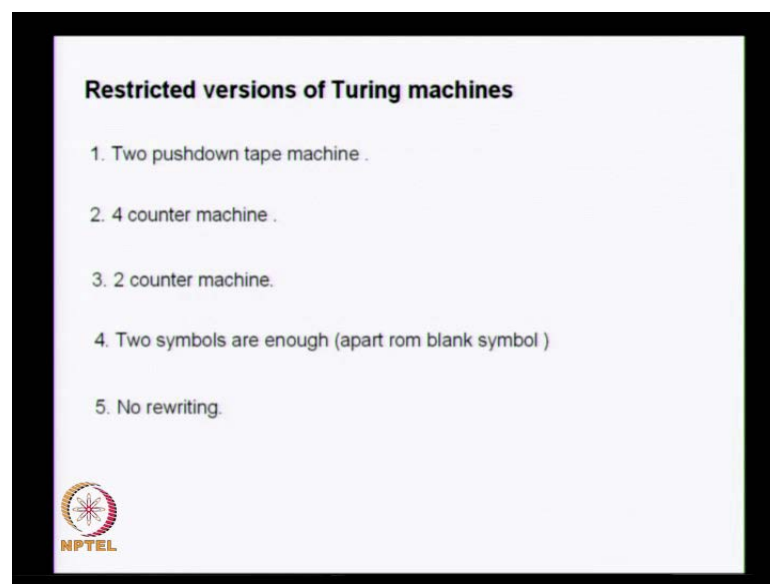


**Theory of Computation**  
**Prof. Kamala Krithivasan**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture No. # 31**

**Recursive Sets, Recursively Innumerable Sets, Encoding of TM, Halting Problem**

(Refer Slide Time: 00:25)

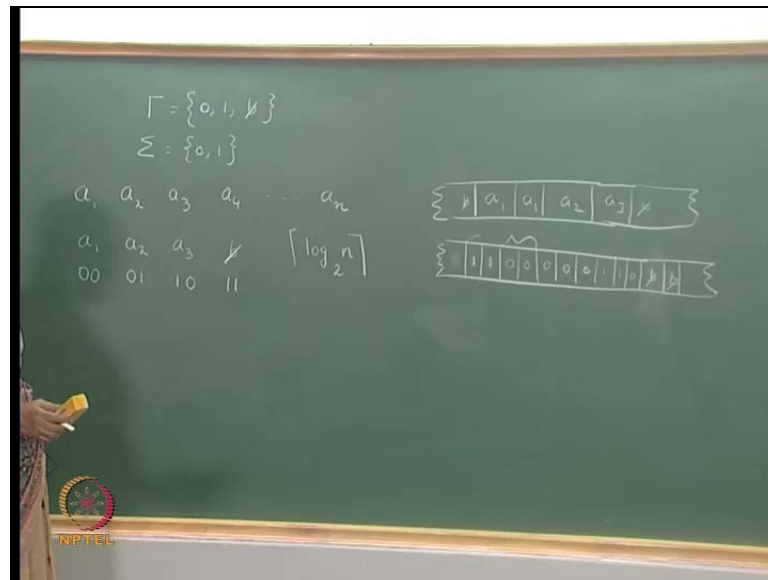


In the last lecture, we considered some restricted versions of Turing machines. Earlier we considered some generalized versions of Turing machines. Then in the last lecture, we considered 3 restricted versions of Turing machines. One of them was a two pushdown tape machine. Any Turing machine can be simulated by Turing machine with two pushdown tapes; that is 2 tapes which behave as pushdown tapes. Then we showed that any Turing machine can be simulated by a 4 counter machine.

A counter machine is a one, where the tapes behave like counters; that is they can store only a number there is only one non-blank symbol and the other symbols are blanked. And the distance from the non-blank symbol to the position of the head is calculated, and it denotes an integer. We showed that one pushdown tape can be simulated by 2 counters and hence 2 pushdown tapes can be simulated by 4 counters; then we showed that even 4 counters are not necessary, you can simulate with 2

counters. The 4 counters store 4 numbers  $i, j, k, l$  and we can make use of one integer to represent all  $i, j, k, l$  by using the concept of Gödel numbering, and so we can simulate 4 counter machine with a 2 counter machine.

(Refer Slide Time: 02:12)



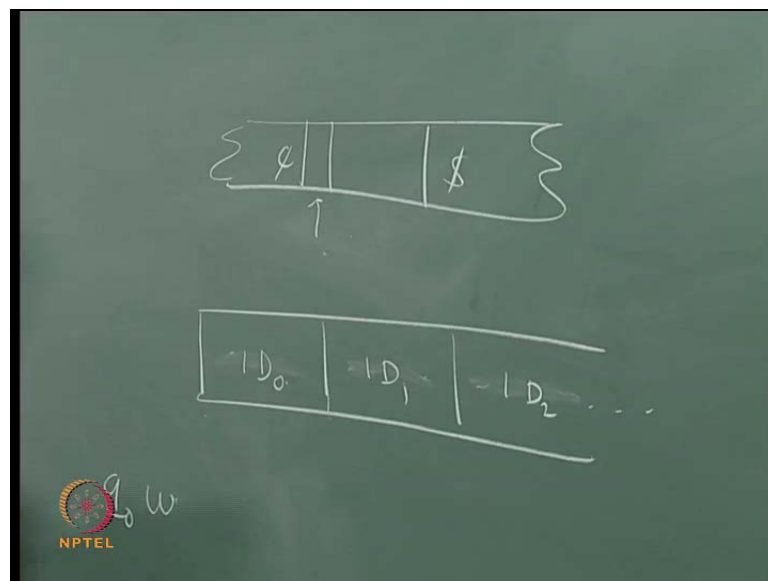
Having said that we have some more restricted versions, we find that two symbols are enough apart from the blank symbol for any turing machine. That is we can take the tape alphabet to be, just two symbols and a blank symbol input alphabet may consist of 0 and 1 only. So, input alphabet you can take to be just 0 and 1; how can we do this? Suppose a turing machine has say  $n$  symbols say  $a_1, a_2, a_3, a_4, \dots, a_n$ . We can encode the  $n$  symbols using just binary alphabet that is 0 and 1; for example, if you have just  $a_1, a_2, a_3$  and then say blank, instead of  $a_1$  you can use 00; instead of  $a_2$  you can use 01; instead of  $a_3$  - 10 and blank - 11 some sort of an encoding.

Probably, we would like to use 00 for blank and 11 for  $a_1$  anyway you can encode the symbols in binary and to encode  $n$  symbols each symbol has to use  $\log n$  to the base 2. See how many bits will be required to encode each symbol. For example, suppose I have the alphabet as  $a_1, a_2, a_3$  and I have a tape like this, where I have  $a_1, a_1, a_2, a_3$  followed by blanks and preceded by blanks. The corresponding encoding here will be for blank I used 11 for  $a_1$  00 again for  $a_1$  00; for  $a_2$  01; for  $a_3$  10, then blank is 11, so 11 like this. You must here you have blanks, but you must realize that you should when you use

blank, you should replace it this is the input alphabet. So, if you use blank here, when you want to simulate this blank you must use 1 1 and then simulate that.

Now, when you want to read a 1 what you have to do is, you have to read these two symbols and then define the mapping. So, if the states will be increased the number of moves will be doubled here. Depending upon the number of cells used it will be multiplied by  $k$ . So, you have to read these two symbols and determine that the symbol read is a 1 and define the mapping. When you want to read this blank instead of blank, you must first think of it as 1 1 and then replace it with the corresponding symbol. So, the next symbol is a 1 again. You have to read these two symbols and then determine the mapping next it is a 2. So, you have to read these two symbols and determine the mapping. So, this way any tape alphabet  $a_1$  to  $a_n$  can be encoded using the binary alphabet 0 0 0 1 1 0 1 1 and so on. So, it is enough if we consider the tape alphabet to consist of two symbols 0 and 1 apart from the blank symbol.

(Refer Slide Time: 06:40)

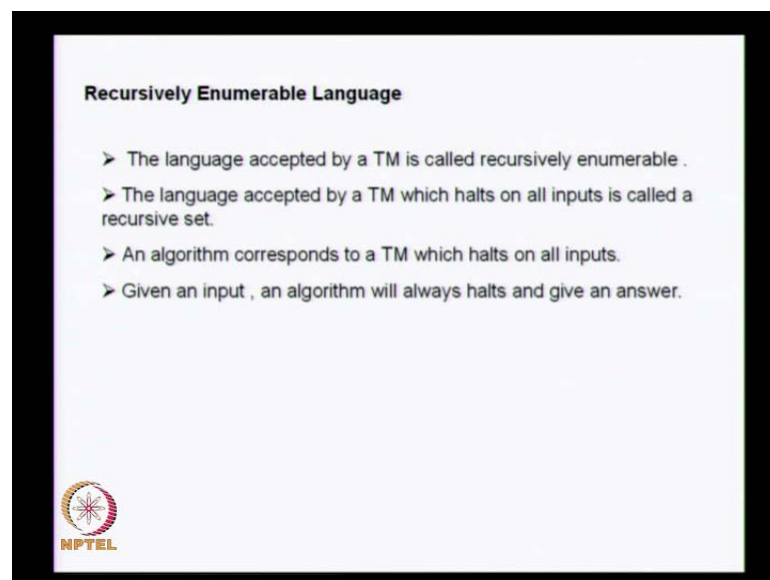


This is one restricted versions and in all our proofs we will be using this restricted version. The next instance even when you use two symbols, you can have an offline turing machine that is one which is having the input alphabet within markers and one more tape you can have where you keep on writing symbols, but you never turn to the left and rewrite something. This is done in this manner, the input is kept here and the initial id is copied on to this tape the initial id will be some  $q_0 w$ , where the tape head is

pointing here. So, initial id will be written here and then instead of changing the id; this id is left as it is and the next id will be written here in the blank portion then the next id; this is id 1 and this is id 2 and so on. The next id is written in the blank portion and you always keep on writing on this tape in the blank portion and you never turn left and rewrite something.

So, without rewriting also you can simulate a turing machine; this is much restricted version this is for interest sake only we will not be using this idea, but we will be using the idea of this that is two tape symbols are enough apart from the blank symbol.

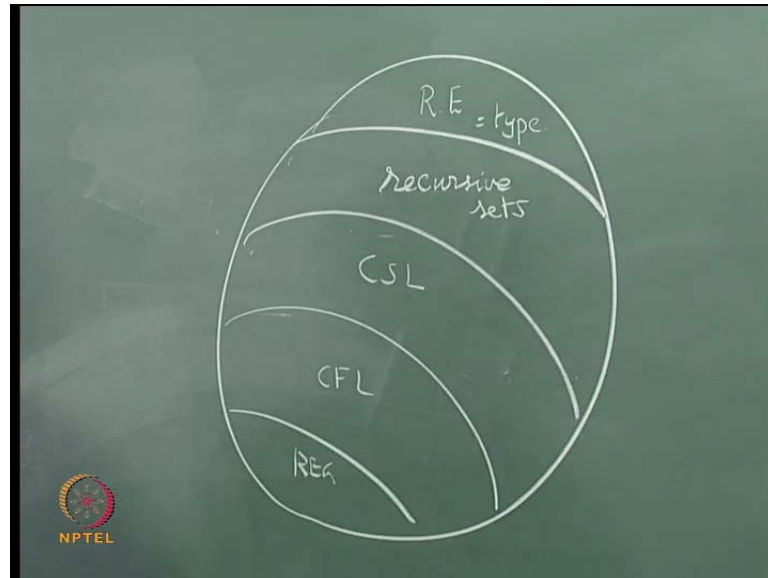
(Refer Slide Time: 08:25)



So with this, we shall go on to the next concept of what is a recursive set? And what is a recursively enumerable set? The set accepted by a turing machine is called a recursively enumerable set. We have also seen earlier that, the language generated by a turing machine can be compared with the language accepted by the turing machine. That is, if a language can be generated by a turing machine in the output tape, then there is another turing machine which will accept it. So, because you are able to generate the string, one by one the set is called a recursively enumerable set and we also know that the turing machine need not always halt on all inputs. On some inputs it may get into a loop and never halt.

Suppose a turing machine halts on all inputs, then the set accepted is called a recursive set the language accepted by a turing machine. The second one, the language accepted by a turing machine which halts on all inputs is called the recursive set.

(Refer Slide Time: 09:45)



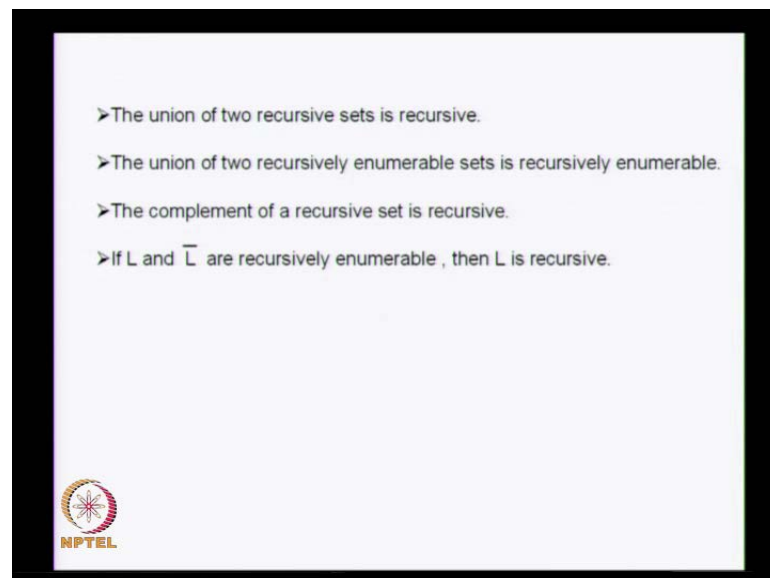
In the Chomsky hierarchy we have this diagram, the regular set is the smallest class which is included in the class of CFL. Then which is included in the class of context sensitive languages and which is included in type 0 or recursively enumerable languages.

The recursive set form a subclass of recursively enumerable sets, but it is higher than context sensitive languages. This is the class of languages accepted by turing machine and this is the class of languages accepted by turing machines which halt on all inputs. Now, when we say recursive set it corresponds to an algorithm, a procedure corresponds to a turing machine. What is a procedure? A procedure is a sequence of statements which tell you how to behave that is you define a set of moves from instance to instance, you know what is the next step that is like a computer program which may not halt. For example, keep on printing integers 1 2 3, you can write a pseudo code for that which keep, which will execute and keep on printing 1 2 3 and will never stop, but you know, what is the next step to be done. That is an example of a procedure.

An algorithm is a procedure, which always halts given a number  $n$  is it a prime or not. You can write a program for that ultimately, it will tell you, whether the given number  $n$  is a prime or not. That is an algorithm. An algorithm always halts and gives you an

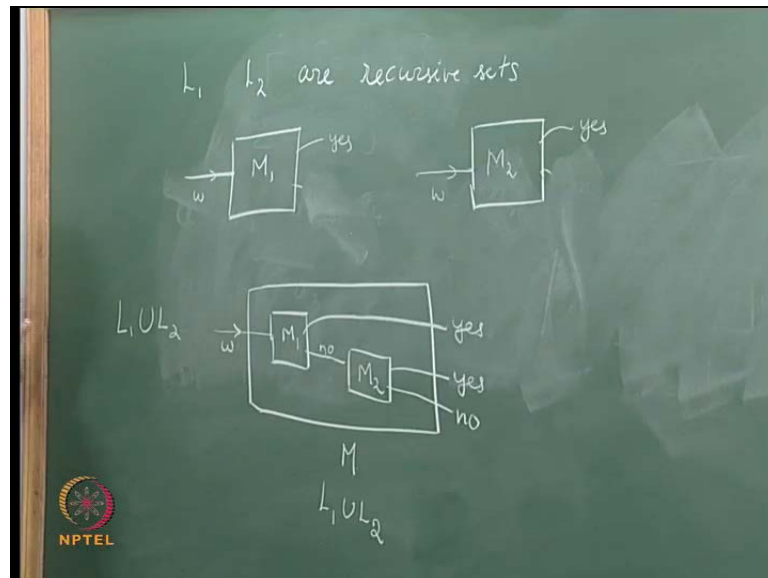
answer. It may be yes or no or some other answer, but whatever it is. An algorithm is a sequence of statements, it is a program like thing which always halts and tells you an answer and an algorithm corresponds to a turing machine which always halts. It corresponds to a recursive set. A procedure corresponds to a recursively enumerable set, you can have a turing machine which will do that. For example, you can write the moves of the turing machine which will keep on writing the integers in binary notation or in decimal notation one by one and that will keep on doing it without stopping. But the turing machine will not halt. So, an algorithm corresponds to a turing machine which halts on all inputs given an input, an algorithm will always give you an answer.

(Refer Slide Time: 12:28)



Now, let us study a few properties of recursive sets and recursively enumerable sets. So, what is that? If you have two recursive sets, the union is a recursive set. How do you prove that?

(Refer Slide Time: 12:55)

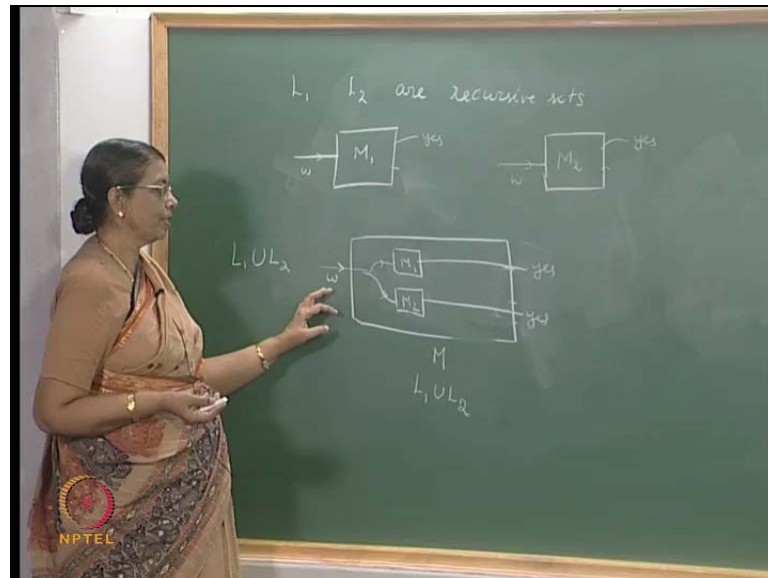


Suppose I have  $L_1$  and  $L_2$  are recursive sets. So,  $L_1$  is accepted by a Turing machine  $M_1$  for which when input  $w$  is given. It will stop and accept it or reject it. Similarly for  $M_2$ , a Turing machine which is accepting  $L_2$  given an input, it will halt and accept or halt and reject. Now how can you have a Turing machine which will accept  $L_1 \cup L_2$ ? It will have the following structure, given the input  $w$ , first  $M_1$  will act on it and if it accepts the compound machine, which is for  $L_1 \cup L_2$  will accept the input. Suppose  $M_1$  rejects the input  $w$ , then the machine  $M_2$  will be started and when  $M_2$  is started. It will come out with **the if it comes out with** the answer yes, then  $M$  accepts it. That is, if  $w$  is accepted by  $M_1$ , you will take this exit. If  $w$  is accepted by  $M_2$ , you will take this exit and if  $M_2$  rejects it. Then the compound machine will not accept, it will say no.

So, this way you can have a Turing machine, which always halts, which accepts  $L_1$  and  $L_2$ ,  $L_1 \cup L_2$ . The same procedure we cannot have for recursively enumerable set. The next result, we will study is the union of two recursively enumerable sets is recursively enumerable. Obviously, we cannot use this construction. What is the reason? The reason is when a Turing machine for a recursively enumerable set, we can have a Turing machine which will say yes, if  $w$  is accepted. But if  $w$  is not accepted, it may not halt. So, it will not say no, it may get into a loop. So, similarly for  $M_2$  also you can have a Turing machine which will say yes that is, it will halt and accept if  $w$  is accepted, but if  $w$  is not accepted  $M_2$  may halt in a non-final configuration or it may get into a loop.

Obviously, for the compound machine for  $L_1 \cup L_2$  you cannot use this construction, because this exit may not be taken at.

(Refer Slide Time: 16:28)

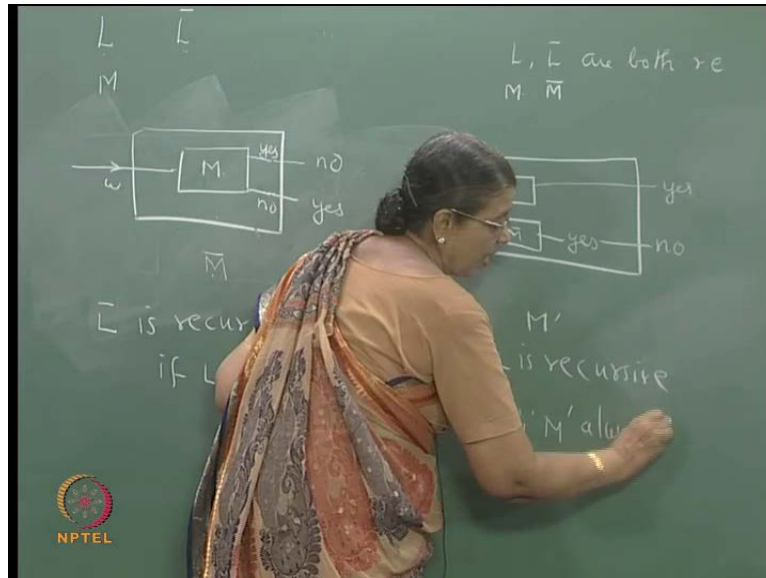


So, for accepting  $L_1$  and  $L_2$ , we will rather have a different machine. A Turing machine which has this structure  $M_1$  and  $M_2$  are simultaneously tried by with input  $w$ , you may have 2 tapes in one, you may simulate  $M_1$  in another one; you may simulate  $M_2$ . So, given the input  $w$ , it is given as input to both  $M_1$  and  $M_2$  and if this accepts, it will say yes, if this accepts also it will say yes. So, if  $w$  is accepted either by  $M_1$  or by  $M_2$  or even if it is accepted by both, you will come out of the yes exit and the machine  $M$  will accept that.

But if  $w$  is not accepted by  $M_1$  and also not accepted by  $M_2$ .  $M$  will not accept that, because it may halt in a non-final configuration or it may get into a loop does not matter. It will not accept, if  $w$  is not accepted by both  $M_1$  and  $M_2$  this way. We can have a Turing machine, which will accept the union of two recursively enumerable sets  $L_1$  and  $L_2$ .

(Refer Slide Time: 18:02)





The next result, we see that is the complement of a recursive set is recursive. That is you have  $L$  as a recursive set. Now I want to show that  $\bar{L}$  is also recursive. How can you have a Turing machine, which always halts for  $\bar{L}$ ? This will have the machine  $M$  as a sub 1, if  $L$  is accepted by  $M$ . We know that  $M$  always halts, so given  $w$ ,  $M$  will say either yes or no and if it says yes machine for the complement -  $\bar{M}$ ;  $\bar{M}$  is the machine for  $\bar{L}$ . It will say no, if  $M$  says no then  $\bar{M}$  says yes. You know that,  $M$  always halts and either says yes or no. if it says yes  $\bar{M}$  will say no, if it says no  $\bar{M}$  will say yes or in other words it is like same as finite state machine. You can interchange the final and the non final states that is what it means really.

So, you can have a Turing machine, which always halts for  $\bar{L}$ . So  $\bar{L}$  is recursive, if  $L$  is recursive that is the family of recursive sets is closed under complementation. We will see that **that** does not hold for the family of recursively enumerable. Now another result we see that if  $L$  and  $\bar{L}$  are recursively enumerable, then  $L$  is a recursive set. How can you prove that  $L$  and  $\bar{L}$  are both recursively enumerable? Then we can have a Turing machine for  $L$  which always halts, how is this machine built given an input  $w$ , Suppose this is say  $M$  dash; the machine for  $L$  is  $M$ , the machine for  $\bar{L}$  is  $\bar{M}$ , they are Turing machines, but they need not halt on all inputs.

Now, how does  $M$  dash accept a string, you give  $w$  to both  $M$  and  $\bar{M}$  as input?  $M$  dash has as subparts  $M$  and  $\bar{M}$  and  $w$  is given simultaneously as input to both  $M$  and  $\bar{M}$ . Now any string  $w$ , it has to be either in  $L$  or in  $\bar{L}$  you know that, it has to be present in one of them, but not in both. So, suppose  $w$  is in  $L$ , this will say yes. Suppose

w is in  $L$  bar, then this will say yes. In that case you can say no M dash will say no. So, you have a turing machine M dash, which always halts given w, if it is in  $L$  it will say yes. If it is not in  $L$ , but in  $L$  bar it will say no. So, if  $L$  and  $L$  bar are both recursively enumerable, then you can have a turing machine M dash accepting  $L$  and M dash always halts. What does that mean? That means that  $L$  is a recursive set. Therefore  $L$  is recursive why because M dash always halts.


(Refer Slide Time: 22:21)

**Encoding of a Turing machine**

**Example**

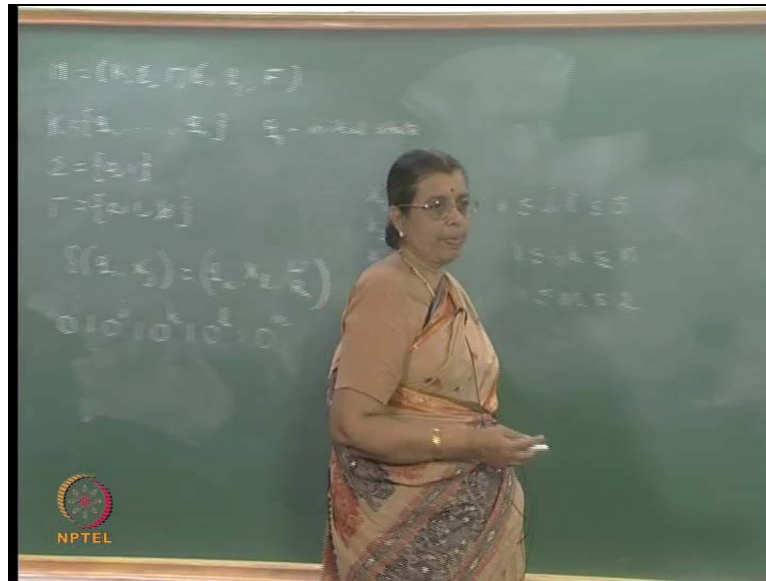
Consider the following Turing Machine with the alphabet  $\{0, 1, \beta\}$  and state set  $\{q_1, q_2, q_3, q_4, q_5\}$ .  $q_1$  is the initial state and  $q_5$  is the final state. The mappings are given by :

	0	1	$\beta$
$q_1$	$(q_2, 0, R)$	—	—
$q_2$	$(q_3, 0, R)$	—	—
$q_3$	$(q_4, 1, R)$	$(q_4, 1, L)$	$(q_4, 1, R)$
$q_4$	$(q_3, 1, L)$	$(q_5, 1, R)$	$(q_3, 1, L)$
$q_5$	—	—	—



Next we will see, how we can encode a turing machine; a turing machine can be encoded as a binary string?

(Refer Slide Time: 22:36)



We will see, how this can be done. Let  $M$  be a Turing machine -  $k, \sigma, \gamma, \delta, q_0, F$ . Let the set of states be  $q_1, q_2, \dots, q_n$  without loss of generality, there are  $n$  states and we can assume that the initial state is  $q_1$   **$q_1$  is the initial state**. You may have some **for zero** final states. Then we have already seen that without loss of generality, we can assume that, there are only two input alphabet and one blanker symbol as the tape alphabet.

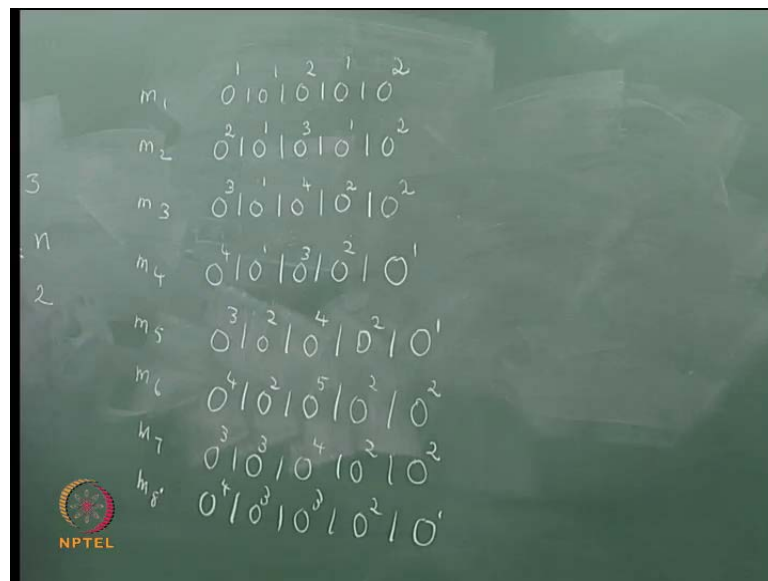
Now, we have a mapping of the form  $\delta(q_i, x_j) = (q_k, x_l, L/R)$  depending upon left or right. The mapping will be of this form that is when the machine is in state  $q_i$  reading the symbol  $x_j$ , it goes to  $q_k$ , prints the symbol  $x_l$  and either moves left or right. Now you know that we can use only three symbols. So, we have three symbols  $x_1, x_2, x_3$  this stands for zero and this stands for one; this stands for blank. We can have something like this; any other order is also acceptable. So, if you do that, you know that the value of  $j$  and  $l$  should be between 1 and 3 only and what can be the value of  $i$  and  $k$ ? There are  $n$  states from  $q_1$  to  $q_n$ . So,  $q_i$  have to be one of them, similarly  $q_k$  has to be one of them. So, the value of  $i$  and  $k$  should be between 1 and  $n$ ; again the move can be left or the move can be right. So, if it is left, you use 1. If it is right, you use 2 for that.

So, this move can be encoded as a binary string like this  $0^{i-1} 1 0^{j-1} 1$ ; that means, in the  $i$ th state, you are reading the symbol  $j$  then it goes to the state  $q_k$  that is 0

power  $k$  1, 0 power  $l$  1. That is you are going to state  $q_k$  and you are printing the symbol  $x_l$  and you are moving left or right. So, you have 0 power  $m$  here, where  $m$  is 1 if it is a left move, and  $M$  is 2 if it is a right move.

So, the value of  $M$  can be only 1 or 2; any move of the turing machine can be encoded in binary like this, this is one particular move of the turing machine and it can be encoded like this 0 power  $i$  1, 0 power  $j$  1, 0 power  $k$  1, 0 power  $l$  1, 0 power  $m$ . You must remember that,  $i$  and  $k$  can have values between 1 and  $n$   $j$  and  $l$  between 1 and 3  $m$  between 1 and 2. This idea we would be again using in universal turing machine, but let us take as an example this particular turing machine and see let us **see**, what is the mapping? Or what is encoding? This is the turing machine, the 3 symbols are 0 1 and blank there are 5 states  $q_1, q_2, q_3, q_4, q_5$ .  $q_1$  is the initial state and  $q_5$  is the final state. You see that there is no mapping for  $q_5$  without loss of generality, we assume that when it reaches the final state, it halts no next move is specified here. Now the mappings are given in this manner.

(Refer Slide Time: 27:44)



How do we encode this turing machine? The first move, how many moves are there? There are 8 moves 1, 2, 3, 4, 5, 6, 7, 8. Let us write them as move 1, move 2, move 3, move 4, move 5 and So on move 6, move 7, move 8. What is move 1? Delta of  $q_1 0$  is  $q_2 0$  or that is delta of  $q_1$  and zero is a... first 0 is  $x_1$ . So, you have 0 power 1, 0 power 1, you can omit the 1 if you want. It goes to state  $q_2$ , so 0 power 2.  $q_2$  it prints a 0, so 0

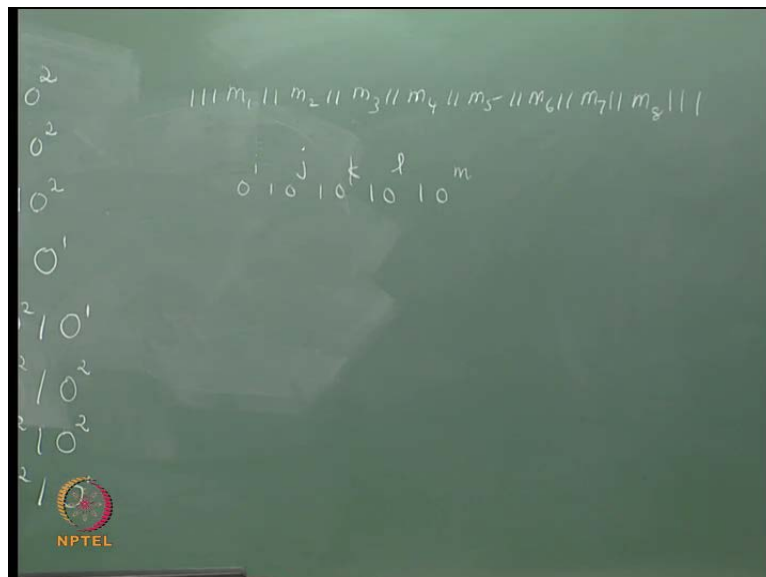
powers 1 and it moves right so, right is 2. So the first move is denoted like this, it is actually 0 1 0 1 0 0 1 0 1 0 0.

The second move is delta of q 2 0 is q 3 0 R. so that will be represented as, q 2 0 is q 3 0 R so, the second move is like this. The third move is delta of q 3 0 is q 4 1 R. So, it will be q 3 0 is q 4. 1 is denoted by symbol 2, so q 2 0 power 2 1 and then again right move. So, you have 0 powers 2. Note that a right move is denoted by 0 power 2 or 0 0 and the left move by 0.

The fourth move is q 4 is q 3 1 L. So, delta of q 4 0 is q 3 1 L, 1 is zero squared; L is only 1, so 0 1. So, this is move 4, move 5 is delta of q 3 1 is q 4 1 L q 3 1 is q 4 1 L is just 1. The next move is delta of q 4 1 is q 5 1 R q 4 1 is q 5 1 right move is this. Then there are two moves using the blank symbol, they are delta of q 3 blank is q 4 1 R and delta of q 4 blank is q 3 1 L. So, they will be denoted by move 7 and move 8. What are they delta of q 3? That is q 3 blank is symbol 3, so you have 3. See we have used x 3 for blank, so when you read a symbol blank, you use 0 power 3. It goes to state q 4, so 0 power 4 1 and it prints a 1 and the move is right. So, you have this.

The next one is q 4 blank is q 4 blank is q 3 1 L. So, q 3 1 L is 1. So, 0 power 1.

(Refer Slide Time: 32:12)

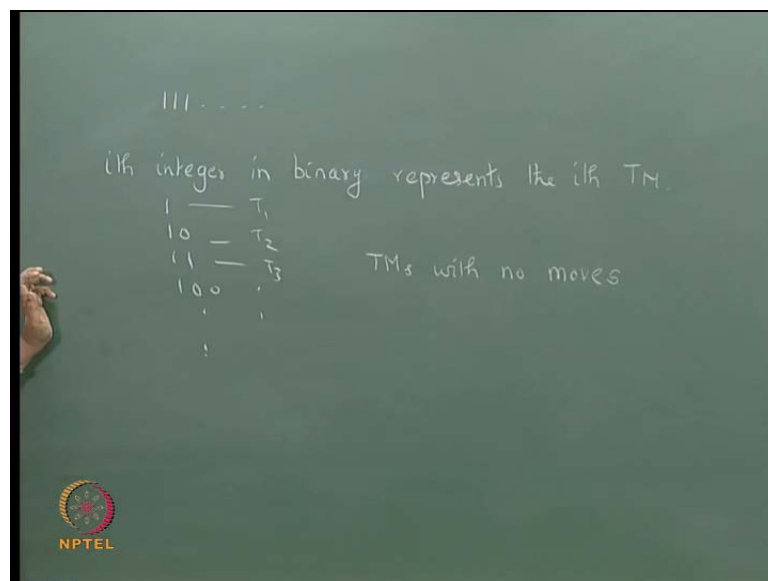


So, these are the eight moves for this example for the given example like this. The encoding of the turing machine will be like this. It is m 1, m 2, m 3 separated by two

ones.  $m_4, m_5, m_6, m_7, m_8$  the moves separated each **move is separated** from the other by two ones. You could have arranged them in a different order also, instead of arranging them as  $m_1, m_2, m_3$  you could have arranged this as  $m_2, m_3, m_1$ ; some other way also that **also** represents the same turing machine. It will be a different encoding that does not matter.

Now in the beginning, you have three ones and in the end you have three ones. So, the encoding of a turing machine begins with three ones and ends with three ones and each move is represented by something like this  $0^{i-1} 1 0^{j-1} 1 0^{k-1} 1 0^{l-1} 1 0^m$ . This is a block and blocks are separated by two ones. Thus you find that a turing machine can be encoded as a binary string; it can be represented as a binary string. If you rearrange them, you may get a different encoding. But that also represents the same turing machine.

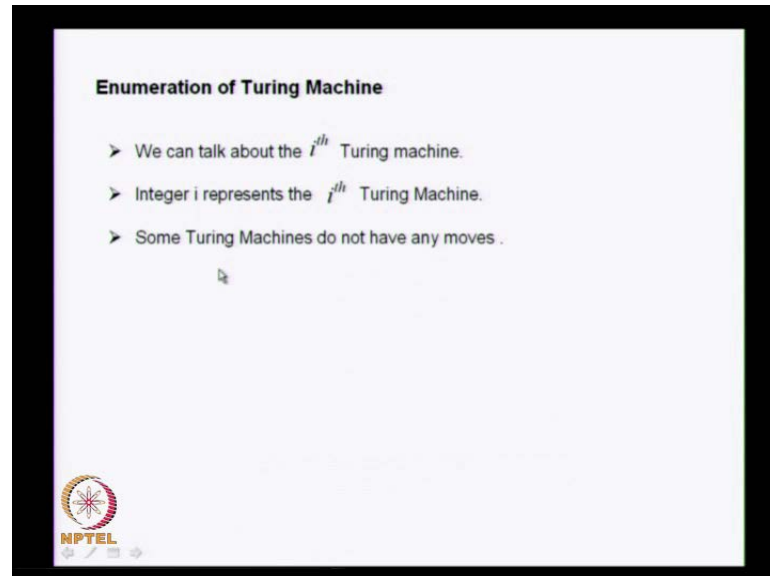
(Refer Slide Time: 34:08)



Having defined an enumeration encoding for turing machine, we can talk about the enumeration of turing machine. So, we know that a turing machine is represented by a binary string. So, suppose we say the  $i$  eth integer in binary represents the  $i$  eth turing machine, so you know that the integers in binary are like this and So on. This is turing machine, one turing machine, two turing machine, and three and so on. Now you must realize that the proper encoding should begin with three ones, some of them do not begin with three ones. So, they are improper encodings and you can look at them as turing

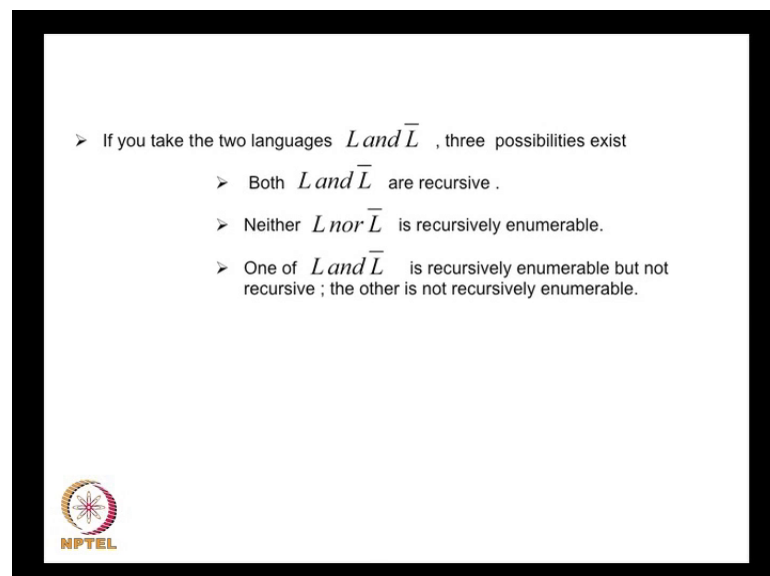
machines with no moves turing machines with no moves if the encoding is not proper, you can consider it as turing machine with no move.

(Refer Slide Time: 35:15)



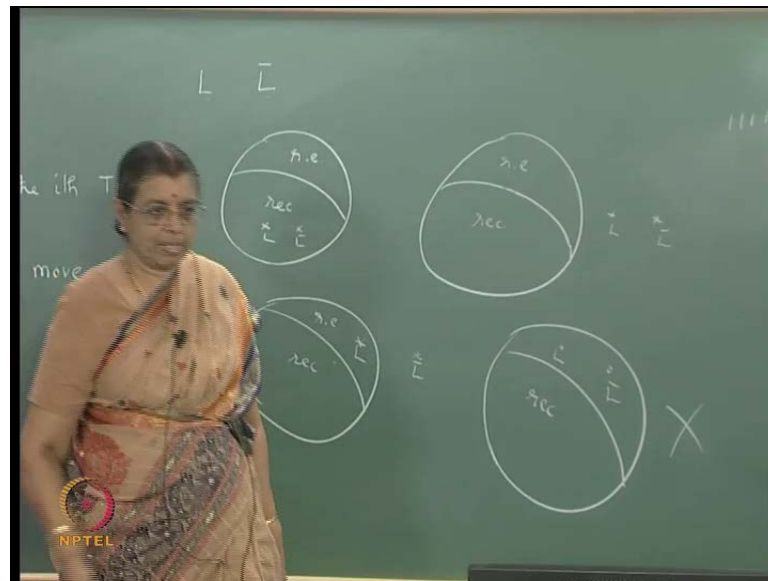
We now, we can talk about the  $i$  eth turing machine; the integer  $i$  represents the  $i$  eth turing machine. If the encoding is not proper or if the integer  $i$  does not begin with three ones or some other problem is there, it may not be a proper encoding. In that case, you look at it as a turing machine with no moves and if a turing machine has no moves, it accepts the empty set, it does not accept any string.

(Refer Slide Time: 35:44)



Now if you take 2 languages  $L$  and  $\bar{L}$ . A language and its complement

(Refer Slide Time: 35:51)



You can have three possibilities: both can be recursive  $L$  and  $\bar{L}$  can be both recursive and neither of them are recursively enumerable or one is recursively enumerable. But not recursive and another one is not recursively enumerable. That is if we look at the diagram, the recursive and the recursively enumerable, there are three possibilities both  $L$  and  $\bar{L}$  can be here. Both of them can be recursive or you can have recursive recursively enumerable, one can be here,  $L$  can be here,  $\bar{L}$  can be here. This is possible and the third possibility is recursive recursively enumerable both  $L$  and  $\bar{L}$  can be here. This is you will not have these possibilities are there, you will not have this situation. That is  $L$ ,  $\bar{L}$  both are recursively enumerable, but not recursive this possibility does not exist. Why we have already seen that if both  $L$  and  $\bar{L}$  are recursively enumerable, they are recursive,  $L$  is recursive and hence  $\bar{L}$  is also recursive. So, this possibility will not exist.




(Refer Slide Time: 37:40)

> We can talk about enumeration of strings and hence enumeration of Turing Machines

> Consider an infinite Boolean matrix  $D$ , where the  $(i, j)^{th}$  entry is 1 if the  $i^{th}$  accepts the  $j^{th}$  string and 0 otherwise. Consider the following two languages

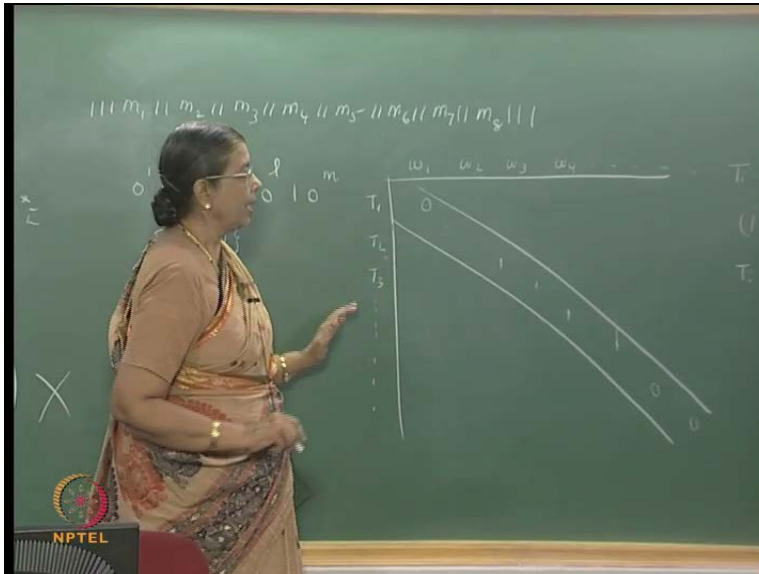
$$L_d = \{w \mid w = w_i \text{ for some } i \text{ and the } (i, j) \text{th entry in } D \text{ is } 0\}$$
$$L_d = \{w_i \mid w_i \text{ is not accepted by } T_i\}$$
$$\bar{L}_d = \{w \mid w = w_i \text{ for some } i \text{ and the } (i, j) \text{th entry in } D \text{ is } 1\}$$
$$\bar{L}_d = \{w_i \mid w_i \text{ is accepted by } T_i\}$$

> It can show that  $L_d$  is not recursively enumerable and  $\bar{L}_d$  is recursively enumerable but not recursive.



Now, you can think of an infinite Boolean matrix.

(Refer Slide Time: 37:46)



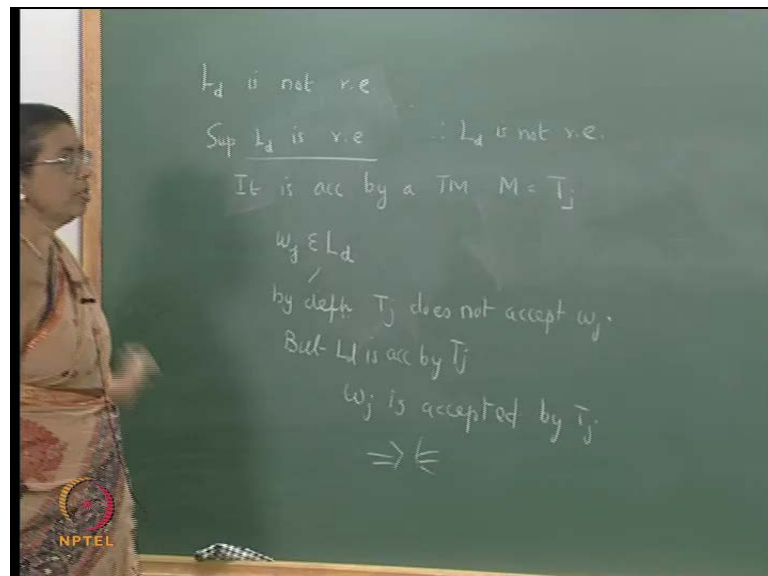
We already saw that, strings over the alphabet 0 1 can be enumerated like this 0 1 then 0 0 1 1 0 1 1 and so on. So, you can always talk about the  $i$  eth string in this enumeration. So, you talk about  $w_1, w_2, w_3, w_4$  and so on. You can also talk about turing machines  $T_1, T_2, T_3$ , because we can have an encoding of the turing machine and hence an enumeration of the turing machine. Now you consider an infinite Boolean matrix where the entries are 0 or 1. The  $ij$  eth entry will be a 1 if  $T_i$  accepts  $w_j$ , **if  $T_i$  accepts  $w_j$**  the  $i$

$j$ th entry will be ... If  $T_i$  does not accept  $w_i$ , does not accept  $w_j$ , then the  $i, j$ th entry is 0. Having said this, we can define two languages  $L_d$  and  $\bar{L}_d$ . Look at the diagonal elements of this infinite Boolean matrix. There will be some 0s, some 1s and so on. Take all those elements which correspond to 0 that is  $L_d$ . Take all those elements which correspond to 1 that is  $\bar{L}_d$ . So, we define like this, taking the infinite Boolean matrix  $d$ , where the  $i, j$ th entry is 1, if the  $i$ th Turing machine accepts the  $j$ th string. Otherwise it is 0.

Consider a language  $L_d$ ;  $L_d$  is  $w$ , where  $w$  is equal to  $w_i$  and for some  $i$ . The  $i, i$ th entry, it is the  $i, i$ th entry in  $D$  is 0.  $w_i$ ,  $w_i$  is not accepted by  $T_i$  that is the  $i$ th string is not accepted by the  $i$ th Turing machine. The complement of that language where you consider the elements to be 1, it is  $w_i$  is equal to  $w_i$ , and  $i, i$ th entry is 1. It is the complement is  $w_i$  is accepted by  $T_i$  because the diagonal entry is 1.

Now it can be shown that  $L_d$  is not recursively enumerable and  $\bar{L}_d$  is recursively enumerable but not recursive. That is, in this one, it comes under this is  $L_d$ , it is not recursively enumerable and the complement  $\bar{L}_d$  is recursively enumerable but not recursive. Let us see how it is possible.

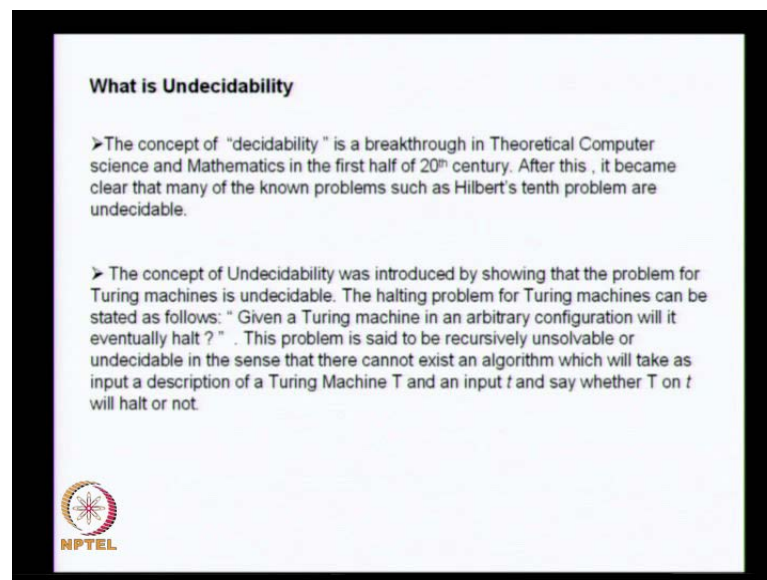
(Refer Slide Time: 41:26)



First we will show that  $L_d$  is not recursively enumerable  **$L_d$  is not recursively enumerable** why? Suppose  $L_d$  is recursively enumerable then it is accepted by a Turing machine  $M$ , if it is say the  $j$ th Turing machine. Now what can you say about  $w_j$ ?  $w_j$

belongs to  $L_d$  means, what does that mean? By definition what is  $L_d$ ?  $L_d$  is not,  $L_d$  corresponds to the zero elements. So, by definition  $T_j$  does not accept  $w_j$ , but  $L_d$  is accepted by  $T_j$  by our assumption. Turing machine I have said  $M$  is equal to  $T_j$ ,  $T_j$ , I have been using the symbol  $T_j$  for that, so  $T_j$ . By definition,  $L_d$  is accepted by  $T_j$ . So,  $w_j$  belongs to  $L_d$  means,  $w_j$  is accepted by  $T_j$ , they are contradictory statements. You say that  $w_i$  is not accepted by  $T_j$  and here you say that  $w_j$  is accepted by  $T_j$ . So you are arriving at a contradiction and the contradiction has come because of the assumption that  $L_d$  is recursively enumerable. Therefore,  $L_d$  is not recursively enumerable. We will make use of this fact in improving what is known as the halting problem for turing machines.

(Refer Slide Time: 44:03)



Now, we will come to the most important concept in the whole of this course I would say, the concept of decidability. What is decidable? This is a very break through concept which was stated by Turing in 1936 and two of the results have been considered as very important in theoretical computer science. One is the Godel's incompleteness theorem and the other is the halting problem for the turing machine till this result was proved people were trying on; so many problems trying to develop algorithms and so on. But after this result was proved, they knew that for some problems, nobody can ever write an algorithm. Hilbert's tenth problem was one of them. In 1901 Hilbert stated this problem that is, can you write an algorithm which will take as input, a polynomial with integer coefficients and that equation polynomial equal to zero, whether the that has integer

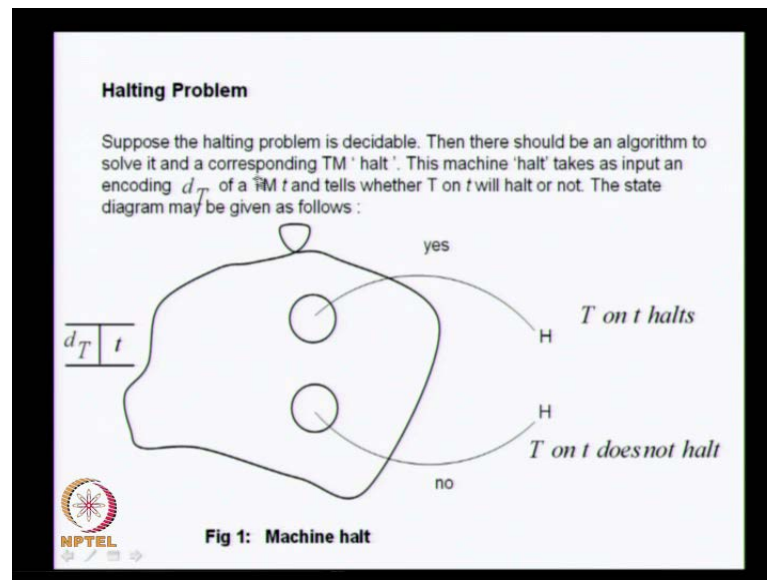
words or not. Without factorizing, how can you go about finding that can you give an algorithm for that?

And people have been trying it for a long time and nobody was able to find an algorithm. So, similarly say there were other problems as well. So, after this result was stated, they knew that nobody can ever write an algorithm. It is a very strong statement to say that nobody can ever write an algorithm for that or an algorithm cannot exist for a problem. Because it is not that you do not know, how to write or I do not know, how to write or somebody else does not know, how to write a algorithm for a problem. Nobody can ever write an algorithm for that.

That also gave an idea about the Fermat's last theorem. Fermat's last theorem is, can you find integers such that  $x^n + y^n = z^n$ , for  $n$  greater than or equal to 3? Can you find integers  $x, y, z$  and his conjecture was that you cannot find integers and for a long time it was not proved and finally, about ten years back it was proved. But this result that the halting problem is undesirable gave a clue to this. That is, thus it is a decidable problem that is Fermat's last theorem can be true. It can be proved or it cannot it can it need not be proved, but there is a possibility either it is true or it is not true .

Nobody may be able to find out the proof, because of Gödel's incompleteness theorem. But still an answer to that can exist. The concept of undecidability is that is why considered as a very important problem. For the turing machine, it is stated like this, we shall deal with more on decidable problems undecidability later. But let us consider the halting problem for turing machine now. The halting problem for turing machine can be stated as follows: Given a turing machine in an arbitrary configuration will it eventually halt? What it means that, can you give an algorithm which will take a turing machine and an input as input and say whether the turing machine on that input will halt or not? The problem is said to be recursively unsolvable or it is undesirable in the sense that, you cannot write an algorithm. There cannot exist an algorithm which will take as input a description of turing machine  $T$  and an input  $t$  for the turing machine and say whether  $T$  on  $t$  will halt or not.

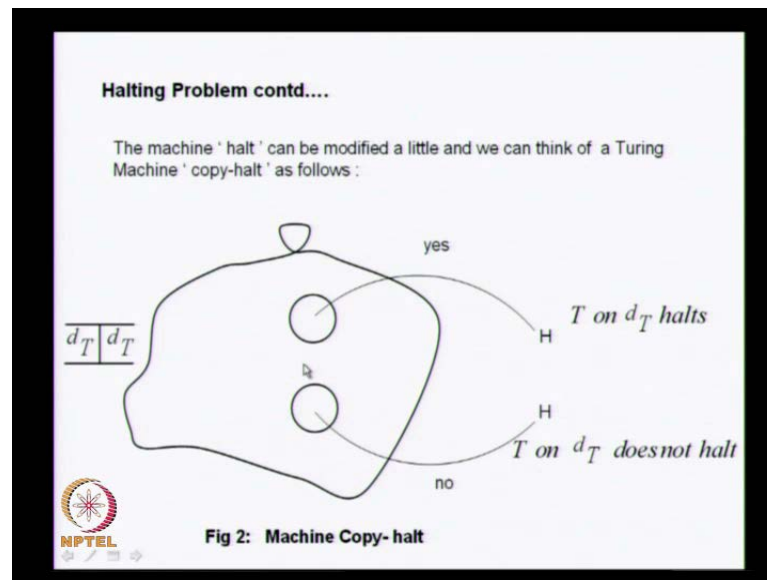
(Refer Slide Time: 48:34)



Let us give a brief proof for this, we shall consider one more proof later. So, I will skip this for the moment. The halting problems suppose, it is decidable, and then there is an algorithm for this. We have already seen that, an algorithm corresponds to a turing machine which always halts. So, you have a turing machine whose state diagram is roughly this. The input for that is the description of the turing machine or the encoding of the turing machine and an input  $t$  for the turing machine and this will always halt and say whether  $t$  on  $t$  will halt or not.

Suppose the halting problem is decidable there should be a turing machine which does that. So given this input, if  $T$  on  $t$  halts the machine will go to this state and take this exit and halt and say yes. Suppose  $T$  on  $t$  does not halt, then the machine will go to this state, take this exit say no and halt. In either case, it always halts and tells you whether  $T$  on  $t$  will halt or  $T$  on  $t$  does not halt. So, assuming that there is an algorithm for the halting problem, there should be a turing machine halt like this.

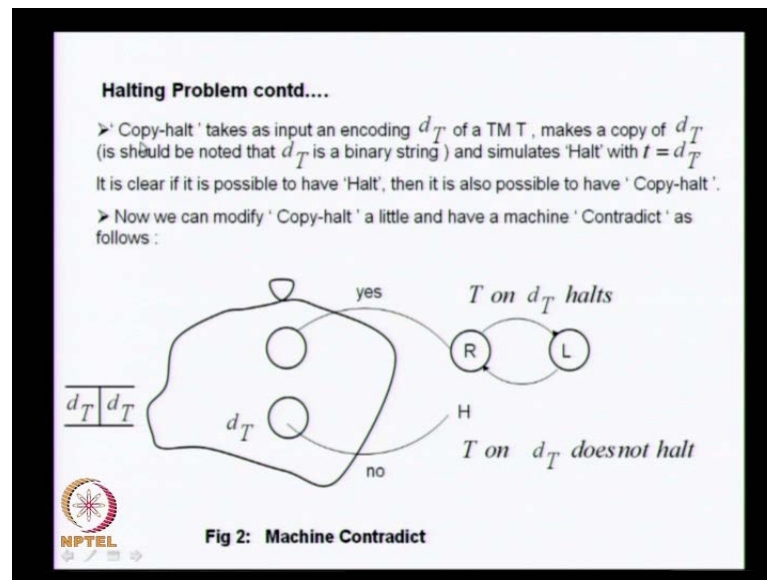
(Refer Slide Time: 49:46)



Now I will modify the halt machine and call it as a copy halt. You know that the encoding of a turing machine is a binary string, the input is also a binary string. So, instead of taking any  $t$ , I take small  $t$  as the encoding itself. That is the machine starts with one binary string  $d_T$ , then it makes a copy of that string. Then calls halts as a sub routine. So, it answers when it calls halt as a sub routine, it will answer the question whether the machine  $T$  taking the encoding its own encoding will halt or not.

So, if  $d_T$  **please remember** that  $d_T$  is a binary string, this is also a binary string. So, if the machine  $T$  halts on this binary string, it will take this exit and halt. If the machine  $T$  does not halt on this binary string, it will take this exit and halt  $T$  on  $d_T$  does not halt. So, if it is possible to have a machine like halt, it is possible to like have a machine like copy halt.

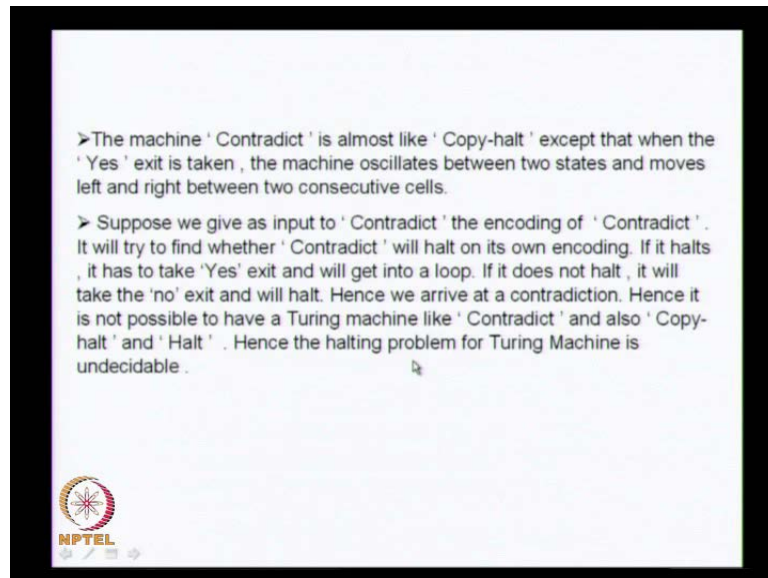
(Refer Slide Time: 51:11)



So, again repeating the same thing, copy halt takes as input and encoding  $d_T$  of a Turing machine  $T$  makes a copy of it and simulates halt with  $T$  is equal to  $d_T$ . So, if it is possible to have halt, it is possible to have copy halt.

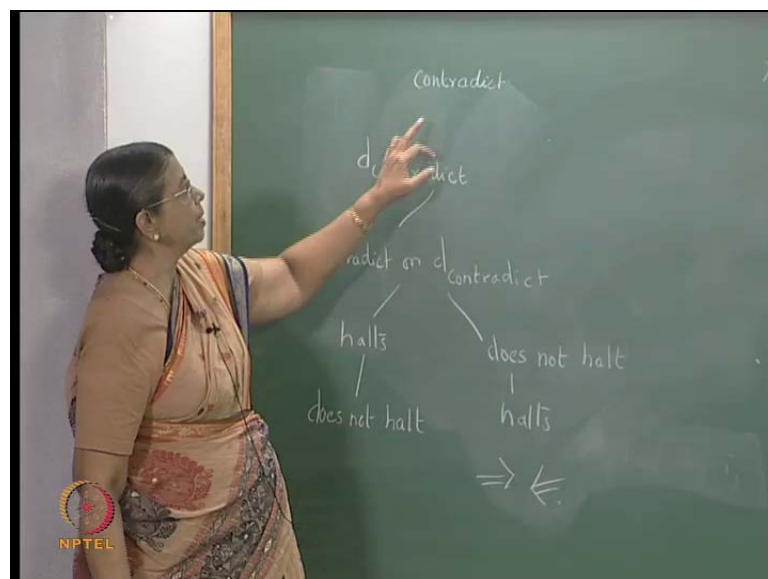
Now, the copy halt machine is modified a little bit, to have a machine contradict. How does contradict work? It is the same as copy halt, but when you take the yes exit instead of going to halt state, it goes to one of these states and afterwards, it just starts oscillating between adjacent cells moving left and right. So, when it takes the yes exit, it goes to this state and starts oscillating between two states that that means it does not halt. Now, what happens when contradict is given as input?

(Refer Slide Time: 52:14)



So, the machine contradict is almost like copy halt except that. When the yes exit is taken, the machine oscillates between two successive states and the two successive cells. Now for this machine, the input is the encoding of any turing machine, **if you give as encoding**, if you give as input the encoding of this machine the d contradict what will happen? There are two possibilities.

(Refer Slide Time: 53:03)



The input is d contradict, this is given as a input to the machine contradict. Now, when this is given as a input, there are two possibilities contradict on d contradict, it halts or



does not halt. When it halts, it has to take this exit and when it takes this exit **exit** it gets into a loop. That means, it does not halt, when it does not halt **when it does not halt**, it takes this no exit, but then it halts.

So when it halts, it takes the yes exit and gets into a loop and you come to the conclusion that, it does not halt and when it does not halt, it takes the no exit and you come to the conclusion that it halts. So, you are arriving at a contradiction and the contradiction shows that you cannot have a machine like contradict.

We have seen that, if you can have a machine like halt, you can have a machine like copy halt and hence a machine like contradict. So, the fact that a machine like contradict cannot exist shows that, you cannot have a machine like halt. You cannot have something like this. That is, you cannot have an algorithm which solves the halting problem of the turing machine.

So, these are a very important concept and after this has been proved, so many well known problems have been shown to be undecidable. Usually, we will consider the decision problems that is problems, which have the answer yes or no and many of them have been shown to be undecidable that is, we cannot have an algorithm for that.

So, let us consider this in detail later. When we deal with problems instances or problems? How you represent them as languages? And what do you mean by the language being undecidable and So on. You also see, what is the connection between languages and problems when do you say that a problem is undecidable? When do you say that a language is undecidable, how do you represent a problem in terms of languages and so on? This we shall consider later, but considering the infinite Boolean matrix, which we looked into earlier like this. We can have another proof for the halting problem of the turing machine, the argument for that will be something like this.

If you can have an algorithm for the halting problem for turing machine, then the language  $L_d$  becomes recursively enumerable. But you know that, it is not recursively enumerable. So, you cannot have an algorithm for the turing machines. So, this second proof we shall consider later.