

Computational Complexity
Prof. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture - 53
Toda's Theorem: Part 2

(Refer Slide Time: 00:15)

Lecture 53 - Toda's Theorem (contd)

In the previous lecture, we saw the following theorem. This shows that a language in PH has a randomized reduction to \oplus SAT.

Theorem 1 (rand. reduction from Σ_k SAT to \oplus SAT)

Let $k, m > 0$. There is a probabilistic poly. time reduction A that given a Σ_k SAT instance ψ , outputs a \oplus SAT instance $A(\psi)$

ψ is true $\Rightarrow P_A[A(\psi) \in \oplus\text{SAT}] \geq 1 - \frac{1}{2^m}$

Hello and welcome to lecture 53 of the course computational complexity. In the previous lecture, we saw the Toda's theorem; we started seeing the proof of Toda's theorem. Toda's theorem shows that, any language in polynomial hierarchy can be simulated by a query by a deterministic polynomial time machine, along with a query to a sharp P oracle. And what we saw in the previous lecture was that any language polynomial hierarchy has a randomized reduction to parity SAT.

Or in other words, it can be simulated by a BPP machine with one query to a parity P oracle, parity SAT oracle.

(Refer Slide Time: 01:03)

Just to for the sake of completeness, I am restating it, if k and m are greater than 0, there is a probabilistic polynomial time reduction, such that any Σ_k instance ψ can be reduced to Σ_m . So, the reduction outputs $A(\psi)$, where A is a reduction. If ψ is a true yes instance, then with high probability yes, $A(\psi)$ will be a yes instance of parity SAT, if ψ is no instance, then with high probability is, $A(\psi)$ will be a no instance of parity SAT.

In other words, it is you can represent it as polynomial hierarchy is contained in BPP with parity P oracle and we need to go from here to this polynomial hierarchy is in P to the sharp P. So, we are weakening the base machine, but we are going to a stronger oracle. So, BPP to P means you are losing randomness parity P to sharp P means instead of counting, just the parity we are actually counting the actual number.

So, let us see how to do that and in this particular lecture will be relatively shorter as compared to the previous one. And, so let us see what is happening here. So, $A(\psi)$ is in parity SAT and with a certain probability. So, $A(\psi)$ being in parity SAT itself is saying that the number of satisfying assignment for $A(\psi)$ is odd or even. And then, we are saying that the probability that this is accepted is greater than $1 - 1/2^m$.

So, let us say we use r random bits or something like that. And so, which means 2^r are possible choices and out of that, we are saying that $1 - 1/2^r$ fraction of all the

possible choices. So, if you had another machine to count, how many of these choices lead to an odd parity. Then we could do that. So, basically, so to count the number of satisfying assignments of $A \psi$ we need one level of one counting.

And then apart from that we need another level of counting to count the number of $A \psi$, the fraction of random strings that, random choices that lead to $A \psi$ being accepted. So, there are two levels of counting. One for the choice of one for deciding whether $A \psi$ is in parity SAT and two for checking whether, this number of accepting computations is at least $1 - 1/2^m$ and same for the no case as well. So, there are two levels of counting here.

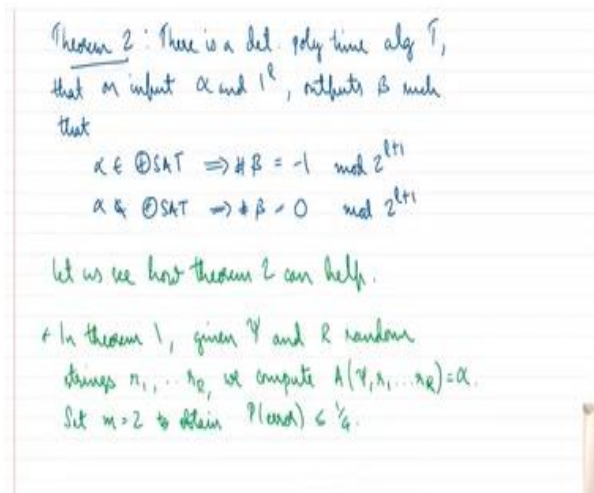
So, when we want to say that, we want to simulate it with just deterministic polynomial time machine with one sharp P oracle, we want to simulate using one level of counting. So, that is the task at hand. So, let us try to understand where we are. So, suppose ψ is true and in that case most of the $A \psi$ has an odd parity of satisfying assignments. If ψ is false means most of the $A \psi$ has an even parity.

But most having odd means, if you look at let us say so you can consider two levels. So, one for the choice of A and then another level where the number of satisfying assignments for $A \psi$. So, if you want to, just if you want one thing that you may want to try is just to add up the number of satisfying assignments over all the possible choices of A . But then, what we know in true in the case, when ψ is true is that most cases had an odd number of satisfying assignments.

And the case of false most cases is an even number. But then even odd could be interspersed mixed up, we cannot really tell it apart. We even if most cases are odd the sum could be even or would be odd and maybe even the yes instance and the no instance, the total sum could be. So, I am talking about the sum over all possible choices of A and all assignments like satisfying assignments of all possible choice of A number of satisfying assignments of $A \psi$.

So, this sum may not really give us much information about whether ψ was true or false. Because, this could be mixed up or it could the same value could result from yes instants as well as the no instance.

(Refer Slide Time: 06:20)




Theorem 2: There is a det. poly time alg T ,
that on input α and l^e , outputs β such
that

$$\alpha \in \text{SAT} \Rightarrow \# \beta = -1 \pmod{2^{l+1}}$$
$$\alpha \notin \text{SAT} \Rightarrow \# \beta = 0 \pmod{2^{l+1}}$$

let us see how theorem 2 can help.

In theorem 1, given Ψ and R random
strings r_1, \dots, r_k , we compute $A(\Psi, r_1, \dots, r_k) = \alpha$.
Set $m=2$ to obtain $P(\text{error}) \leq \frac{1}{4}$.



So, let us try to engineer away, where this sum will tell us the answer and that is what is going to happen. So, instead of A psi from A psi, we will go to another formula which will make sure that the sum itself will tell us whether it is yes instance or no instance. So, in order to get there, so here we have just parity, so odd and even. So, odd and even are mixed, so they are interspersed. There is no real difference.

So, we want to get a more significant difference and from that will be our starting point. So, in other words we want to amplify the gap between yes instance and no instance. So, what we do here is given, so that is our theorem two. This says that given an input formula alpha and given 1 power l which is just the number of or which is just 1 for the sake of an input l . There is a deterministic polynomial time reduction that outputs a formula beta.

Such that if A is a yes instance of parity set then, B will have minus 1 mod 2 power $l + 1$ satisfying assignments. So, think of l as some big number, so if l is 10 or if l is 5 then beta has minus 1 let us say if l is 5, then 2 power $l + 1$ is 64, 2 power 6, 64. Then beta has 63 mod 64 satisfying assignments. So, which is very close to the 64 and if alpha is not a yes instance of parity SAT, then beta has 0 satisfying assignment 0 mod 64 satisfying assignments.

And this is going to create a gap. Why is it going to create a gap? So, in theorem one, given A ψ we had two possibilities. One where the number of one where the probability if ψ is true, the probability that $A \psi$ is yes instance of parity SAT happened with probability at least $1 - 1/2^m$, which is a high probability. And, if ψ is false, it is yes instance with a low probability, $1/2^m$. So, if given ψ in theorem 1 we compute α .

α is basically ψ and with a bunch of random strings we do some processing. That is what we saw in the previous lecture to get an α . And we knew that the probability of ψ and the A ψ agreeing was $1/2^m$. So, let us set m to be equal to 2, so the probability that ψ is true is sorry the probability that A agrees with ψ is three-fourths and the probability of error is one-fourths. And, r is the number of random strings used by usage reduction.

(Refer Slide Time: 09:50)

The slide contains handwritten notes in green ink on a white background. At the top right is the NPTEL logo. The notes are as follows:

- Apply theorem 2 on α to get $\beta(z)$ such that
 - $\alpha \in \text{SAT} \Rightarrow \beta$ has $-1 \pmod{2^{l+1}}$ sat. assign.
 - $\alpha \notin \text{SAT} \Rightarrow \beta$ has $0 \pmod{2^{l+1}}$ sat. assign.
- If ψ was true, for $3/4$ fraction of the random choices, $\alpha \in \text{SAT}$. So for these α , we get β that have $-1 \pmod{2^{l+1}}$ sat. assignments.
- Adding up all the satisfying assignments of all possible choices of r_1, r_2, \dots, r_r , we get

A small tree diagram with a root node and three children is drawn to the right of the first bullet point. A small video inset of a man speaking is located at the bottom right of the slide.

So, what we will do is, so we have a formula, we have we start with ψ and we have all this A ψ . So, these are generated by different randomness, and then for each of these, we will count the number of satisfying assignments and we will take the total count. So, let us see what happens. So, if it is a yes instance so now let us apply theorem 2. Theorem 2 is this theorem that, if α is a yes instance of parity SAT β has $-1 \pmod{2^{l+1}}$ satisfying assignments.

And if α is a no instance, β has $0 \pmod{2^{l+1}}$ satisfying assignments which is what I have just copied down here. So, if so, we apply theorem 2 on α to get β , which has this

condition. And what does this mean? If ψ was a yes instance, most of the A_ψ like three-fourths of the A_ψ will produce yes instances. Which means three-fourths of the random fraction, three-fourths of the random choices will have α as yes instance of parity SAT and for this α we get $-1 \pmod{2^{l+1}}$.

(Refer Slide Time: 11:10)

all possible choices of r_1, r_2, \dots, r_l , we get

$$= (-1) \cdot \frac{1}{4} \cdot 2^k + 0 \cdot \frac{1}{4} \cdot 2^k$$

between -2^k and $-\frac{1}{4} \cdot 2^k \pmod{2^{l+1}}$

when ψ was false, the corresponding number of all satisfying assignments is between 0 and $-\frac{1}{4} \cdot 2^k \pmod{2^{l+1}}$

And, if you add up this, we know that the number of satisfying assignments over all the different choices of A_ψ , so we know that at least three-fourths are yes instances. So, it could be the three-fourths of all the random choices. And, it is $-1 \pmod{2^{l+1}}$ and that is at least and at most it could be all of them could be yes instances. So, it is all of the $2^{r \text{ prime}}$. So, let $r \text{ prime}$ be the number of random bits, $r \text{ prime}$ used in the theorem one.

So, we know at least three-fourths of multiplied by $2^{r \text{ prime}}$ choices give $-1 \pmod{2^{l+1}}$ or all of them give yes instances which is -1 multiplied by $2^{r \text{ prime}}$. So, the total number of satisfying assignments is anywhere between $-2^{r \text{ prime}}$ and $-3 \cdot 4$ multiplied by $2^{r \text{ prime}}$ because each yes instance contributes a $-1 \pmod{2^{l+1}}$. All of this is modulo 2^{l+1} .

So, either minus three-fourths $2^{r \text{ prime}}$ or $-2^{r \text{ prime}}$ or anywhere in between and, if ψ was a false formula if it is no instance, then similarly we can do the same calculation, so we know that the number of yes instances at most is at least 0 anywhere between 0 and one-

fourths. So, if all the instances are no instances, then we get 0 modulo 2^{l+1} . Otherwise, we could have up to one-fourth fraction yes instances.

So, it is -1 by 4 multiplied by $2^{\text{power } R \text{ prime}}$ yes instances. So, in this case the range is 0 and -1 by 4 $2^{\text{power } r \text{ prime}}$. So, mod $2^{\text{power } l+1}$ so there are two ranges.

(Refer Slide Time: 13:30)

Handwritten slide content:

between -2^l and $(-\frac{3}{4})2^l \pmod{2^{l+1}}$

When χ was false, the corresponding number of all satisfying assignments is between

0 and $(-\frac{1}{4})2^l \pmod{2^{l+1}}$

Diagram: A number line from 0 to $2^{l+1} = 2^{R+1}$. A green shaded region is between 0 and $2^l - \frac{1}{4}2^l$. A red shaded region is between $2^l - \frac{1}{4}2^l$ and $2^{l+1} - 2^{R+1}$. A note says "Suppose $l = 2^l$ ".

if $l = R - 1$

The final reduction is to do the following:

NPTL

Video inset of a person in a green shirt.

So, let us look at the entire thing modulo $2^{\text{power } l+1}$. And, just for the sake of easiness for concreteness, let us set R prime to be equal let us set l to be equal to R prime. So, this is from 0 to range $2^{\text{power } l+1}$ which is the same as range $2^{\text{power } R \text{ prime} + 1}$. So, yes instance could lead to $-2^{\text{power } R \text{ prime}}$ and -3 by 4 multiplied by $2^{\text{power } R \text{ prime}}$. So, $2^{\text{power } R \text{ prime}}$ is roughly $-2^{\text{power } R \text{ prime}}$ is roughly midway point $-2^{\text{power } R \text{ prime}}$ is equal to $+2^{\text{power } R \text{ prime}}$ in the case of modulo $2^{\text{power } R \text{ prime} + 1}$.

And $-\frac{3}{4} 2^{\text{power } R \text{ prime}}$ gives somewhere here so the yes instances are in this range. The total number of satisfying assignment in the case of yes instances are in the green range. And the no instances are from $-\frac{1}{4} 2^{\text{power } R \text{ prime}}$ till 0 . So, this entire range is $2^{\text{power } R \text{ prime} + 1}$. So, this is the midway point and this green portion is roughly one-fourth of the half portion or one-eighth of the entire portion.

And, similarly the red portion is also roughly one-eighth of the entire portion. The point is that the green the yes instance contributes the green zone and no instances contribute to the red zone. And the red zone and green zone are disjoint are separated. So, now we can safely so in fact I think, even if you had tried l to be equal to R prime - 1, even then we could have worked. If l was, so we would have got some so if this is set aside, if $l = R$ prime - 1.

Then we would have got something like this, I think. So, let us say this is the midway point is this then we would have got the green zone to be here and the red zone to be here. But anyway, even $l = R$ prime works, so in both cases it works. But, if l is smaller than R prime - 1, I do not think it will work, because then I think the zones will start merging and then we will not be able to tell them apart.

So, now all that we need to do is to take the first reduction first reduction in second reduction put it together. And ask how many satisfying assignments are there for this entire process. So, even the first reduction instead of doing it as a randomized reduction with random coins we have to view it as non-deterministic choices. So, over all the non-deterministic choices how many accepting computations are there? And, if we can do that, we can tell which case it is, so this one counting is required.


(Refer Slide Time: 16:31)


The final reduction is to do the following.

- + Given Σ_k -SAT instance ψ as input
- + Construct an NFM that takes r_1, r_2, \dots, r_ℓ as non-deterministic choices.
- + NFM builds X using ψ, r_1, \dots, r_ℓ .
- + NFM builds B using T and X .
- + Accepts if B is satisfiable.

As seen above, the no. of accepting computations of this NFM helps us conclude if ψ is satisfiable.

What remains is to do the amplification.





So, I am just written down the entire thing here, so given the sigma k instance psi is input. So, you construct a non-deterministic Turing machine that takes r_1, r_2 which is a random choices as non-deterministic choices. In fact, I think I said $r \in R$, so maybe I will just change it to $r \in R$. And, the first it builds the non-deterministic Turing machine builds alpha using the reduction shown in the previous lecture using ψ, r_1, r_2 up to $r \in R$.

And, now alpha is obtained and now we build beta which is deterministic using T and alpha which is using the process T and you accept if beta is satisfiable. We accept if beta has a satisfying assignment, so now there are two levels. So, one is the choices of r, r_1, r_2 up to $r \in R$ and then choices inside beta. So, how many accepting computations are there using both these non-deterministic choices with?

So, even the random reduction in theorem one or the previous lecture we are now viewing it as a non-deterministic set of choices. So, the two sets of non-deterministic choices, so psi and then A psi or alpha and then we get beta which is still random and then how many accepting computations are there for each level of beta. So, at the end we are counting all the accepting computations of all the betas that are situated like this.


And you count the number of satisfying assignments and see which band it lies, whether it writes in the green band or in the red band and that is all. So, just with one counting we know whether beta is in the green or red and beta is satisfiable and that tells us whether alpha was a yes instance or not. No, we do not look at individual beta or alpha. So, just look at the total number of satisfying assignments overall.

So, that tells us whether psi was yes instance or a no instance because the green, red bands correspond to only psi. And, that is all that we need so now there is no randomness, the randomness has been replaced by just the amplification and then in one counting, just one counting. And, this non-deterministic Turing machine, we just want to count the number of accepting computations.

(Refer Slide Time: 19:28)

as non-deterministic choices.

- + NTM builds X using $\mathcal{V}, \eta_1, \dots, \eta_2$.
- + NTM builds P using T and X .
- + Accepts if B is satisfiable.



As seen above, the no. of accepting computations of this NTM helps us conclude if \mathcal{V} is satisfiable.

All that needs to be done: ask the #P oracle the no. of accepting computations of this NTM, and check whether in the \mathbb{P} or \mathbb{R} .



So, all that we need to do is ask the oracle all that needs to be done, ask the sharp P oracle the number of accepting computations of this NTM for the given input and that is it and check whether it is the green or red. So, this completes the proof assuming theorem 2. So, this is complete the proof of the Toda's theorem. So, all we are doing is build this non-deterministic Turing machine.

So, building this non-deterministic Turing machine is entirely deterministic polynomial time process. So, we do not even need to know what is r_1 r_2 anything we just build on a non-deterministic Turing machine. And, so what is alpha the process to construct alpha and the process to construct beta, is all that the non-deterministic Turing machine needs to do. So, it is like writing a program and then asking, how many ways that can this program succeed.

So, build this Turing machine, which completes the reduction and then check and ask the oracle, how many accepting computations it has. So, that is it. It is a very interesting thing, all that we are doing is one amplification going from $0, 1$, to this big $0, 1$. So, this parity SAT is like $0, 1 \pmod{2}$, or $0 - 1 \pmod{2}$, and from there we are going to $0 - 1 \pmod{2}^{\text{power } l + 1}$ and that is what is giving us this big range.

If it was $0, 1$ so if l was really small here, it would not have helped. If l is in fact l is 0 means it is just again parity.

(Refer Slide Time: 22:14)

(Theorem 2)

Recall: Given formula ϕ that has m
and ϕ' that has m' sat. assignments
We can build,

- > Formula with $m+m'$ sat. assign
- > Formula with $m \cdot m'$ sat. assign
- > Formula with C sat. assign
where $C > 0$ is an integer.

The given α is either in Θ SAT or not.

So, now let us see how to do the amplification. So, given alpha we need to build beta so alpha is like 0 1, yes instance of parity or no instance of parity from there we need to move to 0 or - 1 mod 2 power l + 1. So, yes instance of parity means we need to move to - 1 mod 2 power l + 1, no instance of parity means we need to move to 0 mod 2 power l + 1. So, recall we in the previous lecture, we had shown given a formula phi and phi prime that has m and m prime satisfying assignments.

We could build formula with $m + m$ prime satisfying assignments, m multiplied by m prime satisfying assignments and a constant number of satisfying assignments. So, we could do m plus constant satisfying assignments, as long as it is a positive constant. So, these are the tools that we have, you could add two formulas to get a formula with which has the number of satisfying assignments is the sum or the product and we can even add a positive integer.

(Refer Slide Time: 23:22)



→ Formula with $m + m'$ sat. assign
 → Formula with $m \cdot m'$ sat. assign
 → Formula with C sat. assign
 where $C > 0$ is an integer.

A formula τ' that has $4m^3 + 3m^4$ sat. assign when τ has m sat. assign

The given α is either in Θ SAT or not.

Given formula τ , consider the formula $4\tau^3 + 3\tau^4$.

$\tau \equiv 0 \pmod k \Rightarrow 4\tau^3 + 3\tau^4 \equiv 0 \pmod{k^2}$

$\tau \equiv -1 \pmod k \Rightarrow 4\tau^3 + 3\tau^4 \equiv -1 \pmod{k^2}$

$4(a-k)^3 + 3(a-k)^4$



So, given alpha, the alpha is either in parity SAT or not, it is either odd number of accepting computations or even number. So, given a formula, so now let us try to understand we given a formula tau. Consider this formula 4 tau cube, maybe I will just move it, 4 tau cube times + 3 times tau power 4. So, think of this as because, so by this I mean that given a formula tau, we could build a formula that let us say tau accepts 10 satisfying assignments.

Now, I could build a formula that accepts 10 squared satisfying assignments, just by what we saw about by constructing some product etcetera and similarly I could build a formula that accepts 10 cubed. So, consider the formula this means, what I mean by this is that a formula tau prime, that has 4 m cube + 3 m power 4 satisfying assignments where tau has m satisfying assignments. So, that is what I write mean by the shorthand notation.


So, if tau has m satisfying assignments, this new formula must have 4 m cube + 3 m power 4. But then this is just simpler to write 4 tau cube + theta. This is not exact equal to 4 tau cube + 3 tau power 4 in the strictest sense. So, let us try to understand, suppose tau the number of satisfying assignments of tau was 0 mod k. Then 4 tau cube + 3 tau power 4 is 0 mod k square because, number of satisfying assignments is a multiple of k.

Then here we are taking tau cube and tau power 4. So, clearly this is a multiple of k squared. So, if k divides m then certainly k divides 4 m cube + or k square divides 4 m cube + 3 m power 4.

That is all that I am doing here. Now, suppose tau is $-1 \pmod k$. That is the condition $-1 \pmod k$. Then the claim is that $4 \tau^3 + 3 \tau^4 \equiv -1 \pmod{k^2}$. Let us see why this is the case.

So, suppose let us say tau is $a k - 1$, so where a is some constant, so $4 (a k - 1)^3 + 3 (a k - 1)^4$. Let us see what each of this is. If you do modulo, if you expand the cube you will get something like, a cube $k^3 + 3$ or maybe not $+ - 3$ a square $k^2 + 3 a k - 1$.

(Refer Slide Time: 27:24)



$$\begin{aligned}
 \tau \equiv -1 \pmod k &\Rightarrow 4\tau^3 + 3\tau^4 \equiv -1 \pmod{k^2} \\
 (ak-1) &= 4(ak-1)^3 + 3(ak-1)^4 \\
 &= 4(a^3k^3 - 3a^2k^2 + 3ak - 1) \\
 &\quad + 3(a^4k^4 - 4a^3k^3 + 6a^2k^2 - 4ak + 1) \\
 &= 4(3ak - 1) + 3(-2ak + 1)^2 \pmod{k^2} \\
 &= 4(3ak - 1) + 3(-4ak + 1) \pmod{k^2} \\
 &= -4 + 3 = -1 \pmod{k^2}
 \end{aligned}$$

Starting from $x \equiv 0, -1 \pmod 2$, we can repeatedly apply the $4x^3 + 3x^4$

That is for the first term, plus maybe I will just write this a bit differently $3 a^2 k^2$, so I will just write it as $a k - 1$ maybe I will just write a bit differently here also $- 2 a k + 1$. So, I have written it as $a k - 1$ squared squared. So, now this one if you look at it modulo k^2 , I can ignore all the terms that are multiples of k^2 . So, this is just 4 times $3 a k - 1 + 3$ times $- 2 a k + 1$ whole squared modulo k^2 .

And, this is fine 4 times $3 a k - 1$, if you again expand this you will see that the k^2 term vanishes, so what will remain is so the 3 remain outside, $3 - 4 a k + 1$ and modulo k^2 and what will remain is that the $12 a k +$ and $-$ will cancel. So, this term will cancel and what will remain is $- 4 + 3$, which is $- 1$. So, if you started with something $- 1$ modulo k , so the operation this $4 \tau^3 + 3 \tau^4$.

Meaning which is a corresponding formula, that has this many satisfying assignments. If you started with tau, that has 0 modulo k satisfying assignments the new formula has 0 modulo k square. If we started with - 1 modulo k satisfying assignments the new formula has - 1 modulo k square. So, from modulo k we are moving to modulo k squared. And, this is for any k this works for any k.

(Refer Slide Time: 29:40)

Starting from $x = 0, -1 \pmod 2$, we can repeatedly apply the $4x^2 + 3x$ transformation $\log(l+1)$ times to amplify to the 2^{l+1} modulus.

At the end, we want $l = R^l$, where R^l is the no of random bits used in theorem 1. This was $O(l)$ where the original formula was a Σ_2 -SAT instance.

So the resulting formula is $\text{poly}(l)$ size of original formula.

$2 \rightarrow 2^2 \rightarrow 2^4 \rightarrow 2^8 \rightarrow 2^{16}$
 $\rightarrow 2^{l+1}$
 $\log(l+1)$ steps

NPTEL

And, we started with alpha being in either parity SAT or not parity SAT. So, which means the number of satisfying assignments was either 0 or 1 modulo 2 or 1 modulo 2 is the same as - 1 modulo 2. And we could repeatedly apply this operation I could go from 0 and - 1 modulo 2 to 0 and - 1 modulo 4 and 0 and -1 modulo 4 to 0 and - 1 modulo 16. Every time the modulus thing is squaring. So, every time it is squaring, so 4, 16, 16 square 256 and so on.

And, we will we need to get to 2 power $l + 1$. So, but, every time we are repeatedly squaring so we are going something like 2, 2 squared 2 2 power 4, 2 power 8 and so on. So, we need to go to 2 power $l + 1$, so we need to do $\log l + 1$ many steps, $\log l + 1$ many steps, iterations. And so, we had want l to be roughly equal to R prime. That is what we said here, so this ranges red and green ranges remain separate.

So, and R if you remember from the previous lecture, it was roughly the number of it is the number of random bits, and this was roughly the size of the original formula. So, it may be a

polynomial in the size of the original formula. But, the number of iterations that we do is $\log l + 1$. But if you do whenever we do one step the size of the formula kind of also becomes roughly blows up by some constant.

So, we do $\log l + 1$ step, so there is always a some constant power $\log l + 1$ blow up happening in the size. But since $\log l + 1$ is something polynomial in the size of the formula, the resulting formula will also be polynomial size in the original formula. And this so which means from the original formula that we had we are constructing a new formula which is polynomial size of original formula but with the amplification.

So, original was the result of theorem one construction and now we have the theorem 2 formula beta. And finally, we as I said already, we combine theorem 1 and theorem 2 and we count all the number of satisfying assignments or we do not count that is counted by the sharp P oracle and we just check whether which range does lie in the red or the green range. So, just to summarize the whole.

The second part of the proof was just mainly this modulus amplification where we moved from yes or no instance of parity SAT the 0 and - 1 modulus some high modulus $2^{\log l + 1}$ by using this trick. And, then after that we just cascaded this or compose these two reductions, the randomized reductions and the formula beta. And we after the amplification the yes instances would result in the total number of satisfying assignments being in a certain range.

And the no instances would result in the total number of satisfying assignments being in some other range. So, the yes instance and the no instance are separate from which we could tell whether the original formula which was a Σ_k instance was a yes instance or a no instance. And, so that is a deterministic machine with just one query to a polynomial with a sharp P oracle, to a sharp P oracle, deterministic polynomial time machine with one query to a sharp P oracle. And that completes the proof of the Toda's theorem.

So, perhaps I will just revise just to summarize this entire week. We had basically just two results, one was the permanent is sharp P complete which was variance theorem. So, we saw

what was permanent and then we saw that we could convert sharp 3 SAT into a problem of counting the permanent or estimating the permanent. So, this was a somewhat involved reduction with a too many gadgets.

And the second lecture was a continuation of the same. So, it is interesting because permanent and is, we saw that it is only definition was it is not that far away from determinant. If you put the sign here it is a determinant, if you do not put the sign here it is permanent. So, the sign makes up all the difference, the sign this - 1 power sign component. But then, determinant is very nice in the sense that it allows these row operations.

You can add a copy of a row or a multiple of a row to another row, still the determinant does not change you can do Gaussian elimination and all this makes the computation very tractable. But permanent does not allow most of these operations. So, and as you have seen, if you can compute the permanent, we can solve any problem in sharp P and we have also seen that, sharp P can simulate or one query to sharp P can answer any question in the polynomial hierarchy.

So, that is the power of sharp P, so hopefully we appreciate the theorem that permanent sharp P complete even more now. So, even though the small the sign; adding the sign changes the complexity of determining or computing the determinant to computing the term permanent so much. So, this lecture 51 was just a continuation, lecture 52 and 53 were Toda's theorem which was a kind of involved even perhaps even more involved theorem.

And perhaps one of the most involved theorems that we see in this entire course. That showed that, a language in the polynomial hierarchy can be simulated by a deterministic polynomial time Turing machine, with just one query to the sharp P oracle. And 53 was continuation of the same and that is all that we have for week 10 and thank you.