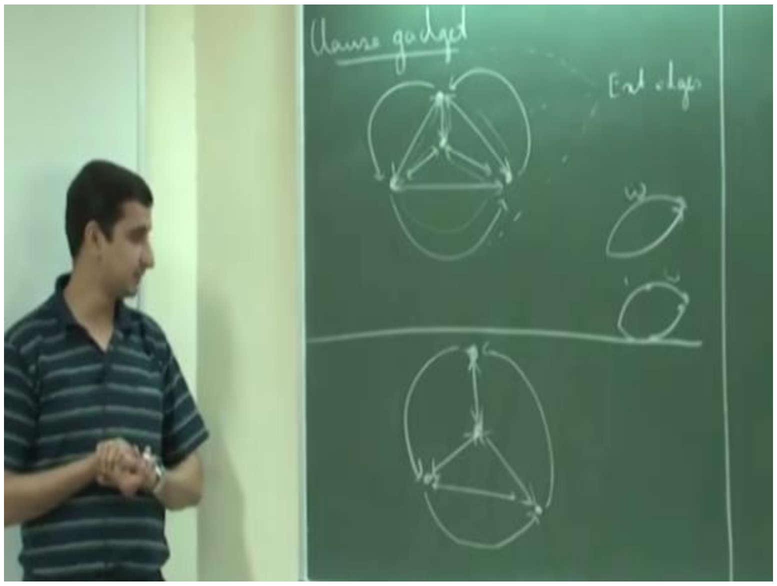
Computational Complexity Theory

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

Lecture -30
Introduction

(Refer Slide Time: 00:16)



Hello, this is the gadget that we had and what we argued. So, these are the three external edges. So, what we want is for every proper subset of the external edges there should be a unique cycle cover. That is what we want. So, what we saw last time is that this gadget has the property that for every proper subset of the external edges there exists a cycle cover. And if you take all the three external edges then of course, you will not get a cycle cover because this vertex will remain uncovered.

So, that we saw last time. So, the question was can we, so clearly so this gadget is not correct in the sense that it does not give us what we want. That will not harm. You can have multiple edges in the same direction and later on you can replace it with one edge having. So, one way to mean that is what you can do you can introduce an intermediate vertex here and here. So, suppose you have something like this.

You can replace it with a graph which looks like this and let us say that this edge had to weigh w.

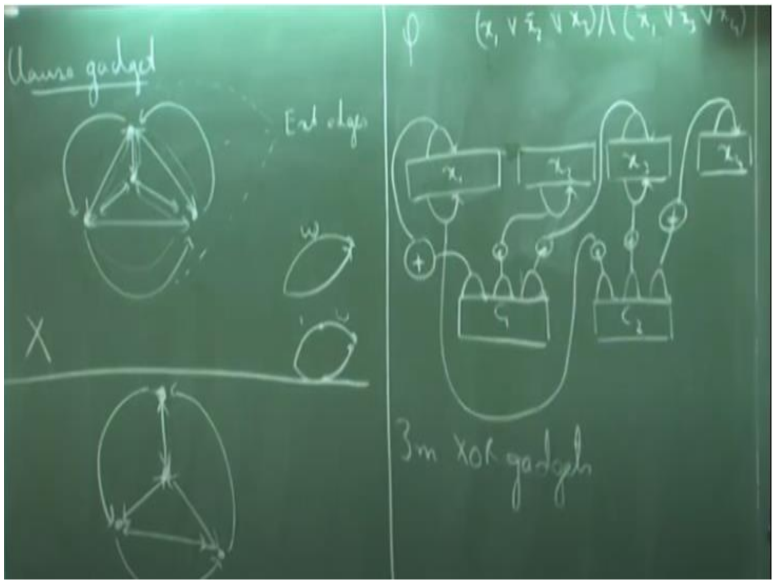
You give these two edges let us say the first edge you give one and the second edge you give w. So, the product still will be w but now you have taken care of multiple edges. So, getting rid of multiple edges is not a big deal. Or parallel edges that are easy. But the question is more basic. What we want is the clause gadget to have the property that for every proper subset of the three external edges.

There should be a unique cycle cover. And that is clearly not the case because suppose I take only this external edge I will have two cycle covers. So, one is I take this then this, this and this. This is one cycle cover and I can also take this. This, I go up and then come down. Yes it can consist of any number of cycles, but they should be vertex disjoint product of the weight of the edges. Weight of one cycle cover I think only two.

So, if I only have this external edge so vertices cannot be repeated in a cycle cover. So, you come here and but then this vertex and then you take these two. Yes, that is true. So, you have more than two actually. So, how do you have any idea how this can be fixed? So, let me give the solution. So, again the solution is just to remove these two edges. So, what we will have is I mean this is just one solution that I came up with. But I am sure there will be other solutions as well.

So, these are the three external edges and basically I have removed these two pairs of edges. So, these pair and these pair. So, I think this works. So, this is not the correct one. So, this let us just check. I mean maybe that is the best way.

(Refer Slide Time: 05:28)


Clause gadget
Ext. edges
X
ϕ $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4)$
x_1
x_2
x_3
x_4
C_1
C_2
3m XOR gadgets

So, suppose if we have none of the three external edges then our cycle cover will be basically taking these two and these two. Suppose if we have this edge then we can go this way. Yes, maybe I shall remove this edge. Does that help? That does not help. So, what construction did I have? That is true. If I have only this and suppose if we remove this edge I will check this. I am sure that something similar will work.

But again I am not able to get the correct solution at this point. But basically some subset of this graph actually works. So, I did have a construction earlier. So, what about the other cases? Suppose if we have only this edge? Then we have a unique cover. All the cycle covers here will have weight one because all the edges have weight one. No, I think it will matter. I mean, at least even if you look at the book's proof what they actually want is, they want a unique cycle cover for every proper subset.

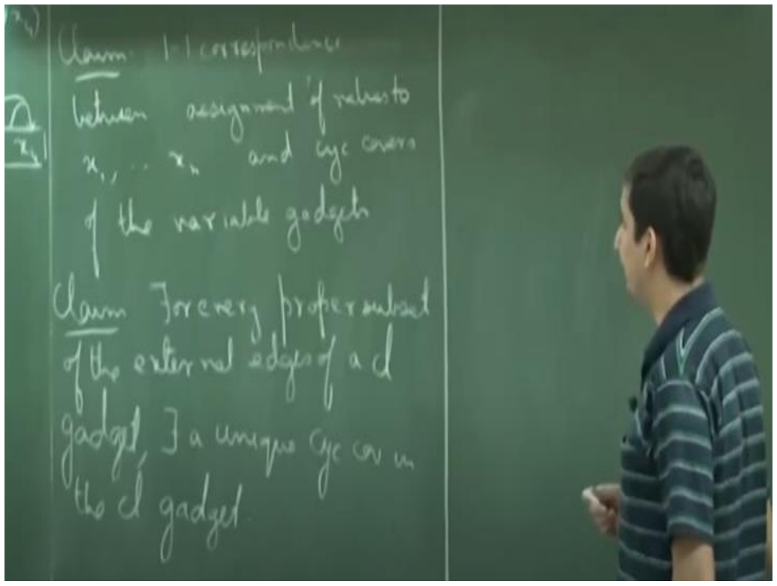
But anyway, I will try to fix that. But clearly what is there in the book is not correct. But I am sure it is fixable. So, let us just make that assumption for the time being and let us see how to argue the proof. So, now suppose we look at a small example to illustrate the idea. So, let us say that our boolean formula phi is a formula which looks like this x 1 or x 2 bar or x 3 and maybe x 1 bar or x 3 bar or x 4.

So, the corresponding graph that we will have is firstly we will have a variable gadget for each of the four variables. This is for x 1, x 2, x 3 and x 4 and we will have a clause gadget for the two clauses. So, now since x 1 appears once as true and once as negative form, we will have a true edge and a false edge. For x 2 we will have just one false edge. For x 3 again, we will have 1 true edge and 1 false edge and for x 4 just 1 true edge.

And now we need to connect them to the clause gadgets. So, all the clause gadgets have 3 external edges. So, c 1 gets connected to the true edge of x 1 and this happens using a XOR gadget to the false external edge of x 2 and to the true external age of x 3. Similarly for c 2 as well. So, this is the entire connection. See even for arbitrary numbers of variables and clauses you can see that this can be constructed.

So, now how do we get the number of cycle covers here? So, that needs to be argued. So, before we argue that how many XOR gadgets do we need? So, if we have n variables and m clauses how many XOR gadgets do we need? 3m. So, what did we argue earlier?

(Refer Slide Time: 12:37)



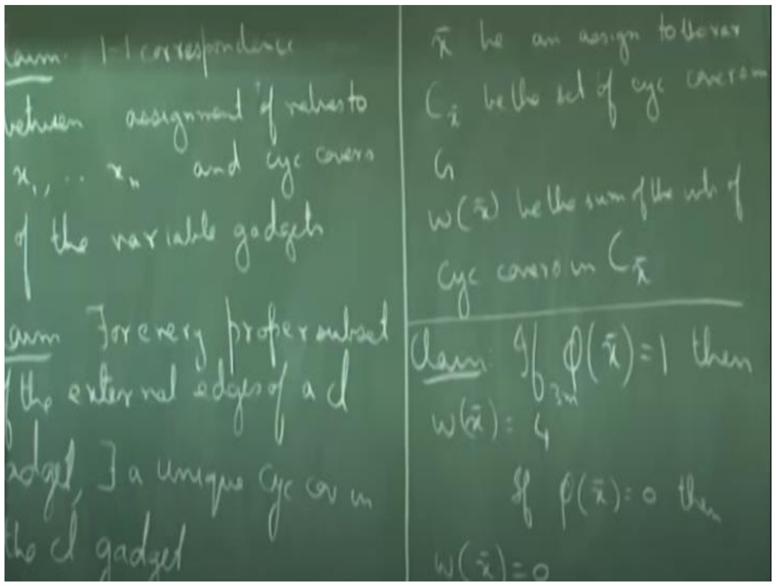
Claim: 1-1 correspondence between assignment of values to $x_1 \ldots x_n$ and cyc covers of the variable gadgets

Claim: For every proper subset of the external edges of a cl gadget, $\exists$ a unique cyc cov in the cl gadget.

So, what we said yesterday was that there is a 1 to 1 correspondence between assignment of values to x 1 up to x n and cycle covers of these gadgets of the variable gadgets. Because for each variable gadget there are exactly two cycle covers. Also, the other thing that we claim is that for every proper subset external edges of a clause gadget there exists a unique cycle cover in the clause gadget. And both these cycle covers are basically cycle covers of weight one because all the edges in the clause gadget as well as the variable gadgets have weight one.

So, any cycle cover also has weight one. It is only the XOR gadgets which introduce larger weights or negative weight cycle covers. So, this again as I said we will verify later on. But let us just assume that the second claim is true.

(Refer Slide Time: 15:37)



laim: 1-1 correspondence
etween assignment of values to
$x_1, .. x_n$ and cyc covers
of the variable gadgets

aim For every proper subset
the external edges of a cl
adget, ∃ a unique cyc cov in
the cl gadget

$\bar{x}$ be an assign to the var
$C_{\bar{x}}$ be the set of cyc covers in
G
$w(\bar{x})$ be the sum of the wts of
cyc covers in $C_{\bar{x}}$

Claim: If $\phi(\bar{x})=1$ then
$w(\bar{x}) = 4^{3m}$
If $\phi(\bar{x})=0$ then
$w(\bar{x})=0$

So, now let x bar be an assignment to the variables of the formula and for this assignment let C x bar be the set of cycle covers in G. So, I use the name G for the graph is it? So, can you just check for the entire graph that we construct from phi. G. So, for every so once we fix an assignment to the variables let us see x bar be the set of corresponding cycle covers and let w x bar be the sum of the weights of cycle covers in C x.

So, the final claim is this that if phi of x bar evaluates to true then w x bar is 4 to the power 3 m and if this evaluates to false then it is 0. So, this is where we left yesterday. Each XOR basically multiplies the weight into 4. Whatever is the weight in the original graph if you put an XOR gadget and replace its 2 end edges it basically multiplies the total weight into 4. So, that is what is the; property of this. So, that is not quite true.

So, the thing is that if exactly one of these two edges are included in the cycle cover then it gets multiplied into 4, otherwise it is 0. So, that is what we said yesterday. So, why is this claim true?

So, let us look at a satisfying assignment. So, let us look at an x bar which evaluates to true. So, then what property does each clause gadget have? So, suppose I consider an x bar and in the cycle covers I consider the cycle cover for each variable as being the cycle cover taking the true external edges if that variable is set to 1.

And I consider the cycle cover in the variable gadget as taking the false external edges if that variable is set to 0. So, for example, in this x bar suppose x 1 is set to 1. So, then I will consider this particular cycle cover. So, we saw that there are exactly two of them. We will consider the top one and if x 1 is set to 0 we will consider the bottom one and we will do this for all variables. So, what can we say about the clause gadgets?

So, what we want finally is we want to have the total weight as 4 to the power 3 m. So, basically what we want is that all the XOR gadgets should have the property that one of its end edges is true and the other end edge is false. So, that is what we want. So, suppose in this clause what are the three literals? The three literals are x 1, x 2 bar and x 3. So, suppose x 1 and x 3 are true then we will consider the cycle cover and c 1 which does not include x 1 and x 3.

I mean the external edges corresponding to x 1 and x 3. So, we will only consider the cycle cover in c 1 which has this external edge. So, this is where we crucially use the fact that there is exactly one cycle cover corresponding to every subset. If all three literals are true we will consider the cycle cover when none of the external edges are present. So, there is again one cycle cover when none of the external edges are present.

So, basically it gets negated. So, if you have a literal as being true, the external edge corresponding to that literal gets omitted from the cycle cover. Now I am not saying that. What I am saying is there will be a unique cycle cover which will have the property that for every true literal the external edge corresponding to that literal is not present in that cycle cover. It is not present.

So, in other words if there is for example a clause which is completely false that is all its three literals are true all the three external edges must be present. But we know that there is no cycle cover which has all the three external edges being present. So, now what do we have? So, that is how we set the cycle cover for all the clauses. So, we look at the variables we do this setting and now since it passes through 3m XOR gadgets.

The total weight of these cycle covers will be four times, four times, four times all the way how many XOR gadgets you have. So, the total weight will be 4 to the power 3m. And suppose if you take an assignment where it evaluates to 0 then you will run into problems. Because then there will be some clause in which all its literals are false. So, now if you try to set that clause, I mean, if you try to consider a cycle cover for that clause, you will have to omit at least one external edge.

But if you omit one external edge in the variable corresponding, I mean in the corresponding variable that edge is also omitted. Because that clause I mean that literal is set to false here. So, both these things are omitted and by the property of the clause gadget if both its end edges are omitted then its total weight contribution is 0. So, that weight assignment will contribute to 0. So, that is basically then.

So, the basic idea is that I mean forget about the XOR gadgets for the time being. You have these individual components. I mean your entire graph consists of the m components corresponding to the clauses and n components corresponding to the variables. And you want to have the property that if a literal is set to a certain value you take a certain cycle cover in the variable gadgets.

And if a clause gets set to true I mean if whatever are the values that get set to true in a clause gadget you take the corresponding cycle cover in the clause gadget. And now you want to multiply all these cycle covers and that happens using the XOR gadgets. So, the basic thing is that it is not something very intricate happening here, but the clever part of this whole thing is this construction. So, once you have the construction then the rest of the analysis is not difficult.

So, any questions? What? When I am omitting an external edge see when do I omit an external edge? So, I omit an external edge if that literal is getting set to true in my assignment. So, if that literal is getting set to true in the assignment which cycle cover do I take corresponding to that literal? So, if that is a positive form literal I take the cycle cover corresponding to the true edge which means that that external edge gets included.

And if that is a literal in the negative form, for example, if that is something like x 2 bar then I take the cycle cover corresponding to the false edges. So, in other words, the external edge that is being taken each XOR gadget does not contribute four. Whatever is the weight of the original cycle cover replacing these two edges with whatever that thing was multiplied the weight four times. So, if the original weight was w.

The weight of the new cycle cover I mean the sum of the weights of the new cycle covers is 4w. It is not that you get only one cycle cover by replacing these two edges. You get a collection of cycle covers and the total sum is 4w. So, that is also something that you should verify. I mean I left that as an exercise yesterday but if you have not verified that please go back and look at that construction.

If you look at the total sum of all those things that will be 4w. So, what property do we have now? So, what we have here is that suppose this x 1 literal gets set to true in x bar then this edge is included in the cycle cover. But in the cycle cover for c 1 this edge is not included which means that now the clause gadget has the property that one of its end gadgets is included. So, whatever weight it was there in the original graph that gets multiplied time into four.

So, let us say if you start without any clause gadgets you have a unique cycle cover for the entire graph. So, for the first clause gadget that gets multiplied into four. So, you have the total weight is four for this XOR gadget. Now for the second XOR gadget again that gets multiplied four times. So, that is four square and then finally it will be four to the power six. So, these XOR gadgets are only ensuring that both these things should not appear simultaneously.

That is all. Yes, c x bar will not be empty. c x bar will contain lots of cycle covers but their total sum will be 0. So, c x bar can contain cycle covers but it is only that their sum will be 0. So, I will give you a very simple cycle cover. I cannot just construct one but maybe you can look at an even more simpler formula and just try it out. I mean try looking at a formula and try looking at a non-satisfying assignment and you will see that you can construct cycle covers basically based on these XOR gadgets. But there will be two of them that will cancel each other off.

Suppose clause one is not satisfying. Yes, so suppose this has a value 0, 1 and 0, then what happens? So, here I will be taking this cycle cover, here I will be taking this cycle cover and here I will be taking again this cycle cover. So, now for this clause to contribute something, what should happen is that all these three external edges must be present because here this external edge has got omitted. I mean, I am sorry.

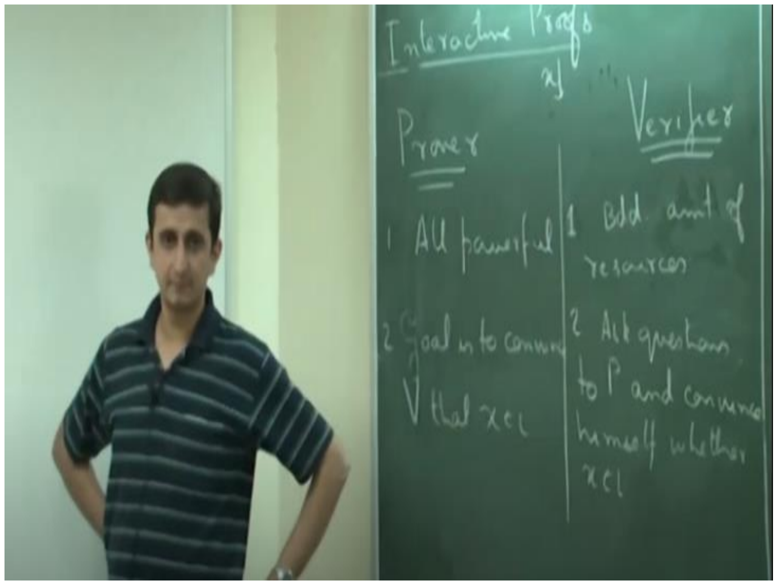
So, this external edge is omitted, here this is omitted and here this is omitted. So, here all three of them must be present for this to make any sense but that we know is not. Not only the middle one all three of them. The middle vertex in that clause, yes that will not get covered. There is no cycle corresponding to c 1 but then c 1 is part of your entire graph. So, if we have a cycle cover for the entire graph that better cover that middle word vertex for c 1 as well.

No, but here I am not getting a cycle cover the weights will go to 0. For example, if let us say I include this edge or I am sorry let us say if I exclude this edge, if I exclude one of the external edges then I know that in c 1 there is a cycle cover. Basically the cycle cover which includes this and this but if this edge is excluded this XOR gadget has the property that if both its end edges are excluded.

Then the total sum of the cycle covers that is present inside that XOR gadget will sum up to 0. So, that is why I said that you actually need to explicitly construct a formula together with the XOR gadget to find out which two will cancel each other off. It basically comes from this XOR gadget because that cannot be done. That is the property which c 1 has because ultimately that is what I want that if a clause is unsatisfiable by a certain weight assignment. Somehow I should not consider that assignment in my final count.

That is what I want. So, I will leave it at this point. So, again what I advise is you go back and check this yourself. Verify some of the gadgets and convince yourself why this is correct.

(Refer Slide Time: 33:24)


Interactive Proofs
Prover
Verifier
1 All powerful
1 Bdd. amt of resources
2 Goal is to convince V that x∈L
2 Ask questions to P and convince himself whether x∈L

So, what I will do next is we will start upon what is known as interactive proofs. So, I think this is chapter seven in your textbook, eight. So, we just keep the permanent function for the time being but later on again we will come back to this permanent function. So, what is an interactive proof and what significance does it have with respect to complexity? So, this notion of interactive proof again is something very natural.

So, I will explain it in a minute. So, the model that we consider is the one consisting of let us say two persons. So, there is a prover and there is a verifier. So, basically what I want to look at is I want to look at a computational model. Basically a model which accepts a certain language where given an instance is, suppose you are given some x there are two persons sitting over here. So, there is a prover and there is a verifier that has the following properties.

So, the prover is all powerful. So, this guy has all the computational resources available to him. I mean, you can also assume that this guy can decide undecidable languages. So, there is no restriction on the power of the prover but the verifier has certain restrictions. So, later fix this. So, a bounded amount of resources are available and what are their goals? I mean the goal of the prover is to somehow convince the verifier that x belongs to the language.

So, his goal is to convince V that x belongs to the language. But this guy, this verifier guy is skeptical in nature. So, he would not accept an x until and unless he was satisfied that whatever the prover is telling him is a correct proof of the fact that x belongs to the language. So, his goal is to, so how do I state this? And convince himself so in other words what you can think is that I mean his goal is to somehow reject the fact that x belongs to the language.

And again when we assume that such a model exists we will always assume that these are fair people. I mean they cannot cheat one another. So, this is basically the general model and what we will look at is we will look at various types of verifiers with various kinds of restrictions upon them. For example, what if the verifier is someone who can only perform deterministic polynomial time computations? What if he is allowed access to random bits, how does the model change?

Verifier can be NP also. That will lead to a more powerful class of languages. Yes, so bounded means that there is some bound on the computational power of the verifier. Yes, now do not think of the certificate model for the time being. So, do not think about that. The way to think of this is that there is this guy verifier who asks questions to the prover given an input x and then based on the responses that the prover gives either the verifier will ask more number of questions or finally he will decide whether or not to accept x.

So, there is basically a dialogue that is going on between these two persons. If the verifier is NP then what do you mean by does it matter? Of course, the resultant language I mean in the complexity of this entire model cannot be less than the complexity of the verifier. Because let us say the verifier is in some complexity class c given an input x I can always just take that input and run that machine for the verifier and accept or reject based on that without even asking the prover what to do.

You mean to say there is a language in NP? What do you mean certificate for a language? So, basically you have an NP machine for L. In the definition of NP, I do not think so. Because what would that mean? So, that would mean that there exists some certificate which makes the verifier accept that string. If the verifier is also an entity means that there exists another certificate which will put that in the language by means of another polynomial time verifier.

But now you can just concatenate those two certificates. So, that will be still empty. So, you mean in this model or in that? It can go right. That is not clear why it can decide sigma 2p. Yes, so not exactly. So, there you have the thing that for all certificates I mean, there exists some certificate let say c 1 such that for all possible certificates c 2 there is a polynomial time machine which accepts given this c 1 and c 2 along with x. But here that is not the case.

Here you do not have access to I mean, you cannot claim something like for all possible certificates. Here basically let me define the model precisely. So, this was just an informal notion. But here I mean as we shall see this model does have a lower computational power than this there exists, for all alternation. Because here; what is happening is basically a set of question answers. So, this guy asks a question and he gets just one answer for that question.

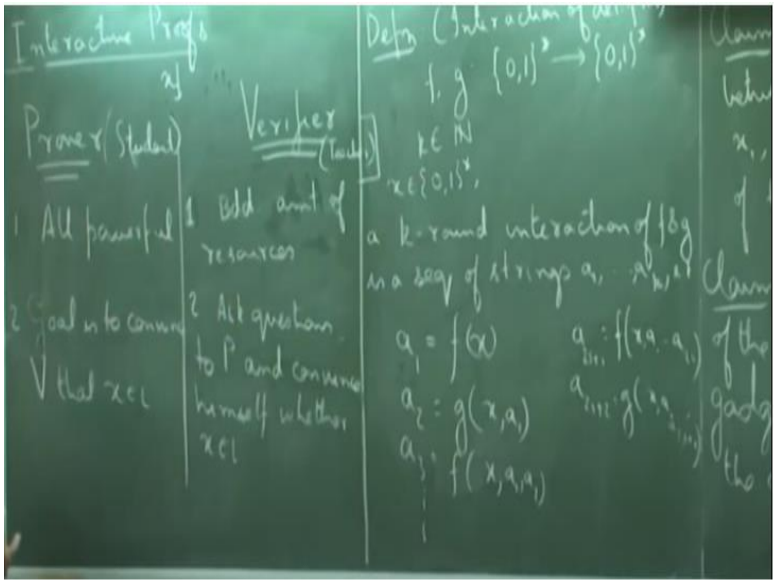
So, the question and answer is basically just one string. So, he provides some string to the prover and the prover takes that string and based on what that string is provides another string back to the verifier and this dialogue continues for a certain number of steps and then the verifier accepts. So, the number of strings that can be interchanged is bounded by polynomials by the definition of this model. So, now it is not counter intuitive.

So, let me give you an example. Suppose you are the prover, suppose the prover in this case is the student and the verifier is myself. Whatever. So, now the goal of the teacher is basically I give you a question and your job is to convince me that whatever solution you are providing me is correct. So, you can use whatever knowledge you have. You can look up any book, anywhere, anything.

But then I will be only convinced if there is a step by step deduction in whatever proof you give me. So, I will basically go through each and line of your proof and if the current line follows from the previous line only then I will be convinced that the final answer that you have is correct. So, the goal of the prover is to convince that whatever is the input that belongs to the language but the verifier can only have a certain amount of resource.

And within that he has to check whatever the proof the prover is giving is correct or not. So, yes, so this is one example. The prover can give a statement. I mean if you give a statement which does not make sense the verifier will discard it and if it makes sense then well and good. Number of questions is polynomial.

(Refer Slide Time: 46:51)

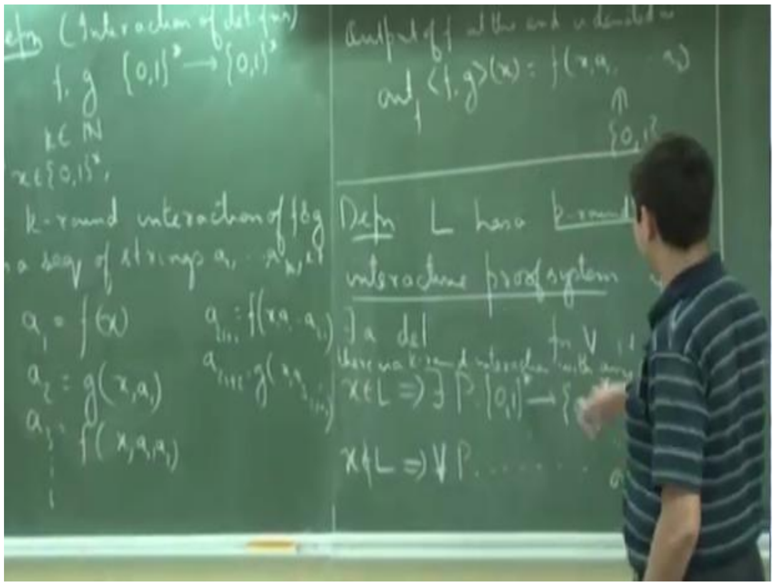

Interactive Proofs
Prover (Student)
1 All powerful
2 Goal is to convince V that $x \in L$
Verifier (Teacher)
1 Bdd amt of resources
2 Ask questions to P and convince himself whether $x \in L$
Defn (Interaction of
$f, g \{0,1\}^* \to \{0,1\}^*$
$k \in \mathbb{N}$
$x \in \{0,1\}^*$,
a k-round interaction of f & g is a seq of strings $a_1 \dots a_k$
$a_1 = f(x)$
$a_2 = g(x, a_1)$
$a_3 = f(x, a_1, a_2)$
$a_{2i+1} = f(x, a_1 \dots a_{2i})$
$a_{2i+2} = g(x, a_1 \dots a_{2i+1})$

So, let me define the formal model here. This is an interaction of we will define a general model first and then we will look upon restriction on the verifier Interaction of deterministic functions. So, this stands for deterministic. So, suppose you have two functions f and g from 0, 1 star to 0, 1 star. And you have some number k in which is a positive integer. So, given an x given a string x k round interaction of f and g is a sequence of strings a 1 through a k such that a 1 is the answer of f of x.

So, then basically a 1 is something which the verifier initially computes given x and then he sends a 1 to the prover. So, a 2 is what the prover computes given x and a 1. And then he sends an answer back to the verifier. So, a 3 is f of x, a 1, a 2 and so on. So, finally a 2 i + 1 is f of x, a 1 up to 2 i. So, of course, whatever answers are being provided in certain round of interaction are known to all future rounds.

So, both the prover and the verifier have access to all the answers in the previous rounds. So, k round interaction is a sequence of strings a 1 through a k such that these strings have this property and the output of so what is this called?

(Refer Slide Time: 50:21)



So, the output of n f at the end is denoted as out f f comma g on x and this is basically whatever is f of x a 1 up to a k. So, finally when the k rounds terminate, whatever is the value of f on all the answers that he gets is basically the output. And this will assume some binary number. This is basically whether x gets accepted or rejected. No, these are strings because these are the answers but whatever is the final output because ultimately I am given an x.

So, at the end of my question answer session I have to finally output whether x belongs to the language or it does not belong. So, finally I have I will just output 0, 1. So, it does not matter actually. I mean if k is odd you can just add one more dummy step. So, basically if k is odd it means that the last question is being asked by the verifier and its answer is not being used. So, it is basically the same as not asking that question at all.

So, we can assume that k is even. So, in this model what we were assuming is that f is our verifier and g is the prover. So, this is the general model. So, now let us look at complexity bounds on these functions. So, we say that a language L has a, k round deterministic interactive proof system if there exists a deterministic poly time function V such that for all strings x if x belongs to L then there exists a function P.

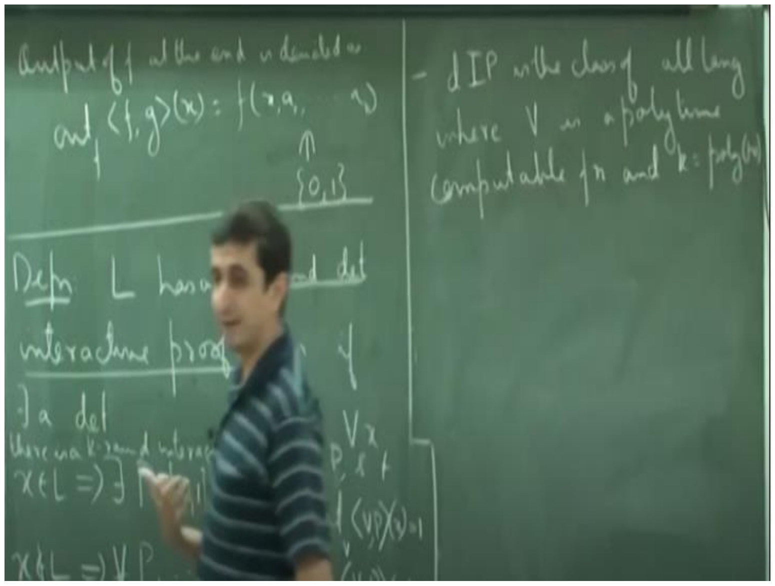
So, basically it means that there exists some prover such that output of the verifier on x is 1 and if x does not belong to L then for all provers and for all functions from strings to strings the output is 0. So, basically no prover can ever convince the verifier. Yes, maybe we can make this more general but then k should be polynomial here.

One second. So, for all x there is a, k round interaction with any function P such that this happens. So, what were you asking? All the provers are of the same power. I mean, you can assume that. Now this means that no matter what function you consider so prover is basically a function. Given a string it outputs some other string. So, no matter what function you consider when we say that the prover has unlimited power.

What we just mean is that to compute that function we do not assume any restriction on our computational model. It can be an all powerful model but what we mean here is that no matter what function we consider that can never convince us that x belongs to the language. I mean, that is what we want. So, for example if you look at an NP language, let us say SAT to see if I want to convince ourselves that a boolean formula is satisfiable, what we want is basically one satisfying assignment.

There should be somebody who provides us a satisfying assignment. But if I want to convince myself that a formula is not satisfiable, no matter what formula I mean, no matter what function I consider here I will never be able to get a I mean, I should never be able to get a satisfying assignment. So, it is very important that this should work for all possible functions. So, here we are since I did not have any polynomial time constraint here we can look at just a general definition.

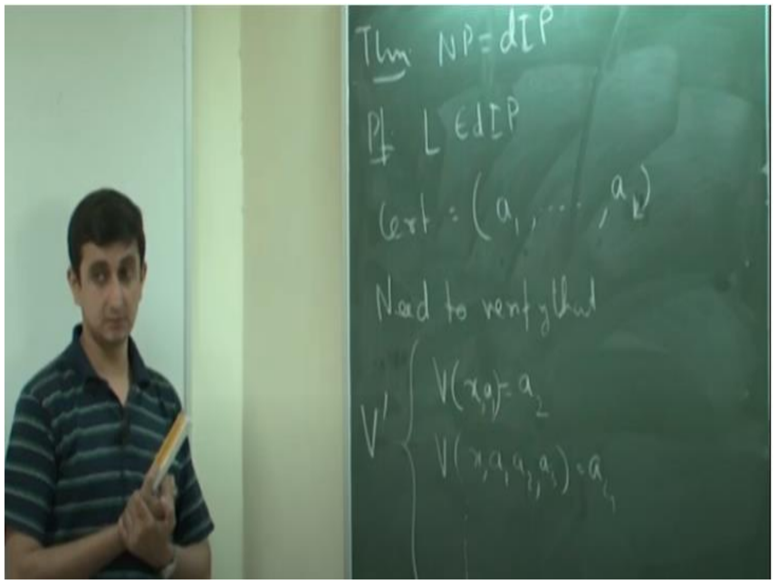
(Refer Slide Time: 58:31)


- d IP is the class of all lang
where V is a poly time
computable fn and k = poly(n)

But then we can define the class d I P. So, d I P is the class of all languages where V is a poly time computable function. So, what is this class d I P? What is the complexity? Certainly NP is contained in this class because if we have I mean, if you consider a language in NP I mean I can always choose our prover to be the one who provides this certificate. And if you look at the certificate based definition given an instance the prover is one who just provides a certificate.

And then the verifier just plugs it into its polynomial function and does it but what about the other side? I mean how is it any more powerful than NP? Yes so I am running a little bit over time. So, can I take 15 more minutes? Is it okay with everybody? So, I just want to complete this part.

(Refer Slide Time: 01:00:18)



So, basically what we said is that this is trivial but what about? So, k is polynomial in the length of the input. So, I think you can also say that k is bounded by polynomials. So, we can say here I guess. So, there is a class of all languages V where V is a poly time computable function and let us say k is some polynomial in the size of x. So, the number of interactions is not more than polynomial. So, as it turns out that these two classes are actually the same and the proof is not very difficult.

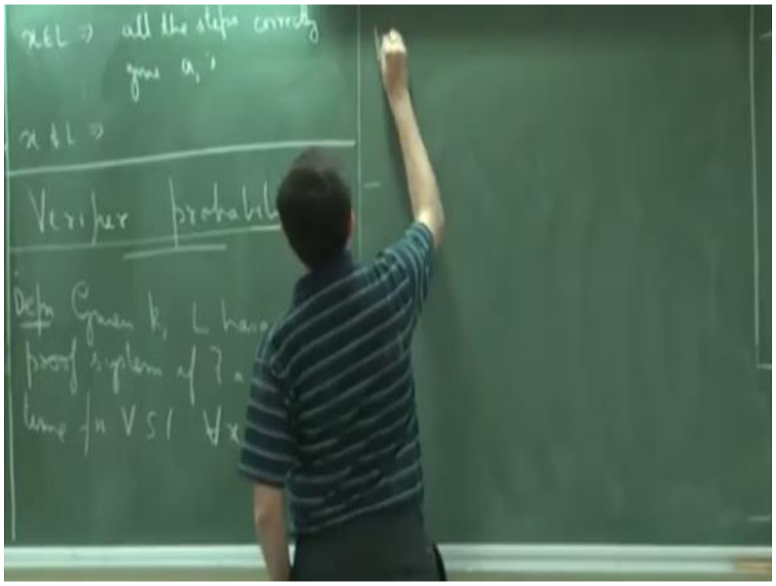
So, how do we prove it? So, let us take a language in d I P. We need to give it an NP definition, so that is we need to give a construction of a certificate. So, what should the certificate be? So, the certificate will be just a concatenation of all the answers that the verifier gets together with questions as well. But the questions do not really matter. So, what is the polynomial time? So, we have the verifier but so what does the verifier do?

I mean the verifier just computes these, a 1, a 3 and all these things which basically just computes the questions. That is the function V here. But that is not sufficient. So, it is not just sufficient to look at the last step. So, what the verifier needs to check here is that we need to verify that V of x is a 1, V of x, a 1, a 2 is a 3 and so on. So, he needs I am sorry this is not correct. So, x, a 1 is a 2.

So, he needs to verify the answers of the prover and so on. So, given a certificate the verifier basically checks that all the answers are the correct answers or not depending on the questions that he had asked. So, this is yes, so this is not exactly the same verifier. So, what we are constructing is a polynomial time machine let us call it V prime which does all these checks. Yes, no, the computation of a 2 is very powerful, but then he has the answers.

So, this machine V prime has already been provided with this certificate. So, again, the proof I mean, the prover verifier model I think of the thing that I already have your solution. So, now I am just checking that corresponding to the questions that I had asked whether the answers are correct or not. So, he already has access to this certificate and he does this step by step verification. So, now what if string x is in the language because V is a polynomial time machine.

(Refer Slide Time: 01:06:22)


$x \in L \Rightarrow$ all the steps correctly give a_i's
$x \notin L \Rightarrow$
Verifier probabil
Defn Given k, L has
proof system of ? a
time fn V s.t. $\forall x$

So, if x belongs to L then there exists such a certificate which implies that finally whatever is the answer so all the steps correctly give whatever are the a i's and the final answer is yes. And what if x does not belong to L? So, what do we mean by that? So, x does not belong to L means here that for all prover functions this is 0. So, basically that for every certificate there is at least one step where it would not match with whatever V is computing.

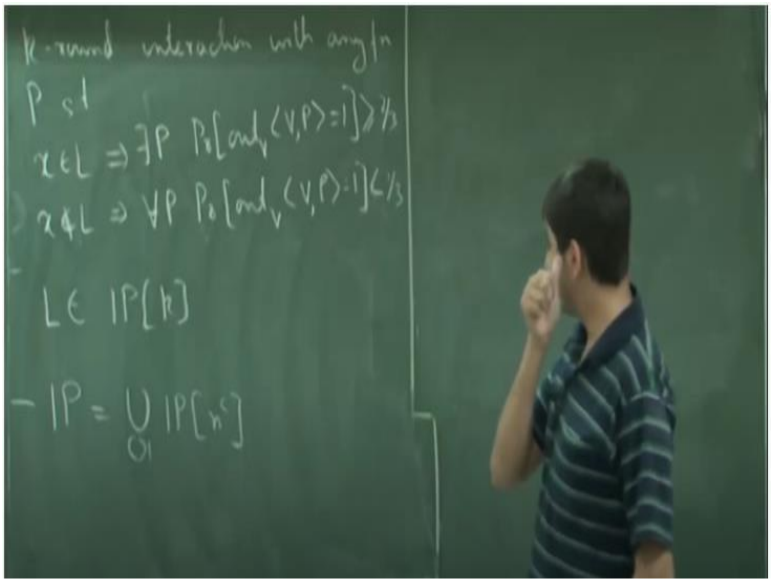
Final answer will be no. So, the final answer is also a computation of one of these steps. But I think the way we defined it we do consider it as a string. So, we consider it as a string that is being outputted in the last step. So, if you recall, it was f of x a 1 through a k, and we assumed that it outputted a binary number. So, in the last step of this thing it will output a 0. So, however you want to look at it. So, what is the point behind this?

I mean the point behind this is that since we have restricted our number of rounds to polynomials it does not matter how many rounds we perform, how many questions we ask, we can always concatenate all these questions. It is basically that I am concatenating all these questions and then asking the prover something and then depending on that I am either accepting or rejecting. There is always a way to go about doing that.

So, this is the crucial difference between let us say something like sigma 2 where you have alternating quantifiers and its interactive model. So, the point is that this is not very interesting because it does not give any power in addition to NP. So, what makes it interesting is if we make the verifier probabilistic. So, suppose the verifier is not a deterministic machine but a probabilistic machine.

So, in other words let me just define it this way that given k L has an interactive proof system if there exists a probabilistic polynomial time function V such that for all x there is a k round intersection.

(Refer Slide Time: 01:11:30)



k-round interaction with any fn P st

$x \in L \Rightarrow \exists P \; \Pr[\mathrm{out}_V \langle V, P \rangle = 1] \geq 2/3$

$x \notin L \Rightarrow \forall P \; \Pr[\mathrm{out}_V \langle V, P \rangle = 1] < 1/3$

$L \in IP[k]$

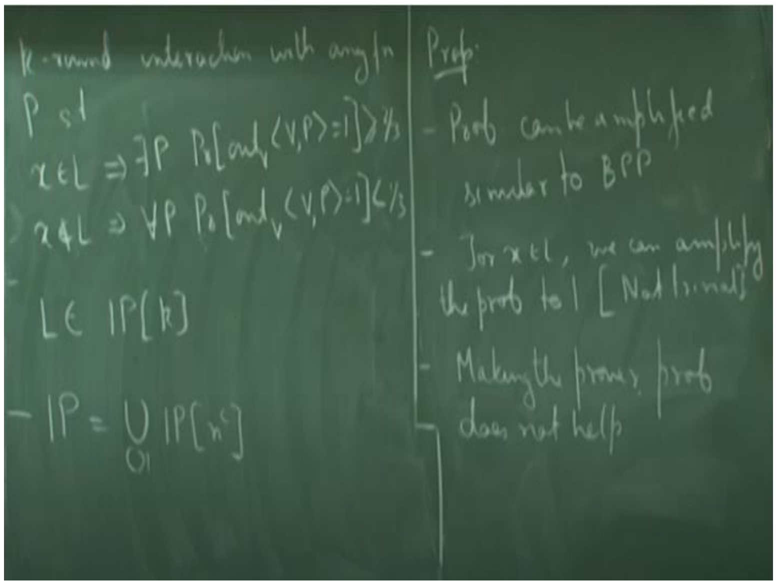
$IP = \bigcup_{c \geq 1} IP[n^c]$

I am sorry k round interaction with any function P such that if x belongs to the language then there exists a prover such that the probability that V accepts is greater than two third and if x is not enough then for all functions P probability that again this guy accepts is less than one third. And the notation that we use to denote this is that we say that L belongs to I P of k. So, this k basically denotes how many rounds of interaction is required to put L in this class.

So, what is the interactive proof model that I am using? How many interactions does it have? And we say that the class I P is just union over all c of let us say into the part c. So, this as we; shall see that this model is the more interesting. So, the moment we make the verifier probabilistic so now basically what the verifier can do is that he can toss coins. So, I want to check whether some of your answers are correct or not, so I just toss some coin and depending on that I will grade you.

Let us just look at some properties of this class I mean some things that we can assume without loss of generality.

(Refer Slide Time: 01:14:22)



k-round interaction with any fn P s.t.

$x \in L \Rightarrow \exists P \; \Pr[\text{out}_V \langle V,P \rangle = 1] \geq 2/3$

$x \notin L \Rightarrow \forall P \; \Pr[\text{out}_V \langle V,P \rangle = 1] < 1/3$

- $L \in IP[k]$

- $IP = \bigcup_{c} IP[n^c]$

Prop:

- Prob can be amplified similar to BPP
- For $x \in L$, we can amplify the prob to 1 [Not trivial]
- Making the prover's prob does not help

So, firstly this probability can be amplified. So, same as what we do in BPP we take multiple trials and then we take the majority of the answers. If you do the same thing here and the same proof technique basically just apply churn of bound to and you will get that it can be amplified. So, that is one thing. There is also a very interesting property that this class exhibits that is what we can show is that for x in L we can, how do I say this.

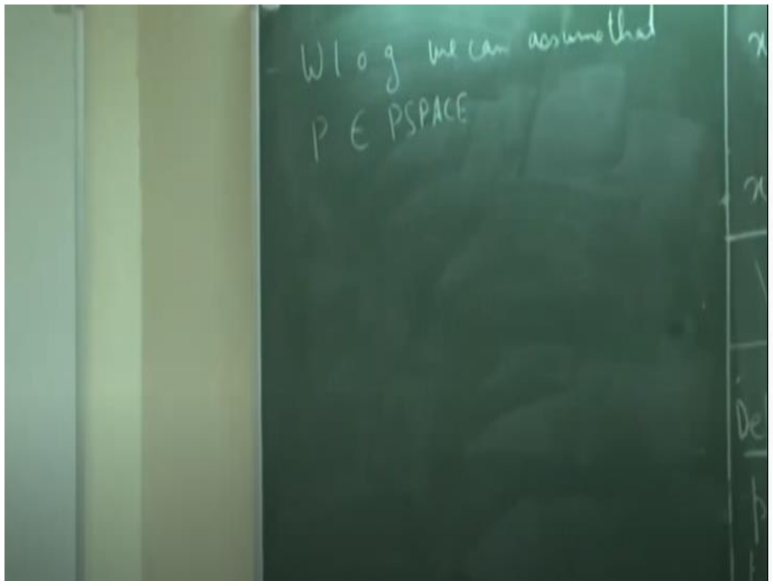
For x in L we can amplify the probability all the way up to one actually. So, in other words, we can make this a one-sided error computation. So, this is not something that is very intuitive. Because as we saw for the case of RL and sorry RP and BPP you can only amplify your success probability, but you cannot make it one sided from two sided. But as it turns out in the case of I P this can be done. And this is actually a non-trivial construction.

So, we will not prove it right now. So, maybe a few classes later, we will look at a proof of this. So, yes, you are looking at answers of your set. No, so the proof that we will look at is definitely not similar to that. Now I do not think this will actually give you the proof. So, that is basically using the fact that serves SAT is self reducible. So, you can compute a satisfying assignment if you have access to SAT. So, we will see later.

So, when we come to this we will see how the proof goes. So, what other things that I had, you mean to say that the verifier is, so the verifier answer expected polynomial time with no error at all. I do not know. Yes for BPP and RP whatever are the corresponding notions here, they are equal. But for ZPP I do not know. For example, if you have zero sided error here what it gives I am not clear. I will try to see that.

So, the third thing is that making the prover probabilistic does not help. So, any way I mean your prover is an all powerful machine. I mean, if you make it probabilistic what you can do is that you can have another prover who runs through all the coin tosses of this original prover and then just take the majority. Because if it is a probabilistic machine it should accept more than half the sequence of coin tosses. So, I can just simulate all the sequences and take the majority again. So, that does not tell.

(Refer Slide Time: 01:19:42)



And the final property that I will state again we will look at its proof at a later time is. So, without loss of generality we can assume that this prover here P means the prover. So, do not confuse it with the class P. So, this lies in PSPACE. So, in other words, even if you consider a function which has more computational power than PSPACE, it can actually be simulated by some other function which lies in PSPACE.

So, PSPACE and the power of PSPACE, is sufficient to capture the complexity of this prover and also preserves this definition. So, even if you restrict the power to PSPACE we will get the same set of classes sorry the same set of languages. So, this again is not very difficult to show but we will look at an argument later on. So, I wanted to do an example also. But I think I will stop here. We are already right over time.