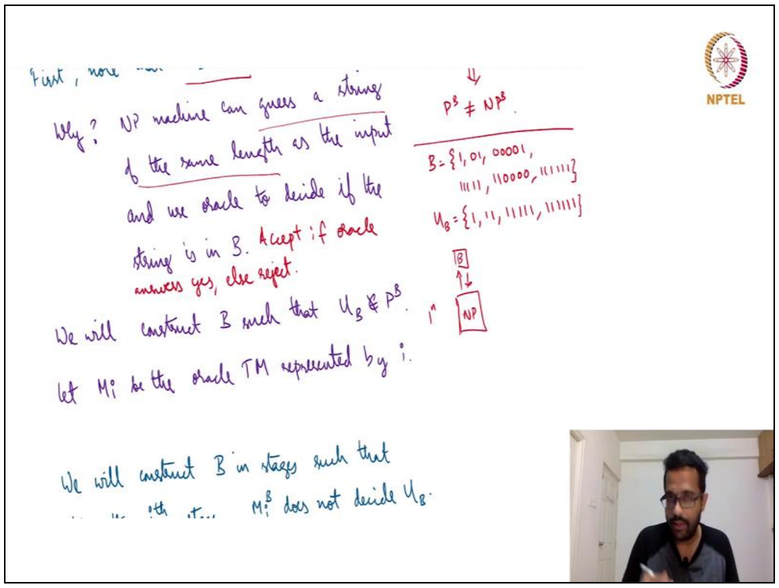
Computational Complexity
Prof. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture -26
Baker-Gill-Solovay Theorem Part 2

(Refer Slide Time: 00:15)


First, note

Why? NP machine can guess a string of the same length as the input and use oracle to decide if the string is in B. Accept if oracle answers yes, else reject.

$\Downarrow$

$P^B \neq NP^B$.

$B = \{1, 01, 00001, 11111, 110000, 111111\}$

$U_B = \{1, 11, 11111, 111111\}$

1^n $\boxed{NP}$ $\uparrow\downarrow$ $\boxed{B}$

We will construct B such that $U_B \notin P^B$.

Let M_i be the oracle TM represented by i.

We will construct B in stages such that

ith M_i^B does not decide U_B.

Hello and welcome to lecture 26 of the course computational complexity. In this lecture we will continue from where we left off in the lecture 25 and complete the proof of Baker Gill Solovay theorem. So, Baker Gill Solovay theorem shows the existence of an oracle A such that P to the a is equal to NP to the a which we saw in lecture 25 and this lecture we will see the second part which is the existence of an oracle B such that P to the B not equal to NP to the B.

And together these 2 show that we cannot use techniques that relate device such as diagonalization to settle P versus NP problem. So, the second part is more interesting than the first because the goal is to construct a language B such that P to the B is not equal to NP to the B. So, and to do this construction we will rely on diagonalization ourselves. So, so this is very interesting because we are trying to show that diagonalization does not really work in order to settle P versus NP.

But interestingly the proof that that shows that diagonalization does not work is actually relying on diagonalization itself. So, that way it is extremely interesting and as all proofs of with all proofs of diagonalization there is an amount of like do not worry if you get confused or if the reasoning is complicated just try to think calmly and try to watch it again or try to see it again after 2 or 3 times you will have a better understanding of what is really happening here.

So, let us see how we get the language B and in fact the language B is actually going to be constructed as part of the proof itself. So, this is another layer of another layer of complication in the proof I do not want to use the word complication because it is another layer of interesting another interesting layer of the proof that the language is constructed in the proof itself as part of the proof itself.

So, let me ask you this suppose there is a let us say polynomial time machine P that has access to an oracle B this is an oracle B. So, it can ask and it will respond. Now will this now suppose you have to decide the language B itself it is it is certainly possible. So, now to decide B you just give it a string and the polynomial time the machine will just give the same string as input the B oracle and then output yes or no correctly but here the requirement is different.

Here we have to get an the decision version of the problem that is asked of the machine should not be this should not be to decide the same language as the oracle it has to be. It has to use the oracle but still not be able to answer this question. So, that is what we will see. So, the language that we that. So, what we will do is the oracle language is B and we will show the existence of a language u B and we will first show that u B is contained in NP to the B.

And then we will show that u B is not contained in P to the B and because P is sub contained in NP there is obvious implication that P to the B is contained in NP to the B and here we are demonstrating one language that is in P to the B sorry that is an NP to the B but not in P to the P which means these 2 are different the simple these 2 together imply that P to the B is not equal to NP to the P. So, this is how the proof goes. So, let us see what is u B? u B is simply is unary language meaning all the strings in the language are just comprised of ones it is just asking it is just one power n if there is a string of length n in I will just illustrate to the side.

So, suppose B is let us say 1 0 1 0 0 0 0 1 in the increasing order of length all 1 1 1 1 1 1 0 0 0 0 and 1 1 1 1 1. So, it has one string of length one one string of length 2 2 strings of length 5 and 2 strings of length 6. So, u B in this case would simply be because it has a string of length one it has a string of length 2 it has 2 strings of length 5 it doesn't matter all it needs is one string of length 5 and 2 strings of length 6.

So, it has a one string of length 6 this would be u B basically if there is a string of length i it includes it includes the unary i which means i once. So, this is u B first let us see that u B is is contained in NP to the B. So, this is this part u B is contained NP to the B regardless of what B is. So, this is easy to see basically u B is contained NP to the B because in order to decide u B. So, you have an NP machine sorry you have an NP machine which has access to a B oracle in order to decide u B. So, basically it is given 1 power n for some number n and what we are asking is what we have access to is the oracle for B.

So, we have to determine from the B oracle we have to determine is there a string of length n in b. So, but then we have an NP machine we have access to non determinism. So, suppose N is 10. So, the NP machine can guess a string of length 10. So, basically if its alphabet is binary it has to length guess a string of 0 1 1 0 something some length some string of length N and then check whether that string is in B by an oracle call.

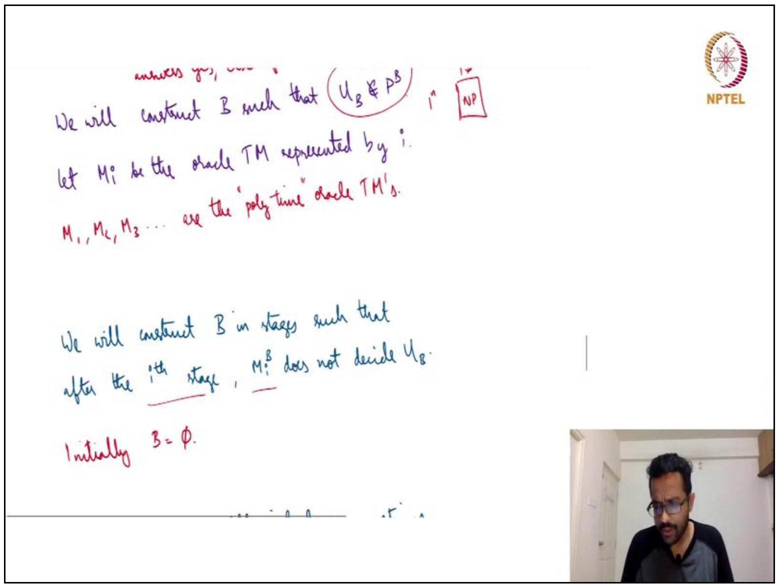
And this is because it is an NP machine and if there are 2 there could be 2 power 10 strings of length 10 if it is a binary alphabet if it is with a ternary or bigger alphabet it will be correspondingly higher. But the point is that you can have an oracle turing machine guess and query and if there is such a string then there will be a correct guess. So, all the NP machine has to do is to guess a string of the same length and query the oracle. It accepts if oracle answers yes else reject.

So, if there is a string of length n out of the many computation pulse there will be one accepting path. If there is no string of length n none of the paths will accept. So, this is y for any oracle B, u B is contained in NP power B. Now what we will do is? So, this is the easy part the hard part is to show that u B is or to construct B. So, till now notice that till now all that B could be anything all that I have told is the connection between u B and B.

Now we will construct B that u B cannot be in P B. So, now you can you can see the kind of intelligence going on here like I said earlier if you had if you were to query if you were to decide B itself for a P for a machine with access to an oracle B it is it is trivial you just have to ask is the given string in the language B. But now u B makes it complicated or u B makes it interesting because given is we are just giving the length given one power n we are just we have to find out if there is a string of length n.

But then there could be if it is a binary alphabet there could be 2 power n strings of the same length now which is like how will the machine determine if or which string to query. And it has to do it in polynomial time it cannot query all the possible to power and length 2 power n strings of length n. It has polynomial time it cannot keep querying all the strings it has to do it in limited time.

(Refer Slide Time: 10:47)



We will construct B such that $U_B \notin P^B$ 1^n $\boxed{NP}$

let M_i be the oracle TM represented by i.

$M_1, M_2, M_3 \dots$ are the "poly time" oracle TM's.

We will construct B in stages such that after the $\underline{i^{th}\ stage}$, $\underline{M_i^B}$ does not decide U_B.

Initially $B = \phi$.

So, what we will do is suppose let us consider let us consider M 1, M 2, M 3 etcetera as oracle turing machines. So, these are all different oracle turing machines is a countable set. So, you can list them all down and what we will do is; we will construct B in stages then we will construct the language B in stages. So, first we will make sure that B M 1. So, we want to show that u B is not contained in P B or P to the B.

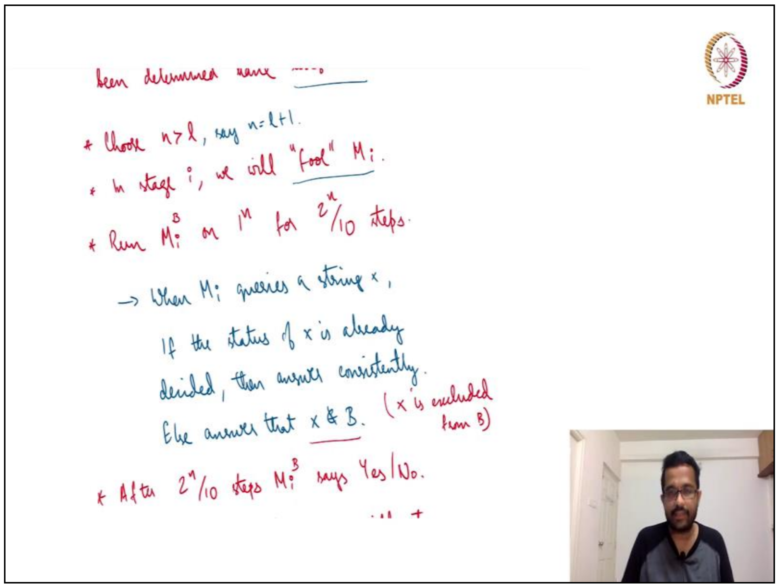
So suppose these are the oracle turing machines or these are the or the let us say polytime oracle turing machines the polytime part is not really required but still just to make the point I will say it like this now ah we will rule out one by one each of these turing machines. So, first we will show that first we will build the language B in such a way that M 1 cannot decide u B then we will add some strings to B.

So, that M 2 cannot decide P B then we will add some more. So, that M 3 cannot decide P B and so on. So, step by step this construction can be done and we will we will explain how that will rule out the possibility of any of these machines even so, not just M 1 to the B. So, we will construct B in stages such that after eighth stage M i to the B does not decide u B. So, none of these turing machines even with access to the oracle B cannot decide u B.

So, the problem is really what I said it is because it basically is getting the input as the length of the string and then it has to decide whether that string is in B or not or that length string is in B or not but then B could have exponentially many or there are potentially exponentially many strings of that length and then you can only have you have only limited time. So, you cannot possibly query all of that.

So, this is the idea that we will do we will we will see how M i operates and we will make sure to include something that M i cannot find. Basically if M i looks at 10 particular strings of a certain length we will look at we will include 11th string that M i will not look at. So, this is what we will do. So, the machine will answer it is not there but but we will we would have included one such that the machine is pooled.

(Refer Slide Time: 13:49)


been determined name ....

* Choose $n > l$, say $n = l+1$.
* In stage i, we will "fool" M_i.
* Run M_i^B on 1^n for $2^n/10$ steps.
  - → When M_i queries a string x,
    If the status of x is already decided, then answer consistently.
    Else answer that $x \notin B$. (x is excluded from B)
* After $2^n/10$ steps M_i^B says Yes/No.

So, to start with B is the empty set and one by one we will build the set B. So, in each stage we will include some strings into B and we will we will forbid some strings from B we will make sure that so, some strings are going into B some strings are certainly not going into B some strings we may not decide in any stage but that is but some strings will be put in the accept or in B phi or some of them are the not in B phi.

So, let us see what we do in a general stage, stage i. So, suppose till now we have been addressing some uh strings and so, some strings we know are in B some things we know are not in B. Suppose the longest string amongst all those strings for which the status has been determined. So suppose consider the longest string for all those strings for which the status has been determined up to the previous stage up to stage i - 1.

Suppose the longest string has length L. Suppose a longer string has length L, now you set n to be L bigger than L say you can even set n to be l + 1. So, basically now what we have is that for this value of n for this value of n the status of all the strings of that length n are open none of them have been put into B none of them have been excluded. So, basically that gives us some flexibility which will be useful in the stage i we will make sure to fool the machine M i.

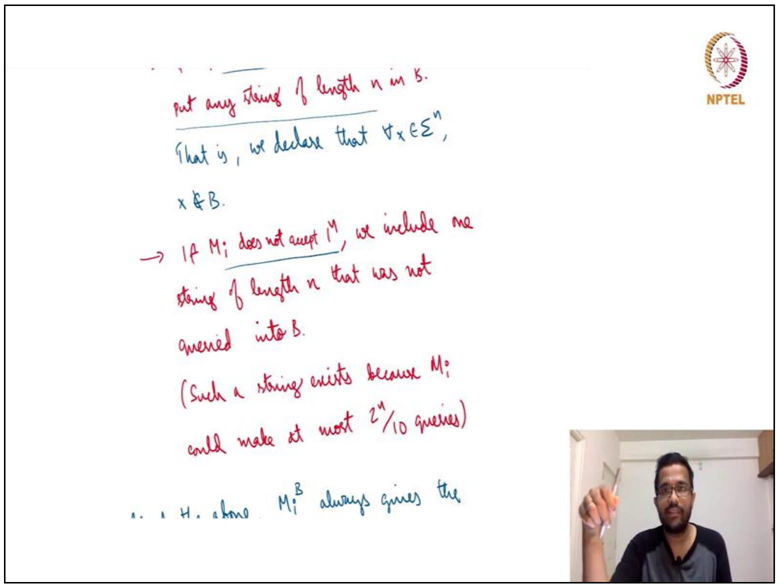
Basically we will again this I said already but we will make sure that M i to the A or M i to the B cannot decide u B that we will ensure that in by excluding or including some strings in into B at the stage i. So, what we do in stage i is to run M i on 1 power n which is the unary input. So, run M i meaning actually you are running M i to the B sorry M i to the B. But at this stage B is only partially constructed it is still under construction but that is .

When it gets to a new query that is not yet decided we will define what it does that is that again; so, this is how B is constructed when there is a new query for which the membership in B is defined we will define at that point . So, suppose M i queries a string x, so M i has access to B oracle. So, now M i wants to know the status of a string x. Now if the status has already been determined of that string then we just report as per whatever it has been decided.

Again this is something that one needs to note if once a string is included into B or excluded from B the status of that string does not change meaning we do not modify already included in we do not remove from the included part or we do not include from the remote part we only include or remove for the undecided part. So, if the status is already decided then we answer as per whatever we decided in the previous stages.

If it is not decided then we say x is not in B. So, suppose we get a new query the status of which is not known we say it is not there. So, now we answer that x is not B not in B and this is; so, now x is excluded from B, x is put in the exclude phi and from now on whenever this particular x is queried it will be said that it is not in B.

(Refer Slide Time: 18:02)


put any string of length n in B.

That is, we declare that $\forall x \in \Sigma^n$,

$x \notin B$.

$\rightarrow$ If M_i does not accept 1^n, we include one string of length n that was not queried into B.

(Such a string exists because M_i could make at most $2^n/10$ queries)

M_i^B always gives the

Now we run M i on M i to the B on one power n for 2 power n divided by ten steps. So, notice this number this is also very interesting because this is kind of exponential time. So, which means that it is bigger than any polynomial that that can that you can think of but it is still lesser smaller than 2 power n and that is that plays a role. So, after 2 power n divided by 10 steps after this many steps the machine M i with the oracle B it may say yes it may say no or it may loop could loop also.

If it loops then that means certainly it is not polynomial time but if it is a polynomial time machine then maybe it just erase this and say if it is a polynomial time machine if it runs in polynomial time. Otherwise it may loop will take will handle all that if it runs in polynomial time. So, there are 3 possibilities or 2 possibilities one is that M i ends up accepting 1 power n or not accepting not accepting could be rejecting or looping.

So, if M i accepts 1 power n, so, this is the most interesting part basically we have to make sure that M i we have to fool M I, basically if the correct answer is yes then we have to make M i say no. If the correct answer is no then we have to say make M i answer yes. So, if M i accepts n, so, what does it mean for M i to accept n which means ideally if M is a decider for u B then it would mean that there is an n length string in B.

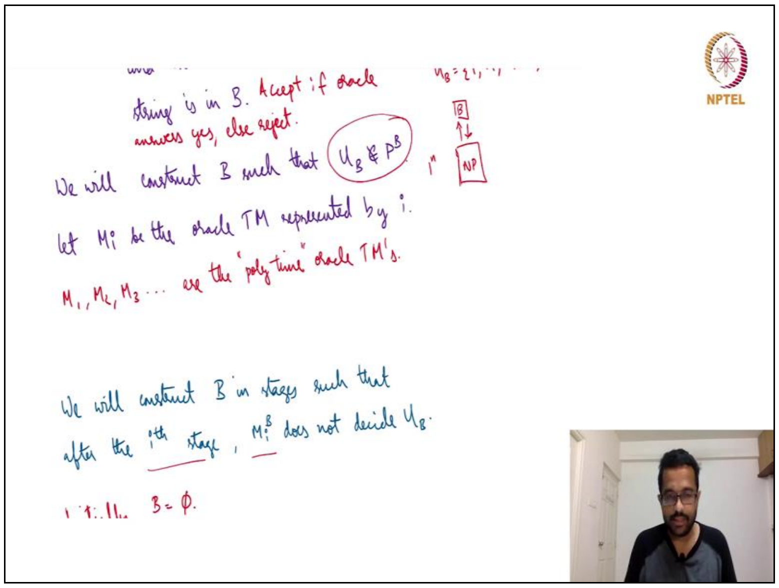
But what will we do then we will want we want a fool M i. So what we will do is we will exclude all the strings of length n from B. So, we will not put any string of length n into B all the strings of length n are excluded. So, whatever query if M i is accepting 1 power n then we will not put any string of any length into B any string of length n into B. So, which means it is giving the wrong answer.

So, which means again we are saying that all the strings of length n are excluded from B. M i does not accept n, 1 power n if M i does not accept one power n which means it is saying that there is no string of length n into B then what we do is we include a string of length n into B. So, to make sure that we fool M i, so, we include one string of the length n. But what if the status of all the strings of length n are already settled well that cannot happen why not.

Recall that before stage i the longest string whose status had already been determined was of length L which was smaller than n which means that all the strings of length L had their status undecided, all the strings of length n. And during the course of this execution M i running on 1 power n the status of how many strings could have been determined. We run M i only for 2 power n divided by 10 steps only 2 power n divided by 10 steps.

So in the worst case it makes 2 power n divided by 10 queries and there are 2 power n strings of length n and in the worst case we decide we fix the status of at most 2 power n divided by 10 strings of any length which means there should be many strings of length n whose status is undetermined. So, we include them into B. So, you again see what is happening here M i is not accepting 1 power n which means B should not have any string of string of any string of length n. But then we are kind of sneakily inserting a string of length and thereby making the answer of M i incorrect.

(Refer Slide Time: 23:`7)


string is in B. Accept if oracle answers yes, else reject.

We will construct B such that $U_B \notin P^B$

Let M_i be the oracle TM represented by i.

$M_1, M_2, M_3 \ldots$ are the "poly-time" oracle TM's.

We will construct B in stages such that after the i^{th} stage, M_i^B does not decide U_B.

Initially $B = \phi$.

So, just to summarize just to just to recap M i B always gives the wrong answer of on one power n why because suppose one power n or suppose there is a string of length B in the language or we suppose M i accepts one power n right then we will not put any string into that in into B of that length. So, therefore the acceptance of one power n is not consistent with the language u B because u B will not have one power n because no string of length n is in B.

And we can do that because whenever a string is queried if the status is not known then the answer we will exclude that from B. So, which means that all the strings of length n are still not in included in B and suppose M i does not accept one power n then we sneakily insert a string of length one power n. So, again if it was a decider for u B if it does not accept one power n that means none of the strings of length one power n should have been there in B but then we end up inserting one string of length n.

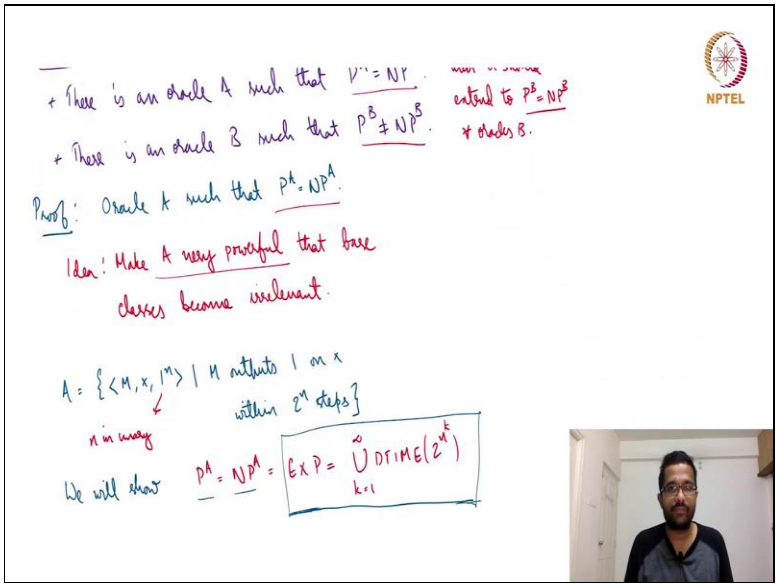
So, that this is not a again we make sure it is not a decider of u B. So, therefore M i B always gives the M i to the B always gives the wrong answer on 1 power n wrong answer meaning different from a decider of u B and at no point do we change the status of any of a string that whose status has been decided. So, we never modified putting something from outside to inside or inside to outside.

So this means that M i B does not decide u B. So, therefore u B is not in polynomial time or polynomial time machine to the with the access to the oracle B again. So, I know it has it can be a bit confusing. So, I will say it again u B is a string of is a set of all lengths 1 power n where there is a string of length n and P. So, it is just u B contains the lengths for which a string of that length is in B but the length is not written in binary but in unary.

First u B is in NP power B because NP to the P because given the length the NP machine can guess a certain string and query the oracle B for any B for any oracle. Second to show that it is not in P to the B what we do is we do it kind of diagonalization. So, essentially what we are doing here is diagonalization. We construct B in such a way that each machine is kind of is pulled.

So, what is the response of m i B we try to understand and then we make sure that we flip the output and there are enough strings in B for each of each length that allows us to flip the output because there are many non-committed strings.

(Refer Slide Time: 27:04)



+ There is an oracle A such that $P^A = NP^A$.

extend to $P^B = NP^B$

+ There is an oracle B such that $P^B \neq NP^B$.

$\forall$ oracles B.

Proof: Oracle A such that $P^A = NP^A$.

Idea: Make A very powerful that base 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 $P^A = NP^A = \boxed{EXP = \bigcup_{k=1}^{\infty} DTIME(2^{n^k})}$

So, at stage i we will rule out we will make sure to full M i. So, let L be the longest string whose status has been determined and then we choose n to be bigger than L meaning all strings any length L a length n the status is not determined and we run M i B for 2 power n divided by 10 steps which is bigger than any polynomial. So, the status is not decided then we answer no and if it is a. So, if M i is a polynomial time machine then it will certainly say yes or no and we will make sure that the answer is incorrect.

If it says yes then what we will do is on this on the link on the string in 1 power n then what we will do is to make sure that we do not put any string of length n into B if it does not accept 1 power n then what we will do is we will find a string that was not queried and insert into B. So, therefore making M i B and answer always different from that of a decider of u B. Hence M i to the B does not decide u B for any i.

So, this is what is happening in this proof hence. So, since all the turing machines or even all the oracle turing machines can be enumerated we can therefore conclude that there is no polynomial time machine with access to oracle B that can decide u B. So, maybe one thing that you can think through is why does this where have we used the fact that it is deterministic polynomial time in this proof.

If it was non-deterministic polynomial time would the same proof not work; where do we use the fact that there is only determinism or deterministic polynomial time. This is something that you can think about just to recap I have already recapped. We constructed an oracle B such that P to the oracle B is not equal to NP to the oracle B. Already recapped and summarized the proof.

But then if it is confusing feel free to go through it again and of course use the forums, use the discord and please ask questions if there is any confusion and perhaps we can also discuss it in one of the live sessions. And this completes the proof of the Baker Gill Solvay theorem where we show that no proof that relativizes more specifically a diagonalization based proof cannot be used to resolve P versus NP problem. And with that I will conclude lecture 26, thank you.