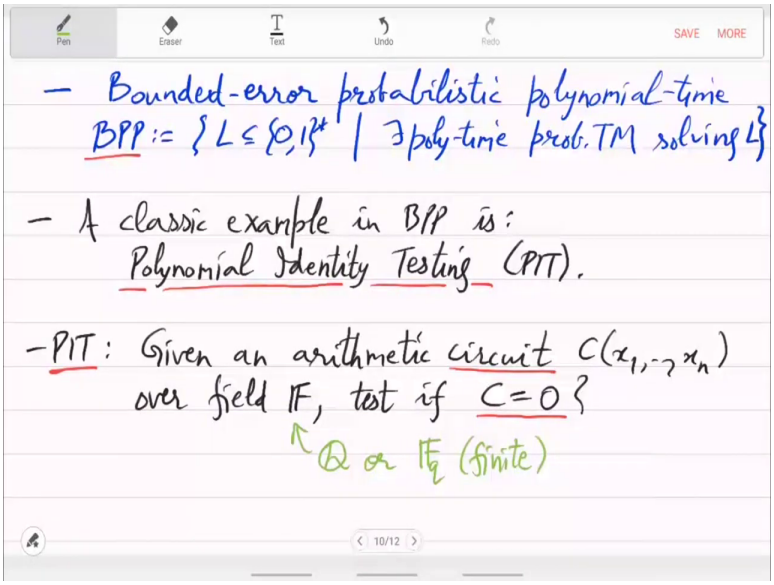
Randomized Methods in Complexity
Prof. Nitin Sexena
Department of Computer and Engineering
Indian Institute of Science – Kanpur

Lecture 02
Randomized Methods in Complexity

In the last class we did a crash course in formalizing problem. What is a problem in computer science? What is a solution? What is an algorithm? Then which algorithms are bad which algorithms are good, which problems are hard, which problems are practical. Then we define many complexity classes. Complexity classes are basically collections of problems.

(Refer Slide Time: 00:40)

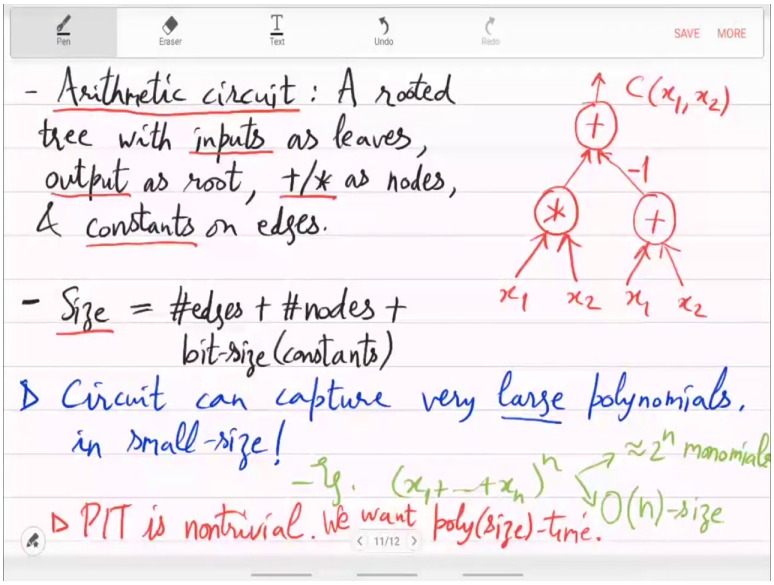


That share similar type of resources for example polynomial time complexity class contains problems collects problems which are all computable in polynomial time. Then we introduced non determinism so the class NP. We introduced space so class P space and finally we introduced probability. So the ability to toss coins and that defined the complexity class BPP. So a prominent problem in BPP is the problem of polynomial identity testing.

Given a circuit whether it is 0 this is what you want to test. So here the field of interest the field here will be you can think of it as the field of rationals or you can think of it as a finite field. These are the 2 prominent examples of fields that we are interested in this course. Other fields

are not very interesting for us because they are the even the presentation of constants is not clear. So we defined arithmetic circuit and thus what is the size of it.

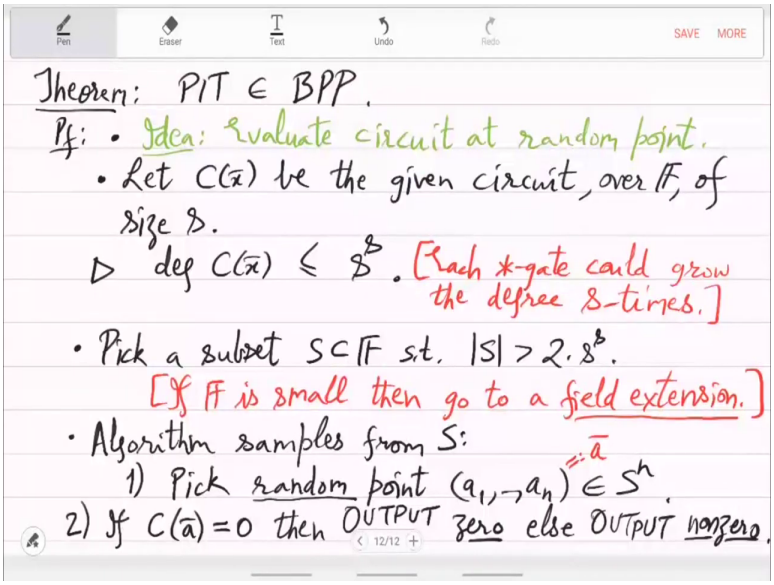
(Refer Slide Time: 02:12)



And circuit is a way to actually write down very complicated polynomials in a small way it is a small presentation it is a compact representation of difficult polynomials. So which is why the problem of PIT (polynomial identity testing) is non-trivial it is a non-trivial problem because of the compact representation. So you have to first somehow you have to go I mean it is asking you to actually go over all the monomials without the monomials being given.

And if you try to find them then it will be very expensive. So the first question is can we show some kind of an upper bound for this problem is so you ideally you will ask the question whether there is a practical algorithm.

(Refer Slide Time: 03:00)



And you will be surprised to know that there is; so what we will show today is that this problem of PIT ϵ BPP .So what is BPP? BPP is this class of decision problems L such that there is a polynomial time probabilistic turing machine solving it probabilistic algorithm is there. So the algorithm will make choices based on coin tosses in the end it will give an answer now the answer being wrong that probability should be very small.

So we need to design such an algorithm for PIT. So the idea here is that evaluate your circuit at random points. So the idea is just evaluation at random places. So it is a simple idea obviously if the evaluation comes out to be non-zero then the circuit is non-zero polynomial is non-zero. If the evaluation comes out to be 0 then you do not get any information because it is either the circuit was indeed 0 or the circuit was not 0 but your random point is bad the point you picked is bad.

So, it is actually a root. So in that case how do you show that the error probability is small. So that will need a lemma which is very useful in computer science. So let us do all that so let circuit C($\bar{x}$)be given over field $\mathbb{F}$ of size s. So it is a size s circuit. Now from the size what can you say about the degree of this polynomial how big can the degree be? So remember that in a circuit you can do repeated squaring here you can have s multiplication gates and then you can feed the output of the previous one to the input of the next one.

So that way you will be squaring so x becomes x^2 , x^2 becomes x^4 and x^4 becomes x^8 and so on. So that can give you degrees^s. So deg C($\bar{x}$)$\leq s^s$ each time you are multiplying s things and you can do this s times. So you get s times s times s dot dot s times. So each multiplication gate could grow the degree s times that is the reason. So you know an upper bound it is actually a pretty large upper bound it is exponentially large.

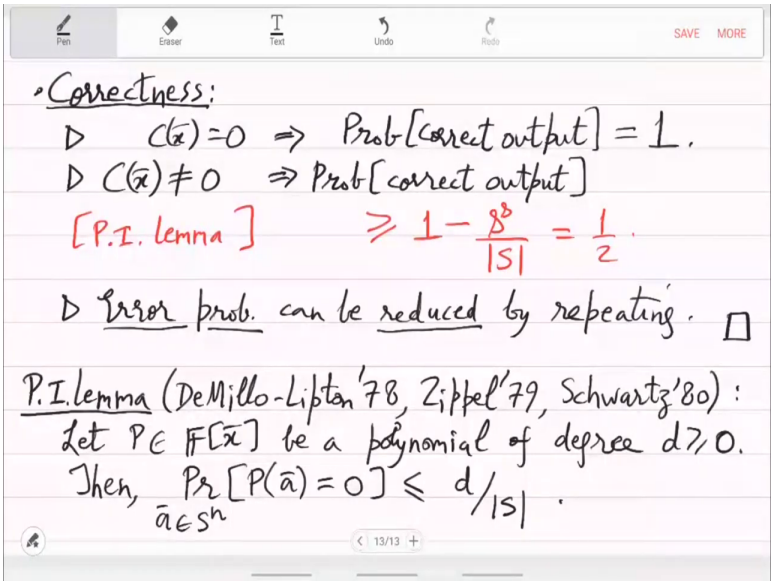
So if C was a univariate polynomial there could be these many rootss^sroots there cannot be more than serious roots but there can be s^sroots. So you have to actually come up with an argument of how can you avoid these possible roots. So for that you pick a large enough subset. So pick a subset $S \subset \mathbb{F}\, s.t\, |S| > 2.s^s$ such that it is large it is bigger than 2 times the degree. And in case field is small then you will not have such a subset.

So if $\mathbb{F}$ is small then go to an extension. So fields have this beautiful property that you can go to a bigger field contained in your field and as big as you want. So you just go to a bigger field which has size more than 2 times s^s and from there you can pick a suitable subset and then when you are sampling the point sample from s. So the algorithm samples from s. So the steps are actually very simple you pick a random point $a_1 \ldots\ldots a_n \epsilon S^n$

And just evaluate so if C at this point so let me call this point $\overline{a}$ .so if C($\bar{a}$) = 0 then you will output then you will output zero else output non zero. So this is the simple algorithm it just samples a random point and then evaluates the circuit at this point. Now note that evaluation can be done very efficiently because you are given addition multiplication gates so once you have fixed your point even to n are not very large because you pick them from this sample space s.

So you can easily compute by just doing addition multiplication in the field this is a very fast algorithm you can do it practically implement it practically. The only thing is what is the error?

(Refer Slide Time: 10:09)



So let us discuss that. So if the circuit is actually 0 then the probability over the random choice of output being correct, which is saying 0 outputting 0 that is that will always happen. If circuit is 0 it evaluates to 0. So you will say algorithm will say give the correct output. On the other hand if the circuit is non-zero then the probability of correct output which is non-zero that probability is so intuitively it should be one minus the error probability.

And when will your algorithm say 0 that will happen only when $\bar{a}$ is a root so you have to see how many roots are there in s^n space. So if it was a univariate polynomial then the number of roots could have been at most $\frac{s^s}{|S|} = \frac{1}{2}$. So this is called the thing in red this is true for univariate but why is it true for more variables when there is x_1, x_2, x_3 and so on.

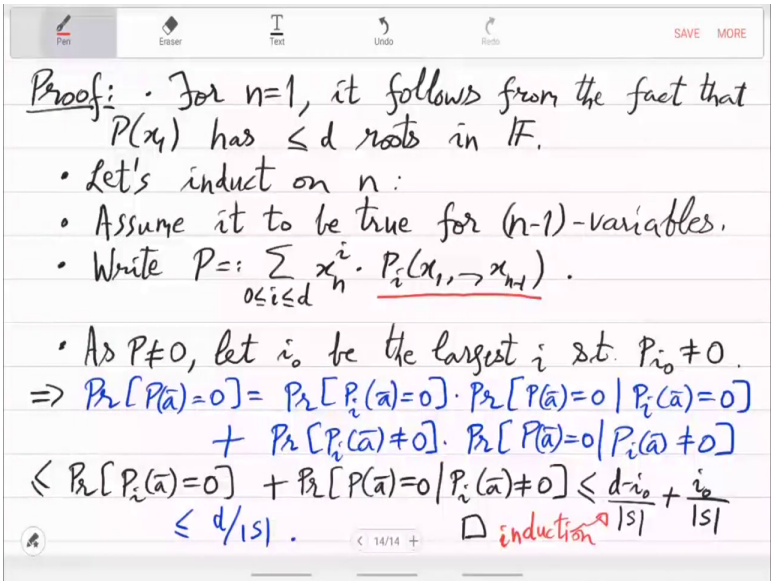
So that we have to prove separately, we will call it the polynomial identity lemma. So this follows from polynomial identity lemma which will prove next. But here at least the correctness now you have a sense of the correctness that for 0 circuit it is always correct for non-zero circuit it is correct 50% of the time. So that is a decent probability you repeat this algorithm again and again and the error probability if you repeat it twice then the error probability becomes one fourth and so on.

So error probability can be further reduced by repeating the algorithm. So this is a very good practical algorithm. So that at least finishes our proof that PIT is in BPP. Now let us prove the lemma the polynomial identity lemma. So this has been proved by many people let me give you the name Demillo Lipton, Zipple, Schwartz. So this Demillo Lipton Schwarzepaler lemma says that if you have a polynomial $P \in \mathbb{F}[\bar{x}]$ any number of variables be a polynomial of degree $d \geq 0$.

Which means that so degree 0 means that it is a constant non 0 constant, degree one means that there is this term x in it and so on. So in particular P is a non 0 polynomial then $Pr\,[P(\bar{a}) = 0] \leq d$ on a random point $\bar{a} \in S^n$ this probability is quite small. This probability is at most $\frac{d}{|S|}$. This is immediate you can see it immediately when the number of variables is 1 when P is a univariate polynomial.

Because univariate polynomial over a field can have at most d roots and now we have to generalize this fact for any number of variables.

(Refer Slide Time: 15:44)



So let us do that. So for n = 1 it follows from the fact that $P(x_1)$ has $\leq d$ roots in $\mathbb{F}$ or any extension there cannot be more than d roots because the degree is d or less than d. So n = 1 it is true. Now from n = 1 we will give basically an inductive proof induction on the number of variables. So let us induct on n so assume it to be true for(n- 1), variables.

So you write ,$P =: \sum_{0 \le i \le d} x_n^i . P_i(x_1 .... x_{n-1})$. So in terms of x_n the coefficients are these polynomials P_i in the remaining variables(n - 1)variables and one of these P_i is non-zero. So we can call that let us say we pick i_0. So as P $\neq$ 0let i_0 be the largest i such that $P_{i_0} \neq 0$everything after i_0 is not appearing in P and now we will argue using this and the induction hypothesis.

So $Pr\,[P(\bar{a}) = 0]$=Pr$[P_i(\bar{a})=0].Pr\,[P(\bar{a}) = 0 |\ P_i(\bar{a}) = 0]$+Pr$[P_i(\bar{a})\neq 0].Pr\,[P(\bar{a}) = 0 |\ P_i(\bar{a}) \neq 0]$

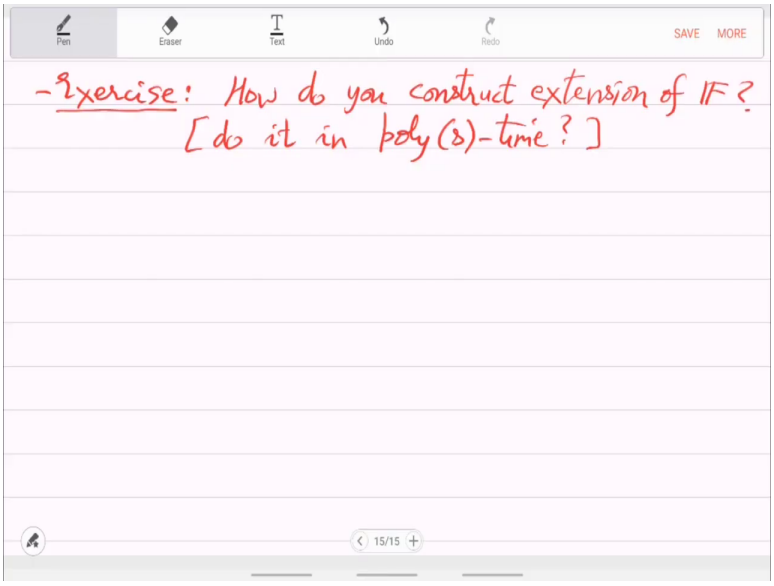
So we are basically dividing the event into exclusive events one event is P_i vanished other event is P_i did not vanish and then we are conditioning the event $p(\overline{a})$=0 with respect to these 2 exclusive events.

Now we can upper bound each of these. So the first summoned is definitely not more than probability that P_i vanished. And the second summoned is not more than the probability that P vanished at when P_i did not and now what are these probabilities. So what is the probability that P_i vanishes well. Now P_i is in one less variable so you can use induction hypothesis and its degree is what? Its degree can is at most $\frac{d-i_0}{|S|}$ that is the probability of it vanishing on a random $\bar{a}$

Now in the case when P_i did not vanish. So when you look at $a_1 ..... a_{n-1}$ what will be the degree of this P substituted P it will i_0 Now what is the chance that when you plug in a_n it vanishes? Well it is a univariate case, so you will get $\frac{i_0}{|S|} \le \frac{d}{|S|}$, which you wanted to prove. So that is the end of the proof. So this is where you used induction $d - i_0$ is the induction part on (n - 1)variables.

And the second part is just base case univariate. So this is a very nice lemma the proof is very simple but clever. So this P_i lemma tells you that in the sample space the roots are actually very few. So randomness helps so let me leave you with this question that how fast can you do this if an F is small how do you go to a field extension that algorithm also you should know this is an exercise for you.

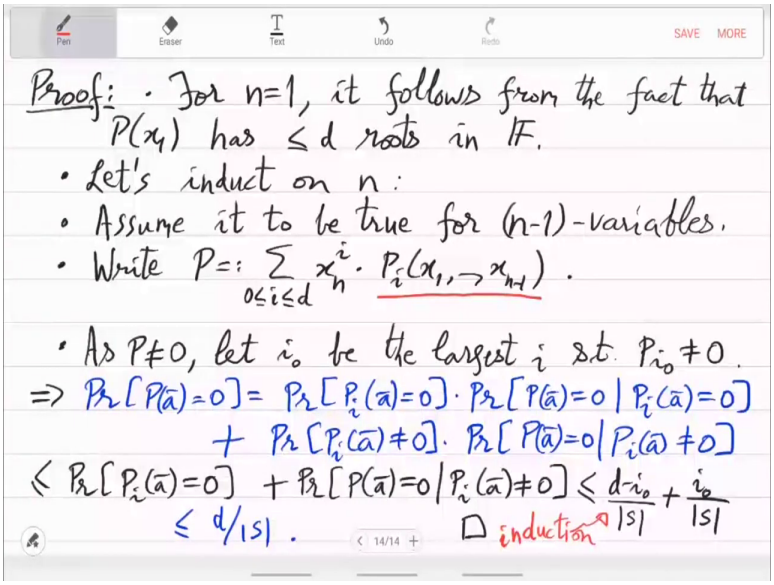
(Refer Slide Time: 23:13)



How do you construct field extension of $\mathbb{F}$, if $\mathbb{F}$ is small how you do construct field extensions can you do it fast. It can be done actually very fast it can be done in poly s time. So do it in poly (s) -time ok. So this will give you the full practical algorithm if either the field is large then you pick a big enough sample set or the field is small then you construct an extension and there you pick the sample set.

So this is the first randomized algorithm and the full analysis that you see. So it gives you an idea about circuits probabilistic algorithms and the analysis of the error probability. So let us develop this circuit model a bit more because we will be using circuits extensively in this course later.

(Refer Slide Time: 24:40)

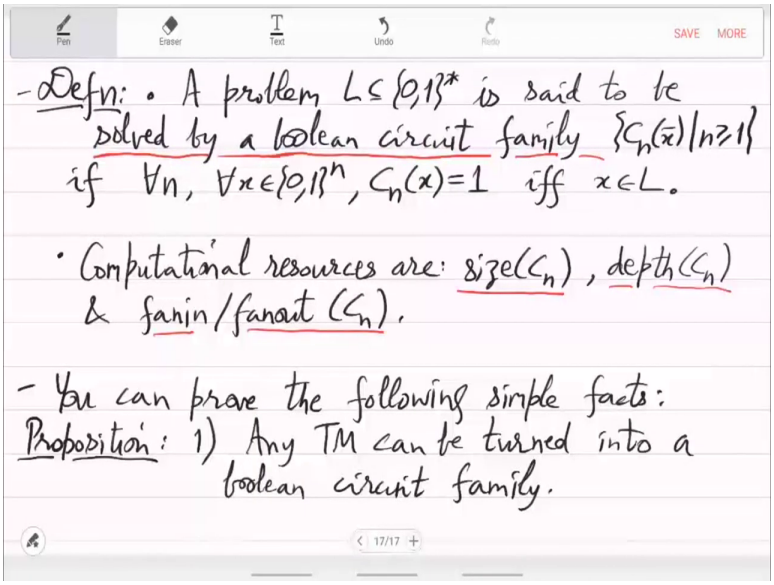


So we have already discussed the definition of arithmetic circuit. So arithmetic circuit over a field has addition multiplication gates and field elements and what is called a boolean circuit that has and or not gates has and or not gates and constant {0,1} or false and true. So that is an arithmetic circuit and a boolean circuit. So what does an arithmetic circuit compute well as you saw before it computes polynomials.

So this computes polynomials and what does a boolean circuit compute? Computes boolean formula. So this is computation in 2 different words one is in the world of polynomials which is algebra the other is in the word of boolean formulas which is a propositional formula in the word of logic proposition calculus. So these actually are computational you can see them as computational models for the respective words these are actually computational models.

The model computation even real computation so that is what we will do. Instead of turing machines just like you saw so many models in theory of computation course these are 2 more. And this course will majorly focus on these 2 models only. So once we have accepted this let us then build complexity classes new complexity classes based on this.

(Refer Slide Time: 28:30)



So before that we have to define what a problem is and then we look at a collection of problems. So a problem in the boolean circuit case is a language it is said to be solved by a boolean circuit. So in the case of turing machine we said that a turing machine exists which solves the problem here we have to talk about circuits boolean circuits and they will be one circuit for every input size n.

So it will actually be not one circuit but infinitely many circuits one for every n. So it is actually a boolean circuit family and we are defining solved what is the meaning of solution? By a circuit family$\{C_n(\bar{x})|n\geq 1\}$ if ∀n ,∀x $\epsilon\{0,1\}^n$ you get $C_n(x) = 1$ iff$x\epsilon L$. so if $x\epsilon L$ then the circuit of that appropriate input size n C.

So the circuit C_n it should result in 1 in the output. So that is the meaning of solving something by a boolean circuit. And the computational resources are in this model. So one is size of the circuit as a function of n since you have infinitely many C_n as a function of n how is this size growing so size similarly depth and fanin fan out of these circuits. These are the computational resources based on which you can now define a complexity class.

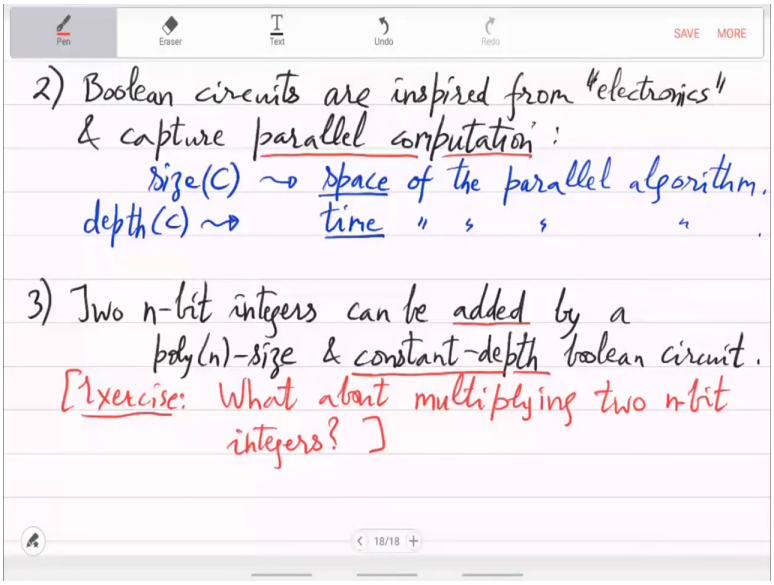
So we will define the complexity classes next but let me say something about the relationship between this boolean circuit computational model vis-a-vis turing machine model what is the

difference and what is the similarity? So you can prove the following simple properties do this as an exercise it is a good warm-up exercise especially if you have not done a complexity course before. So any turing machine can be turned into a boolean circuit family.

So, turing machine takes any length input and gives an output thus solving that decision problem. The steps of that the turing machine takes for let us say input $x_1 \ldots\ldots x_n$ every step is it is so simple that you can actually convert it into an expression using and or not so overall you actually get a circuit boolean circuit this you can do by just unfolding the definition of turing machine.

What about the converse? So can you convert a boolean circuit family into turing machine? This I leave as a question to you think about that. Hint is this cannot be done. The converse is not true but why is that? Think about the reason.

(Refer Slide Time: 34:12)



So we are calling it circuits so which means that this is somehow inspired from electronic circuits. So what is the comparison? So let us say something about that. So boolean circuits are inspired from electronics and the capture parallel computation. Why is that why parallel? Well we are using the keyword parallel because these gates which you have and or not gates they can run in parallel.

If you look at it as an electronic circuit then you have a and here and you have a or there so once their inputs are available they can simultaneously compute. So in one second lot of gates can simultaneously compute things. So this can be very fast. So this is actually the parallel computation model. So, size of the circuit. So size (C) actually is related to the space of the parallel algorithm.

And the depth of the circuit depth is related to the other resource time. This depth thing should be easy to imagine this is because so all the gates are processing in parallel but the one below it has to first compute something so that the one above can start the computation. So whatever is the depth that many seconds it will take it cannot be faster because the one above waits for the one below.

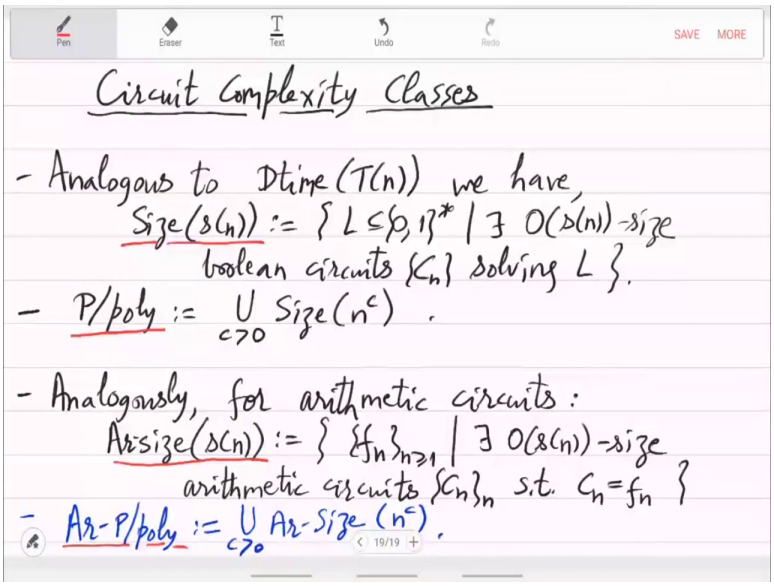
So depth basically corresponds to time parallel time and size corresponds to this space of the parallel algorithm because every gate is storing some information so that much space is being used up. Another simple observation is addition of two n-bit integers. n-bit integers can be added by a poly(n) size poly(n) size is not surprising but what will be surprising is the depth. So in how much depth can you do addition of the circuit.

So the surprising thing is that you can do it in constant depth. So, addition is possible in constant depth which is constant time in terms of parallel processing. So two numbers can be added in constant time but the price is that then you will need many processors many gates poly(n) gates. So again I can ask you as an exercise to prove this design this circuit and what about multiplication?

What about multiplying two n-bit integers? So these are some simple properties of boolean circuits once you have this definition. And similarly you can define computing a polynomial by a by an arithmetic circuit. So again a polynomial family because the numbers of variables are changing so when will you say that a polynomial family is computed by a arithmetic circuit family. When this C_n is equal to the polynomial f_n so polynomial families can be modelled by arithmetic circuit families.

And the resources are the same size depth fanin fan out. So the same definition you can do for arithmetic circuits and based on all this you can now define complexity classes.

(Refer Slide Time: 40:13)



So what is the complexity class that is analogous to polynomial time efficient computation what is the analogy here? So here remember there we said turing machine should take polynomial time. Here we could say that circuit family is polynomial size. So analogous $Dtime\ (T(n))$ we have,$Size\ (s(n)) := \{L \subseteq \{0,1\}^* \mid \exists\ O(s(n)) - size$. So these are actually the problems which are solvable in s(n) size boolean circuits $\{C_n\}$solving L.

So for boolean circuits this is the first complexity class you need for a function s size of s collects all those problems which are which require only s (n) size order s(n)size boolean circuits. And now you can define the analog of P here which is $P/poly := \bigcup_{c>0} Size(n^c)$ so all polynomial sizes allowed. So there you had P in turing machine here in boolean circuit you have P /poly which is all the problems which can be solved in boolean circuit size n to the C for some absolute constant C.

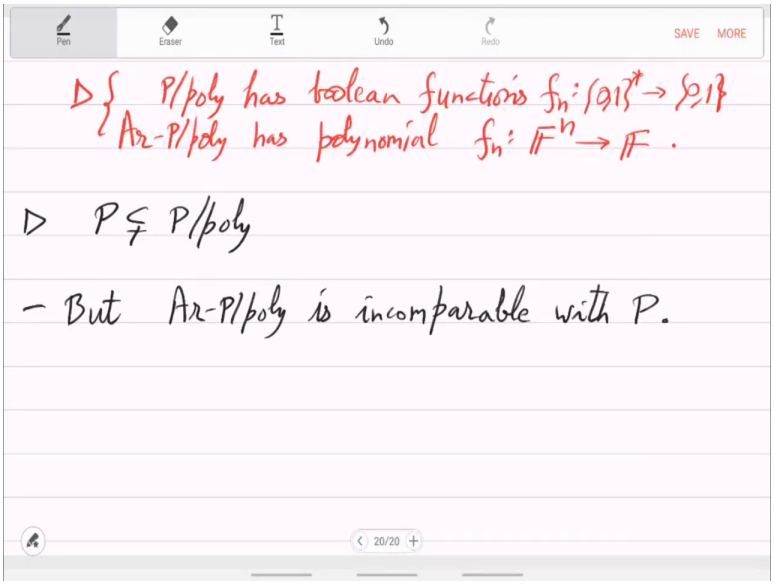
And the same thing you can do for arithmetic circuits or similar things arithmetic size s(n) is the collection of those problems. Problems now is polynomial family such that there exists s(n) size

arithmetic circuit family $C_n = f_n$. So syntactically it looks the same as above except that instead of a language we are talking about polynomial family.

It is a list of infinitely many polynomials f_n one for each number of variables or arity one for each ritn. So you want these respective arithmetic circuits C_n, $C_n = f_n$, C_n computes f_n and then you can define this complexity class. Let us call it arithmetic P/poly.

Arithmetic P/ poly:= $\bigcup_{c>0} Ar - Size(n^c)$. So you have P you have P/poly and now you have arithmetic P/poly.

(Refer Slide Time: 45:47)



So remember that P / poly has boolean functions $f_n: \{0,1\}^* \to \{o,1\}$. So P / poly actually collects boolean functions because it was solving a problem. So every boolean circuit is just solving a decision problem so its output is only 0 and 1. So you can think of P / poly as having boolean functions while arithmetic P / poly has polynomials which takes F to the an input and then output is a field element ($f_n: \mathbb{F}^n \to \mathbb{F}$).

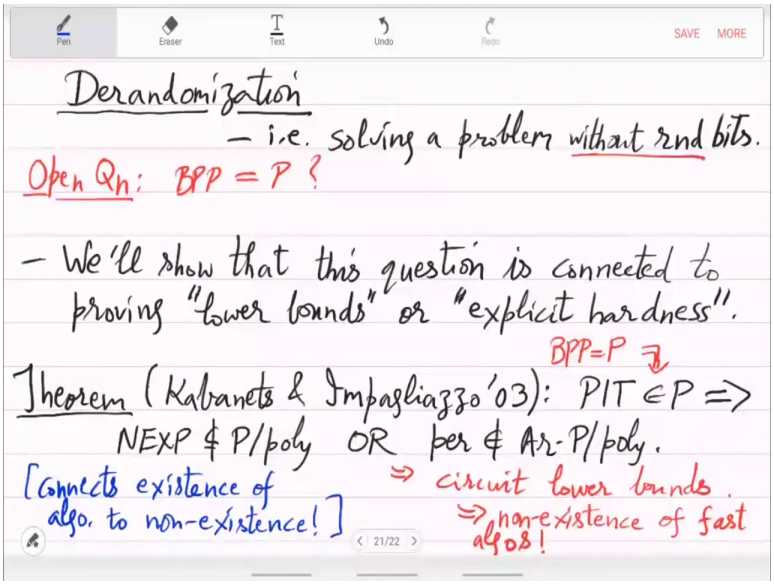
That is these this is how you can distinguish the two boolean functions versus polynomial functions. And it is easy to see that P is actually contained in P /poly and also P is strictly smaller

this you can show as an exercise. Because any algorithm is a turing machine which you can simulate as a boolean circuit but not the other way. So P is actually and you can do it in poly time versus poly size.

So P is actually contained in P / poly strictly but this arithmetic P / poly is incomparable. It is incomparable with P and it is also incomparable to P/poly this arithmetic circuit is doing something very different it is actually computing a polynomial. So it is not really clear how to compare polynomials with values. So now we have seen actually many definitions.

Complexity classes with respect to various versions of turing machines boolean circuits related complexity classes and arithmetic circuit related complexity classes. Let us now do something more concrete. So what we will now try to study is this randomized identity testing algorithm that you just saw. Can the randomness there be eliminated? So is there a D randomized Pit algorithm that is simultaneously fast.

(Refer Slide Time: 49:01)



So, derandomization means solving a problem without randomness. So, random bits are not available, so in PIT algorithm that you saw the RNA testing algorithm that actually requires random bits because you have to sample a point from a big space a very big space. So suppose you are not allowed to do that you are unable to do that. Then can you still give a practical algorithm that is the question?

So the open question is BPP=P so we will show that this question is connected to proving lower bounds or hardness explicit hardness. So lower bound means that you are able to show that an algorithm fast algorithm does not exist or that a small circuit does not exist. And explicit hardness means that this problem or this polynomial that you are proving hard this should be a natural object.

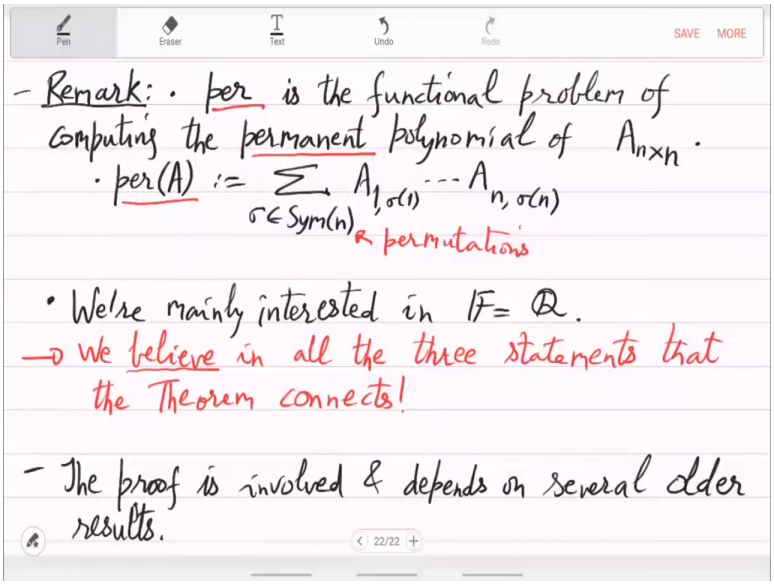
We actually it is not it is easy to show that problems there are problems which are very hard and there are polymers that are very hard but they have to be explicit as well to be of interest. So identity testing derandomization will actually give you that this is what we are interested in now. So this was shown very concretely by first time by Impasliazzo And Kabanets. They showed that if you prove $PIT \epsilon P$ then so if you find an algorithm for PIT which is fast.

And derandomized then either next is not in P/poly. So either this big very big class next non deterministic exponential time solvable problems either they will not have small circuits which is a very believable thing but we do not know a proof or this polynomial called permanent this will not have arithmetic circuits small ones. So one of these things will be true we do not know which one but we believe both of them to be true.

But the interesting thing about this theorem is that it is actually connecting existence of an algorithm with the non-existence of circuits either boolean or arithmetic. So notice that since PIT is in BPP, so BPP=P would imply this; if you show if you de-randomize all problems then you in particular de-randomize PIT also and that then gives you circuit lower bounds. So this is why we say that derandomization or BPP = P implies circuit lower bounds.

And if there are no small circuits then there cannot be fast algorithm this is the connection. So BPP = P saying that there is a fast algorithm for certain problems and in the end what this is implying is non-existence of fast algorithms for certain other problems. So this is why this theorem is of great interest it actually flips the existence to non-existence of algorithms.

(Refer Slide Time: 55:20)



So let me just define permanent. So permanent is the functional problem of computing. The permanent polynomial of a matrix $A_{n\times n}$. So it looks like this $Per(A) := \sum_{\sigma \epsilon Sym\,(n)} A_{1,\sigma(1)} \ldots A_{n,\sigma(n)}$ sigma is a permutation on n these are permutations. So if you think of a as a symbolic matrix withn^2variables. Then you carefully pick n of the variables out of n^2 and you form these monomials based on the permutation sigma you have $A_{1,\sigma(1)} \ldots A_{n,\sigma(n)}$ the product of that is a monomial and then take the sum of all these monomials this is the permanent polynomial.

If you know determinant this is very similar to that so this has no negative sign determinant is a negative sign. So we are mainly interested in the field what is the field here? The field of rationals. You can assume that the constants come from rationals and then is there a arithmetic circuit for permanent that is the question. So let me write down here that connects existence of algorithm to non-existence.

So we actually believe in all these three things we believe that BPP is P so PIT is in P. We believe that next has actually hard problems even x has, hard problems which cannot have small circuits. And permanent is a polynomial which is hard and cannot have arithmetic circuits of poly size we believe in all these things. So but we do not have a proof. So we do not yet have a proof for any of these three.

But this theorem is connecting all these three properties that we believe in. In all the three statements that the theorem relates so what is this connection? How do you prove this connection? We do not have proofs for the three statements any of the three statements but then how will we connect them. So this will be a topic for the next lecture. The proof will be involved and depends on several results.

So we have to approach the proof in step by step we will break it up into lemmas. Some of them will require concepts from basic complexity and some of them will require more advanced concepts. So the more advanced one we will do towards the end of this course but assuming that one lemma we will try to finish the proof in the next class.