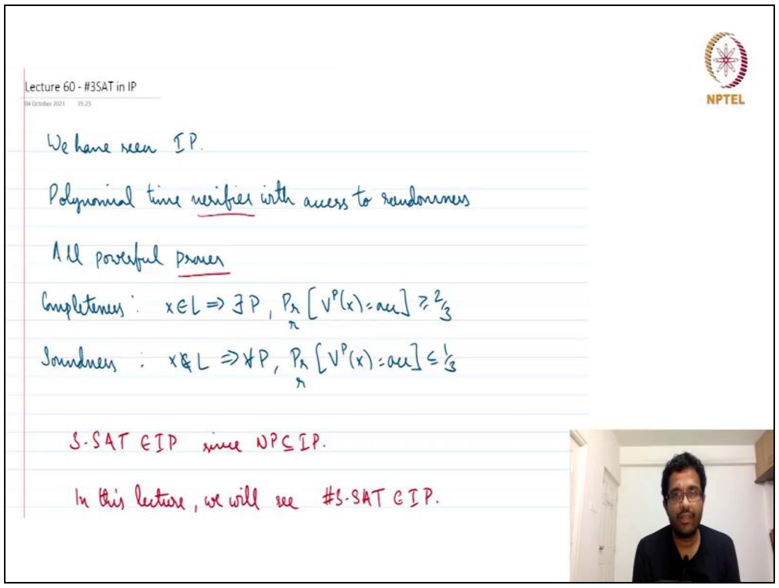
Computational Complexity

Prof. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture -60
#3 SAT is in IP

(Refer Slide Time: 00:15)



Lecture 60 - #3SAT in IP

We have seen IP.

Polynomial time verifiers with access to randomness

All powerful prover

Completeness : $x \in L \Rightarrow \exists P, \Pr_r [V^P(x) = acc] \geq 2/3$

Soundness : $x \notin L \Rightarrow \forall P, \Pr_r [V^P(x) = acc] \leq 1/3$

3-SAT ∈ IP since NP ⊆ IP.

In this lecture, we will see #3-SAT ∈ IP.

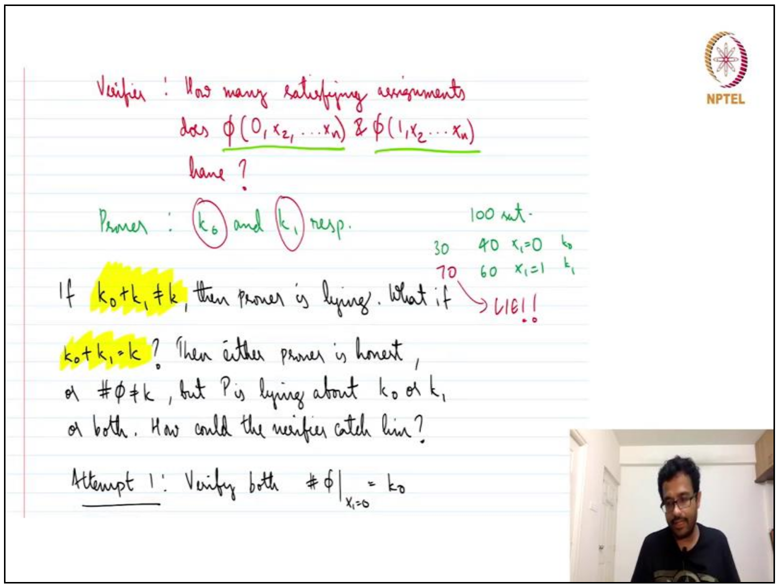
Hello and welcome to lecture 60 of the course computational complexity. In the previous lecture we saw interactive proof systems where the verifier has is computationally bounded his polynomial time verifier but has access to randomness and the prover is all powerful we saw that graph non-isomorphism is an IP which was a language that we did not know to be in NP. So, we want to know how big IP is. We already saw that IP is contained in P space even not even though it is not a formal proof I just mentioned as a brief remark.

So, in this lecture we will we will see that sharp 3 SAT is contained in IP. So, as we already know sharp 3 SAT is a sharply complete language and it is by TODA's theorem it contains all of all of polynomial hierarchy. So, interestingly we see that sharp 3 SAT is in IP sharp3 SAT as an interactive flow system. So, just to recap completeness and soundness, completeness corresponds to the situation where x is in the language then there should access prover that can convince the verifier to accept with probability at least 2 thirds.

Ad soundness is when x is not in the language no matter what the prover tries to do no matter which grouper shows up they should not be able to convince the verifier to accept their accept the string beyond the probability 1 thirds. And as we said as we saw this 2 thirds can go up to 1 - 1 by 2 power some constant and this can go up to 1 by 2 power some constant. So, basically and in fact; so, this is just by boosting repeating and doing character taking the majority.

And we also said that the perfect completeness we can push the probability of 2 by 3 to close to 1 or exactly 1.

(Refer Slide Time: 02:32)



Verifier : How many satisfying assignments does $\underline{\phi(0, x_2, \ldots x_n)}$ & $\underline{\phi(1, x_2 \ldots x_n)}$ have ?

Prover : $\boxed{k_0}$ and $\boxed{k_1}$ resp.

100 sat.

30	40	$x_1=0$	k_0
70	60	$x_1=1$	k_1

If $k_0+k_1 \neq k$, then prover is lying. What if $k_0+k_1 = k$? Then either prover is honest, or $\#\phi \neq k$, but P is lying about k_0 or k_1 or both. How could the verifier catch him?

$\rightarrow$ LIE!!

Attempt 1: Verify both $\#\phi|_{x_1=0} = k_0$

Anyway in this lecture we will see that sharp tree set is contained in IP. So, basically we will provide a interactive proof protocol where the prover will be able to convince the verifier a polynomial time verifier to the number of satisfying assignments of a 3 SAT formula boolean research formula and interestingly this will also be a public coin protocol. So, the protocol that we saw for graph non-isomorphism was a private coin protocol where the verifier did not reveal the randomness to the prover.

But this will be a public coin protocol. So, what is this; what is the problem 3 SAT is given a formula 3c in the formula phi how many satisfying assignments are there for phi. So, instead we will slightly tweak the problem we look at the decision version even though I just formally stated now but later will not be may not be. So, careful so given phi and a number k we are asking does phi have exactly k satisfying assignments this is the question.

So, this is a decision version. So, the provers task is to convince the verifier that this formula has exactly k satisfying assignments. And we have seen many of these some of the ingredients of this proof in the previous lectures. So, I may; when they show up I will probably remind you. So what is a 3C in a formula. So, phi is a conjunction of clauses or an AND of clauses phi is C 1 and C 2 and so on up to C m.

There are m clauses and n variables m clauses and n variables the variables are x 1 up to x n now. So, let us try to because we do not know we can we do not know any otherwise let us try to see how this proof will go. So, verifier may ask tell me how many satisfying assignments are there prover may say it has k satisfying assignments. So, now verify okay now tell me convince me that there are k satisfying assignments.

So, how will he convince. So, there has to be a protocol. So, some partial information or some specific information could be provided that the approver could that the verifier could verify locally. So, for instance 1 possibility is this verifier could ask suppose I set x 1 to be 0 how many satisfying assignments are there in that boolean formula. So, which is this where phi that x 1 is set to 0 or false and others remain variables.

So, x 2 and x 2 x 3 up to x n could take 0 or 1 how many satisfying assignments does this have what if the prover can provide this number. Let us say prover says that this number is k 0. Similarly the verifier could ask how many satisfying assignments are there for phi of 1 x 2 up to x n and prover may say this has k 1 satisfying assignments. So, now we know when x 1 is set to 0 there are so, many k 0 when x 1 is x 2 set to 1 there are k 1 satisfying assignments.

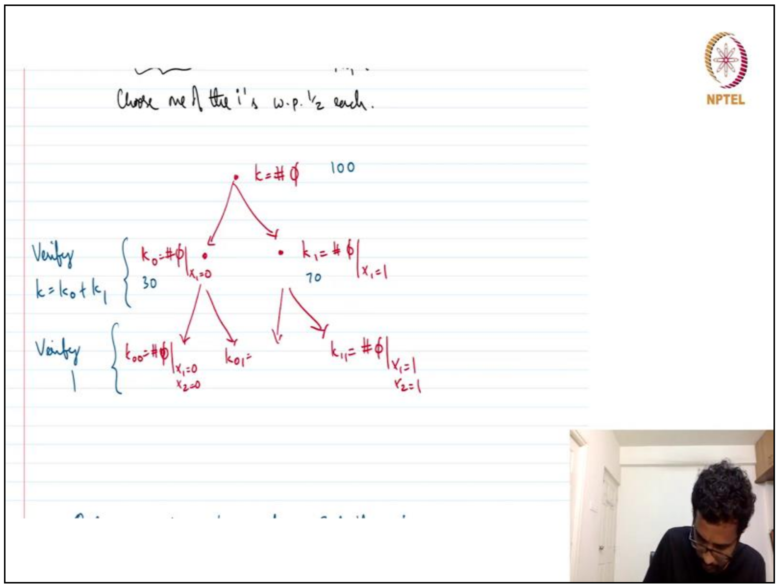
And now the total number of satisfying assignments should be the sum of these 2. So, verifier could possibly check whether k is equal to k 0 plus k 1. And if case is not equal to k 0 plus k 1 then verifier can immediately reject. But suppose k is equal to k 0 plus k 1. So, now what next? Well 1 thing that could be done is. So, it is possible that if k is indeed the actual number of satisfying assignments then it is and k 0 is actual number where x 1 is set to 0 and k 1 is actual number where x 1 is set to 1 then everything is being done truthfully.

So, there is it is all fine but what if the actual number is not k. Let us say the actual number of satisfying assignments is not k now what does this mean but k 0 and k 1 are such that k 0 and k 1 is equal to plus k 1 is equal to k. So, let us say for instance let us say there are 100 satisfying assignments 40 when x 0 x 1 is 0 and 60 when x 1 equal to 1. So, maybe just write them by the side. So, 100 satisfying assignments; 40 when x 1 equal to 0 and 60 x 1 equal to 1.

So, this is k this is k 0 and this is k 1. Now let us say the prover says 100 satisfying assignments now verifier asks what is x how many when x 1 equal to 0 and how many when x 1 equal to 1 now verifier checks maybe there are only thirty satisfying assignments when x 1 equal to 0. So, he outputs 30 k 0 but then the actual number of satisfying assignments when x 1 equal to 1 may be something else maybe that is also 30.

So, because then 30 + 30 equal to 60 that is why 100 is not the actual number but then the prover is trying to convince the verifier that there are 100 satisfying assignments. So, what does he do he could possibly lie and say there are 70 satisfying assignments when x 1 equal to 1. So, this is a lie because he just wants to say this 100. So, this is what this is something that can happen or maybe both of them are lies now how does the verifier catch this.

(Refer Slide Time: 08:50)



So, in other words there should be some way. So, these themselves these numbers itself may not be the actual numbers. So, how does a verifier catch this? So, 1 possibility, so, let us see now how this could how this could unravel. So, as i said like k 0 and k 1 there is a number of satisfying assignments when x 1 equal to 0 and number of satisfying assignments when x 1 equal to 1 respectively could be asked maybe we could recurse.

So, perhaps we could ask now to verify k 0 we could recurse. So, we could ask what is the number of satisfying assignments when x 1 equal to 0 and x 2 equal to 0 and what is the number of satisfying assignments when x 1 equal to 0 and x 2 equal to 1 and this may be k 01 sorry 00 and this may be k 01 and similarly I could have k 10 and k 11 and then the verifier will and this numbers are provided by the prover and then the verifier verifies that.

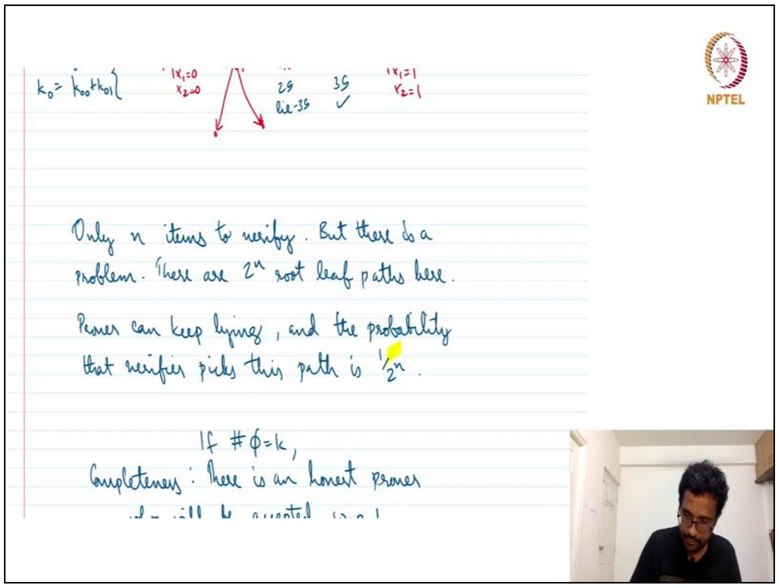
So, verifier verifies that verifier checks k 0 is actually k 00 plus k 01 and over here he checks that k 1 is equal to k 10 and k plus k 11 this is this is something that can be done and now we have the same problem with k 00 k 01 and. So, on k 10 k 11 but then you can see that the number of items to check is exponentially increasing at each step and the end this leads to 2 power and many options. So, basically this just boils down to checking all the assignments. So, this is not really this is not really that clever or anything we are just exponentially many communications all the numbers need to be sent.

So, at the end we are just it is perhaps even worse than trying out all the 2 power and possible assignments and checking whether each 1 of them is satisfiable. So, this is not efficient. So, this is not efficient. So, the verifier will need to check more than polynomial many, many things even though this may be correct perfect completeness and soundness. So, this is not efficient. So, let us. So, this is clearly not feasible let us try something else.

So, instead of checking k 0 and k 11 possibility is you just check 1 of them. So, as I said. So, maybe k is 100 and k 0 is 30 or case reported reported to be 100 let us say this is reported 100 and this is k 0 is reported to be 30 k 1 is reported to be 70. It is possible that k 0 is like. So, the it is possible that some of them are correct some of them are not correct. So, instead of verifying both, so, earlier we just said that we verify both. So, in this another attempt is to you just pick 1 path and verify that.

So, we do not check everything we just check 1 path and how do you verify k 0 is 30 you again ask what is k 00 and what is k 01. I am not writing them in detail this is when so I am not writing them here because it is there is not much space. So, you just check okay sorry there's some problem ok. So, sorry k k 01 you can check that. So, k 0 is you you can ask the prover to give k 00 and k 0 1 and verify that. So, here we verify that k is equal to k 0 plus k 1 here and here we verify that let us say k 00 is equal to k i am sorry a bit of trouble erasing stuff it's not clear why verify that k 0 equal to k 00 plus k 01 and every time we just choose 1 path to verify. So, maybe in maybe over here we will check again k 01 we will just check this path.

(Refer Slide Time: 14:00)


$k_0 = k_{00} + k_{01}$

Only n items to verify. But there is a problem. There are 2^n root leaf paths here.

Prover can keep lying, and the probability that verifier picks this path is $1/2^n$.

If $\#\phi = k$,
Completeness: there is an honest prover

So, this path which path to be verified may be chosen randomly. So, and and. So, ok initially we have to verify k then k 0 then k 01. So, at every level there is only 1 item that we are verifying. So, this seems ok because we do not have an explosion of paths because every level we are only verifying 1 item each. So, there are n items. So, there are maybe 2n rounds or something like that but there is a small issue with this. So, if k if the prover is an honest prover and he has an honest correct thing to verify k being the actual number of satisfying assignments then it is all fine.

However let us say this formula has actually 90 satisfying assignments, actually 90 and it has 30 when k is equal x 1 equals 0 and 60 when x 1 equal to 1 okay 90 and 30 when x 1 equal to 0 and 60 when x 1 equal to 1. So, which means k 0 is 30 and k 1 is 70. So, the verifier sorry the prover could be very smart here he will he could report third k 0 s 30 truthfully but then his goal is to convince that k is 100.

So, if he reports k 1 to be 60 truthfully then he will be immediately caught. The verifier can check 30 + 60 and see that it is not equal to 100. So, he will be immediately the prover will be immediately caught. So, instead he can potentially lie he can lie and say 60 instead of 70. So, can lie in 1 of this; now if in the; next round the verifier chooses to verify. So, in this in this attempt we are only verifying 1 path. So, suppose the verified decides to verify the correctness of case 0 then the prover has no problem because at this point the actual k 0 is actually 30 and the prover has reported 30.

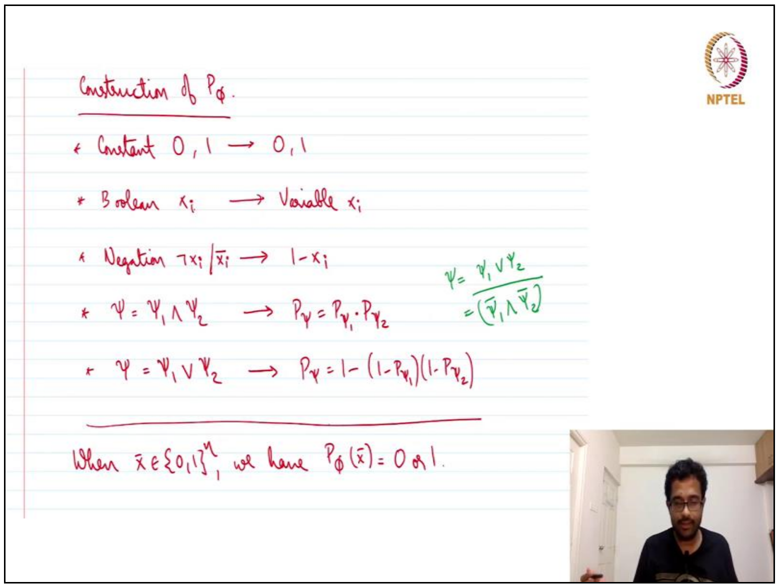
So, he can actually convince him honestly convince the verifier honestly that the k 0 is equal to 30.. So, there is no issue with that. But there is an issue if the approver if the verifier asks the prover to verify 70 or 70 but when the actual number is 60. Now again suppose k 11 or sorry k 01 is actually let us say 25 and this number is 45 and sorry this is this is a lie. So, we said it is 60. So, this is actually 35. Now let us say the brewer decides to say k 11 correctly and he decides to lie here he says this is also 35.

So, basically what I am saying is that at every level he can lie he can choose to lie in 1 path because if he says the report the correct number is 25 and 35 he will be caught because at this point prover will check whether 25 + 35 is equal to 70 at which point verify approver will be caught verifier will check this. So, instead he lies in this path and he will not be caught if the prover sorry if the verifier decides to go by k 11 instead of k 01 I am sorry it is not k 01 over here it is k 10. So, instead of k 10.

So, at every point there is a possibility that the prover may escape if the verifier decides to go the other way. In fact for the prover to be caught through his series of lies at every stage at every level the verifier has to guess the correct way where he lied and up to the root up sorry up to the leaf. So, he has to go all the way down and then he can possibly be caught. At the base case it will probably boil down to verifying whether certain assignment is a satisfying assignment or not a satisfying assignment.

At which point the approver will be caught but then so there is a; so, there is a probability of the prover getting caught but if the prover does this strategy like lying only in 1 of the 2 options at every step the probability of the verifier catching him is at most 1 divided by 2 power n which is what I have written here. So, the verifier the prover can try this very evasive strategy and almost never be caught.

(Refer Slide Time: 19:13)



Construction of P_ϕ.

* Constant $0,1 \longrightarrow 0,1$
* Boolean $x_i \longrightarrow$ Variable x_i
* Negation $\neg x_i / \bar{x}_i \longrightarrow 1-x_i$
* $\psi = \psi_1 \wedge \psi_2 \longrightarrow P_\psi = P_{\psi_1} \cdot P_{\psi_2}$
* $\psi = \psi_1 \vee \psi_2 \longrightarrow P_\psi = 1-(1-P_{\psi_1})(1-P_{\psi_2})$

$\psi = \psi_1 \vee \psi_2 = \overline{(\bar{\psi}_1 \wedge \bar{\psi}_2)}$

When $\bar{x} \in \{0,1\}^n$, we have $P_\phi(\bar{x}) = 0$ or 1.

So, the completeness is fine this all this protocol that we have been discussing. So, far whenever the prover wants to convince the correct thing the honest prover will always be able to convince with probability one. So, with perfect completeness but the soundness is a problem here the evasive prover that we just described gets accepted with the probability of 1 - 1 divided by 2 power n and this is this is very high it is very high it is too close to 1.

So, this is not really going to help us. So, we want to. So, this gap of 1 and 1 - 2 power n is too close we wanted to we want to do something else. So, we will do something else completely different or roughly in this template but then we will use a will bring in a new ingredient which is completely different. So, we will use polynomials. So, given phi which is a Boolean formula so, variables negations and or etcetera we will come we will first write down a polynomial called P phi.

With the property that the polynomial will have n variables x 1 to x n the same looks like the same as the boolean variables x 1 to x n with the property that when whenever x 1 to x n is from the Boolean hypercube meaning x whenever x 1 is 1 x is from the 0 1 power n whenever. So, this denotes x bar denotes sorry this x bar denotes x 1 up to x n whenever x 1 up to x 1 is from a 01 case. In that case if you evaluate this polynomial it will be the actual the truth value of the boolean formula.

So, P phi of x bar will be equal to phi of x bar meaning if x bar is a satisfying assignment P phi of x bar will be 1 and if x bar is not a satisfying assignment this will be 0. So, it will be 1 or 0 if x bar is a satisfying or not satisfying assignment and degree of P 5 will be at most 3n and further this polynomial will be very structured polynomial that can easily be computed given the inputs x 1 x 2 up to x n or even given alpha 1 alpha 2 up to any integer input also this can be efficiently computed.

So, in fact we have seen almost the same technique when we saw the parity in AC 0 proof not in AC 0 proof by Raspberry when Smallinski in lecture 31 or sorry 41. So, you can go look up those lectures but there I think it was for a specific field but the idea is exactly the same. So, how to construct P phi? Given phi, so, what would phi have as its components. So, it could either have constant 0 or 1 we could use the same constant 0 or 1 in the polynomial we could have variables x i we will use boolean variables x i will be replaced by polynomial variables in the polynomial x i.

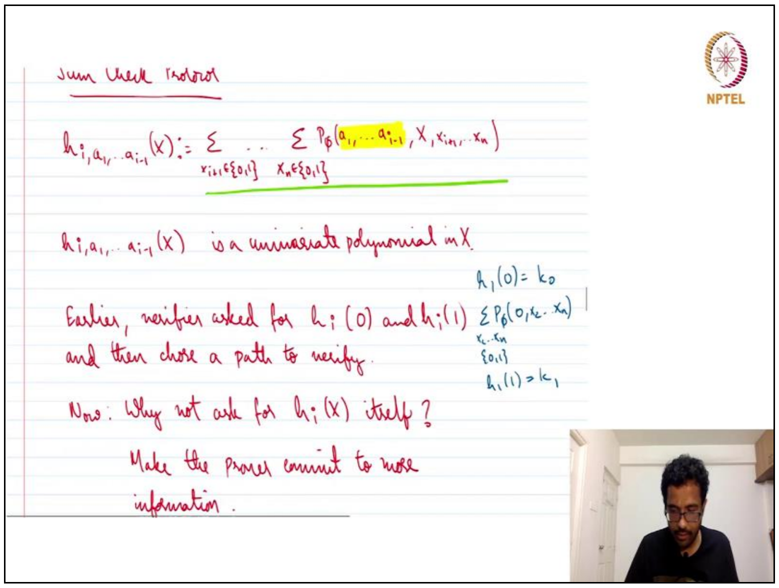
When there is a negation of a variable like it is denoted neg x i or x i bar we can use 1 - x i mean when x is 0 1 - x is 1 when x is 1 1 - x i is 0. Then we have an AND of 2 formulas phi psi 1 and psi 2 then the resulting polynomial will be the product of the 2 polynomials. So, because when psi 1 is 0 and psi 2 is 0 the product is 0 when either 1 of them is 0 the product is 0 when both of them are 1 the product is 1.

So, the product replaces the AND so OR how to represent OR well we have the de Morgan's laws. So, we have the de Morgan's laws. So, we have psi equal to psi 1 or psi 2 which is actually psi 1 negation and psi 2 negation the negation of the whole thing this is a de Morgan's law we use exactly this because we have already seen how to deal with negation and how to deal with AND you could just use both of that. So, we negate P psi 1 and P psi 2.

So, with 1 - P psi 1 1 - P psi 2 take the product and then negate it again. So, 1 minus this entire thing. So, this is the OR. So, what do we have at the end? We have if we do this operation successively basically we take a clause clause is an r of 3 things we use this operation and then an end of everything. So, we just take the product of the clauses. So, we get a polynomial and the degree is the degree 3 SAT instance. So, 3 CNF.

So, each clause will be represented by something of degree 3 and there are suppose there are m clauses the par the final polynomial will be 3 multiplied by m. So, degree of the final polynomial will be at most 3m. So, this is what we have and it should be fairly clear from the construction that the inputs x 1 up to x n are from 0 and 1 the polynomial evaluates to 0 or 1 depending on whether x 1 to x n that particular assignment is a satisfying assignment or not.

(Refer Slide Time: 25:18)



Sum Check Protocol

$$h_{i,a_1,\ldots a_{i-1}}(X) := \sum_{x_{i+1}\in\{0,1\}} \ldots \sum_{x_n\in\{0,1\}} P_\phi(a_1,\ldots a_{i-1}, X, x_{i+1},\ldots x_n)$$

$h_{i,a_1,\ldots a_{i-1}}(X)$ is a univariate polynomial in X

$h_1(0) = k_0$

$\sum_{\substack{x_2\ldots x_n\\ \{0,1\}}} P_\phi(0, x_2 \ldots x_n)$

$h_1(1) = k_1$

Earlier, verifier asked for $h_i(0)$ and $h_i(1)$ and then chose a path to verify.

Now: Why not ask for $h_i(X)$ itself?

Make the prover commit to more information.

So, whenever x 1 x is from this the polynomial evaluates to 0 or 1. So, and this is a very easy computation the derivation of this polynomial both prover and verifier could do this independently it is not something that is very difficult to do. And now the goal is instead of checking the number of satisfying assignments of a Boolean formula we could just say that out of all the possible x 1 up to x 1 x 2 x 3 etc taking values from 01.

We just want to sum the polynomial evaluations at these 2 power n values because the evaluations is just simply 0 or 1 depending on whether they are satisfying assignments or not. So, we just want to see whether the sum of this is 2 is equal to k. So, instead of the Boolean formula deciding whether the Boolean formula has k satisfying assignments we want to check take the sum of the polynomials at these 2 power n values 2 power n assignments.

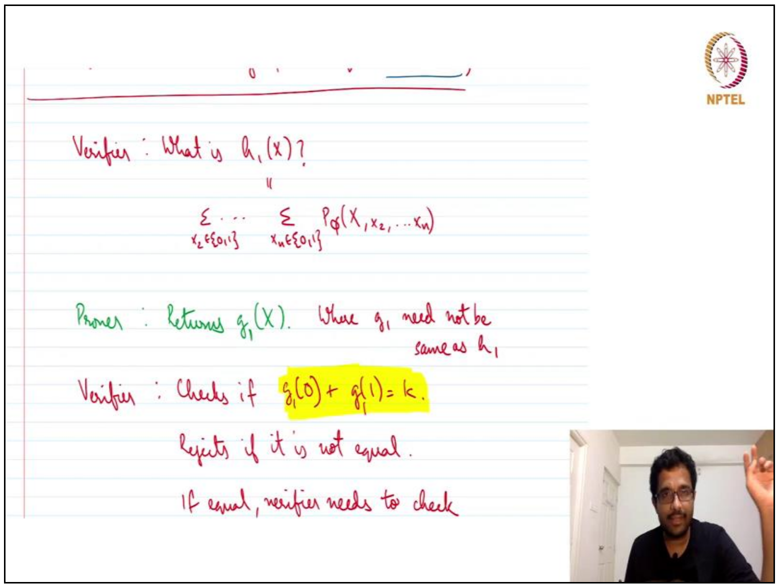
And this protocol is called sum check protocol where we will do this and moving to polynomial gives us some additional advantages as we will see in this protocol. So, we will define a new polynomial called on single variable x. So, we will call it h i of a 1 a 2 up to a i - 1 of x so what is this. So, think of the P phi polynomial suppose all the i - 1 variables are fixed first i - 1 variables are fixed to a 1 a 2 up to a i - 1.

The i - 1 variables are fixed and the ith variable is the free variable x. So, that is what the same x over here in the left hand side in the argument of h i and we want to find out how many we want to find out the sum once we fix a 1 x 1 to x i - 1 as a 1 to a n - 1 we want to find the number of or we want to find the sum of the polynomial ranging where x i + 1 to x n are chosen from the 0 1 values. So, it is exactly whatever is written here. So, it may be a bit difficult to describe the key thing is that the first i - 1 variables are fixed the i + 1 to n variables we just sum it for all possible combinations 01 combinations.

So, it just becomes a polynomial on x a single variable okay. So, the ith variable. So, that is why it is called h i. So, the x correspond to the ith variable here. So, it is a univariate polynomial in x and this will play a crucial role in the for in the protocol that is going to come up. So, earlier we the verifier asked for h i 0 and h i 1 in other words it is what I mean is. So, maybe it is like what is what is h 10 and h 11.

So, what is h 1. So, h 1 does not have any other a 0 or anything because it is from a 1 to a i - 1. So, there is nothing. So, h 1 of 0 is simply you sum up over all x i's from x 2 to x n. So, basically you sum up over everything x 2 to x n and so, sum up over x 2 up to x n from 0 1 P of P phi of 0 x 2 to x n. So, it is in other words we are asking how many satisfying assignments are there where x1 is fixed to 0.. So, this is actually k 1 sorry k 0 similarly h 11 is k 1. So, earlier we asked for k 0 and k 1 which is h 10 and h 11 and then we chose 1 of them to verify.

(Refer Slide Time: 29:43)



Verifier : What is $h_1(X)$?

$$= \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} P_\phi(X, x_2, \ldots x_n)$$

Prover : Returns $g_1(X)$. Where g_1 need not be same as h_1

Verifier : Checks if $g_1(0) + g_1(1) = k$.

Rejects if it is not equal.

If equal, verifier needs to check

So, the problem was we chose 1 of them to verify and there were way too many options for the approval to escape. So, what we will do here is not ask for h 10 and h 11 instead we will just ask for the entire polynomial h 1 x. So, notice that P phi is degree at most 3m. So, h will be also of degree at most 3m in fact if you consider x i and assume make some simple assumptions you can say that in sense that if x i appears only once in a clause at most once in a clause then the then the degree of x i will be at most m.

Because it can appear at motion it will not appear twice in a clause it does not make sense for a specific variable to appear twice in a clause. So, the individual degree of any variable is m. So in fact the degree of h x will be of at most m. So, instead of asking for the evaluations at 0 and 1 why not ask for h itself the polynomial h i itself. So, the problem the point here is that h i is a degree m polynomial. So, which means the prover will have to tell the entire polynomial. So, 1 way to describe the degree m polynomial is to give the m coefficients or n + 1 coefficients constant term degree 1 term degree 2 term up to degree m term.

And so what is happening is that the prover is giving out much more information than just 2 evaluations. So, he has to commit to a lot more information and this makes it difficult for the prover to escape like in the previous situation. And what we will do is also we will work to make things simple in to limit the size of the numbers we will work in a big field the finite field F q where q is a prime that is around 300 mn.

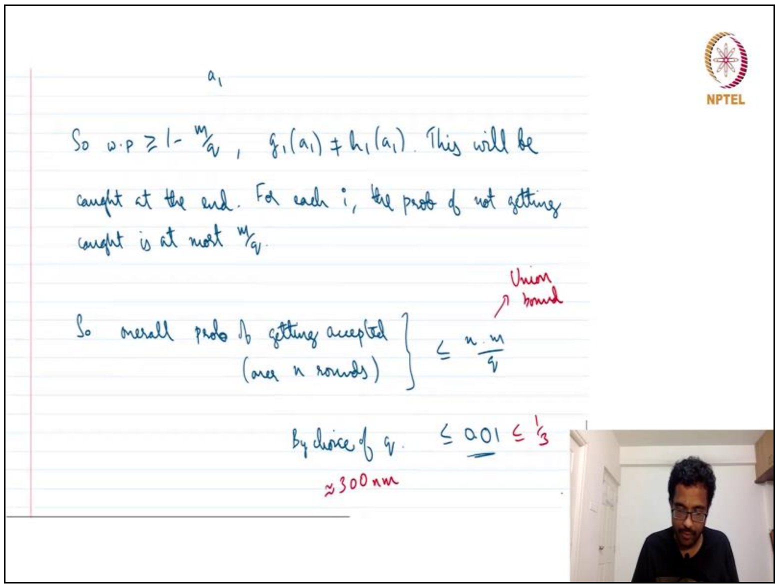
In fact even 100 mn may be enough. So, we will choose q to be a prime of the size roughly this size. So, let me say left roughly this size. So, that this will be large enough. So, that there is no this will be large enough to perform the computations at the same time this will be this will give us this is not blow up the size of the computations. So, maybe let us try to see the protocol once again.

So, verifier asks what is h 1 x which I have already described h 1 x. So, h 1 x is simply the summation over x 2 x 3 up to x n where the first variable is capital x and the rest are ranging over 01. So, you take the sum. So, now the prover will want to lie but now in this case unlike the earlier case early case here to provide k 0 and k 1 but now he has to provide a polynomial. So, perhaps he could still be smart and provide a polynomial.

So, he lets say he returns some polynomial it is a g 1 x and where g 1 need not be same as h 1 it need not be same as h 1 but he could be smart enough and craftingly produce g 1 such that g 1 of 0 and g 1 of 1 if you sum them it will it will add up to k. So, prover could craftery create g 1 such that g 1 of 0 and g 1 of 1 will add up to k because this is the next thing the verifier is going to check whether the; so, g 1 is the number of g 1 of 0 is the number of satisfying assignments where x 1 is 0 and similarly g 1 of 1 is the number of satisfying assignments where x 1 equal to 1.

So, here is going to check whether g 1 of 0 and g 1 of 1 add up to k. So let us say verifier sorry prover is up to this task and produces g 1 carefully enough. Suppose the verifier is trying to be honest then there is no problem the verifier can actually he can give the actual h 1 h 2 sorry h 0 h 1 but the problem arises when the verifier is trying to fool the prover then the number of satisfying assignments is not k. So, he will he could craft g 1 such that g 10 and g 11 add up to k.

(Refer Slide Time: 35:06)



a_1

So w.p $\geq 1 - m/q$, $g_1(a_1) \neq h_1(a_1)$. This will be caught at the end. For each i, the prob of not getting caught is at most m/q.

So overall prob of getting accepted (over n rounds) $\Big\} \leq n \cdot \frac{m}{q}$ ↗ Union bound

By choice of q. $\leq \underline{0.01} \leq \frac{1}{3}$

$\approx 300\,nm$

So, now suppose the verifier checks and this is ok now the verifier will want to indeed check whether g 1 is the correct polynomial g 1 is actually equal to the h 1. So, this is the rest of the algorithm. So, basically it is the same thing it you can think it is a recursion but just for clarity I will explain couple of more steps. So, there is 1 more aspect that I have not mentioned but I will mention it now. So, instead of asking g 10 g 11 now verifier needs to check whether g 1 is indeed h 1. So, what verifier does is picks an a 1 from the field the field contains not just 0 and 1 it has q elements F q so, 01 up to q - 1.

So, verifier picks a 1 from this field at random the; verifier computes g 1 of a 1 and let us say verifier sends a1 to the prover. Now the prover has to send. So, asks prover to give this means the protocol once it is established which is established even before the polynomial or anything the prover knows to send this h 2 a 1 of x meaning a 1 is fixed the second variable is the variable and x 3 to x n is the variable over which the summation will range.

So, prover has to provide h 2 a 1 of x. So, what is h 2 a 1 of x sorry P phi ok summation of x 3 up to x n P phi of a 1 x sorry a 1 x now x 3 up to x. So, prover has to give this polynomial. So, suppose prover gives a polynomial g to x. Now the verifier has already or can already compute g 1 of a 1 and you can verify that g 2 of 0 and g 2 of 1 should add up to give g 1 of a 1. So, this is an important identity because g 2 of 0 and g 2 of 1 should give you whenever a 1 is fixed it should give you the summation it should give you g 1 of a 1.

So, this is because maybe I will just say 1 more line since h 1 of a 1 is equal to h 2 a 1 of 0 plus h 2 a 1 of 1 because g 1 is present pretending to be h 1 and g 2 is pretending to be h 2 a 1 this has to be true. So, verifier checks this if equal fine if not equal he can reject it there if equal he picks a 2 and computes h 2 a 1 of a 2 sorry not h 2 he does not have h 2 he computes g 2 sorry he computes g 2 of a 2 and asks prover for h 3 a 1 a 2 x.

And the same cycle repeats. So, every time the prover has to. So, the challenge here is that every time the approver does not know which is the random value that is chosen from the field. So, the prover can possibly engineer so as I said before g 1 could be engineered in such a way that g is 10 + g 11 equal to k but then he is already committed to the polynomial g 1. Now prover a verifier is coming up with a random value a 1 and the prover has to or the prover in order to convince the verifier the prover has to make sure or that g 1 of a 1 is equal to the correct the correct value of h 1 of a 1.

But if g 1 is equal to h 1 then it will automatically be ensured but when g 1 is not equal to h 1 suppose when the number of satisfying assignments is not k and if the prover has been crafty in making the polynomial g 1 he cannot the point is that 2 different polynomials cannot be cannot coincide on too many values we have already seen this in polynomial identity testing I think lecture 43, 44 something like that.

The number of given f and g the number of points on which f and g coincide is at most the degree because f minus it has to be the root of the power the difference polynomial. So, there are only. So, many points where they can coincide and we are chosen the field to be large enough. So, that most of the points the number of points available are way more than the number of points where they can coincide.

So, if a 1 happens to be a point where g 1 is not equal to h 1 then the prover is doomed and even if a 1 even if the prover gets lucky here even if the prover gets lucky here and g 1 a 1 is happens to be a correct no if the prover gets lucky here even then potentially he could be caught at the end because I think if the prover gets lucky here he may get escape but then that that is accounted for in the probability.

So, the pro the point here is that the probability of the prover getting lucky is very small and at each stage suppose approval does not get lucky suppose g 1 a 1 is not equal to h a 1. Now again there is a bat the prover has to kind of lie to to and create another polynomial to make sure that it matches up with this like to keep up with this lie. So, it is like this philosophical thing where sometimes when you it is better to tell the truth because if you tell a lie then you have to make another lie to cover up for that and then another lie to cover.

So, you have to keep lying up 1 after the other and. So, there is a lot more work going on there. So, it is the same thing over here. So, so if g 1 a 1 is not a good g 1 a 1 is not equal to h 1 a 1 then the prover has to continue lying. So, at every stage the probability of him escaping is very small and at the end the verifier will just check whether g n - 1 a n - 1 is equal to g n 0 plus g n 1 which is just like what we did above.

Same thing over here and if they are equal it will also the verifier will pick n at random and check whether finally just a complete check n is actually equal to the polynomial value. Evaluated at a 1 a 2 up to a n. If all of this is fine then the verifier accepts else it rejects else the verifier rejects. So, the proof how many rounds may be n or n + 1 rounds and at each time the prover sorry the verifier is simply sending the only thing that the verifier needs to send is just the random bit a 1, a 1 a 2 and so on nothing else is needed to be sent because the verifier knows sorry the prover knows that the verifier is asking for this polynomial h 1 h 2 and so on.

So, the key thing here. So, the efficiency is let me just come to the efficiency first verifier just receives a polynomial of degree at most m and has to perform just 3 evaluations at 0 at 1 and at the randomly chosen value at a 1 a 2 or whatever the ith stage a i. And we have already assumed that we have already seen that the polynomial is something that can easily be evaluated. So, it is this is an efficient process deterministic polynomial time process.

And completeness if the number of satisfying assignments is actually k that means the polynomial sum of the polynomial over these 01 power n points is also k then the prover just this acts in all honesty every time he just outputs whatever is asked for by the verifier and the verifier is convinced. So, perfect completeness there is no possibility of him not being accepted the proof not being accepted.

So, the completeness is perfect 100 percent what about soundness. So, this was this was where we faltered last time where we saw that the probability of acceptance was still very, very close to 1, 1 - 1 by 2 power n. So, the number of satisfying assignments is not equal to k then the prover has to lie at each stage and hope that he does not get caught and the point here is that at each stage he is providing an entire polynomial and not just 2 values. So, in the earlier attempt we saw that if he provides 2 values it is easy to kind of slip away and the verifier not being able to catch the prover but when you are providing the entire polynomial thing is that he is likely to be wrong in most of the values.

So, it is a very good chance that the verifier will go to a point where the prover is the verifier will go to a point where the prover is is not did not get lucky. So, then he has to repeat the lie and then he has to repeat the lie again. So, what is the probability? So, we want we are interested in suppose x not equal to l or suppose number of satisfying assignments is not k what is the probability that the prover will get lucky or right or what is the probability that proverb will not get lucky.

So, let us just consider the step 1 and this will be easy enough to calculate proven has to send g 1 such that g 10 plus g 11 is equal to k this is what we he had to send. So, if the number of satisfying assignments is not k prover has to lie and he cannot send the correct h 1 because if the correct h 1 is sent h 1 of 0 + h 1 of 1 will not be equal to k what it will be the correct number of satisfying assignments.

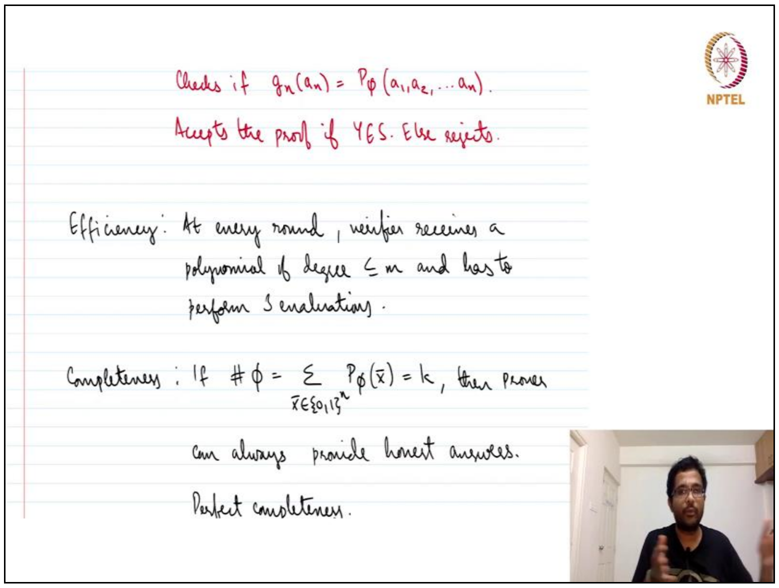
So, he has to lie. So, he gives a wrong g 1 this means that now the verifier is going to choose a random a 1 and at this value at this a 1 it is very unlikely that g 1 and h 1 coincide. So, why is this. So, what is the probability that g 1 a 1 equal to h 1 a 1 where the probability is over the choice of a 1 this is m where because m is a degree of h 1 and g 1 because the number of places where they can coincide is at most the degree.

Number of places 2 distinct m degree polynomial can coincide this is at most m because their degree m they can have at most m roots and the choice of a1 is over q values f q has q values. So, the probability of the prover getting lucky is m by q. So, this is the probability that he gets lucky in round one. With probability 1 - m by q is not going to get lucky and at each stage there are n stages because there are n variables x 1 to x n what is the probability that he gets lucky it is at most m by q. So, what is the probability that he gets lucky in any of these stages.

So, 1 is a union bound. So, 1 way is a union bound. So, he gets lucky in round 1 or round 2 and so on. So, it is the n round. So, the probability is at most n times m by q and if you remember we chose q to be something like 300 nm I think something like this I think 100 nm would have sufficed but so, basically this fraction becomes something like 0.01 some very very small quantity. So, this is the probability that the prover gets lucky in any round this is an upper bound to that probability only then can the prover succeed.

When the number of satisfying assignments is not k what is the probability that the prover gets successful it is something like 0.01. So, this is something very close to 0. So, the soundness is good here. So, it is less than 1, one-third or whatever as per the definition. So, that is the proof.

(Refer Slide Time: 50:50)



Checks if $g_n(a_n) = P_\phi(a_1, a_2, \ldots a_n)$.

Accepts the proof if YES. Else rejects.

Efficiency: At every round, verifier receives a polynomial of degree $\leq m$ and has to perform 3 evaluations.

Completeness: If $\# \phi = \sum_{\bar{x} \in \{0,1\}^n} P_\phi(\bar{x}) = k$, then prover can always provide honest answers.

Perfect completeness.

So, we have shown that by this protocol some check protocol the prover can convince the verifier or the there is a protocol for approver to convince the verifier that the Boolean formula given Boolean formula 3c in a Boolean formula has at most or has exactly k satisfying assignments and if it is not the case if the formula does not have k satisfying assignments if the number is different then there is no even an all powerful prover cannot really help to convince the verifier.

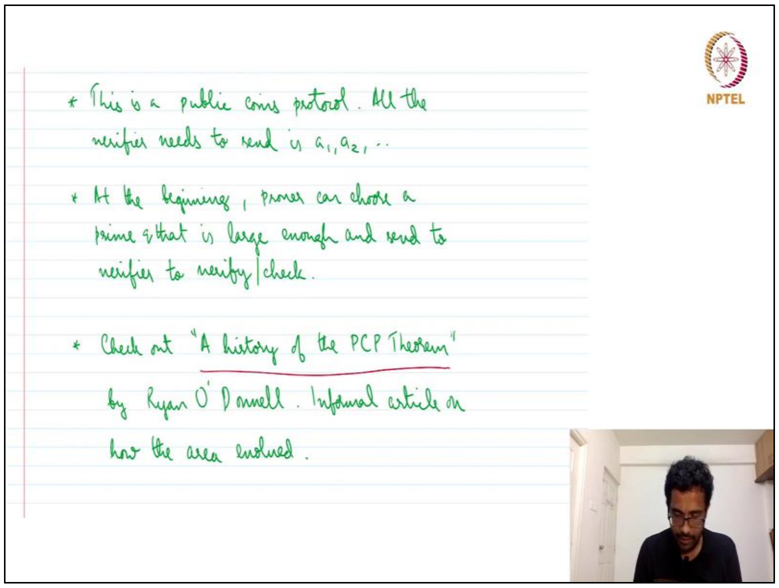
Because the probability at which by which he can convince is going to be small. So, some remarks 1 is that it's that I mentioned already at every stage the verifier just asks for just sends the random information. So, verifier sends a 1 at round 1 and verifier sends a 2 at round 2 and because approver can compute everything else brewer can compute g 2 prover can prover knows that verify is asking for h 3 a 1 a 2 whatever at each step he knows what is being sought what information is being sought.

And this is the prover all the verifier all he needs to do is to send the random coins. So, it is a completely public protocol; public coin protocol just that it is not like all the random coins are there up front. So, suppose all the random points were there upfront and the prover knows these random coins then it is possible that the prover will make the polynomial according to these random choices in which case it is not really random.

So, the prover knows this and then engineers is polynomial. So, as to succeed but that is not the case the prover commits to a polynomial g 1 and then comes a 1 then prover has to commit to g 2 and then comes a 2 and so on. So, the verifier is tossing coins and sending. So, even though the random coins are public the random coins are being generated only as and when required and this is what is helping or this is what is keeping the prover in the dark.

And now he has to commit to something that he has no clue of and this is this is making him falter with a big probability.

(Refer Slide Time: 53:20)



* This is a public coins protocol. All the verifier needs to send is $a_1, a_2, \ldots$
* At the beginning, prover can choose a prime q that is large enough and send to verifier to verify/check.
* Check out "A history of the PCP Theorem" by Ryan O'Donnell. Informal article on how the area evolved.

And so sharp 3SAT is a is a sharp P complete language. So, this implies that P to the sharp P is contained in IP because sharp 3 SAT is a sharpie complete language and we already seen that IP is contained in P space in the previous lecture. And it is actually true that IP is equal to P space this was a proof by Lund, Fortnow, Karloff, Nissan working together and at the end Adi Shamir just completed the proof building on top of a lot of work that was built up by Lund, Fortnow, Karloff and Nissan proving IP equal to P space.

How does the proof go in fact the proof it is very much like what we have seen here. So, it is exactly like this just that the polynomial has to be a bit different because we are dealing with P space. Now it is not just a 3 CNF formula it also has a quantified formula. So, how do we deal with quantifiers? How do we get a polynomial that corresponds to the quantified formula. So, in addition to these things in addition to having how to deal with negation and or etcetera we also have to deal with their exists n for all.

We can do something like AND and OR. So, for x's can be thought of or universal quantifier can be thought of as and but what it does is this will this can this results in a polynomial of huge degree and this huge degree is not something that we can work with because; so, here q was 300 mn. So, which was fine because every in every field element could be sent easily and not just field element even the polynomial only 3q number of elements needs to be sent.

So, all this is fine whereas when the degree is large it becomes very difficult like the degree itself is exponential then exponentially mini terms need to be sent. So, that is the main bottleneck for the proof that IP equal to P space and what is done is a multi linearization trick. So, because we are working in 01 it was observed that x squared is equal to x for 01. So, x cube is also equal to x and so on. So, whenever there are terms of individual degree greater than 1 it can be reduced to degree 1 terms.

And this operator needs to be integrated with the proof to get the polynomial of a sufficiently low degree and these needs to be repeated every time every time some other operation is done. Apart from that it is roughly the same roadmap and if you are interested you can read the paper and as I said this is a public coin protocol. And I said that they both do it over some prime field. So, once they know x or once they know the formula for instance the prover can help because he is all powerful he can help choose a prime number that is roughly of the correct size and sent to the verifier.

And the verifier can verify. So, prime numbers can also verify can also decide that this is a prime number at this end we can use a deterministic or randomized prime number algorithm to check that it is indeed prime. So, prover sends a prime number of the appropriate size and the verifier verifies and then they agree to work on that prime number. And all this has an interesting history this theory developed starting from the early 80's, 82, 83 or something.

And there is an article somewhat informal article called a history of the PCP theorem this article is by Ryan O Donald it is not explicitly mentioned by Ryan O Donald something just Google this title a history of the PCP theorem. It is an article it is a 10 page article but it is lot of photographs of all the parties involved and interestingly all the photographs are of those times at the time where they like now these people may be older. So, this is at that time 80's late 80's it is an easy read it is it is not much technical it is mostly in formula.

So, who proved what at what time and then some set of people proved this and somebody else proved that some and. So, on how this built up over a bunch of over a period of over the years, it is a very interesting article. So, I suggest you have a look at this yeah. So, PCP stands for probabilistically checkable proofs which are something that was very important in which played an important role in complexity theory which evolved from interactive proofs.

So, that is another interesting offshoot of interactive proofs and yeah I think that is all I wanted to say in this lecture. So, just to summarize once again we wanted to have an interactive proof for sharp 3 set. So, given sharp 3 SAT given a formula and a number k we want to the proverb to convince the verifier that this formula has k satisfying assignments. So we we saw couple of wrong attempts and then we modified our approach.

So, first we made a polynomial P phi that has a same 01 value as phi when phi is given as your Boolean input. So, p5 also evaluates the same value as phi over Boolean inputs and P phi was also easy to evaluate and degree small degree. So, this is this trick is called arithmetization maybe I should say that somewhere here. So, this is called arithmetization. So, Boolean formula is converted into an arithmetic formula with numbers and we explained how to construct this.

And then we explain the sum check protocol where we have the verifier send a random a a field element at each round and the prover has to make sure that our prover has to hope that at this field element his polynomial will evaluate to the correct value. So, there is a probability that the prover will get lucky but then this probability can be upper bounded because at each at each stage the probability of him getting lucky is small.

So, overall the probability of him getting lucky is also going to be upper-mounded. So, there is a significant probability of him not getting lucky and whenever he not he does not get lucky he will be caught out at the end. So, the main idea is that instead of just asking for 2 values we make the prover coming to commit to a polynomial itself. So, which is like degree d or m values m degrees. So, now he may engineer the polynomial in such a way that it works out correct to be in 0 and 1 but he cannot do that for all the values.

Because and now can he give the correct polynomial because if he gives the correct polynomial he will be caught at the beginning because at 0 and 1 itself will not give you the correct values. So, there is a limit to what how much he can lie. And the probability of him being able to lie is upper bounded by mn by q which is something like 0.01 which is way less than 1 third. So, it is not less it is way less than one third.

And this shows that there is an interactive through system for a sharp 3 SAT which shows that sharp P is contained in P space sharp P is contained in IP which and we also said that IP is equal to P space without and a very high overview of the proof which roughly follow the same road map but some additional ingredients are required and yeah that is it, thank you.