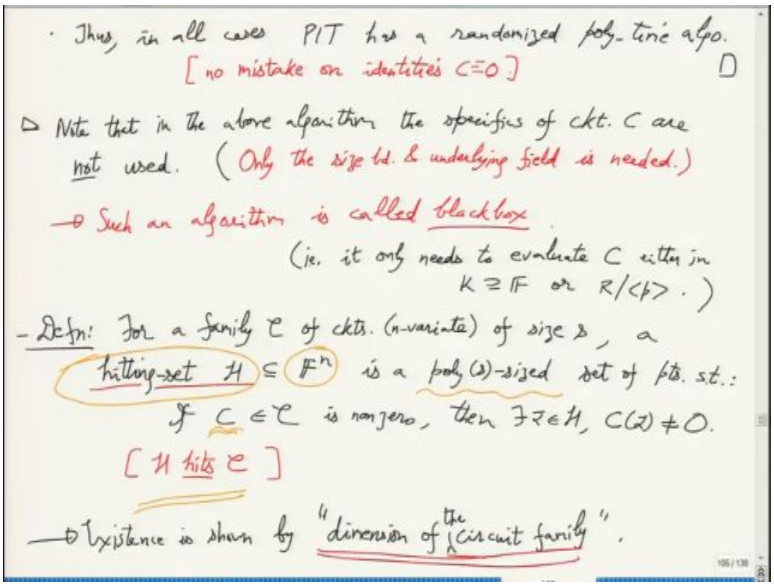
Arithmetic Circuit Complexity
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture-23
Arithmetic Circuit Complexity

(Refer Slide Time: 00:17)



So, last time we defined hitting set alternately blackbox PIT. So, ideally it should be a small set poly size in the parameter which is the size of the circuit s. So, that many points in the full space F^n such that so we will say the H is a hitting set for a family of circuit C, if for every nonzero circuit C in the family there is always a point $\overline{\alpha}$ such that C does not vanish. So, in this case we say that hitting set this set H hits C.

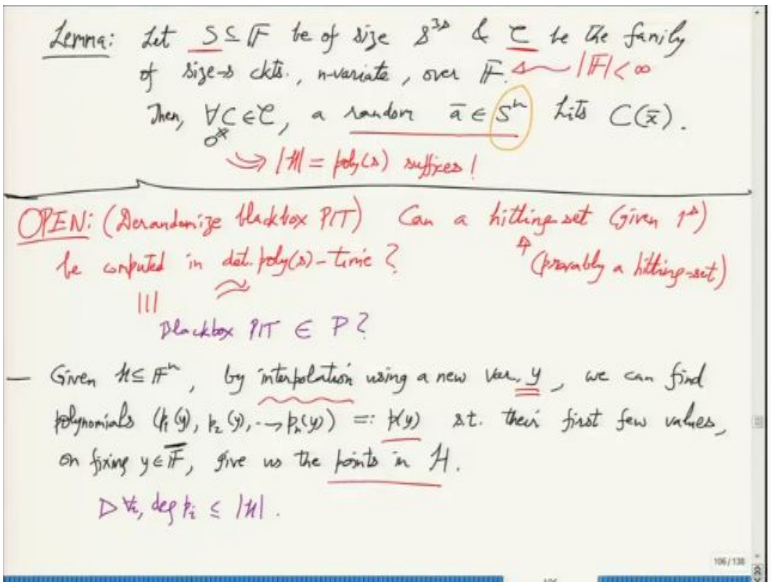
The first point is whether a small hitting set exists. Why should it exist? Because if the family if the field is infinite then size s circuits are actually infinitely many. So, why should there be a finite point infinitely many polynomials will have infinitely many roots overall. So, why should there be a finite set of points H that are non roots. Intuitively the reason is that you should do a simple back of the envelope counting then you will see that the number of circuits although is infinitely many but the effective dimension.

So, if you look at the number of free variables in your circuit they are only s because you have only s edges and on these s edges you will put s constants. So, in a way, if you think of each of these unset variables on the wires which will later on be fixed constants. This degree of freedom is only s. So, formally we can define it as the dimension of some variety. But we have done enough structural theorems to get to that it is a log s, depth, alternating addition, multiplication gates, tree. So, for that tree, you want to just fix the constants.

By dimension arguments you can actually show that there is always a small set of points which will be able to hit all possible circuits of sizes whether the field is finite or infinite that will not matter because it mainly depends on the degrees of freedom that you have not on the actual number of circuits. So, we will not formally do that a simplified question is there in the assignment for finite fields because the dimension argument actually requires algebra geometry if you do want to do it precisely, that is beyond the scope of this course.

But for finite fields you can do a simple counting you can actually count the number of circuits that will be finite and compare that with the number of possible zeros and show that there is a small hitting set so, existence is shown by the dimension of the circuit family which we will leave for now undefined. Just intuitively whatever you think of dimension geometrically.

(Refer Slide Time: 04:34)



And by Schwartz Zippel you can do something with Schwartz Zippel you know that if you just pick a random a in S^n where S is big enough. S is basically more than the degree of the circuit. By Schwartz Zippel you know that if you pick a random point then it will be able to hit. So, this basically shows that there is a hitting set of size 1. Does it show that? Not quite, no here is the statement random $\bar{a}$ will hit a given circuit. So, if suppose you had assumed that this field is finite.

So, size of F is, let us say, so this is hitting only one that is true, this probably was not a good idea. Let me skip this part. The correct Lemma statement is given in the lecture notes. So, actually this has to be corrected by saying something more. You also need the field to be finite so that you can do a counting. Assume a finite field so that you can actually count how many circuits there are? And the second thing you will need is from this you will have to deduce from the above statement by Schwartz Zippel to deduce that hitting set of size poly(s) will suffice.

So, not just one point but if you take enough points bounded by poly(s) then they will be able to actually hit all these exponentially many circuits, but I will skip the proof of this. I think something more interesting is asked in the assignment. So, with that, all these other statements will imply. So, the question that is of interest and that is open is the following. It is not about existence but it is about actually constructing or designing a hitting set.

So, this is also called derandomization question of PIT, so in fact to be precise blackbox PIT. So, can a hitting set be computed given 1^s parameter, size parameter. So, can a hitting set be computed in deterministic poly s time? This is the question which is open, the key word being this determinism. So, you know that these hitting sets exist and you know that there is a practical way of generating them. But the questions which are open is how do you design it in deterministic polynomial time.

So, remember that designing also means verifying. So, not only you have to output a set of points, but you also have to be, there should also be proved that this is a hitting set. You cannot just randomly guess the points and make a claim that this is a hitting set you have to give a proof of it. So, verification goes with this provably, it should be done provably. So, it is a design and verification question or you can just ask whether blackbox PIT is in P, it is the same thing.

So, alternately these are the 2 algorithmic equations they are equivalent to. It might be easier to show that whitebox PIT is in P, but that is also open, so that is a question of lesser interest. Although if you have done the course CS640 there we proved that even if you do whitebox PIT is in P it gives you lower bounds. So, that has complexity lower bound consequences. So, in that sense, even whitebox is interesting. A side remark we will not go into that will only study blackbox PIT for now there is an equivalent concept.

So, instead of thinking of a set of points, you can actually use interpolation and look at a single point. So, given this hitting set H so by interpolation using a new variable t, so introduce a new variable t and use interpolation. So interpolation just means that you have these set of points H look at the first coordinate of all of them. These are just field points field elements. So, design a polynomial let us say $p_1(t)$ such that $p_1(1)$ will be the first number $p_1(2)$ will be the second one and so on.

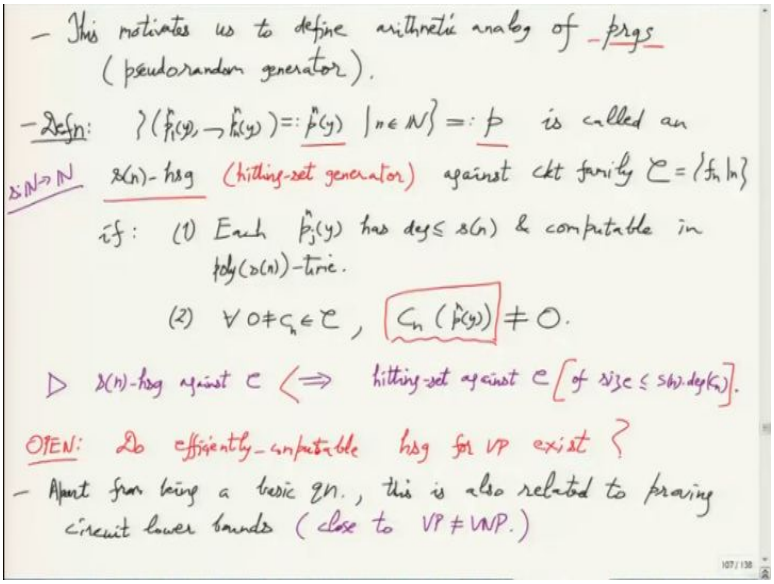
$p_1(t)$ will produce all these values in sequentially. Now, you may be working in a finite field, so, you may not have 1 2 3 but whatever so, you just pick field elements and some sequence of them and p_1 in order on these points should evaluate to your first coordinate of points in H that is what is meant by interpolation. What will be the degree of p_1 ? What you get is called a hitting set generator because it is a single function which can produce not only points in H, but infinitely many points.

You can just keep evaluating it and you will get newer and newer points. So what is the degree of p_1 ? What does? No, not unique. It will be equal to size of H. No, it is not weird. We want $p_1(i)$ to be the ith points first coordinate. It will actually be the size of H. We do not care about what exactly the values are? We just care about sequence matching. So, by interpolation we can find polynomials p_1 let me use y instead of t. $(p_1(y), p_2(y) \cdots p_n(y))$ and we will call this vector of polynomials $p(y)$.

What we have done is $p_1(y)$ interpolates on the first coordinates of H, then $p_2(y)$ interpolates on the second coordinates of H and so on such that their first few values on fixing y from F give us the points in H. You should just think of the values of this vector of polynomials as reproducing the hitting set points, if the field is not closed, so, I mean for in this definition actually you can even take the field to be smallest, size 2 because you can ask y to be fixed in $\overline{F}$, algebraic closure.

Algebraic closure is always infinite. So, irrespective of what the parameters are, you can always look at just this single vector $p(y)$ says that on some given sequence of field elements, it will evaluate 2 points in H, it will produce the points in H. So it is an easy thing to show that degree of p_i for every i is at most the size. So the number of points in H, is that clear? Hence interpolation motivates the definition of hitting set generator.

(Refer Slide Time: 16:32)



So, this motivates us to define an arithmetic analog of something which is studied in Boolean word, which is called pseudo random generators. So, have you taken any course in prg? Not yet. So prgs something older than algebraic complexity, it is already a fundamental concept both in cryptography and complexity theory, pseudo random generator. Obviously, we do not have constructions known for optimal prgs.

So, prgs have been in cryptographic terms these are again, multi valued functions which will actually take a string it will stretch it to a bigger string with the guarantee that this bigger string will look more or less random, so now defining this more or less takes a lot of work. So I will not mention that. But it is something which you also want in algorithms when you want to flip a coin, so a computer flips a coin using these prgs.

So there are no proofs that the thing being used is a prg but it is basically a heuristic function which is employed in practical algorithms but the abstraction of that is exactly prg. So prg, may be outputting a single bit, or in general, it might be outputting a string and usually it is used to stretch strings and then you can compare these questions. Many of the questions are actually equivalent. This is a whole area.

So, hence this hsgs is not really unmotivated it is analogous to what was being studied already. So let us define it. So, $p_1{}^n(y), \cdots p_n{}^n(y)$ let us put n in the superscript to signify that it is an n tuple. So, this we are calling $p(y)$ and this whole set we are calling p. So this set p is called a set of tuples is called a s(n)-hsg. So, that is a hitting set generator. So, this is always stretching because it takes y and it stretches it into n tuple. So this is also stretching and what should s(n) signify?

So, there are 2 resources: how big are these polynomial degrees and second more important resources the time taken to produce them. So, s(n) will signify both. So, it is called an s(n) hsg against some family, against circuit family C, So, recall that circuit family will have polynomials f_n for n variate. This is also an infinite family and what should happen? Respectively $p_n(y)$ or put n on top, so, $p^n(y)$ should be a hitting set for these f_n so let us write that down if each $p_j{}^n(y)$ has degree less than or equal to s(n), s(n) is a function on n.

So, function on n. So, each $p_j{}^n(y)$ has degree s(n) and is computable in some polynomial in the degree time. So, for example, a motivating case will be take s(n) to be into some constant. So, we are seeing that the degree is small and also it can be computed efficiently. Now, computation obviously means that in $p_j(y)$, you will have many coordinates, many coefficients. These coefficients are integers and you want to compute them in bits. The whole polynomial should be computable in poly time.

This is here when we say time we really mean bit operations, every bit should be computed. It is not like algebraic complexity where we will ignore the constants. If you actually ignore the constants, then these questions become trivial. It is not that easy. I mean, you actually want to compute all these integers in polynomial time. And second is, of course, for all nonzero C in your family. Let us say it is n variate. $C_n(p^n(y))$ should be nonzero. Is that clear?

So, efficiently computable tuple of polynomials such that every nonzero circuit when you substitute this tuple it evaluates to nonzero. Is it clear that if there is an hsg then there is also a

hitting set. So, why is that clear? So s(n) hsg against C implies hitting set against C of size. How much will be the size? There is a technical, no, nothing to do with the field. Somebody has given you a s(n) hsg. It is a powerful thing. Does not matter what the field is.

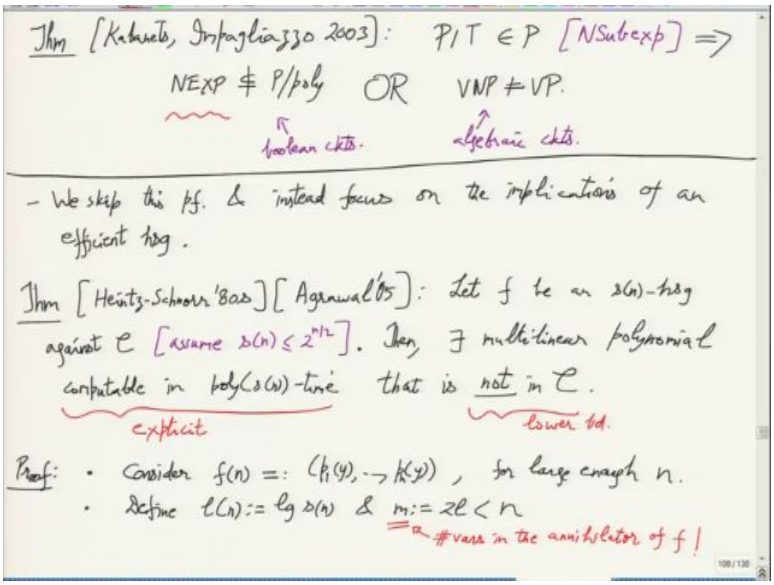
Now hitting set if the field is too small, you are free to go to an extension, but then you have to actually output points not a polynomial. So you have to output n tuples with the coordinates as constants not polynomials. So if you look at this red part, you have to look at the degree of y in this. So degree of y in this is bounded by s(n) $\times$ the degree of C_n. So that many values of y you have to substitute. So it will be of size s(n) $\cdot$ $\deg(C_n)$. That is an additional technicality.

It is an important technical point because If your family has exponential degree circuits, then these 2 are not equivalent because $\deg(C_n)$ is exponential. You are getting a hitting set of exponential size which is hit a big deal, C_n is a circuit in your family. If you are looking at size s circuits the degree can be s^s or 2^s . But as long as you are in VP, its equivalent. So, for VP where the degrees are also restricted, we are fine.

Otherwise there are some technical points but it is not very important for our lecture or even interest. So, this will quickly give you the equivalence. From hitting set you can go to hsg by interpolation and from hsg you can go to hitting set by evaluation. This is morally correct. And now the open question becomes do efficiently computable I mean, so do efficiently computable (hsg for Vp exists?). So, this is the question equivalent to is blackbox PIT is in P.

Currently this is a big open problem even in complexity theory, the reason is that so apart from being a natural or basic question this is also related to proving lower bounds circuit lower bounds. So close to our VP different from VNP. So PIT results are basically you can think of them as algorithmic results. But interestingly you can also view them as trying to prove circuit lower bounds and the better hitting sets or hsg you can generate the stronger lower bound results you will get so, existence of an algorithm is connected to non existence of an algorithm that is the flip connection. Any questions till this point?

(Refer Slide Time: 29:47)



So, there is an advanced result which I can just, it takes several lectures to actually prove this in CS 640, it is the result by Kabinets Inpagliazzo. They showed that if PIT which is whitebox PIT if you find an algorithm for PIT. This P is actually quite strong though so the result will even work if I will give you NSubexp. Do it as an exercise. So, if you can improve PIT to I mean a PIT is in BPP. It has a randomized polynomial algorithm but that does not mean that it is a NP that also does not mean that it is a non deterministic subexponential time.

But it does mean that it is a NEXP. In fact, it is in EXP because exponential time obviously, you can just open up the circuit and check so, it is a EXP, it is a Pspace, but those things do not imply that it is in Subex or even NSubexp. So, if you prove any of these cases then what you get is NEXP is not NP / poly, so, if you do not know what is P / poly, this is basically a Boolean analogue of VP. So, in VP we are looking at algebraic circuits. In P / poly, we are looking at Boolean circuits.

So Boolean means that you are just computing a Boolean function using AND, OR, NOR gates. It is again a tree, it is a kind of, in many ways it is incomparable. But sometimes you also try to put artificial comparisons. But in many cases, it is an incomparable word because we cannot say

anything about the depth there is and there is no result on the depth reduction. But we have these simple observations that you can assume. So, you have now 3 kinds of gates.

So the or even the alternative thing is not very clear, but you can at least see that if there are 2 layers both are the same then you can merge them. So, you have AND, OR and then you have NOR gates spread around which probably you can bring to the bottom or the top. No, not the top I think bottom, so, you can do these simple things, but other than that, there is not much structure in algebra circuits we have seen far more powerful structure, structural theorems.

But what we are saying here is something very much believable, because we are saying that this in particular, I mean, it does not tell you anything about NP. So, NP may or may not be in P / poly which means satisfiability may or may not have small circuits, but NEXP will not have small circuits. So, NEXP has a complete problem which is called succinct 3 SAT. So succinct 3 SAT will not have small Boolean circuits, this is the Boolean you can say Boolean circuit lower bound, but it does not stop here, it is an OR condition.

So, either this or VNP different from VP. So, this is the full theorem. So, it says that if you have a whitebox PIT algorithm or something weaker than that some non trivial whitebox PIT algorithm then either you will prove a Boolean circuit lower bound or algebraic circuit lower bound. So either way, it is related to lower bounds, is this clear? Proven this actually requires a lot of work which we cannot do here. But if you are interested, you can follow lecture notes of CS640 and then ask me questions.

So, let us just skip this proof because it actually, believe it or not it will go through prgs. Although in the statement you do not see any prg I mean in fact in the statement you do not see any mention of randomization but still the proof will actually go through prg which is a bit unsatisfactory but anyways that currently I think is the only known proof. So, we skip this proof and instead focus on blackbox PIT. You focus on the implications of an efficient hsg. What is the relationship between blackbox PIT and lower bounds.

Obviously, because of this result, there will again be a lower bound connection, but the proof will be a bit easier and they will definitely be algebraic. There will be no Boolean stuff involved. So we will try to complete that so for hsg there are some older results from the 80s due to Heintz and Schnorr in the 80s but we will just follow more recent version due to Agrawal. So we will prove 2 things. So, first we will show that if you can, if there is an efficient hsg then you get a lower bound result.

And in the second result the second theorem will show that if you have a lower bound result then you can design non trivial hsg. So, hsg is kind of equivalent to the lower bounds. Let f be a s(n) hsg against some family C. So, we want the hsg to be non trivial. So, it is fair to assume that s(n) is not very large function it is smaller than $2^{n/2}$. If s(n) is $2^{n/2}$ or bigger then it is not an interesting assumption, you are picking a very large, very bad hitting set. Then there is no reason to expect that you will get lower bounds because it is a trivial hypothesis.

Let us assume that you have an interesting hsg against C then there exists a multilinear polynomial. There exists a polynomial which is computable in poly s(n) time, so what do you mean that polynomial is computable in poly s(n) time? So the polynomial will have exponentially many monomials. We do not want to compute all of them. So we are just talking about our favorite coefficient. So somebody gives you a monomial and asks for the coefficient, so that can be computed in poly s(n) time, obviously, if you want to compute all the coefficients, then it will take 2^n time. So we do not want that. We just are saying that coefficient can be computed in this polynomial in poly s(n) time that is not in C. In case there is a polynomial that is kind of explicit.

So this is the explicitness part. But it is outside the circuit family C, so that is a lower bound, is that clear? So this is the strength of a hsg or the implication of an efficient hsg. So if you have an efficient hsg against some circuit family then what you will get is an explicit polynomial that is

outside your family. So any idea how you will show this? So this actually has a very simple proof. What is the proof? I would not say interpolate.

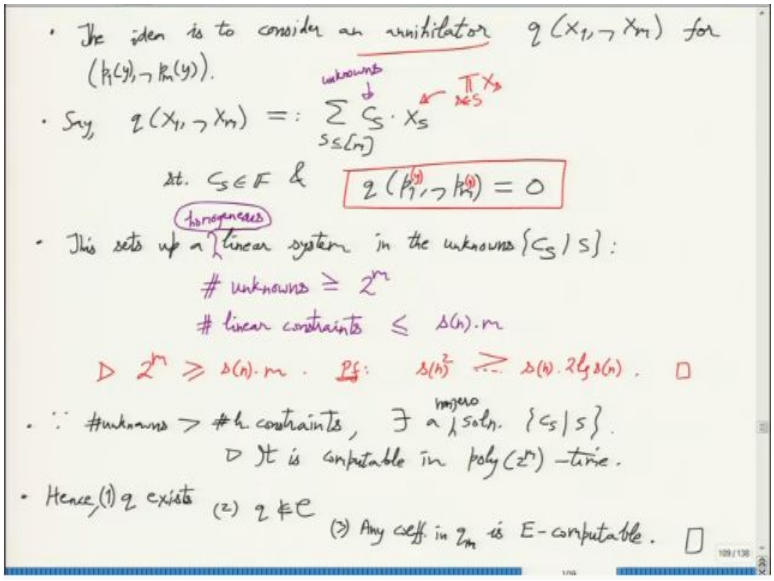
I would say the opposite. Well, I would say. Which should be called compute the annihilator. The coefficient. No, they have 2 results. So Heintz and Schnorr showed that very small hitting sets exist and are even computable in Pspace and second part of the paper actually does this, which is hitting set generator is just a sequence of univariate polynomials. It is a tuple of univariate polynomials. So you basically find an annihilator that will be n variate and when you feed this hsg it vanishes.

If your polynomial is vanishing at the hsg, it means what? It cannot be in C. So that is it. That is the proof. Well, so you have to implement this idea. So some parameters are involved. And also you want a very simple polynomial you want multilinear polynomial. So let us do that rigorously. Well, and also you want it to be explicit so why am I claiming that annihilator is explicit. So those issues we have to work out. Consider the hsg $(p_1(y), \cdots, p_n(y))$ say for large enough n just to avoid border cases.

So we want a multilinear annihilator. So if it is n variate you will have how many monomials or coefficients? 2^n. So these 2^n things are your unknowns you want to find them that may not be enough to motivate, but somehow that exponential is the reason why I will keep a log here. So log(s(n)) and m is so this will be the number of variables in the annihilator, so actually I will need to construct this hard annihilator I will use only around log s many, log(s(n)) many p_i's not all of them.

So I will only use these many because the point is that these first m p_i's which I am using, this already gives me 2^m unknowns in the annihilator and what are the constraints how many constraints are there?

(Refer Slide Time: 46:25)



Let us do that comparison. So the idea is to consider annihilator $q(X_1, \cdots, X_m)$ for $(p_1(y), \cdots, p_m(y))$? say this annihilator is it has coefficient C_S X_S it is this is just the monomial. Since we wanted a multilinear annihilator, so the monomials which appear are just they are in bijection with subsets of [m]. So there are 2^m and C_S is the unknown coefficient. These are the unknowns which we want to find and which we want to find, it should be explicit. It should not just be random fixing. So such that C_S is in the base field and q. Yes, I have been calling you annihilator which means that when you substitute $p_1 . \cdots, p_m$. What happens here? So that is the cardinal equation.

Remember this equation that q vanishes at the point at the substitution $p_1 . \cdots, p_m$. $p_1 . \cdots, p_m$ are given to you by an efficient algorithm, q is a very long polynomial and its coefficients are unknown. Actually, it is not, how long is it, how many unknowns are there, only 2^m.

The way we have defined 2^m is around $s(n)$ and so assuming that $s(n)$ was something efficient, q actually does not have too many nonzero coefficient. We can just find all of them using some linear system solver. So this red box actually gives you linear constraints in C. You solve for C if you compare on both sides coefficients of $\bar{x}$? No after you substitute; it becomes a univariate in y. So you compare y monomials both sides, p_1 is in y. This is in y. So let us write that down.

So this sets up a linear system in the unknown C_S. So let us compare unknown constraints. So how many unknowns are there? Unknowns are well exactly, they are 2^m many. And how many constraints will you get from the red box? Number of linear constraints, so what is the degree of y in the LHS? So this is well it is bounded by $s(n) \cdot deg(q)$. No, no s(n) is the degree of y in p_1 and then the degree of q is m. So $s(n) \cdot m$ exactly.

If you have a linear system which is by the way, homogeneous. Why is it homogeneous? Because if you look at the definition of q above in every monomial you have an unknowns hitting which is C_S, so it is a homogeneous system that gives you a homogeneous system. In a homogeneous linear system, you have the number of unknowns more than the linear constraints. So 2^m is greater than equal to $s(n) \cdot m$ whereas the, so let us check this. So 2^m is $s(n)^2$ and m is, what was m? 2 log s(n).

So with the right hand you will get this as greater than $s(n)$ will be greater than $2\ log\ s(n)$. So this is quite safe. This is why we needed a big enough and sn is a doing function of course. So this will ultimately happened. The number of unknowns is more than the linear constraints. So which means what? It is a solvable system. There is always in fact infinitely many solutions. Since the number of unknowns is greater than the number of linear constraints there exists a solution for C_S. There is a nonzero solution.

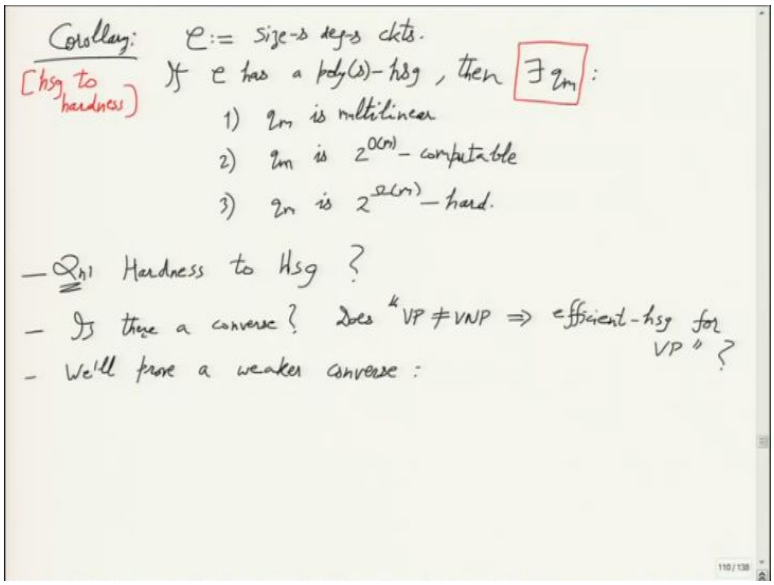
And it is computable in how much time? So this is a $2^m \times 2^m$ matrix. This linear system so polynomial in that much time and probably you cannot do any better than 2^{2m} that is a lower bound for solving this system, so hence q exists first of all and secondly q is not in C because if q was in C then, $q(p_1, \cdots, p_m)$ cannot vanish that was a hsg and third is any coefficient in q is, so q is m variate. It is q_m.

It is m variate so when you give did you ask for a monomial, its coefficient is computable in poly 2^m, so we see that this is E computable. So E is a class just below EXP, exponential time where you can solve I mean the problems which are in E are those that are solvable in $2^{O(n)}$

time, n is the input size. So q_m is of that type. You asked for a monomial the coefficient will be outputted in $2^{O(m)}$, so you have E-explicit polynomial that is hard for C.

Any questions about this proof? It is a simple proof but there are these parameters and there are some interesting specializations of this theorem also, so currently we were taking C as just some abstract complexity class. But you can take C to be for example size s circuits of degree s. So if you do that then what happens? What do you do is?

(Refer Slide Time: 57:01)



Take C to be size s degree s circuits. So if C has a poly(s) sized hsg poly(s) hsg then what do you get? Consider the proof which we just did. There the m will be around log s. So you will get a annihilator which is a hard polynomial that has log(s), around log(s) many variables and it is computable in poly(s), it is poly(s) computable basically, it is called coefficients of poly(s) computable and finally it is not in C which means that this log s variate circuit. Its circuit complexity is more than s. So it is actually exponentially hard.

So then there exists a q_m so it has many properties. So it is q_m is multilinear, q_m is $2^{O(m)}$ computable and q_m is $2^{\Omega(m)}$ hard, so you have this amazing polynomial construction. It is a E computable exponentially hard multi linear polynomial, is this clear? Yes, so this specialization is my favorite but the previous theorem actually gives you always a polynomial that will be

outside C if you have an hsg for C, then there is a polynomial outside C which is also explicit but this really is the motivating case for doing all this.

So if this is clear then let us try to study the converse of this. If you have a q_m like this, can you design a hitting set generator for VP? How do you do that? Where? did we? There was no algebraic complexity. I think I do not avoid using even algebraic circuit as a term. No because of prg simultaneous design is actually I mean it started with prg. So that is the historical motivation that in the Boolean word. There is a stronger connect between prg and lower bounds Boolean circuit lower bounds.

So hence here also, you would try to prove a strong connection. OSo one direction we have shown, this we can call hsg to lower bound hsg to hardness. Let us say and the converse question will be hardness to hsg. If there is hardness then can you design hrgs which is also called sometimes hardness vs randomness. If hardness is there then randomness cannot be there and if randomness cannot be there then hardness is. It is equivalent. Is there a converse?

For example does VP different from VNP imply efficient hsg for VP yes so such a strong thing, we will not prove. You would not be able to show that efficient hsg is something as powerful as separating VP from VNP that also does not happen in the Boolean word. So P not equal to NP is not directly connected with prg construction, So P not equal to NP is a hard or at least incomparable question with constructing prgs or derandomizing BPP but we will get close.

So we will get very lower bounds which are closed and if our hsg is very nicely behaved then we can even show that VP is different from VNP. But I will not go into those properties. So we will prove a weaker converse. So I think our time is nearly up. So let me not give the formal statement just mentioned that assuming this q_m existence satisfying these 3 properties tomorrow what we will show is an hsg which is not as good as poly s but as good as s to the log(s). If there is a hard polynomial then if there is hardness then there is a nearly polynomial time. So, quasi polynomial time which is okay, this will be a very close converse.