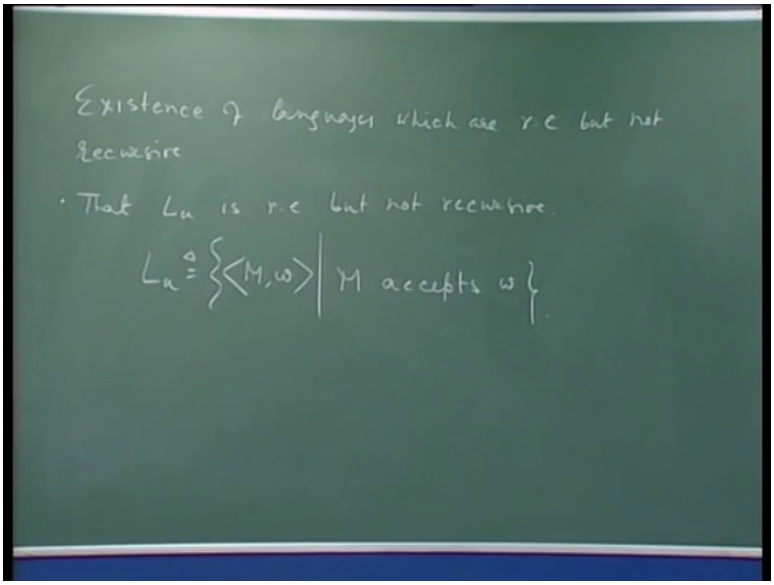
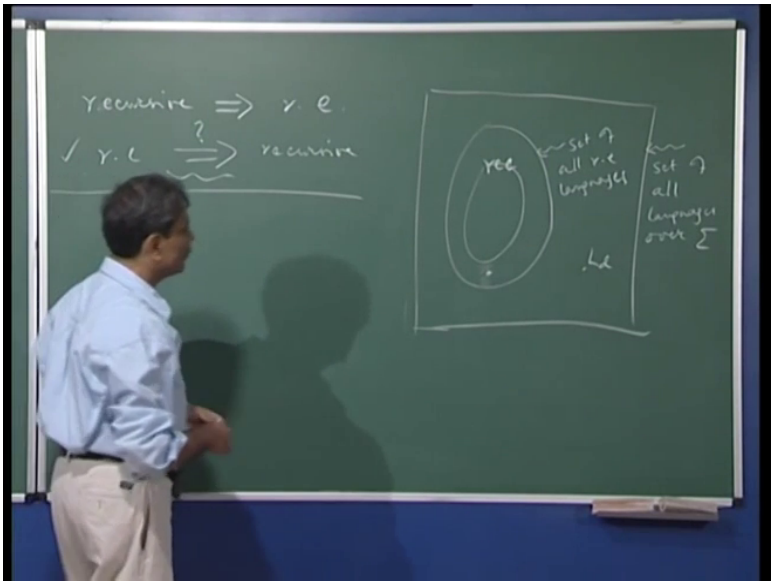
Course on Theory of Computation
By Professor Somenath Biswas
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
Lecture 42
Module 1
Separation of recursive and r.e. classes, halting problem and its undecidability.

(Refer Slide Time: 0:28)



What we shall prove now that there are languages which are recursively enumerable but not recursive we will try to prove existence of r.e or existence of languages which are r.e that is recursively enumerable but not recursive. In particular we will show that Lu is r.e but not recursive definition of the language Lu is if you recall is this that set of so this notation stands for that you are given as input a string which encodes a pair and first element of that pair is the code of some machine Turing machine M and the second one is a string w.

(Refer Slide Time: 2:08)



So we will show that this language to be recursively enumerable but not recursive and therefore if we go back to the picture that we drew last time that this square box if you imagine it to be the set of all languages over sigma subset of it is recursively enumerable languages and it is a proper subset because we had seen the diagonal language Ld is of course a language over sigma but it is not recursively enumerable.

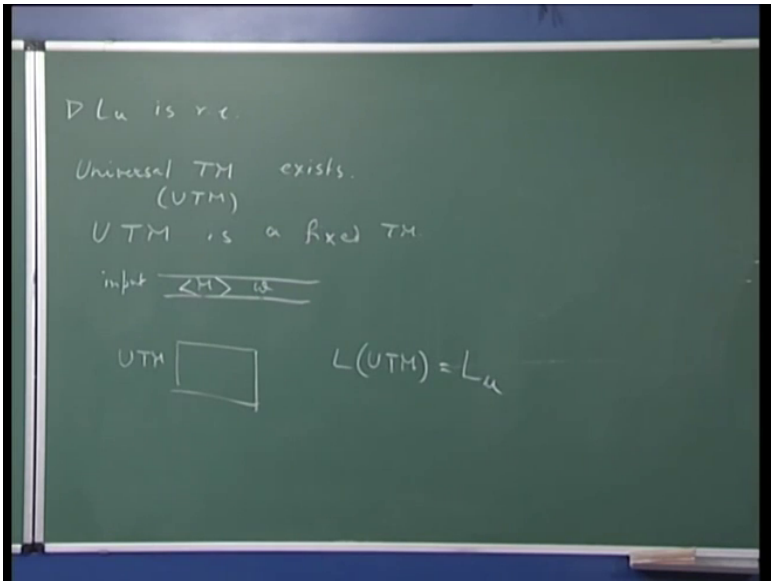
So we have a set of r.e languages and then by definition set of recursive languages will be a subset of r.e languages because it comes from definition because you see a recursive language is a language which is accepted by a Turing machine which halts on all inputs, the first condition that a language is accepted by all, excuse me, a language which is accepted by a Turing machine means it is a r.e language.

So recursive languages are by definition recursively enumerable languages so we drew a subset and the question is that whether this subset is a proper subset whether the set of recursive languages that set is a proper subset of the set of all recursively enumerable languages. When we prove this that Lu is recursively enumerable but not recursive we show the separation between these two sets recursive and recursively enumerable we show that the separation is strict and in fact Lu is such a language which is r.e but not recursive.

So you can see that we need to do two things to show that Lu is re that is point 1, the point 2 is Lu is not recursive and we will do it one by one, so let us first prove that Lu is r.e. The way we

prove this that Lu is r.e by demonstrating something which is very important but fairly simple that is the existence of universal Turing machines.

(Refer Slide Time: 4:43)



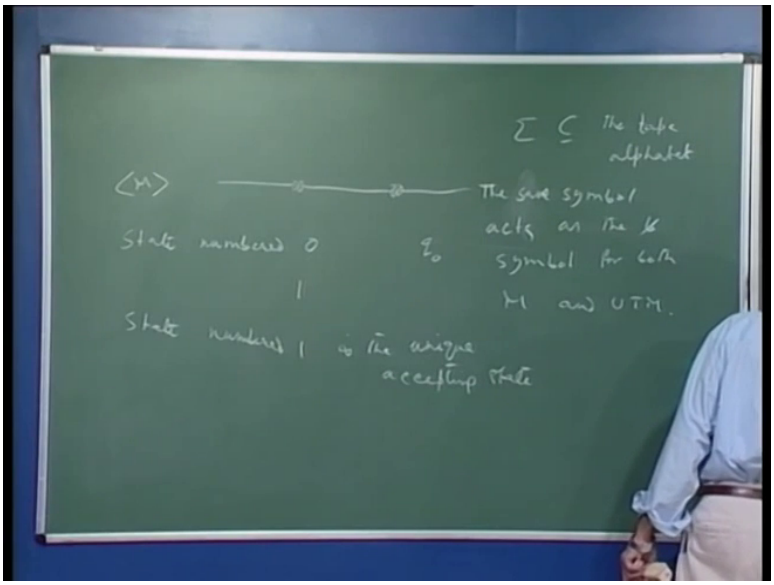
So right now we are trying to prove that Lu is r.e and for the proof of this statement we will first of all require that a particular kind of Turing machine which is called universal Turing machine such a Turing machine exists.

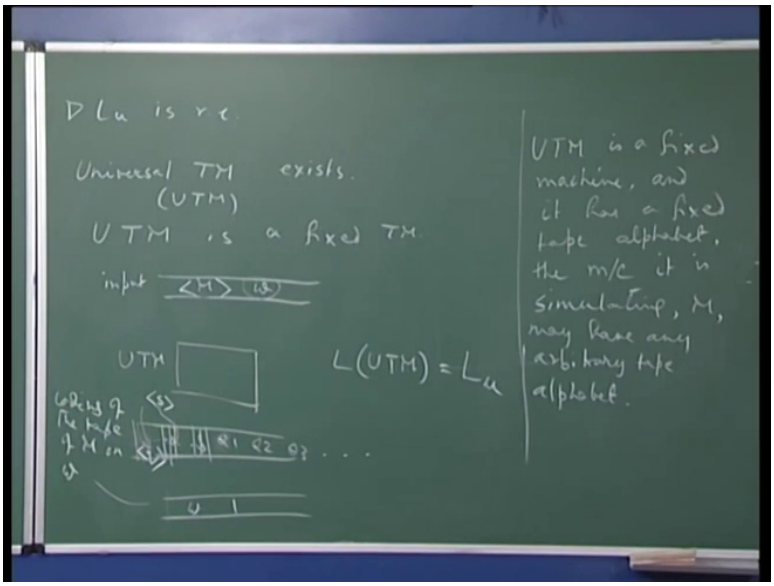
In fact the name Lu, Lu stands for this u stands for universal, now what is a universal machine? Universal machine is one fixed Turing machine which can simulate other Turing machines so far as language recognition is concerned for the purpose of language recognition that is the simulation should be such that that this fixed machine will be able to tell you whether a any given M accepts its own input or whatever input you have given to that machine M or not.

So essentially universal Turing machine so if I write it like this abbreviation UTM, so UTM is a fixed machine fixed TM and what such a TM can do is that such a TM so this is your UTM its on its input tape two things will be given, one is the code of a Turing machine and a particular string w and this UTM will decide or not decide really this what it will do is that it will simulate the working of M on w step by step and because it is simulating M on w step by step if ever M accepts w UTM will know that M would have accepted w and therefore it would accept its input.

Now its input was this pair M w so this UTM therefore the language accepted by UTM by the way we have described it is clearly Lu by now it should not be very surprising to you how why such a machine exists so let us see after all what we mean by we have given the code of a Turing machine M code we said that that is something we have done before code of a machine is in terms of its quintuples and.

(Refer Slide Time: 8:10)





So the code of M is really a string which is essentially the quintuples of that machine each quintuples is coded if you recall as a five tuple of numbers each number was represented in a

unary form all that we know this quintuples where separated by some double 0's this is quintuple 1, quintuple 2 etcetera, right.

And we have some further let us say convention that is the state numbered 1 or numbered 0 maybe we can start with 0 state numbered 0 basically what otherwise you would have called as q0 whose such a state numbered 0 that means it is the number is 0 and its representation in unary is 1 this state is the initial state and we also said the state numbered 1 is the unique accepting state, alright? Now so this UTM what will it have is maybe it will have a few other tapes and what one of these tapes will be is to carry the coding of the tape of M as it works on w, right?

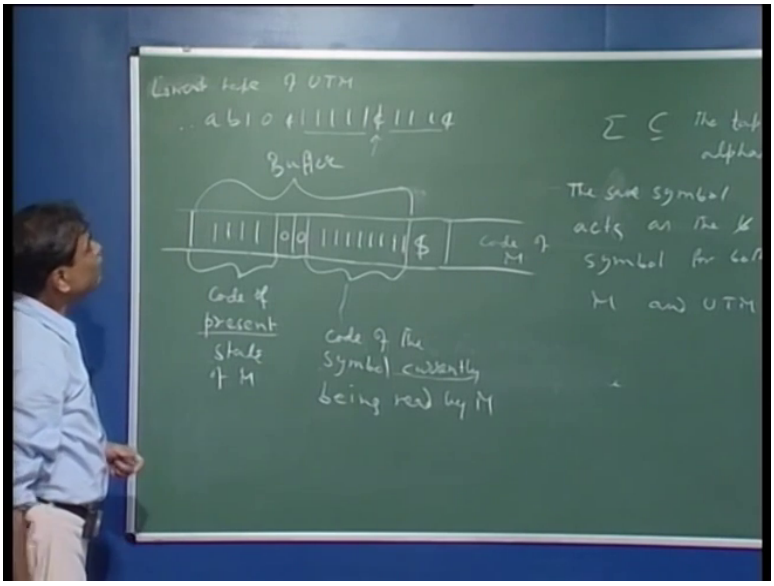
So imagine this last tape is the coding of the tape we are assuming M is a one tape machine and the code of the tact tape will be appearing here. Now why do we not directly work with a tape as M would have seen it or as M's tape. See the reason is UTM is a fixed machine and it has a fixed alphabet the problem basic problem that non matching of UTM tape alphabet with the alphabet of M by in the manner in which we coded M itself because you see even for the quintuples M we need to state the symbols that M uses so we do not really use the symbols directly then those symbols as such of no importance but if we what we code them by unary numbers and therefore since 0 and 1 we are assume to be in the tape alphabet of UTM we can at least express the coded version of every symbol of M for UTM to work on.

So this tape as I said this last tape is the coding of the tape of M on w, right? Without any loss of generality you may assume that sigma is that is the input alphabet of M is a subset of the tape alphabet, also we can assume that blank symbol, okay that is we say the blank symbol is same that is the same symbol let us say the same symbol acts as the blank symbol for both M and the machine UTM universal Turing machine.

So non blank portion initially will have w, right? So what UTM can write away do that it can copy the w in the beginning here and also it can copy let us say after some special symbol the code of M, code of M recall will be the quintuples of 1 quintuple 1 followed by quintuple 2, quintuple 3 and so on that is the code of the machine, right? And let us also say that this part so before this dollar it uses something the space here you can imagine that it will use this part will store the current symbol so let us say or the coding of the current symbol, okay so let me write it as s to say coding of the current symbol and this part which maybe a number of cells all these are

it will consist contain coding again in unary of the current state, okay and these two are separated by two 0's, alright?

(Refer Slide Time: 15:19)



So let me draw this part a little more clearly here for better readability what we are saying is our intent is this part will have code of present state and this part will have code of the symbol currently being read by M. So see we are saying present the word we are using present the we are using the word current so this is in the context that we imagine that M which you want to simulate is working on w and it is going from step to step at every step it has it is in some state that is the code that is the state present state of M so let us say code of present state of M and at that time in a particular beginning of a move of M, M is reading a particular symbol and x code appears here.

Then what we have said that there is this dollar and here we have code of M, right? We call as I said that code of M is nothing but quintuples one after another. In the beginning what is what will be in the beginning in the sense in the beginning of M works on w the state is going to be the initial state which we just said is the 0th state and that code of that state is just the number just the string I mean unary string 1 and here code of the current symbol so here whatever is the symbol in w the first symbol let us say a and as we said the tape you do not need to code sigma so it is we can say the a is there, right?

Though we will just have the convention we are just making one convention and that we can follow that in the in coding the quintuples if a symbol from sigma appears that we will write directly we would not code it in unary, alright? So this is there and otherwise the tape therefore the tape coded version of the tape of M as it works on w would consist of symbols from sigma which we will directly write and you know we can have some coding for symbols which of course they are not there in not in sigma those symbols will code then in unary and separate them by some special symbol, so let us say you know cent and then, okay.

And then here it we may have something like a, b, 1, you know maybe 0 which is okay. So this is how the coded or the last or the lowest tape of UTM lowest tape is this one tape of UTM will look like something like this of course there will be lots of blanks because blanks symbols are common for both. Now you see so what we said is initially suppose the symbol a is the first symbol of w that we can copy in this part which is the expanded version here and this we call the buffer region so buffer contains the code of the present state of M code of the symbol currently being read by M and now what would be the action of M when it is in this state and reading the this symbol that is define precisely by a quintuple whose first two components are these two, right? Separated by two zeros we also assumed that you know there is no confusion that these two zeros are separating the coding of the state and coding of the thing the symbol, alright.

So now at any given time so now imagine dynamically what is the situation how does UTM simulate the working of M1 w at any given time M would have been scanning a particular symbol and it will be in some particular state at that for that at that time the buffer would be filled in the appropriate manner and now UTM would like to know what would be the next state of M what is the symbol that will be written and what will be the direction of move.

So in general the UTM would this the head of this tape will be scanning the left most of the code of the current symbol being scanned. So as I said and then these are appropriately written imagine that that has happened. And now by scanning the code of M UTM would know what would have been the next state, right? So next state suppose is the third state, so therefore this has to be updated by the code of that state 3 which is of course the unary representation of 3 is four 1's so we write that.

And now it also know from the code of M (())(22:59) which is the next the symbol to be printed so let us say the symbol to be printed is the seventh symbol so that is 11111111 so the number 7 is of course represented in unary by eight 1's so this particular will come here. And now what UTM would do is to appropriately update two things here see somewhere that at this position the whatever was what was there previously is the contents will be the code of the symbol that was just read in its place this should come, so essentially it should copy this part here, now only problem is you know this may be of the length may mismatch, right? This maybe small the code of that symbol maybe small code of this new symbol to be written there is large.

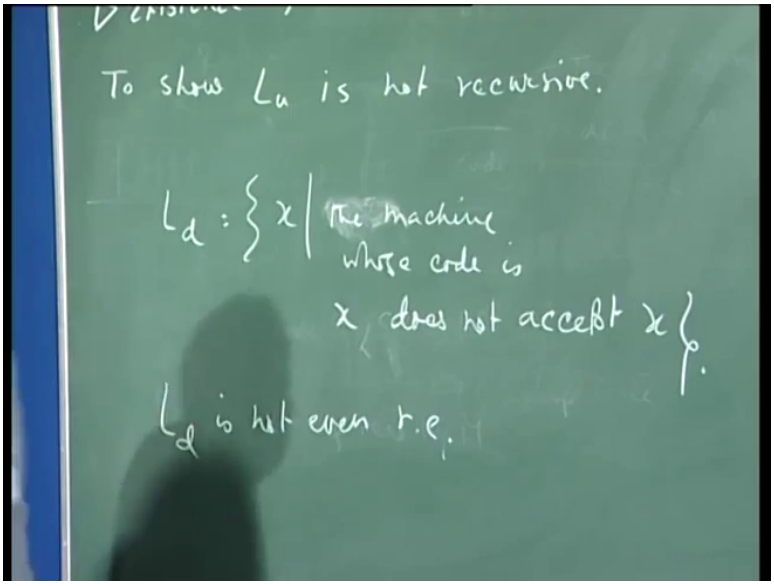
So it has to shift some symbols to accommodate all these, similarly it might have to push some symbols from right hand side to create a smaller space if the new symbol to be written there had a code whose length is less than the code of the symbol that was previously there. So this is correctly updated and again the appropriate quintuple would say that whether the head will move to the left or to the right. So if it is supposed to move to the left you know it will supposing it was here so it updated the correctly the symbol that it was reading previously and then it positions itself to the beginning of the this code of the symbol which M step would have appeared left to the present or the symbol, right?

So now it comes here and we are back in the beginning of the simulation of a another next step of M because correctly now we have the present state, present symbol and the head is at the right place and this will go on some details need to be worked out because of the way we said that you know in not all symbols are codes of coded but some symbols are directly written so in that case if it sees a symbol which is directly written and it will just move one step otherwise if it was to move left otherwise it will move back to the previous cent sign and then position itself appropriately.

So what I am trying to say that this is not difficult to see that every step of M on w can be simulated by the UTM with a number of steps. So therefore if ever M goes to an accepting state as it works on w UTM would now that because you see every time whenever the new state information comes here it can check whether it is the accepting state or not if the accepting state is reached then M would accepts or rather the UTM would accept its input UTM's input was recall M and w.

So what are inputs it is accepting? It is accepting those inputs such that M accepts w. So precisely it is accepting the language Lu and since we have a Turing machine to accept the language Lu that shows that the language Lu is recursively enumerable.

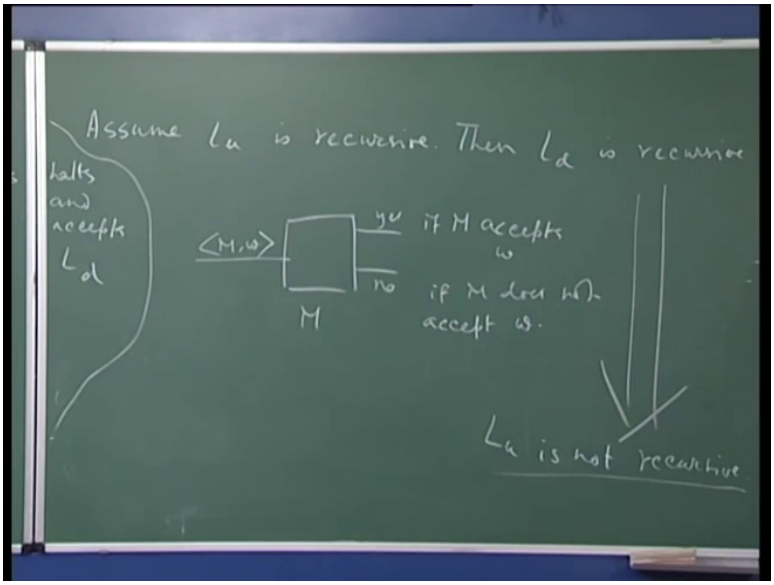
(Refer Slide Time: 27:30)

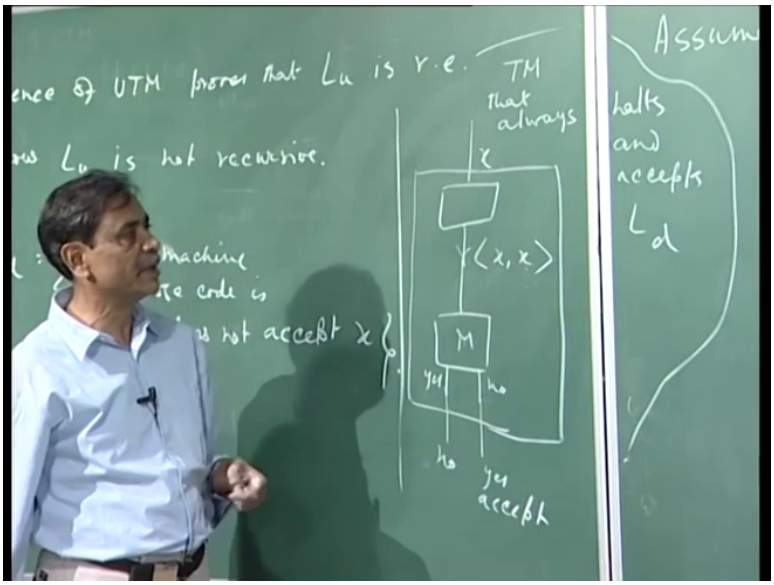


So we have shown existence of UTM proves that Lu is r.e. Is very clear that the only inputs UTM would accept UTM would accept its input only when an this in the during this simulation it finds the accepting the accepting state would have been reached by the machine which is being simulated otherwise it will just carry on with another step of simulation of this machine that is being simulated.

So since it accepts all and only inputs M w such that M accepts w it precisely accepts the language Lu. So first part of our job in proving existence of recursively enumerable languages which are not recursive is done because our candidate for that was Lu and we have proved that Lu is recursively enumerable. What is now left is to show that Lu is not recursive our next task is to show Lu is not recursive, now this is very simple because you see idea is if Lu was recursive then Ld would become recursive, why? Recall what was Ld? Ld was this language these are all binary strings the machine whose code is x and we had proved that Ld is not even r.e recursively enumerable, right?

(Refer Slide Time: 30:22)





So now imagine Lu is recursive assume Lu is recursive so then we would have a Turing machine M some Turing machine M which would say yes on an input like this M, w it will always halt and saying yes if M accepts w and no if M does not accept w, right? If Lu is recursive then such a Turing machine would be there, alright? Because Lu is recursive means Lu would be accepted by a Turing machine which always halts and therefore if it halts in a non-accepting thing state you know that M does not accept w if it halts in an accepting state then M accepts w.

Now what you can do therefore imagine something like this that suppose you had some x now x from here you have very simple transformer which codes x, x so from x it creates a pair x and x,

right? But now you can think of this x is a code of a machine and this x is a input to that machine and now here you have M and if M would have said yes that M would have accepted that means the machine whose code is x accepts x and here no means machine whose code is x does not accept x, right? And here your final output if M would have not accepted then this algorithm would accept and if M would have accepted this algorithm does not accept.

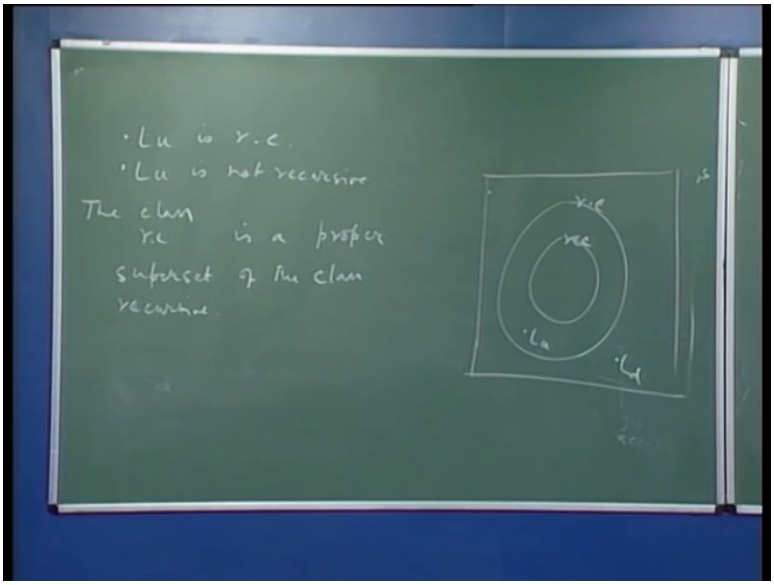
So what I am saying is imagine a new algorithm or a Turing machine because this is very simple you just imagine a Turing machine which takes one string as input creates a pair out of it then invokes the machine which always halts and accepts Lu and depending on whether that machine is accepting or rejecting the input if it is it would have accepted the input, which input? M's input is x, x then this composite machine is going to reject its original input x otherwise it is no.

Is very clear what is this composite machine doing, right? This composite machine would be accepting, see it is accepting here so yes by that we mean accept it accepts if M would have rejected, M would have rejected means the machine whose code is x accepts x, right? I mean does not accept x, right? Only that is the reason M would have rejected this M rejects its input x, x if and only if machine whose code is x does not accept x that means the string x is in the language Ld and otherwise the string x is not in the language Ld.

So this whole thing is a machine Turing machine that always halts and accepts Ld, right? So this Turing machine always halts you see because this obviously can be done by a machine this transformation can be done fairly simply and trivially and that there is no problem there is no question of non-halting in doing this transformation and this by definition always halts and depending on the answer you are just giving a certain answer which is switching the two answers.

So in that case this is a machine which always halts and it precisely accepts the language Ld. So that means what? That if such a machine exists in other words if Lu is recursive then Ld is recursive but we know Ld is not recursive because it is not even recursively enumerable so from here the conclusion is Lu is not recursive.

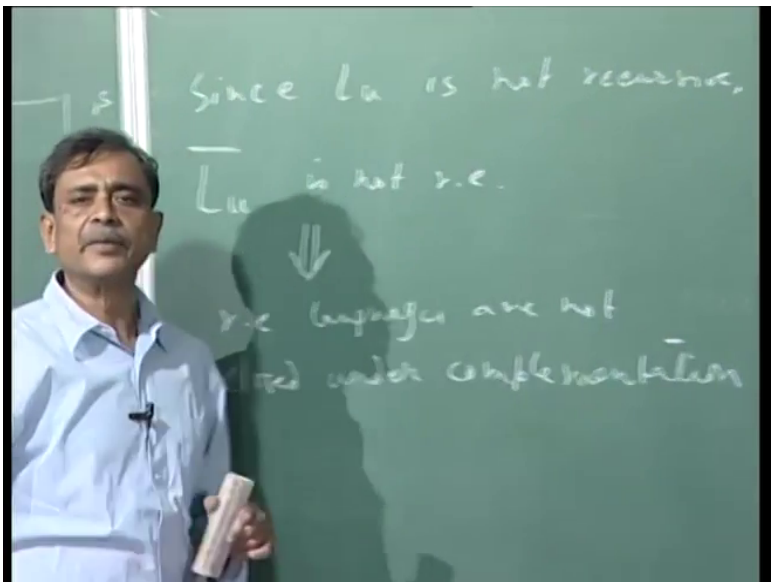
(Refer Slide Time: 36:31)



So we have shown both these so these two things we have proved that 1, Lu is r.e and second thing we just now proved that Lu is not recursive, so therefore the class r.e we can say like this the class r.e is a proper super set of the class recursive, okay.

So indeed the old picture that we drew the two circles remember that we said that this is the class of all languages this is the r.e languages, this is the recursive languages, right? So here is Lu, here is Ld and so therefore recursive is a proper subset of r.e.

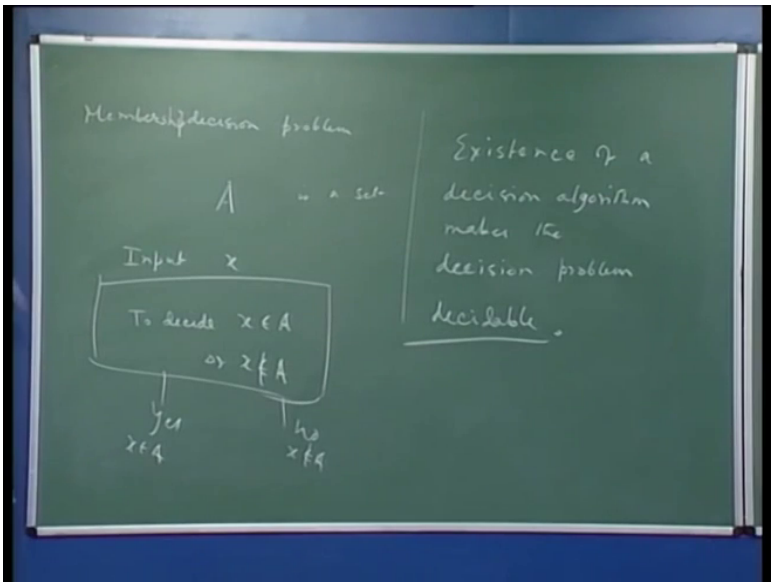
(Refer Slide Time: 38:03)



Now what about the compliment of r.e? Compliment of Lu clearly since Lu is not recursive but r.e Lu compliment is not r.e because if Lu of course we know Lu is recursively enumerable if Lu compliment is also recursively enumerable then Lu would have been recursive but we know Lu is not recursive so Lu compliment is not recursively enumerable.

So you see I mean you have these cases that both Ld I mean take this language Lu its compliment is not recursively enumerable it comes here so this also shows this simple observation also shows that r.e languages are not closed under complementation so this is a property of r.e languages whereas of course recursive languages are closed under complementation.

(Refer Slide Time: 39:31)



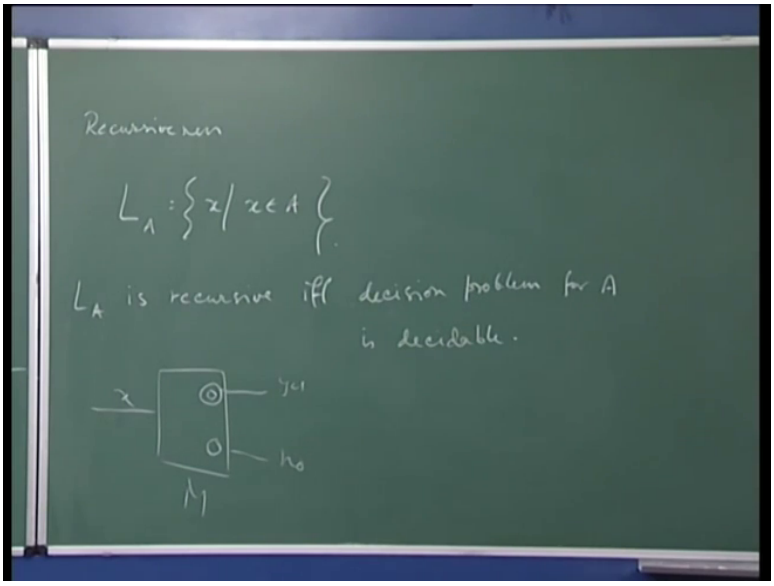
We will first of all very briefly review something which we have talked about earlier notion of membership decision problem a membership decision problem is a membership decision problem, so membership of what? Membership of some set A, A is set and the kind of problem you have is that you are are given an input which is x and to decide what you need to decide is whether x is in A or x is not in A, right?

So in case x is in A you will say yes x is in A and no you will say no if x is not in A, right? So if we have an algorithm to do this correctly then we say that the membership decision problem of A is decidable and this is called a decision algorithm existence of a decision algorithm makes the

corresponding problem decision problem membership decision problem decidable. So what does it mean to say a decision problem is undecidable that means no such algorithm exists, right?

Now also there is a correspondence between this notion a problem decision problem membership decision problem being decidable and the notion of recursiveness.

(Refer Slide Time: 42:12)



Now consider this set or language point is and is fairly simple to see La is recursive if and only if decision problem I am not writing membership decision because in the context it is clear membership decision problem for A is decidable, let us go through the argument quickly suppose La is recursive so there is a Turing machine which always halts and accepts only the string which are in A this set A. So now we can create an algorithm for the for solving the decision problem is that given the input x you essentially run that Turing machine M if the Turing machine goes to an accepting state you say yes otherwise you say no, right? Since M precisely M always halts so either it halts in an accepting state or in an you know state which is not accepting. So in case it halts in an accepting state you know that x is in A this condition is satisfied by the input and therefore answer of the decision is yes otherwise it is no.

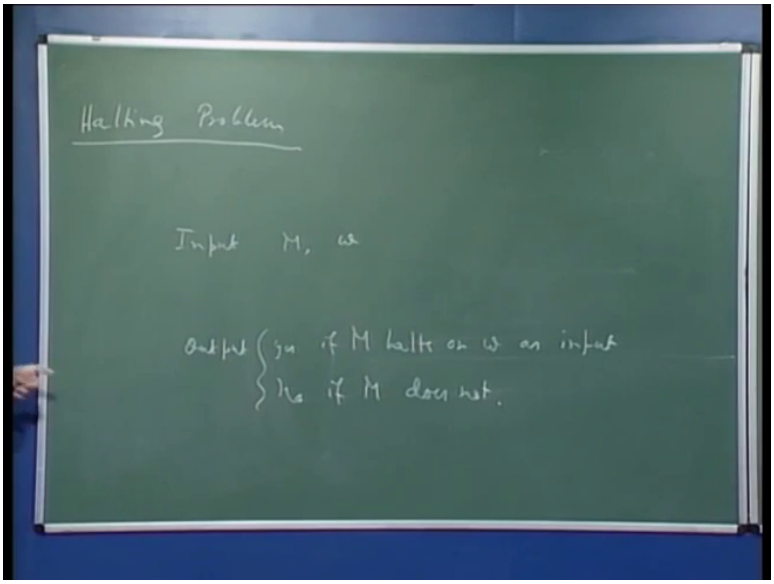
And similarly on the other hand if I have an algorithm for the decision problem for A then we have said if something can be done by any algorithm by Church Turing thesis it can be done by a Turing machine. So there is a Turing machine to decide whether an input x is in A or not I am

just turn it into a recognizer that same Turing machine by going to an accepting state if you know the input x is accepted by our input x is in the set A.

So essentially from an algorithm by invoking Church Turing hypothesis we claim the existence of a Turing machine which solves the decision problem and from a solution of decision problem by a Turing machine we get a recognizer for a language La, so this is clear. So some decision problem is undecidable, right? So corresponding to every decision problem we can create the set of yes instances as we did here and that is a language and if the decision problem is undecidable that means the corresponding language of yes instances is not recursive, alright?

So what I am trying to say is that undecidabilty proves essentially can be couched in the language of languages show the corresponding language to be not recursive but one particular problem of undecidability we would like to prove directly that it is undecidable because that problem is so well known and all of you might have I mean I am sure most of you would have heard of this problem and that is called the halting problem.

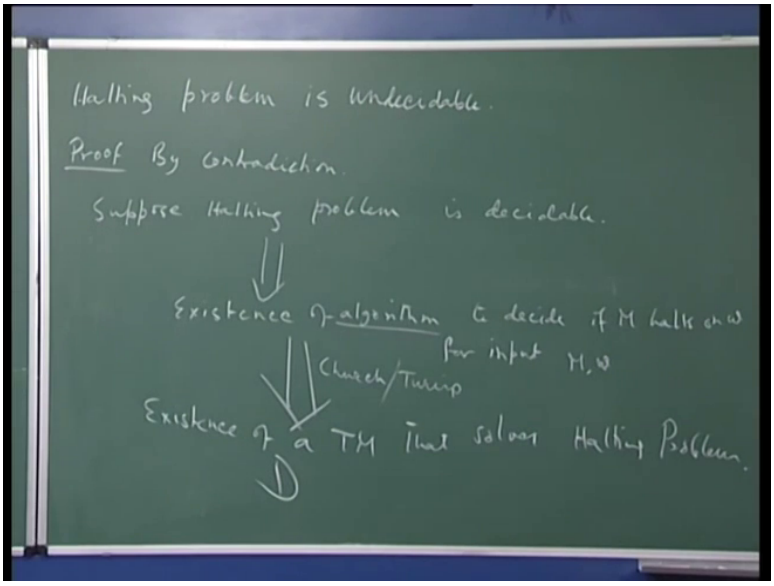
(Refer Slide Time: 46:21)



So a halting problem is not really a language problem but here we what we want though it is about Turing machines the problem is like this you can state the problem this way that as input you will be given code of a Turing machine and some string let us say w what you are supposed to decide output yes if M halts on w as input and output no if M does not. So essentially we are

looking for an algorithm which will decide given any Turing machine and some string which is it is considered as its input whether the Turing machine would have halted on that string or not.

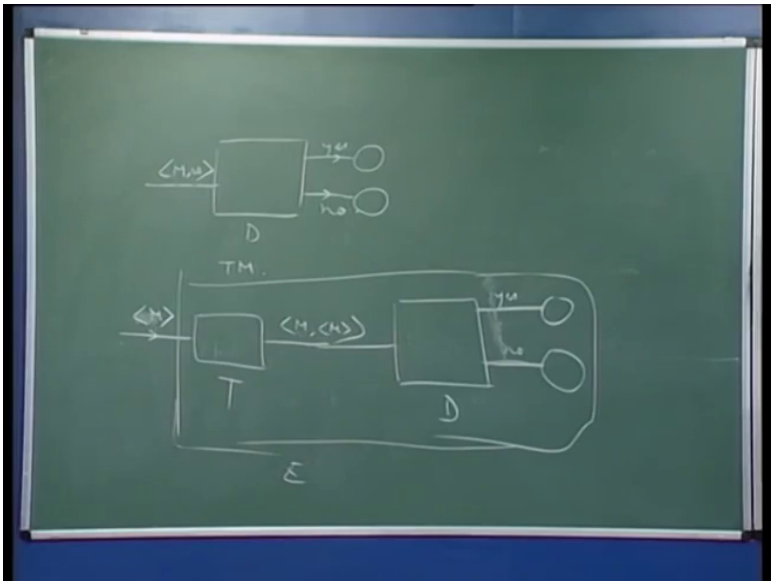
(Refer Slide Time: 47:58)



Now this is a classical problem and it is known that halting problem is undecidable. Halting problem is undecidable and we will try to prove this kind of directly and see how it is done the proof is by contradiction, okay. So we will assume we will start by saying that suppose halting problem is indeed decidable that means what there exist some algorithm which decides given M and w as input whether M halts on w or not.

Now existence of such an algorithm immediately means through Church Turing so this means let us say first of all existence of an algorithm to decide if M halts on w for input any input M, w. Now here we will invoke Church Turing hypothesis to say that this algorithm can be carried out by a Turing machine. So existence of a particular Turing machine that solves the halting problem let us name this Turing machine let us name this Turing machine D, alright?

(Refer Slide Time: 50:46)



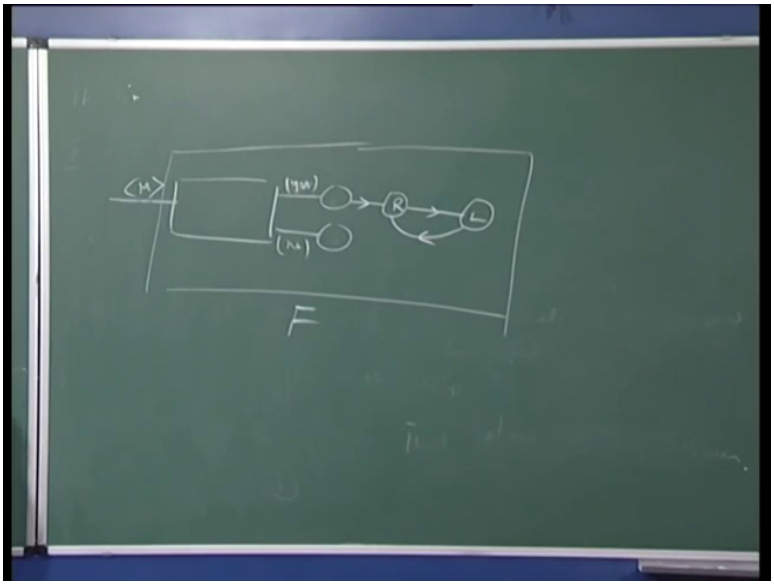
Now pictorially let us see what D is something very simple, I mean whatever we said this D it will take as input something which is like M, w and it will decide so remember this is a Turing machine. Let us also say at this time it says yes means the Turing machine knows that M would accept or M would halt on w and no means it says that M would not halt on w. So let us also say that by saying yes it goes to a this Turing machine goes to a state from where there is no transition and so therefore D itself halts there, alright?

So these two are since no transitions are shown these two are halting states. Now since D exists imagine another machine which something like this that given any string w, right? Or think of this way that given any string which is of course we know any binary string can be a code of a Turing machines so imagine that given any code of a Turing machine it first of all copies this M and creates a pair so basically by that what we mean we essentially we have two copies of codes of M, right? And now this machine D is there and D as before would look like this yes and it halts, no does not and no and again it halts.

So this is some transformer which just copies M to that output line with another copy. So basically now we can see what is happening when would such a for what kinds of Ms this composite machine will go will say yes this composite will say yes if and only if the machine M halts on its own description, right? The machine M halts on its own description that it take it to this line, okay.

So this let us call it the machine E, alright? And now let me slightly change E to obtain a new machine F.

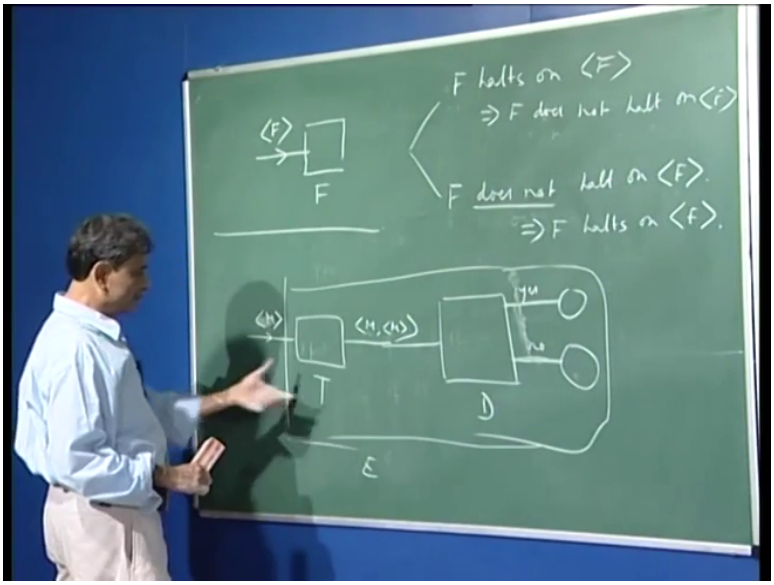
(Refer Slide Time: 54:10)



So E basically was that it will take code of a machine and it will go to this state if M would halt on its own input halt on its own code as input and it will go to this state if M would not have halted on its own description as input. So this is where old thing would have said yes this is no but instead of writing yes and no so let me put this in brackets because now we no longer we are not interested in this kind of output as such that here we go into an infinite loop on any symbol.

So essentially once it comes to this state then it goes to this state and keeps moving here but this one is as before, okay. So this is a different machine which we obtain from E and call this F the machine F. What happens now if two F you had given the input which is the code of F.

(Refer Slide Time: 55:52)



So basically this is F consider the situation where the input is the code of the machine F itself, alright? This is the situation we are considering. So in now there are two things which are possible that F halts on its own description, okay this is case 1, case 2 is F does not halt on its own description which is F.

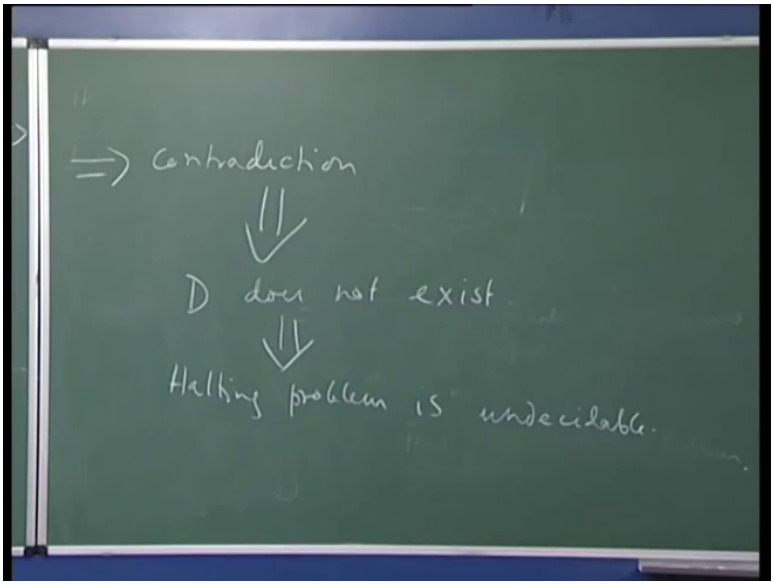
So let us take the first case F halts on F then this machine E, so basically here what that would have done if this was given F was given as input? It should have gone to this state, right? Now F is very similar to E except so therefore F also on input code F will come here but once it comes here it goes into an infinite loop, right? Then assuming F halts on F we get that F does not halt on and if we assume let us therefore may consider the other case does not halt on F.

Now if F does not halt on F E would have come to this state this line and come to this state, as we said E and F the only difference is here so F also then on given this as input would have come here but in this case when it comes to this state it halts. So if we assume F does not halt on F that implies from there we are getting F halts on F, so what is this situation? This situation is that if F exists we are getting into a contradiction because either it halts on its own description or it does not halt on its own description, in either case we are getting something ridiculous because if you assume this then its opposite is true if you assume the opposite then that would have been true.

So this is a classical way you say that we have reached a contradiction, what is it that we what is it that this entire thing contradicts, some assumption that we made and what is the assumption

really that we made? Here we said F exists but clearly now this F cannot exist if F cannot exist E cannot exists, if E cannot exist of course T does, right? T just copies something make a copy to it only reason E cannot exist is because D cannot exist, and what was D? D was a Turing machine to solve the halting problem.

(Refer Slide Time: 59:34)



So therefore we see a contradiction and which implies that D does not exist and what we are saying is by this we mean that since D does not exist there is no Turing machine to solve the halting problem and since there is no Turing machine to solve the halting problem there is no algorithm to solve the halting problem, therefore halting problem is undecidable.

You should spend a couple of minutes on this proof and contrast it with some of the earlier things that we discussed you should see it also as a diagonalization proof and you should also realize that we could have come to the same conclusion using notion of recursiveness etcetera but this is a more direct demonstration of a famous problem being undecidable. One final remark some of you might wonder that the reason we said we got the contradiction was because we tried this kind of stunt that is we gave a machine its own description and that took us to a contradiction.

So is it the case that the halting problem is undecidable because we choose to gave such a funny input but point is if we do not give such inputs, inputs which are codes of (())(61:46) codes of the machine or codes of the algorithm in that case otherwise can we do everything, so this is a question you can think of and answer to this is no I mean this is not as simple as that that the

halting problem seen as a function is not computable only at one point, right? So you can think about that and this is you know classical problem and we must understand all then once sense of it in a very very manner formal as well you know grasp the intuition behind this proves.