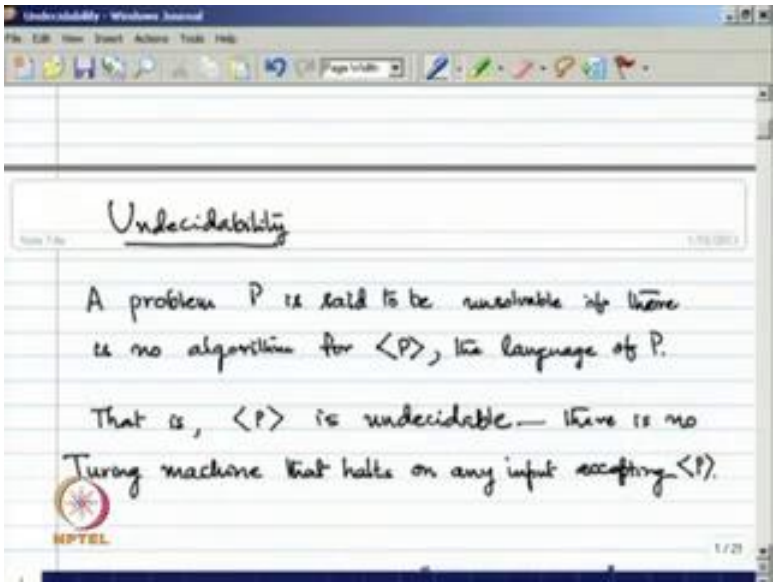
Formal Languages and Automata Theory
Prof. Diganta Goswami
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module - 13
Decidability and Undecidability
Lecture - 2
Undecidability Part I

So, in this lecture we discuss Undecidability, we are curious to know that given any problem, whether the problem can be solved or cannot be solved at all. So, if the problem is solved we say that it is decidable, otherwise it is not decidable; that means undecidable. So, we have already seen that there are some problems, which are decidable and is was shown by giving Turing machines, which holds on every input to decide a corresponding language; that means it just decide a Turing machines to accept the corresponding language. So, for every problem we have the corresponding language and if that language can be decided by a Turing machine, there we say that the problem is solvable or decidable, so today we will see undecidability.

(Refer Slide Time: 01:39)

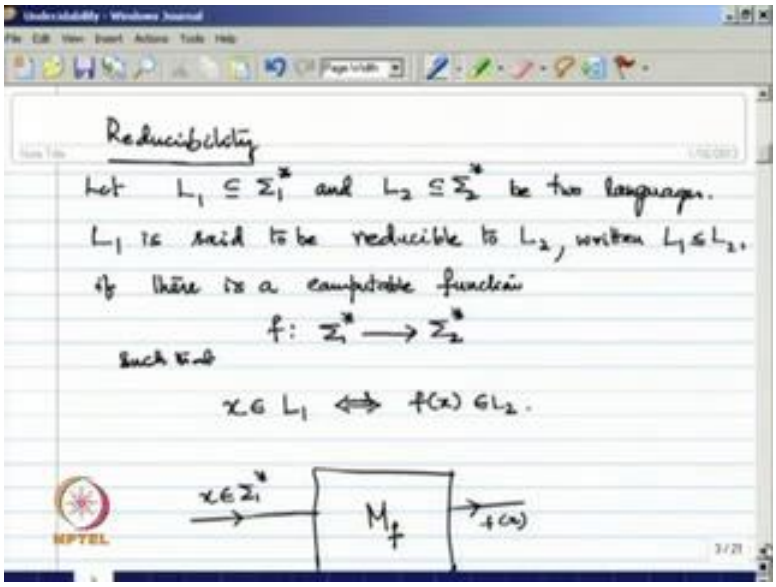


Now, a problem P is said to be unsolvable, if there is no algorithm for P, that means the language of P, that is the language of P is undecidable a corresponding language for problem P, which we and denote like this in angular bracket is undecidable. That means, there is no Turing machine that halts on any input accepting P that is what you say. Now

showing that a problem is decidable, we can construct a Turing machine there is a decider to decide that language.

But, to show that a problem is undecidable, we should show that there is no turing machine which can decide a corresponding language. So, constructing a Turing machine, which it decider for a given language may be easy in some cases, but showing that no such Turing machine exist for a given language is not that easy. Therefore, given a problem to show that it is a undecidable it may not be very easy to show that the problem is undecidable. Now, there another way to show that suppose that we have a problem or we have shown it to be undecidable, then there is a way to prove that, some other problems on decidable by using a tool by which basically tense less from one problem to another.

(Refer Slide Time: 03:42)



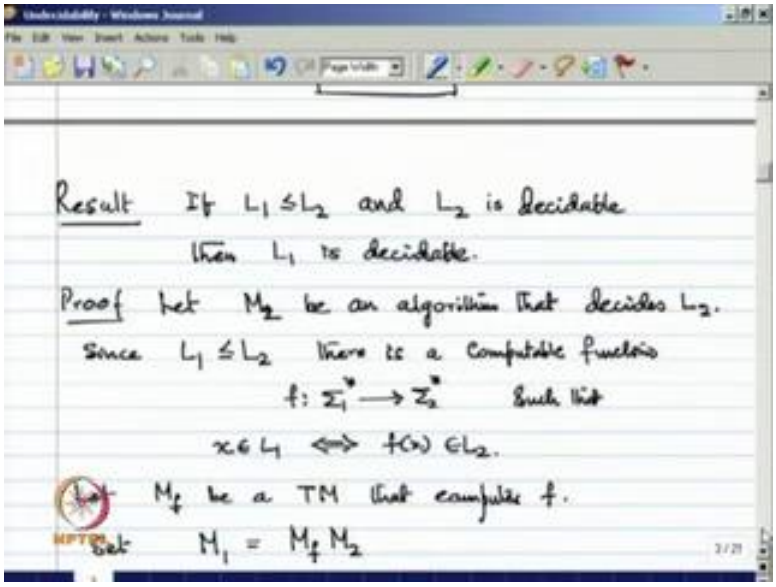
And there is only we said we reducibility, now reducibility is used to prove that if one problem is undecidable, then some other problem is also undecidable, similarly if some problem is decidable, then some other problem is also decidable. So, let us define reducibility say L 1 is a language and L 2 is a language, so L 1 is subset of sigma 1 star 2 is a subset of sigma 2 star.

So, the L 1 and L 2 are two languages, we say that L 1 is reducible to L 2 and written as L 1 reduces to L 2, we use this less than equal to sign to indicate that. So, we say that L 1 is reducible to L 2, if there is a computable function f from sigma 1 star to sigma 2 star

such that, for any string x any string x belongs to L 1, if and only if f x belongs to L 2. That means, since we say that there is functions are computable, that means there is a Turing machine say M f, that takes any string x that belong to sigma 1 star as input.

And translates it by means of f, it terminates f of x to a string that belongs to L 2, so if x belongs to L 1, then f x belongs to L 2, similarly if x does not belongs to L 1 then f x does not belong to L 2. So, that is what the computable function does, so if satisfies that, then we say that L 1 is reducible to L 2 via this function computable function f.

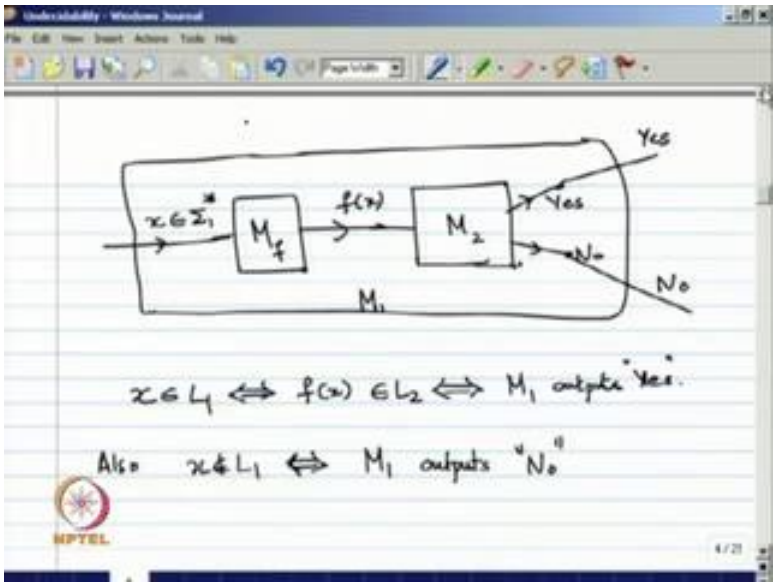
(Refer Slide Time: 05:48)



Now, once we have this from this we can show that, suppose L 1 is reducible to L 2 and L 2 is decidable and we can show that L 1 is also decidable. So, you can prove it by this, since L 2 is decidable there must be an algorithm say M 2 that decides L 2 this is our assumption. Now, L 1 reduces to L 2, since L 1 reduces to L 2 there is a computable function say f such that, x belongs to L 1 if and only if f x belongs to L 2.

Say M f is a Turing machine that computes f as we have given earlier, there must be a Turing machine to compute this function f, say M f is a function. Now, will have M 1 the Turing machine, which is composition of M f and algorithm M 2, so M 2 is a Turing machine is a algorithm decider, similarly the M f is Turing machine which computes the function f. So, M f composition M 2, if the algorithm M 1, now we show that M 1 is basically an algorithm for L 1, that means L 1 also decidable that is what we have to show; so how to show that M 1 is an algorithm that decides L 1.

(Refer Slide Time: 07:31)



Now, we have M 2 it take some input and answers yes or no, if answers yes if that input belongs to L 1 2, if it does not belong to L 1 2 it says no and M f computes the function f it takes an input x and it outputs f x. So, if x belongs to L 1, then f x belongs to L 2, if x does not belong to L 1, f x does not belong to L 2; now we have this composition of these two Turing machines, just combine these two Turing machine M f and M 2 to have M 1.

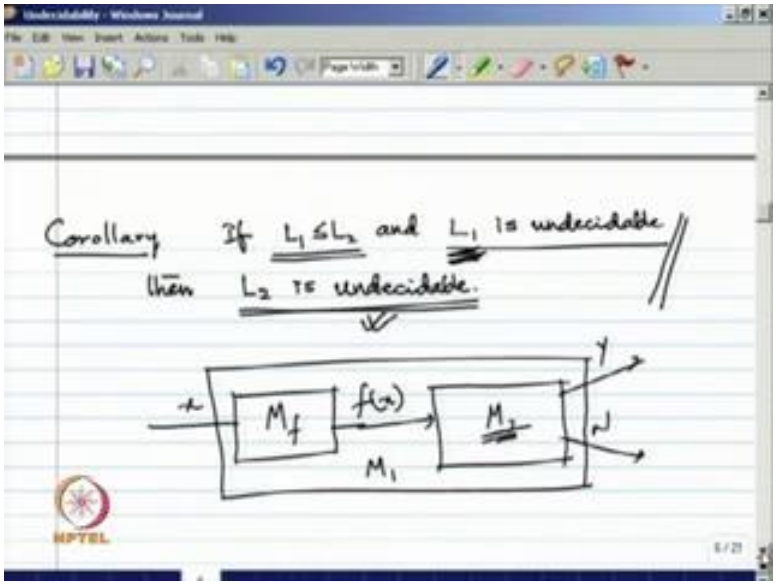
So, this M 1 takes any input x belong to sigma 1 star, so here we you is the Turing machine M f to compute f x and gives f x input to M 2, now since M 2 will always say whether yes or no. So, whenever M 2 says yes, M 1 also says the yes, whenever M 2 says no, M 1 also says no, so therefore given any string x belong to sigma 1 star M 2 always will say yes or no.

Therefore, M 1 also says yes or no and whenever it says yes, so if x belongs to L 1, then f x belongs to L 2 and then, M 1 outputs always yes, because since f x belongs to L 2, M 2 always says yes. So, therefore, M 1 also outputs yes, because whenever x belongs to L 1 M 1 always outputs yes, suppose x does not belong to L 1 in such a case f x also does not belong to M 2.

Since f x does not belong to L 2, M 2 says that no whenever M 2 says no, M 1 also says no, therefore M 1 outputs no, so therefore whenever x belongs to L 1, M 1 outputs yes and whenever x does not belong to L 1, M 1 outputs no. Therefore, clearly M 1 is an

algorithm and M 1 decides L 1, so therefore L 1 is a decidable language, assuming that L 2 is decidable, so from this clearly we can say that now is a corollary.

(Refer Slide Time: 10:18)



If L 1 reduces to L 2 and L 1 is undecidable, then, so is L 2, L 2 must also be undecidable, so you can use the same concept. Suppose, we can use it, I mean you can prove it by contradiction, suppose that L 2 is decidable then we have M 2 for L 2 and always say yes or no on some input and then, we can use the Turing machine M f, which computes f.

So, given a string x we first give it as input to M f which outputs f x, so if x belongs to L 1, then f x belongs to L 2, if x does not belong to L 1, then f x does not belongs to L 2. Now, f x we use as a input to M 2 and whenever M 2 says yes, we say that M 1 also outputs yes and if M 2 says no, then we say that M 1 also outputs no. Now, in such a case assuming that M 2 is decidable, we now have an algorithm for L 1, if M 2 is decidable, then so is L 1. But, since L 1 is undecidable already we know that, there no such Turing machine M 2 which is decided for L 2 can exist, so therefore L 2 must also be undecidable and so this is simple corollary from the previous one.

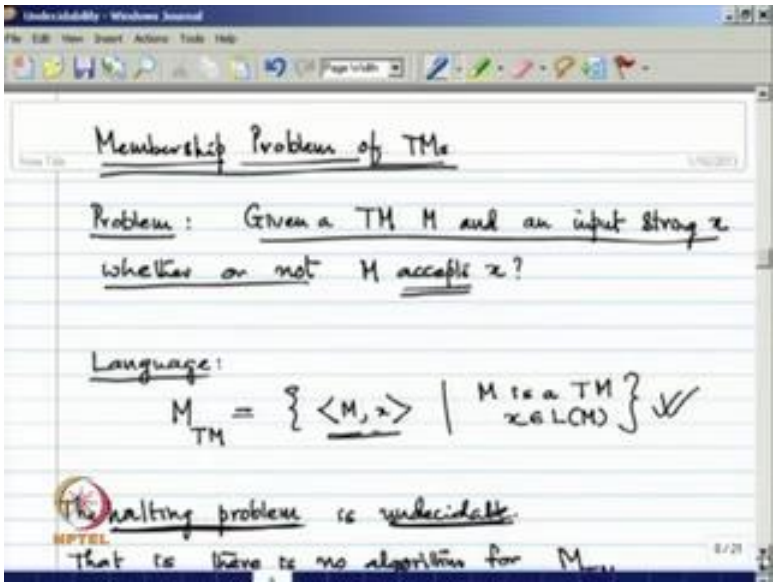
(Refer Slide Time: 12:44)

Now, you can take it as a exercise to show that, if L 1 is reducible to L 2 and L 2 is reducible to L 3, then L 1 is reducible to L 3, that means it satisfies the transitivity property. Now, what you can show is that, to show that a language is a language L is undecidable, we reduce an already known undecidable language to L, that is how we can use this technique reducibility.

That means, you can apply this reduction from L 1 to L 2 to show that L 2 undecidable, whenever if we already given that L 1 is undecidable, so L 1 is already known undecidable problem, we reduce L 1 to L 2 to show that L 2 is also undecidable. So, that is how we can use this technique the reducibility to show that some problems are undecidable, further we can use this transitivity property to show that some more problems are undecidable.

So, more undecidable languages can you shown to be undecidable by using this transitivity property through reduction. Now, to prove that a problem is undecidable by using reduction, we must have some already existing undecidable problem, so will first see one such problem which is undecidable.

(Refer Slide Time: 14:47)



Let us consider the problem of membership for Turing machines, already we have seen the membership problem for regular languages, for contractive languages, that means given any string x and a regular language L, whether not x is a member of L is the membership problem for regular language. Similarly, you can define the membership problem for contractive language, both these two problems I have shown to be decidable in the previous lecture.

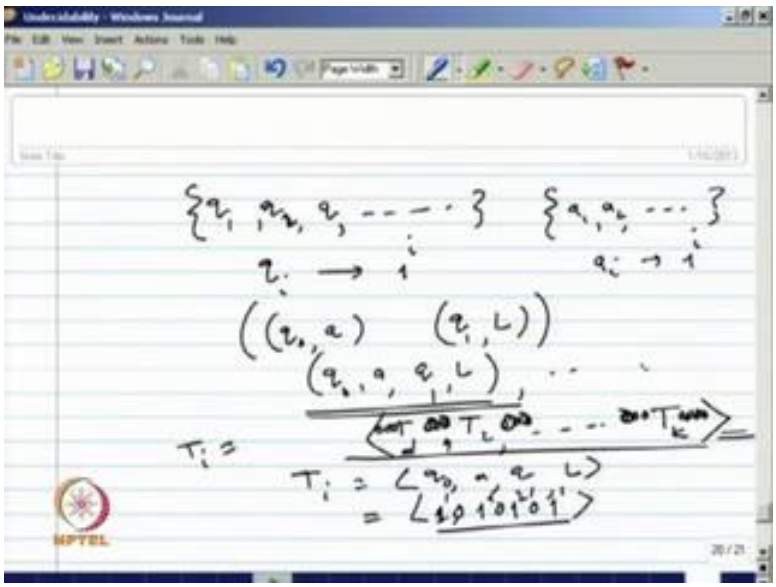
We know that we can construct an algorithm, that means a Turing machine is a decider, to decide a corresponding languages for those membership problems. But, if we now consider the membership problems of Turing machines, we can show that this membership problem of Turing machine is undecidable. So, the problem is given a Turing machine M and an input string x whether or not M accepts x, so in this case M accepts x, that means it halts on every input x.

So, whenever x belongs to L M, it says that it is yes, otherwise it says no, a corresponding language is that M TM, it is a encoding of all those strings M x where m is Turing machine and x is string and x belongs to L M. So, this is a corresponding language for the membership problem of Turing machines, we also say that this is A Halting problem for Turing machine, because the Turing machine has to halt on each and every input. We want to show that this halting problem is undecidable, that is there is no algorithm for M TM.

(Refer Slide Time: 16:46)

Now, whenever we say that this M x within angular bracket, that means this is a encoding for the Turing machine M and x, so we have already seen how to encode a DFA. Similarly, we can also encode any given Turing machine, because the Turing machine we have a finite number of states, a finite set of symbols is a alphabet, the way we encode it the states and the symbols in case of DFA can be used this case also.

(Refer Slide Time: 17:45)



That means, if say q 1, q 2, q 3 these are all set of states for Turing machine, then we can use a sequence of strings that is for q i, we use 1 to the power i, that means a sequence of

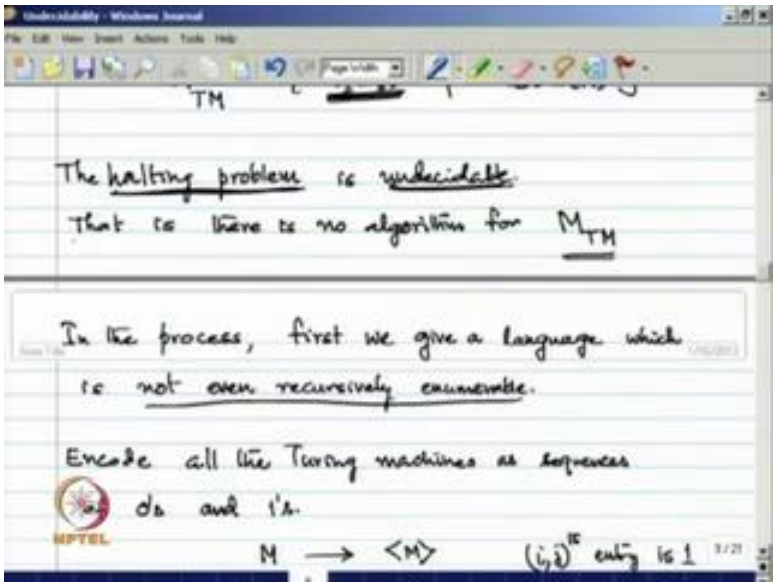
1 i ones we encode the state q i. Similarly, if you have a 1, a 2 these are all symbols the alphabet, then a i will be represented as 1 to the power i sequence of i ones and we know that the transition function for Turing machines can be represent by a quadruple.

That means since we know that is say q 0 some symbol a from a i, it can go to say some state q 1 at a L or R or whatever, so you can write it as q 0, 0, q 1, L, so this is the simply say q 0, a, q 1, L. So, by this quadruple we can represent a corresponding transition, so for Turing machine we have a sequence of such transitions, so as register mission 1, register mission 2 and so on. So, all these transitions now can be say k transitions encoded by using a sequence of 0's and 1's.

So, once we have sequence of 0's and 1's for T 1 the way we did in case of DFA, so T 1 and T 2 can be separated by say three 0's, T 2 and T 3 can be separated by three 0's, we can represent this lay bracket by three 0's and so on; eventually we have three 0's at the end of this. So, in any transition say T i the symbols say q 0, a, q 1, L whatever we have in q 1, the encoding for each can be separated say T i is q 0, a, q 1, L. So, what do you have q 0, we say 1 power 1, a maybe say 1 power 1, say 1 power 1 and 1 power 1 will be separated by symbol 0.

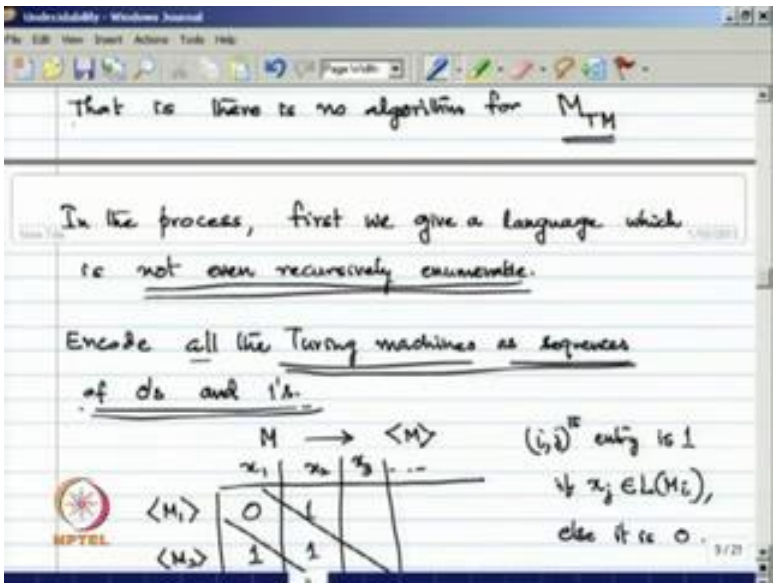
And similarly it will be separated by symbol 0, then that codes for q 1, it may be 1 power 2 single 0 say, L May be 1 power 1 and whatever, so therefore this may be code for say T i and so on. So, it is two transitions separate by three 0's or we can say two 0's as similar to DFA and then, when say when it ends it will have a sequence of three 0's. So, we can use same approach where we used in case of say DFA to encode a Turing machine and at the end of this after this we can give a corresponding sequence for the input string x. So, we have a way to represent or encode this thing M x, so the problem is when they show is that this is undecidable.

(Refer Slide Time: 21:50)



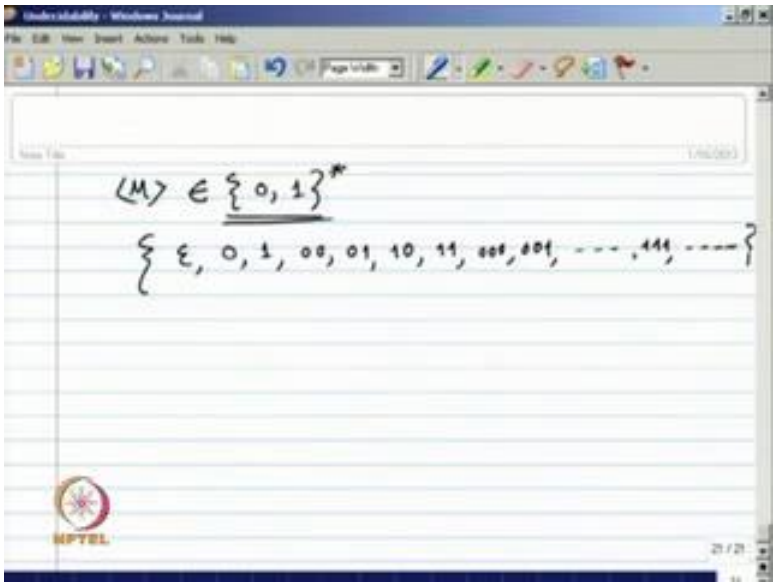
Now, to show that this is undecidable, that means there is no algorithm for M TM, we first give a language which is not even recursively enumerable. We should showing that there is a like this any Turing machine for this language M TM, which is decided a 1 to first to show that, there exist a language which is not even recursively enumerable, that means there is no Turing machine that recognizes that language. Then we come back to this our original problem of showing that M TM undecidable that means, there is no decider for M TM.

(Refer Slide Time: 22:26)



Now, to show that there is a language, which not even recursively enumerable we first encode all the Turing machines as sequence of 0's and 1's, so there will be infinite number of Turing machines. So, you can encode all the Turing machines, so since Turing machines are sequence of 0's and 1's, so the code of a Turing machine will be a member of sigma star, where sigma is equal to 0 and 1; now these sequences of Turing machines can be ordered using some ordering.

(Refer Slide Time: 23:20)

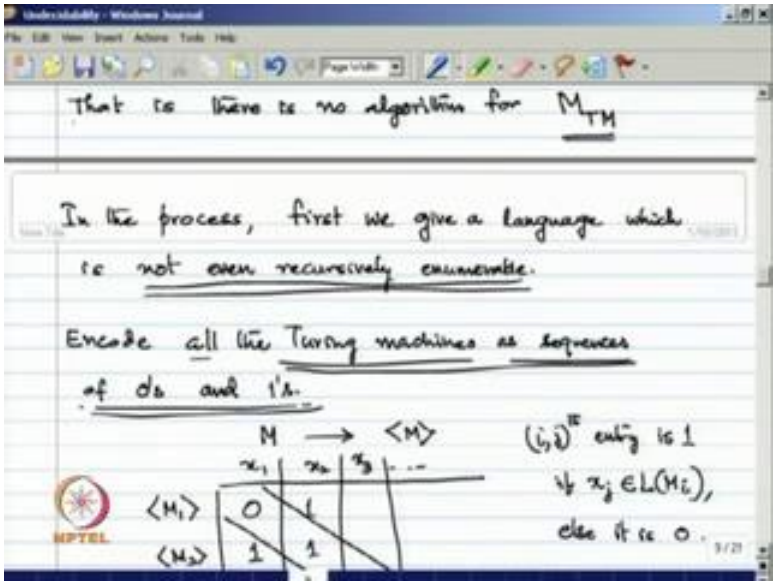


For example, suppose if the alphabet is 0 and 1, since we have Turing machine codes, which is sequence of 0's and 1's. So, any Turing machine code will belong to that sigma 1 star, so any Turing machine code will belong to the sequence of sigma star, where sigma is 0, 1 contains two symbols. Now, we can have an ordering of all the strings over 0 and 1's, for example say it is alphabetic ordering considering that or you can say canonical ordering.

So, where strings are ordered according to the alphabetical ordering, but the strings of shortest length will come first, that means in this case the ordering will be say initially will be epsilon further the string has length 0, next string of length one there only two possible cases 0 and 1. The string of length two we have four such strings and those four such strings will come in alphabetical order, that means 0 0, 0 1, 1 0, 1 1 and 1 1, next we have strings of length three again in how you order say 0 0 0 0 0 1 and so on.

Finally, say 1 1 1 and then, strings of length four and so on, so therefore we consider say this is in canonical ordering, so all the Turing machine codes can be ordered using this kind of canonical ordering. So, Turing machine sequences which we order in canonical order, so Turing machine M will be encoded by this M in angular bracket and since we order this some canonical order.

(Refer Slide Time: 25:05)



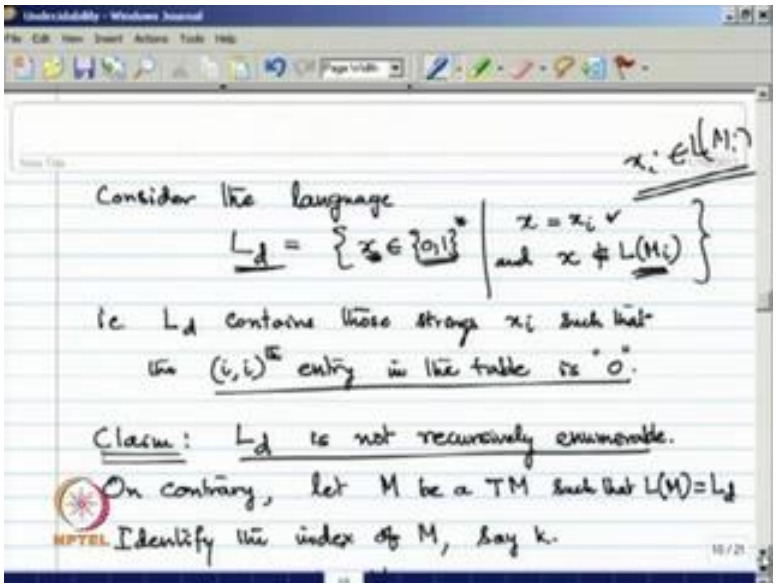
So, you put in the ordering say M 1 comes first, next comes M 2 and so on, similarly all the input strings x 1, x 2, x 3 will be ordered using some in the same canonical ordering and you put here in the sequence. So, accordingly we get a table arranging x in the column and M in the rows and in this table we see the entries we play of the entries, we play of the entries in such a way that the I just entry is 1, if x a belongs to L of M I, since we are talking about i j'th entry.

So, row i we have the corresponding machine Turing machine M i and column j, we have the string x j and now if x j is accepted by the Turing machine M i will just fill up that entry by 1, otherwise if x j does not belong to l of M i, then we the corresponding entry will be 0. So, that is how we fill up all the entries, so in this case M 1 accepts does not accepts x 1, so therefore this is 0 say M 1 accepts x 2 that is why this is 1, suppose M 1 accepts x 3 will be 1, suppose that M 1 does not accept x 4, so it will be 0 and so on.

Say M 2 x accepts x 1, so it is 1, M 2 accepts x 2, so it is 1 suppose M 2 does not accept x 3 that be 0 and so on, so we fill up all the entries of this table according to this rule. So,

please note that we have a sequence where we put all the Turing machine codes M 1 to M n like that, similarly where the sequence where you put we have ordering where you put all the strings in the sequence.

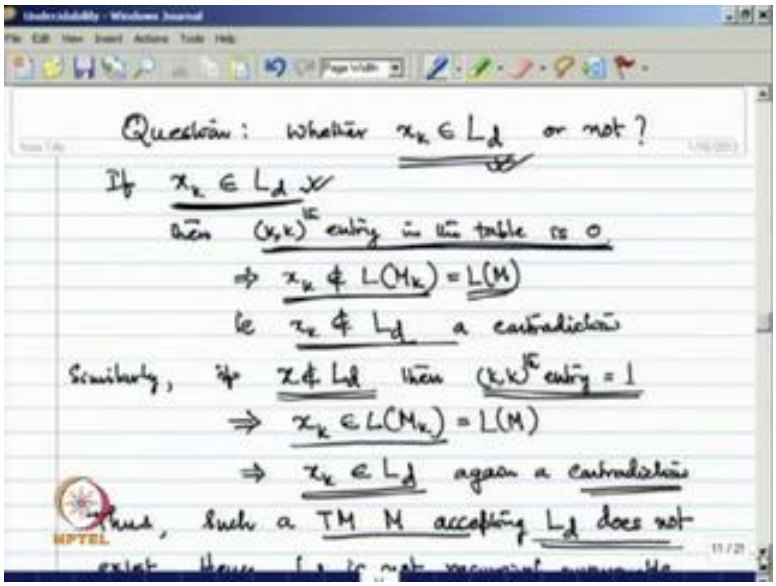
(Refer Slide Time: 27:45)



Now, we consider language say that language is named as L d, so LL dis set of all strings over 0 1 star, all the string x over 0 1 star such that, x is x i for some index i and x does not belong to L of M i. That means, all those strings all those strings x i such that, x i does not belongs to L of M i, that means it contains the entries from the diagonal, because we are talking about x i does not belong to L of M i, so if consider the entries from the diagonals.

So, if x i does not belong to L of M i, then this is a member the language L dthat is why it is said to diagonal language once you have defined the language L d, that means L d contains those strings x i such that, i i'th entry in the table is 0. Because, we have used this rule to show this is 0, if x i does not belongs to L of M i and we have talking about the diagonal elements, now we want to show that this language L dis not recursively enumerable, so this is our claim. So, language L dis not recursively enumerable, that means there is no Turing machine that recognizes this language L d.

(Refer Slide Time: 29:38)



Now, how to show is we assume that, just say Turing machine M that recognizes L dthat means, say M a Turing machine that recognize L d, L M equal to L d, so this is for contradiction, assuming that there exist a Turing machine M such that, L dequal to L M, we arrive at a confliction. So, if M is a Turing machine, then M must appearing in the ordering that we have already said say M 1, M 2, these are all Turing machine codes and so on.

So, this M must also appears some more in the ordering say this is M k for some k; that means, we identify the index of M say k such that M equal to M k, so this m must appear in the ordering, now once we have found out this index k for which M equal to M k. We ask the question why there are x k the corresponding string x k for that k belongs to L d or not whether x j, x k is a member of L d, we ask this question.

Suppose, x k belongs to L d it may belong to L d, in such a case according to a definition the k k'th entry in the table must be 0 which how we have defined if k k'th entry is 0 that is how we have defined this L d. If k k'th entry is in the table is 0 what it says the way we fill up the table, it says that x k does not belong to L of M k, because that is how we defined the rule over here, if it belongs to L of M i then, this is one otherwise it is 0.

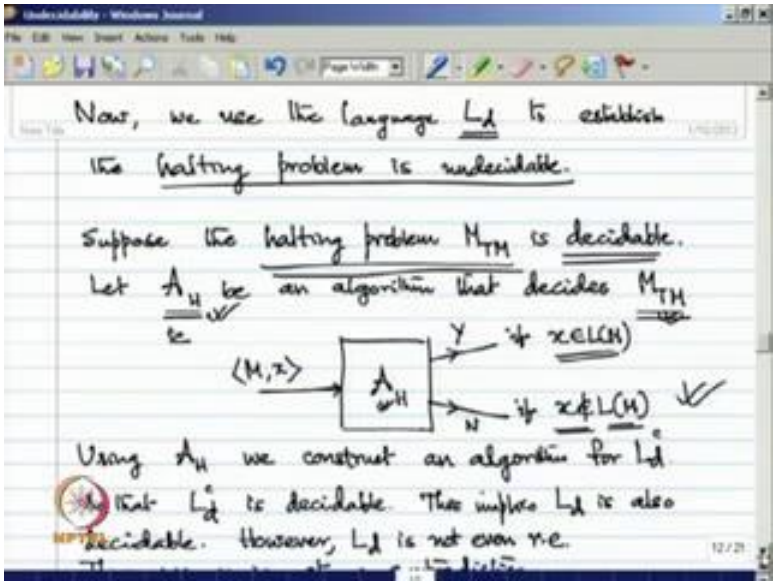
So, if x k belongs to L d, k k'th entry in the table is 0, it says that x k does not belong to L of M k; that means, x k does not belong to L of M k for because M is the M k therefore, it says that x k does not belong to L d according to our definition for l d. But,

this is a contradiction because we say that x k belongs to L d assuming that x k belongs to L d, we have a conversion there x k does not belong to L d.

Similarly, if we assume that x does not belong to L d, where again I have a confliction because if x does not belong to L d k k'th entry must be equal to one according to our definition of language L d, if that is the case if k k'th entry equal to 1. Then, according to our rule that we used to fill up the table elements we know that x k does x k belongs to L of M k; that means, x belongs to L of M, but according to again the definition of L d, we know that x k now belongs to L d again confliction.

So, in both cases whether x belongs to L d or x k belongs to L d or x k does not belong to L d, a arrived a confliction otherwise therefore, such a machine Turing machine M, recognizing L d cannot be there that is not exist, such a Turing machine M accepting L d or recognize a L d does not exist. So, this must be the case; that means, of such a Turing machine to recognize M must be wrong therefore, L d is not recursively enumerable since there is no Turing machine recognizing L d, L d must be not recursively enumerable.

(Refer Slide Time: 33:52)



Now, we use this language L d because is not recursive enumerable is always shown to establish that the halting problem is undecidable, now I have again come back to the our original problem of showing that halting problem is undecidable for that will be using this diagonal language L d. Now, we assume that for contradiction the halting problem is

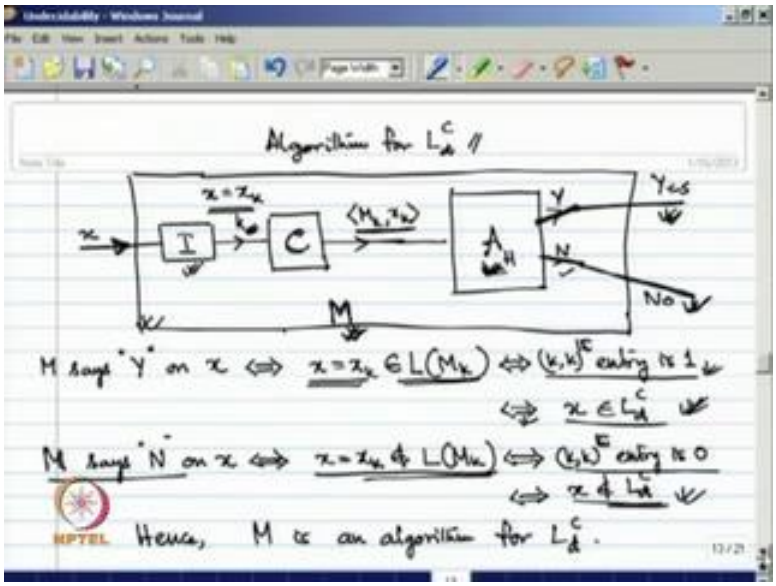
the halting problem MTM is decidable, so that we have assumed or convection, so since MTM is decidable there must be an algorithm say this is A H that decides MTM.

So, for convection we have assumed that halting problem decidable, so therefore, it must have an algorithm that decides MTM and that algorithm is A H, so that means, if A H is the algorithm given any input string M and x encoded, where M is a Turing machine and x is a string A H is will always say yes. If x belongs to L M and a s A H will say always say no if x does not belong to L M, so this is an algorithm it will always have an yes and no answer if m accepts x it will say yes, if m does not accepts x it will say no. So, this is a algorithm for A H, now we use A H to construct an algorithm that x in that decides the compliment of the language L d compliment.

So, if compliment of the language L d is decidable because we have we construct an algorithm, then see since we have algorithm for compliment L d; that means, L d compliment is must be decidable, if L d compliment is decidable this implies that L d must also be decidable. Because that decidable language are closed in the compliment because if we have an algorithm for a language, then its compliment can also be residing just by complimenting or reversing the output.

But you have already shown that L d is not even recursively enumerable therefore, our original assumption that such an algorithm A H exist to decide MTM is wrong; that means, since we have arrives a contradiction our original sums on that such an A H exist to decide MTM must be wrong.

(Refer Slide Time: 37:00)



Now, let us see the algorithm for compliment of L d to decide a compliment of language L d by using the algorithm for MTM, so this is A H algorithm for the halting problem given any input say M k x k encoding of a Turing machine. And x k, it will always say yes or no depending on whether x k belongs to M k, it will say yes if x k does not belong to M k it will say no.

Now, the algorithm for L d compliment is M, we use in m this algorithm A H, M takes a string x as input, then it uses a machine which is an indexed machine that finds out the index k, such that x is x k in the sequence in the sequence, we have arranged all the strings in some order wise we call say canonical order. So, what is index k will be decided by this index machine and that is always possible a Turing machine can always durable.

Once, it has found out this in the x k there is a constructor Turing machine that constructs the sequence M k and x k that is also let us possible, if we know the index k was a corresponding Turing machine for that k, M k and the string x k. So, this will be given as the output for c and this output we used input to the Turing machine whether regarding say A H and whenever A H say yes M also says yes, whenever says no M also says no now suppose that this M says yes on x.

So, when M says yes on x, we know that x is x k it must belong to L of M k, because A H says yes whenever x k belongs to L of M k; that means, k k'th entry is 1, so x k

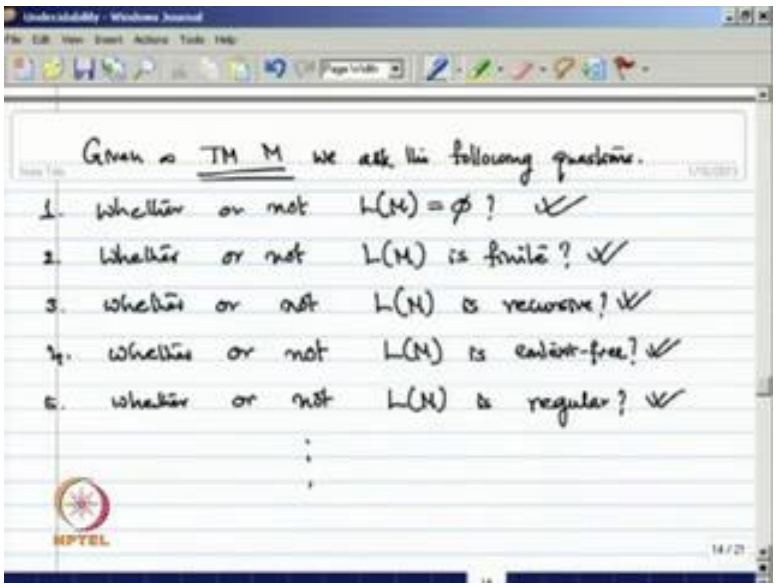
belongs to L of M k, if and only if k k'th entry is 1, if and only if x belongs to L d compliment, so that is our definition of L d compliment. Similarly, M says no on x on the string x if and only if x k does not belong to L of M k.

So, this M will say no if and only if this M k, x k entry this is rejected by A H; that means, x k does not belongs to L of M k if and only if k k'th entry is 0 and by definition if and only if this x k or x does not belong to L d compliment. So, therefore, we have whenever say given an input x to this machine M will always have yes or no answer depending on whether m x belongs to L d compliment or x does not belongs to L d compliment.

So, therefore, this is clearly an algorithm for L d compliment, so since now we know that we have an algorithm for L d compliment, but we have already know that since we have an algorithm for L d compliment, we have also an algorithm for L d. Because, the is close on that compliment, but we know that L d is not even recursively enumerable therefore, our original assumption that such an algorithm A H, H is for deciding the halting problem is wrong.

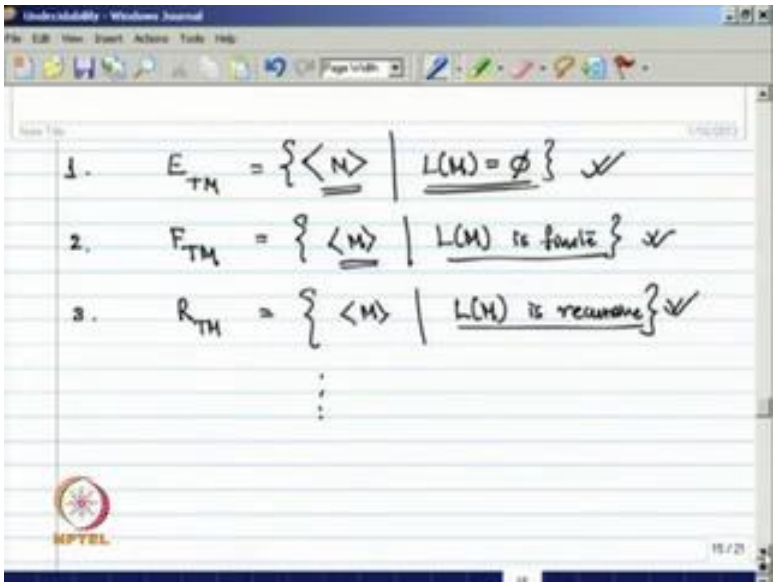
So, no such algorithm exist to decide this halting problem therefore, we prove the original claim that halting problem is undecidable. Now, given any other problem which is undecidable we can reduce this halting problem to that problem to show that the given problem is also undecidable.

(Refer Slide Time: 42:20)



Now, consider the following problem for a given Turing machine say M is a Turing machine, we consider a following problem, so whether or not L M is empty; that means, the language accept by the Turing machine is empty. Whether not the language accept by m is finite ho whether or not L M is recursive or is L M a regular or whether the language accept by the Turing machine is context free we want to know whether these problems are decidable or not.

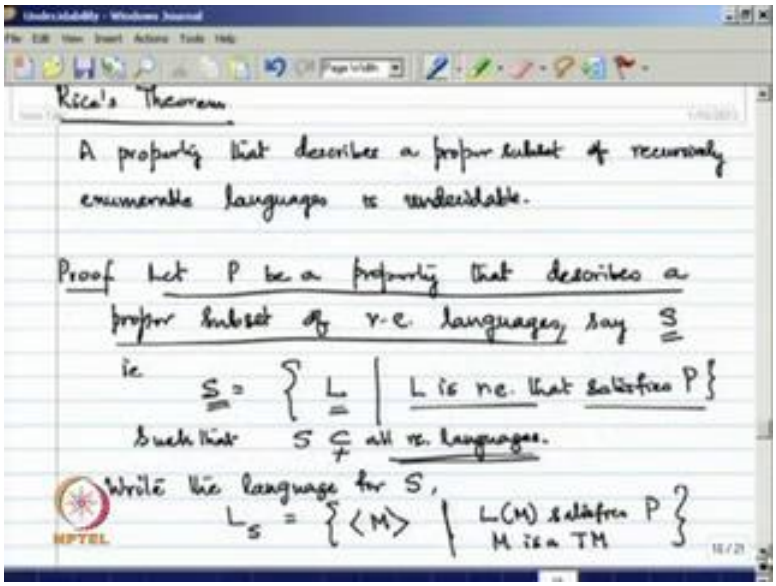
(Refer Slide Time: 43:10)



A corresponding language problem, the language for those problems can be written as say ETM emptiness for Turing machines set of all those encodings of Turing machines such that L M is phi. So, we are in finiteness set of all those Turing machines say M such that L M is finite similarly recursive.

So, set of all Turing machines m L M is recursive or say L M is context free that way we can find out or denote the corresponding languages, we can show that all these problems are basically undecidable to show that these problems undecidable. We need to reduce each and every problem or I mean we can reduce halting problem to each and every problem since halting problem is already undecidable, this problems also undecidable. Now, instead of giving separate derivations for each of these problems what we can do is that we give a general theorem that using the theorem, we can show that each and every problem is that we have already stated these are all undecidable.

(Refer Slide Time: 46:57)



Now, this general theorem is say to be your rice's theorem, so what is says is that a property that describes a proper subset of recursively enumerable languages is undecidable. That property it may be any property like say it is finite finiteness or emptiness or whether is regular recursive these are all properties the state be subset from say sigma star is a recursively enumerable language.

We say that there is a property and there is a proper subset one is the emptiness and the whole set these are not proper subsets and these are decidable the sense that, suppose we consider or the whole set just take the example. Suppose in a university there are some faculties lots of faculties we have and there suppose there is only one department say is computer science department.

If we asked the question given a faculty say this is a faculty L is a faculty, whether he belongs to whether he is a computer science faculty, so this is quite obvious because the where function of whole set the proper set of faculties and there is a only one department. So, it is quite obvious that the faculty must be from computer science department similarly, if we do not give any then that is also obvious there is a emptiness.

Other, that any other set which a proper subset of recursively enumerable language and you can show that a property, that describes a proper subset of a recursively enumerable language is undecidable. That means, say P be a property that describes a proper subset

of recursive language say S is a property a set of all the languages L, such that L is recursively enumerable that satisfies P.

So, that is how we the set represents the proper such that S a proper subset of all recursive enumerable languages now we write the language for S like this like this is L S the set of all Turing machines such that L M satisfies P M is a Turing machine. So, M is a Turing machine and L M satisfies P, so what we have understood is that this language L S is undecidable.

This is a proper subset and this describes this for all those every Turing machine m satisfies this L M satisfies, the property whatever this can say that language, it is a recursive or that language is regular or that language of is Turing machine and like that. And we know that this is a must always be proper set for example, say emptiness there may Turing machines some might accept empty language and when Turing machines which may not accept empty language.

So, therefore we consider all those Turing machines some accept empty language and some do not accept empty language, so therefore, that set is basically a proper subset of a recursive enumerable languages. So, we will want to show that this language L S is undecidable, if you can show that this language L S undecidable. So, any property related to recursively enumerable languages will be undecidable any property means, it must be say subset of proper subset of recursively enumerable languages, so will consider this proof in the next lecture.