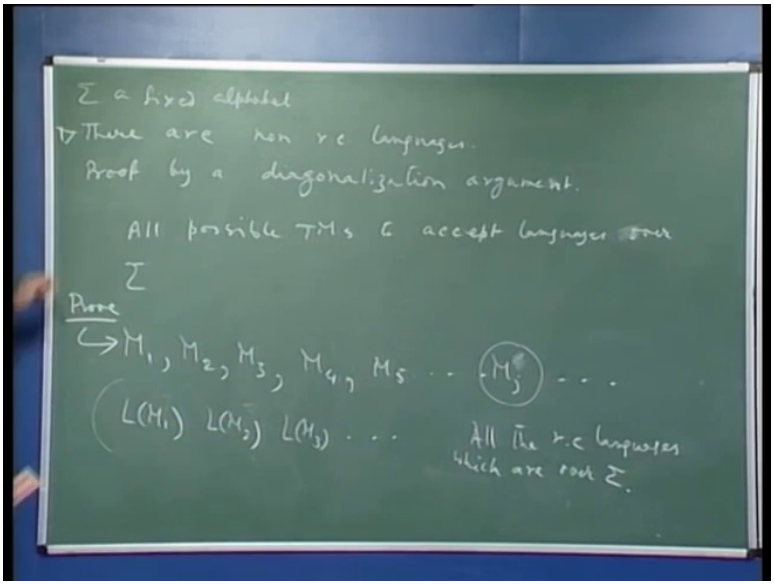
Course on Theory of Computation
By Professor Somenath Biswas
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
Lecture 41
Module 1
Existence of non-r.e. languages, recursive languages, notion of decidability.

(Refer Slide Time: 0:32)



We continue our arguments that will show that there are non-recursively enumerable language, we recall that we said that the proof will be by a diagonalization argument and to begin that argument first thing we need to prove is this fact that all possible TM's which accepts languages over sigma you know sigma is a fixed alphabet. So all possible TM's which accepts languages over sigma that is all those languages which are subsets of sigma star these Turing machines can be lined up.

What we mean by lined up? That you can imagine we have a sequence M1, M2, M3, M4 etcetera these are all Turing machines and each one of these L(M1), L(M2) these are languages accepted by the corresponding Turing machine and all these languages are languages over sigma that is one thing so that means each one of these Turing machines accepts some language could be empty would be sigma star they all accept languages over sigma and the other thing is that this enumeration this listing is exhausted in the sense that if there is any language which that is

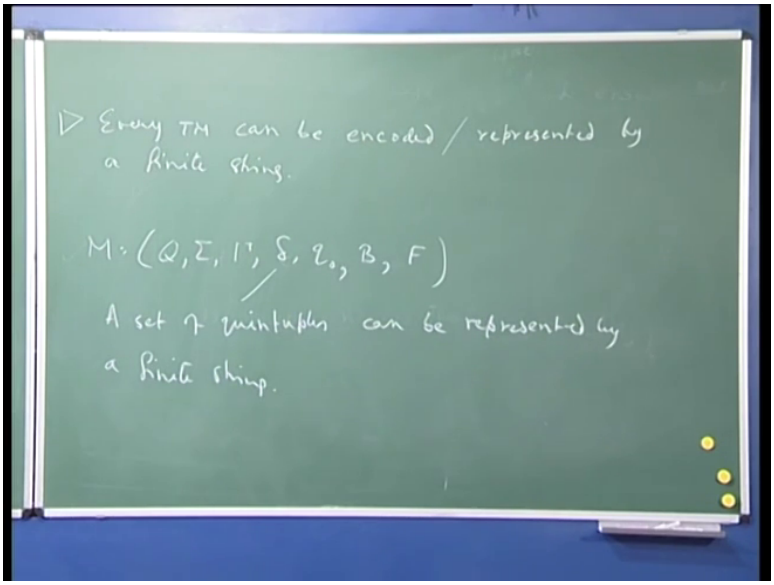
consider any language which is over sigma, so for that language if there is a Turing machine to accept it that Turing machine will occur somewhere in this enumeration, alright?

Now so that would mean what? So the two things that we are saying that all these are Turing machines which accepts languages over sigma and any Turing machine that accepts a some language over sigma will come somewhere in this enumeration which really means if you see immediately this point that the all the re languages which are over sigma they can be thought to be in this define through this enumeration because you know they are an r.e language over sigma they must by definition be a Turing machine to accept that re language and because of the property that we said that somewhere down the line that Turing machine would occur.

So therefore the list of corresponding r.e languages as we have put the shown the correspondence that list will also have all languages which are r.e over sigma. Now what we show by diagonalization that language that we construct is different that language we call it Ld d for diagonalization is different from each one of these and therefore Ld is not r.e over sigma because had it been an r.e language then there would have been a Turing machine to accept it and that Turing machine would have occurred somewhere down the line and then the corresponding language would have been here in this enumeration but since our construction will be such that that it will be different from each one of these languages Ld is not re and by construction we will see Ld is also a language which is over sigma.

So the first point is this that enumeration of all Turing machines that accepts languages over sigma and that we show by showing that each one of these Turing machines can be encoded as a string.

(Refer Slide Time: 5:11)



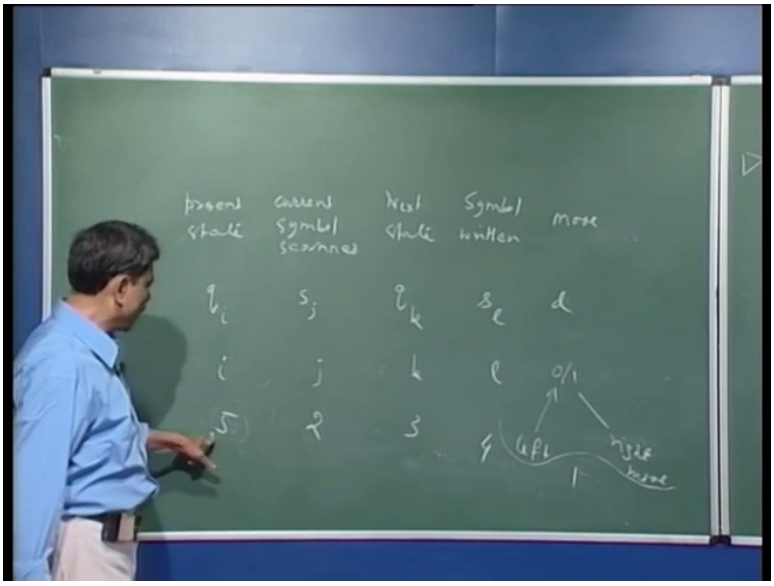
And this is a very important point that every Turing machine can be encoded for equivalently you can think of represented by a finite string. What we are saying is you gave me any Turing machine I will find a finite string which is encoding or which is the representation of that Turing machine.

Now of course we know a Turing machine is what? Turing machine can be seen as what defines a Turing machine recall Turing machine M has a set of states, then it has a input alphabet sigma, tape alphabet gamma, then of course transition function, then it has initial state we need to specify, right? Then it will also have a symbol which we call blank because those are the that particular symbol will there initially in most of the tapes, right? And then of course it will have some finial accepting states, right? This will be a Turing machine.

Now recall that this delta we can be represented by means of a set of quintuples, alright? Now if I just give you the set of quintuples corresponding to a particular Turing machine M from there you can figure out what are the what is this set which is the set of states what are the tape alphabet recall that input alphabet by default is sigma that is standard the things that you do not have this information that which are the which is the initial state which is the blank symbol and which is or the set of final accepting states, alright.

So first of all we will figure out how to represent that but first of all that the set of quintuples can be represented, okay can be represented by a finite string. Now remember this is possible because this set of quintuples for any Turing machine M that is a finite set, alright?

(Refer Slide Time: 8:32)



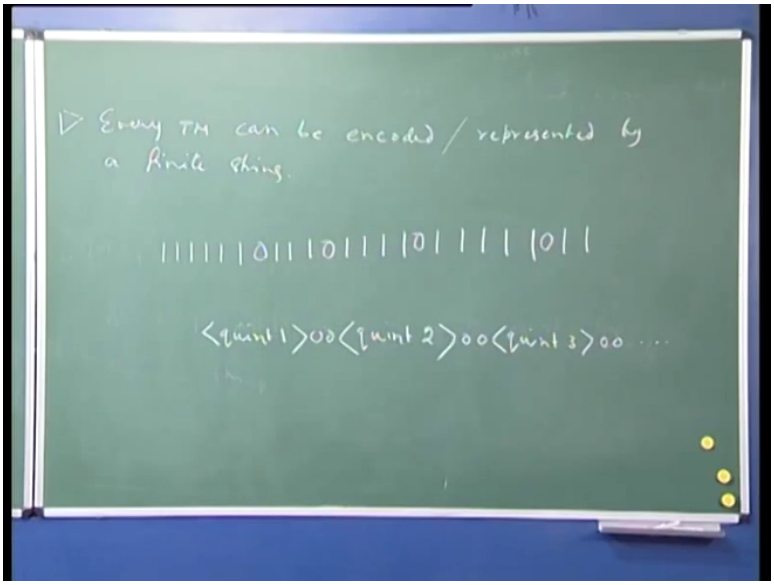
So let us what is a quintuple we can think of you know a quintuple as quintuple by definition of course is a five tuple and these five things are present state, the current symbol scanned, then next state and then the symbol written on the square he was scanning at that particular step and move by this mean whether the after writing the symbol whether the Turing machine will move one square to the left or to the right.

Now supposing the present state is qi the current symbol that we were scanning is the jth symbol in next state is kth state symbol that we were writing is s of l and the move is d, d can be left or write and each one of this can be coded as some number you see for example I can think of representing by i the number I the state qi which is very clear what the names of states are unimportant so long I have some ways of differentiating each of these states from other states.

So imagine that I am naming states as really numbers so state qi is the number i that is it is presentation same with symbol the symbol is j I can represent a number j and this of course will be represented by a number k, this will be represented by a number l and the move is either 0 or 1, right? 0 for maybe left move and one for right move. Now we are continuing proving this fact one particular quintuple can be represented just my means of this five numbers.

Now suppose I choose to represent this numbers unary, take a particular example let us say this is the this is the fifth state we have the number 5 the symbol is second symbol it goes to third state, writes the fourth symbol and it moves to right. So the 5 numbers 5, 2, 3, 4, 1 would capture this quintuple.

(Refer Slide Time: 11:37)



So as I said we will suppose I write this numbers in unary, so 5 I will have to put you know an unary input five 1's plus 1 so six 1's to represent 5 because the number 0 you will represent by just 1 and so on, right?

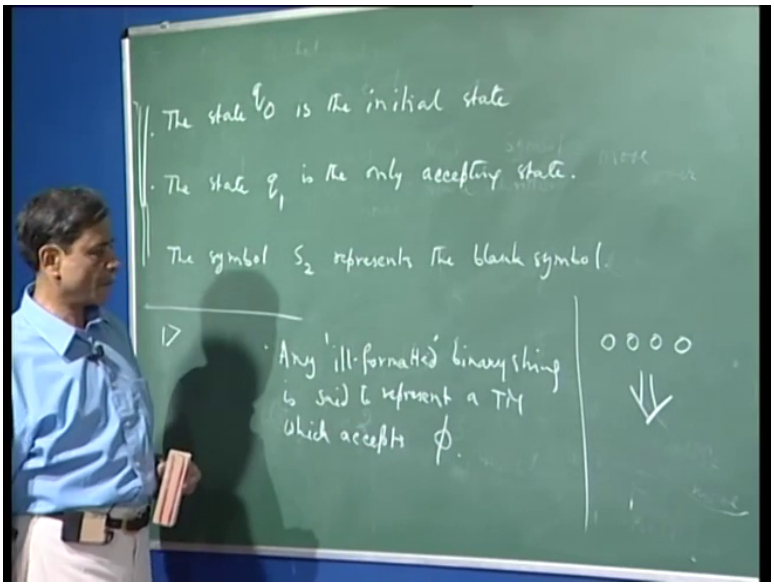
So for 2 you will have three 1's that is the unary representation of 2 for 3 you will have four 1's, right? For 4 again you will have five 1's and then I said 1 that is the right move so you will have two 1's. Now what we can do is to separate these by 0's, right? So a binary string therefore represents the information contain here, right? And now you want to represent a set of quintuple, so supposing the representation of one quintuple in a binary string let us say you know quint 1 this is our representation of first quintuple which will be a binary string notice in that binary string two zeros are never consecutive then you have quintuple 2, okay.

Now you can separate them out by putting two 0's in between, okay. So a set a finite set of quintuple can be represented by means of a binary string the idea is very simple the two consecutive 0's will separate the representation of quintuples and in the each quintuple a single 0 will separate two parts of a quintuple, right? So this is you can imagine that this therefore we will

represent this idea is a very simple idea we will represent any finite set of quintuple does not matter how big this set of states is or how big the set of symbols is because we are after all each such set is finite there will some finite representation of quintuple the total number of quintuples for a Turing machine will be finite so therefore we have a binary string to represent all the quintuples.

Notice this has all this information like the set of states, set of symbols, right? These information we have we do not have in this representation what is the initial state neither we have the information what are the final states.

(Refer Slide Time: 15:04)



So that this two pieces of information we can keep implicitly so let us say we just choose arbitrarily but fix it once for all that the state 0 is the initial state or rather if you are seeing q0 that means the number the states whose representation is the number 0 here now is the initial state, right? And you also do not have an information about what is the or what are the accepting states.

So again without loss of generality you can assume that the state q1 whose representation will be the number 1 that is the only accepting state. Now suppose you had a Turing machine which had many accepting states from there wherever the machine enters such an accepting state instead of halting there you go to a special state and halts there and say that is the accepting state. So is not difficult at all even if the original machine had a number of accepting states to define another

Turing machine with a single accepting state which will be equivalent so far as the language recognized is concerned.

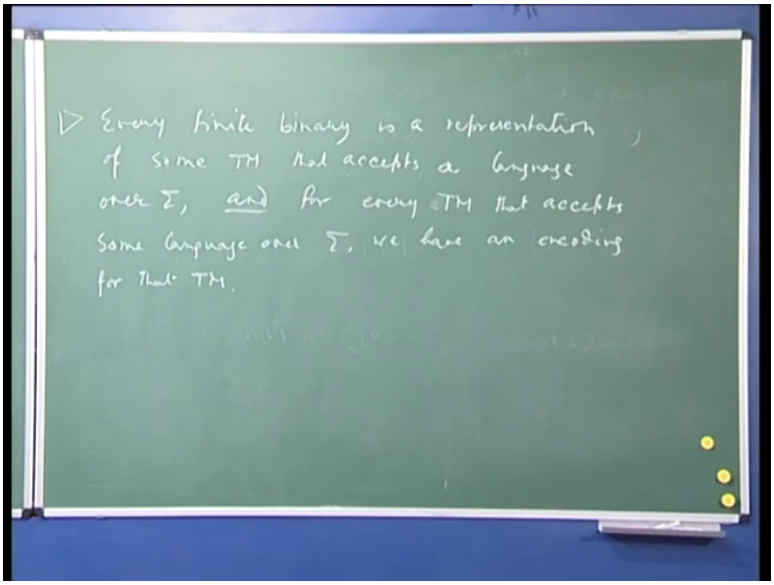
Same way you know if you somebody has designed the Turing machine which is not in which q0 is not the initial state you can just map to a simple map and (())(16:51) state with q0 and that would give a initial state, so these thing you can have without loss of any generality. Now only thing we have not said what is the blank symbol again we can arbitrarily choose one particular symbol to be the blank symbol so let us say the symbol you know is to represent the blank symbol.

So once I have these implicit understanding or convention in place then I can claim, what is that claim is this that binary string by a finite string and in fact over the alphabet 0, 1, right? The binary string finite binary string will input a Turing machine, right? What more? You know this to make our life very simple so what we have said for every Turing machine we have a binary string to represent it. Now I would like to make it go the other way also I would like to say that every finite string represents some Turing machine, okay.

So right now we will, ohh how is that possible? You know a string like 0000 what does it represent? After all there are no quintuples I can see in this etcetera etcetera etcetera. Now basically any such string which is not really representing a set of quintuple in this manner I choose to call them to represent I choose to say that yes they represent a Turing machine which accepts the empty language.

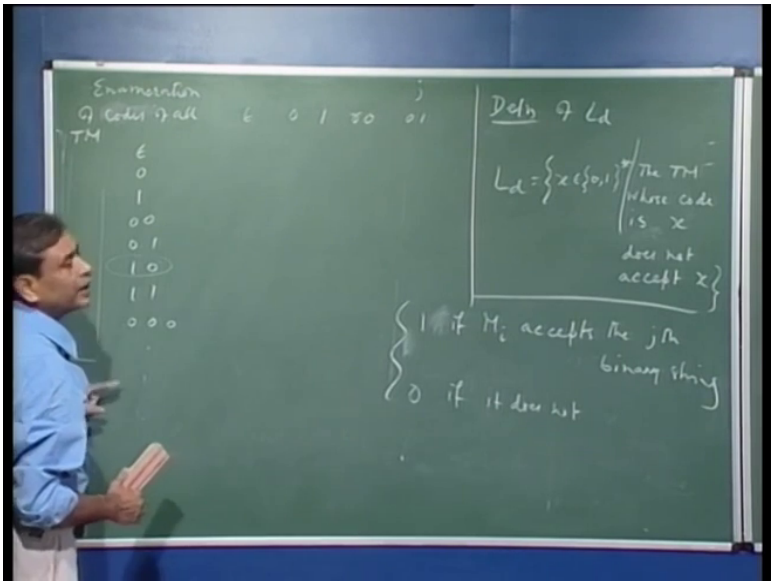
So now we say that any ill formatted binary string, so what do mean by ill formatted? Which you cannot see as representing a set of quintuple and that is we need to see even such a string you can easily figure out whether it is ill formatted or not we are now saying any ill formatted string is said to represent a Turing machine which accepts no string so basically which accepts the language phi that is the empty set, alright?

(Refer Slide Time: 20:20)



So put together all these what I am saying is every finite binary string is a coding our representation of some Turing machine that accepts a language over sigma and for every Turing machine that accept some language over sigma we have an encoding for that TM, okay. So now once you believe this statement then we have this situation just think of a canonical ordering of all binary string all finite binary strings you know such a canonical ordering is you can one possibility is that you consider the strings to be ordered in (())(22:20) graphically you know with the smallest first, so basically first you will have the empty string, then 0, then 1, then 00, then 01, 10, 11, 000 and so on.

(Refer Slide Time: 22:25)



Now where choosing to consider each one of these string to be representing a Turing machine which is accepting some language over sigma, right? So therefore this I can see as enumeration of all Turing machine or enumeration of codes of all TM's accepting languages over sigma, right? So this is this and of course you we also can enumerate the same manner the set of all finite binary strings supposing we do that in this fashion same enumeration except our understanding of these is that they are strings binary strings and these are Turing machine representations which accepts languages over some sigma, right?

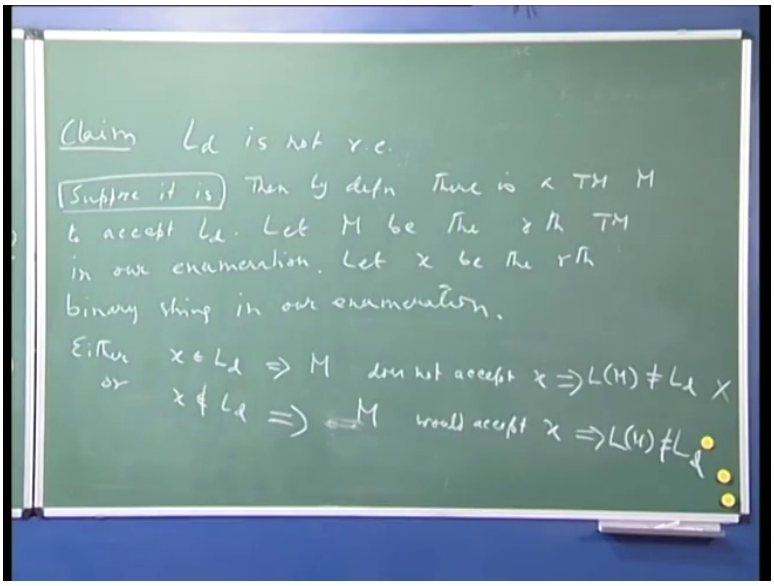
Now without any loss of generality you can assume that both these symbols will be in sigma, right? So our input alphabet has both these symbol 0 and 1 this is we can assume is of loss of generality. Now you see in a manner of writing this what I am defining is a kind of matric which is A or let A and the entry aij corresponds to our the ith Turing machine and the jth binary string, okay and let me define it this way conceptually of course this is going to be an infinite matrix and nobody is going to (())(25:15) you know fill it up conceptually imagine that you have defined the entry this matrix in the following manner that aij is going to be 1 if write it this way 1 if the Turing machine i accepts the jth binary string, okay that is in this enumeration jth means the jth thing here is enumeration and 0 if it does not.

Now just so basically now our job will be fairly easy job of describing Ld the language which we define and which we will show to be not recursively enumerable that Ld is something let us say

we define here definition of Ld Ld is a binary string x so x will be the set of all x's all those x's will be in Ld which will satisfy a certain condition and what is the condition, the TM whose code is x does not accept x the string x. So you see it is like this a string x is let us say the jth string, right? Supposing x is the jth string there and then we will look at the jth Turing machine here and we are concentrating about what is the behavior of the jth Turing machine on the jth string.

So in a way we are talking of the diagonal elements of this matrix which we will define whether or not that particular string is a member of Ld, alright?

(Refer Slide Time: 28:18)



Now the claim is Ld is not recursively enumerable, why? Suppose it is in by definition then by definition there is a Turing machine, okay M to accept Ld. Now that Turing machine will come somewhere in the enumeration of all Turing machines which accepts languages over sigma, so let us say that let M be the rth Turing machine in our enumeration, okay.

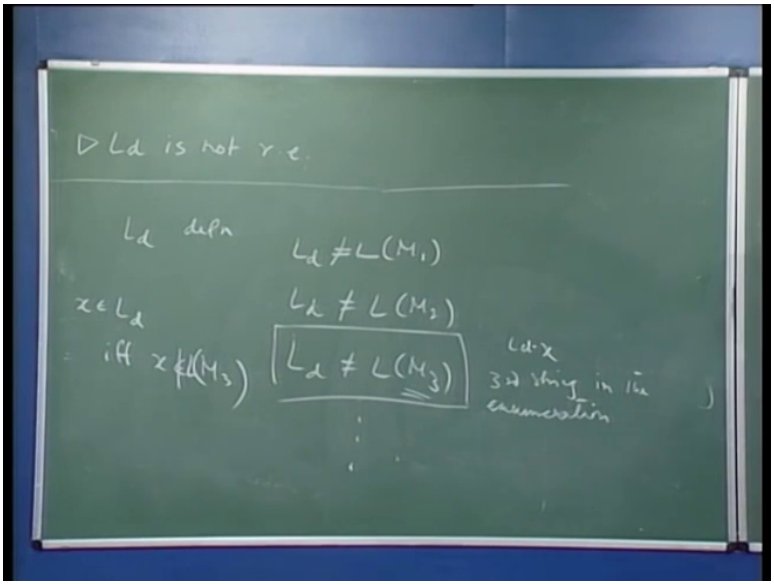
Now consider let x be the rth binary string in a enumeration is fairly simple thing what we are saying that if Ld is recursively enumerable then there is a Turing machine somewhere whose code is somewhere down this line would accept Ld let that Turing machine be the rth Turing machine in the sense the rth binary string here represents that particular Turing machine and now we are considering x to be that rth binary string, right? Do you understand that we are making, we actually looking at a string in two ways that string as a string also that string as a encoding of a Turing machine.

So now there are two possibilities are there, alright? What are in the sense that either this x, x is in Ld or x is not in Ld, okay. Now consider each of these cases separately supposing x is in Ld that means what x would have satisfied this condition and what is that condition, why see again since x is in Ld only those strings which will are in Ld which satisfy this particular condition what is this condition saying the Turing machine whose code is x but what is that Turing machine that Turing machine is M, M does not accept x.

So if x is in Ld then from the definition of Ld we have that M does not accept x but then it is clear that the language accepted by M is not Ld, right? Because we said that M is the Turing machine which accepts Ld, alright. So this case leads us to a contradiction because x is in Ld that means what if you x is accepted it should be accepted by M but then we are seeing that L(M) is different from Ld.

So the other possibility is x is not in Ld but if x is not in Ld that means what x is not in Ld would mean that x or you know look at it this way the rth Turing machine would accept the rth string, but what is that rth Turing machine? rth Turing machine is of course the machine M, so I can say M would accept the rth string and of course the rth string is what is x. So now here is a string x which is accepted by M which is not in Ld, so therefore again I can conclude that the language accepted by M is different from Ld, alright? So these are the only two possibilities either x is in Ld or x is not in Ld in either case assuming each one I am going to a contradiction. Therefore what can I conclude? I can conclude that this supposition is false we said that suppose Ld is recursively enumerable so this proves by contradiction that Ld is not recursively enumerable.

(Refer Slide Time: 34:55)



You know I mean if you look at that proof once more you will see our construction of Ld or our definition of Ld what do I do one in defining Ld I want Ld to be not an r.e language so I should have Ld is not the language L(M1) where M1 is the first Turing machine in our enumeration of all Turing machine that accepts languages over sigma.

Similarly I have to ensure that L is not L(M2) Ld is not L(M3) and so on, right? Which is the string see for each one of these we have a witness of this fact if you see that the second string in this case or third string consider the third string in the enumeration of all binary string let it be x let x be the third string in the enumeration of all binary string and that of course represents the machine M3 also.

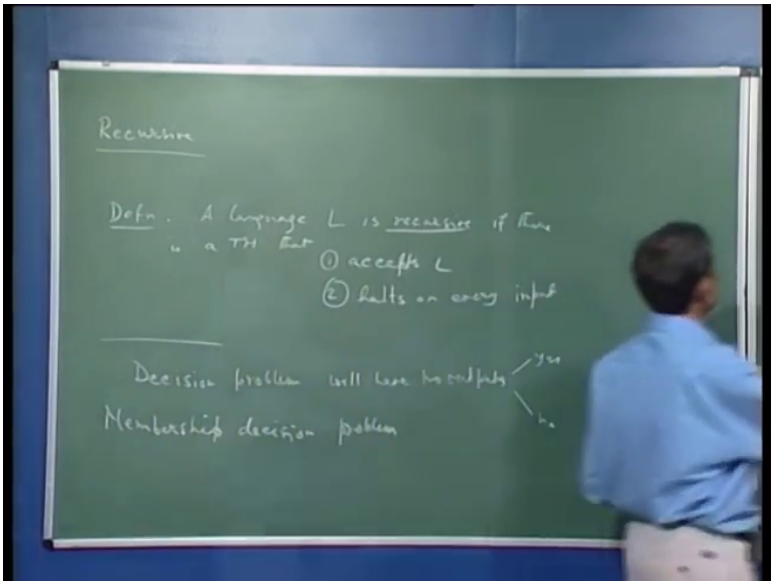
Now what we have said that we will put x in Ld if and only if x is not in the language accepted by the machine whose code is x but since x is the third string in the enumeration this machine is also the third Turing machine, right? That is how we have to define, right? We are we will put x in Ld provided x is not in L(M3) we will not put x in Ld if and only if x is in L(M3). So this string x which is the third string in the enumeration is the witness of this fact.

So essentially what you have done by your definition of Ld that you have a string which witnesses the correctness of each one of these aspects. Since Ld is not the language accepted by any of these Turing machines and since all these Turing machines collectively exhaust the class

of all Turing machines which accepts languages over sigma we get to the fact that Ld is not recursively enumerable.

So this is our diagonalization proof that there exists recursively enumerable there exists languages which are not recursively enumerable.

(Refer Slide Time: 38:30)



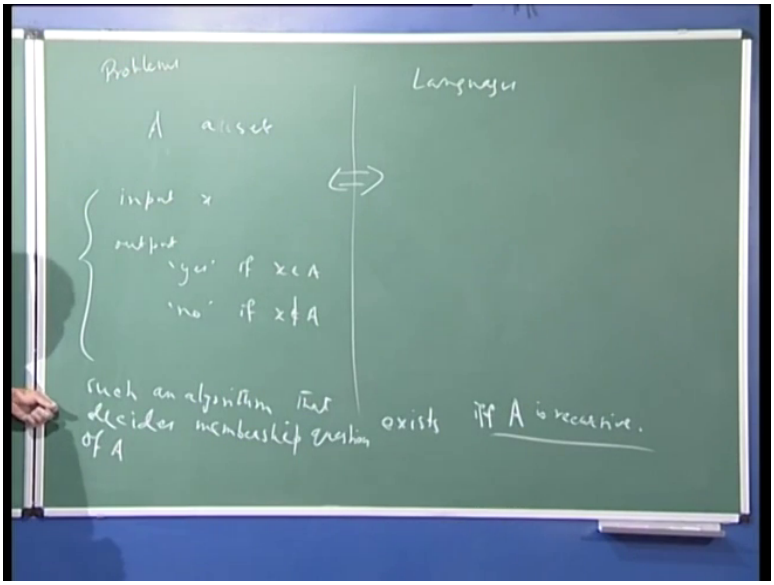
Let us now define a new class of languages and that class is called recursive and the definition, a language L is recursive if there is a Turing machine that 1, accepts L and 2, halts on every input. Now you contrast this definition with recursively enumerable language there we said a language is recursively enumerable if there is a Turing machine that just accepts that we did not insist on whether it halts on all I mean every input or not.

So if you contrast recursive language with a recursively enumerable language the difference is this that whereas for a recursive language you will always have a Turing machine which will halt on each input whether the input is in the language or not the Turing machine that accepts the language will always halt. Whereas in case of r.e if the language is re what all we said was that the there will be a Turing machine which will accept the language, of course acceptance means the Turing machine will halt on all input which are in the language but if the input is not in the language then we are we have said nothing so in fact the Turing machine can go into an infinite loop and that is one way of rejecting its input because it is not in the language.

Now why would we make this definition? Because you go back to our notion of the computational problem, what is a computational problem? Computational problem is something for which we require an algorithm to solve it for example, the computational problem would be given two numbers compute the (())(41:14) of course you cannot do it by means of a table so you have to have an algorithm compute the GCD of two numbers.

So a solution of a computational problem if it exists is an algorithm and a special class of computational problem is decision problem and a decision problem is one where you have to decide that means you have to just say yes or no, so we will have two output yes for some inputs and no for some inputs. Now a particular class of this decision problem will be membership decision problem we had talked about these things earlier but let us once more go over this idea that a membership decision problem essentially is there is some you have some set A in mind and you have the problem the computational problem of deciding given something whether that is an element of the set or not.

(Refer Slide Time: 42:42)



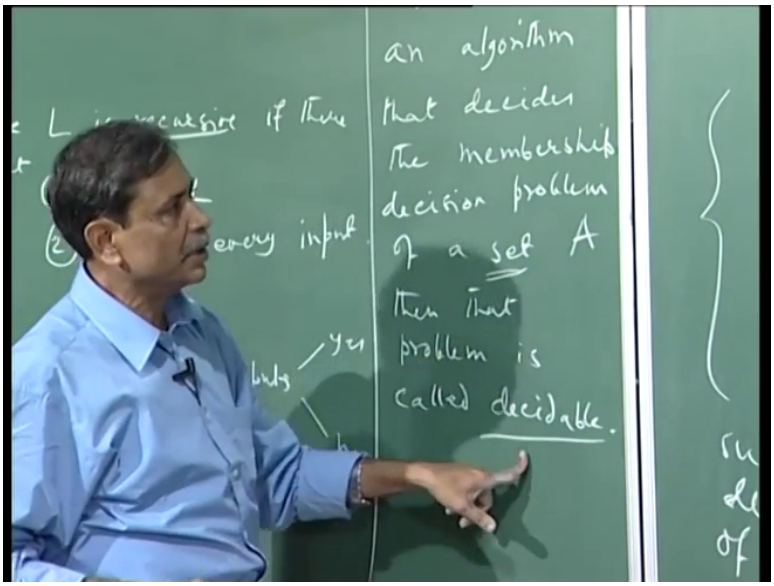
So your algorithm will be you know it will take some input and the output will be yes if x is in A and no if x is not in A of course we have many many examples of decision problems which are basically membership set membership decision problem for example you would like to decide primility of a number. So given a number you would like to decide whether it is prime or not so

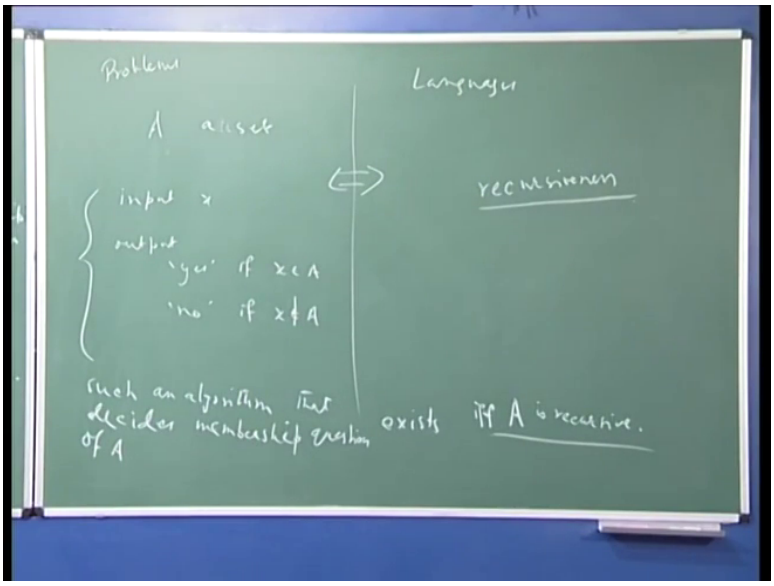
you would like to output yes if the number is prime so A is here is the set of all prime so you will say yes if x is in A, no if x is not in A that means it is not a prime.

So clearly you can do this so this is the decision problem for A membership decision problem for the set A to have such an algorithm and my point is that such an algorithm that decides membership question of A exists if and only if A is recursive, right? So what am I saying is that in this side I mean I put a line it is to say this is the domain of problem and this is the domain of languages and the correspondence that I want to emphasize is the existence of a decision algorithm like this corresponds to the recursiveness of some languages.

So in this case the membership decision problem for A is can be done by a program like this in algorithm like this if and only if A is recursive, right? So another term we have something is such a problem, right? Such a problem which admits an algorithm is called computable or decidable.

(Refer Slide Time: 46:01)



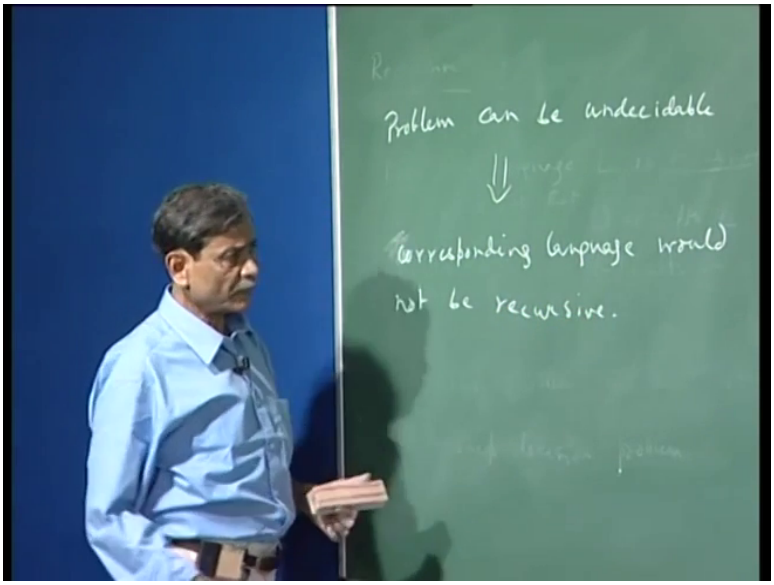


So once more what is the point I am making I am saying that if there is an algorithm that decides the membership decision problem of a set then that problem is called decidable.

So decidable is an adjective which we will use in the context of problem and that too decision problem you have an algorithm to solve a decision problem you will say that the problem is decidable but just now what we said is if the corresponding decision problem is decidable you know seeing this definition and what we have then it means the that set is recursive, right? So my point is the you can talk of decidability in one hand and really an equivalent question is recursiveness, okay.

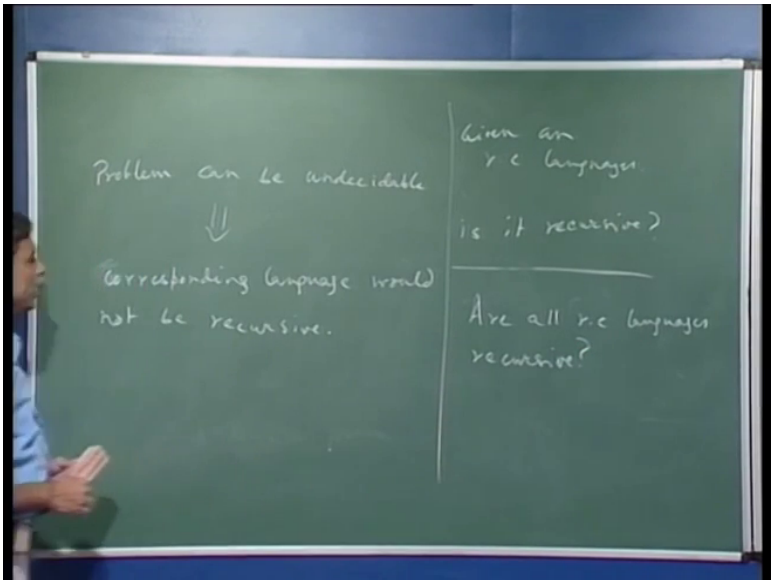
So the set let us say name as set A so we say the membership decision problem for the set A is decidable if and only if A is recursive. You should see that if you wanted an algorithm to decide something you know some set membership problem equivalently we would like to show that the set is recursive.

(Refer Slide Time: 48:52)



So what the notion decidability notion as I said is equivalent to or tantamount to recursiveness notion and if set membership problem is not decidable you call a problem is problem can be undecidable, alright? That would mean what? That corresponding language would be would not be recursive, okay.

(Refer Slide Time: 49:41)



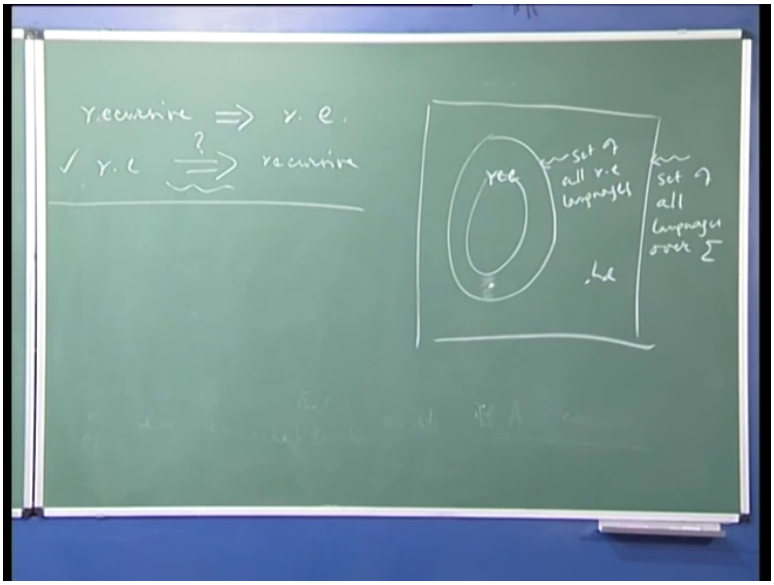
Now we have a notion of recursively enumerable languages so we had this notion of r.e languages if a language is r.e then of course you have a Turing machine to accept it the only problem with that Turing machine is that Turing machine may not halt on those inputs which are

not in that language but then it is not a that Turing machine can be cannot be used to decide the membership problem of the language because it would it may not halt to give out n1.

So therefore the question that is of important is that given an re language is it recursive, why is this question important? Because if I can show that an r.e language is recursive then for that language I have a membership decision algorithm, right? The problem the decision problem would be decided we will have a useful algorithm. See the point I am trying to say is that just contrast this case that consider primality, okay. Now suppose you have a Turing machine which accepts the set of all primes but the kind of question you would like to answer is somebody gives you a number n you would like to say yes if it is a prime, no if it is not a prime.

So that Turing machine which of course we will say yes to all primes when given as input but may not halt at all on some inputs which are composite so you cannot really use that Turing machine to make that practical decision that is a number prime. So this question you can frame it in another way are all re languages recursive clearly every recursive language is recursively enumerable because for a language to be recursive first of all it has to have a Turing machine to accept it and that immediately makes a recursively enumerable.

(Refer Slide Time: 52:29)



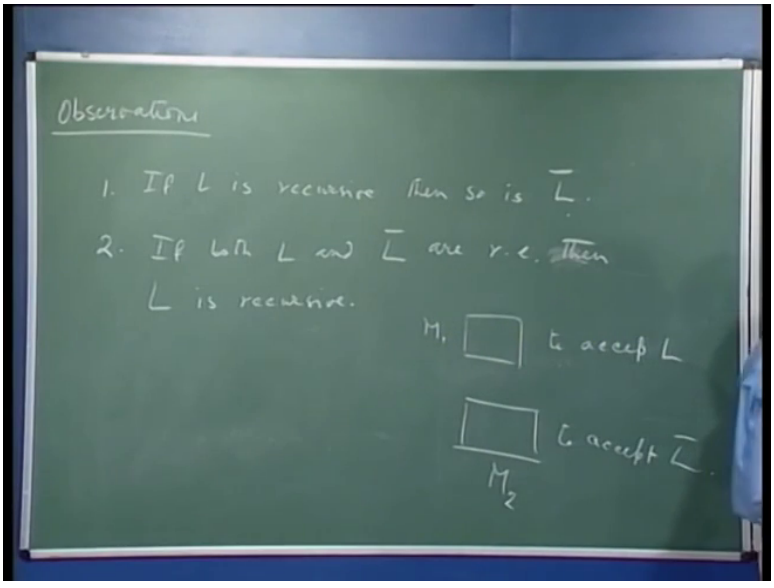
So our recursive implies recursively enumerable and the reverse question is therefore we are asking, if the answer to this question is yes then of course should be very useful whenever we have a language to accept some set we have an algorithm to decide that language. Now you can

just put these questions in a slightly pictorial form so just imagine this is your set of all languages over sigma, right? And this is the set of or class of all r.e languages, right?

So let me just mention this that we already know something which is here and not here, what is that? Ld the language Ld, right? We know Ld is a language over sigma but Ld is not recursively enumerable so it is not within this. And since every recursive language is also recursively enumerable the class of recursive language if I represent by this that class so that set is clearly a subset of this thing the question this really this question that we have put here is the separation is about the separation is the separation strict that is is there a language here or not and that is an important question.

So that means if I have something here although we have we can have an program which will always terminate on an the yes instances it will not be yet decidedable, okay. So in fact what we are going to show soon that indeed this separation is strict there will be something here which is in other words there will be a recursively enumerable language which is not recursive we will show that but before that let me quickly tell you the couple of very simple facts about you know recursive and recursively enumerable languages, first these are really observations of simple facts if first fact is if L is recursive then so is L compliment, right?

(Refer Slide Time: 56:00)

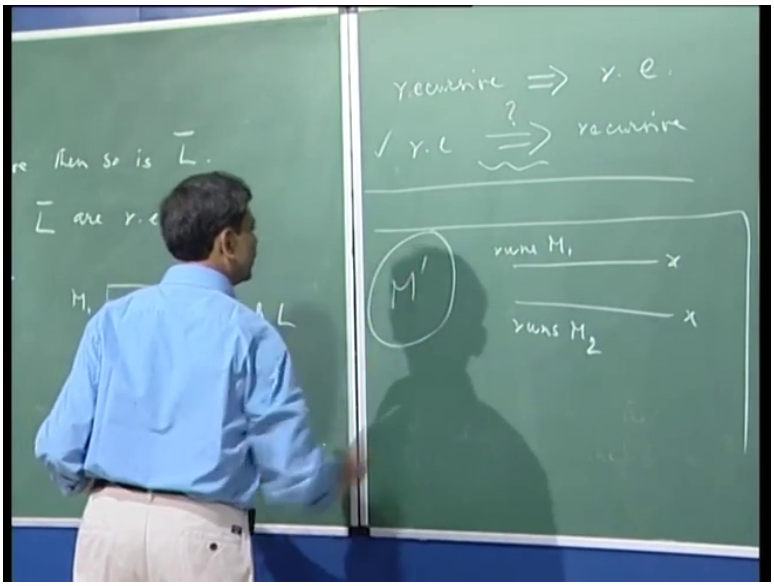


Clearly because this is something we did for DFA, right? That same kind of situation will apply that any input you know suppose L is recursive means you have a Turing machine which accepts

L and it always halt so if the input is in the language it will go to an accepting state and halt of course if it is not in the language it will go to a state in which it will halt but that state is not accepting.

So now if you just switch accepting and non-accepting states of that machine you will be recognize L compliment so that is clear. Second thing is interesting also that if both L and L compliment are recursively enumerable then L is recursive of course when I say L is recursive immediately we have L compliment is recursive. So if both L and L compliment are recursively enumerable then both of them are recursive, right? Now how do I show this so you see basically imagine that I have a Turing machine M1 to accept L and another machine M2 to accept L compliment, right?

(Refer Slide Time: 57:46)



Now think of as a machine M dash which on any input x you know it makes a copy of that input x on another tape and alternatively you know it runs M1 so runs M1 here and runs M2 here now alternately what I mean by that see basically M dash will simulate or you know behaves or let us say even number of steps like M1 and for odd number of steps like M2, so it is switching between the work of M1 and M2 and since M1 accepts L and M2 accepts L compliment one of these you know one of these machines will accept the input because if it is the input is in the language then M1 would accept if the language if the input is not in the language then M2 will accept.

So sooner or later one of these machines will halt in an accepting state and from there you know what is it if it was M2 which accepted it then x is not in the language L and if M1 which accepted it and halted then you know x is not in the language. So either case M compliment will or M bar this M dash this new machine would always halt on every input and therefore (()) (59:37) you using this machine you know whether the input is in the language L or not, right?

So in case you find this M prime fine it is M1 which has accepted it will say yes if M prime finds that it is M2 which was accepting the language L2 has accepted the input then it will say no the input does not belong to language L, thereby since we have a Turing machine M dash which would always halt after saying yes or no it would halt and that would accept L we have proved correctness of this statement that if both L and L compliment are recursively enumerable then L itself is recursive.