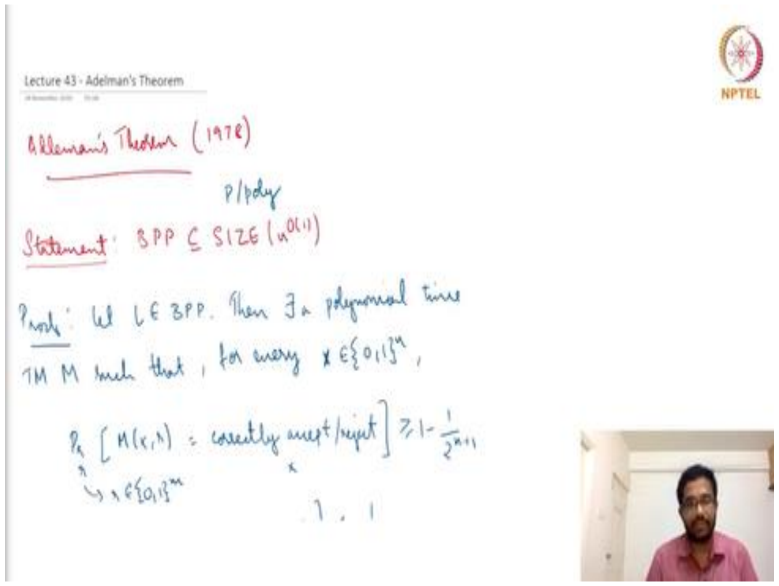
Computational Complexity

Prof. Subrahamanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture - 43
Adleman's Theorem

(Refer Slide Time: 00:15)



Lecture 43 - Adelman's Theorem

Adleman's Theorem (1978)

Statement: BPP $\subseteq$ SIZE$(n^{O(1)})$ P/poly

Proof: Let $L \in$ BPP. Then $\exists$ a polynomial time TM M such that, for every $x \in \{0,1\}^n$,

$\Pr_r[M(x,r) =$ correctly accept/reject$] \geq 1 - \frac{1}{2^{n+1}}$

$r \in \{0,1\}^m$

Hello and welcome to lecture 43 of the course computational complexity. In this lecture we will see Adelman's theorem which says how the class BPP is relates to circuits? So, what we will show in this class is that BPP is contained in size n power O1 or as we know it now P by poly. So, these are the same thing, so polynomial size circuits, so in fact the proof is very short and very brief. So, we saw earlier how BPP was related to other classes such as we saw BPP is in sigma 2 Adelman's theorem.

And here we see that BPP has polynomial size circuits, we still do not know whether BPP is a proper subset of P or not or proper. It is even a subset of P or not we do not even know if BPP is in NP for instance but interestingly we know that BPP is in P by poly. So, that, so let us see how that the proof goes. So, it the main ingredient is boosting that we are already seeing. So, suppose consider the language l in BPP like we always do we take an arbitrary language in BPP.

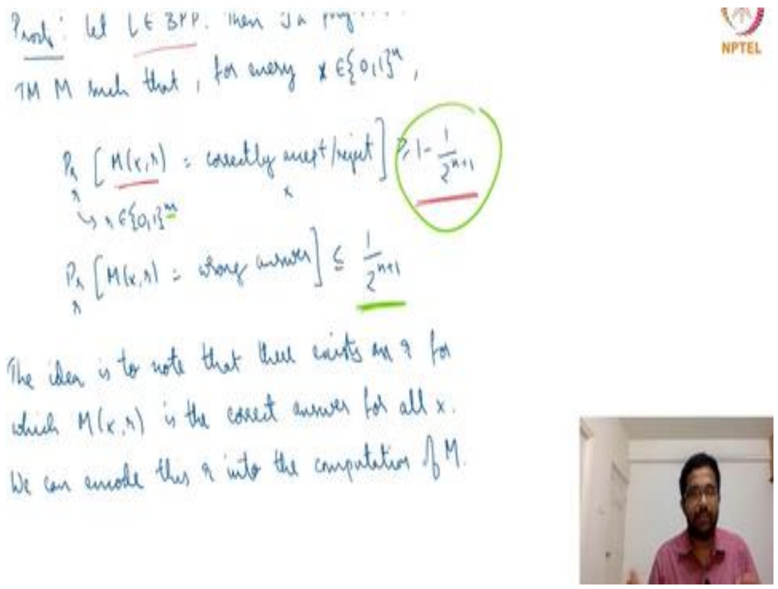
And then we will show that it is in it has polynomial size circuits. Then we could by virtue of boosting, we could boost the probability of success to such a high number that the probability of a string of length n getting accepted is at least 1 -1 divided by 2 power n + 1 if it is in the language, so where m is in the language sorry x is in the language and the probability of it rejecting is at least the same probability if x is not in the language.

So, in fact we saw boosting earlier when we studied BPP I think in lecture 30 or 31 we saw boosting in detail using Chernoff bound the BPP boosting. So, we said that when x is in l it should accept with this probability etcetera. And in lecture 32, we saw how BPP is in sigma 2 uses boosting, so we said that if x is in l it should accept with at least this much and x is not in l it should reject with at least this much.

So, the point is so here I am writing it a bit differently, whether x is in l or not in l the correct answer should come with the probability of 1 divided by 2 power 1 - 1 divided by 2 power n + 1. And again, this is, this should be true for all x. And the probability is taken over the choice of randomness, so let us say these random bits are chosen from some string of length M, M. So, M is the length of the random string.

And across this choice of the random strings the correct answer is chosen with property 1 - 1 divided by 2 power n + 1. So, it is predominantly correct and the probability of wrong answer is just simply 1 divided by 2 power n + 1. So, if so the meaning of correct answer is so if x is in the language, it should say accept if x is not in the language, it should say reject.

(Refer Slide Time: 04:10)



Proof: let $L \in BPP$. then $\exists$ a poly

TM M such that, for every $x \in \{0,1\}^n$,

$$\Pr_r [M(x,r) = \text{correctly accept/reject } x] \geq 1 - \frac{1}{2^{n+1}}$$

$r \in \{0,1\}^m$

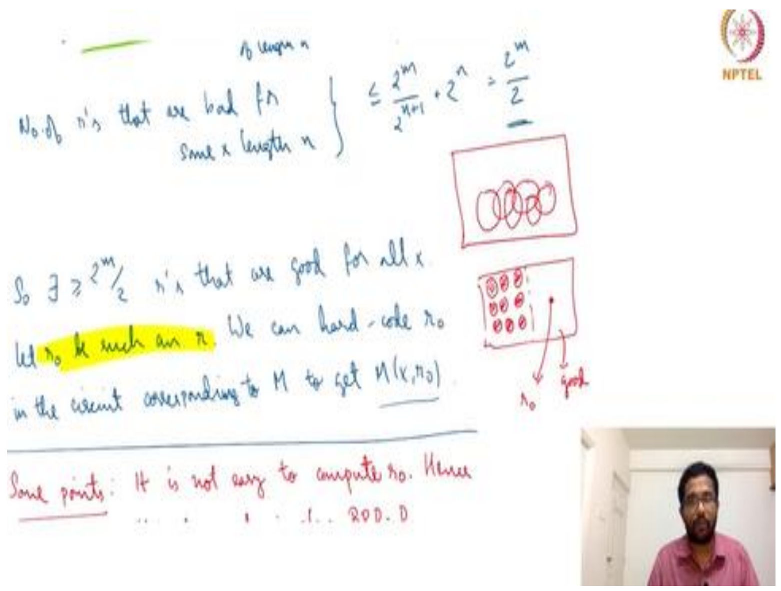
$$\Pr_r [M(x,r) = \text{wrong answer}] \leq \frac{1}{2^{n+1}}$$

The idea is to note that there exists an r for which M(x,r) is the correct answer for all x. We can encode this r into the computation of M.

Now the point here is that we do the boosting so much that the probability of error is so small and we can now look at the random strings are. And for each x we are saying that a large number of random strings output the correct answer, for each x a large number of random bits output the correct answer. Now what we will do is we will for specific input length n, we will look at all the strings of that length all the two power n strings.

And since and we will say that there is one string that correctly answers for all the 2 power n strings, meaning one random string that gives the correct answer for all the 2 power n input strings. And this correct random string can be used to make a circuit this is the proof, this is a brief idea the proof.

(Refer Slide Time: 05:14)


No. of r's that are bad for some x length n $\Big\}$ $\leq \frac{2^m}{2^{n+1}} \cdot 2^n = \frac{2^m}{2}$

of length n

So $\exists \geq 2^m/2$ r's that are good for all x.

Let r_0 be such an r. We can hard-code r_0 in the circuit corresponding to M to get $M(x, r_0)$

Some points: It is not easy to compute r_0. Hence

So, let me just explain it in a bit more detail. So, we have already seen the boosting up to 1 where the probability of error is at most 1 divided by 2 power n + 1 and the idea is to show that there is one random string which is the correct answer for all x of let us say of a certain length. Let me say of length n so there is one r that works for all the string of length n and this r can be used to build a circuit.

Anyway, in circuits we have non uniformity, meaning for each input length we could have a separate circuit and that is all that we are going to use here. So, how do we show that there is an r that works for all the all the inputs of a certain length. So, for a given x we say that r is good, we say that r is good. If for that x the Turing machine gives the correct answer or the algorithm gives the correct answer when it takes r as a random input.

So, r is a random string and r is considered good for x, if the machine gives a correct answer when it takes r as a random input. And what we know because of boosting is that most of the probability of wrong answer is 1 divided by 2 power n + 1, which means the so there are the random strings are chosen from 0, 1 power m that means there are 2 power m random strings, out of which there are 2 power m 1 by 2 power n + 1 fraction that gives a wrong answer.

So, the ones that gives the wrong answer we will call them bad r s. So, good are the correct are the random strings that give the correct answer for a given x and bad r s are the random strings that give a wrong answer. So, number of bad r s is actually total number of strings total number of random strings 2 power m divided by the 2 power n + 1, because or multiplied by the probability which is 1 divided by 2 power n + 1.

So, number of bad r s is at most, this is an upper bound on the probability one by two power n +1. So, how many strings are there of a certain length of the length n? So, for a given x let us say of length n we need to add this for a given x of length n, number of bad r s is this much because we are using n here. So, number of so now consider all the strings of length n, so maybe I will just change things a bit.

So, for some x number of string r s that are bad, for some x of length n, so for a given x of length n the number of bad r s are 2 power m divided by 2 power n +1 at most. How many r s are there? That are bad for some length some x of length n, you could just add like the worst case is that all the bad r s are separate. You could just take the union the size of the union is at most the size of the individual sets and you add them up.

So, it is 2 power m divided by 2 power n + 1 multiplied by 2 power n, the multiplication because there are 2 power n strings of length n. So, 2 power n divided by 2 power n + 1 multiplied by 2 power n, which is simply 2 power m divided by 2. So, the number of r s that are bad for some x of length n is simply 2 power m divided by 2, which means half the there are two power m random strings are so which means half the random strings are good for all the x.

These are the random strings they are bad for some x that is 2 power m divided by 2 which means the remaining ones. So, it is like this, so let me draw a quick figure, so these are the let us say these are the bad r s for some x these are another bad or there is another bad, so there could be overlapping and all that. But even if they are not overlapping what we show is that even if they are not overlapping even if you pack them all like if they are all separate even then roughly half the strings are good for everybody.

So, these are the small circles are the bad r s. And what we are saying is that? There are at least half the random strings are good for everyone. Now pick one such so this is good, this is good and pick one such random string and call it r 0, so let r 0 be such an r so this r 0 is good for all the x now this means that m of x, r 0 gives the correct answer for all the x of length n. So, we have now starting from a randomized algorithm where r were chosen randomly.

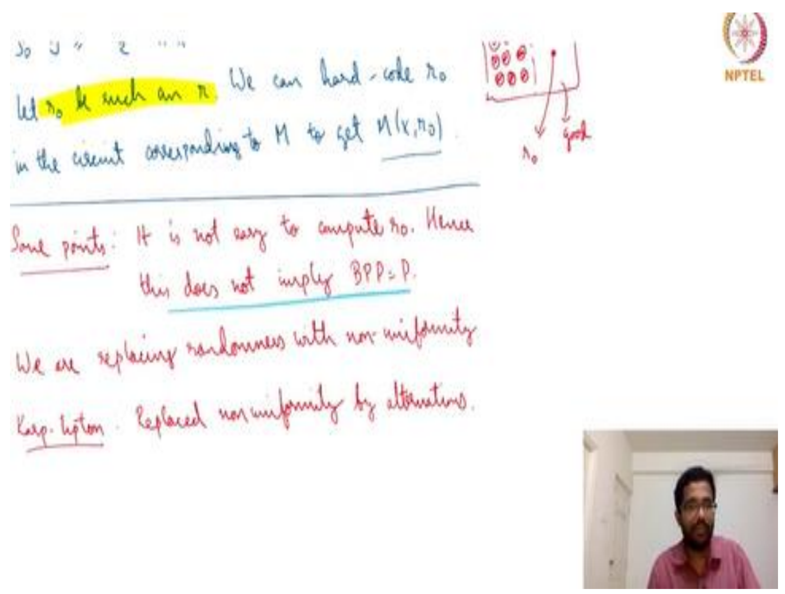
Once we identify r 0, we know that m of x r 0 gives a correct answer. So, now m of x r 0 is simply a polynomial time a deterministic polynomial time algorithm once r 0 is determined. But we do not know what r 0 is but we know that m of m x of x, r 0 is there. Now we could convert this deterministic polynomial time machine into a circuit and this circuit will we will have r 0 hard coded in it into it.

Because anyway we are considering x only of a certain length and we can also feed r 0 as a hard coded input. So, now one once x is input to that circuit it will always give the correct answer. So, now once you hard code x r 0 into that circuit it give the correct answer. This means that there is a circuit so we started with the BPP language now we are showing a polynomial size circuit, because it is a polynomial deterministic polynomial time algorithm m.

So, there is a polynomial size circuit for this for all the strings of a certain length, which is exactly what we want to show. So, just to summarize we just boosted it big enough so that booster this probability of error is small. And we say that the r s the random strings are good if they give the correct answer for a certain input and even if they consider all the bad possibly bad random strings we note that there are many that are good so we pick such one good random string.

And then hard coded to get a deterministic polynomial time algorithm and look at the circuit corresponding to that. This circuit will work for all the inputs of that length and that is the proof.

(Refer Slide Time: 12:42)



So ,, 2 ,, ,,

Let r_0 be such an r. We can hard-code r_0 in the circuit corresponding to M to get $M(x, r_0)$

Some points: It is not easy to compute r_0. Hence this does not imply BPP = P.

We are replacing randomness with non-uniformity

Karp-Lipton: Replaced non-uniformity by alternations.

So, one point to note is that even though we say that such an r 0 exists. It is not easy to compute such an r 0 just like we saw in the in the proof of the fact that like Shannon's theorem, where we said that there are hard circuits that require a certain size. But in fact, we saw that most circuits require that high large size, but we could not really find a specific function that requires a dark circuit. Similarly, we do not know how to compute an r 0.

If you know how to compute an r 0 then we could even make the circuit. But we do not know how to determine the r 0. So, all we know is that there are circuit families if a language is in BPP there are polynomial size circuit families for that language. So, we are just relying on non-uniformity here we do not know how to make these families make these circuits in fact if we could make this then we could even say that BPP is equal to P.

So, we cannot infer that BPP is equal to P, so this is still we do not know, so this does not imply that BPP is equal to P what really is happening here is that we are using the non-uniformity for each length we could use separate circuits. We are using the non-uniformity here. So, BPP is a random randomized polynomial time class. Now we are we want to we do not want randomness but then we are relying on non-uniformity.

We are kind of trading randomness and then using non-uniformity. So, it is like using one thing instead of the other. And just to compare in Karp Lipton theorem we saw that we were replacing the non-uniformity by the alternations. So, we so remember so we said that if sat has polynomial size circuits we could use self reducibility and we could this polynomial hierarchy could collapse. So, there we said that we could replace non-uniformity by alternations.

Here we are saying that we replace randomness with non-uniformity. That is the proof that BPP is in a P by poly. So, it is interesting to note how these classes which are defined in different models how even then they relate to each other. And that is all I wanted to say for this lecture. Thank you.