

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

Lecture - 02

Okay, so we have defined arithmetic circuits.

(Refer Slide Time: 00:17)

- Fanin (resp. fanout) of a circuit is the max. indegree (resp. outdegree) of the graph.
A circuit with fanout = 1 is called a formula.
- Suppose $\mathcal{F} := \{f_i(x_1, \dots, x_n) \mid i \geq 1\}$ is a family of polynomials (call it a problem).
A family of circuits $\mathcal{C} = \{C_i(x_1, \dots, x_n) \mid i \geq 1\}$ solves \mathcal{F} if $\forall i, C_i = f_i$.
- In this case, we say that \mathcal{F} can be solved in size bounded by $size(C_n)$ & depth bounded by $depth(C_n)$.

It is a DAG with internal vertices and leaves. So leaves are the formal variables x_1 to x_n . Then there are edges and the wires with the constants and internal vertices are gates. They can add or multiply polynomials. So, we will say that an arithmetic circuit solves a problem or computes a polynomial in the following sense.

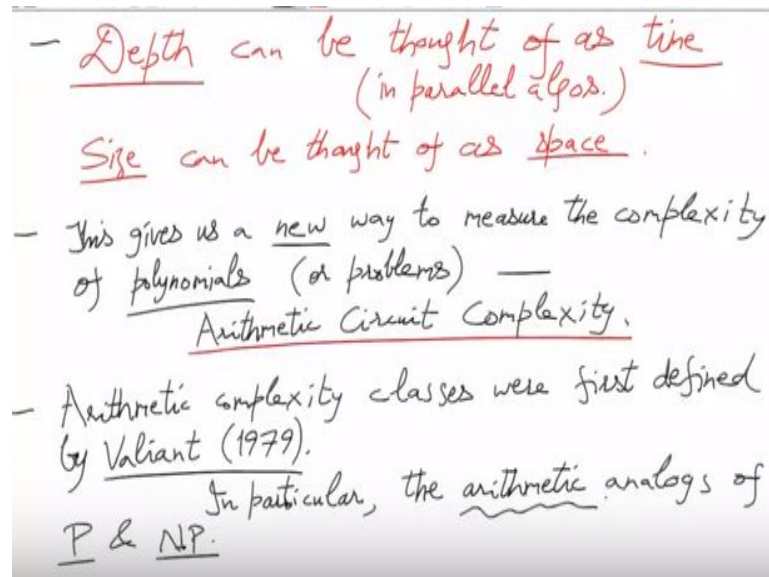
We define that in terms of family of polynomials and family of circuits. Given a family of polynomials and given a family of circuits, we will say that those polynomials are being computed by the circuits if $C_i = f_i$. So this is the meaning of solving a problem. We basically mean that an infinite family of circuits is computing an infinite family of polynomials.

Okay, so in this sense now we can talk about the resources. In this case, we can say that the set of polynomials \mathcal{F} can be solved in size bounded by $size(C_n)$. For n variate

polynomial in your family f , that can be computed by a circuit of size C_n . That is the upper bound on size of circuit. And depth is bounded by $depth(C_n)$.

The $size(C_n)$ you should see it as a function of n and $depth(C_n)$ also you should see as a function of n and these two functions basically tell you the circuit complexity of the set of polynomials f .

(Refer Slide Time: 03:11)

- 
- Depth can be thought of as time (in parallel algos.)
 - Size can be thought of as space.
 - This gives us a new way to measure the complexity of polynomials (or problems) —
Arithmetic Circuit Complexity.
 - Arithmetic complexity classes were first defined by Valiant (1979).
In particular, the arithmetic analogs of P & NP.

To compare with Turing machines, very vaguely speaking, depth can be seen as or can be thought of as time, because if you think of a set of microprocessors each being a gate addition or multiplication gate, then the time taken to compute a value of the polynomial is basically the maximum length of gates which you have to traverse.

Depth can be thought of as time in parallel, in parallel algorithms. And size corresponds to the other parameter which is space, in parallel algorithms. Well, if you try to implement a circuit on a Turing machine, then the size of the circuit will basically mean we will give you the space complexity of the Turing machine.

So vaguely speaking, you can think of depth versus size as time versus space. Okay, but formerly we have defined these things. So we will just work. **“Professor - student conversation starts”** Maximum number of gates in one layer. I mean, over all layers. **[Professor]** No space will be the overall size of the circuit. If you

implement this circuit in a Turing machine, in the Turing machine model then the bigger the size is, the more will be space requirement because all that, all those gates you have to actually take care of them in the tape. **[Student]** For Circuits, like variables may be reused somewhere in later layers. For formula like once it is used we can reuse that layer. For example, input layer, if you read the variables once we do not need that space anymore. **[Professor]** I am not sure whether you can reuse at the level of the cells in the tape. But anyways, for circuits it is this. For special models we have to think. **“Professor - student conversation ends”**.

So this gives us a new way to measure the complexity of problems. And problems now would mean polynomials. So now we can systematically measure the complexity of polynomials. And these are our new problems. So the study of this is called Arithmetic Circuit Complexity.

Okay, so arithmetic circuit complexity theory or algebraic complexity theory is the study of polynomials with respect to the size, basically size of arithmetic circuits that can compute them, okay. So arithmetic complexity classes, so based on, so once you have these measures formally defined you can now talk about complexity classes. So now you can characterize many problems in the same class depending on how easy or how hard they are in the arithmetic circuit model.

So such classes were first defined by Valiant. Arithmetic complexity classes were first defined by Valiant around in the 70s. What will be the most interesting complexity class that you would want to define? Just like P is the most interesting complexity class in the Boolean world, here the most interesting complexity class will be an analogue of polynomial time.

And once you have defined that, then the next important thing would be or interesting thing would be analog of NP. So those were formalized by Valiant. So in particular, the arithmetic analogs of P and NP. But we have to do it in an algebraic or in this arithmetic sense. So let us do that.

(Refer Slide Time: 09:17)

- Definition: VP consists of polynomial family say $\{f_n\}$, that can be solved/computed by arithmetic circuits of size $\text{poly}(n)$ & degree $\text{poly}(n)$.
- Remark:
 - Why degree condition? We don't want the circuit at a point to become too large (say over \mathbb{Q}). This is done to be able to implement such circuits efficiently (if one wants).
 - e.g. The family $\{x^{2^n}\}_n \notin \text{VP}$. Though, it is computable by $O(n)$ -size arithmetic circuits. The degree is not $\text{poly}(n)$!
 - We keep some field F in mind while defining VP. VP_F . Constants don't count in size!

So the arithmetic version of P is called VP, okay V stands for Valiant, so this is Valiant's P. So how do you want to define VP? So in P we said polynomial time, right? Here should we say polynomial depth? No so a better thing would be to say polynomial size. So we will define VP or Valiant defined VP in terms of families of polynomials so VP consists of polynomial families say $\{f_n\}_n$.

So every element of VP is actually an infinite sequence. It is this sequence of f_n 's; f_n is a n variate polynomial. And so for this polynomial family the property that you want is that it can be solved or computed by circuits, arithmetic circuits of size? So what do you want the size to be bounded by? Polynomial in n , right n is the only parameter here. So by size $\text{poly}(n)$.

So $\text{poly}(n)$ is just n to some absolute constant over all the constant right. Every constant is allowed. So it could be n or n^2 or n^3 or n^{1000} or n^{10^6} . All these are shoved together into this class. So is this enough? No variables are parameters. So whatever f_n you are looking at in this family for that f_n the size should be $\text{poly}(n)$. Yeah, so there is an additional thing.

So which may not be very intuitive at this point. So you want the degree also or Valiant wanted the degree to be not too high. So the size polynomial n is still understandable, but why is this condition on the degree also needed? So exactly why

the degree condition? So this needs some explanation and this is basically what was already suggested that you want. So once you have a circuit, which is small, small means $\text{poly}(n)$, you also wanted it to be practically implementable, right. So practical implementation means that every aspect of the computation you would ideally want to be efficiently implementable on a Turing machine or on a normal computer. So one necessary condition for that is the degree should not blow up too much.

Because in size n you have seen in the last class the degree could grow up to more than 2^n . Now if $n = 100$ then this basically means that the degree could become 2^{100} . So that kind of conflicts with time and space, poly time and space. So a good way to control that is to control the degree. So degree is restricted to $\text{poly}(n)$.

So now in every gate in your circuit for polynomials in VP first of all, there are few gates. So poly n many gates. Moreover at every gate a polynomial of degree only poly n is being computed. So when you want to evaluate your circuit at a point let us say point $x = 2$ then in no intermediate node could you get 2^{2^n} in the evaluation. So the number computation will be small.

So we do not want the circuit at a point to become too large, say over rationals. But this still is not enough for efficiency because we are not saying anything about what kind of constants are used on the wires. So constant on the wires could themselves be too large right. But that I think is fair because if your circuit already has constants which are too big then obviously if you try to evaluate it at a point, values could also be too big okay. So that is not an implementable circuit.

But as long as the constants are small, the circuit evaluation at small points this is the definition of VP. So this is something external to our model, to our algebraic model, this degree condition is external. But this is how VP is defined.

“Professor - student conversation starts” So if you wanted efficient implementation you said in earlier class that sometimes you do include constant's sizes, right?

Why was that not included in this definition itself?

[Professor] Right. So that also could be included. So you can think of algebraic size as looking only at the size of the graph. And you can think of the arithmetic size as including everything, also including the bit size of the constants. So somehow algebraic size is the first thing you would want to study. If you start looking at constants, it becomes a harder problem to study. So it is an intermediate thing which already has a nice algebraic structure. So you want to study that first. But yeah, if you want to implement everything practically then also the bit sizes should be small of the constants. That also has to be, but what happens is when you show that a problem in a polynomial is in VP polynomial family is in VP usually the constants are not too big okay. So till now this definition has been able to achieve whatever we want to achieve.

[Student] Seeing that is constant what does it mean?

[Professor] No constant only means that it is in the base field. It does not mean that it is , its bit size is constant. That's because for every n there is a circuit, you are looking at a circuit right so the constants will depend on n . They are actually, they grow like a function of n . They may grow, they may not be absolute constants.

[Student] Sir, why was the depth parameter restriction there in the definition of the VP?

[Professor] Right, yeah. So that will have a deep answer later in the course. Yes, so that Valiant did not know. But ultimately he was proven right. So the depth condition is not needed. It can be deduced from the definition of VP with a complicated proof, we will see that proof.

[Student] As of now depth is already followed by size.

[Professor] Yeah so right now trivially the depth is just you can only see it as $\text{poly } n$ but it will be far smaller. So we will see a theorem later in the course where we will show that the depth you do not need depth more than $\log n$, okay. So the depth will turn out to be far smaller than what you see apriori in the definition of VP.

So that also shows that VP's definition is the most natural one and it gels very well with the idea of parallel algorithms instead of sequential algorithms which is what Turing machines simulate. **“Professor - student conversation ends”**.

So this is done to be able to implement such circuits efficiently if one wants. That is the remark about the definition. You do not need to read the remark. But if you are curious why the degree condition then this is kind of a vague explanation. Yeah, any other question?

[Student] Yes, so x^{2^n} is not in VP?

[Professor] So let's take that as an example. The family $\{x^{2^n}\}_n$ and let us look at this sequence of polynomials where effectively the variable is only one for the n th, the n th polynomial is just it has only one variable. But the degree is actually growing with n . This family is in VP or not? By definition it is not in VP, right? Because if you just match with the definition $f_n = x^{2^n}$. And, so you can actually draw a circuit of size only order n . But it will not have the property that degree at every node is $poly(n)$, right. The degree ultimately becomes 2^n . So this is an example that looks like an easy example but it is not in VP. And the reason why you would not want it to be in VP is because if you evaluate it at $x = 2$, then it is giving you the number 2^{2^n} which is too big. This is not considered an efficiently storable number. So we want to exclude such computations. So though it is computable by $O(n)$ size circuits, right. But the degree is too high. The degree is not $poly(n)$.

The other point is the class VP will depend on a field. So we have some implicitly we have some field in mind. We keep some field in mind, we keep some field F in mind while defining VP, okay. So, for every field there will be a different VP. So ideally we should call it VP_F . Okay, but usually we will know what the field is, so will not use VP_F . But this is really a definition for VP_F .

“Professor - student conversation starts” Field constant cannot be dependent on n right?

[Professor] No the field, elements in the field, whatever elements are available. So if you are looking at the field of rationals or the field of complex, then you can use 2^n , a constant. Why not? You can actually even use 2^{2^n} . You can use arbitrarily large constants, because they do not contribute to size or degree. So constants do not contribute. That is another thing to take care of or to take note of. So constants do not contribute, do not count in size. Sorry?

[Student] $(x + 1)^n$. The constant does not grow exponentially.

[Professor] Right, yes which is fine here. No, so $(x + 1)^n$, the constant grows implicitly. In the circuit you do not use big constants but you are also allowed to use big constants. Arbitrarily big constants can be used because as I have written the definition, there is no mention of constants being counted in the size or any other resource like degree. **“Professor - student conversation ends”**.

Okay, so this is why it is a purely algebraic model, there is no, the constants are being ignored. And this is only because we want to create a good theory. So we do not want to, if we start counting also the constants and their bit sizes then we will be basically doing what people do in Boolean circuits or in Turing machines, okay. That will become more, I mean harder to analyze algebraically.

So for now we want to see what we can do algebraically. So the conjectures we make here in a way is stronger, because we are saying that when we say that a circuit does not exist, we actually are saying that a circuit using whatever constants you like, does not exist. So these nonexistence theorems are stronger in this model. Any questions?

So yeah, so you saw a polynomial that is not in VP just because of its, so the proof is formal proof is via degree argument. Final degree is so large that it can never be computed by VP.

(Refer Slide Time: 26:27)

— An interesting polynomial (family) in VP is the determinant:

$$\det_n(X_{n \times n}) := \sum_{\pi \in \text{Sym}(n)} \text{sgn}(\pi) \cdot \prod_{i=1}^n x_{i, \pi(i)}.$$

$n!$ such monomials $\sim n^2$

▷ Given a specialized X we can compute $\det_n(X)$ in P . (nearly quadratic-time!)
(Pf: Use Gaussian elimination.)

— This doesn't give $\{\det_n\} \in \text{VP}$ because the above algorithm uses division, if-then-else, permutation etc.

— Value computation vs. Function computation.

Can you think of an interesting polynomial that is in VP, a very complicated polynomial in the definition, but still it is in VP? That is a rookie question. So an interesting polynomial family. So we will skip the word family but always it will be an infinite family. So an interesting polynomial in VP is the Determinant, right. So what is determinant?

Determinant is this polynomial, we will call it Det_n . It takes an $n \times n$ matrix let us say X . So $(i,j)^{\text{th}}$ entry of this matrix is x_{ij} . So there are n^2 variables in this matrix and it is just this polynomial is a sum of some very special monomials right, which go over all the permutations on n . Look at the sign of the permutation and the monomial is just this, okay.

$$\text{Det}_n(X_{n \times n}) = \sum_{\pi \in \text{Sym}(n)} \text{sign}(\pi) \prod_{i=1}^n x_{i\pi(i)}$$

So the monomial is just of degree n . You are picking those n variables where the second coordinate is a permutation of the first coordinate of 1 to n , okay. So if you think of the matrix then you are essentially picking variables in the matrix which do not have the same row or the same column, right. So how many such choices are there? $n!$, right.

It is a $n!$ such monomials. So $n! \approx n^{n/2}$. As n grows the number of monomials in this sum grows rapidly. It is more than 2^n . There are a lot of monomials and we are summing them but with signs, right. So the constants are only plus minus one or zero, okay. So that is the definition of determinant as a polynomial and right. So this, you already know.

So given a point, given a specialized matrix, we can compute determinant in? So how fast can you compute a determinant given a matrix? At least in deterministic polynomial time. You can actually compute it even in smaller than n^3 time, sub cubic. So there are sub cubic algorithms or nearly quadratic algorithms. But that is not very important.

What I want to point out here is that it is in polynomial time. So $\text{poly}(n)$, despite the definition using n^n many monomials. So which is surprising. So clearly the algorithm polynomial time algorithm does something very special. It just, it is not using the definition directly. It is using the properties that the definition entails right. So what is this algorithm? Right. So this is using Gaussian elimination.

So use Gaussian elimination. So over a field you can use Gaussian elimination to bring the matrix X basically in triangular form and then once you have a triangular matrix you know that the determinant is just a product of the diagonal entries. That is the algorithm. Yeah, so this is basically an incremental algorithm. Or if you want, it is a recursive algorithm, but you always, usually you implement it in an iterative way.

But does that mean determinant is in VP? So this shows that determinant evaluation can be done in P. Right? But does this also mean that the determinant polynomial is in VP? The Infinite family? So Gaussian elimination actually requires division. And we cannot do division at least, by definition we cannot do it. We can only add or multiply, right.

So this does not mean; this does not give the determinant polynomial in VP. So here again I am abusing it, I should actually use the set notation because when I say determinant, I do not mean a single determinant, I mean all the determinants for every n . So this infinite family we do not deduce that it is in VP because we have used division above, because the algorithm uses division. So is that the only problem?

Well, the algorithm will use not just division but it will also use if-then-else, right? It will check whether an entry is zero or nonzero. It will also permute the rows and columns. So all that how will you do by addition multiplication gates? So almost anything that this algorithm is doing, we cannot directly implement in arithmetic circuits, right? So division, if-then-else, permutation etc.

It may be confusing at first sight but later on it will become natural. The reason why this algorithm does not mean; does not give you an arithmetic circuit is because to run this algorithm you need a specific matrix. You need a matrix of constants. You cannot run this algorithm on just a symbolic matrix. You cannot assume n^2 formal variables.

So if you think about the C program that will implement this, that C program will actually assume that you are given field constants and then it will do arithmetic in that field of constants. Not with formal variables. With formal variables yeah it will become too expensive just because of repeated division and blow ups. Yes, if you just, right; if you just do it trivially then you will be incurring a cost of $n!$.

There will be no advantage, right. But if you try to translate this algorithm step by step, then the problems are, well how do you translate division? How do you translate if-then-else? How do you translate the permutation? So it is not, these things are not really clear. So the translation is not very smooth between algorithms and circuits. So yeah, this will actually need a full scale investigation whether determinant is in VP or not. Right, this will be an additional property of determinant if true. It is not trivial. So this is; so value computation versus function computation. That is basically the difference. Okay, so when you look at a C program that is computing determinant that

is actually computing the value of a determinant at a fixed point while an arithmetic circuit computes the whole function at once. So it gives you the function as a circuit.

And then you can do whatever values you want, whatever points you want to fix. So this is much stronger than just the Gaussian elimination algorithm that we are asking for. Okay, so yeah, some of you already know that determinant or you can guess the determinant will be in VP, but we have not shown it yet. Okay, so we will prove that in the future. Let us move on to the next complexity class.

“Professor - student conversation starts”

[Student] The circuit size is itself polynomial, the evaluation will also be polynomial?

[Professor] Well, just by the definition of VP that is not implied because what if you are using crazy constants. Such large constants that so if you are using a constant that has let us say 2^n bits. So that arithmetic circuit you may not be able to just code as a C program, because you will need that much of space to store 2^n bits. But that is not how life works. So whenever we will show some problem in VP we will always get good constants and a good uniform circuit okay. So it will also be a practical algorithm.

“Professor - student conversation ends”

So we have defined VP and we have shown what is there and what is not there. So let us now move to NP.

(Refer Slide Time: 38:42)

- What is the analogy of NP (non-determinism)?
 (do a large sum.)

- Definition: Polynomial family $\{f_n\}_n \in \text{VNP}$ if

$$f_n(\bar{x}) = \sum_{\bar{w} \in \{0,1\}^{t(n)}} g_n(\bar{x}, \bar{w})$$
 where $\{g_n\}_n \in \text{VP}$ & $t(n) = \text{poly}(n)$.
 (Annotations: \bar{w} is witness, $g_n(\bar{x}, \bar{w})$ is verifier)

- If we replace \sum by \prod (in the boolean world), we get the defn. of NP.

$\triangleright \text{VP} \subseteq \text{VNP}$.

- A standard problem in VNP is permanent:

$$\text{per}_n(X_{n \times n}) := \sum_{\pi \in \text{Sym}(n)} \prod_{i=1}^n x_{i, \pi(i)}$$
 (Annotations: $\prod_{i=1}^n x_{i, \pi(i)}$ is verifier)

What is the analog of NP? And in general what is the analog of non-determinism, right? What is the answer? “**Professor - student conversation starts**”

[Student] Projection, kind of. Because in NDTM each path is like a deterministic. Right. “**Professor - student conversation ends**”. So instead of projection I will simply say sum, right. So non-determinism is basically, I mean in the Boolean world it was whether there exists a certificate.

So over all certificates you want only one. So you are kind of taking OR over exponentially many things. So here in the arithmetic or algebraic world you will just take a sum. So do a large sum. Okay that is vaguely speaking that will be the analog, arithmetic analog of NP. So how is this implemented? So again Valiant gave this definition. So a polynomial family, so again f_n is n variate for all n.

This is in VNP if you can write f_n , let us say it has variables x, if you can write it as a large sum over some other polynomial g.

$$f_n(\bar{x}) = \sum_{\bar{w} \in \{0,1\}^{t(n)}} g_{n+t}(\bar{x}, \bar{w})$$

So the sum is over the whole space. This is an exponentially large space; t_n is some polynomial in n. So t_n is not much bigger than n. So g is g has how many variables? g has n of this x and t_n of the w. So g has $n + t$ variables.

And we want $\{g_n\}_n \in \text{VP}$ and $t(n) = \text{poly}(n)$. Well $t(n) = \text{poly}(n)$ so arity of g will be $\text{poly } n$. That is again obvious. Right. So this, well, so why this definition is natural may not be obvious. You have to look at it for a while.

But, to compare it with NP, let us give these things names. So let us call \bar{w} to be a witness or certificate okay. So \bar{w} is a witness string and let us call g the verifier. So we will see that a polynomial family f_n is in VNP if f_n can be written as a sum over witnesses with the verifier evaluated okay.

So if you replace this sigma by OR then you can recover the definition of NP. Right in the Boolean case, because g you can think of g as the verifier algorithm, which will be polynomial time, polynomial time algorithm and it is just going over all the possible certificates and for one of them. **“Professor - student conversation starts”** If we just look at, at some certificate it is Boolean one.

Yeah, so or of this will be happening in the Boolean world. G will be an efficient verifier. **“Professor - student conversation ends”**. So syntactically that is the motivation for this definition. Every summand? The summand is a polynomial, which is in VNP. It is a complicated thing. It can be any, it can be determinant for example.

“Professor - student conversation starts”

[Student] Sir, polynomial is self-reducible right. It is kind of because you can write it as a projection of, no but has to be in detail **“Professor - student conversation ends”**. So yeah if we replace sigma by OR in the Boolean world, we get the definition of NP. So that is the motivation for this definition.

Yeah more than this there is not much to say that why we are using this. So we are using this solely because of this definition of NP and so now how do VP and VNP compare? What can you say immediately? Right, so is it true that VP is contained in VNP? Why is that? So if you have a VP family g_n right then you can just in the definition just ignore the witness string.

So then $f_n = g_n$ and if $g_n \in \text{VP}$, it is also in VNP. So this is, this is trivial containment. Right? But are there candidates of, are there candidate polynomials that you can see immediately in VNP? But you do not believe them in VP? Actually, currently you do not even know whether determinant is in VP. But that later on, you will see it is in VP. So determinant is out.

So what is the next candidate? So a standard problem in VNP is what is called permanent. Permanent is very close in definition to determinant, but very different in reality. So permanent polynomial, again evaluates on an $n \times n$ matrix and it is again a sum over those monomials as you saw before but what is the change? That there is no sign.

$$\text{per}_n(X_{n \times n}) = \sum_{\pi \in \text{Sym}(n)} \prod_{i=1}^n x_{i, \pi(i)}$$

So it is again a sum over all the elements of the permutation group on n elements. So again you have $n!$ monomials but there is no sign. So all these monomials appear equally, right. **“Professor - student conversation starts”** So why would Gaussian elimination fail on this? We do not have the property. What is for the negative.

If you take those three properties which allow you to do Gaussian elimination you get the determinant that kind of is unique. Right. **“Professor - student conversation ends”**. Yeah that is a stronger thing, but this just this one simple step of adding two rows or adding two columns that does not change determinant, right? But permanent you do not know or in general it will change. It will mess up the permanent okay.

So you cannot just add two rows and this was critical in Gaussian elimination. This is how the rows were reduced and you ultimately got row echelon or column echelon forms or ultimately a triangular matrix. So because that property is gone well, which

is actually the equivalent to saying that if two rows are the same, then determinant vanishes, but permanent does not, okay.

So since that property is not there we do not know, we cannot use Gaussian elimination and then other than Gaussian elimination we have no other algorithms, no ideas. Okay. So this problem is still in all the practical ways, this is the hardest problem. Okay, this is even harder than SAT in a way. Okay, so now is this problem in VNP?

So we do not expect it to be in VP because practically it is a hard problem. If it is in VP then I mean something unnatural would happen. We do not expect it to be in VP, but is it in VNP? So let us show that.

(Refer Slide Time: 50:33)

Theorem: $\text{perm} \in \text{VNP}$.

Pf: Let g be the polynomial that takes $n \times n$ & $\vec{b} \in \{0,1\}^n$ & computes

$$g_n(X, \vec{b}) := \prod_{i=1}^n \left(\sum_{j=1}^n b_j x_{ij} \right) \cdot \prod_{i=1}^n (2b_i - 1)$$

Claim: $\sum_{\vec{b} \in \{0,1\}^n} g_n(X, \vec{b}) = \text{perm}(X)$. [Ryser's formula]

Pf: Rewrite LHS as $\sum_{T \subseteq [n]} \prod_{i=1}^n \left(\sum_{j \in T} x_{ij} \right) \cdot (-1)^{n-|T|}$

- It is supported on monomials like $x_{1,i_1} \dots x_{n,i_n} = m$
- Here, say i_j repeats $r_j > 1$ times.
- The monomial m can be associated to 2^{n-1} many subsets T : with $\text{sign} = \sum_{S \subseteq [n-1]} (-1)^{|S|}$
- For $\pi \in \text{Sym}(n)$, $\prod_{i=1}^n x_{i, \pi(i)}$ survives in LHS with $\text{sign} = 1$. \square

That permanent is in VNP. Right, so look at the definition again. So this is one monomial. Now is this monomial function computable in VP? It is obviously computable in VP right because this is a single multiplication gate. So this is in VP, but so you can use this potentially as a verifier the definition in the definition you had a g , so you can use this as a verifier.

But the problem is that as you look at different summands for different permutations from the permutation, your verifier has to compute this, this monomial, right which is

not clear how to do. So if you want to put this inside this g , the g that you had in the definition and g is a polynomial, g has to be a VP polynomial. So can you think of a g that can compute this? Can you connect these two?

So that is not clear because \bar{w} in the definition of VNP is just a string. So you will have to encode your permutation π as a string, but then from that string just using polynomial arithmetic, it is not clear how to get to this $x_{i,\pi(i)}$.

[Student] We can look g as some combination of extra monomials, if they cancel out somehow.

[Professor] Right. So well so first of all the first observation is that it is not direct.

It does not follow from the definition and second is well, so maybe we should look at a more complicated g instead of just this monomial we have to think of something more complicated and that is what we will do in the proof, okay. So let us use the following. So let g be the function, or be the polynomial that takes this n by n matrix, symbolic matrix and a vector and computes a product.

$$g_{n^2+n}(X, \bar{b}) = \left(\prod_{i=1}^n \sum_{j=1}^n b_j x_{ij} \right) \prod_{i=1}^n (2b_i - 1)$$

So the product will actually be of linear polynomials, okay. So how many variables does g have? g has $n^2 + n$ variables, right? This is the arity of the polynomial g . It is clearly a polynomial. Well if you look at these linear polynomials it is basically doing a inner product of vector b with the i^{th} row, right? So you for a given b what it is doing it is just taking in a product with every row. So there are n inner products and it is taking a product of those inner products.

So that is what g is. Well I will also need for technical reasons this constant product. So this is the sign part. This is just giving you so $2b_i - 1$ is either -1 or $+1$, right. So this is just a sign of, of what? Depending on how many ones there are in the \bar{b} , the weight of \bar{b} this will be a sign.

So this sign times this is the, the other is the main part inner products, right. And now, so this is our g and on top of this we will do a big sum on all the \bar{b} s. So what do you expect to get out of this, when you do the big sum? So the claim is that when you do a big sum, right, on all the vectors you will exactly get the permanent.

$$\sum_{\bar{b} \in \{0,1\}^n} g(X, \bar{b}) = \text{per}_n(X)$$

Okay, so this is also called Ryser's formula. So what is the proof of this?

Or do you already consider this obvious? Yeah, so the proof will be actually looking at these monomial products. So every, so this red part here, this part is producing these monomials in \bar{x} . And what you can show is that the monomials that are where I mean which are not coming from a permutation, those monomials cancel out because of the sign.

Well, so there might have been some linear algebra or some physics motivation. I do not know the exact details. But one rule of thumb here is if you look at this, this formula that the claim identity in the claim if you go modulo 2, right so modulo 2 determinant and permanent are the same, right. This is the only place where you can compute permanent because permanent is the same as determinant and the determinant even compute.

So modulo 2 there are the no signs and then the sum, big sum is giving you determinant and maybe that is another reason to look at the sum. It is able to cancel out all these monomials which are which do not correspond to permutations, okay. So once you do that then you might try to generalize it to other characteristics. It will be Det_n . Modulo 2 permanent n is the same as Det_n .

Okay, so let us quickly go through this proof. So we can rewrite left hand side as right so since \bar{b} is a 01 vector, let us go in terms of subsets okay. So let us write it as subsets of 1 to n and the product is over $x_{i,j}$, j in t and i is in 1 to n and with a sign. So the sign you can see is -1 to the n minus the size of the subset.

$$\sum_{T \subseteq [n]} \prod_{i=1}^n \left(\sum_{j \in T} x_{ij} \right) \cdot (-1)^{n-|T|}$$

Is this believable? Right, this is just translating inner product of a 01 vector with a row, with the i th row.

So when you do the inner product, you are just selecting some elements of the row and adding them. So that is this main part $\sum_{j \in T} x_{ij}$. And then you are going over all the rows. So that is the product, i equal to 1 to n and with a sign and then you are taking the big sum over all the subsets right. So now in this, now since this is straddling over all the n rows, so it is basically multiplying the variables that appear in distinct rows. Right?

So it is supported on monomials like $x_{1,i_1} \cdots x_{n,i_n}$. So from the first row you pick i_1^{th} variable, second row i_2^{th} variable dot dot n th row i_n^{th} variable and then you multiply them, right. That is, these are the kinds of monomials this product will produce. So here say this i_1 repeats r times. So the point is we want to analyze the situation when this monomial does not correspond to a permutation which means that i_1 to i_n they are not distinct.

There is some overlap happening. So suppose i_1 is repeating two or more times. Right, so when i_1 repeats, we want to show that this particular monomial will get ultimately cancelled in the big sum. Okay, so why is that? So in how many t 's will this monomial be produced? Well 2^{r-1} , right. So i_1 since it is repeating r times the number of t 's which can contribute, which will contribute this is exactly 2^{r-1} , the number of subsets.

So the monomial, let us call it m . So the monomial m associates or can be associated to 2^{r-1} many subsets T , okay. So well so whenever this monomial is contributed there is a sign attached. So let us look at the sum of the signs. So with sign what is

that? Well, it is basically $\sum_{S \subseteq [r-1]} (-1)^{n-|S|} = \sum_{S \subseteq [r-1]} (-1)^{|S|}$ over all the subsets of r -

1 minus one to the n minus size of s or even simpler size of s. Is this clear? So why are you getting $r - 1$?

Well so $r - 1$ appears because you have to pick at least something, right. **“Professor - student conversation starts”** So I mean you have to, when you pick those you have to pick i_1 r time. So 1, i_1 . So the i_1 is varying, so you have to pick i_1 r times. **“Professor - student conversation ends”**. No so let us say x_{1,i_1} appears and then in the remaining there are $r - 1$ one i_1 ’s, okay.

“Professor - student conversation starts” But then at least three n minus I mean once you if you from like a 1 to n you want to create a subset that has r i_1 ’s. You have to, you have r i_1 ’s and then you have the rest free right. It is like $n - r$, 2^{n-r} **“Professor - student conversation ends”**. Well so this monomial in how many ways is it being produced? This is what you are looking at.

So the number of ways it is being produced is 2^{r-1} and the sign is exactly this. And what is the sum of this? What is this sum? Well, this is it is basically just the product of $1 - 1$ s. So this is 0. Okay. Basically you can factorize this sum as a product of $1 - 1$ s. Right, so monomials so the thing we assumed is that r is at least 2, okay.

If r is 1 then this argument will not give you anything. So the only monomials that survive are those where nothing is repeating. So i_1 to i_n is actually distinct. So those so at least the support is correct as we wanted. Next thing you can see is so what about those other monomials? So this finishes our proof. Well not really finishes, but you have to make another observation.

So for permutations product of $x_{i,\pi(i)}$ survives in LHS. And it survives with the sign here why is it +1? Well it is +1 because all these all the b_i ’s are 1, right. So it survives in LHS with the sign 1, okay. So the sum of these monomials is all that remains and that by definition is called permanent. Okay, so that finishes the proof, right. So this is called Ryser’s formula.

It is important because if you try to compute permanent at some matrix, how much time will you will it take $n \times n$ matrix. More than n factorial, right. So something like n to the n . On the other hand if you use Ryser's formula, so in Ryser's formula you are just multiplying simple things just linear combinations is what you are multiplying. And how many times? 2^n .

So this will give you a 2^n algorithm, right? So in practical applications where you care for this difference you would use this because this is 2^n . The other thing the original thing is n^n , okay which for some for large n could be significant, significant difference. So this is the best algorithm known to compute permanent in generality.

Okay, so for general permanent, this is the best algorithm and it is quite non-trivial because in the definition you had far more monomials. So somehow that has been compressed. Now the question that remains, which is open is can you do better? Well, anyways, so we are digressing now. So this our goal was to prove that permanent is in VNP. So that also we have achieved because this g is now your verifier.

So once you have proved this claim, you should see g as the verifier circuit. Well, why is it in VP because it is just a simple product. You just compute these inner products by addition gates and then you multiply by a single multiplication gate. So it is clearly just a depth two arithmetic circuit. So that is your verifier and b bar is your witness. So this identity also shows that permanent is in VNP, okay.

We just need to compute this big sum over VP, right. And it also gives you an algorithm and you can see that here the constants are very small. So this always happens when we give a VP circuit or this kind of a VNP expression the constants are never too large okay. So although in the definition we are allowing arbitrary constants, in actual proofs they are always small.

So you can also use this as an algorithm. Okay, so I think I can stop. Any questions.
So next time we will look at determinants more carefully. Okay once VP, VNP are defined and we have this candidate polynomial we will prove more. Okay.