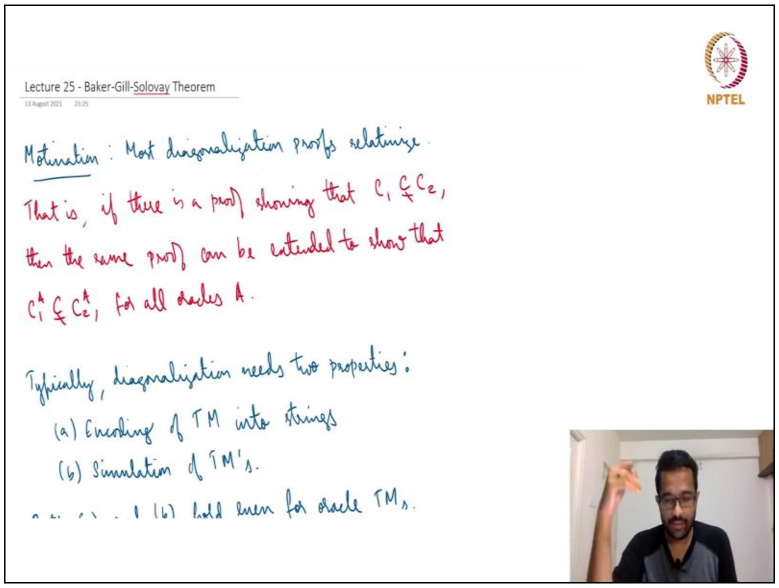
Computational Complexity

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

Lecture -25
Baker-Gill-Solovay Theorem Part 1

(Refer Slide Time: 00:15)



Lecture 25 - Baker-Gill-Solovay Theorem
13 August 2021 23:25

Motivation: Most diagonalization proofs relativize.

That is, if there is a proof showing that $C_1 \subsetneq C_2$, then the same proof can be extended to show that $C_1^A \subsetneq C_2^A$, for all oracles A.

Typically, diagonalization needs two properties:
(a) Encoding of TM into strings
(b) Simulation of TM's.

[illegible] (b) hold even for oracle TMs.

Hello and welcome to lecture 25 of the course computational complexity. Today we are going to see an interesting theorem called the Baker Gill Solovay theorem. So, Baker Gill and Solovay approved in 1975. So, let us first describe the motivation and then I will say the theorem the thing is that we have seen time hierarchy theorem space hierarchy theorem. So, the undecidability of halting problem all of this were proved using diagonalization and most diagonalization proofs relate to wise.

So, what do I mean by relativise is that they can be extended to a setting where the turing machines use oracles as well. In other words if there is a proof that shows that a complexity class is con is strictly contained in another complexity class let us say C 1 is subset but not equal to C 2 then the same proof can be extended to show that C 1 with the oracle A is subset but not equal to C 2 to the oracle A for any oracle A.

So, again this is very important. So, this is this will be true for any oracle that you choose this because the the I will explain the reasons below because the thing is that proofs that use diagonalization basically relies on 2 things one is that we need to be able to encode the turing machine into some strings. So, we usually say things like sorry given the description of a turing machine M then you run the turing machine M on its own description and then flip the output this is what this is what you typically do in the case of diagonalization.

And then this will be part of a description of a turing machines let us say called it D. So, D may have other things and then you ask questions like what happens when d is fed its own description and then you arrive at a contradiction this is how proofs involving diagonalization work, halting problem space hierarchy everything. There are some other things like how much time how much space then you then you curtail it truncate it and so on. So, basically it relies on the notion of interpreting or encoding a turing machine as a string and reading a string as both the description of a turing machine as well as the input to the above mentioned turing machine or to some turing machine.

And the second thing that that proofs using diagonalization need is the capability to simulate one turing machine using another turing machine. So, you we the proofs have things like again what I just said we say things like you run M on its own description. So, basically the turing machine that we have I earlier called it D. So, D should be able to run M for any M. So, if M 1 is given it should be able to run M 1.

So, basically the tutoring machine D has to be has to have the capability of running simulating any other turing machine. So, both of these are required and essential for a diagonalization proof and both of these can be well be extended to oracle turing machines as well because if you can simulate a turing machine then you can also simulate an oracle turing machine. Because all the only new thing that we have for an oracle tutoring machine is the fact that there are some oracle to calls to be made.

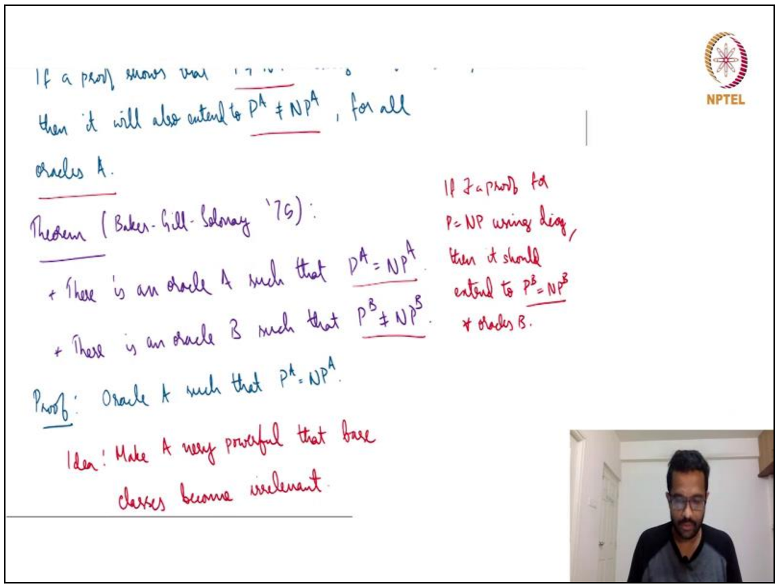
So, now oracle calls is like checking the membership and this part can also be included in the simulation checking the membership and then answering yes or no. Again if necessary it can be done by the simulating machine as well. Second is the encoding of turing machine into strings again this can be also done for oracle tutoring machines because as I described in the one of the previous lectures oracle turing machines are just like any other turing machines but they have 3 special states q query, q yes, q no and then an extra tape.

So, it is well fits well within the framework of of any other turing machine. So, you can have a similar encoding and it can be interpreted by another turing machine. So, both of these encoding as well as simulation can be done for the oracle turing machines. So, consequently any proof that can be that uses diagonalization you could just mildly tweak the turing machines involved in those proofs and get a proof for the relativized version of the classes as well.

So, if there is a proof that says complexity class C 1 is contained in complexity class C 2 you can use the same using diagonalization you can use the same proof to show that C 1 to the oracle A is contained in C 2 to the oracle A if it says that C 1 is not contained in C 2 then again same proof works to show that C 1 to the A is not contained in C 2 to the A. So, whatever you say between C 1 and C 2 it relativizes meaning it can be set for C 1 to the A and C 2 to the A this is the main this is what relativization means.

This holds because both of these A and B encoding and simulation can be hold even for oracle turing machines. So, this is the property of relativization. Relativization means even upon modifying the turing machine to include oracles the statements should hold good.

(Refer Slide Time: 06:43)


If a proof shows that

then it will also extend to $\underline{P^A \neq NP^A}$, for all oracles A.

<u>Theorem</u> (Baker-Gill-Solovay '75):

+ There is an oracle A such that $\underline{P^A = NP^A}$.
+ There is an oracle B such that $\underline{P^B \neq NP^B}$.

If ∃ a proof for $P = NP$ using diag, then it should extend to $\underline{P^B = NP^B}$ ∀ oracles B.

<u>Proof</u>: Oracle A such that $P^A = NP^A$.

Idea: Make A very powerful that base classes become irrelevant.

So what this means is that. So, let us try to understand the consequences. So, one may wonder like because we saw P versus because we saw time hierarchy space error cases undecidability which seem to be big problems in the case of computability theory. So, Now can one possibly solve P versus NP using diagonalization it is a very natural question that somebody may ask can we use some diagonalization technique very smart very creative to show that P is equal to NP or P is not equal to NP.

So, one thing to note is that because of the relativization thing that we just noted if there is a proof that P is equal to NP, suppose P is equal to NP if there is a proof that uses diagonalization to show that P is equal to NP then the same proof should extend or should relativize to show that P to the A is equal to the is equal to NP to the A for any oracle A. If there is a proof that shows that P is not equal to NP using diagonalization then that should also relativize meaning it should extend to the statement that P to the A is not equal to NP to the A for any oracle A.

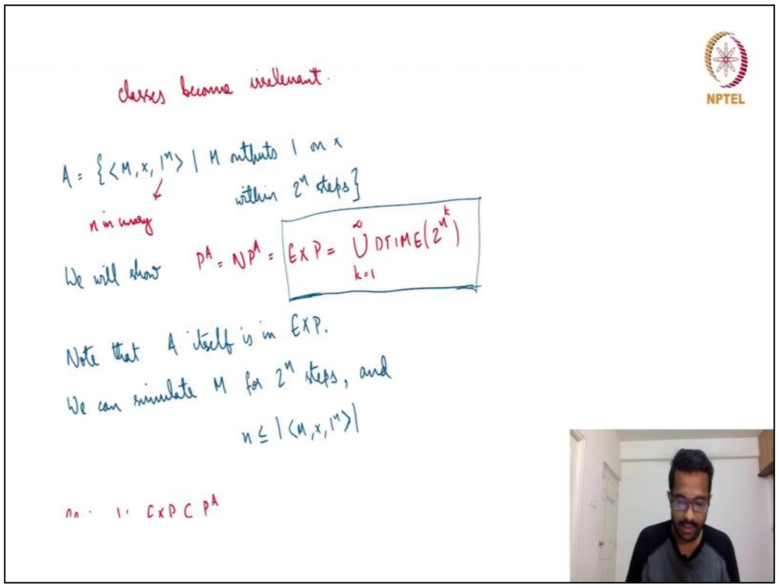
So, this is what if there were there was a there was there was a way to settle P versus NP using diagonalization then the same statement should hold good even with oracles and what Baker Gill and Solvay showed in 1975 is that P versus NP cannot be resolved using diagonalization in either way P equal to NP is also not possible P not equal to NP is not also not possible, how did they do that?

They showed 2 oracles A and B such that with respect to a P to the A is equal to NP to the A and P to the B is not equal to NP to the B. So, if if someone showed anyway it is written above if somebody showed that P is not equal to NP using diagonalization then you should be able to relativize meaning P to the A will be not equal to NP to the A for all oracles. But here we have an oracle that shows that P to the A is equal to NP to the A we have an oracle where P to the is equal to NP to the A.

So, that rules out a proof of P not equal to NP using diagonalization. Similarly if there was a proof if there is a proof for P equal to NP using diagonalization then that should also relate device meaning it should extend to P B equal to NP to the B for all oracles B. However Baker Gill and Solovay showed another oracle B such that P B is not equal to NP to the B. So therefore you cannot show that P equal to NP using diagonalization as well.

So, you cannot solve it P equal to NP you also cannot solve P not equal to NP using diagonalization hence you cannot settle P versus NP using diagonalization this was the result of Baker Gill and Solovay.

(Refer Slide Time: 10:43)



classes become irrelevant.

$A = \{ \langle M, x, 1^n \rangle \mid M$ outputs 1 on x within 2^n steps$\}$

n in unary

We will show $P^A = NP^A = \boxed{EXP = \bigcup_{k=1}^{\infty} DTIME(2^{n^k})}$

Note that A itself is in EXP.

We can simulate M for 2^n steps, and

$n \leq |\langle M, x, 1^n \rangle|$

$EXP \subset P^A$

So, basically they ruled out told people do not even try tank analyzation in an attempt to solve P versus NP such diagonalization alone is not going to be enough because because of this. So, it's very, very interesting. So, it is not a proof that P equal to NP it's not a proof that P not equal to

NP it is a proof that shows that a certain technique to solve the problem will not work. So, so it like many as i have said before many complexity theorists have spent a lot of time and effort and that 2 very intelligent people have spent a lot of time on the P versus NP questions.

So, even this kind of an input do not try this approach even that is helpful but this is not just some random statement do not try this approach it is actually a statement backed up with a perfectly logical and correct proof that if you try this you will not succeed because of this and this reasons. So, that way it is very, very very interesting result perhaps the first result of this type this proof attempt or this proof approach or this proof technique does not work.

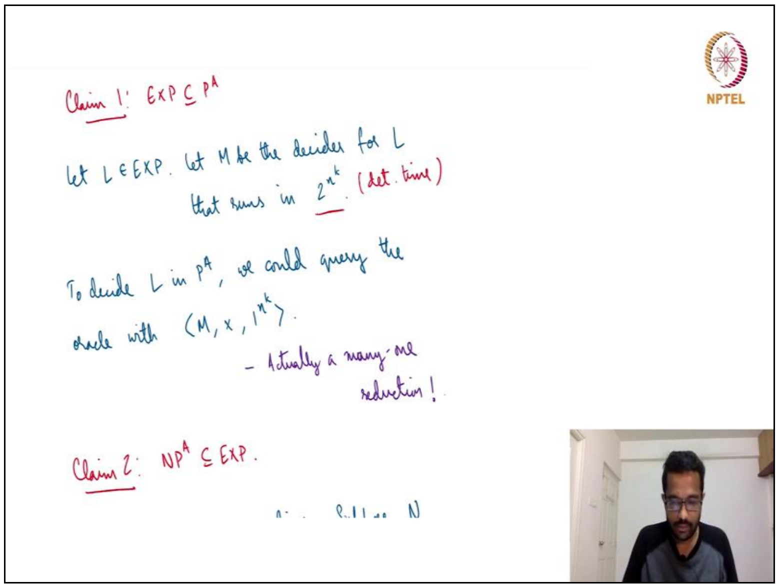
So, let us see the first part of the proof first which is an oracle a such that P to the A is equal to NP to the A. So, the basic idea is to make A. So, you want an oracle such that. So, we do not know what P and NP how they are related we it may be equal it may be a proper subset. So, Now how to what kind of A will make P to the A equal to NP to the A. So, the idea is to make A very powerful and you make it. So, powerful that it does not matter what kind of base machine is asking queries.

So, even if the one has non determinism one does not even then the oracle is. So, powerful that it does not really matter if the base machine had not non-determinism or not. So, that is the idea. So, to make a very powerful, so, the language that I am describing is something that runs in exponential time. So, what is x time? So, the complexity class x time or sometimes in denominator e x B I am not defined this before in this course is the union of all DTMEs 2 power n power k. So, if I had DTME n power k it would have it would be polynomial time 2 power n power k means it is the exponent is polynomial, so, its exponential time.

So, this is x and the language and the oracle or the language A is M x 1 power n one power n just means n ones. So where M is a description of a turing machine x is description is some string and one power n is well just n once. And what is the language the set of all such 3 tuples or 3 tuples where M outputs one or maybe you could say M accepts on x given on x within 2 power n steps.

So, where 2 power n steps is the number of steps and that is so and that is given by one power n one power n meaning you are given n in unary. So, this is just one power n is nothing but n in unary. So, you are given n in unary and the language is that M should output one on the input x within 2 power n steps M should output 1 on x within 2 power n steps. And first of all notice that this language is an exponential time why is that?

(Refer Slide Time: 14:59)



Claim 1: EXP $\subseteq P^A$

Let $L \in$ EXP. Let M be the decider for L that runs in $\underline{2^{n^k}}$. (det. time)

To decide L in P^A, we could query the oracle with $\langle M, x, 1^{n^k} \rangle$.

– Actually a many-one reduction!

Claim 2: $NP^A \subseteq$ EXP.

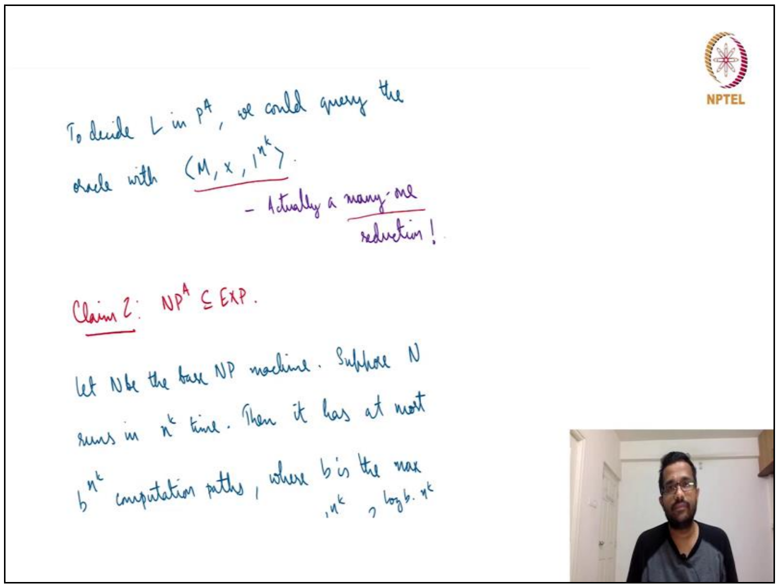
Because we could simulate M for 2 power n steps on x on the input x for 2 power n steps. So, it may take maybe roughly or order 2 power in steps maybe 2 per multiplied by some constant or some something like that and the in. So, and the input length is bigger than n. So, the input length is the entire is basically equal to the disc the size of the description of M plus length of x plus n.

So, this is certainly bigger than n or at least n and it runs in 2 power n steps. So, which means that the running time is 2 power n which is less than exponential time or at most exponential time. So, but in 2 power n steps so, whatever so, whether it outputs 1 or not we can decide and then we can simulate and respond. So, A itself is an exponential time. Now what we will show is that P power A or P to the A and NP to the A are both equal to x let us see why?

So, first of all we will show that x is contained in P to the A. So, as I have said in the previous lecture we take an arbitrary. So, whenever we have A containment like this to show we take an arbitrary language from the left hand side and show that this belongs to the hand side. So, let L be in arbitrary line an arbitrary language in x time which means there is a x prime decider for L. So, let that decide to be M and let M run in 2 power n power k time this is what x time was 2 power n power k should be the running time complexity.

So, basically it runs in it is also deterministic time non dominant because it is in P it is in x time and to decide L in P to the A again this is P to the A. So, which means it is a polynomial time machine that has the language a as an oracle. So, it we can ask questions of the form M x 1 power n. So, what we can do. So, whether M runs we know M runs into power n power k time. So, we can just give it we can just ask the oracle does it does it give the answer into power in power k time.

(Refer Slide Time: 17:59)



To decide L in P^A, we could query the oracle with $\underline{\langle M, x, 1^{n^k} \rangle}$.

– Actually a $\underline{\text{many-one}}$ reduction!

$\underline{\text{Claim 2}}$: $NP^A \subseteq EXP$.

Let N be the base NP machine. Suppose N runs in n^k time. Then it has at most b^{n^k} computation paths, where b is the max

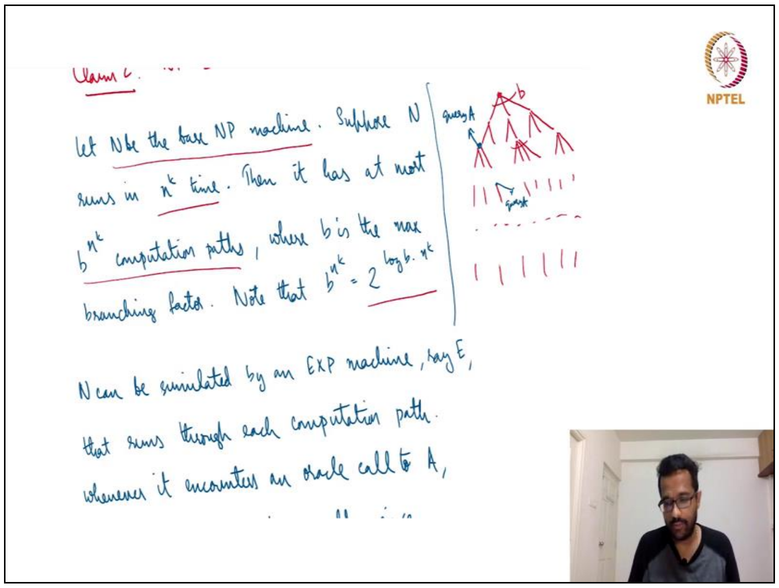
$, n^k \quad 2^{\log b \cdot n^k}$

So, to ask this you need to ask you need to give n power k in unity that is and n power k once you have to give and the description of the turing machine M and the input x. So, to decide whether x is in L or not we give M, x, 1 power n power k which is just n power k in unary to the oracle. So, if it M accepts x into power n power k time then the oracle will respond yes otherwise it will respond no and that is it.

So, and what does the P to the A machine actually the base machine have to do? The base machine just has to just write down this one M x 1 power one power n power k in the tape in the query tape and this takes this can certainly be done in polynomial time because yeah n power k is what it has to write and M and x are fixed x is the input. So, this can be done. So, in fact this is actually a many one deduction.

So, this is actually a many one direction why because it is like to decide whether x is in L you are just making a call to making a call to the a oracle with that with the help of a polynomial time running machine. So, we just make one query. So, it is a many one reduction as well you can view it as a many one direction. So, a language in arbitrary language in x prime is contained in a machine that AP turing machine.

(Refer Slide Time: 19:58)



Claim 2.

Let N be the base NP machine. Suppose N runs in n^k time. Then it has at most b^{n^k} computation paths, where b is the max branching factor. Note that $b^{n^k} = 2^{\log b \cdot n^k}$

N can be simulated by an EXP machine, say E, that runs through each computation path. whenever it encounters an oracle call to A,

query A
b
query

Second part is to show that NP to the A is contained in x time . So, this is slightly more involved let us see the proof. So, let N be the NP machine. So, there is NP to the A. So, the base machine be n suppose. So, it is an NP machine. So, it runs in polynomial time non-deterministic polynomial time suppose the running time is n power k. Now as you may recall. So, when you have a non-deterministic machine there are several possible computation paths.

And then so this is this how the structure looks like what is the number of computation paths we do not know but let B be the branching factor from in this figure I think four is a maximum branching factor if the max branching factor is B which means the number of children in this computation tree maximum number of children for any node in this computation tree then it can have at most B power height and the height is the time taken and in which in this case is n power k.

So, B power height is B power n power k computation paths and it has that many computation paths which means B power n power k can also be written as 2 power log B multiplied by n power k. So, that many computation paths are there. So, we need to show that this is contained in x. So, some consider some language in NP to the A that has to be shown to be contained in x. So, Now there is an NP machine which has at most 2 power log B times n power k which is in which is an exponential quantity and we could use the x time machine to to just run each or go through each of these paths.

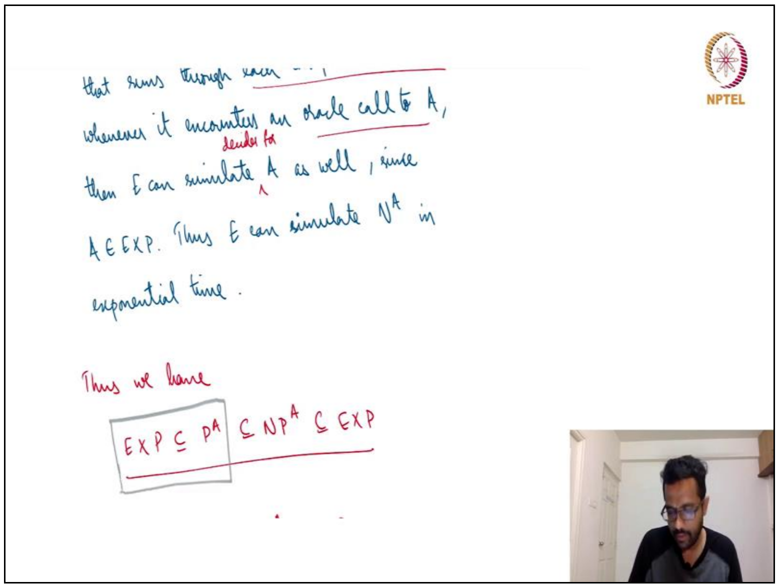
So, there are that B power and power k paths and each of length n power k. So, it is it is x time it is upper bounded by x prime the only thing is that it is not simply an NP machine but it is also an entity machine with A oracle. So, now it may encounter let us say at this this particular point it may say Now query A. So, it is just not just the matter of counting down each computation paths it may say query A it may say query a many places.

So, this queries also have to be performed but the x prime machine does not have access to any oracle which means it has to actually run the line run the decider for a or maybe check for the membership in A by its own means but as we have already seen the language A itself is in x. So, as we have already seen a itself is an x. So, when there is a need to query A what can be done by the x prime machine is to is to simply run the exponential time algorithm for A.

And then output the answer and then the x time machine can continue the simulation of the non-deterministic machine. So, all that one does or the x prime machine does is to run through each of these paths in the computation tree and we know there are at most exponentially many paths in fact 2 power log B multiplied by n power k with and you may encounter the A queries but when you encounter the A queries all that one has to do is to run the x prime decider which is also.

So, and that will be another exponential quantity the time taken for that but multiplication of 2 exponential quantities is also that will also be another exponential time quantity.

(Refer Slide Time: 24:14)



that runs through each

whenever it encounters an oracle call to A,

then E can simulate decider for A as well, since

$A \in EXP$. Thus E can simulate N^A in

exponential time.

Thus we have

$\boxed{EXP \subseteq P^A} \subseteq NP^A \subseteq EXP$

So, one now n can be simulated by an x prime machine may be let us call it E. So, it can com run through each computation path and whenever there is an A oracle call each computation path and then whenever there is an oracle called to a then E can simulate the a decider I have written simulate A. So, I mean decider for A as well since A is a cell itself in x prime. Thus E can simulate n power A in exponential time this means that n power a in arbitrary language in n power a sorry not any power A n power A.

So, an arbitrary language in NP power A is contained in x. So, we have shown first that x is contained in polynomial P to the a then we have shown that NP to the A is in x. And finally we have this connection also simply because P is a subset of NP. So, P to the A will also be a subset of NP to the A whether it is A subset or not we do not know but we do not need to know that because we have a chain of containments x contained in P to the a containing NP to the A contained in x.

So, we have a chain of containments where the left hand side and the hand side are the same. So, what if just to understand what if you had an e chain of equality inequalities like which says that 3 is less than or equal to x just in terms of numbers less than or equal to x less than or equal to y less than or equal to 3 this would there is only way to there is only one way to make all this true the one way is that x and y are both equal to 3.

Similarly there is only one way to make all this true it is that P power A and NP power A are equal and equal to x hence P power A equal to NP power A which is what we wanted we wanted an oracle a such that P power a equal to n P po wer A and again just to just to remind you this rules out any oracle sorry any proof of that shows that P not equal to NP using diagonalization.

I will just summarize very quickly the idea is to we have to show P power A equal to NP per A the idea is to make a powerful. So, that the base machine does not the the limitations of the base machine does not come into picture first of all exponential time is contained in P power A. Because you could just simply query a once and get the answer and NP power A is contained in exponential time.

Because you could simply run through all the computation paths in exponential time and whenever there is an A oracle query that query also can be performed in exponential time. Hence NP power A is contained in exponential time and obviously P power is contained in P power A because it's a subclass hence it follows that NP power A equal to P power A. There is one more thing I just wanted to say is that instead of;

So, here we had an x time language A x prime oracle A another thing that works is a PSPACE complete language like TQBF, TQBF also works and the proof outline is pretty much the same. So, you could show that NP power TQBF is contained in PSPACE and PSPACE is containing P power TQBF and then the finally we have the sandwich equation people PSPACE is contained NP power TQBF contain in P or NP power TQBF containing the PSPACE again.

So, it has to be the case that P power TQBF equal to NP power TQBF. So, A equal to TQBF also works but I leave it to you to work out the details of this particular proof. So, with that I think I will conclude this this lecture part because and I will show the oracle B that for which P power B is not equal to NP power B in the next lecture.