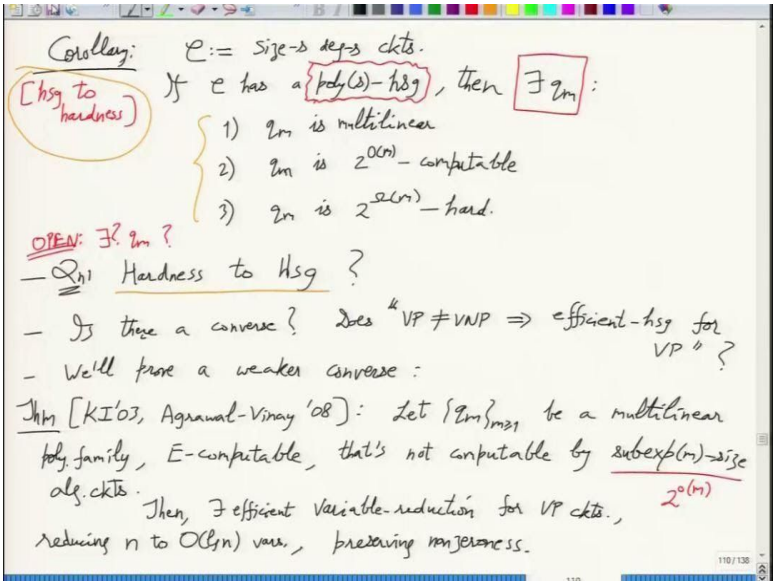
Arithmetic Circuit Complexity
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture-24
Arithmetic Circuit Complexity

(Refer Slide Time: 00:17)



So last time we discussed how hsg implies hardness. So, if you have an hsg and look at the annihilator, the annihilator polynomial will be explicit and it will be hard for that family of circuits for which you had a hsg. So, in particular, if you solve blackbox PIT for circuits, then you get these 3 properties. So, you get a multi linear E computable polynomial q_m that is 2^m hard. So, constructing this q_m currently is an open question.

So it is open whether q_m exists. Existence of this family of q_m is not known, this is implied by blackbox PIT. Now, we ask the converse question which is if there is hardness if there is such a q_m does it give blackbox PIT? Our theorem is due to Kabanets Impagliazzo. And then a version by Agarwal Vinay, so we will assume that there is a q_m.

So let q_m be a multi linear polynomial family E computable which is condition number 2 above that is not computable by sub exponential sized circuits. Obviously this polynomial is hard you cannot compute it even by cannot compute it basically sub exponential in m here

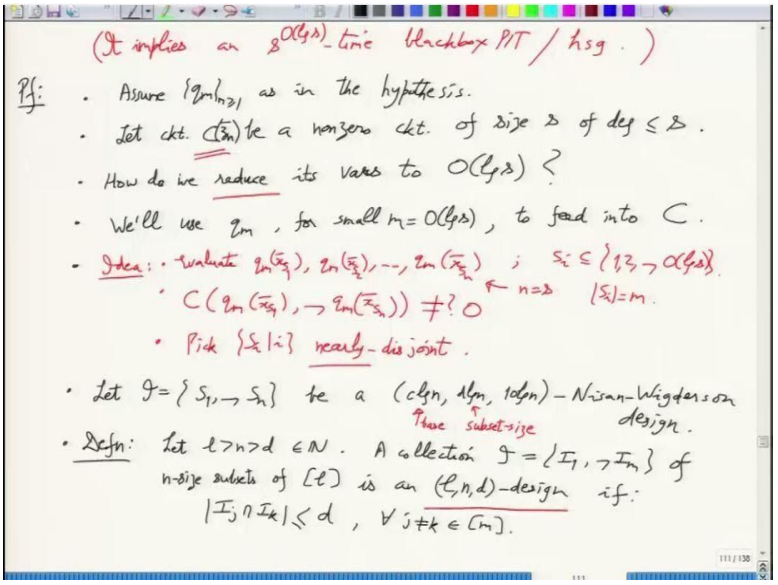
means $2^{o(m)}$ to be precise, it will require $2^{\Omega(m)}$ you cannot do it in $2^{m^{\varepsilon}}$ for example, there is no algebraic circuit of that size.

But when you look at the coefficients and the bits there they are computable in 2^m in the circuit size is also expected. We are assuming the circuit size also that is required is 2^m then, what we will show or what the theorem concludes is that there is an efficient map that will take a circuit with n variables and reduce the n variables to largest variables without changing the non zeroness of the circuit. There is a variable reduction or a dimension reduction, so that is what the theorem says.

Then there exists an efficient variable reduction for VP circuits reducing n to log n variables preserving non zeroness. Up to PIT this is a very useful polynomial time variable reduction because it will reduce the number of variables from n to log n. So, how is this related to PIT? If you have a circuit where the size is s and the number of variables is log s how fast can you solve PIT deterministically, in this polynomial the number of monomials will be s to the log s.

So, you expect just by opening up the circuit you expect it to be doable it $s^{\log s}$ time. It is a quasi-polynomial time algorithm, so this basically implies a quasi polynomial time blackbox PIT. But this is much stronger. It is a much stronger statement because it is actually giving you a variable reduction is the statement here so we will prove this.

(Refer Slide Time: 06:14)



It implies an $s^{\log s}$ time blackbox PIT algorithm which also means hitting set generator, size of the hitting set and the time in which will be computable is $s^{\log s}$ which nearly solves the blackbox PIT result but still it is not a polynomial time. It does not tell you anything about the structure of the circuit. The circuit is still blackbox seems to be only a PIT result, because this variable deduction was what is it preserving? It is preserving only one thing that is non zeroness.

So it is only a PIT, the application seems to be only PIT, I do not see any other application. So assume this q_m exists. Assume q_m as in the hypothesis and suppose somebody gives you a circuit C of sizes s computing a polynomial of degree less than or equal to s. This we can always assume because we are only looking at circuits kind of in VP. So, whatever is the size we can assume is also the number of variables.

And we can assume it is also the degree because everything is polynomially related, we can just take an upper bound s. So, now using q_m how do you reduce the number of variables in C? That is what we want to do. How do we reduce its variables to order log s? Potentially it has s variables and how do you reduce it to log s? So this by itself is a completely non trivial question. I mean, why should there be a map that is efficient and reduces the number of variables? But there is an interesting way to use q_m for this.

So, we will use q_m for very small m very small means, around log s, it is log s variate. So, for s variate circuit C we will actually use around log s variate q_m, m = log s but q_m is just one polynomial, I mean we can feed q_m in C but that is only we are just feeding it in x_1 what do you do with x_2, x_3 and x_s? How do you feed q_m in the second variable x_2, for example, if you keep changing the arguments in q_m, then you keep increasing a variable n's.

So then there is no variable reduction. So you have to start with a small set of variables, log s variables and compute q_m, for example, at different subsets of this, so that is the only thing you can do at this point. So we will use q_m to feed into C for different arguments chosen from a small set of variables, how do you design these subsets? See the number of variables you have fixed is $100 \log s$ and then you want to evaluate q_m on various subsets each of size say log s.

How many subsets are there of size log s of a superset $100 \log s$, definitely more than s. You will get enough subsets. You will get enough evaluations of q_m which you can then just feed into $x_1, x_2, \cdots, x_s$ respectively. The question is what are these subsets, what are the good subsets? Well, you can see if we take all the subsets but then the problem is ultimately you have to prove that C will not vanish at these values at these evaluation points, we have to do this carefully.

What we will try to simulate is nearly disjoint subsets. Suppose in the ideal case if all the subsets are disjoint and you evaluate q_m on these disjoint subsets then is it easy to see that C at this will not vanish. But then that we cannot afford, because there are so many variables in C so, we cannot afford so many disjoint subsets because you are evaluating q_m at disjoint subsets, it is like one is x_1 the other is x_2, the other is x_3. There is no overlap and it was nonzero.

So when you evaluate it at algebraically independent things, it will remain nonzero because x_1 to x_s is exactly that it is, they are only special because they are algebraically independent. Idea is evaluate q_m at $S_1, \cdots, S_n$ where these $S_i \subseteq \{x_1, x_2, \cdots, x_{O(\log s)}\}$, from this big set of variables we will pick subsets let me be precise with the notation.

So, this will be $\bar{x}_{S_1}, \bar{x}_{S_2}, \cdots, \bar{x}_{S_n}$; $S_i \subseteq \{1, 2 \cdots, O(\log s)\}$. These subsets are not too large because the universe is only log s big, but there are many, these subsets are n many, you can think of n and s as the same. That will not hurt and n is also some multiple of log s. So q_m has m many q_m is m variate and the size of s_1 is also m. So, size of s_i is m.

So, it is well defined, these evaluations are well defined $q_m(\bar{x}_{S_i})$. Now, if s_1 to s_n were all disjoint then C will not evaluate to 0 at this. If you want to basically achieve this that C at these values of q_m should remain nonzero that is what we want to achieve. This is definitely true when s_1 to s_n are disjoint because then the variables you are using are really different algebraically independent.

Evaluations of q_m are also independent and so C will remain nonzero. Is that clear? But we cannot afford that because that will be too many variables, we cannot achieve that with log s without the base thing that so, what we will try to achieve is nearly disjoint. So, this is a rough idea, but implementation will be complicated. First we will define what is the meaning of nearly I mean nearly cannot mean truly disjoint because you need so many subsets from a small base set. Obviously they will overlap.

How can you really minimise the overlap and to what extent it first we will define that, second part would be to actually show that although there is an overlap the circuit does not vanish at these q_m's and that will need some algebraic results. Let us try to go through the details. Any questions about the PIT? So let us say C has n variables, z_1 to z_n. These are the n variables. It is n variate.

Let I be the family of subsets s_1 to s_n. What we want in combinatorics is called Nisan-Wigderson design so $(c \log n,\ d \log n,\ 10 \log n)$ Nisan-Wigderson design some people have heard about this. Because this is also used in Boolean complexity which we have seen CS640. So, let me define it. Basically, before definition I should say that this the first parameter here refers to the size of the base set.

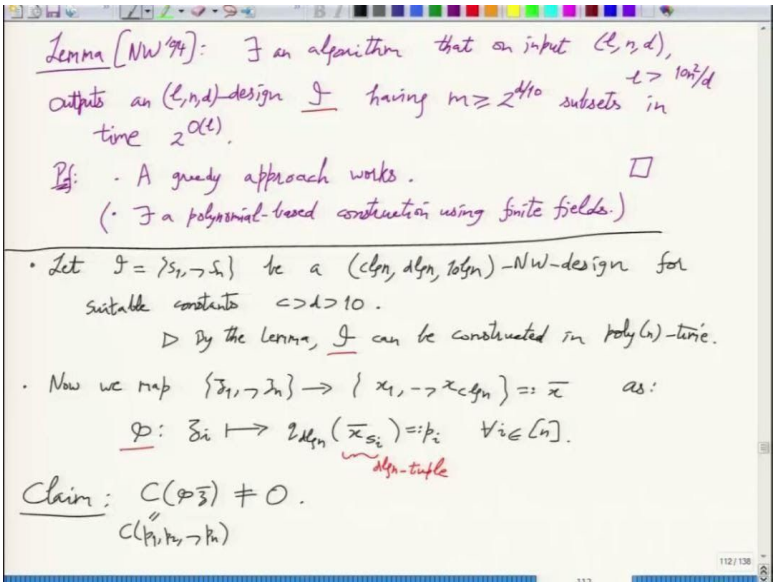
So there are $c \log n$ variables overall, $d \log n$ refers to what is the size of the subset you want. That is the base or universe. This is the size of the subset. Obviously, $d < c$, these are absolute constants. And finally, $10 \log n$ refers to the intersection size, so $10 \log n$ bound is the overlap, we want a lot of as much as possible as many as possible $d \log n$ sized subsets of $c \log n$ universe with intersection bounded by $10 \log n$, this is what design is.

Now let us now formally write the definition. Let $l > n > d \in N$, are natural numbers. A collection $I = \{I_1, \cdots, I_m\}$ where m is something else, some other parameter. So a collection I of n size subsets of the universe 1 to l is called (l, n, d)-design if $|I_j \cap I_k| \leq d; \ \forall j \neq k$. (l, n, d) - design is just a family of subsets of a universe 1 of the universe 1 to l such that the subsets all of them of equal size n and intersection are bounded by d.

When you look at this general object (l, n, d)-design, what would you think of l if d was 0 if you want d to be 0, which means disjoint subsets, l has to be nm. You cannot work with l smaller than nm. So if d was 0 and similar arguments you can make if these are constant. If you want the intersection to be very small, then your l will blow up. And we do not want that because our m is actually large in the application, if you see m, if you see it is exponentially bigger than l, n, d.

We actually want l to be exponentially smaller than m something like log n. For that, we have to actually increase d also to log n. And then there is a theorem that such designs exist, if l, n, d are comparable then designs exist. That has many proofs. We will skip all of them. You can see that in some other place.

(Refer Slide Time: 23:15)



Let me just state the lemma by Nisan Wigderson 94. In fact it is a constructive proof, there exists an algorithm that on input l, n, d assuming obviously $l > n > d$ but also l sufficiently large. So $l > 10n^2/d$. If this 3 tuple is given to you the algorithm will output a design. Design I having a lot of subsets, so having m greater than equal to $2^{d/10}$.

So exponential in d subsets in time, which will be quite reasonable to just be 2^l That which is the number of subsets of your base set or the universe set. So, essentially in the algorithm you are allowed to go over all the subsets of your universe and out of these subsets, it will actually identify m subsets, which are (l, n, d)-design in these subsets are of size n all of them.

So can you guess the algorithm? Here you can also just use a greedy algorithm. You just start with an n size subset and then try to grow this as much as you can and you will there is a probabilistic argument saying that your greedy algorithm will continue to grow till you reach m many subsets it will not stop. It is an explicit algorithm. It is completely explicit, you can go home and implement and see if you want structural details that they are absent.

It does not tell you any structure of or how is this working. A greedy algorithm or approach works. It is an interesting exercise and there are algebraic constructions. Yes, they are beautiful algebraic constructions, which have, in some sense, better parameters. But what this

greedy approach will give you will also be optimal in a way and I can mention there is also polynomial based construction over finite fields.

There are at least 2 proofs, maybe more which will give you such designs. The remaining details I skipped, because this is really a combinatorial thing, it's a detour on what we are doing. Now going back to our hard polynomial q_m. For that we need these, this design where the subsets are nearly disjoint and many and using a very small base set let φ. We continue where we left our m becomes n I think.

So, there is some confusion between the lemma and the application, but s_1 to s_n. We want n subsets which is a c log n, d log n, 10 log n design Nisan Wigderson design for constants, suitable constants $c > d > 10$. And you also wanted that property this l greater than $10n^2/d$. What does that give you? $c > 10d^2/10$.

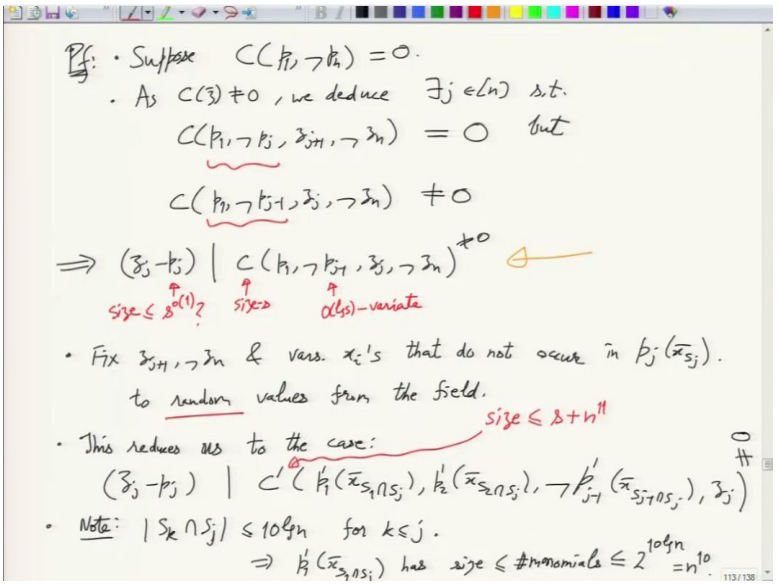
Which is satisfied trivially so, that isn't a condition, extra condition. Just pick constant $c > d > 10$ and apply the above level and there is a design. By the lemma I can be constructed in how much time so $2^{\log n}$, so poly(n) time. So the construction is free. You had n variables or size s circuit for that in poly(s) time you can construct the design.

There is an algorithm and you were already given by someone q_m. So just evaluate q_m on the design and claim that this is the map, variable deduction map so the construction is over. Now we map z_1 to z_n variables to very few variables, x_1 to $x_{c \log n}$ it is called this $\bar{x}$ as, so that I will be mapped to? So, this variable reduction map is I will call it φ. So, where does z_i go to directly, so evaluate $q_{d \log n}(\bar{x}_{S_i})$.

This is for all i_1 to i_n. s_i is of size d log n, so this $\bar{x}_{S_i}$ is actually a d log n tuple. You can happily evaluate $q_{d \log n}$ at this tuple and feed this into z_i and do this for all the variables that z_1 to z_n. This says reducing n variables to exponentially few variables just $x_1, \cdots, x_{c \log n}$, c is a constant, now the surprising claim. So, even after reducing the variables exponentially fewer, we are claiming that the things in C cannot cancel out.

So, $C(\varphi \bar{z})$. That is the claim. So, let me introduce some notation. Let us call this p_i and then this is $C(p_1, p_2, \cdots, p_n)$ polynomials. When we evaluate C at these n polynomials, log n variate we do not get 0. Do you see a proof of this? So what happens if this is 0? The only thing you can contradict with is that q is hard. You have to basically deduce from the zeroness of the circuit that q was easy. So, how is that possible? That is a really new phenomena. Suppose not.

(Refer Slide Time: 34:23)



Suppose $C(p_1, \cdots p_n)$ vanishes. So then since $C(\bar{z}) \neq 0$ and when you substitute p_1 to p_n it vanishes. We deduce that there is a j says that $C(p_1, \cdots, p_j, z_{j+1}, \cdots z_n)$ just fix the first j places that vanish. But if you fix 1 less, it does not vanish. That this is obvious because you know that if you do not fix anything then it is nonzero. So, you will from the left you will start fixing the variables and at some point nonzero will become 0. So, that is the point j, j - 1 to j transition is the place where non zero becoming 0.

This tells you what if in a polynomial you substitute something and it vanishes, then it means you found the root. So, this means that $(z_j - p_j)$ is a linear factor of $C(p_1, \cdots, p_j, z_{j+1}, \cdots z_n)$, which is nonzero. There is this nonzero polynomial whose root we have identified as p_j. Now C has a small circuit, and p_j is the root of that. What does it mean? p_j by itself is also a polynomial. It is a polynomial root of a small circuit. So, intuitively p_j should also have a small circuit.

And if p_j has a small circuit, then q_m also has a small circuit, which contradicts our cardinal assumption that q_m is exponentially, hard that is the line of argument, but you have to look at the parameters, let us actually check that. So, just wondering how to motivate why a lot of calculation would be necessary. Why is this; what I just said is not a proof. One thing is that you have to compare exactly what is the bound on circuit size of p_j that you are getting. And because for q_m what did you assume.

The assumption long time back was this that for q_m there is no $2^{o(m)}$ circuit. Now, m we have taken to be d log n, d is a constant. This is $n^{o(1)}$. So to get a contradiction, we have to actually show that this root that we go identified, the root has a circuit of size $n^{o(1)}$. Say $n^{0.9}$ no not 0.9. I mean it should be a small function $o(1)$. So, I cannot put a constant there.

Something like maybe $n^{1/\log\log n}$. It is a function that tends to 0 in the exponent. You have to show that strong upper bound on the root. But that is not possible because C, already your circuit C is small, but it is not that small it is not n to the inverse polynomial. It is only polynomial sized. It is size s and you have plugged in these p_j's. So it has an even bigger size.

So from that you cannot really hope to deduce that p_j has a very small size. So we have to work on this. This is not done yet. Let me just go through this. C we assumed has size s here and m we taken to be log s. It is s versus log s. So n and s you can assume to be the same thing. And m is log s. Your Nisan-Wigderson design was for parameters around log s. Maybe we write it down. This is size s and this p_j's are around log s variate.

And the question we are asking that for p_j is the size smaller than $s^{o(1)}$ that is the question. Let us now do some tricks here and there to get to some sort of a contradiction, we will actually not get this will not get this $s^{o(1)}$, but we will still be able to contradict the hardness of q_m. That is why we will do a lot more calculations. What we will do is first we will fix the variables. Let us fix z_{j+1} to z_n and variables x_i's that do not occur in p_j.

So, remember p_j is $\overline{x}_{s_j}$ is given by this s_j's subset. Since we are only interested in the complexity of p_j let us fix all the other variables, all the other x's and all the z which you are

seeing here except z_j. Because they are really unnecessary and a disturbance, we fixed them to random values from the base field. I mean, if you fix the variables, extra variables to random values, then your circuit non zeroness and divisibility conditions would not change.

A nonzero circuit cannot vanish at a random fixing; this is again a consequence of Schwartz-Zippel lemma. And obviously, when you do this the divisibility condition will not get violated. We are really not changing anything, we are just making our life simpler by removing the extra variables. So where does this get us? This reduces us to the case z_j - p_j is unchanged and it is still a factor of some circuits $C`\left(p`_1(\bar{x}_{s_1\cap s_j}),\ p`_2(\bar{x}_{s_2\cap s_j}),\cdots,p`_{j-1}(\bar{x}_{s_{j-1}\cap s_j})\right) \neq 0$. So, p_1 corresponds to the subset s_1.

Now, in $\bar{x}_{s_1}$ you might have fixed some of the variables and which are these? These are the ones other than s_j. So, when you fix them what remains? Only the variables which are in s_j, they remain. Even $p`_1$ has arguments $\bar{x}_{s_1\cap s_j}$ then $p`_2$ has $\bar{x}_{2\cap s_j}$ and so on. And z_j survives and this is still nonzero. Now we have this. Basically, on the RHS these $p`_i$ which we have there, variable set is the intersection with s_j and what do you know about the intersection?

It is only 10 log n; actually there was another point why we did this variable restriction. If you look at this identity, here we do not have a good bound for p_1 and p_2 because these are coming from our hard polynomial, so the bounds for this is actually bad. That is another reason why we do not immediately get something for p_j. But now, once you have restricted some of the variables, you see that now the variables are only $s_1 \cap s_j$ which is very small.

So, now p 1 prime is a polynomial on very few variables instead of original d log n, it is now only 10 log n so, there is a shrinkage in the variables set. So, what is the trivial size bound you get for p 1 prime? So, note that s k intersection s j is less than equal to 10 log n for k less than equal to j. So, this means that p 1 prime at whatever variables it has size, in fact, less than the number of its monomials.

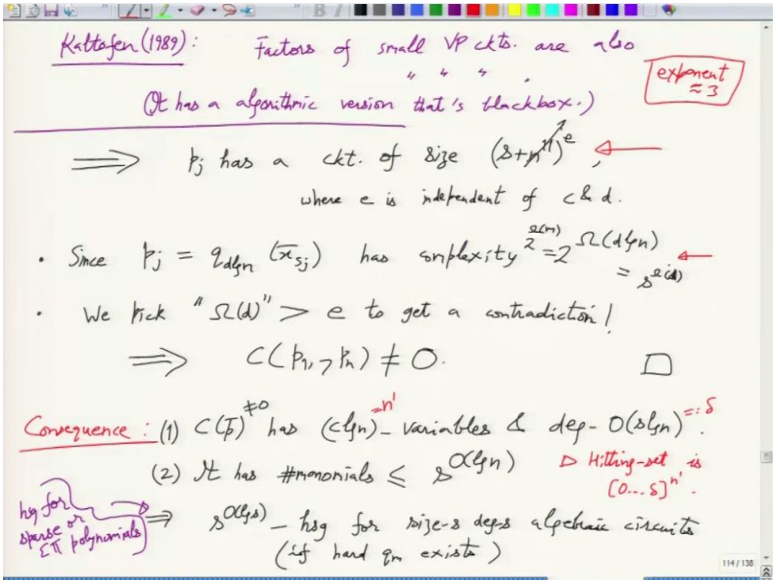
And what is the number of monomials? We started with a multilinear q m. So this is just 2 raised to 10 log n. So let us continue with this notation. So this is n to the 10. So the overlap

size basically gives you a complexity upper bound on these p i prime the trivial one, just counting the number of monomials that it has. So that is n to the 10. So, now for the whole thing, the C prime with these hard polynomials now restricted being fed in overall size is what?

For this now the overall size is the size of C which was $s + n^{11}$ certainly not more than that. That is the overall size of this RHS. If we had done the same calculation before, then we would have gotten d in the exponent instead of 11. So that I do not want. Now we have $s + n^{11}$ and there is a very fundamental result saying that for small circuits, the factors are also small circuits. So this we do in another course CS681. It also has an explicit algorithmic version as well.

This is the computational number theory in algebra, Kaltofen. This is so we do the strongest version of this which is an explicit algorithm for finding these factors, that is harder. Well, the circuit size or circuit complexity proof is much easier. That algorithm is more complicated, so a non constructive proof is easier. It is implied by whatever tools we developed much simpler. So, again I cannot we cannot afford to go into that.

(Refer Slide Time: 50:10)



So, let us just state it so Kaltofen 1989. So, this result is, factors of small VP circuits are also small VP circuits. In words it says that and quantitatively says that if you have a size s degree s circuit and you look at a factor then the factor also has size s^3 and degree obviously at most

s because there is a size blow up of fully cube not much more than that which is highly non trivial.

I mean the connection between circuit and its factors. It is not immediately clear that there should be a connection on the size complexity because we are talking about multivariate circuits. Multivariate circuits have exponentially many monomials, factors also have exponentially many monomials but why should there be a small circuit computing this. This is a highly non trivial fact. But it is true and there is an algorithm also.

It has an algorithmic version as well, that is blackbox. Which is why it is even more fundamental because it not only shows the existence but even when you give it when you are given a blackbox circuit then Kaltofen's algorithm will output all the blackboxes for the respective factors which is an amazing thing. So, it works completely at the level of blackboxes.

So, when you apply this what you will get is that p_j has a circuit of size, this $(s + n^{11})$ was the circuit size, which it was of which it is a root or $z_j - p_j$ is a factor of that circuit. And so by Kaltofen there is an absolute constant e so $(s + n^{11})^e$, which bounds the size of p_j. So where e has nothing to do with independent of is independent of c and d, e is just this constant which appears and it is around 3.

So, the exponent is around 3 and c and d we have not even fixed in the Nisan Wigderson design. Let us now; so s and n were comparable. Let me just simplify it, let me remove n^{11} from here just can replace it by s. So $(s + n^{11})$ is some polynomial in s and we just absorb it in e. So I say that p_j has a circuit of size s^e, where e is an absolute constant totally independent of c and d. For example, e may be 33 or 35. It is some fixed constant, c and d are unfixed right now, yet to be fixed.

So, since p_j which we started with is $q_{d \log n}(\bar{x}_{s_j})$ has complexity. What is the complexity that we assumed? Has complexity $2^{\Omega(d \log n)}$, we assume $2^{\Omega(m)}$ and m in this case is d log n.

So, p_j has complexity $2^{\Omega(d \log n)}$ and d is still unfixed. We can keep using d log n as a variable, now these 2 expressions. Let me make it $s^{\Omega(d)}$.

One side you are saying that p_j has complexity s^e, on the other side, you are saying that p_j has complexity more than $s^{\Omega(d \log n)}$ with Ω hiding some absolute constants, but it definitely suggests that you can pick d to be much bigger than e to get a contradiction. So that is the source of a contradiction that this e that you get from Kaltofen and this $\Omega(d)$ that you get from your premise they will contradict because d is free so you pick d suitably and then use that Nisan Wigderson design.

That is the proof so, we pick $\Omega(d)$ to be sufficiently bigger than e. This exponent which is sitting depending on d we pick big enough to get a contradiction. What does the contradiction mean? That it could not be a factor, so which means that it cannot be 0. So that is the proof of the claim which is assuming some fundamental things. I have left Nissan Wigderson design construction as an exercise and also the whole area of blackbox factoring as an exercise.

So that is a plug for the next courses sure. Existential is easier; it does not mean it is trivial. Well, you have to look at the Newton iteration. So what is the consequence of this? So the consequences of all this for PIT is that you have in deterministic polynomial time you have reduced the variables from s to log s. So, $C(\overline{p}) \neq 0$ and it has c log n variables and the degree is what is the degree so, initially the; if you assume the degree to be s, no d is the smaller one; the base set is c log n.

Yes, that is a good point it seems that we are working with d log n, but the base that is bigger that is the range where you are mapping and the degree originally or C was s, but when you plug in p you may grow it slightly because p_1 has degree around log n, degree is not too big s log n. That is one consequence. The second consequence is which follows from this. So how many monomials are there?

Well, degree s polynomial on log n variables, so it is $s^{\log n}$. Maybe in the nodes I should merge n and s. It is unnecessarily hanging around. There is no need for this so, you get $s^{\log s}$

monomials and we have not done it, maybe we can do it in the next class as a final example of a PIT algorithm, that if your number of monomials is small then you can do blackbox PIT. This actually implies $s^{\log s}$ time or let us hsg for algebraic circuits. If obviously if q_m exists; not just hard q_m, but with all those conditions that it should be explicit E explicit and multilinear.

Hence if there is hardness then there is a hsg, highly non trivial hsg. This in this implication we are actually using the hsg for sparse polynomials or $\Sigma\Pi$ polynomials, so I will show you at some point what is this hsg designed for depth-2, so just a sum of monomials. If somebody gives you a sum of monomials in blackbox there is a hitting set designed for that, in polynomial time.

Here you can do better. There is a very simple hitting set because even the number of variables is so small. Secondly, what you can do is if you take the degree to be d or $\Delta + 1$ many field elements and take cartesian products into a number of variables. That is a hitting set. So, if you call this degree Δ and if you call the number of variables $n`$ then hitting set is something like $[0, \cdots, \Delta]^{n`}$. That is a simple observation.

So just $\Delta + 1$ many field elements and try out all these possibilities for all the variables. This is the $\Delta^{n`}$ sized hsg. Here it is actually even easier. You do not need to invoke sparse PIT but that also is true that will need more work. Then we will do a similar calculation for depth 4 PIT. We will show that if you can design hsg for depth 4 then you can design the same hsg for general circuits. We will do that result and we will maybe, if we have time, we will do this sparse polynomial hsg.