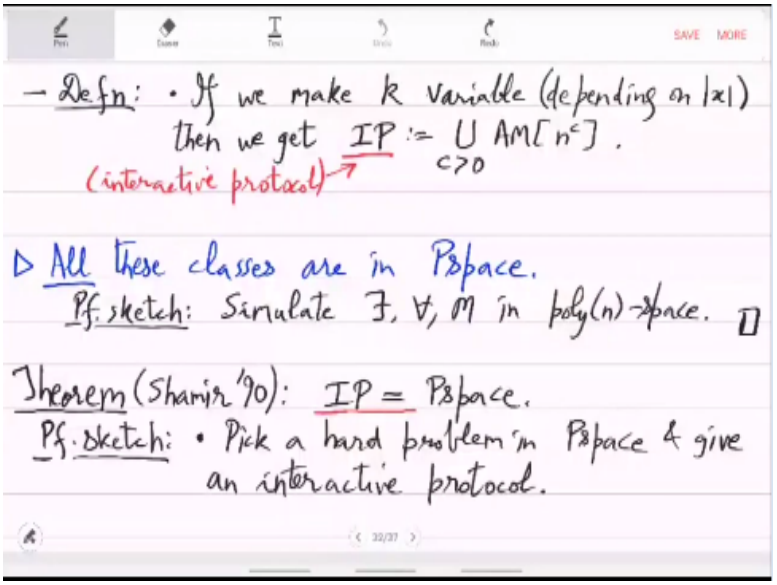
**Randomized Methods in Complexity**
**Prof. Nitin Saxena**
**Department of Computer Science & Engineering**
**Indian Institute of Technology - Kanpur**

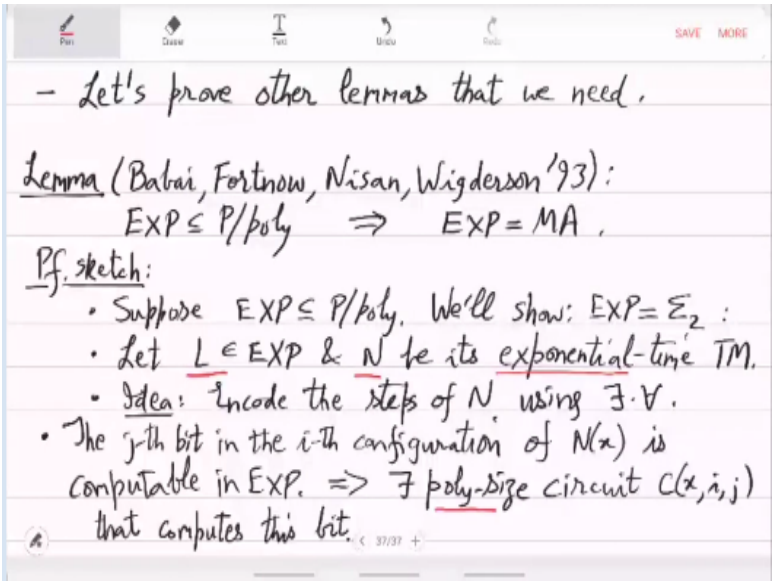
**Lecture - 05**
**ACC0 Lower Bounds**

**(Refer Slide Time: 00:13)**




Last time, we proved this theorem due to Shamir. We showed that IP = Pspace. Any problem that can be solved by Interactive Protocol, can also be solved in Pspace and any problem in Pspace has an Interactive Protocol. The hard part here was to actually show that quantified Boolean formula which is a complete problem for Pspace that has an interactive protocol. We showed that and then using that theorem we will show that if exponential time has circuits.

**(Refer Slide Time: 00:55)**



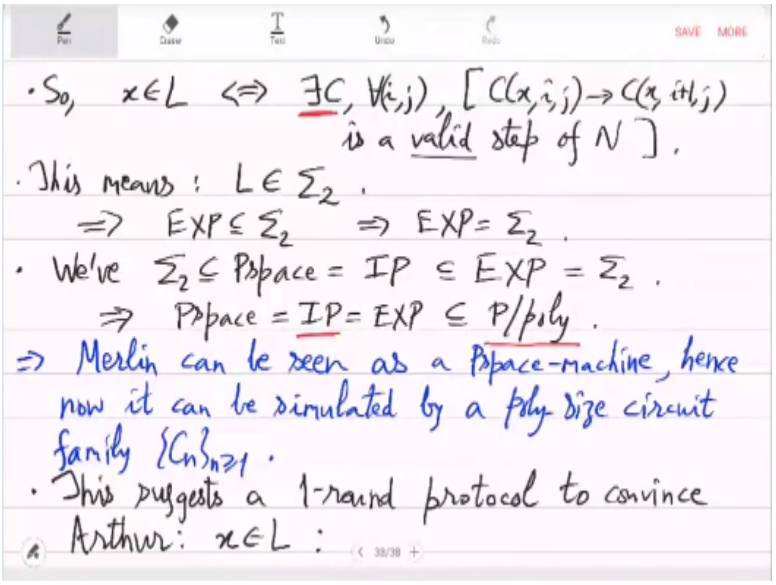
Then exponential time solvable problems also have a one round Merlin Arthur protocol. We do not believe any of these conditions, any of these statements but this lemma will actually give a connection. In other words you can also say the contrapositive, which is that if exponential time problems don't have Merlin Arthur protocol then they also don't have circuits.

We assume that exponential time is in P/poly. And we will show that immediate or a easy consequence of that would be that any exponential time solvable problem, say L with this Turing machine N. It can be the steps of Turing machine N can be encoded using ∃, ∀ quantifiers. Let us see that encoding - the j-th bit in the i-th configuration.

Remember that Turing machine starts it steps the computation then in any moment of time, it is in a configuration. So, configuration means where is the head? What is the state? And what is there on the tape? The j-th bit in the i-th configuration of N started with input x is computable in EXP. This just follows from the hypothesis that L is in EXP.

This implies that there exists a poly-size circuit C which will take x, i and j in binary that computes this bit. This is because of the hypothesis that x in P/poly. Hence once you have assumed that j bit can be computed in EXP then there is in fact a circuit that can also compute it and the circuit is only poly-size in x.

**(Refer Slide Time: 03:50)**




Which is the same as saying that if x is a yes string, then and only then there is a circuit, says that for all i, j, C(x,I,j) → C(x, i+1, j) is a valid step. We are saying that x is a yes string, if and only if there is this circuit C which is small sized such that for every i and j, C(x, i, j) computes the j-th bit in the i-th configuration.

Going from i-th configuration to i + 1 configuration, this is one step of the Turing machine. And the j-th bit, respectively of these 2 configurations is being verified. This verification whether the j-th bit of when you go from i-th configuration to i + 1 configuration whether this is correct. This can be checked simply by using the control of N. N is a Turing machine which has a finite control. So, one step is very easy to check.

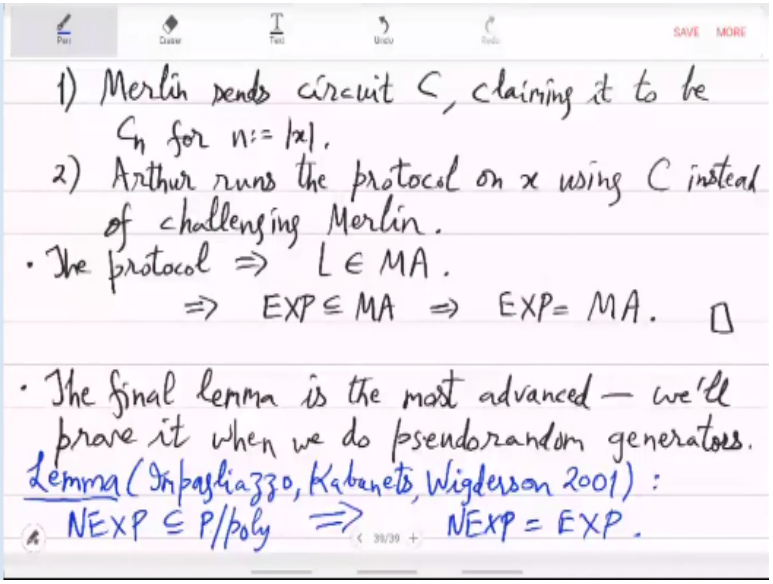
This thing in the square bracket, can be checked easily. And there is in fact a circuit C, which will be able to check the claim for every i and j. It can actually check the whole computation and the computation has exponentially many steps. This is the property of the circuit C that is how we write in ∃, ∀ quantification. This means that L is in $\Sigma_2$ and L was any language in EXP.

You have actually shown that EXP is in $\Sigma_2$ and $\Sigma_2$itself is trivially in EXP. So, this means that EXP = $\Sigma_2$. This is a very strong conclusion. It is a very strong statement that EXP is equal to $\Sigma_2$ . But, the hypothesis was also very strong that EXP is in P/poly, from that you are getting this. Let us continue the connections, we also know that $\Sigma_2$ is in Pspace, which has been shown to be IP, which is trivially in EXP because interactive protocol can be checked and simulated in EXP.

EXP we have shown to be equal to $\Sigma_2$. This means that all these containments are equalities. This means that Pspace is equal to IP is equal to EXP, EXP was given to have P/poly circuits. In particular, you have reduced that Pspace or more interestingly IP. What you have reduced from here? This is that IP, it has poly size circuits. Now the way, we will use this fact or this conclusion is that in any interactive protocol Merlin now can be eliminated and replaced by a circuit.

Arthur can just talk to a circuit in the protocol instead of this advisor Merlin. This is what we learn from here. Merlin can be seen as a as a Pspace machine. Hence now, it can be simulated by a poly-size circuit family. Let us call it $C_n$. This $C_n$ is basically Merlin. You replace Merlin by this circuit $C_n$, where n is basically the input size x. For that particular input size, Merlin can be replaced or simulated by the circuit.

And Arthur will now just invoke this as an oracle. This suggests a 1-round protocol to convince Arthur that x is in L . Merlin will just send the circuit $C_n$ to Arthur. And Arthur can now play with the circuit. Arthur does not need Merlin. That is the 1-round as a simple trick.
**(Refer Slide Time: 10:19)**



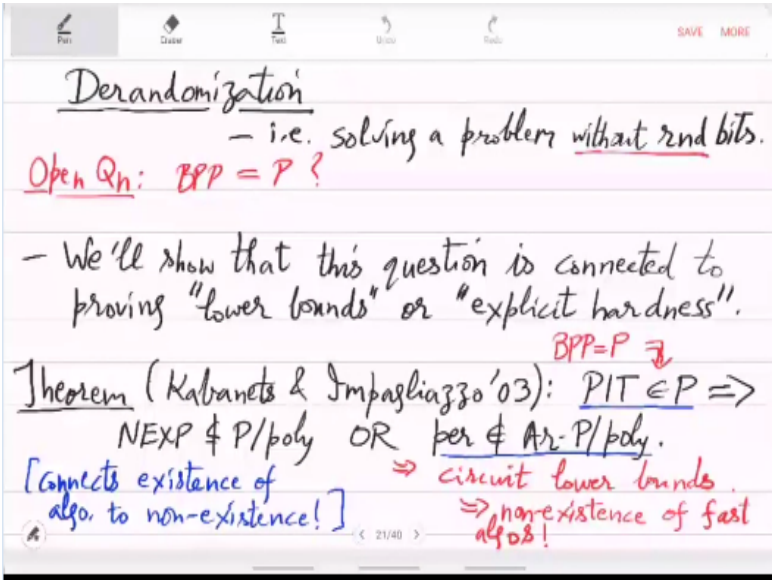

Merlin sends circuit C claiming it to be $C_n$ for n size of x. And Arthur can run the protocol using C instead of challenging Merlin. What has happened here is that Merlin which in the definition of IP or MA or AM, Merlin is all powerful. Merlin just gives answers, whenever challenged by Arthur and then it is up to Arthur to verify. But here, because of the strong hypotheses, we have been able to show that actually Merlin is only as good as a poly-sized circuit.

Now in the protocol, this circuit is presented to Arthur and then Arthur can work on his own, which means that L is in MA. This protocol implies that L is actually in MA and L was any language in EXP, so, actually EXP is in MA. MA is obviously in EXP, so, this means that EXP = MA. That finishes the lemma. That if you assume exponential time problems to have small circuits, then EXP has a single round protocol.

Let us now move to the final lemma. The final lemma that we will present. We will see the proof when we start or when we do Pseudorandom generators. The lemma was first shown by Impasliazzo, Kabanets and Wigderson. Here the hypothesis is even stronger than before, which is NEXP in P/poly, assume that then what can you deduce. Nobody believes that NEXP can be solved in or using poly-sized circuits, because exponential time problems are already hard enough.

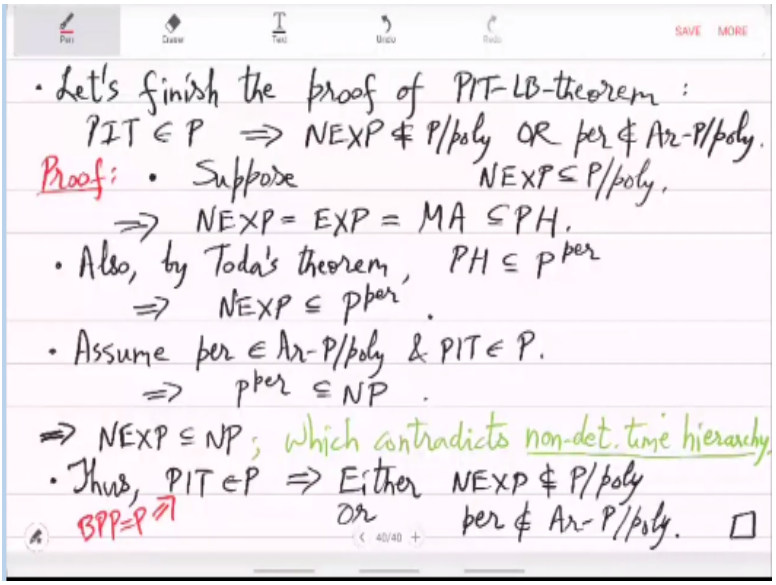
NEXP could solve another level of exponential hardness. We do not expect these doubly exponential time problems to have such small circuits. But if you assume this, then what you get is NEXP equal to EXP. This non-determinism can be eliminated at the exponential level, NEXP = EXP. That is the conclusion if you assume this absurd containment. We will prove this towards the end of the course, for now you just remember this lemma.

And let us now finish, the proof of our PIT lower bound connection which we stated long time back. So, we stated it here, this Kabanets Impagliazzo theorem.
**(Refer Slide Time: 15:20)**




That if BPP = P then either NEXP does not have circuits or permanent does not have circuits. Former circuits are Boolean, later circuits are arithmetic circuits. Let us finish this theorem.

**(Refer Slide Time: 15:37)**




Let us finish the proof of PIT lower bound theorem, which is that if PIT, polynomial identity testing, if it is in P or if BPP = P. Then, either NEXP is not in P/poly or permanent doesn't have poly-sized arithmetic circuits, this arithmetic P/poly. What is the proof? Now the things that we have developed till now, the tools and the lemmas the complexity classes using them the proof is actually very easy.

We will just assume, all these 3 things PIT in P, NEXP in P/poly, permanent in arithmetic P/poly, and we will deduce a simple contradiction. Suppose PIT is in P and at the same time NEXP is in P/poly, then you get from the previous lemma, that NEXP and EXP both are equal to MA. This is the lemma, which we have not proved, we just stated. Remember that MA is in the polynomial hierarchy. And then there is a theorem called Toda's theorem, which tells us that polynomial hierarchy is in permanent.

Using permanent as an oracle you can solve anything in polynomial hierarchy. This is called Toda's theorem. This is a class between PH and Pspace. So, this NP, $NP^{NP}$, NP to the NP to the NP to the NP all these classes, they actually can be computed by just permanent using it as an oracle. All this implies that NEXP is in $P^{per}$. Let us remember this that NEXP is in $P^{per}$.

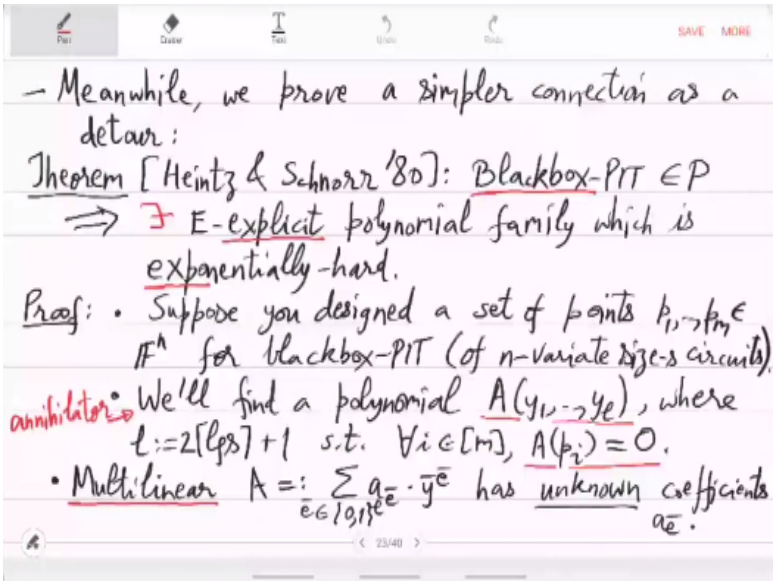
Now assume that permanent is in arithmetic P/poly and PIT is in P. If you assume these two things, then there was a lemma, which we, I think we proved it first. The first lemma which tells you that $P^{per}$ has NP proofs, so permanent can be verified efficiently. The idea was just that you will guess the small arithmetic circuit for permanent and then use PIT after expanding permanent row by row recursively.

That told us that $P^{per}$ is in NP. Which together with NEXP in $P^{per}$ implies that NEXP is in NP as well, which contradicts the non-deterministic time hierarchy. So, non-deterministic time hierarchy theorem says that if you have a complexity class where the non-determinism is function f. And then there is another non-deterministic time complexity class using time with function say $f^2$.

Then you get a strictly bigger class. So, more time means you can solve more problems. That is contradicted by NEXP = NP. This contradiction means that one of the three assumptions is false. Thus, PIT in P will imply either NEXP not in P/poly or permanent not in arithmetic P/poly. This finishes our big theorem which required defining all these new complexity classes by different means and proving connections between these impossible statements.

What we have learnt in the end is that if you can de-randomize which means that if BPP = P, then there are lower bounds. So, either NEXP will not have circuits or permanent will not have circuits. This is a very interesting connection and remember that we had also shown earlier.

**(Refer Slide Time: 23:04)**

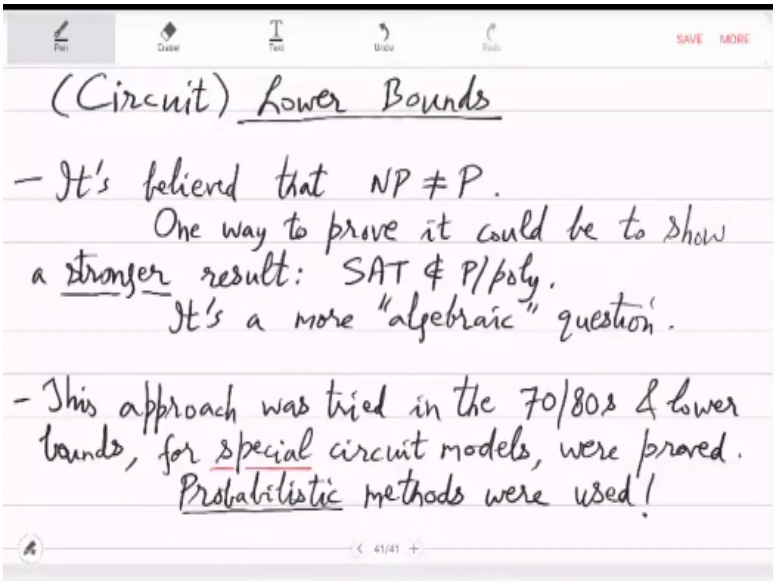



We had shown using a simpler proof that if you assume black-box PIT in P, then you actually get explicit polynomials which are exponentially hard for arithmetic circuits. Both these theorems are excellent examples of connection between existence of an algorithm versus

non-existence of circuits and hence non-existence of algorithms. In the rest of the course, we will discuss both these high level concepts.

One is this de-randomization which means that you design pseudorandom generators, that randomization can be eliminated. What are the possibilities, constructions, objects, related to this? On the other hand, we will also see, what we can do? or how can we prove lower bounds, circuit lower bounds and mainly Boolean circuit lower bonds.

**(Refer Slide Time: 24:19)**




Let us start with proving lower bounds for the circuit model which is actually far stronger than proving it over Turing machines. Because, if there is a fast Turing machine, then there are also small circuits. If you show that there are no small circuits, then you are also showing that there are no fast Turing machines. Obviously, the prime motivation for this is, it is believed that NP ≠P, which is saying that problems whose solution verification is easy They may not be easy to solve.

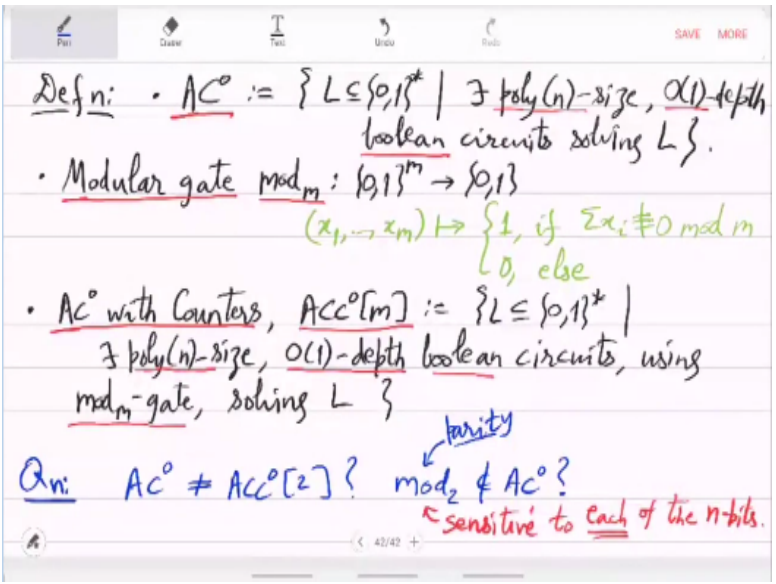
Given a solution you can verify. But, that does not mean that you can actually find the solution. Philosophically it is a self-evident fact but mathematically how do you show it? So that is an open question. One way to show it is to show something stronger, i.e. to show a stronger result that SAT is not in P/poly.

The first conjecture is SAT is not in P, but that is implied by SAT is not in P/poly. SAT does not have poly-size circuits and this seems to be a better formulation because, it is some more algebraic question, which we can study by developing or using mathematical tools. It is still a statement about Boolean functions, but the circuit has AND or NOT gates and maybe you can simulate or you can simulate it algebraically.

And then maybe you can use algebraic tools as well to prove this. This approach was tried in the 70s and 80s and lower bounds for special circuit models were obtained for special circuits. However no general circuit lower bound results is known, but for special interesting circuit models this approach was implemented. And this is where the probabilistic method part in the course title will enter.

Probabilistic methods were used to do this. We will now start working on this path and prove some very interesting lower bound results using very interesting algebraic and probabilistic tools.

**(Refer Slide Time: 28:53)**




To exhibit the implementation of these ideas, the complexity class that we will use is called $AC^0$. We want to look at restricted forms of Boolean circuits and the first thing we restrict is depth. Let us restrict the depth to a constant. Say we are looking at Boolean circuits, where the depth is 100, 200, 1 million or 1 billion, any constant. Which means that it is not allowed to grow with the input size n.

With respect to the number of variables n, depth is constant. AC0 is the set of those Boolean problems. It means languages L such that there exists poly(n)-size, constant depth Boolean circuits solving L. The key thing is poly(n)-size, constant depth and Boolean circuit. Boolean circuits have gates, AND or NOT.

Our methods actually will require one more gate called the modular gate. So, $mod_m$ is a binary function which takes m bits and gives 1 bit. It will just check whether the sum of the m input bits is divisible by m or not. So, $x_1$ to $x_m$ are the bits and the answer will be 1, if the sum of $x_i$'s is not 0 mod m and 0 otherwise. That is the modular gate.

In $AC^0$ you can also introduce these $mod_m$ gates where m is a constant. So m will not depend on the input size, as it is a constant. You can think of mod-2, mod-3, mod-5 and so on. When you introduce these gates, then the circuit is called $ACC^0$. That is $AC^0$ with counters; this is now the collection of those Boolean languages L.

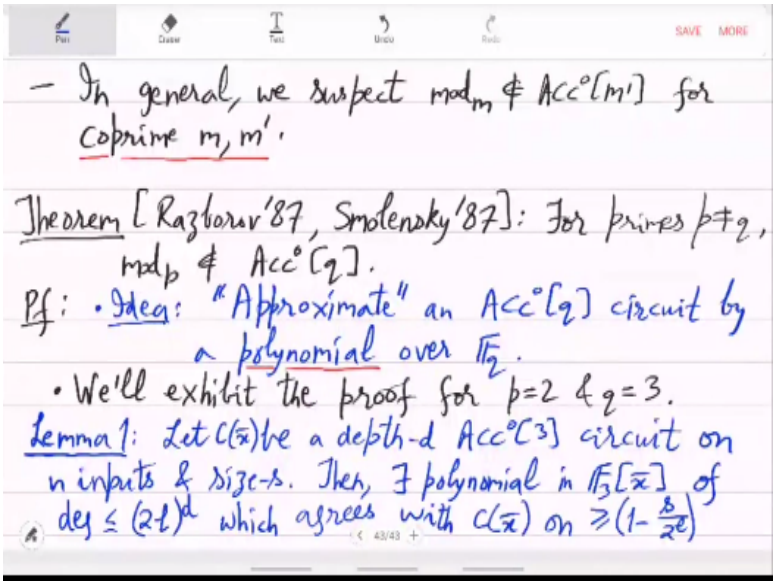
Languages L such that there exist poly n-size, constant depth, Boolean circuits using mod-m gate, solving L. So, $ACC^0$ is $AC^0$ with the counters, but in particular it counts the values $mod_m$, size should be polynomial, depth should be constant and circuit is Boolean. So, everywhere only 0, 1 is used. The question you should ask at this point is, what about $AC^0$ and $ACC^0[2]$? Which is equivalent to asking, is $mod_2$ in $AC^0$?

So, in constant depth can you compute parity? Note that $mod_2$ is parity, meaning given m bits whether the number of 1s is odd or even. When it is odd, answer is 1, when it is even the answer is 0. Could that value be computed? Could that parity bit be computed in constant depth? It is not immediately clear, but if you think about it then it seems that you have to computing the sum and also division $mod_2$.

And you have to do it, kind of n times or you have to do it in a way which is dependent on n. Because, the function of parity is very sensitive on each and every bit. It changes by flipping even a single bit. So, it is sensitive to each of the n bits. We will think of inputs as $x_1$ to $x_n$. You have to look at each and every bit that may be hard to do in just constant depth and poly size.

We actually believe it to be not there and hence, we believe $AC^0$ to be different from $ACC^0[2]$. This is what we will prove next. This is an amazing statement with an amazing proof technique.

**(Refer Slide Time: 36:42)**




In general, we suspect that $mod_m$ should not be in $ACC^0[m']$ for co-prime m and m'. In particular, we suspect that $mod_2$ should not be in $AC^0$, $ACC^0[1]$. The question is whether $ACC^0[1]$ and $ACC^0[2]$ are different? And we would assume, we will we would suspect it to be different, whenever this the two mod are different and to be precise co-prime. So, that was shown by Razborov and Smolensky. So, for primes p and q different, mod-p is not in $ACC^0$ $mod_q$.

This is what we will prove and the proof technique will be very interesting. Let us first mention the idea. So, "approximate" an $ACC^0[q]$ circuit by a polynomial over $F_q$. Following the hunch that circuits are more algebraic, we will follow that and formalize it by replacing this circuit $ACC^0[q]$ circuit by a multivariate polynomial $mod_q$, where q is a prime.

We are actually then working over this finite field $F_q$ and this polynomial will be approximating. It will not exactly compute the circuit, but it will approximate it well. You can read it as $ACC^0[q]$ circuits are easy to approximate by a polynomial. On the other hand, we

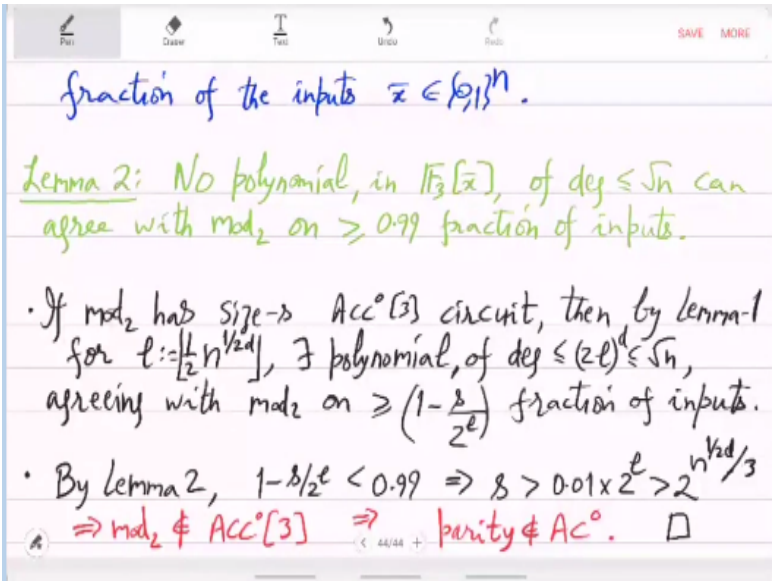
will show that $mod_p$ does not have that property. So, $mod_p$ function cannot be approximated easily over $F_q$, which will show that these two are different.

Instead of doing this in the most general form, we will exhibit the proof for p = 2 and q =3. And I leave the general proof for the assignment. This will be a long proof. Let us break it into lemmas. The first is the approximation part. Let C be a depth-d, $ACC^0[3]$ circuit on n inputs and size-s. Then, there is a multivariate polynomial in $F_3$.

So, in the polynomial ring $F_3[\bar{x}]$, $F_3$ is the finite field with 3 elements. There is a polynomial whose degree is not too high - $2l^d$, where l is some parameter which we will fix afterwards. Think of this as a low degree. We are saying that there is a polynomial in $F_3[\bar{x}]$, which is low degree, and agrees with the circuit. The circuit has variables, $\bar{x} = x_1, \ldots, x_n$, the same variables there are in the polynomial also.

And this polynomial has low degree and it will agree with C($\bar{x}$) with very well. It will agree on $1 - s/2^l$.

**(Refer Slide Time: 43:19)**




The input space is $\{0,1\}^n$ and out of this a good fraction – $1 - s/2^l$, which we can make small. This is a big fraction. For example, the polynomial may agree with C on 99% of input space. It is not 100%, but it can be made pretty large, almost 100%. That is a surprising fact. Why is this happening? How can you convert a circuit?

Boolean circuit into a polynomial, which approximates this well and is of low degree. So, that is lemma 1 that we will show. The other lemma that we will show, is the opposite for $mod_2$. That no polynomial in the same polynomial ring of degree smaller than square root n can agree with $mod_2$ on 99% of the input. These two lemmas will together imply that $mod_2$ cannot be computed by $ACC^0$ in $ACC^0[3]$. Why is that?

Now, if $mod_2$ has size s $ACC^0[3]$ circuit then by lemma 1, for a certain l we will get a polynomial and that polynomial will have low degree. Suppose $mod_2$ has size s $ACC^0[3]$

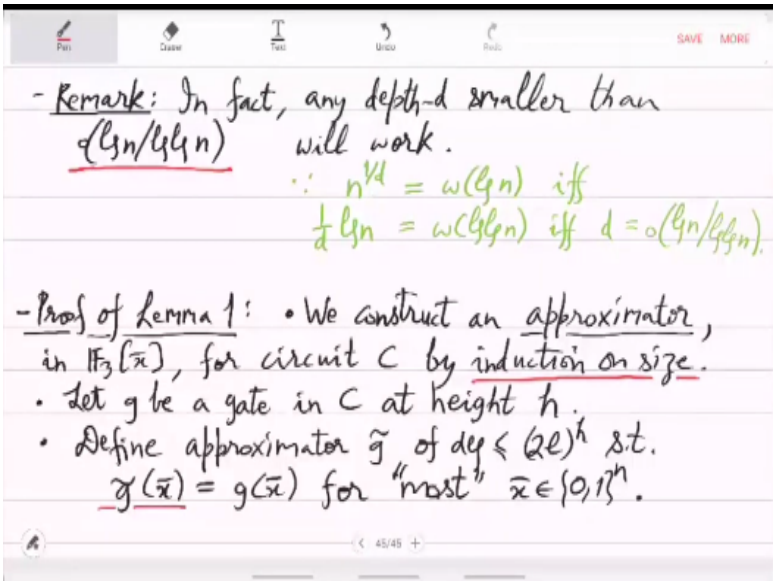
circuit then by lemma the degree is roughly $l^d$ and in lemma 2 the degree is $\sqrt{n}$. You basically want to match these two. Which means l should be around $n^{l/2d}$.

By lemma 1 for $l = 1/2 \cdot n^{1/2d}$. There exists a polynomial of degree less than equal to $2l^d$ which is at most $\sqrt{n}$. And whenever we use these expressions, we actually mean the closest integer. So $l = \lfloor 1/2 \cdot n^{1/2d} \rfloor$. And this polynomial agrees with $mod_2$ on $1 - s/2^l$ fraction of the inputs.

Now, lemma 2 says that whenever this happens, the fraction has to be small. So, $1 - s/2^l \leq 0.99$ which means that s has to be larger than this. And how big is l? l is around $n^{1/2d}$. This is actually bigger than $2^{1/3 \cdot n^{1/2d}}$. So, you can pick n to be sufficiently large.

And then the lower bound you are getting, because of lemma 1 and lemma 2 together is, s is actually more than $2^{n^{\epsilon}}$ where $\epsilon$ is absolute constant $1/2d$. This is not polynomial size, it is exponential size. You deduce that $mod_2$ cannot be in $ACC^0[3]$ which also means that parity cannot be in $AC^0$, because $AC^0$ is even weaker than $ACC^0[3]$. So, $mod_2$ or parity cannot be in $AC^0$. This will finish our claimed theorem by Razborov and Smolensky.

**(Refer Slide Time: 50:49)**




And you can see that in this proof, you can take d to be non-constant as well. So, it is slightly more than $AC^0$. Any depth d smaller than $\log log n / (2 + \epsilon) \log log \log log n$ will work, where epsilon is a constant close to 0. This is because this $n^{1/d}$ you want this to be sufficiently bigger than log n. In that case, your s will become super polynomial.

So, we want $n^{1/d}$ to be more than $\omega(\log log n)$ as it should be asymptotically larger than log n. So, $1/d \cdot \log log n$ should be equal to $\omega(\log log \log log n)$ then the d has to be smaller than $\log log n / \log log \log log n$. That is the correct thing to say. So, let me just correct this, so if depth of Boolean circuit is smaller than $\log log n / \log log \log log n$, then the previous lower bound that you got is super polynomial.

Even circuits of that depth will not be able to compute parity. Now, all that remains is to prove lemma 1 and lemma 2. So, lemma 1 is the approximator lemma and lemma 2 is the parity hardness or inapproximability lemma. Let us prove the approximator one first.

We construct an approximator in the polynomial ring $F_3[\bar{x}]$, for the circuit C inductively. The proof will be by induction on size. Which means that we will first design approximator for AND gate, approximator for NOT gate, approximator for OR gate and then build this one by one, gate by gate. Let g be a gate in circuit C, at height h and let us look at an approximator for g.

Define g of degree $2l^h$ such that $\widetilde{g}(\bar{x})$ equals $g(\bar{x})$ for most inputs. We have to design this approximator $\widetilde{g}$. Looking at the gate g at height h, we will design it. We will assume that we have an approximator for the inputs of g and using that we will design $\widetilde{g}$.