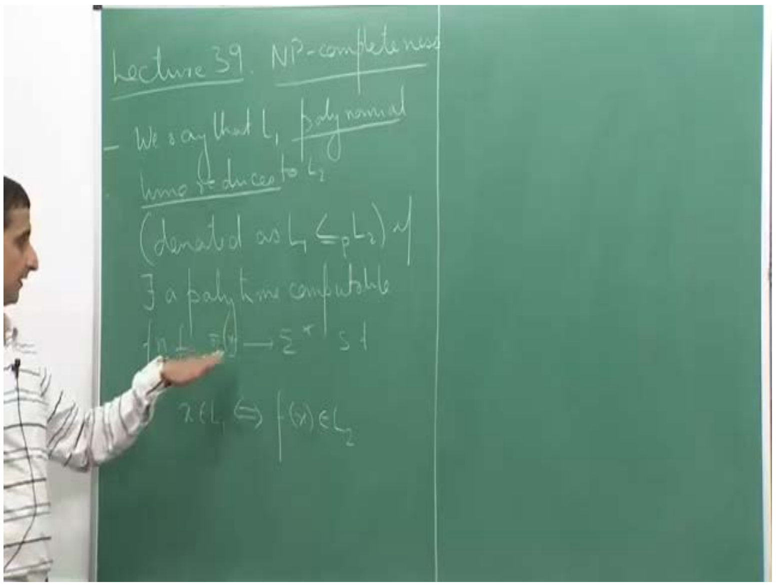
Theory of Computation

Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 39
NP-Completeness

Welcome to the 39th lecture of this course. Today, we will discuss the topic of NP-Completeness. So, last time I mention that, there are these problems in NP, which are in some sense the hardest problems in NP. So, how do we characterize these problems and how do we will define hardness of a problem? So, that is going to be our agenda today.

(Refer Slide Time: 00:40)



Lecture 39. NP-completeness

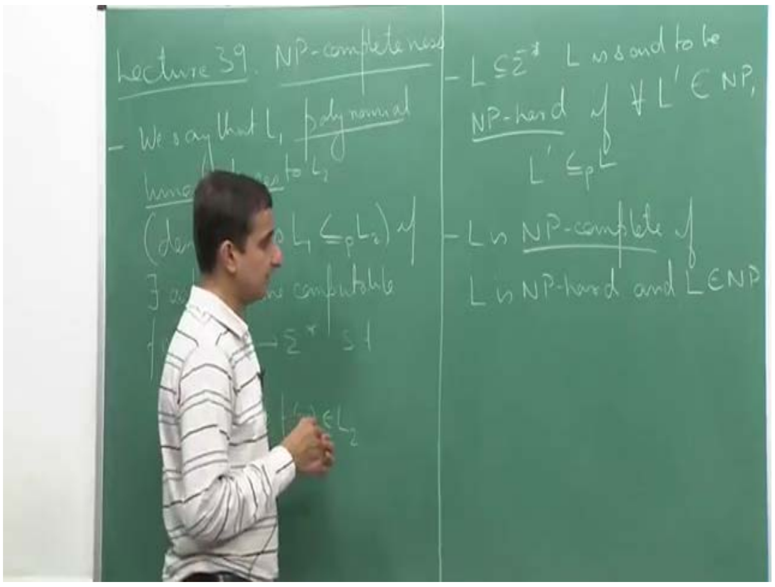
- We say that L_1 polynomial time reduces to L_2 (denoted as $L_1 \leq_p L_2$) if $\exists$ a poly time computable fn $f: \Sigma^* \rightarrow \Sigma^*$ s.t.

$x \in L_1 \Leftrightarrow f(x) \in L_2$

We have seen the definition of reduction, reducing 1 language to another. So, today we are going to find tune the definition and talk about polynomial time reduction. Let say, L 1 and L 2 are 2 languages. So, we say that L 1 polynomial time reduces to L 2 and this is denoted as L 1 with a subscript p, if there exist a poly time computable function f from sigma star to sigma star such that x belongs to L 1, if and only if f of x belongs to L 2.

The only difference is that, we say that L 1 polynomial time reduces to L 2, if the function f is a poly time computable function. So, the difference is that, we have a turing machine. We say that a function is computable, if there is a turing machine that outputs the value of f of x when given x. Now, in addition, if we restrict the turing machine to run for a polynomial number of steps in the length of x then we say that the function f is poly time computable, so that is the only difference the rest of the definition is similar to that of reduction.

(Refer Slide Time: 02:58)



Lecture 39. NP-completeness

- We say that L_1 polynomial time [illegible] to L_2 ([illegible] $L_1 \leq_p L_2$) if $\exists$ a [illegible] computable f [illegible] $\rightarrow \Sigma^*$ s.t. [illegible] $f(x) \in L_2$

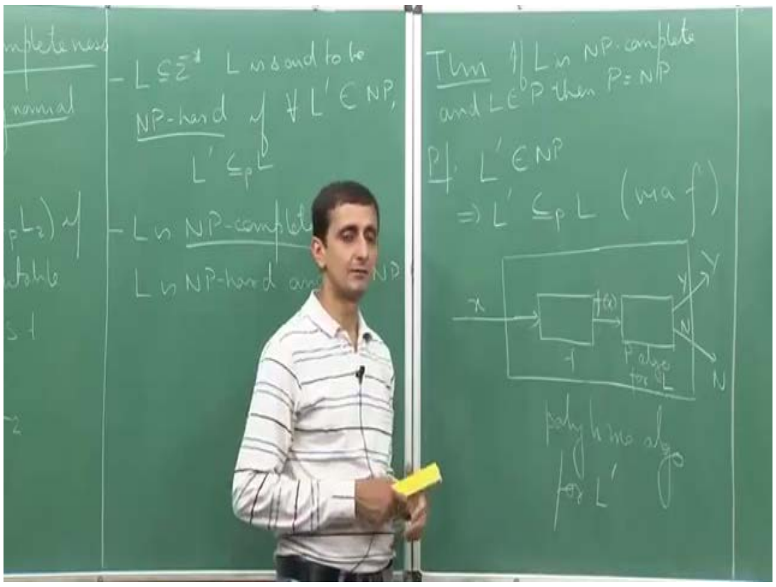
- $L \subseteq \Sigma^*$ L is said to be NP-hard if $\forall L' \in NP$, $L' \leq_p L$

- L is NP-complete if L is NP-hard and $L \in NP$

Let L be some language. L is said to be NP-hard if for all languages L prime belonging to NP, L prime reduces to L l prime polynomial time reduces to l. In other words, we say that language L is NP-hard if every other language in NP or every language including L itself must reduce to L in polynomial time. In other words, there is a polynomial time computable function which maps instances of L prime to instances of L.

Now, we say that L is NP-complete, if L is NP-hard and L belongs to NP. So, if a language is NP-hard it need not necessarily lie in NP. So, maybe it lies outside NP that can very well happen. So, all we say here is that every language in NP reduces to l, but L can lie anywhere, but we say that it is NP-complete if it is NP-hard as well as it belongs to NP. So, this is our definition of the highest level of hardness inside NP in some sense. So, basically if 1 language if we have an NP-complete problem if we have a language which is NP-complete then of course, it belongs to NP by definition and every other language in NP can be reduce to this language. So, what are some of the properties that we can say about NP-complete problems?

(Refer Slide Time: 05:25)

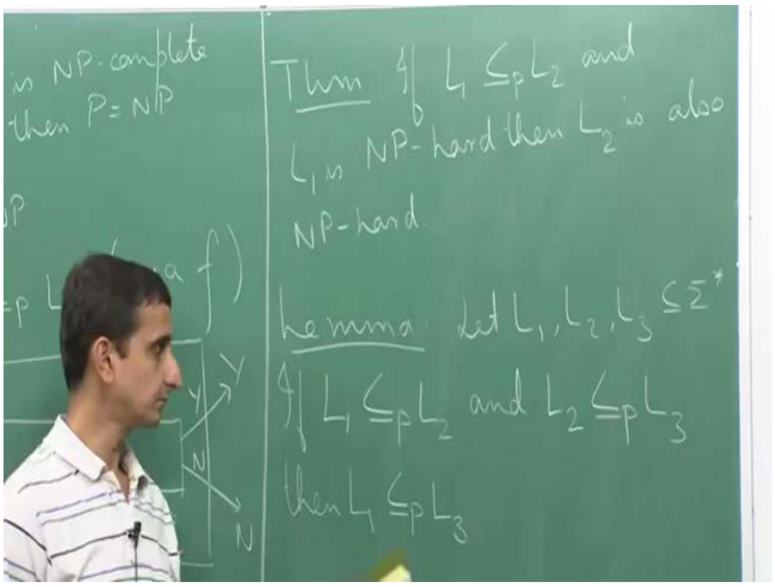


The first result is state it as a theorem. So, if L is NP-complete or even NP-hard and L belongs to P then P equals NP. In fact, this is probably the most popular strategy to show that PN and P are the same, you take an NP-complete problem and you give a polynomial time algorithm for that problem. If we can do that then we have would have shown that P is equal to NP and the proof is quite easy. So, the idea is as follows.

Basically take any problem in NP. So, let L prime be a problem in NP, because L is NP complete it means that L is NP-hard. So, we have that L prime reduces to L and now because L belongs to P, we have that L prime is also belongs to P the reason is. Suppose, if I take; if I want to show that y is L prime in p. So, take a box, this is poly time algorithm for L prime. So, given an input x, look at this reduction. This reduction is via some function f. So, via f say, first compute f of x. So, given x, feed x to this box and then compute f of x, now if f of x, what is f of x? f of x is an instance of the language L and our assumption is that L belongs to P.

Now, I apply that box on top of this. So, we make it bigger, I get f of x and then I apply the box for. So, this is P algo for the language l. So, I apply the box to this. Now, if this guy output S then I output S and if this guy outputs no then I output a no. If I am given an input x I first convert it into f of x, f of x is an instance of L because of this reduction and we know that L belongs to p. So, there is a polynomial time algorithm for l. Now, I just check if f of x is in L or not if f of x is in L I output S, if it is not in L I output no and that this total algorithm consist of 2 parts, first computing f of x and then applying the algorithm for L on f of x and both these algorithms are polynomial time hence the total box also takes polynomial time.

(Refer Slide Time: 09:09)



Thm If $L_1 \subseteq_p L_2$ and L_1 is NP-hard then L_2 is also NP-hard

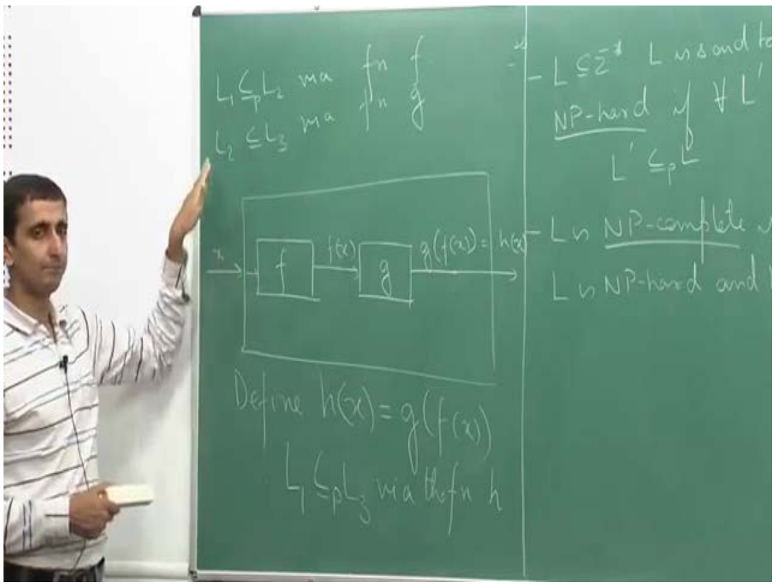
Lemma Let $L_1, L_2, L_3 \subseteq \Sigma^*$
If $L_1 \subseteq_p L_2$ and $L_2 \leq_p L_3$
then $L_1 \subseteq_p L_3$

So, this is a strategy to show that a problem, if you want to show that P is equal to NP then here is a strategy to that 1 can use.

Now, I will discuss a mechanism to show problems to be NP-complete and this theorem is similar to the way in which we showed that language is are un decidable that is if L 1 polynomial time reduces to L 2 and L 1 is NP-hard then L 2 is also is also NP-hard. So, if I have a reduction L 1 to L 2 such that the first language is NP hard then it implies that the second language is also NP hard. So, to prove this what I will do is I will use a lemma. So, this is again a lemma that is of interest by itself. Suppose, we have three languages L 1, L 2 and L 3, let L 1 L 2 and L 3 are 3 languages. So, if L 1 reduces to L 2 and L 2 reduces to L 3 then L 1 reduces to L 3 also.

In some sense, reduction is transitive in nature. So, if 1 reduces to 2 and 2 reduces to 3 then 1 reduces to 3. So, how do we prove this?

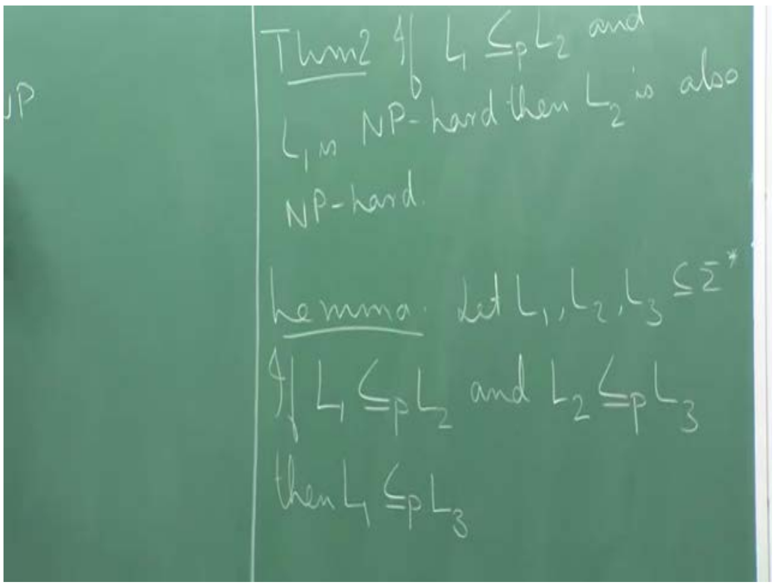
(Refer Slide Time: 11:45)



Suppose that L 1 reduces to L 2 via a function f and L 2 reduces to L 3 via a function g. Now, I want to reduce L 1 to L 3. So, I have to give a function. What is the function going to be? So, this function is going to be nothing, but a composition of f and g. So, first given an input x, I apply f to it. This is my box f, this produces f of x. So, OBS observe that if x is in L 1 then f of x is in L 2 and if x is not in L 1 then f of x is not in L 2 by the definition of reduction.

Now, I feed f of x to the box for g. So, g is also a polynomial time computable function. There is a polynomial time algorithm for g and this will give me g of f of x and I just output this as the answer. So, let us call this as h of x. So, define h of x to be g of f of x and therefore, L 1 reduces to L 3 via the function h, why is that? So, take an x which belongs to L 1. So, if x is in L 1 then f of x is in L 2, if f of x is in L 2 then g of f of x is in L 3. Hence, that is the same as h. So, therefore, h of x is in L 3.

(Refer Slide Time: 14:11)



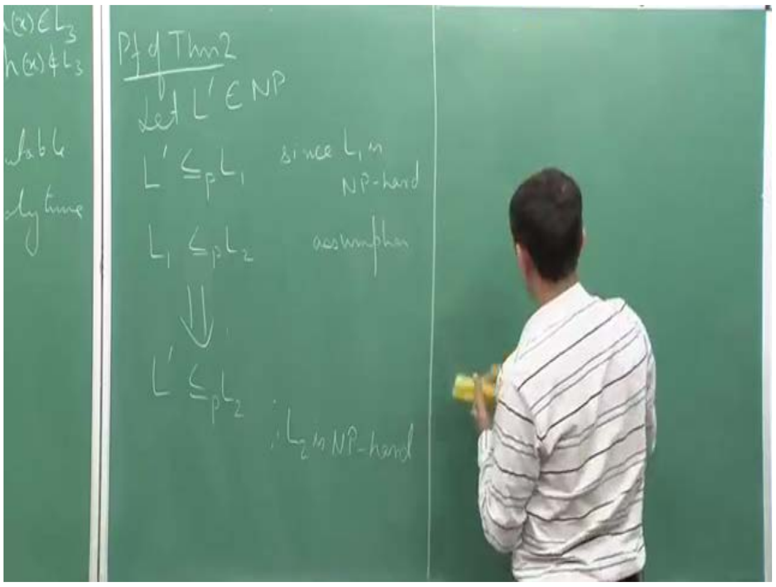
Thm2 If $L_1 \leq_p L_2$ and L_1 is NP-hard then L_2 is also NP-hard.

lemma. Let $L_1, L_2, L_3 \subseteq \Sigma^*$ If $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ then $L_1 \leq_p L_3$

So, if x belongs to L 1 then f of x is in L 2 which implies that h of x will be in L 3. On the other hand, if x does not belong to L 1 then by the definition f of x will not belong to L 2 and if f of x is not in L 2 then g of f of x is not in L 3. Therefore, h of x is not in L 3 and why is h computable in polynomial time. So, we also have to prove that. So, note that h is nothing, but applying f first and then g.

This takes some polynomial number of steps; the total is some other polynomial. So, h is poly time computable since f and g are poly time computable. Now, we will use this fact to prove our theorem. So, this was the lemma that we showed this is the proof of the lemma. Now let us go back to the proof of this theorem. This is what we want to show that if L 1 reduces to L 2 and L 1 is NP-hard then L 2 is also NP-hard. So, what I have is. So, what I want to show is that L 2 is NP-hard.

(Refer Slide Time: 16:30)



Pf of Thm 2

Let $L' \in NP$

$L' \leq_p L_1$ since L_1 is NP-hard

$L_1 \leq_p L_2$ assumption

$\Downarrow$

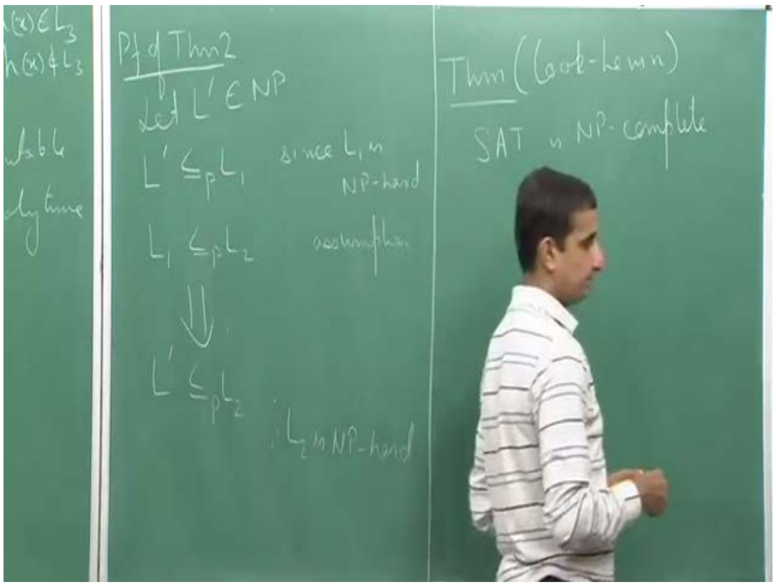
$L' \leq_p L_2$ $\therefore L_2$ is NP-hard

So, let me call this proof of let us give this a name theorem 2. So, proof of theorem 2. So, take any language in NP. So, let L prime be a language in NP to show that L 2 is NP-hard I have to show that L prime reduces to L 2.

Now, take we have a language in NP and we know that L 1 is NP-hard. So, we have that L prime reduces to L 1 since L 1 is NP-hard and of course, we also have another statement that L 1 reduces to L 2 we have that L 1 reduces to L 2. So, this is our 1 of the hypothesis or that I call it hypothesis I will just call it assumption. Now, we apply the lemma that we have L prime reduces to L 1 and L 1 reduces to L 2. Therefore, combining both these we have that L prime reduces to L 2. This proves that L 2 is NP-hard as well. So, this is what we actually wanted.

Now, we have a mechanism to show languages are NP-hard, but then we need a first NP-hard problem because we need to. So, to show that language is NP-hard I need to at least pick some NP-hard problem and then reduce gives a reduction from that problem to my given problem. So, what is the first NP-hard problem? So, this was a famous result again in computer science due to Cook and Levin. So, this is called the Cook Levin theorem.

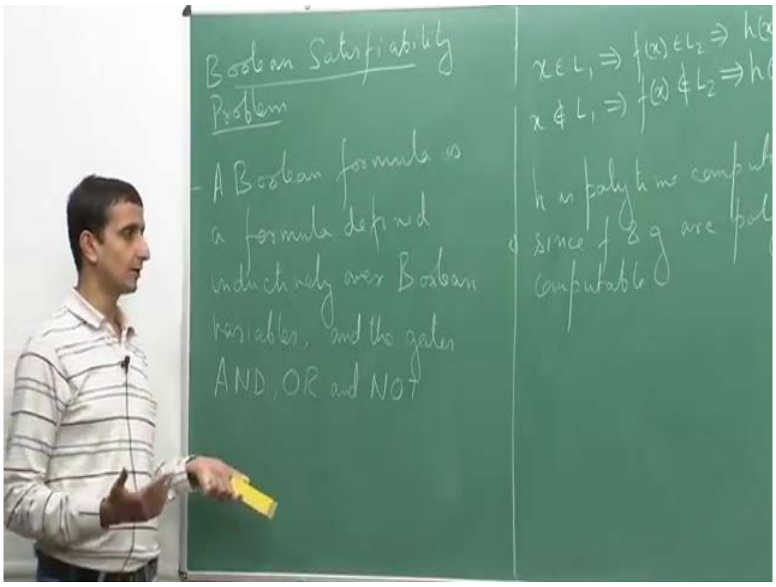
(Refer Slide Time: 19:11)


Pf of Thm 2
Let $L' \in NP$
$L' \leq_p L_1$ since L_1 is NP-hard
$L_1 \leq_p L_2$ assumption
$\Downarrow$
$L' \leq_p L_2$
$\therefore L_2$ is NP-hard
Thm (Cook-Levin)
SAT is NP-complete

There actually independently proved the following that there is this problem called the satisfy ability problem. SAT is NP-hard and also it is in NP. What it implies is that SAT is NP complete. So, what we will do next is that we will look at the definition of this problem. So, what is the Satisfiability problem? Remember that to show the first language to be NP-hard, we have to somehow argue that every other language in NP reduces to this language.

So, they can be infinitely many languages in NP and we cannot do it one by one. So, we have to give a generic strategy. So, the strategy was to take a NP machine. So, take any language that is in NP consider the machine for that language look at it is ah configurations how it how they behave and using that somehow argue that this problem SAT is NP hard. Let us first understand what the problem SAT is?

(Refer Slide Time: 20:42)



Boolean Satisfiability Problem

- A Boolean formula is a formula defined inductively over Boolean variables, and the gates AND, OR and NOT

$x \in L_1 \Rightarrow f(x) \in L_2 \Rightarrow h($

$x \notin L_1 \Rightarrow f(x) \notin L_2 \Rightarrow h($

h is poly time comput

since f & g are pol

computable

The Boolean Satisfiability problem - A Boolean formula is a formula defined inductively over Boolean variables and the gates – AND, OR and NOT. So, I am not going to give the precise definition of what a Boolean formula is, but I am going to give it with an example.

So, I am assuming that you have seen what a Boolean formula is at some point of your curriculum, but if not I would suggest that you go back and I mean refer to it and find out what is the exact definition, but what I am going to do is I am going to give some examples which would make it clear. So, here are some examples of Boolean formula.

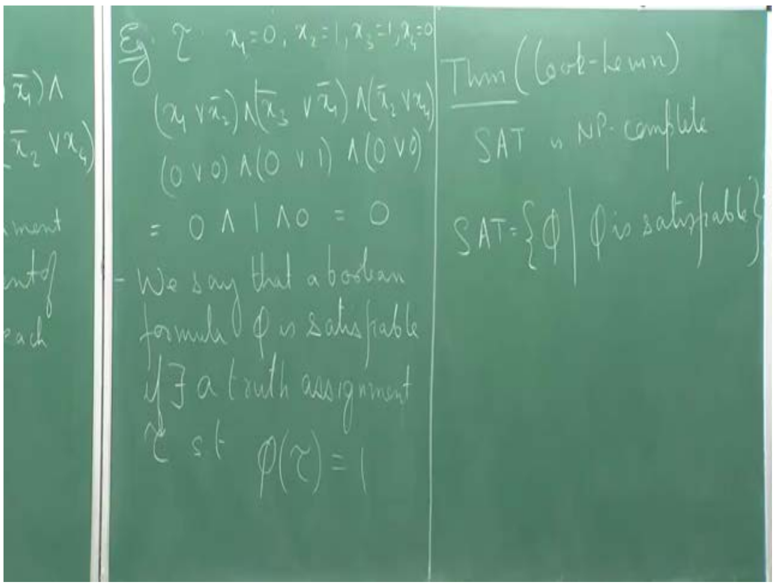
(Refer Slide Time: 22:40)

Boolean Satisfiability Problem
- A Boolean formula is a formula defined inductively Boolean variables gates AND
Eg: $(x_1 \vee \bar{x}_2) \wedge x_3$
$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_3 \vee \bar{x}_1) \wedge (\bar{x}_2 \vee x_4)$
- A truth assignment τ is an assignment of values 0 or 1 to each Boolean variable
Pf of Th

So, it is basically a formula that is defined over Boolean variables, Boolean variables are variables that can take only 2 values 0 and 1 and gates AND, OR and NOT. For example I can have something like x 1 or x 2 bar and let say x 3. This is a Boolean formula. So, x 1, x 2 and x 3 are Boolean variables and I have. So, this is a or gate this is an and gate and this is a a not gate. I can also have another formula; let us look at another formula x 1 or x 2 bar and x 3 bar or x 1 bar and x 2 bar or x 4. Here I have a Boolean formula that is defined over four variables. So, there are four variables x 1, x 2, x 3 and x 4 and it is written in this manner. So, it is defined.

Basically, I said inductively because at the base level the variables are also Boolean formula and then I combine Boolean formula with the help of the Boolean operators. So, the or of 2 Boolean operators is a Boolean formula and of 2 Boolean formula is a Boolean formula and not is a operator which takes only 1 argument. So, not of a Boolean formula also gives a Boolean formula and I use brackets to define precedence. So, which part gets precedence? This is similar to arithmetic expressions also. Now, I will define what is truth assignment; tau is an assignment of values 0 or 1 to each Boolean variable.

(Refer Slide Time: 25:30)



$\bar{x}_1)\wedge$

$\bar{x}_2 \vee x_4)$

ment

nt of

each

Eg: τ: $x_1=0, x_2=1, x_3=1, x_4=0$

$(x_1 \vee \bar{x}_2)\wedge(\bar{x}_3 \vee \bar{x}_1)\wedge(\bar{x}_2 \vee x_4)$

$(0 \vee 0)\wedge(0 \vee 1)\wedge(0 \vee 0)$

$= 0 \wedge 1 \wedge 0 = 0$

- We say that a boolean formula ϕ is satisfiable if $\exists$ a truth assignment τ s.t. $\phi(\tau) = 1$

Thm (Cook-Levin)

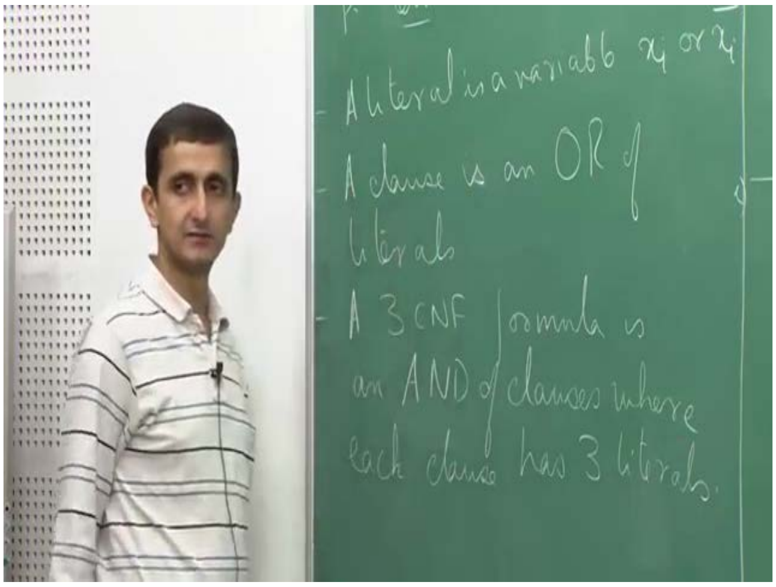
SAT is NP-complete

SAT $= \{\phi \mid \phi$ is satisfiable$\}$

For example, if I look at let say if I look at a truth assignment tau which is define which is giving let say x 1 equal to 0, x 2 equal to 1, x 3 equal to 1 and x 4 equal 0. So, under this assignment, what is the value of this Boolean formula, because x 1 equal to 0 and x 2 equal to 1. The first argument I mean this the Boolean formula within the first parenthesis x 1 is 0. So, let me write it down. If I have x 1 or x 2 bar and x 3 bar or x 1 bar and x 2 bar or x 4. So, x 1 is 0 x 2 is 1, x 2 bar will be 0. This is 0 or 0 with x 3 bar is 0 x 1 bar is a 1 x 2 bar is a 0 is 0 and x 4 is a 0. So, lot of 0s, now if I evaluate this, this is. So, 0 or 0 is a 0 0 or 1 is a 1 and 0 or 0 is again a 0 and now I have and of 0 1 and 0 which is equal to 0. So, under this truth assignment the Boolean formula evaluates to 0.

We say that a Boolean formula, phi is satisfiable if there exist a truth assignment tau such that phi of tow evaluates to 1 and now we can define the problem SAT. So, SAT is basically the set of all formulas phi such that phi is satisfiable. We can define a few more things we can define another language as well. So, let me define it here.

(Refer Slide Time: 29:05)

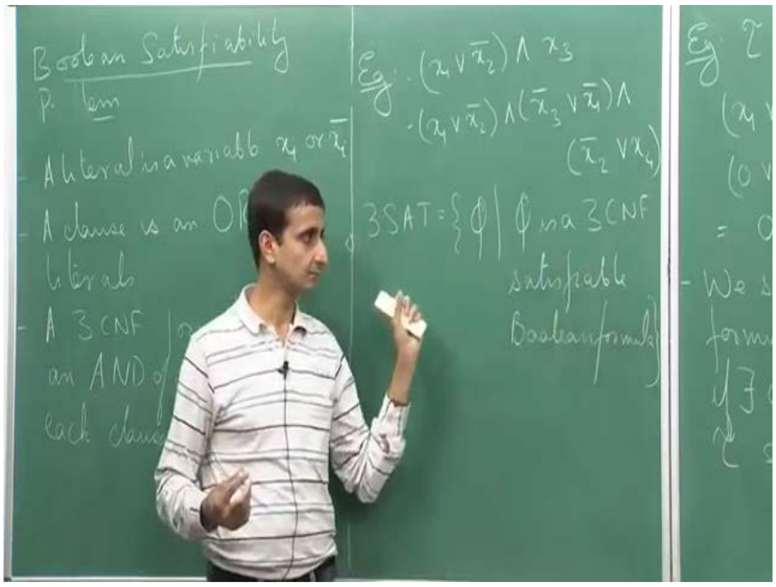


- A literal is a variable x_i or $\bar{x}_i$
- A clause is an OR of literals
- A 3 CNF formula is an AND of clauses where each clause has 3 literals.

A literal is a variable xi or xi bar. So, it is a variable or it is compliment a clause is an OR of literals, for example, in this Boolean formula this is a clause this is a clause and this is a clause a 3 CNF formula is an and of clauses where each clause has three literals, for example, this is a 2 CNF formula because each clause note here has 2 literals this 1 has 2 literals this 1 has 2 literals and the third clause also has 2 literals.

Similarly, if I have three literals it is called a 3 CNF formula and now I can define another problem which is similar to the satisfiability problem.

(Refer Slide Time: 30:42)



Boolean Satisfiability Problem

Eg: $(x_1 \vee \bar{x}_2) \wedge x_3$

$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee x_4)$

3SAT = { ϕ | ϕ is a 3CNF satisfiable Boolean formula }

It is known as the three SAT problem which states that phi such that phi is 3 CNF satisfiable Boolean formula. So, this is a variant of satisfiability where the Boolean formula is restricted to be a formula which has this particular form. So, it is a 3 CNF form. So, I will stop here today, we will continue more on NP completeness in our next lecture.

Thank you.