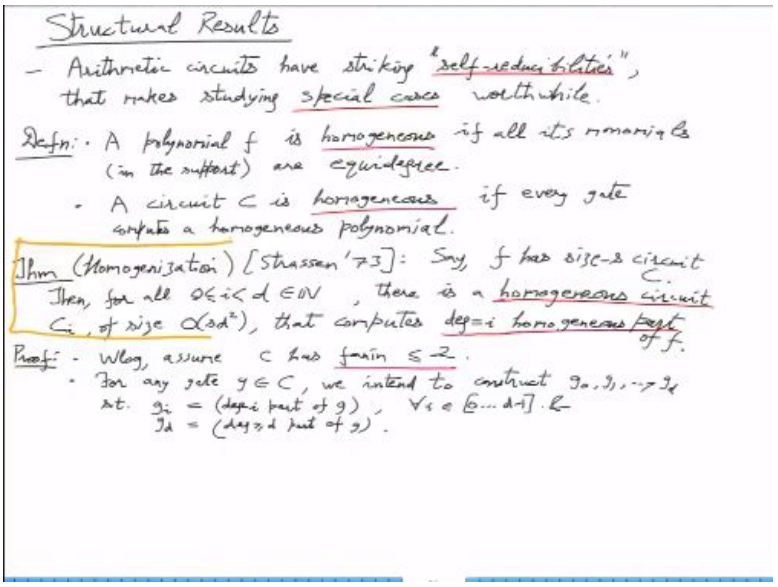
Arithmetic Circuit Complexity
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture - 07

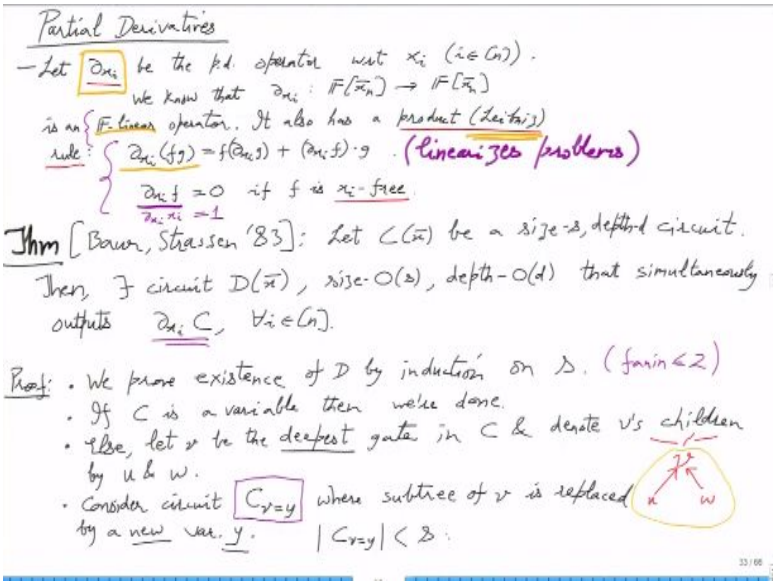
Last time we proved this theorem about homogenization that, so this theorem.

(Refer Slide Time: 00:19)



That given a circuit of size s does not matter what the degree is, degree can also be exponentially large, but as long as you are interested in lower degree parts, they can be extracted by using an efficient circuit, the size is merely quadratic, it is this sd^2

(Refer Slide Time: 00:49)



The next structure result that we will show is to do with computation of derivatives. We will use ∂x_i to denote the first order derivative of a polynomial with respect to the variable x_i. This is a linear operator and the most interesting thing is that it satisfies Leibniz rule ,which is basically the action on the product. So, the action on the product is differentiating one at a time and taking the sum.

The effect of this is essentially to linearize a problem. This actually linearizes problems. The reason why differentiation is so helpful, generally in mathematics is because of this identity. It can be make non-linear problems somewhat linear, and then you can use linear algebra algorithms to solve ultimately. And the obvious thing that if f is constant with respect to x_i .These are the properties or you can also take them as the definition.

These are the axioms defining differentiation. how do you compute this at the level of circuits? We will prove this theorem by Baur and Strassen from 83. Let $C(\overline{x})$ be a size s, degree-d circuit. Then there exists a circuit $D(\overline{x})$ whose size is $O(s)$ and depth is also roughly the same. Yes, I wanted the depth not degree.

Depth is also the same $O(d)$ that simultaneously computes all the derivatives, first order, $\partial x_i C$ .In the same size and depth, you can actually compute all the n derivatives. This is the theorem by Baur and Strassen. So actually, in the beginning, it was not even clear whether you can compute the derivatives efficiently. And this theorem is actually giving you much, much more.

How do you prove this? Most of these proofs in circuits are based on induction on size or depth. So in this case, we will do basically induction on the size, but also on the depth. We will basically I mean if you are looking at a variable then differentiation is very easy . So derivative at that variable is 1 and otherwise it is just 0. And in the leaves you have actually variables.

So you start from the leaves, your inductive argument and then gradually go to upper levels.

"Professor - student conversation starts"

Student: What is the definition of ∂x_i? **Professor**: What is the definition of differentiation? The definition of differentiation is given here. This is how it is defined. It is an inductive definition, including the linear operator thing. So these three things give you the definition of differentiation of any order.

Oh, so then maybe that it should be 1 yes and that you are not getting by $x_i \times 1$ .. So we can add the base case that defines differentiation. And this is how so you can basically implement this definition from the deepest levels of the circuit and then move, move upwards.

"Professor - student conversation ends"

The idea is very simple and the implementation is also is not hard. So, let us do that.

It will be a constructive proof. You will actually get an algorithm to construct D, efficient algorithm. So we prove existence of D by induction on the size s. So if C is a variable then obviously we are done by the definition of differentiation.

"Professor - student conversation starts"

Student: Here, will the size not depend on the number of variables because suppose u as a variable then we so the circuit then computes all the derivatives. It should be like n many, it will not have like n many? **Professor: -** So first of all, usually the way we have defined it size includes n and secondly that has I mean n has to be included in s otherwise we should simply look at the roots the outputs they are n. So order s means that well n is subsumed. **student :** So sn is O(s). Sorry.

So suppose we make it as size s and then we cut short and use circuit as size say s times n. **Professor:** No, the expressions written here are correct. So I am just reminding the definition. The definition is size includes n. It is not s times n. **"Professor - student conversation ends".**

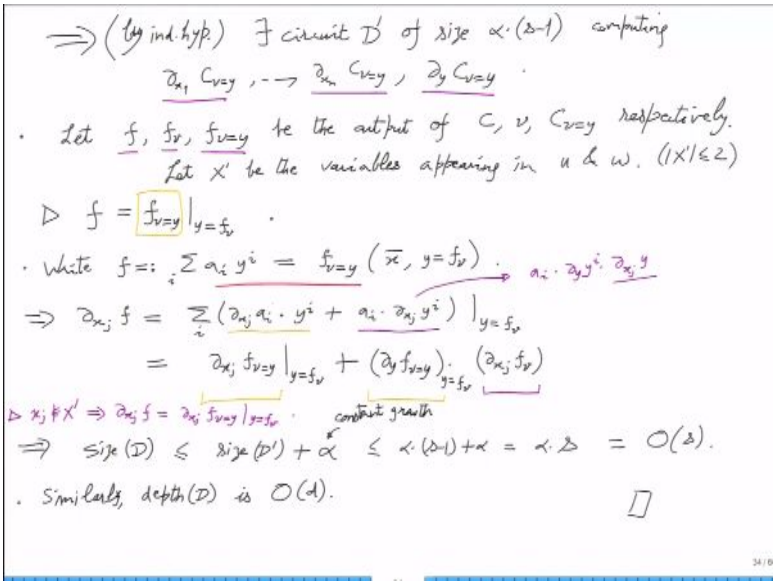
So otherwise when it is not when circuit is not just a variable so then it has things above the leaves and you pick the deepest gate, call it v.

Let v be the deepest gate in C and denote v's children by u and w. Maybe I am assuming that the fanin is 2, that is possible. So let me add that assumption, fanin is 2. You have v here. So the circuit is above v and v is the deepest, one of the deepest gates which means that u and w are not gates. They are just variables or constants. So what do you do now, for induction?

Obviously the idea should be to just surgically remove this these 3 nodes. So remove them from the circuit, replace the replace them by a variable, a new variable y. Think of it as a new variable and a new leaf. So that is the modified circuit that it has clearly smaller size because three things have been removed and replaced by one. The size has fallen. And so for the remaining circuit, you know that by induction hypothesis all the derivatives can be computed including this new variable y.

And so use, together with that use Leibniz rule. Use the axioms of derivatives to finish the proof. Let us do that. Consider circuit $C_{v=y}$. So y is now a new variable where subtree of v is replaced by a new variable y. We have defined this. That is a smaller circuit. The size of this circuit is smaller than s clearly.

(Refer Slide Time: 12:25)



What this means is that by induction hypothesis $\exists$ a circuit D' of size $\alpha(s-1)$, where alpha is a constant that we will fix later. So induction hypothesis gives you alpha times the size which is at most $(s-1)$. Size is a circuit D' computing all the derivatives. So $\partial x_1 C_{v=y}, ...\partial x_n C_{v=y}, \partial y C_{v=y}$.

So these we have. We have a circuit computing all these n + 1 thing simultaneously. So now let so now we have essentially 3 kinds of circuits and corresponding polynomials. So original was f, f_v is the polynomial computed at v inside f. So that is a very small polynomial. And $f_{v=y}$ is what remains after you remove the subtree and replace it by y. So these are the three relevant polynomials.

These are the output of C, v and $C_{v=y}$ respectively. Let us define these three polynomials. So let X' be the variables appearing in the circuits u and w, . These were the children. So either they are constants or they are variable. So X' is at most 2. We have at most 2 variables. And what can you say about the relationship between these 3 polynomials?

So in $f_{v=y}$, if you substitute $y = f_v$ what do you get? You get the original polynomial, right? This is the basic identity.This inner thing is this polynomial is in y and if you remove y by substituting f_v you will get the original polynomial. So on this let us do differentiation. let us first write $y =: \sum_i a_i y^i = f_{v=y}(\overline{x}, y = f_v)$. In y you are substituting f_v.

That is the association. this is just to define the $a_i's$. That is the definition of $a_i's$. It just follows from the above observation. So this is the equation which we now want to differentiate, So differentiating with respect to x_j will give you the result :

$$\partial x_j f = \sum_i (\partial x_j a_i y^i + a_i \partial x_j y^i)$$

where $a_i y^i$ is a product.. And we are always thinking of this as $y = f_v$ in this equation.

Let us look at the individual sums. The first one this one with Σ gives you $\partial x_j f_{v=y} \mid_{y=f_v}$ and the second one gives you the sum. And I want to write it like this ∂y $f_{v=y}$. you just differentiate w.r.t y. No, so maybe we should do it in steps first. This thing this other one instead of, so to differentiate with respect to x_j, you can first differentiate with respect to y and then y with respect to x_j.

So use that, use the chain rule that follows from the definition of derivation. So maybe I write that down. So this is equal to this is equal to the chain rule. So first with respect to y then with respect to x_j. So on this you take the sum. So then this part is independent of i. So that comes out and what remains with the sum is

$$(\partial y\, f_{v=y})_{y=f_v} . (\partial x_j f_v) .$$

That is it. You differentiate it in different ways. This new polynomial $f_{v=y}$ you differentiated once w.r.t x_j and another time w.r.t y and then you just did some simple operations. Now f_v is such a trivial polynomial, that computing ∂x_j is just some constant size being added. And both these derivatives ∂x_j and ∂y, I mean both the derivatives of $f_{v=y}$, they have already been computed simultaneously.

That is what D' did. So D' already gave you these two values and you just have to do some trivial computation to get the circuit for $\partial x_j f$. I want to deduce from this that , size$(\partial x_j f) \leq$ size (D') + O(1)

So as, you sorry, X' has to be used actually otherwise this would seem like as you vary over j, you get multiplied by n.

"Professor - student conversation starts" I think it is because you can take D', you can first substitute for $y\, f_v$ and then that will increase s by constant and then in two of the output gates we just have to modify it to r, d whatever. **"Professor - student conversation ends".**

So otherwise you do not even add. The addition is not there, actually. So that is an additional observation. Let us write it down. So if $x_j \notin X'$ then the above formula gives you, so $x_j \notin X'$ is basically you are saying that x_j is not in u not in w. So then it is not in f_v. So this part actually vanishes. So this is just a transfer.

$So,\ \partial x_j f = \partial x_j f_{v=y} \mid y = f_v$. whatever was computed by D that is just being transferred. That does not take any size.

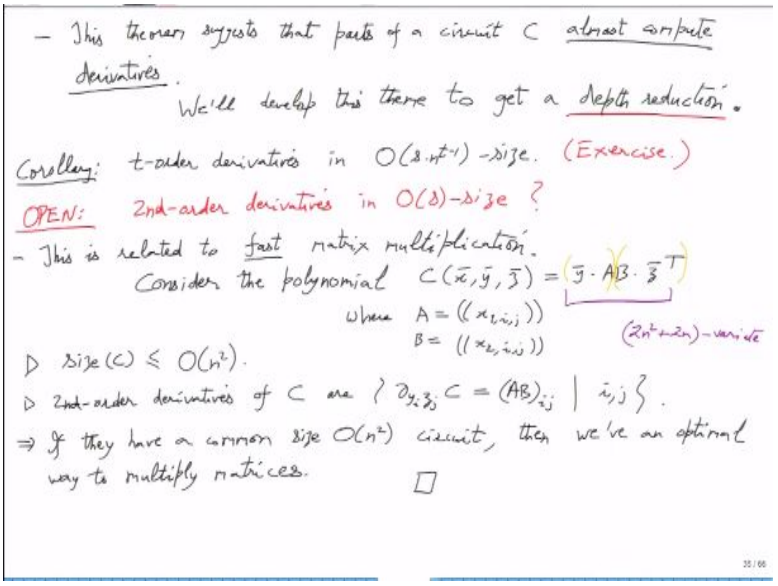
The size contribution, I mean the addition and multiplication will happen only when x_j is in X' which happens only 2 times. So actually you get something stronger. So $size(D) \leq size(D') + \alpha$

So for the two variables x_j and X' you will add multiply also. So these will be constantly many gates being added. So let us call that α. That is the constant growth. And let us use the same α in the induction hypothesis. It's the α that was yet to be fixed is this α. So this is, $\alpha(s-1) + \alpha = \alpha.s = O(s)$

And similarly, depth of D gets bounded. So that finishes the constructive proof to compute derivatives simultaneously, without changing size and depth. So it is an interesting proof although it is only using the definition of differentiation. But it is also inventing something about circuits which is that you can remove a sub circuit and replace it by a new variable. So that's an auxiliary circuit.

And you are working with that auxiliary circuit to compute the original polynomial. So that is the idea which we will now mine in the next theorems. We will develop this idea more generally, that when you look at the parts, various parts of a circuit, they are also computing some useful things in a way in this proof, they are computing derivatives already. If you look at the parts of a circuit, you can actually see derivatives. You just have to combine them in a simple way.

(Refer Slide Time: 27:03)



Let me write that down. This theorem suggests parts of a circuit almost compute derivatives. So just have to look at the correct parts of a circuit and you will be able to see by combining in a simple way derivatives. So we will develop this theme to get a very strong depth reduction property. This is a property which was never seen in other models.

This property will say that if you want to compute a polynomial of degree delta or d then there is no need to go beyond depth log d. So somebody gives you a circuit computing a degree d polynomial with depth arbitrary. Say the depth = size. So then we will give an algorithm that will modify the circuit to another circuit of similar size where the depth has come down to log d.

And that is a very surprising thing, it actually seems impossible that if originally the depth was d, how can you reduce it exponentially to log d. So we will be able to do it because of these hints which were available in this proof that actually parts of the circuit are already computing useful things.

And since in the end you are going to compute only a degree d polynomial, if somehow you are able to make sure that every multiplication gate doubles the degree then the number of multiplication gates cannot be more than I mean in a path the

number of multiplication gates cannot be more than log d. So that is how we will get to depth reduction theorem.

But before that there is an open question for derivatives. So what does this give you if you want to compute derivatives of the second order? How much size? Correct. No, not the same size. You get a multiplication. There is a multiplicative growth. Because just to look at x_i and x_j with how many i j 's are there, n^2.

So when you have this circuit D that has computed all the first order derivatives, so the circuit D which has given you all these ∂x_j 's naively what you have to do is for every ∂x_j again apply the same thing for a different or for another x_i. So there will be a multiplicative growth. So you get order O(ns) .It is a multiplicative growth.

So and then hence, I mean similarly you can go to any order derivative. So maybe I write that down that is kind of important but a simple observation. So t -order derivatives in let us say, $O(s \ . n^{t-1}) - size$. This seems fair right? For $t = 1$ you are getting s times n and then similarly. So the open question is to improve this. So let us leave this as an exercise.

And the open question is could all the second order derivatives
"Professor - student conversation starts" Will there also not be an $\alpha \times t$ type term. **Professor:** $\alpha \times t$? This is just like $\binom{n}{t}$. This is $\binom{n}{t}$. **"Professor - student conversation ends".**

So can you compute second order derivatives, all of them in linear size? So without paying much of a price, can you also go to second order from first order.

First order was a surprise and maybe the surprise continues to second order. There is no reason to disbelieve this. And you will not believe that if you do this, you will solve a very old practical problem. So this esoteric question if you solve it positively

then you have solved a very practical basic problem of the history of decades. Do you know which problem?

You will solve matrix multiplication in quadratic time, you will solve it in quadratic. So currently n by n matrix requires $n^{2.41}$ or n^{38} , .. and you will solve it in n^2 .

"Professor - student conversation starts" PIT will also be solved? **Professor:** Sorry? Well PIT is practically solved. Nobody doubts. **Student:** I mean after this PIT will also be solved because. Yeah, no but no second order derivative. If second order is done and then, third will be, Professor: Why? There is no promise. I am not sure that second order implies third order. Just like first order did not apply second order. **"Professor - student conversation ends".**

This is related to fast matrix multiplication. Anything that depends on matrix multiplication or basically any linear algebra computation depends on this. Sorry. No, that is too complicated.

Anything that any linear algebra computation is matrix multiplication based. I mean, I think the list is literally into millions, problems that reduce to linear algebra. In fact, I heard from insiders that all Netflix does to decide where to invest the money, what series to make is matrix multiplication. So they just buy billions of GPUs and run matrix multiplication to solve this. I mean, essentially it is a joke on ML.

So ML algorithms, they ultimately do matrix multiplication. If there is really a practical way to hasten matrix multiplication then it helps them. This connection may not be obvious. Let us just go through this. So we will look at a polynomial that is related to matrix multiplication and then we will differentiate it using second order derivatives to get matrix product. So let me just give the construction because once written down it is easy.

The polynomial is this. You have two matrices say for simplicity, these are $n \times n$ square matrices AB. So you are interested in A times B. So to get a polynomial out of

this you just consider the corresponding multi linear form. So you multiply on the left side by a row vector y and on the right side a column vector z. So say $A = ((x_{1,i,j}))$ *and* $B = ((x_{2,i,k}))$. So these are formal matrices.

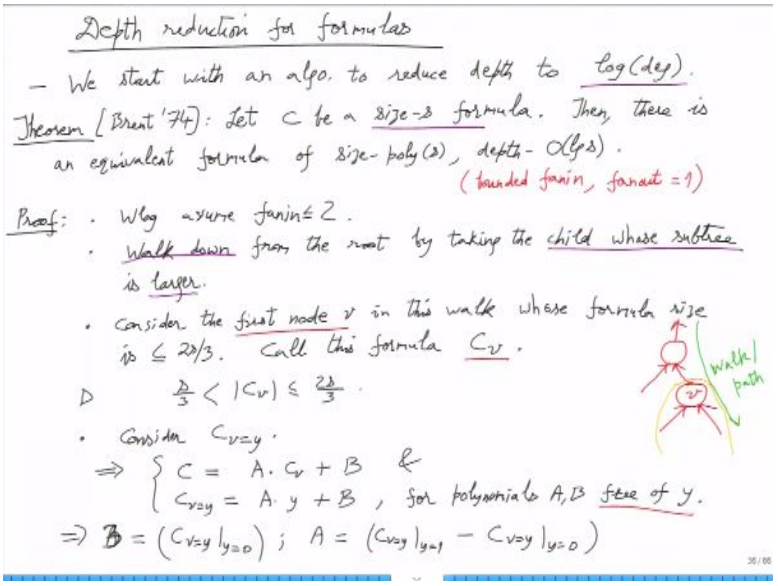
There are these lots of x_1's and x_2's and y's and z's. So x's are 2n square many and y and z are n many. So this is $2n^2 + 2n$ variate. So what is the size of this polynomial? I mean what is the smallest circuit you can think of for C? So size of this circuit, circuit size of this polynomial is at most, so it seems nq but you are wrong because the magic is that this breakup significantly simplifies life.

Use first multiply by y on the left side, so that will give you only n^2 and similarly symmetrically on the right side with B and then it is only a question of taking inner product So the circuit is order n^2. So this polynomial construction actually is not n^3, but only the size is only n^2. And what about its second order derivatives? So if you compute that second derivative, $y_i z_j$ right?

That is your product entry. So that is just $(AB)_{ij}$, for all i, j. So if you can compute all these second order derivatives simultaneously, in size C, that will be n^2. Then we have an optimal way. So this actually seems even better than matrix multiplication algorithm because this is actually a circuit of size merely quadratic. So there are n^2 variables and the circuit size is also n^2, O(n^2).

So this is actually too good to be true. But you can modify it slightly, you can maybe give it $n^{2.1}$ leeway or $n^{2.2}$ leeway and ask the same question, right? Or $n^{2.36}$ and that will still break the world record if you are into world records. Now we want to prove depth reduction results. We will start with formulas before moving to circuits.

(Refer Slide Time: 42:05)



That is also how it happened historically. And I do not think this will follow from circuit depth reduction. So let us first do this depth reduction for formulas. If you use circuits, no, so in circuit reduction, then we have to carefully check whether fanout is being preserved to 1 or not. So usually for formulas you have to repeat proofs. Because just proofs that work for circuits do not directly work for formulas because in circuits, you can just compute something using it hundreds of times.

That is not allowed for in the case of formulas. So we start with a result or even an algorithm to reduce depth to log (degree). This is the goal. So it is not clear why you would conjecture this but it was conjectured and proven by Brent. So because you have to note that the degree is, in formula's degree is like size and depth originally was size and now you are bringing it down to log of size.

So it is an unbelievable reduction in the depth. So let C be a size s formula. Then so we are starting with the assumption that it is a size s formula, no talk about depth. So depth could be as high as s. But the conclusion says that there is an equivalent formula of size will be poly s, depth will be $O(\log s)$. It will have more properties. So fanin will be bounded and obviously formula so fanout = 1.

So bounded fanin is important. This shows that there is no chance of cheating. It is really an optimal result because fanin is 2 of multiplication gates especially and still

only in log s, so minimal depth of multiplication depth required to get to degrees s is $(log\ s)$. If you are below that, then you cannot even compute degree s monomials. So $(log\ s)$ really is the minimum. So this depth is really an optimal at the scale.

It is an absolute optimum. So how is this shown? This is shown by identifying what will be later called frontier gates.These frontier gates are always multiplication gates at which the degree suddenly jumps. So say the degree at the root is d. So then frontier gates would be those multiplication gates somewhere in the middle of the circuit where the degree is jumping from d / 2 to d. That is not quite correct actually.

Maybe I should say that it will be a gate where I should actually say that it is a, frontier gate is a gate where the degree of the children $\leq$ d /2. But at this gate it grows, it goes beyond d/2. So it is the place where this d/2 is being crossed. Those are frontier gates. So these gates are somehow important, and we will work with them.

And then we will, we will also need to modify the circuit using the frontier gates. So it will be a constructive process, it will be an algorithm. The proof will be an algorithm. So we assume in the beginning that fanin is 2. It starts with this. So from the root, let us walk down looking for frontiers. So walk down from the root by taking the child whose subtree is larger. It is a randomized algorithm.

So practically this can be done, yes. Actually for formulas we will not talk about the degree. It is more about the size. Then we have to change the proof for circuits. So consider the first node. Anyways for formula size and degree are kind of synonymous. They are analogues to each other. we will talk about the size. Consider the first node v in this walk that you are doing whose formula size $\leq$ $2s/3$.

This is the first one where the formula size drops below this threshold $2s/3$. So this is think of this as a frontier node. We will call this C_v. So this sub formula we are defining it C_v. So this is let us say v and the path that you are or the walk that you are

taking is this. So you are coming down from the root towards the leaves and the first place where the sub formula size falls below the threshold of $2s/3$ is this v.

And in this walk remember that we will always take, we will always go to the child where the which is heavy. So heavy means subtree is bigger, the size of the sub formula is bigger. Well, if the two are the same, then you can just say you move to the right. Anyways, we can we could have transformed the formula so that it is right heavy. You can assume that the path always goes to the right because otherwise you can just flip that part.

So all those things you can do efficiently. So this is what we are calling C_v the sub tree or the sub formula rooted at v. And so now what we will do is we will write, we will use this frontier node to rewrite the final polynomial being computed. And let us first make this simple observation. So the size of $C_v \leq 2s/3$ and it is at least correct it is at least $s/3$.

The reason is its parent has size more than $2s/3$ and C_v is heavier. It is at least $s/3$. You can even take it strictly. And let us use the old notation of $C_{v=y}$. So that is the modified circuit after removing this v replacing it by a new variable y. The original circuit $C = A.C_v + B$. This is obvious because if the parent of v is a multiplication gate then, no that is not clear, this is not about degree.

Yes, so it is obvious by the fanout of C_v. The fanout = 1, so it contributes so it gets multiplied by some A if the parent of v was a multiplication gate and eventually might it eventually it will get multiplied, sure. So it is just $A.C_v + B$. And what is $C_{v=y}$? So you remove C v, replace it by y, i.e $C_{v=y} = A.y + B$

And what do you know about A, B? So for polynomials A, B that are free of y, so that is their specialty.

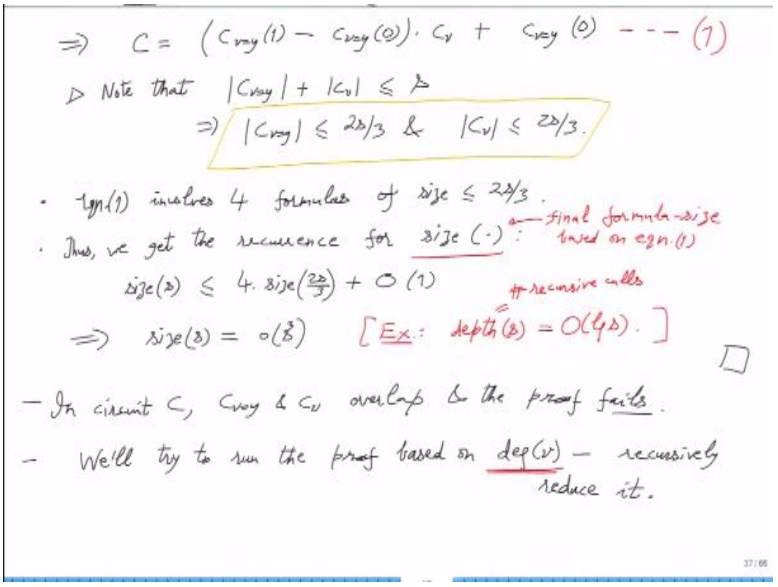
They are y free. These two identities basically eliminate A, B from this. A, B are the unknown polynomials. You can eliminate them and you can write C as just a simple

formula in C_v and $C_{v=y}$. So let us do that. So maybe in steps. So what is B? B = $C_{v=y}\ |_{y=0}$ and what is A? So A = $(C_{v=y})_{y=1} - B$ So that is it. You have A and B and so you have rewritten C in terms of these, in terms of C_v and $C_{v=y}$.

This is clearly reminiscent of the derivative construction and this is again a recursive proof. So instead of using the way C was given before you use this representation. This representation is basically C, to compute C you have to compute $C_{v=y}$ recursively and then take two different evaluations, difference of that. So an addition gate and multiply it with C_v which is again a recursive call and then add B which is again a recursive call.

Let us analyze this new representation. This can be done efficiently if you want an algorithm. But it is also enough to see this as an existential proof.

(Refer Slide Time: 57:20)



So that is the new representation, We will write C in this way $.|C_{v=y}| + |C_v| \le s$. So which means that $|C_{v=y}| \le 2s/3$, So this is, so that is the key thing. That when you are making this recursive call $C_{v=y}$ is smaller by a fraction,. So recursive call is happening on this, this small n instance.

And C_v as well. So this is this should now be taken as the motivation for whatever we are doing in this proof. So we have identified a frontier gate such that when we do

recursive calls, two recursive calls, both of them are fractionally smaller. This reminds you of merge sort. So you know that something good is going to happen in this recursion. So equation 1 involves 4 formulas of size $2s/3$.

Thus we get recurrence for the size function, let me say. Thus we get the recurrence and similarly for depth you will get a recurrence. So if you started with the formula of size(s), then this process will give you ultimately a formula of
size $(s) \leq 4.\, size(2s/3) + O(1)$. This is, so remember that this is the final formula based on 1.

You start with formula size (s) and you end up with formula size, size (s), s is being thought of as a parameter and this recursive proof will give you this recurrence and what is the solution? So size (s) is polynomial in s, sure. Certainly sub cubic, not even cube. So it is between quadratic and cubic. Correct. And as an exercise you can show that the depth function for the formula which is actually the important thing, we wanted to show depth.

So the depth of this resulting formula based on equation 1, repeated applications of equation 1. This will be the number of recursive calls , which is log s. Is that clear? So this is just a number of recursive calls. We have a super quadratic size log s depth formula which is equivalent to a given and arbitrary formula of size s. So this is a very powerful theorem. Any questions?

What is the problem if C was a circuit and not a formula? The formula will, the equations will be wrong. A and B will not be y free. And then what happens? So since A and B are not, I mean in the case of circuit or if A and B were not y free, so that we have used implicitly here. So if this B is a function of y, then this equation is not computing B at all. It is computing $B(0)$, right? Sure you can.

Correct, but then it is not the 4. It is not a constant. The recursion is now full-fledged. It is d. So the recursion is now into d instances. So this is full-fledged. So you will get

super poly. Yes, so that was an implicit understanding we had that these two equations are indeed giving you A and B because A and B are not functions of y. If they are functions of y then obviously, you are not computing B, you are computing B(0).

That is the problem. So this proof has to be drastically changed. But at least now you can make the conjecture. You can ask the question why is this not true for circuits, right? Circuits also should have a log depth. After all, they are a much stronger model.

"Professor - student conversation starts" Then you can probably count and say that there is a lot of circuits. So and it is more circuits than formulas.
Professor: No, but the counting argument is only on the size. How will counting tell you specifics about depth and fanin and fanout? These are more these are semantic details in your inside your model. You cannot see this in the count. Counting is just for the number of representations. You are just thinking of your circuit as a string, and then you just see the size of the circuit. Everything else is invisible. **"Professor - student conversation ends".**

So that is the problem. I do not think we can do it now. So let us just mention it here. In circuit C, $C_{v=y}$ and C_v overlap and the pool fails. We will try to run the same proof based on the degree and recursively reduce it. So we will instead of defining the frontier with respect to size, we will define it with respect to degree, that is just one thing.

There will be more details which we will see when we come to it and that will give a similar depth reduction for circuits as well. But in that case, there will also be more parameters. So here there was only one parameter which is s. In the case of circuit if you just work with the size (s), then the degree can be as high as 2^s. So $\log(2^s)$ is again s. So this is not impressive at all.

So what you would want to prove is that if your polynomial ultimately has degree d, then the depth required is log d. Already the semantics suggests that, you should use degree as your potential function. And we will do that. So we will define frontiers with respect to degree and then show that in the recursive calls number is only log of the degree. No well, degree is d. There are two parameters. Circuits have two parameters.

No but what he is seeing cannot be achieved if you are given size s circuit degree s does may not give you anything because the circuit maybe computing something much more than s like 2^s by repeated squaring for example. So d is a different parameter. Whatever d is, based on that you get depth. Exactly, it is useful in those cases when d is poly, poly in s.

Well, we have to see. We have to then say, make these statements that the degree of both the things are small. So we can check that. Any other questions?