which gives us a natural number. So, there a certain natural numbers which correspond to a Turing machine, but there might also be some natural numbers which do not correspond to a fix Turing machine. So, what I do in that case is that, so if a number does not map to a Turing machine by the earlier representation then we map it. So, then we map it to some arbitrary, but fixed Turing machine, let say a Turing machine that accepts all its input, map it to the Turing machine that accepts all in puts.

(Refer Slide Time: 11:13)




Basically what is necessary is to have a function of the following form. So, suppose if I look at the set of all natural numbers, so this is the set of all natural numbers on the left hand side. And on the right hand side, let us say I have the set of all possible Turing machines. So, then I should have a function let us call it f from the set of all natural numbers to the set of all Turing machines such that this f is a onto function.

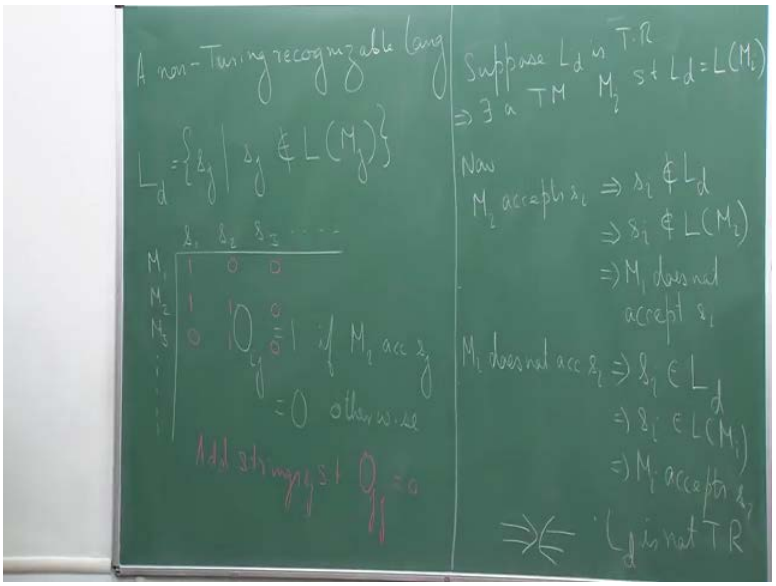
So, in other words, every Turing machine must have a unique well it need not have a unique, but every Turing machine must have some pre image. So, then for every natural number gets mapped to a unique Turing machine, and there is no Turing machine that is left out, there is no Turing machine that does not have a natural number as its pre image. For example, I can map this to a number here a Turing machine there, and I can map another number to the same machine, I can have a mapping here, something here and so, on. What is important is that this is onto map.

Therefore, because I have an on to map, what I can do is that for the jth natural number, we give the following notation. Let M j denote the jth Turing machine, so because for every Turing machine I have a number associated with it. So, therefore, I can basically number all the Turing machines, may be that some of those Turing machines get repeated in the enumeration, may be Turing machine number 1 and Turing machine number five are the same Turing machine that is ok, but the point is that for every j there is a Turing machine M j. So, this is very important. So, this is number 1

And number 2 is I can obviously enumerate binary strings. Let S j be the jth binary string. So, once again I can look at every binary string, I had a one to the beginning of it, just to avoid preceding zeros, and then I can just enumerate them. Let us say that s 0 would be the string 0 or may be the string epsilon, s 1 will be the string 0, s 2 will be the string 1 then s 3 will be the string 0 1 and then 1 0 and then 1 1 and so on.

So, I can basically enumerate all strings. Now, that we have these two things. So, we have an enumeration of Turing machines and we have an enumeration of binary strings. Now, we are in a position to talk about our first undecidable language. In fact, we will prove something more here. We will give a language that is actually not Turing recognizable.

(Refer Slide Time: 15:29)




So, non-Turing recognizable language; so let us define this. So, I define the language L d as the set of all binary strings s j such that s j does not belong to the language of M j. So, this argument is what is famously known as the diagonalization argument. So, this was first introduced by George Cantor and later on it has been used by several mathematicians to prove various results. And this is I mean this is the approach that Alan Turing took in his famous paper where he showed that there are problems that cannot be computed by a Turing machine, a diagonalization approach.

To understand this language let me draw a table. So, we have this semi infinite matrix. So, it is finite on the left and the top side; and on the right and the bottom, it is infinite.

So, on the rows, I will put Turing machines, so first I will have M 1 M 2 M 3 and so on. And on the columns, I will have the strings s 1 s 2 s 3 and so on. And the way I fill it up I fill up this matrix using 0 and 1. So, let us give this matrix a name. So, I will call this matrix let say l. Let us give no may be not l. So, I will call this matrix O. So, o of i j is equal to 1, if M i accept s j; otherwise, it is equal to 0. So, essentially I look at this matrix. So, I look at the first Turing machine M 1, I check whether M 1 accepts s 1, if yes I put a 1; otherwise I put a 0, then I go to s 2, s 3 and so on, so I fill this up then I fill the second row and so on.

The way this language L d is constructed is basically, so suppose if I have a 1 over here. Let say I have the following entries 1 0 0 and then I have may be 0 1 0, and then 0 1 0 in this matrix, what I will do is that I will replace so I will basically swap the entries of the diagonal. So, I look at the diagonal of this matrix, if there is a 1, I flip it with 0; and if there is 0, I flip it with 1.

So, in other words, if I have a 1 in the diagonal it means that M 1 accepts s 1 in which case I will not add s 1. So, I will only add those elements of the diagonal, which have a 0. So, in this case, what we have is that M 3 does not accept s 3. So, basically s 3 does not belong to the language of M 3 in which case we add s 3. So, this is what we do. So, we go through the diagonal whenever we have a 0, I look at the corresponding string and I add it on to the language L d So, add string s j such that O jj is a 0.

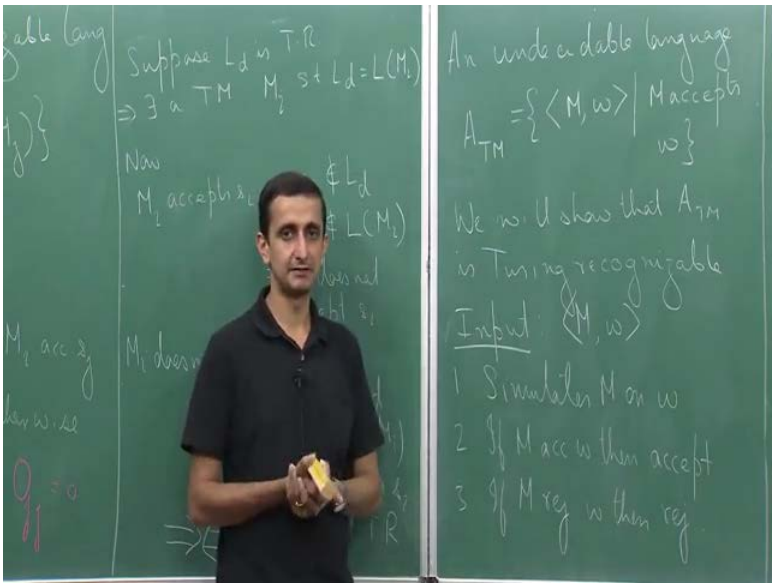
Now that we have this we can argue the following. So, suppose L d is Turing recognizable, so for the sake of contradiction, let us say that L d is Turing recognizable. So, then there exists a Turing machine let us call it M and because I have enumerated all Turing machine, this Turing machine must be ith Turing machine for some i. So, there exists a Turing machine M i such that L d is equal to the language of M i. If this is Turing recognizable it must be accepted by some Turing machine that Turing machine is somewhere in the row let us say it is our ith row, so I say that M i is the Turing machine.

Now consider the following case. So, if M i accepts, so what have. So, what does M I do to the string s i. So, if M i accepts s i, what do we have. So, if I look at definition, so if M i accepts the string s i, it means I have a 1 on the table at that point which means that by the definition of the Turing machine, it should not belong to the language.

So, this implies that s i does not belong to the language L d because it only accepts those where it the corresponding machine does not accept the string. This implies that so now, what is the machine M i. So, M I is the machine for the language L d. So, if s i does not belong to L d, then s i also does not belong to the language of M i which implies that M i does not accept s i. So, we have that if M i accepts s i, it implies that M i does not accept s i.

Similarly if M i does not accept s i, again if I go to the table I have a 0 over there, so I flip it to get a 1. So, if M i does not accept s i, by the definitions I belongs to the language L d which means that s i is accepted by the machine M i. So, therefore, we have a contradiction that if M i accepts s i if and only if M i does not accept s i. So, this gives us a contradiction. So, therefore, L d is not Turing recognizable. So, therefore, we have exhibited a language which is not Turing recognizable.

(Refer Slide Time: 25:19)




So, next we are going to look at an undecidable problem. So, in our next lecture, we will prove that this language is undecidable. The language that we are going to consider is we define it as follows. So, it is called ATM; and it consists of encodings of machine M and a string w such that M accepts w. So, this is a language which has an encoding of a Turing machine M and a string w. So, they are given to it together. And it consists of all those pairs such that the machine M accepts w. If M does not accept w, it is not in the language A TM.

In our next lecture, we will prove that this is undecidable, but what we are going to show today is that his language A T M is actually Turing recognizable. So, we will show that ATM is Turing recognizable. So, what would be an algorithm to show? So, remember that if I want to show that something is Turing recognizable then it is enough to show that for instances that are in the language, I have a Turing machine which accepts A, and instances that are out of the language we do not care. So, here is an algorithm. So, given as input machine M and a string w, what algorithm does or what a Turing machine does is, it simulates M on w. If M accepts w, then accept; and if M rejects w then just reject. It is a very simple algorithm.

So, observe that when I simulate M on w, there are three possibilities; either M can accept w, or it can reject w, or it can loop forever on w. So, if M accepts w, then it accepts M comma w. So, it is an instance which is in the language A TM. If M does not accept w, if M rejects w, then anyway I am rejecting, so it is not here. And if M loops forever on w, then anyway the first step never stops.

The first step goes on forever; in which case also I do not accept, hence it is not in the language A TM as required. So, what we will show in the next lecture is that A TM is not only it is certainly it is Turing recognizable fine, but it is not decidable. So, I will stop here today.

Thank you.