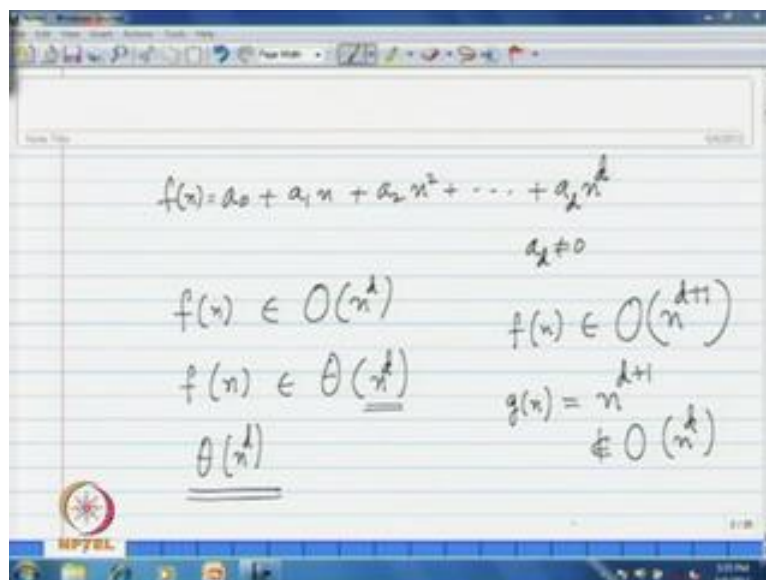


**Formal Languages and Automata Theory**  
**Prof. Dr. K. V. Krishna**  
**Department of Mathematics**  
**Indian Institute of Technology, Guwahati**

**Module – 14**  
**Introduction to Complexity Theory**  
**Lecture – 02**  
**P and NP**

Complexity theory, today we will be discussing one of the important problems in mathematics and computer science. So, in connection to that, first let me overview and what we have already discussed about you know regarding time bounded turing machines. And in that context, we have talked about class of languages, which are decidable in time  $t$  the class time  $t$ . We have introduced the notation, which is representing which is to denote the class of turing decidable languages, of course decidable languages, which is the time parameter  $t$ . So, in decidable in time  $t$ , that we have introduced.

(Refer Slide Time: 01:17)


$$f(n) = a_0 + a_1n + a_2n^2 + \dots + a_dn^d$$
$$a_d \neq 0$$
$$f(n) \in O(n^d)$$
$$f(n) \in \underline{\underline{\Theta(n^d)}}$$
$$\underline{\underline{\Theta(n^d)}}$$
$$f(n) \in O(n^{d+1})$$
$$g(n) = n^{d+1}$$
$$g(n) \notin O(n^d)$$

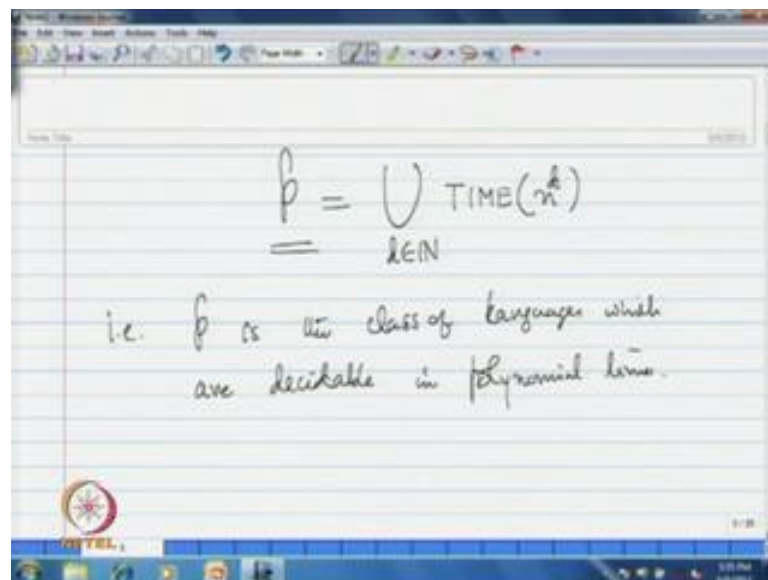
Now, let me just quickly review, some of the fundamentals related to the polynomials. If I consider a polynomial say  $a_0$  plus  $a_1n$  plus  $a_2n^2$  and so on, plus  $a_dn^d$ , where  $d$  is the degree of this polynomial. I will assume  $a_d \neq 0$ , if I call this polynomial say  $f(n)$  you know clearly that,  $f(n)$  is in the class  $n^d$  is big  $O$  notation, you all might know. Now, more over you can see that this is in the class  $\Theta(n^d)$ ,

so this is of this polynomial  $f(n)$  is of same rate of growth with the polynomial just  $n$  power  $d$ , in so...

In fact, you can look at that this class,  $\Theta(n^d)$  you know in this class you have those polynomials of degree  $d$ . And in this context, you may understand that the polynomial  $f(n)$  is in the big  $O$  class of  $n^d + 1$  or may be higher number than  $d$ . But this, if you take a polynomial say for example  $g(n)$ , which is of degree say  $d + 1$ , you can see that say let me the simply take this polynomial,  $n^{d+1}$  this is a polynomial, we can see that this is not in  $n^d$  class.

So, here what I wanted to just quickly mention that, if you take those polynomials of same degree, whatever the lower order terms; what are the coefficients that you have. You can quickly understand that I hope all these things that you would have learnt, and you have done in other courses, particularly in algorithms also you have used all these things. You can say that a polynomial of degree  $d$  is of same rate of growth with any polynomial of degree  $d$ , what are the coefficients at the lower order terms, and what are the coefficients...

(Refer Slide Time: 03:51)


$$P = \bigcup_{d \in \mathbb{N}} \text{TIME}(n^d)$$

i.e.  $P$  is the class of languages which are decidable in polynomial time.

So, that is a  $\Theta(n^d)$  class that you have, now using this I will define an important class called  $P$ . So, what is this, this is I will define it as, now union time  $n^d$ ,  $d$  is a natural numbers running over natural numbers. So, what is the meaning of this, this is  $P$  is that is  $P$  is the class of languages, which are decidable in polynomial

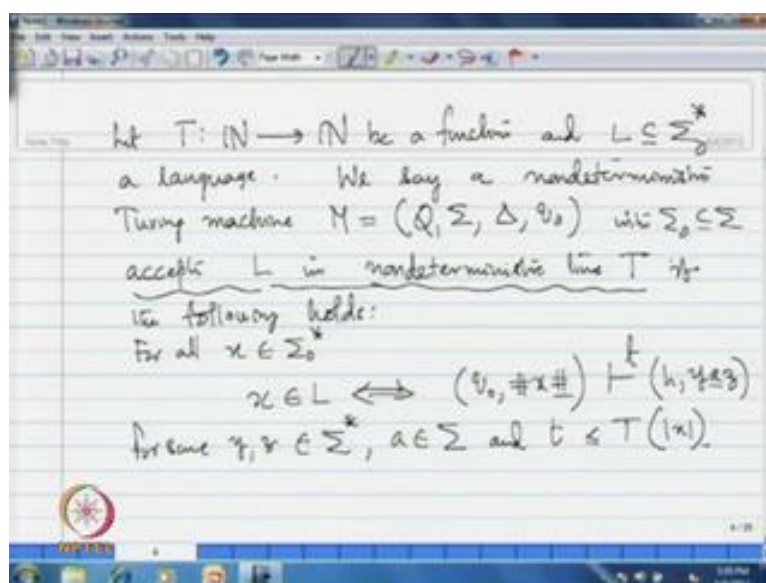
time, because if you take any language which is decidable in time  $t$ ; which is the polynomial you know that is in time  $t$  class.

Now, that  $t$  if it is a polynomial, it is in the class of time  $n$  power  $d$ , where  $d$  is the degree of that particular polynomial. Now, we have taken union of all those languages, even all those classes. So, this  $P$  we are defining the script to be I am writing, distinguishing between some other you know normal letters are wherever I would be using. So, this class  $P$  I have defined to be the class of all those languages, which are decidable in polynomial time. Now, if you ask for examples, of course there are what are the examples in the previous lecture that we have discussed.

They are all clearly in polynomial time; I have given some linear time languages, which are decidable in linear time particularly, regular languages which are decidable in  $2n + 4$  time. And we have some example discussed, that is working in quadratic time and what are all the languages, many languages so far we have discussed here, they are all in clearly in  $P$ , because what are the deciders that we have constructed. You can verify that, they are working in polynomial time; there is a polynomial time decider to compute to decide those languages.

Now, so thus for the class  $P$  I am not giving any more examples, because we have we all are already equipped with many of the examples. Now, I will talk about a new concept called  $NP$ , what is that  $NP$  this is essentially non deterministic variant of the class  $P$ ,  $P$  we have just defined. Now, I will look for the non determinism, when I am looking for the non determinism, particularly non deterministic turing machine. The way that we have introduced that is not a decider, so it is not actually taking a decision on the inputs, and in contrast with the standard turing machine.

(Refer Slide Time: 06:52)



So, we will include that feature, when I am talking, when I am introducing the time bound on non deterministic machine, and accordingly we will define the class NP. First let me start with associating time to a non deterministic machine. So, for that let me start, so let  $T$  from natural numbers to natural numbers, this is the time function be a function, and consider a language  $L$  over some alphabet, let me say  $\Sigma_0$  a language as earlier.

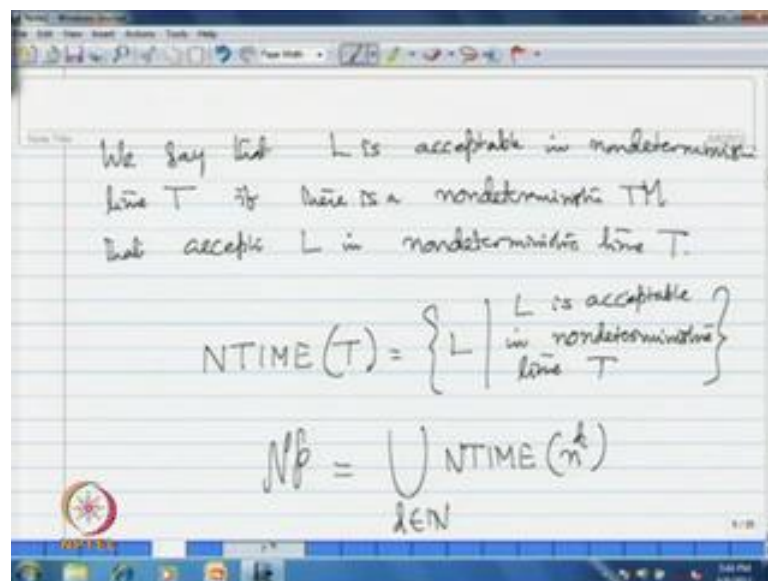
Now, I will start with the non deterministic machine instead of standard deterministic turing machine. We say a non deterministic turing machine, non deterministic turing machine, say I may call it as  $M$   $Q$   $\Sigma$ , this is the transition relation there  $q_0$  the initial state with the alphabet under consideration is subset of the  $\Sigma$ , alphabet of that. This accepts  $L$  in non deterministic time, I have do not want to call it as time, because the variants is there I call it as non deterministic time  $T$ .

So, this time what we are associating for a non deterministic machine, we call it as it is accepting this machine is accepting  $L$  in non deterministic time  $T$ . If the following holds following holds, what are the conditions I write now is holds. So, what is that for all  $x$  in  $\Sigma_0^*$ , you take any string  $x$  is in  $L$ . If and only if, if you give that as input to the machine, so in this format we give, it will halt in time  $t$  in within  $t$  number of steps, of course when it is halting, what is there on the tape it is not decider.

So, I do not expect any yes or no sort of thing, but it will leave some something on the tape, that means for some  $y z$  in  $\Sigma^*$  something and of course,  $a$  in  $\Sigma$  and this  $t$  what it has to maintain, it should be less than equal to  $T \bmod x$ . So, the number of steps, it takes to compute on it by the time it halts, you know the number of steps it takes is less than or equal to  $T \bmod x$  with respect to with input parameter  $\bmod x$ , the length of  $x$ .

So, what we have defined essentially, this is a non deterministic variant of the earlier in the earlier lecture whatever that we have defined about time bounded turing machine. Here, when you are considering non determinism, we know that there is no decision here, which is something like saying yes or no on a given input. So, if it is halting it has to halt within the required number of steps that is with respect to the time, we are associating  $t$ .

(Refer Slide Time: 10:13)



So now, as earlier we say this is a continuation of that definition, we say that  $L$  is acceptable in non deterministic time  $T$  in non deterministic time  $T$ . If there is a non deterministic turing machine, there is a non deterministic turing machine that accepts  $L$  in non deterministic time  $T$ . So, we know just we have defined, when do we say a non deterministic turing machine accepts a language in non deterministic time  $T$ .

Now, if you want to say a particular language is acceptably non deterministic time  $T$ , you should have one non deterministic turing machine doing this performing that. So, in connection to that as earlier, now I introduce this notation  $N$  time a class, so  $N$  time  $T$  I

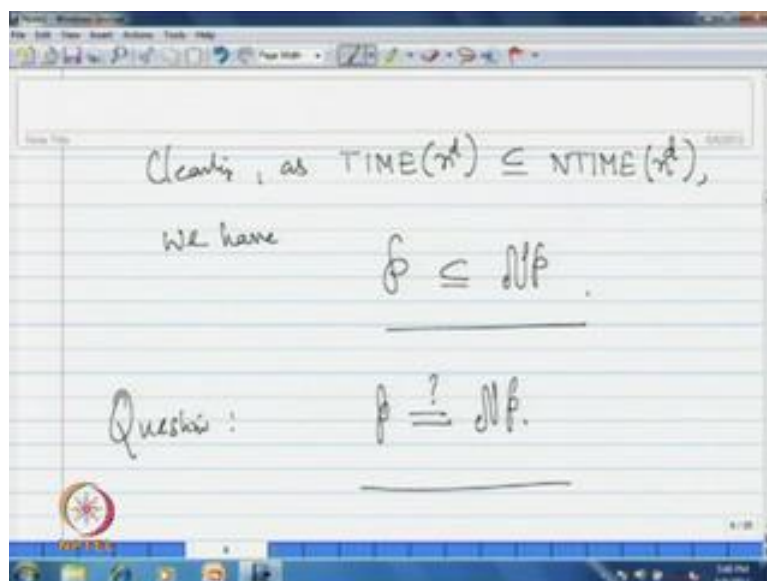
mean all those languages  $L$ , which are decidable else, because this is non determinism here which is acceptable, because there is no decision here, acceptable in non deterministic time  $T$ .

So, as earlier when we have talked about time  $t$ , those are the languages which are decidable in time  $t$  is that is deterministic turing machine. Here, we are having a non deterministic version, so  $N$  time  $T$  is all those languages which are acceptable, because non deterministic machine would not take a decision. If the string is in the language, it will simple accept, so within the required time it has to accept. So, acceptable in non deterministic time  $t$ ,  $NT$

So, extending this now, let me introduce the class what I was talking about NP, I am ((Refer Time: 12:44)) script letter to distinguish from other classes, that would be talking about union  $N$  time  $n$  power  $d$ . So,  $d$  runs over natural numbers, so what is the meaning of this, this is all those languages which are acceptable in non deterministic polynomial time, because  $n$  power  $d$  when I am writing. So,  $N$  time polynomials if you look at you know this class is essentially same as  $N$  time  $n$  power  $d$ , where degree is of that particular polynomial.

So, we know this, so if you if I am considering union of all those languages on all those classes. So, that is what I am calling NP. So, in words I can say that these are those languages, which are acceptable in non deterministic polynomial time. Instead of this saying non deterministic time that time I am associating it should be a polynomial. So, NP is the class of all those languages which are acceptable in non deterministic polynomial time  $t$ .

(Refer Slide Time: 14:06)



Now, you can quickly see that, we know that every deterministic turing machine can be seen as a non deterministic turing machine. So, that is the restricted version, and so clearly as time  $t$  or in particular I can write  $n^d$ , it is a subset of  $N$  time  $n^d$ . We have this class  $P$  is contained in  $NP$  is very quickly to see, because every deterministic turing machine can be treated as a non deterministic turing machine, and thus I can say that  $P$  is contained in  $NP$ .

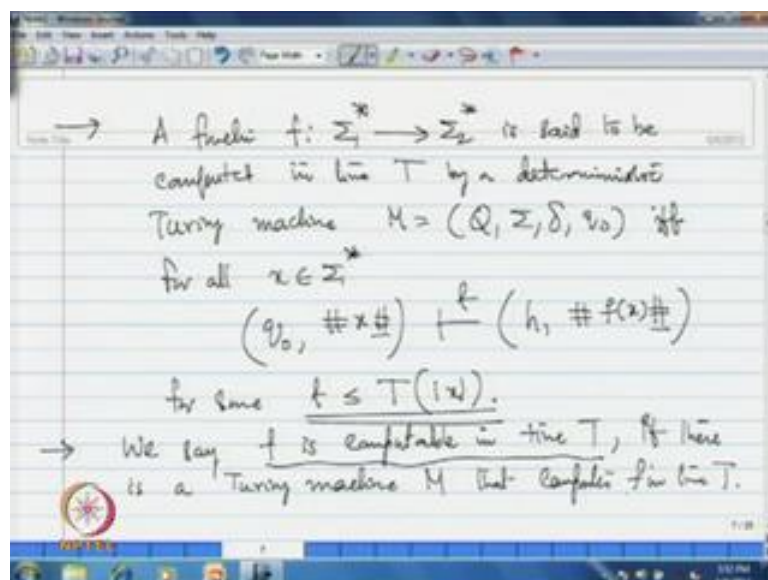
Now, the question is whether this two are equal, so the question is whether or not equal to  $NP$ . Now, this particular problem in mathematics and computer science, this is one of the you know toughest problems and this problem has such an importance, the importance of with respect to the importance of this particular problem. You know, this is one of the seven problems listed for million dollar prize by clay mathematical institute.

So, clay mathematical institute just to give some you know importance of this with respect to that, clay mathematical institute has announced million dollar prize for one for each of the seven problems. And only one problem so far is settled, and this is one of the problems among those seven problems. And if one settles this that means, to prove whether  $P$  equal to  $NP$  or  $P$  not equal to  $NP$  it is very clear, that  $P$  is contained in  $NP$ .

Now, the point is whether there is a language in  $NP$  which is not in  $P$ , so that is the question. So, if one can address this problem, who will get; he will get; he or she will get, 1 million dollar prize as per the announcement of clay mathematical institute. So,

this is one of the important and long standing problems in mathematics and computer science.

(Refer Slide Time: 16:44)



Now, in order to talk about more details about this particular problem, whether P equal to NP I will little more create some background to understand about this. In this lecture, that is let me start with this particular concept, a function  $f$  from  $\Sigma_1^*$  to  $\Sigma_2^*$  is said to be computed by a deterministic are computed in time  $T$  by a deterministic. Because now, I have to mention earlier I used to simply say turing machine, because now we are talking about non determinism also simultaneously, I will now mention that deterministic turing machine  $M$ .

So, let me write  $Q \Sigma \delta q_0$ , if and only if for all  $x$  in  $\Sigma_1^*$ , if you give that as input. So, the number of steps I am counting that is the main variant here,  $f x$  the output that you are getting with  $T$  number of steps for some  $T$ ,  $T$  less than or equal to  $\text{mod } x$ . So, in connection to this definition, I will now make as earlier that we say  $f$  is computable in time  $T$ . If there is a deterministic turing machine, I will simply write a turing machine  $M$  that computes  $f$  in time  $T$ .

So, look here whatever the earlier procedures or you know concepts that we have discussed with respect to turing machines. We are now quantifying them, for example in the previous lecture, when I am talking about time bounded turing machines. First standard turing machine I have taken there, I have consider only the so called a decider

and a quantifying method of that, that means we have the decision we are quantifying. So, with respect to a particular number of steps, the particular number is associated through a function, so we are calling it as a time function.

Similarly, when I am talking about non determinism, so for non deterministic machine also, we have associated the number of steps, because when it is taking, when it is computing certain thing. So, the number steps that we are counting and with respect to a particular time function that we are with respect to which we are talking about, we are associating a time function. Now, when I am talking about standard turing machine, you know there is a concept a turing machine computes a particular function.

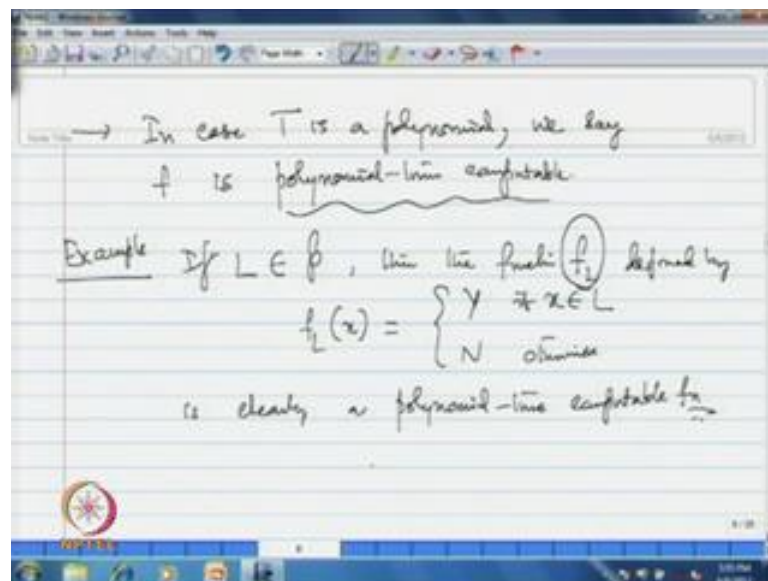
Now again, in that particular context also we are quantifying it, that means we are associating a time function to that particular concept. Now, in the similar thing I need not actually, I need not introduce, but as formalism I am introducing here. So, if you take a function a computable function that means, a turing computable function means there is a turing machine, which computes that particular function. Now, if you want to associate time to that how many steps it is taking to compute, whatever the on the input that you are giving.

You know, in a if a turing machine computes a particular function on every input, it will leave certain some output. Suppose  $x$  is given as input  $f x$ , if  $f$  is the particular function, it is halting by printing  $f x$  as output. Now, when we are associating time to that as earlier, now here you will count the number of steps and the number if you want to say, it is computing in particular time capital  $T$ , the time function. So, the number of steps it is taking should be less than or equal to capital  $T$  of that particular string length, in terms of that. So, that is what is the variant now we are discussing here.

So, a function  $f$  from  $\Sigma^1$  to  $\Sigma^2$  you take a function, which is the computable function, if you say there is a turing machine which is computing this. But now, when I am associating time to this, it is computed in time  $T$  by a deterministic turing machine  $M$ , if for all the inputs that is from  $\Sigma^1$ , when you take the number of steps it is taking if I write  $T$ , so it should halt within  $T$  number of steps, where  $T$  should be less than equal to capital  $T \bmod x$ . And now, a function  $f$  is computable in time  $T$ , if there is a turing machine that computes in time  $T$ , so this is the variant that we have defined.

Now, whatever the concepts that is associated to turing machines. We will now, in this particular topic complexity issues when we are discussing. We will give the parallel formalisms and discuss with respect to the quantifying that, any way this concept I am introducing, because we have just mentioned, we have just talked about a problem whether P is equal to NP what I will do, I will introduce an important class of languages, which are actually very much useful in order to understand, well about the problem P equal to NP.

(Refer Slide Time: 22:53)



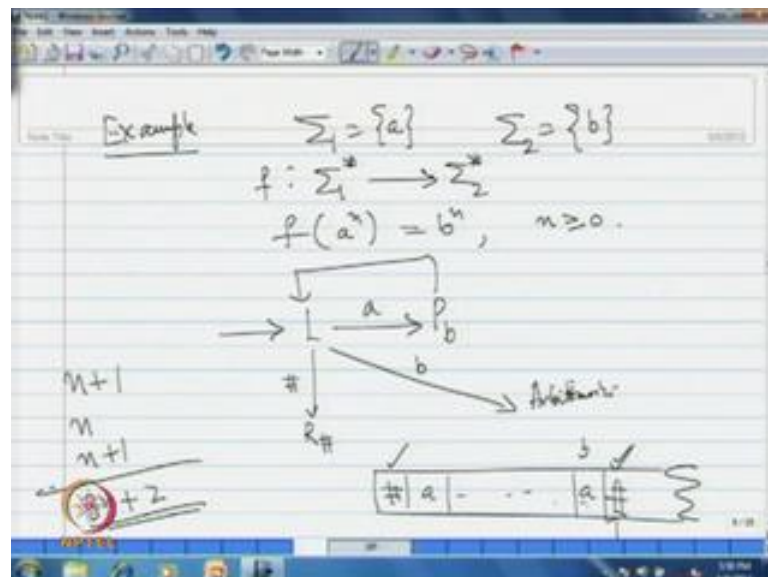
So, in that direction I will introduce a class called you know NP complete languages, NP complete the class NP complete. So, for which you know I will just build up the background. Now, through this concepts and now we will talk about the NP complete languages. In this particular context, in case if you have the  $T$  to be polynomial, in case  $T$  is a polynomial, in case  $T$  is a polynomial. We say that function  $f$  is polynomial time computable say this is polynomial time computable, so just we have mentioned when do we say function is computable in time  $t$  that we have mentioned.

Now, if this particular time what we are associating, if this time if it is a polynomial, then we say it is a polynomial time computable function. Now, just to look at examples, if you take a language, which is already in  $P$  that means there is a polynomial time decider for that if this is if you take a language in  $P$ , then this is simple example straight forward

example. Then the function, if I consider the function  $f: L \rightarrow \{Y, N\}$  defined by I define naturally this way  $Y$  if  $x$  is in  $L$  otherwise, suppose if you define like this.

You know the corresponding turing machine on  $x$ , it prints  $Y$  if it is in  $L$ , if  $x$  is not in  $L$  it prints  $N$ , you know this thing. Now, since this is in  $P$  there is a polynomial time decider, so the same decider you know computes this particular this thing is clearly. And thus clearly, this is a polynomial time computable function a polynomial time computable function. This is clearly a polynomial, so  $f: L \rightarrow \{Y, N\}$  whatever we have defined here, so the same decider will be useful.

(Refer Slide Time: 25:18)



Let me give some other example, which you may understand better of course, that is a trivial and straight example, what I have discussed now. Let me take this take  $\Sigma_1$  to be say singleton  $a$  a very simple example, so that you know you will understand this concept better. Consider singleton  $b$ , if I define the function  $f$  from  $\Sigma_1^*$  to  $\Sigma_2^*$ , you know strings of  $\Sigma_1^*$  there simply  $a^n$  (Refer Time: 25:38) a power  $n$  form, what do I consider  $f$  of a power  $n$ , I will simply send it to  $b^n$ ,  $n$  greater than or equal to 0, so I will just define like this.

Now, you can give a decider for this, you can give a turing machine which computes this function. As earlier our input format this is we have fixed, so certain number of  $a$ 's that you will give on the tape, we are starting here. So, what do you do, you may simply go through this and replace each  $a$  by  $b$  this what you will print, and you come back to this

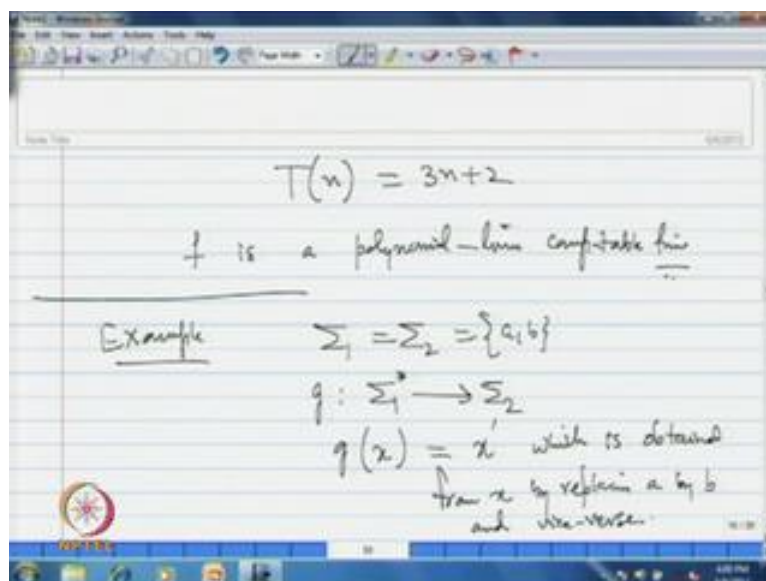
position and halt. So, either you ((Refer Time: 26:21)) going you can print b (s) are while coming you can print b (s), whatever your wish take a left move.

If you have a print b there, and take a left move keep doing this, when you receive blank here you may simply take R hash, that means you will come to this end from here you will come to this end. So, because you have to halt finally here, by that time you know you would have replaced all the a (s) if they are available with b (s). Now, you can ask me, because when I am constructing this particular turing machine. The alphabet can be can have this blank a (s) and b (s), because b needs to be printed. Now, in this particular context, if you received say for example b you can define arbitrarily, because we do not want to handle that here.

So, you define arbitrarily whatever that you want, because particularly when you take the input which is in  $\Sigma^*$ , because look at the definition the way that we have defined. If you take  $x$  in  $\Sigma^*$ , if you give that as an input what are the number of steps it is taking to halt with the output  $f(x)$ , because  $x$  we are taking from  $\Sigma^*$  only. So, I am not worried about if there is b, so you define arbitrarily. So, this particular decider, what is doing if you have input a power  $n$ , then it will print  $b^n$  on the tape, and it will halt how much time it will take?

Suppose, if you look at that, look here you are starting from here, you take a left move, you take a left move here, and then left move and so on. So, if you have  $n$  number of a (s) here, you take  $n$  number of left moves and one more left move to this. So, there are  $n + 1$  left moves and while coming you are replacing each a by b, that means there are  $n$  number of printing steps, because each a (s) so  $n$  number of printing steps. So, from this position, when you want to go here again you have  $n + 1$  right moves, the total is  $3n + 2$  steps that you are taking.

(Refer Slide Time: 28:26)



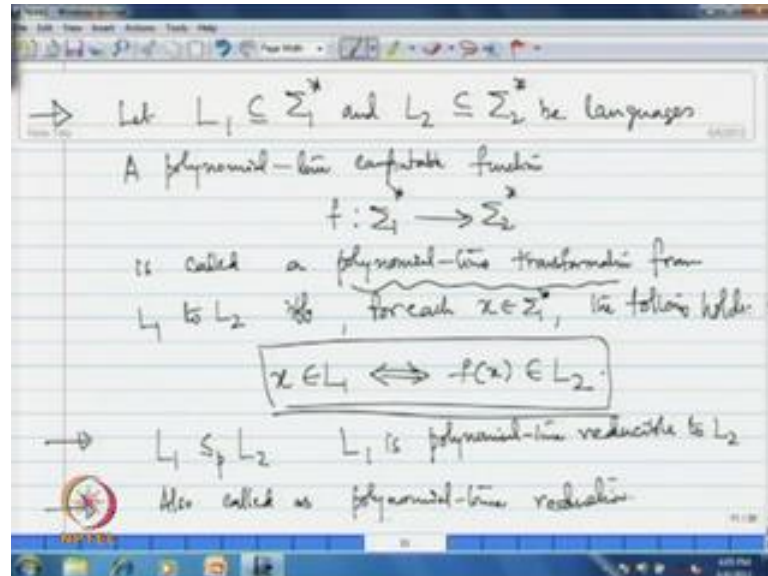
So, you can quickly see that this particular function, if you consider  $T(n)$  to be  $3n + 2$ , this function  $f$  I have considered,  $f$  is computable in time  $t$  in particular this  $t$  is a polynomial. And therefore, is a polynomial time computable function, likewise you know just to make this concept familiar, you know you can consider several examples of this sort and practice and see this say some more let me just give you here.

If I consider say  $\Sigma_1$  equal to  $\Sigma_2$  equal to say  $\{a, b\}$ , I may consider a function, say for example  $g$  from  $\Sigma_1^*$  to  $\Sigma_2^*$ , I give like this. If you take any string  $x$ , I give  $x'$  where  $x'$  is obtained from  $x$ , which is obtained from  $x$  by replacing  $a$  by  $b$  and vice versa. Suppose, if you consider this function, what you are supposed to do, wherever  $a$  (s) there on the input tape you have to replace it by  $b$ ; wherever  $b$  (s) there you have to replace it by  $a$ , that is what you have to do. You will scan through that once, and you will come back and halt the original position.

Because, the output format is when you give  $x$ , you have to print  $f(x)$  on the tape and halt at the right blank. So, this kind of function also you can quickly realize that I hope you can observe that it will, you can make it in  $2n + 2$  time. So,  $3n + 2$  is a polynomial, this is also a polynomial time computable function. And there are several such, you know deciders that you have several such turing machines that you have constructed to compute functions. So, you can now cross check like, where whether it is

taking polynomial time, what is the polynomial that you are getting, accordingly you can report that some of the examples which are computable in polynomial time.

(Refer Slide Time: 30:47)



Now, let me give another definition here, let  $L_1$  a language over  $\Sigma_1$  and  $L_2$  a language over  $\Sigma_2$  be languages. I am defining this now, what I am defining a polynomial time computable function  $f$  from  $\Sigma_1^*$  to  $\Sigma_2^*$  is called a polynomial time transformation. So, I am introducing this technical term here, polynomial transformation from  $L_1$  to  $L_2$ ; if and only if, for each  $x$  in  $\Sigma_1^*$  the following holds, what is that  $x$  is in  $L_1$ ; if and only if this  $f$  of  $x$  is in  $L_2$ .

Now, can you guess what is the corresponding concept without talking about the time here, look what is the concept we have talked about. If you consider language  $L_1$  over  $\Sigma_1$ ,  $L_2$  over  $\Sigma_2$ , we are talking about a computable function from  $\Sigma_1^*$  to  $\Sigma_2^*$  satisfying a particular property, what is that particular property  $x$  is in  $L_1$ , if and only if,  $f$  of  $x$  is in  $L_2$  what is this particular concept, thus when you want to talk about this you have we have discussed in the context of undecidability.

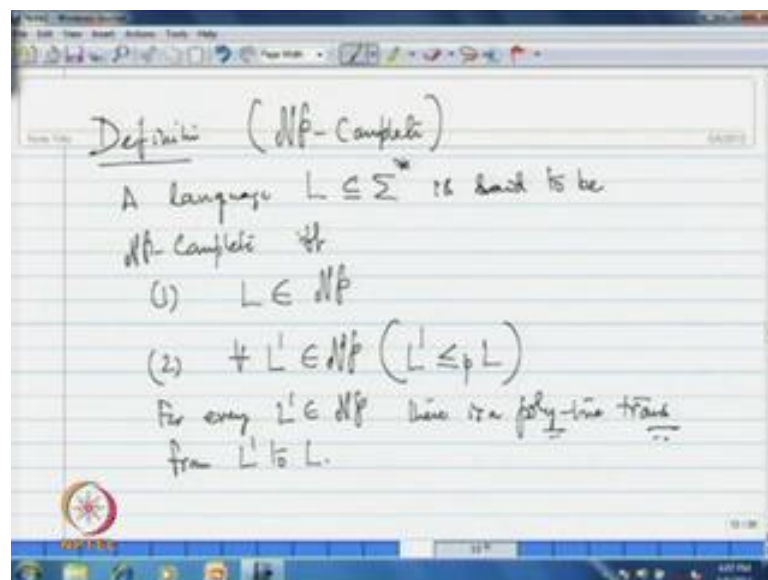
The reduction, this is a reduction mapping as we say a language  $L_1$  reducible to a language  $L_2$ , when do we say that if there is a computable function from  $\Sigma_1^*$  to  $\Sigma_2^*$  satisfying this condition  $x$  is in  $L_1$ , if and only if,  $f$  of  $x$  is in  $L_2$ . So, this is essentially quantitative version, this is quantitative version with respect to time of reducibility concept. So, I can also say this particular thing, because in which case what

we have written, the notation  $L_1 \leq L_2$  that means  $L_1$  is reducible to  $L_2$ .

So, corresponding to that now this in this particular context, I will put a  $p$  here which means that  $L_1$  is polynomial time reducible to  $L_2$ , so we know you know already the concept  $L_1$  reducible to  $L_2$ , that means there should be a turing computable function from  $\Sigma_1^*$  to  $\Sigma_2^*$  satisfying this particular condition, whatever I am marking here. Now, we are associating to the computable function, it should work in polynomial time, so it should be a polynomial time computable function  $f$ .

And therefore, corresponding to that polynomial time, I am just subscripting this  $p$  also here in this notation. So, we call this as  $L_1$  is polynomial time reducible to  $L_2$ . You can now say, instead of calling this polynomial time transformation, if you want you can also use this is called polynomial time reduction is also, you can call it as also called polynomial time reduction. So, that you are taking the reduction takes place from  $L_1$  to  $L_2$  more over this reduction takes polynomial time.

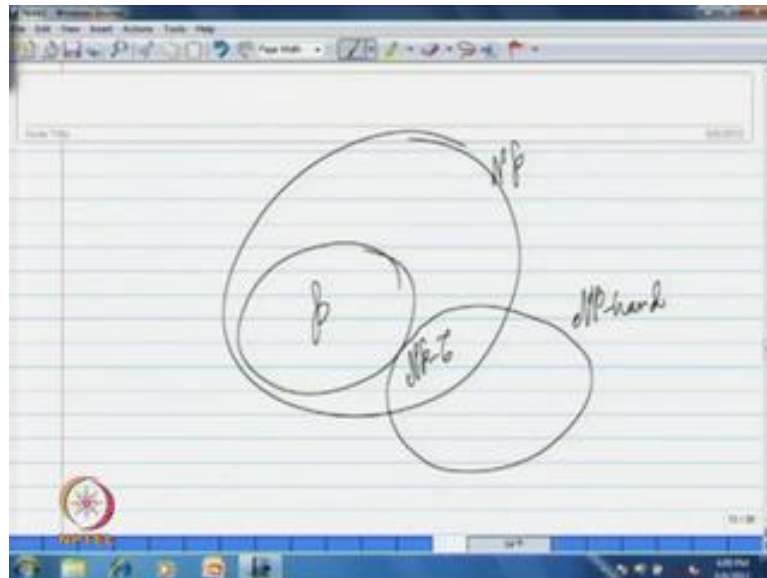
(Refer Slide Time: 35:13)



So, the notation that we have introduced is this. Now, using this particular concept, let me give the definition of NP complete class, what we are looking for this is for NP complete language, a language  $L$  say over some sigma, over some is said to be NP complete, if and only if. The first condition, we put is that language should be in NP, number 2 for all  $L$  dash.

If you take any language in NP, what do I require this L dash should be reducible to L in polynomial time. So, that is for every L dash in NP there is a polynomial time transformation. Let me write short cut, polynomial time transformation from L dash to L. So in other words, every language in NP is reducible to L in polynomial time.

(Refer Slide Time: 37:03)

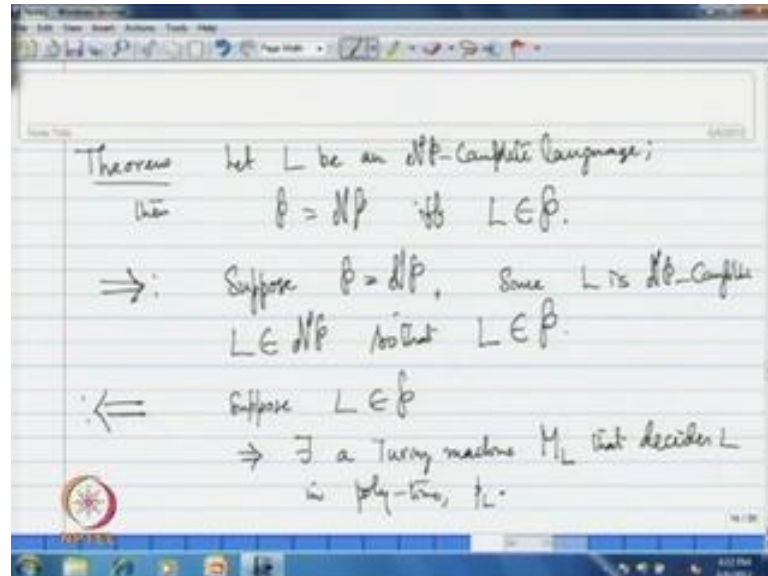


So, the class NP complete we define like this, clearly this is a subset of NP. We are now, we can you know suppose if this is NP class, we have already observed that this is you know P is contained in NP, but we do not know whether there is whether there are some elements here or not in this gap. And now, this statement which is given in number 2 is also people call it as NP hardness, what is the meaning of this that means the problem L is as hard as any NP problem that means, if you reduce a problem in NP to this. You know the statement, that if L is solvable then L dash is solvable, so that means this problem L is as hard as any NP problem, therefore this is called it as this condition is also called it as NP hard.

So, this if I write like this NP hard, this intersection because a problem which is in NP and NP hard, this is what is NP c, these are what NP complete problems NP c. Now, I have drawn purposefully, you know by just with marking like this. Now, I will present to you one important result as follows: anyway let me just illustrate that once again reemphasis a language L in sigma star is said to be NP complete, if it is in N, if it is NP

as well as NP hard, that is how this diagram is clearly showing this intersection of NP hard is essentially NP complete problems.

(Refer Slide Time: 39:00)



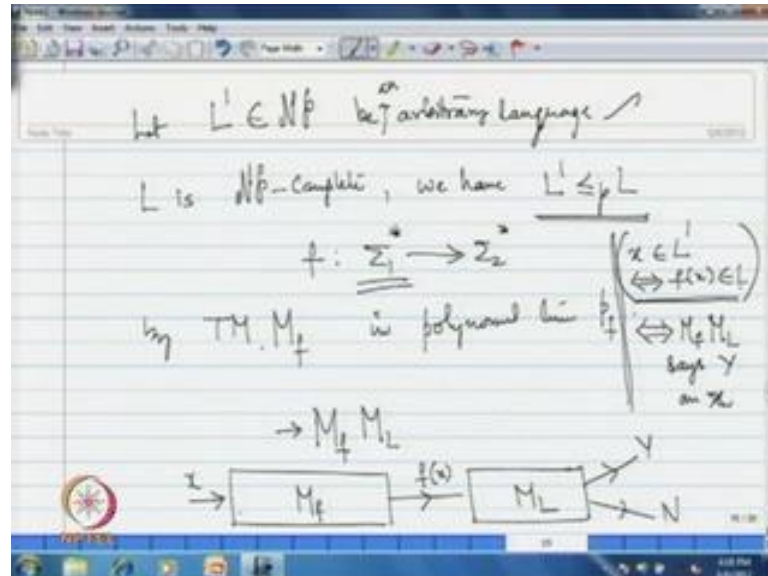
Now, the following result will be helpful to understand the big picture about the problem with respect to NP complete class. The theorem is this, the theorem is let  $L$  be an NP complete language, then  $P$  equal to  $NP$ , if and only if  $L$  is  $NP$  you see the statement. If a language is NP complete and if that language is also in  $P$ , then  $P$  equal to  $NP$  and  $P$  equal to  $NP$  is possible, if  $L$  is in  $P$ , that means to get that million dollar prize, what you have to do, if you can identify an NP complete language which is in  $P$  that is all.

So, understanding this particular concept you know through NP complete is giving some characteristic property, about the problem  $P$  equal to  $NP$ . First let us prove this, because one side is very obvious very clear that is this portion. Suppose,  $L$  is in  $P$  this implies there is a deterministic turing machine. So, this side is obvious, the forward direction is obvious, let me consider that first. Suppose,  $P$  equal to  $NP$  then it is very clear, that since  $L$  is NP complete from the definition, you see this  $L$  is NP complete,  $L$  is in  $NP$  from the definition of NP complete  $L$  is in  $NP$ .

So that, and since  $P$  we have assumed  $P$  equal to this  $P$  equal to  $NP$ , we can say that  $L$  is also  $NP$ . So, this side is very obvious, from the definition of NP complete. Now, let us consider this part, suppose  $L$  is in  $P$  this implies there is a deterministic turing machine, say  $M_L$  this implies there exist a deterministic turing machine, let me write simply

turing machine  $M_L$ , that decides  $L$  in polynomial time. So, let me say that is  $P \subseteq L$  the polynomial corresponding polynomial let me say it is  $P \subseteq L$ .

(Refer Slide Time: 42:27)



Now, what do you have what we have to show  $P$  equal to  $NP$ , we know clearly that  $P$  is contained in  $NP$ . Now, if you choose an arbitrary language in  $NP$ , we will observe that that is also in  $P$ , so that means let me start with an arbitrary language. Let  $L' \in NP$  be arbitrary, let us take an arbitrary language from this the arbitrary language, so let us consider an arbitrary language since  $L$ .

Now, let us use the hypothesis that  $L$  is  $NP$  complete, since  $L$  is  $NP$  complete what do we have, this  $L'$  should be reducible to  $L$  in polynomial time, that means we have this  $L'$  is less than equal to  $P \subseteq L$ . So, that is there is a polynomial time transformation  $f$  from this is a polynomial time transformation. Let me write like this, say there is a polynomial time transformation under consideration say  $\Sigma_1^*$  to  $\Sigma_2^*$ , because  $L$  is an  $NP$  complete language I said. So,  $L'$  is subset of  $\Sigma_2^*$  if you assume.

So, there is polynomial time this is a polynomial time transformation, which reduces  $L'$  to  $L$ , so that means this is a turing computable function, that computes that is computed by some turing machine  $M_f$  in polynomial time, say let me call it as say  $P \subseteq f$ . So, I have if I am clear here, you start with an arbitrary language  $L' \in NP$  take an arbitrary language. Now, since  $L$  is  $NP$  complete we have  $L' \leq_P L$ .

$L$ , that means  $L$  dash is reducible to  $L$  in polynomial time, that means there is a polynomial time transformation  $f$  from the underlying alphabets.

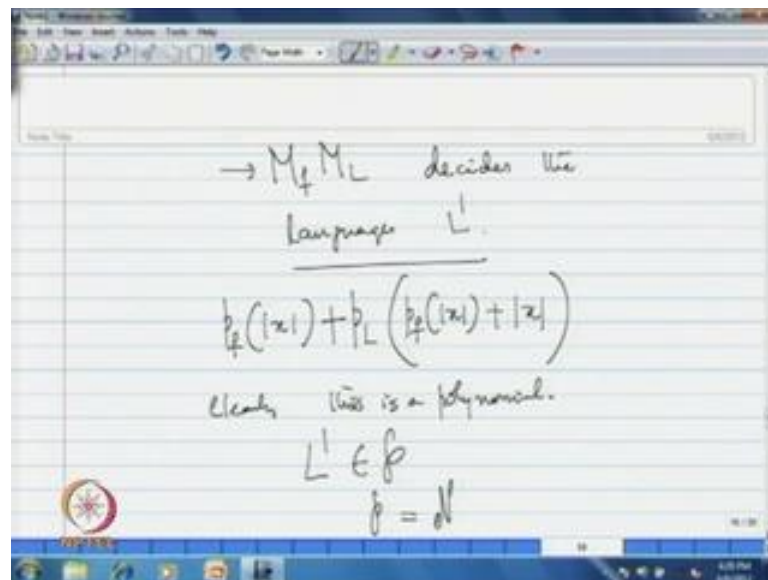
So here, I am assuming that  $L$  dash is subset of  $\Sigma^1$   $L$  is subset of  $\Sigma^2$ . So,  $f$  is a computable function, which is computed by turing machine, so this is turing machine, so this is computed by turing machine  $M_f$  in time  $P_f$ . Now, what do I do I will give you a turing machine, which computes  $L$  dash in polynomial time, which decides  $L$  dash in polynomial time. So that, this is a arbitrary language  $L$  dash goes in  $P$ , and thus you will have that  $P$  equal to  $NP$ .

So, what is that turing machine, you can also quickly see this if you consider  $M_f M_L$ , what is  $M_L$ ?  $M_L$  is a turing machine that decides  $L$  if I compose this two turing machines  $M_f$  and  $M_L$ . So, in this form now, you observe that if you give any input to  $M_f$ , what happens from  $\Sigma^1$ , whatever that you give by that time. So, let me draw a small picture here, what I am the composition you know this kind of construction. So, the input  $x$ , say for example, if you give to this what are the output, because this  $M_f$  produces  $f(x)$  this  $f(x)$  will be given as input to  $M_L$ , and  $M_L$  is a decider it says yes or no this is the idea.

Now, you observe that if you give  $x$  to  $M_f$ , it gives  $f(x)$  and  $f(x)$  is fed to  $M_L$ , if it says yes, what is the meaning of that  $f(x)$  is in the language  $L$ ; if it says no,  $f(x)$  is not in the language  $L$ . So, now using this condition, since  $f$  is a polynomial time transformation from  $\Sigma^1$  to  $\Sigma^2$ , what is the condition associated to this here, whatever that  $x$  is in  $L$  dash; if and only if,  $f(x)$  is in  $L$  this is the condition associated to this Polynomial, it is a polynomial time transformation.

Now, it effect is in  $L$  then corresponding to that it says yes, if  $f(x)$  is not in  $L$ , then it says no, that means if and only if, this particular turing machine this  $M_f M_L$  says yes  $y$  on  $x$ . I am just marking here once again, if you give an arbitrary input  $x$  from  $\Sigma^1$  from this alphabet,  $M_f$  will compute and gives  $f(x)$  as output. So, with the condition that  $x$  belongs to  $L$  dash; if and only if  $f(x)$  is in  $L$  with this condition. And now,  $f(x)$  when you give input to  $M_L$ , it says yes only when  $f(x)$  is in  $L$  otherwise it says no. So, what is the condition we have  $f(x)$  is in  $L$ ; if and only if, this  $M_f M_L$  this particular turing machine says yes on  $x$ .

(Refer Slide Time: 47:49)



So, that means clearly this  $M_f M_L$  this Turing machine  $M_f M_L$  decides the language  $L^1$ . So, what did you get, we have got a decider to decide the language  $L^1$ . Now, how much time it takes, if you look at that since  $M_f M_L$  how much time it is taking if you look at, this particular one by that time it finishes, as we have assumed  $p_f$  is that particular function polynomial. So, it takes this much time to get the output here,  $f(x)$  and what will be the length of  $f(x)$ . If you look at, the length of  $f(x)$  will be maximum this much, this is the maximum length of  $f(x)$ .

Now, on this particular output  $f(x)$  will be applying this  $M_L$ , where  $p_L$  is the polynomial that much time it will take, so sum of these two. So clearly, sum of these this is a polynomial with the parameter  $\text{mod } x$ , and this is the polynomial composition of a polynomial. So, clearly this is a polynomial, so that means for  $L^1$  we have got a decider, which decides in polynomial time and thus  $L^1$  is in  $P$ . So since,  $L^1$  is arbitrary we have  $P$  is equal to  $NP$ . So, through this result you understand the importance of  $NP$  complete languages.

So, if you know some just look this statement, if you know some  $NP$  complete language. If you can obtain a polynomial time decider for this, then  $P = NP$ . The problem will be settled, but of course getting such a language,  $NP$  complete language. And in fact observing a language is  $NP$  complete is very difficult, because if you look at the

definition, you have to reduce every NP language to that in polynomial time, so that is one hard condition to observe.

And you know, even if you get an NP complete language in hand, observing that it has a polynomial time is even tougher task, so that means to show that  $P = NP$ , we have a characterization here. So, through this I am just presenting, you know the concept of NP complete through the concept of NP complete, and we have observed the importance of NP complete languages as well. We will continue these discussions in the future lectures, you know by identifying certain NP complete languages, etcetera.