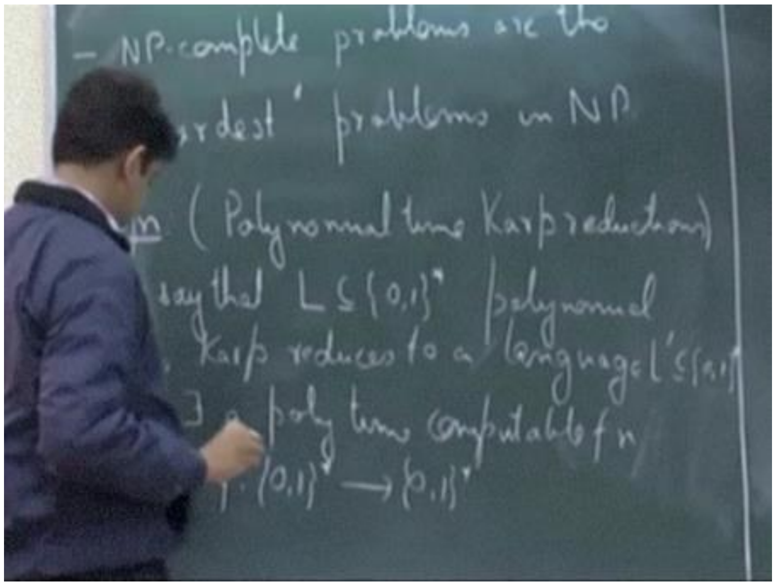
Computational Complexity Theory

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

Lecture -02
Introduction

So let us get started, so last time we were discussing the class NP and we saw certain examples of problems that are there in NP. So let us continue with our discussion. So what we will see today is this notion of completeness, completeness of a complexity class.

(Refer Slide Time: 00:46)



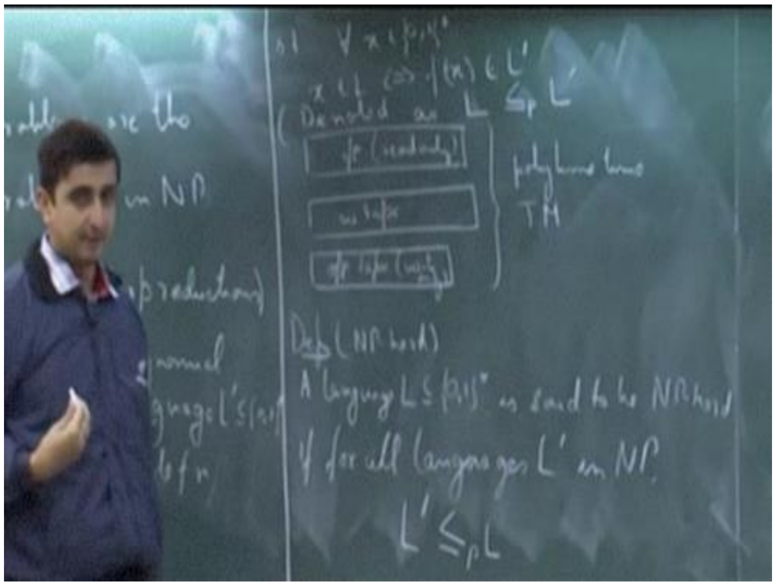
– NP-complete problems are the
rdest' problems in NP
n (Polynomial time Karp reductions)
say that $L \subseteq \{0,1\}^*$ polynomial
Karp reduces to a language $L' \subseteq \{0,1\}^*$
$\exists$ a poly time computable fn
$f: \{0,1\}^* \to \{0,1\}^*$

So let us look at the class NP and see what we mean by NP completeness. So intuitively NP complete languages or NP complete problems are those problems in NP that are the hardest problems in NP. So NP complete problems are in some sense the hardest problems in NP, so we need to define a notion of hardness. So what do we mean by hardness? So whenever we talk about hardness or easiness we do so in relative terms.

So something is hard means it is hard with respect to some other stuff. So we need to define a notion of relativity here and so that comes from the notion of reductions. So let us look at reductions and formally define them. So we will see what are known as a polynomial time Karp reductions. So the book uses this term polynomial time Karp productions to denote these kind of reductions but in other texts or in other places you might find things like polynomial time many one reduction and or just polynomial time reductions and they typically mean the same thing.

So what are these reductions? So let we say that a language time L, polynomial time Karp reduces to a language. Let us say L prime, if there exists a polynomial time computable function f. So f basically takes strings and outputs string just write it here triple function f from 0, 1 star to 0, 1 star.

(Refer Slide Time: 04:50)


s.t. $\forall x \in \{0,1\}^*$
$x \in L \iff f(x) \in L'$
(Denoted as $L \leq_p L'$
i/p (read only)
w. tape
o/p tape (write only)
polynomial time TM
Def (NP hard)
A language $L \subseteq \{0,1\}^*$ is said to be NP hard
if for all languages L' in NP,
$L' \leq_p L$

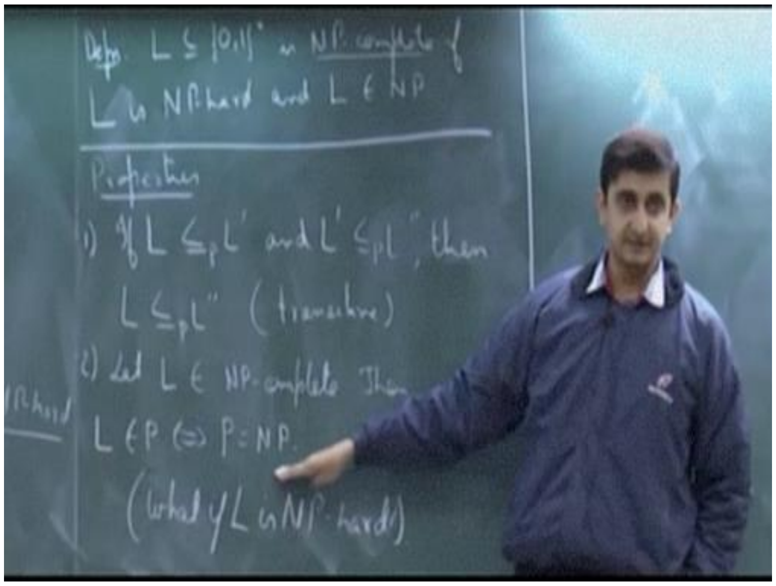
Such that for all strings x, x is in L if and only if f of x is in L prime. So basically f is mapping the s instances of the language L to the s instances of the language L prime and the no instances of language L to the no instances of language L prime. It means that there is a polynomial time Turing machine which given a string, let us say it is given the string x it will output the string f of x on its output tape.

So we consider the following models. So our Turing machine has these three tapes, so it has an input tape and the input tape is read only it has a all powerful work tape, which means that you can read as well as write on to this tape and then it has an output tape and this is just write only so you cannot read information of this tape. So now this is a polynomial time machine which when given the input x on its input tape it will output f of x and then halt.

So now that we have the tool of reductions with us let us define what we mean by these so called hard sets. So first we will define this class known as NP hard. So a language is said to be NP hard if for all languages L prime in NP L prime reduces to L. So this is the notation that we use for these kinds of reductions. So let me just mention it here, so I just write it here denoted as so here we were reducing L to L prime so that is denoted as L reduces to L prime and this subscript p means that it is a polynomial time reduction.

So we say that a language is NP hard if all other languages in NP or any language in NP reduces to L in polynomial time.

(Refer Slide Time: 08:56)



Defn. $L \subseteq \{0,1\}^*$ is NP-complete if
L is NP-hard and $L \in NP$

Properties

1) If $L \leq_p L'$ and $L' \leq_p L''$, then
$L \leq_p L''$ (transitive)

2) Let $L \in$ NP-complete. Then
$L \in P \Leftrightarrow P = NP$.
(What if L is NP-hard?)

And, we say that L is NP complete if L is NP hard and L belongs to NP. So it is not necessary that all NP hard languages will belong to NP in fact you can very easily find v hard languages that are not in NP I mean take some really hard language may be an undecidable problem an undecidable language. Of course, every other language in NP will reduce to it but there is no algorithm in particular there is no NP algorithm for that problem.

So NP completeness in some sense characterizes the most hard languages in NP up to polynomial time reducibility, any questions? So, let us look at some easy to prove properties of these concepts firstly if L polynomial time reduces to L prime and L prime polynomial time reduces to let us say some language L double prime then L also reduces in polynomial time to the language L double prime in other words this operation the polynomial time reduction operation is transitive. So this is known as the transitivity property.

So why is this true? Because if you take; a polynomial and look at its product with another polynomial what you get is also a polynomial. Let us look at another property. So let L be an NP complete problem then L is in P, if and only if P is equal to NP, again this is easy to see because if you take any NP complete problem and if you assume that language is in p then all problems in NP will also reduce to that particular language.

And, then by using that the p algorithm for this language you can show that all of NP is contained in p this is again by the same idea and the other way is also true. So if p is equal to NP it means that all problems in NP line p in particular the NP complete problems also lie within p so both directions are true, well so it depends how you are looking at the reduction. So if you just assume that you have access to many tapes then what you are saying is correct?

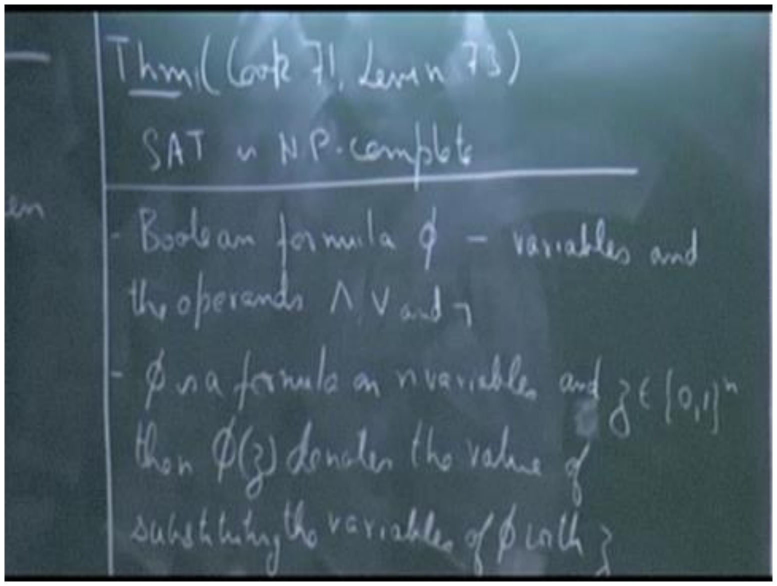
You can just look at the sum because then what you do is that you have a machine which computes this reduction let us say m1 and you have another machine m2 for this reduction you construct a third machine m we just takes the output of m one feeds it onto the input of m2 and then just outputs the output of m2 but it is using one extra tape for that but suppose you do not want to make use of that extra tape.

Suppose, you have only one work tape you want to construct a third machine which only has one work tip then what you do is that for each bit that is required by m2 you go back to m1 and you do the entire computation of m1 get what that bit is and then carry on with the computation of m2. So you are doing some extra work but just cutting off the use of an extra tape so you are right.

So what if we assume L to be NP hard here? Can we make the same statement? Can we still have the same conclusion? Why not, so basically we can conclude one of the directions. So if we assume L to be NP hard then if L is in p it would imply p is equal to NP but the other way need not necessarily it will be true. So if L is NP hard and p is equal to NP it need not imply that L will also belong to p because as you said it is a wider set.

So, let us move on and so what we will see next is our first NP complete problem in fact we will construct the first NP complete problem.

(Refer Slide Time: 16:29)



Thm: (Cook 71, Levin 73)

SAT is NP-complete

- Boolean formula ϕ – variables and the operands $\wedge$, $\vee$ and $\neg$
- ϕ is a formula on n variables and $z \in \{0,1\}^n$ then $\phi(z)$ denotes the value of substituting the variables of ϕ with z

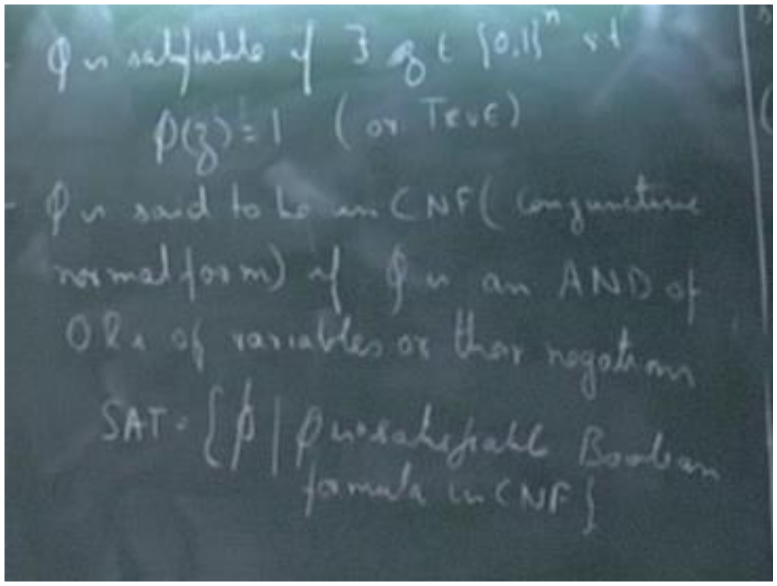
Because, so far I mean with all these theory I mean nowhere are we saying that there has to exist an NP complete problem I mean it might turn out that I mean even proving existence or constructing an NP complete problem is not possible. So what is the guarantee that there are problems which have this property or there are natural problems which have this property. So we will prove that this problem satisfiability is the first as our first NP complete problem.

And, historically also when researchers were studying NP completeness and they came up with the first NP complete problem this is what they had shown. So this was proved in early 70s independently by two people Cook in 71 and Levin in 73. So cook was in a]America and Levin was in Russia so communication was not that good in that period so they both basically came up with the same result but independently they showed that SAT is NP complete.

So, let us quickly see what the problem sat is? So a Boolean formula phi basically consists of what? It consists of some variables and the operands AND, OR and NOT. So Boolean formula is basically a string of variables together with the following operands where the operands have the usual definition. Suppose you have a Boolean formula on n variables, so suppose phi is a formula on n variables and z is some n bit string then phi of z denotes the value of substituting the variables of phi with z in order.

So basically you take the first variable of phi wherever it occurs and you replace it with the first bit of z and so on. So whatever value that you get as a resultant is what phi of z is. So then when do we say that a formula is satisfiable?

(Refer Slide Time: 21:22)



ϕ is satisfiable if $\exists z \in \{0,1\}^n$ s.t.
$\phi(z) = 1$ (or True)

ϕ is said to be in CNF (Conjunctive normal form) if ϕ is an AND of ORs of variables or their negations

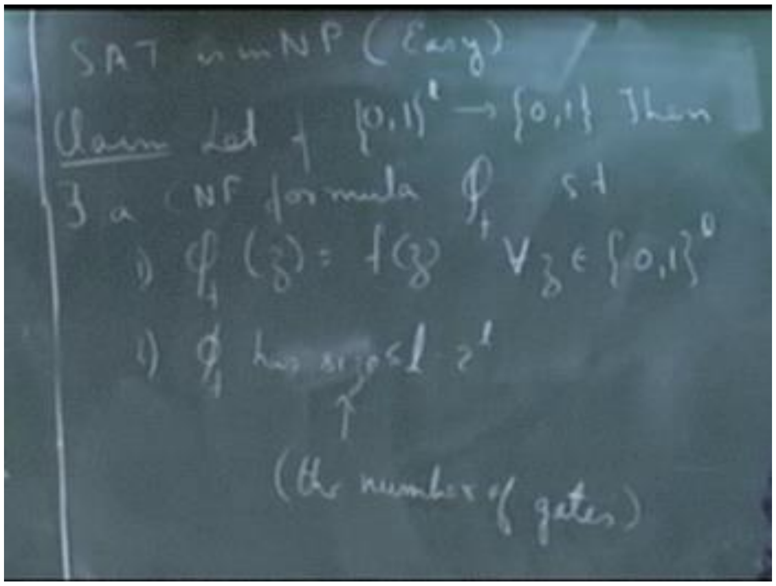
SAT = $\{\phi \mid \phi$ is satisfiable Boolean formula in CNF$\}$

So we say that a Boolean formula phi is satisfiable if there exists some string. So let us say we are looking at Boolean formulas over n variable if there exist a string z such that phi of z evaluates to one or sometimes will refer this to also as it evaluates to true I mean the same thing. Otherwise we say that phi is unsatisfiable and there is a special kind of Boolean formulas that we would focus at for the purpose of our proving this theorem that is formulas which are known as CNF formulas in conjunctive normal form.

So phi is said to be in CNF which stands for conjunctive normal form, if phi is an AND of ORs of variables or their negations. So this is what a CNF formula is and in fact what we will prove is that a Boolean formula which is in conjunctive normal form basically satisfies the language SAT will consist of all Boolean formulas in conjunctive normal form which are satisfiable. Set of all formulas such that phi is a satisfiable Boolean formula in conjunctive normal form.

So we will see this proof for the remaining part of our talk. So the first thing that we need to argue in order to show that SAT is NP complete is to show that SAT is in NP and that is easy to see so how do we prove that SAT is in NP. So what would that SAT equal to NP? It is just a string z corresponding to assignment of values to the variables of phi and if phi satisfiable then we know that such a certificate exist which would evaluate phi to true otherwise not.

(Refer Slide Time: 25:57)



SAT is in NP (Easy)

<u>Claim</u> Let $f: \{0,1\}^l \to \{0,1\}$ Then $\exists$ a CNF formula ϕ_f s.t.

1) $\phi_f(z) = f(z) \quad \forall z \in \{0,1\}^l$

2) ϕ_f has size $\leq l \cdot 2^l$

$\uparrow$

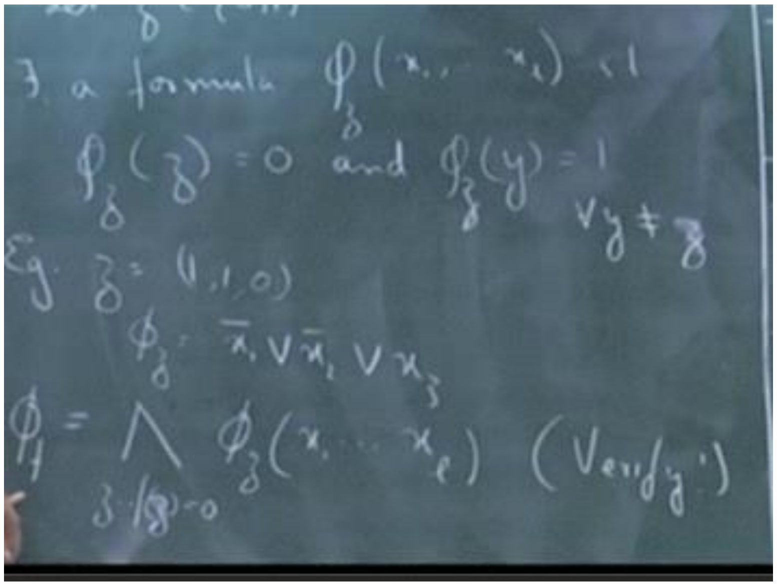
(the number of gates)

So SAT is in NP, so this is easy to see, the hard part is to show that it is NP hard. So before we go into the actual proof of that result let me state easier to prove claim. So let us look at the following claim first suppose we have a Boolean formula on L variables. So let f be a Boolean formula on L variables then there exist a CNF formula, let me call this phi of f just to denote that this formula will depend on this function such that it has two properties.

Firstly, phi of f on any input z is equal to f of z and second this formula has size l times 2 to the power l and by size what we mean is the number of gates in the circuit. So essentially what we are saying in this claim is that if you are given a Boolean function any arbitrary Boolean function from l variables to 0, 1 we can construct a corresponding CNF formula which will exactly replicate that function in the sense that it will have the same value for all strings of length l.

And, it will have size actually at most let me just say this it will have size at most l times 2 to the power l, how do we show this I will not give the complete proof but I will give the essence of it, you can always fill in the details.

(Refer Slide Time: 29:19)



$\exists$ a formula $\phi_z(x_1 \ldots x_l)$ s.t.

$\phi_z(z) = 0$ and $\phi_z(y) = 1 \quad \forall y \neq z$

Eg. $z = (1,1,0)$

$\phi_z = \bar{x}_1 \vee \bar{x}_2 \vee x_3$

$\phi_f = \bigwedge_{z : f(z)=0} \phi_z(x_1 \ldots x_l)$ (Verify!)

So suppose if you look at string, some string of length l. So let z be a string of length l then for z actually we can create a CNF not a CNF but actually we can create a formula which has the following structure there exists a formula let me call this a phi subscript z on l variables. Let us say from x 1 through x l such that phi z on z is equal to 0 and for any other y that is not equal to z it is 1 for all y not equal to z. So this is not difficult to see.

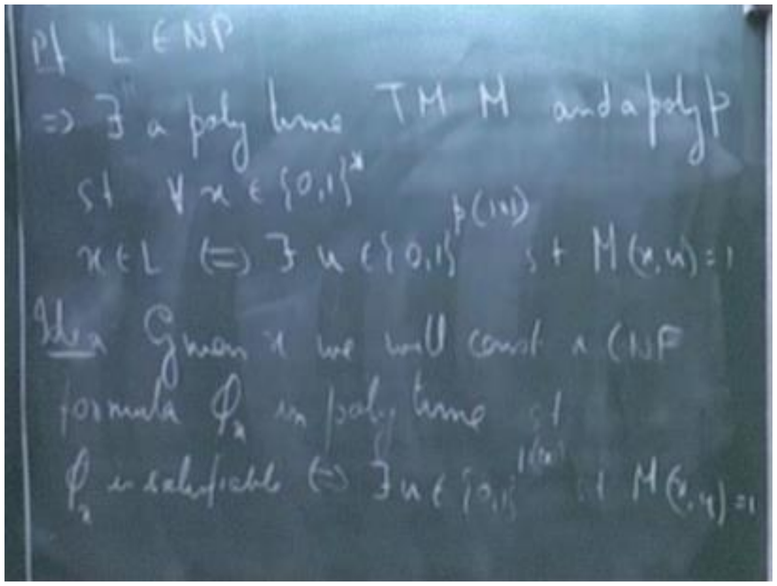
So any ideas how we can construct a formula like this? Exactly, so let us just see this with a small example so suppose if you have a string so suppose z is the string 1 1 0 your corresponding phi z will be the formula x 1 bar because I have a 1 here odd with x 2 bar odd with x 3. So the only string for which this formula would evaluate to 0 is z and nothing else.

So now that we can do this let us go ahead and construct the formula phi f. So what would phi f look like? So phi f would be the AND of it will be AND of all phi z's; let us say our variables x 1 through x l such that which z do we consider no so we hope so we look at all z such that f of z is zero. So I will not it is not difficult to verify I will just leave it as an exercise. So you can verify that this formula will have the property that for any string of length l it will be equal to f of that string and the other thing is about the size.

So now, so what would be the size of this thing? So, if you look at phi z so phi z will have size l and at most there can be two to the power l such strings of length l. So therefore the total length will be at most l times two to the power l, we can take, yes, but then it will not be a conjunctive normal form know, we need it to be in conjunctive normal form but yes you can take that also. In fact, there are many ways to do this.

So this is just one way of doing it but the point is that whichever way you pick you will see that always it will have I mean it will always require exponential size. So let us come back to the main claim.

(Refer Slide Time: 34:43)



Pf $L \in NP$

$\Rightarrow \exists$ a poly time TM M and a poly p

s.t. $\forall x \in \{0,1\}^*$

$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)}$ s.t. $M(x,u)=1$

Idea Given x we will const a CNF formula ϕ_x in poly time s.t.

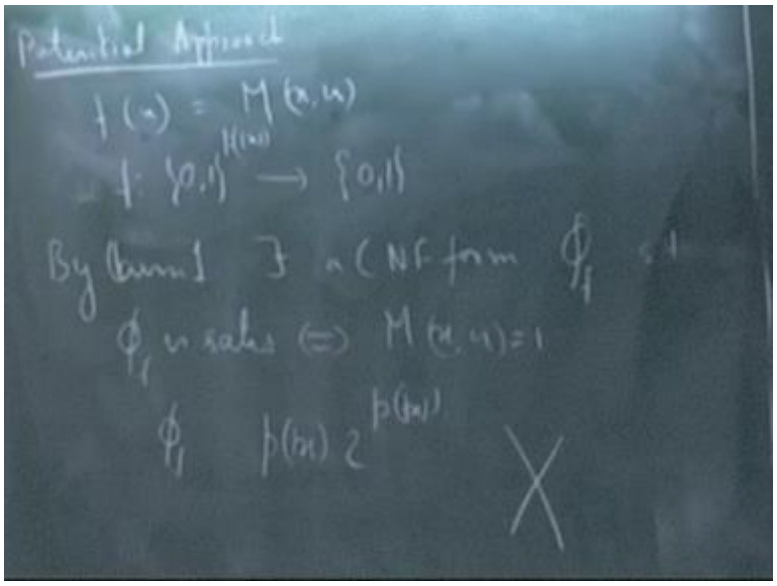
ϕ_x is satisfiable $\Leftrightarrow \exists u \in \{0,1\}^{p(|x|)}$ s.t. $M(x,u)=1$

So what does it mean to say that SAT is NP hard it means that if we take any language L in NP there should be some way of reducing L to SAT. So let us take a language L in NP this means that there exists a poly time Turing machine m and a polynomial p such that for all strings x, x is in L if and only if there exist some u of length p of x such that m of x, u is equal to 1. So what we will do is we will given this language L and given a string any arbitrary string x.

We will construct a formula phi of x such that if x belongs to L then phi of x will evaluate to true and otherwise it will evaluate to false so the idea is given x, we will construct a CNF formula phi of x in polynomial time such that phi of x is satisfiable if and only if the following happens if and only if there exists u. So there is a trivial way. So let me first discuss a method which will not work and then we will see how that can be modified.

So if you look at this claim and if you try to replicate this claim there is a very easy way in which we can come up with a formula phi of x. So what we do is so let us just consider the following function.

(Refer Slide Time: 38:43)



Potential Approach

$f(u) = M(x,u)$

$f : \{0,1\}^{p(|x|)} \rightarrow \{0,1\}$

By Claim 1 $\exists$ a CNF form ϕ_f s.t.

ϕ_f is satis $\Leftrightarrow M(x,u)=1$

$\phi_f \quad p(|x|)\, 2^{p(|x|)}$

X

Let me just call it potential approach. So let me consider a function f which does the following it takes strings u, so f basically takes a string u and it outputs whatever the value m of x, u would be so we can always define a Boolean function. So given a string x we define a function f from 0, 1 to the power p of x to 0, 1. So we can define a Boolean function like this and now you see that this f will have the property that if such and u exist then m of x u is equal to 1 and hence f of u is equal to 1 and if such a u does not exist then f of u will never be 1.

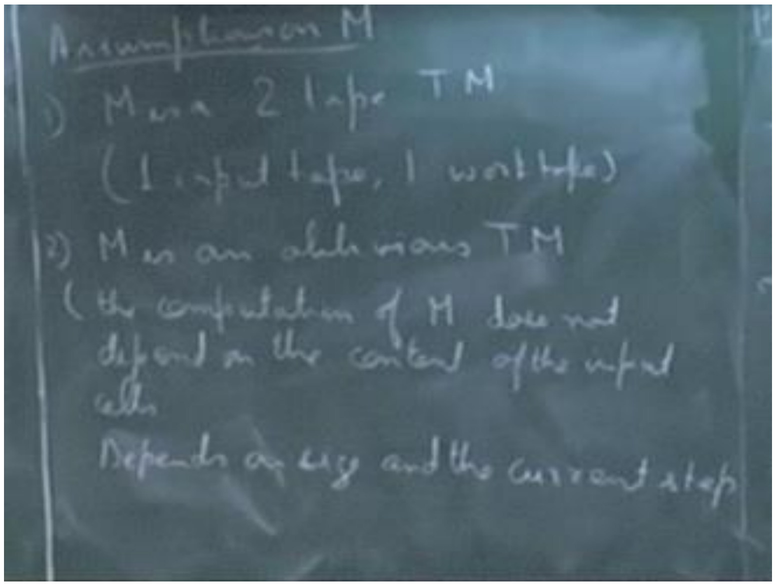
So now what we can do is given such a function f I can just replicate this claim. So by claim one so let us give a number to this claim so by claim one there exist a CNF formula phi of f such that phi of f is satisfiable and only if m of x u is equal to one. So we have come up with a CNF formula but what is the problem with this approach? So it will take exponential time and the reason for that is that phi of f what can be the potential size of phi of f?

If you just see what the claim says, so phi of f potentially can have size how large it will have size p of x times 2 raised to p of x. So we have a CNF formula but it has a very large size and hence this approach does not work but we were being very lazy in this approach in the sense that we never use the fact that m is a polynomial time deterministic machine. So that fact we never used and also we are not using another crucial fact that is the computation of m is very local.

In the sense that every time, so if you look at two steps of the Turing machine so going from one step to another step or going from one configuration to another configuration involves only a constant change in the configuration of machine. So it will go maybe from one state to another state and it will probably replace one bit with some other bit and so on. So we are just assuming that we have a two tape Turing machine so there is only a constant change that is happening.

So these two facts somehow need to be used and to come up with a CNF formula which has polynomial size. So let us look at the correct construction now. So we will assume two things about our Turing machine m and both these things can be assumed without loss of generality.

(Refer Slide Time: 43:30)



Assumptions on M

1) M is a 2 tape TM
(1 input tape, 1 work tape)

2) M is an oblivious TM
(the computation of M does not depend on the content of the input cells
Depends on size and the current step

Assumptions on m; so firstly we will assume that m is a two tape Turing machine in the sense that it has a one input tape and one work tape and the second assumption that we will make. So if you looked at here the exercises that I gave last time regarding this assumption we can assume that m is oblivious, so m is what is known as an oblivious Turing machine. So what does this mean so this means that the computation of m does not depend on the content of the input cells.

So it only depends on the size of the input and the current step. So if the machine is taking the i th step no matter what is present in the tapes of that Turing machine it would always take the same step. So the transition function is a function of the size of the input length and the position of where the tape head is. It does not depend on the actual, which one, right. So the movement of the head is dependent only on the index of that head not on the content of that head for difference and yes it might be different of course.

So we will make these two assumptions and the fact is that even with these two assumptions it only creates a polynomial amount of overhead. So suppose if you have a Turing machine which is general enough a corresponding Turing machine which has these properties only creates a polynomial amount of extra overhead, so we are still fine. So I guess I will stop here today next class we will continue and will complete the proof of this theorem.