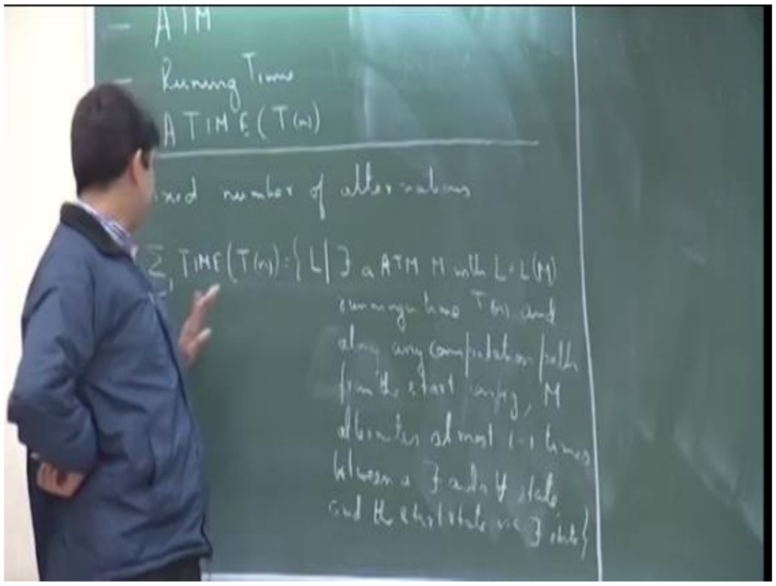
Computational Complexity Theory
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture -12
Introduction

(Refer Slide Time: 00:27)



– ATM
– Running Time
ATIME(T(n))
…xed number of alternations
Σ_i TIME(T(n)) = { L | ∃ a ATM M with L = L(M) running time T(n) and along any computation path from the start config, M alternates at most i-1 times between a ∃ and ∀ state and the start state is ∃ state }

So, Last time we were looking at alternating turing machines. So, we saw the following. So, we saw what an alternating turing machine is. Basically, every state is associated with the particular quantifier whether existential or a universal. And depending on what quantifier it is you decide whether to accept x or not. So, if it is an existential quantifier then at least one of the choices should lead to an accepting state.

If it is a universal quantifier then both the choices from that state should lead to an accepting state. We also saw how to define running time of alternating turing machines. So, I mean it is the time taken. I mean it is the maximum time taken over all input and overall computation parts. And it is looked upon as a function of the input length. And based on the running time we define the class A TIME T of n.

So, why were we studying this? I mean, what was the motivation behind looking at these special types of turing machines? So, what I said last time if you recall was that we wanted a different characterization of the polynomial hierarchy. So, the way we defined polynomial hierarchy was based on these certificates. And we wanted a machine based definition. So, that was the motivation for looking at these machines.

So, let us carry on and see how polynomial hierarchy can be studied using these machines. So, we look at a special subclass of these alternating turing machines, basically those alternating turing machines which have a fixed number of alternations. What I mean by that is if you look at the alternating turing machine so any computation path can alternate between an existential and a universal quantifier.

But suppose if I restrict that suppose if I say that you can alternate but let us say you can only alternate so many times. So, what does that give us? So, let us define a class based on restricted number of alternations. So, we denote sigma i time. Suppose you are setting a limitation let us say 10. Maybe the longest path only has 2 but maybe there is a shorter path but which has more frequent alternations.

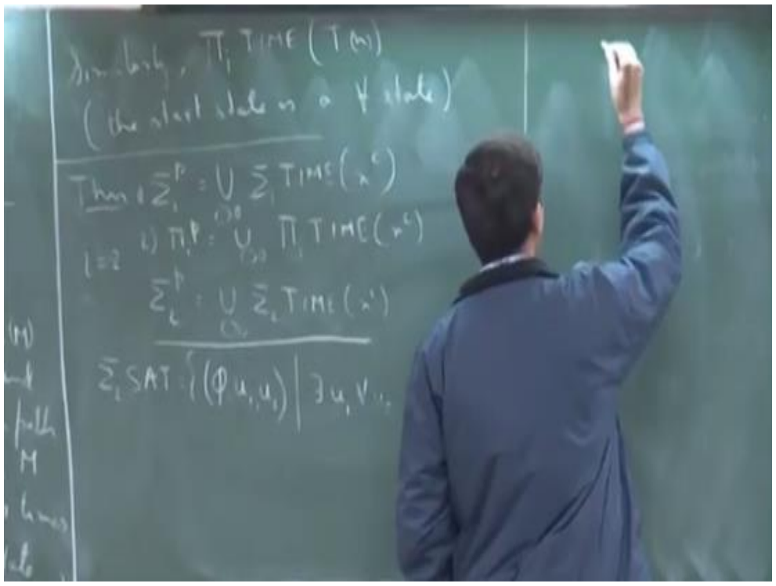
But no matter which part you pick the number of alternations should not increase this upper bound of 10. So, that will become more clear from this definition. So, sigma i TIME T of n is the class of all languages L such that there exist an alternating turing machine M with L being equal to the language of M running in time T of n. And along any computation path from the start configuration M alternates at most i – 1, times between there exist and for all state.

So, one more thing, so it must begin with a for all. And the start state is a for all state. I am sorry is a there exist state. This is sigma i time. So, yes, it is longish definition but nothing very complicated. So, all we are saying is that it is an alternating turing machine which runs in time T m and the start state is a there exists state. And along any computation path there are at most i minus 1 many alternations.

So, there can be may be several there exist along a path but then it switches to a for all. And then may be another there exist. So, the number of such switches should not be more than i minus 1 along any point. So, if you so then if you look at this definition what does sigma 1 time T of n mean? So, yes, it is not exactly NP because we are not saying what the function T of n is. So, if T of n were to be union of for all polynomials then many of you who said NP you are right.

But you have to be careful. I mean since T of n can be any arbitrary function if I fix i to be 1 then this gives the class N TIME T of n. Because I am starting with a there exist state and basically all the states are there exist. I cannot switch to any form.

(Refer Slide Time: 08:15)


Similarly, Π_i TIME $(T(n))$
(the start state is a $\forall$ state)
Thm: $\Sigma_i^p = \bigcup_{c>0} \Sigma_i \text{TIME}(n^c)$
i) $\Pi_i^p = \bigcup_{c>0} \Pi_i \text{TIME}(n^c)$
i=2
$\Sigma_2^p = \bigcup_{c>0} \Sigma_2 \text{TIME}(n^c)$
Σ_2SAT $\{(\phi u_1 u_2) \mid \exists u_1 \forall \ldots$

So, similarly we can define pi i TIME T of n. So, the entire definition is the same. The only difference is that the start state is a for all state. So, I will just mention that the start state is a for all state. Yes, so now we are ready to prove the first equivalence. So, sigma i p is equal to what do you think it should be equal to in terms of these classes. So, sigma i TIME of n to the power i. n to the power c. Yes, so basically look at all machines which have a polynomial running time.

And they have at most i alternations and they begin with a there exists state. So, I would not prove this. So, basically, I will give you the idea as to how. So, let us look at the case when i = 2. So, when i = 2 this means that sigma 2 p is union of c greater than 0. Sigma two times n to the power c. So, let us just look at the idea as to how this proof goes about and basically you can generalize it. And you can also prove the other theorem.

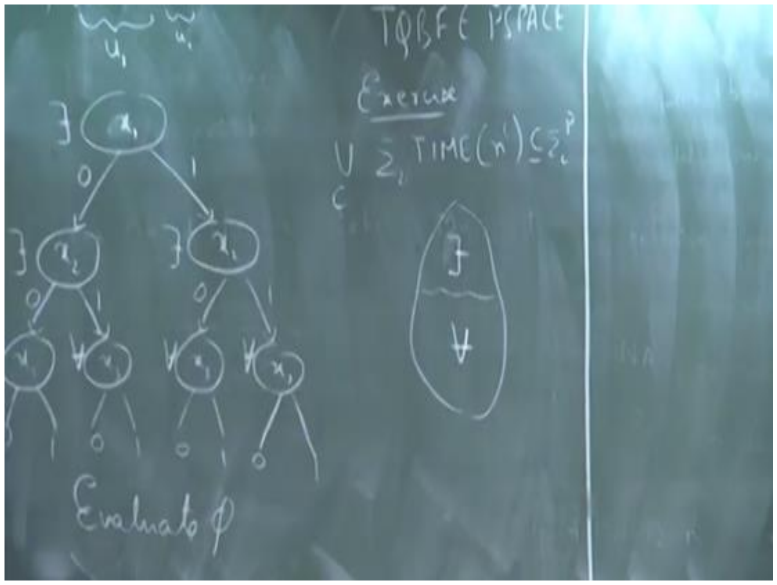
So, the other theorem is that so this is part one, the second part is pi i p is union of c greater than zero pi i TIME n to the power c. So, anything else? So, how do we prove this equality? So, let us look at the complete problem that we know for sigma 2. So, we know that this language sigma 2 SAT is complete for sigma 2 p. And what is the definition of sigma 2 SAT? So, what is the definition of SAT? So, basically SAT is you are given a boolean formula phi and you check is there an assignment to all the variables which makes it satisfiable. So, sigma 2 SAT is nothing but you divide the set of variables into two sets.

Let us say u 1 and u 2. And then you ask the question that is there an assignment is there some assignment to the variables in u 1 such that for all possible assignment to the variables in u 2 phi minus p satisfiable. So, this is, let me just write this again. So, it is all tuples of the form phi u 1 and u 2. So, u 1 and u 2 are two vectors of variables such that there exists some assignment to the values of the variables in u 1 such that for all possible assignment of values to variables in u 2 phi of u 1 comma u 2 is equal to 1.

So, this first sigma 2 SAT. So, how can we show that sigma 2 SAT is in this class? Because if we can show that then we have showed one direction of this equality. So, we have a machine which has this form. Basically, we have an alternating turing machine which can make one alternation. So, it can start with certain there exists states and then it can move to some for all states. And depending on what it does it can either go to an accept state or a reject state.

So, what do you think we should do if we are given an input of this form? So, basically that is the most natural way to proceed. So, what you do is what is suggested? So, look at the variables that are present in u 1. So, you start at your start state and you start guessing values for the variables and u 1. So, let us look at a small example.

(Refer Slide Time: 14:29)



So, maybe we have a formula phi on three variables x 1, x 2, and x 3. And let us say that these two variables are belonging to the vector u 1 and this belongs to u 2. So, they should exist some values to x 1 and x 2 such that for all possible value to x 3, phi should be satisfiable. So, what you do is the following. So, you start at your start configuration and then you guess a value for x 1 using a there exist state.

So, there can be two possible guesses. So, you can either guess a 0 or you can guess a 1. So, that is the non deterministic choice that you make. And you assign this to be a there exist state. So, what we are doing here is we are constructing a machine that will belong in this class. Again, I do the same thing but now for x 2. So, I guess a value for the variables x 2. And again, I do this with there exist state and it can be the 0, 1. And now I do the same thing for x 3. But this time I use it for all state.

And I will get some values 0, 1, 0, 1 and 0, 1, 0, 1. So, now I need to designate which are my accepting states and which are not. So, what would be my accepting states? Basically, I need to just club them. So, if x 1 is 1 and x 2 is 1. So, these two will be my accepting states. Phi is the input that is given to us. Yes, so the accepting states will basically depend on the formula what the formula is.

Because so this will give you a setting of the values. So, now you just evaluate the formula phi on these values and then depending on whether it accepts or reject you go to an accept state or a reject state. So, at this point you evaluate phi and you accept or reject. So, I am just I give a sketch of the argument this can be generalized. How do we show the other way round? Suppose you have an alternating turing machine which runs in polynomial time with one alternation. So, yes, you have to be little bit careful but essentially that is the thing.

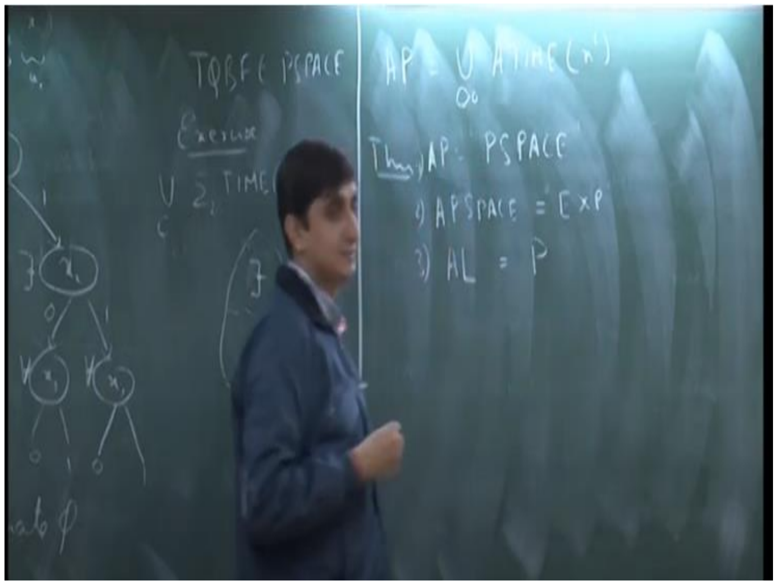
So, what you do is you look at the configuration graph. You look at so you let say you take a language in this class and as you suggested you look at any string x and you want to decide whether x belongs to that language or not. So, look at the configuration graph of that instance x with respect to the machine. And now I want to decide basically reachability if the final c accept is reachable from c start.

So, what I do is something similar to what we did for. So, when we showed that TQBF is in P SPACE. So, recall what we did. So, what we did was we for each possible assignment of the values to a variable whether 0 or 1. We recursively did the computation and then we were basically reusing that space. So, we basically again do the I mean the same kind of computation here and that will be a sigma 2p computation.

Yes, you need to come up with certainly. Actually, you need to come up with a formula also. So, you need to come up with a formula and then you need to specify which what the certificates would be. How do you construct the formula and the certificates from the graph? I was trying to argue that this reachability problem can be reduced to this sigma 2 SAT but it is basically the same thing.

I mean, you can also give a machine with two certificates that will also work. So, what can happen is I mean, you can have the graph. Suppose this is your graph. So, maybe up to this level you have all there exist configurations and here you have all for all configurations. So, a certificate will basically depend on these vertices and these vertices. So, just think about it. So, any other questions. So, let us move on. So, now what if we relax this condition if we consider arbitrary number of alternations?

(Refer Slide Time: 22:00)



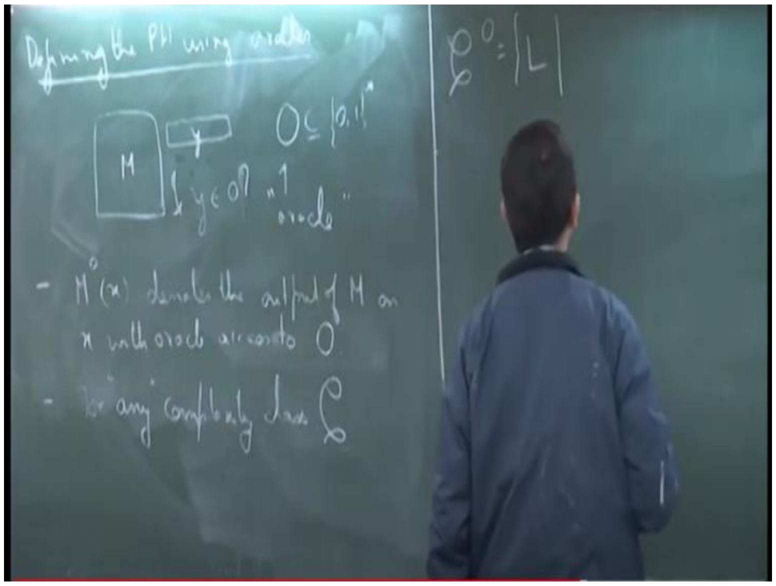
So, we define the class AP to be union of c greater than 0 A TIME n to the power c. Basically all alternating turing machines which have a polynomial running time. So, we do not have any restriction as to what the initial state would be or how many alternations it will have. So, it is a generalization. And what is known is AP is equal to P SPACE. So, at initial glance this looks quite surprising. I mean it does not look very surprising but it looks nice.

But if you just think about it a little bit the proof is not very difficult. Actually, the proof is exactly the same as the proof of this theorem. It is essentially the same proof. But it is quite nice that you have some polynomial turing machine I mean, although it is a very powerful kind of a turing machine but still it has polynomial running time. But it does characterize the class P SPACE. And you also have other results.

So, this is the first in the series. It is also known that AP SPACE. So, these are alternating turing machines which have polynomial space bound. So, this is equal to the class. Exp. Again, the proofs are very similar. And similarly, even in the log space domain if you look at the class A L that is alternating turing machines which have only log space what do you think this should be equal to? What is your guess?

That is equal to the class P. So, that is another way of characterizing P. So, all those languages which are accepted by log space bounded alternating turing machines. So, there is also a third way of describing the polynomial hierarchy. So, we saw a certificate based definition, we saw a machine based definition. There is also another machine based definition, but it is little different.

(Refer Slide Time: 24:58)



Defining the PH using oracles
M
y
O ⊆ {0,1}*
Is y ∈ O? "1 oracle"
- M^O(x) denotes the output of M on x with oracle access to O
- for any complexity class C
C^O = {L|

So, it is based on the use of oracles. So, recall what oracle turing machines are. So, what are oracle turing machines? So, basically you have a regular turing machine and it is fitted with some an extra oracle tape and it has as you said three special states. So, one is the state query and there is a state q yes and there is a state q no. So, the idea is that at any point during the computation of this machine M, it can write a string on to the oracle.

So, let us say string y on to the oracle. Now, there is a designated language with respect to machine. So, let us say we have a language O. And this language is called the O. So, now after the machine writes y on to its tape it moves on to the state q query and that is when the oracle checks does y belong to this language or not? So, it just checks that in one step and if it belongs to the language it returns it moves on to the state q yes and if it does not it moves on to the state q no.

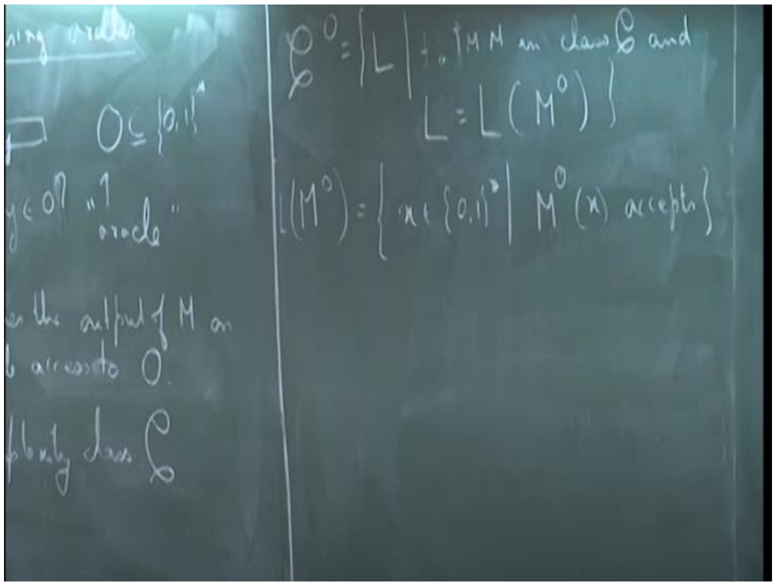
So, the essential idea is that in one step you are able to perform the entire computation of this language O. So, you are bypassing the entire computation and you are able to answer whether a given string belongs to that language or not in just one step. So, is this clear? Is this model clear to everybody? Any questions? That will be considered within the running time of M. So, it I mean it can be several things.

Now maybe it writes one bit of y. But it does not enter the state q query. It then goes and does some other computation within M maybe then it gets the second bit it writes on to the oracle tape and so on. So, finally when it finishes writing out the entire string y then M says that now I am done writing the entire string y and I want to check if y belongs to O or not. So, this step is y in O. So, this question can be answered in just one step of the machine M.

So, if you are using space bounded machine and you want to use a long enough oracle tape, then you have to make it a write only tape. But I am just assuming that it is a time bounded turing machine in which case it does not matter. So, depends on the definition I mean. But that is a good point. So, some quick notations about these things. So, M O of x denotes the output of machine M on x with oracle access to some language O.

That is what M 1 x denotes. And similarly if we have, so we can define for any complexity class. So, this any is I will just put it within quotes because there are certain complexity class for which this does not hold. But at least for our purposes we can assume it is for any complexity class C.

(Refer Slide Time: 30:21)



$O \subseteq \{0,1\}^*$

$y \in O?$ "1 oracle"

the output of M on
access to O.

plexity class $\mathcal{C}$

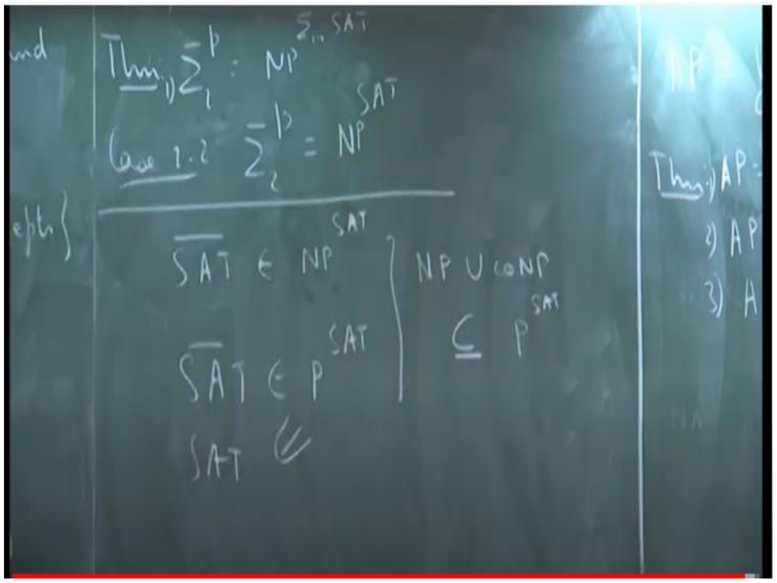
$\mathcal{C}^O = \{L \mid \exists$ TM M in class $\mathcal{C}$ and $L = L(M^O)\}$

$L(M^O) = \{x \in \{0,1\}^* \mid M^O(x)$ accepts$\}$

C to the O is the class of all languages L such that there exists a turing machine M such that L prime is equal to the language of this machine and L is equal to L of. I can just let us see. And L is equal to L of M with oracle access to O. So, this I did not define here, but basically this means all the strings that are accepted by oracle access to O. So, I defined what M of x means. So, for all strings which are accepted by oracle access to O that denotes the language L.

Maybe I can just write it also to be complete. So, M the language of M with oracle axis to O is the set of all strings such that M O of x accepts. So, now again with respect to these oracle turing machines we have a very nice characterization of the hierarchy.

(Refer Slide Time: 33:04)



Thm: 1) $\Sigma_i^p = NP^{\Sigma_{i-1}SAT}$
Coro 1.2 $\Sigma_2^p = NP^{SAT}$
$\overline{SAT} \in NP^{SAT}$
$\overline{SAT} \in P^{SAT}$
$SAT \in$
$NP \cup coNP \subseteq P^{SAT}$

So, the theorem is that sigma i p is equal to NP to the sigma i-1 t. So, this is you find it kind of a recursive manner. Again, in a similar way pi i p is co NP to the pi i-1. Actually, this does not matter. So, we will see that in a moment. But we can just define it this way. So, what do we mean by this class over here? This class. So, maybe just change this a little bit for the time being. So, let me write it as NP to the sigma i minus 1 SAT.

Because this is more consistent with the way we are defining oracle. So, later on we will see that why we can write it in the earlier form that I mentioned. I just erase this. So, what does that mean? So, it means that I have a base machine which runs in NP and it makes oracle queries to a sigma i minus 1 SAT language. So, if I look at the case when i is equal to 2 this means that sigma 2p is what?

So, what sigma 1 SAT. Sigma 1 SAT is just SAT. So, it is just NP to the power SAT. So, yes, so again, I mean the general thing can be proved very easily once we see what is the proof of this statement. This is actually one direction is quite easy, the other direction is a little bit nontrivial. But before I proceed to the proof of this theorem let us look at a simpler statement. How do you show that SAT is?

So, this is very trivial but let us say how do you show that SAT bar is contained in NP to the SAT. So, first of all, we do not know whether SAT bar is contained in NP or not. Because SAT bar is a complete problem for co NP and we do not know whether these two classes are the same. But what I am claiming here is that we can show that SAT bar belongs to this class. How do you prove this?

So, what is I mean, how do you show? So, basically what are you given? You are given some formula phi and you have to accept if this is not satisfiable and you reject if this is satisfiable. So, what do you do? That is very easy. It is just a one line algorithm. Yes, so basically if you have oracle access to SAT what you can do is you just copy this on to the oracle tape and you ask does this belong to SAT or not. It will say yes or no.

If it says yes then you output no if it says no then you output yes. So, in fact the base machine is not even doing anything. The base machine is just copying its input to the oracle tape and whatever answer it is getting its transferring it to again the base machine. So, it is not even as difficult as NP SAT. What we just showed is so as somebody suggested that SAT bar is in just a class P to the SAT. So, it is in a much smaller complexity class.

And not only SAT bar the same argument also shows that SAT is in this class. And since these two languages are complete for their respective classes what this implies is that both NP and co NP are so I will just write it as NP union co NP they are contained in P to the SAT. And hence of course NP to the SAT. So, this is just to illustrate as to how the oracle is getting used. And we know that this is not sufficient because when we first looked at this class, we saw that this is actually larger than the class co NP also.

So, co NP does is contained inside this class. So, we have to do a little bit we have to spend some more effort to show that this is contained in NP to the SAT. So, I am running out of time. So, I will just leave the proof here. You guys can think about it. But next time we will complete this proof. And that will also end our discussion of polynomial hierarchy. So, we will move on to other topics.