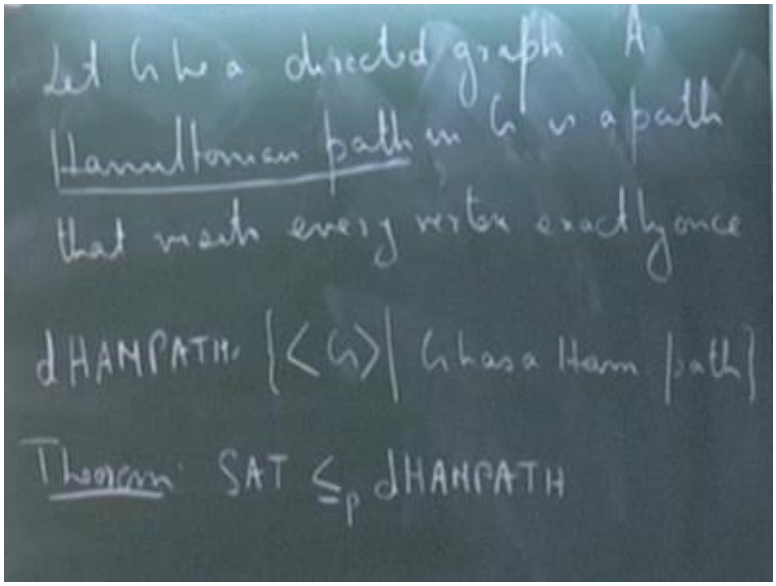
Computational Complexity Theory
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture -04
Introduction

So let us get started. So last time, we saw the reduction we saw the complete reduction as to why SAT is NP complete and we also saw how SAT can be reduced into three SAT. So let us look at one more NP complete problem. So this is a this problem is related to graph theory and will show that the problem of checking whether the graph or whether an indirect or let say directed graph has a Hamiltonian path or not is NP complete. So how many of you here know what a Hamiltonian path is? So let me just define it.

(Refer Slide Time: 01:19)



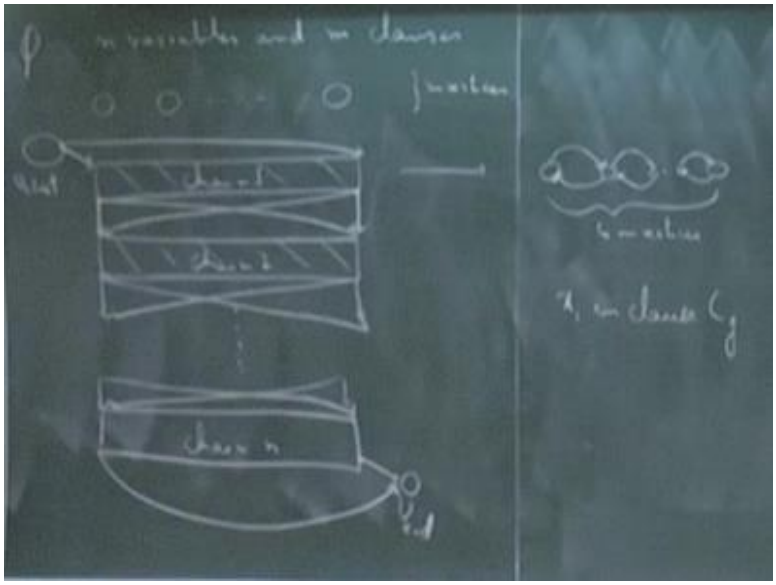
So let G be a directed graph, a Hamiltonian path in G is a path that visits every vertex exactly once. So in other words, so when I say path in a directed graph, it means a directed graph and since it visits every vertex exactly once. What it means is that the length of the path is you count the number of edges, it is n - 1. So the language associated with this notion is, let us define the language as d HAMPATH.

So d HAMPATH consists of encodings of all graphs, all directed graphs G. So for the purpose of this problem whenever I am referring to graph, I will mean a directed graph. Such that G has a Hamiltonian path, just d stands for the fact that G is a directed graph and I am just defining a language d HAMPATH. So it is just the name of this language, I mean you can also define under a Hamiltonian path for undirected graph.

And, even there it is NP complete but we will look at the directed version in this class. What will show is SAT reduces to this problem. So, this is problem in NP first of all. How many of you think it is in NP? How many of you think it is not? So, why do you think it is in NP? So, you just guess a sequence of n vertices and now you check that does that sequence form a path or not? That can be checked in polynomial.

So that is a polynomial length certificate and a polynomial time algorithm for verifying that d HAMPATH in it. So therefore this reduction would show that d HAMPATH is NP complete as well. So I will give a sketch of the proof and skip some of the details that you can fill out later on. So the idea of the construction is as follows. Suppose you are given a formula phi;

(Refer Slide Time: 05:10)



We will construct a graph as follows. So let us say that phi has n variables and m clauses. So our graph will look as follows. So for each of the m clauses will first add m vertices in this graph. So, let me draw pictures. So we have some m vertices and for each of the n variables, we have chain

of some n sub graphs. So for each variable here I will define a chain as follows. So we have a chain one corresponds to the first variable then we have a chain two.

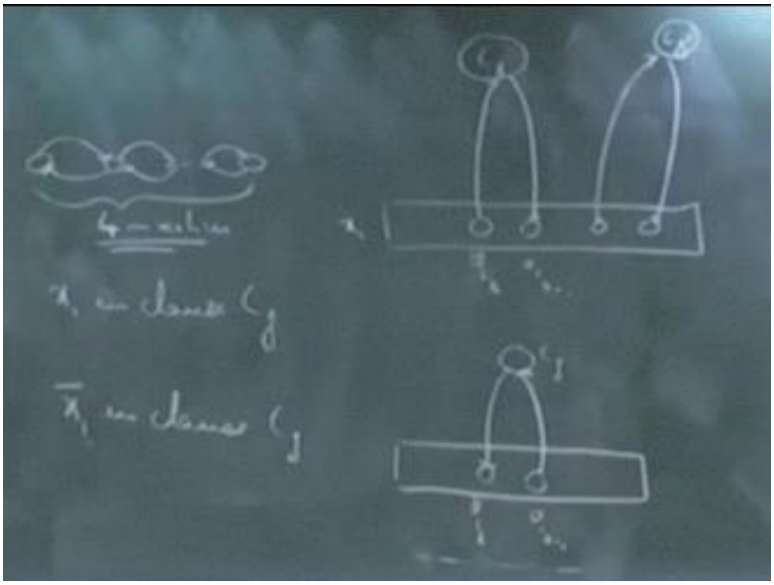
And what does each of these chains look like? So they have a simple structure, so each chain has four m vertices as follows. So it looks like this. So I take a vertex, I take a sequence of four m vertices and I add edges between them in this manner. So I add an edge between vertex and it is adjacent vertex in both directions. So this is what each of these chains look like and I have n sub chains.

Is it clear so far? So how do I connect these chains? So, what I do is that from the first and the last vertex of each chain, I connect the first and the last vertex of the next chain. So from here I put an edge to the first vertex in chain two and to the last vertex in chain two and as well from the last vertex in chain one I put an edge to the last vertex in chain two and the first vertex in chain and so on. So this is a chain.

Similarly I do this, all the way up to chain n. So we have to change and in addition, I add some more edges, so I add two more vertices add as well there is a start vertex, let me call it V start and there is a end vertex, I will call it V end and I connect the first vertex of chain one from V star as well as the last vertex and the same thing I do for V end. So now what told as these n vertices say?

So now I have basically defined the graph for these chains and their corresponding vertices, what is the role of these m vertices appear? So if a vertex or here if a variable. Suppose we have a variable x i in clause let us say C j. what I do is I add two edges from that particular from the vertex representing the j th clause to the chain representing the i th vertex and how do I add it? So let me illustrate it over here.

(Refer Slide Time: 10:34)



So suppose if I have this vertex representing the j th clause and I have a chain representing the i th vertex, sorry the i th variable. I take two adjacent vertices in this chain. Let us say I call them Vi and let me not use i, so we use V ik and V ik+1 one and I connect them as follows. So if xi is present like this then I put a connection from V ik to C j and from C j 2 to V ik + 1 and if xi is present in it is negative form.

So if xi is present as xi not in cause C j what we do is. We just add these edges in the other direction. So I add an edge from V ik + 1 to C j and then from C j to V ik. So now there is one more thing about this construction that needs to be mentioned and that is all. So the question is that which of these two vertices do we pick? So one property that I said these two vertices must have is they should be adjacent vertices.

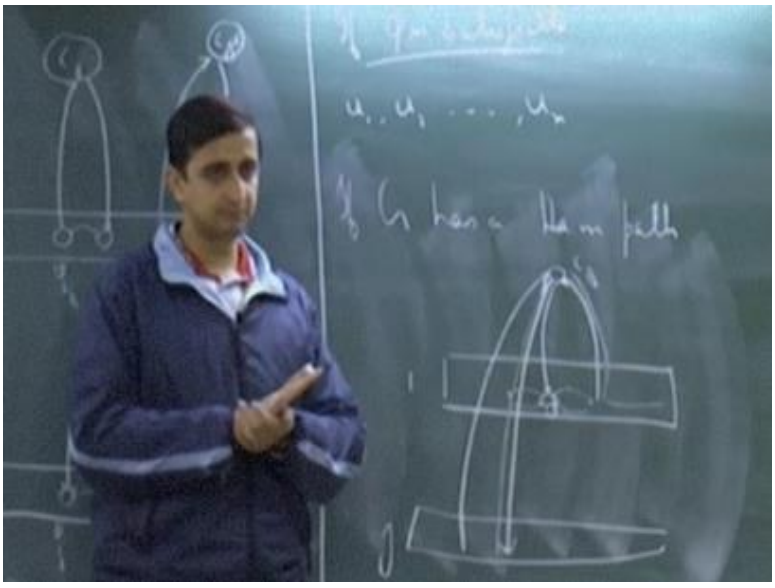
But, what is this value of k that we are fixing? So that value of k is some arbitrary value such that these two vertices are not part of some other clause. So that is the reason why we choose four m vertices. So we have this fourm vertices sitting over here. So these vertexes must be spaced out in such a manner so that there is always a gap of at least one such link between the edges going to let us say a clause j and some other clause j prime.

So maybe that there is also some other clause C j prime which uses the variable xi. So I will just make sure that these two vertices are not used to connect C j prime just some othervertices,

maybe something here and something here. Some other k prime, so that is all. Two m is sufficient actually, two m is sufficient because of vertex cannot be part of more than m clauses but then four m is just to be safe, I mean, just to make sure they are spaced out enough.

I mean, I think two m is also sufficient, but basically just some constant which is sufficient enough here. So now what is the claim? The claim is that if phi where to be satisfiable then there is a Hamiltonian path and otherwise not. So let us check that.

(Refer Slide Time: 14:33)



If phi is satisfiable what can we say that there is an assignment to the n variables of phi. So there is some assignment, let us call it u 1, u 2, so on up to u n of values to the n variables of phi which makes it satisfiable. So what we do is we construct a Hamiltonian path corresponding to this satisfying assignment as follows. So if a variable xi gets a value zero, we will think of traversing the chain from right to left for that variable.

And if a variable gets a value one, we just think of traversing that chain from left to right. So we start at v start. Suppose the chain one which corresponds to the first variable that gets a value, zero. So then what we do is that we traverse this chain 1 from right to left. We come to the head of chain one, then we look at change two and we see what value does it get? If it gets a value one which reverse it from left to right if it gets a value zero again, we take this edge come to the end of chain two and we traverse it from right to left.

And, we keep on doing this until and unless we reach the last chain and with then go and stop at V end. So basically this gives us a mechanism to traverse all the vertices from V star to V end and all the variables in the n chains but we have not traversed these m vertices. So, Hamiltonian path must visit every vertex exactly one. So the way we can visit these m vertices as well is because of the fact, so because of the way we defined these edges and the way we are traversing the chains.

So, If a variable got a value zero then note that, if the literal xi compliment got satisfied in some clause C j then we have these two edges connecting C j to the i th chain. So therefore if C j got satisfied by the i th variable using this using a literal which has this form. We just traverse it from right to left and when we reach V ik + 1, I just take these two edges come to V ik and then go ahead and if a clause got satisfied by some literal, which is in its non-negated form.

Which has xi then we know that with only weight could have got satisfied by xi is if xi got the value one and if xi got the value one, I am traversing it from left to right. So I can use a connection, like this to go from V ik to V ik + 1. So instead of using this edge and just using these two edges and the same thing is happening here instead of using this edge and using these two edges.

So the question is when a clause is getting satisfied we just check which is the variable that is responsible for it and in which form is it getting satisfied by a variable in it, it is non-negated firm or in its negative form. Just to traverse the suppose there is a variable which is not responsible for making a clause evaluate one or basically for going from one end of the chain to the other end of the chain that is where we use those things.

If the i th variable is responsible for satisfying the clause C j that is when we use that. So it is not necessary. So there can be many variables present in a clause, but we do not use all these jam pages corresponding to all those variables. Only use it for that particular variable or one particular variable, which is responsible for setting that clause true. So when we reach the first position, but see one variable can be responsible for setting many clauses to true but it can only have one particular value for that.

Let us say with the value zero or value one. So then we have many SAT jumps. So we let say it satisfies clause C j prime as well as C j. So when I come to this vertex, I jump to C j prime come back or here other direction. So when I come to C j, I jump to sorry when I come to V ik, I jump to C j and then come back. Again when I reach this vertex, I jump to C j prime and again come back. There by satisfying both the clauses and now the convers also not difficult to show.

So if this G has a Hamiltonian path, what you need to observe is that any Hamiltonian path in G must start at V start and end at V end. Because V start has in-degree zero and V end has out-degree zero and moreover if you look at the way these chains are designed you can only go from a chain i to a chain i + 1, there is no other way in which you can visit the chain. So any Hamiltonian path that you have it must start at chain one then go to chain two chain three, all the way up to chain n and then end.

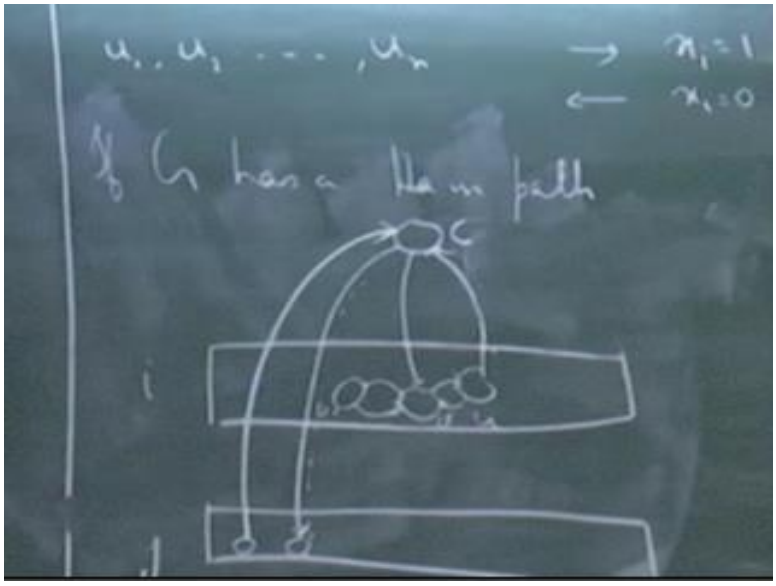
So what you do is if you take any Hamiltonian path and then in the middle of a traversing these chains there are some jumps to one of these vertices and down below. So what can happen is so there is only one problem case suppose you are at. Let say some chain i and you make a jump to variable corresponding to some clause C j, so they can be many variables contained in C j. So, let us say there is also another variable j.

We have these two, also since there is another variable j present i also have things like this, may be in the other direction. What if you go to C j using this edge and then instead of taking the next coming down to the i th chain, you take this edge and come down to the j th chain. What is the problem? Suppose if you take this, go to C jand take this edge and come down here, then just consider this vertex, let us call this vertex V, what happens to this vertex?

So there is no way in which you can your Hamiltonian path can visit V because what are the outgoing edges out of V, so there is one outgoing edge coming to this vertex and there is one outgoing edge coming to this vertex over here so and what are the incoming edges, so there is only one one incoming edge from here and one incoming edge from here. So the only way in which you can come back to V is by using this incoming edge.

But, then if you use this incoming edge you cannot go to this vertex or you cannot go to this vertex because both those vertices have already been visited by your path. So if you are climbing to some clause C j you must climb back down to the same level or to the same chain i. So, let me just draw this a more clearly.

(Refer Slide Time: 23:54)



So my claim is that if you are in some chain i and you climb to a clause. Let us call it is C from some vertex u. So, of course, you have some edge coming back here to a vertex V and you have edges like this, there is also another adjacent vertex sitting here; let us call this V prime. Now they can also be another chain j which corresponds to some variable x j which also belongs to C. So I wish I had another colored chock, but anyway.

So suppose after you come to you; so, if the Hamiltonian path that I am considering after it visits you it climbs to this clause C, I take this dotted path and then instead of climbing back down to the same chain it climbs back down to the j th chain, what is the problem with this kind of a Hamiltonian? The problem is that by definition the Hamiltonian path should visit every vertex exactly once.

So consider the point when this Hamiltonian path is coming to this vertex V. What are the ways in which the Hamiltonian path can come to V? Now note that it cannot come to V from u because u is already visited somewhere prior, only way it can come to V is from V prime. So this

is the only way in which the Hamiltonian path can come to V but if it comes to V from V prime then it has nowhere else to go.

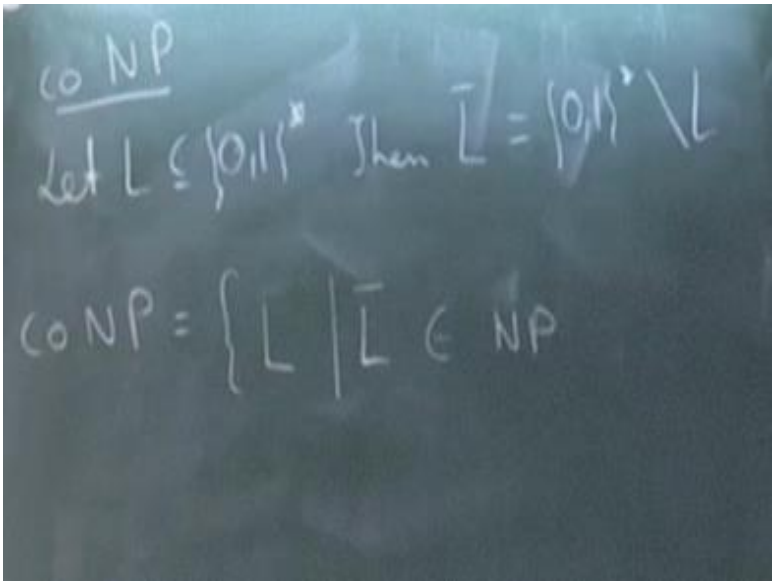
In particular, it cannot reach let us say V end because all the vertices that are connected to V have already been visited by that Hamiltonian path, so it gets stuck. So therefore if always must go in back down to the same level. Well, we will not do that. So basically what we do is so now so I did not complete the entire proof. Suppose there is a Hamiltonian path. So, of course since it is a Hamiltonian path, it should not visit C again.

By definition, it should visit every vertex exactly once. So what I do is that so now given a Hamiltonian path I will define an assignment which is satisfying assignment for phi as follows. If the Hamiltonian traverses a chain from left to right the i th chain, I will set xi is equal to one and if it SAT traverses a chain if it traverses from left to right and if it traverses a chain from right to left, then I will set xi as zero and my claim is that this is a satisfying assignment for fine.

So every clause will be visited by from some particular i, maybe that many clauses are visited from a particular i. But there will be some i which will from which there will be a jump to a clause and then back to the adjacent. Any other questions? So what I suggest is you go back and complete this proof. So the ideas are all present just make sure you write down the complete proof that we help you to verify the complete correctness.

So let us move on to a different concept. So we will define this complexity class known as Co NP.

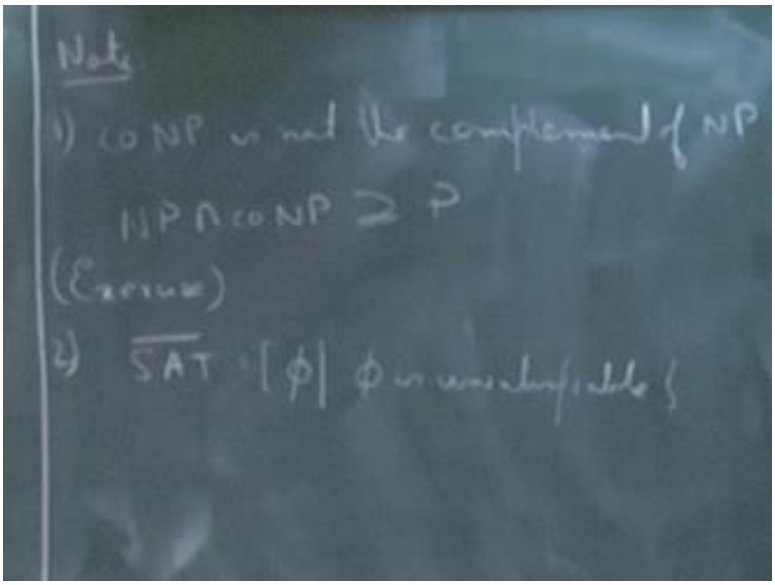
(Refer Slide Time: 28:22)



So what is Co NP? So first let me define what it means to talk of the complement of a language. So suppose you are given a Language L, then L complement, what is the definition of L complement? So sigma is zero one for us. So it is zero one star minus L all those strings which are not present in M. So, Co NP is defined as the set of all problems or the set of all languages whose complements are present in NP.

If you want to be a little bit carefully, I can just say that set of all languages L such that L complement is an N, both way of writing it is correct. So a couple of points regarding this definition; so the first thing is that;

(Refer Slide Time: 29:40)



So note that, Co NP is not the complement of NP. So sometimes this term Co can be a little bit misleading and so that is why I just wanted to emphasize this Co NP is not the complement of NP and as it turns out the intersection of NP and Co NP is not even is not NP as well. It contains a rich collection of classes in particular it contains the class P; we know that P is contained in NP; you can also show that p is contained in Co NP as well.

So this is can try this as an exercise; to show that NP intersection Co NP contains P. So the second point is, let us consider the complement of the language SAT. So, what is the complement of SAT? It is the set of all formulas phi, such that phi is unsatisfied. So is this the correct complement of this language; why not? Who said no? So what was SAT? What was the definition of SAT?

There exists, so it is all formulas phi, let us see in a form. Such that there is some satisfying assignment and now here I claim that it is SAT complement or SAT part is the collection of all those formulas which do not have any satisfying as m. So unsatisfiable basically means that there is no satisfying assignment. So is this the correct complement? Let us look at the definition the way I define the complement of a language here that is the point.

So I am so when I wrote down this definition, I was a little bit lazy and I kind of abused the notation that I defined earlier what if you have a string with does not encodes a formula. What

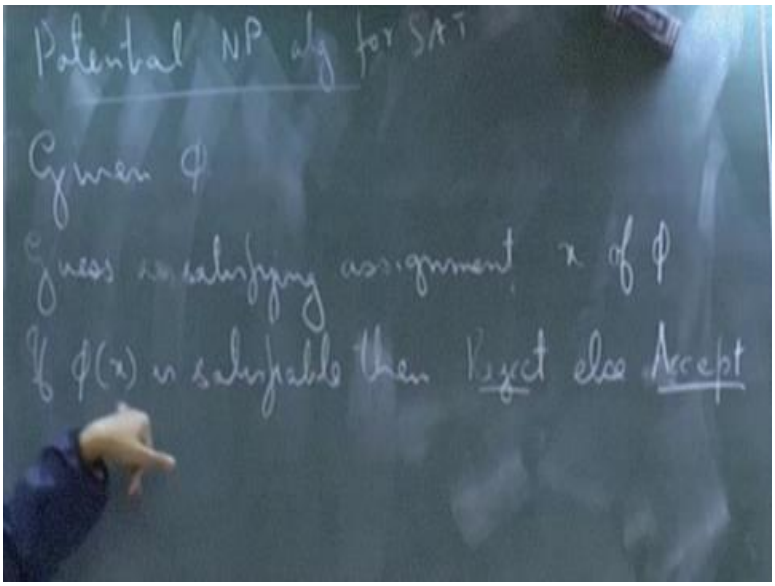
about those strings? So by definition SAT is encoding of all those strings which are satisfying formulas. So therefore SAT bars should not only comprise of all these kind of formulas but also those strings which are not let us say formulas in CNF form or some arbitrary string.

So generally, so this is a point that I want to make the beginning of this course. So generally when we look at complement of languages we are kind of abusing this notation and we only look at those strings which represent an instantiation of that problem, correct instantiation of that problem which gives an alternate answer sorry, in the earlier case satisfy. I mean all those formulas which belong to SAT they had a satisfying assignment.

And, here I am considering formulas which do not have any satisfying assignment and the reason why I can do that is because if you have given some other string, I can easily check that. I mean, I can easily run through the string and I can check whether it is a valid encoding of a formula or not and depending on thatI can just accept or reject. So, that is the reason why I can make such a definition. So just keep this point in mind.

So, coming back to this language, so now consider the following potential NP algorithm for SAT bar, we know that SAT is in NP. Let us try to prove that let us try to see if we can come up with an NP algorithm for SAT bar as well.

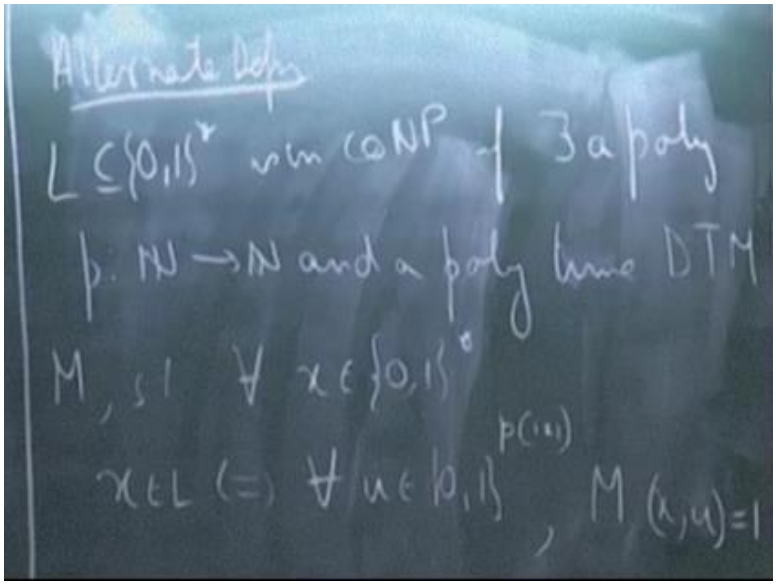
(Refer Slide Time: 34:36)



So, potential NP algorithm for; SAT bar. So what we do is given a formula phi, Guess and satisfying assignment x of phi; you just to make a, construct a string x comprising of assignments to the variables of phi and if now plug in those values on to phi and then if phi of x is satisfiable then reject else accept. So this algorithm is clearly guessing a string which has polynomial length and this string is satisfiable or not that of that can also be carried out in polynomial time.

So is this a correct algorithm for this language SAT bar. So, therefore this is a real incorrect algorithm. So this is again a common mistake that people tend to make that is by wanted to point this out. So when you so when we say that language is in Co NP, what it means is that for all possible certificates. So not only for one certificate, for all possible certificate there is a polynomial time Turing machine which does which accepts or rejects how however way you want to define it.

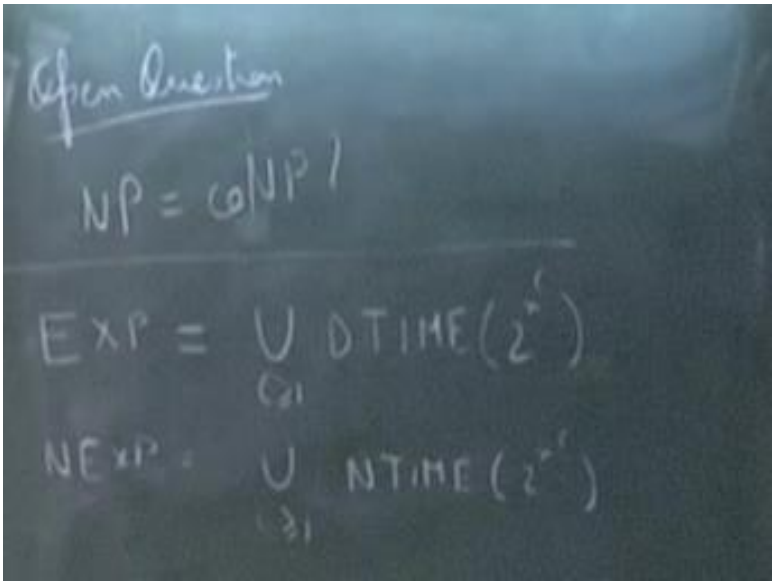
(Refer Slide Time: 37:32)



So let me write that, so we say that a language L is in Co NP, If there exists a polynomial P and a polynomial time deterministic Turing machine M such that for all x if x is in L; when x is in L if and only if all possible certificates u of length, P of x, M given x, u would accept. You can verify that these two definitions are the same. So, therefore to show; that a particular formula is unsatisfiable.

You need to basically check for all assignments x whether M on x, u satisfies or not. It does not suffice to check it just for one particular assignment. So this kind of motivates why this question is so interesting.

(Refer Slide Time: 39:44)

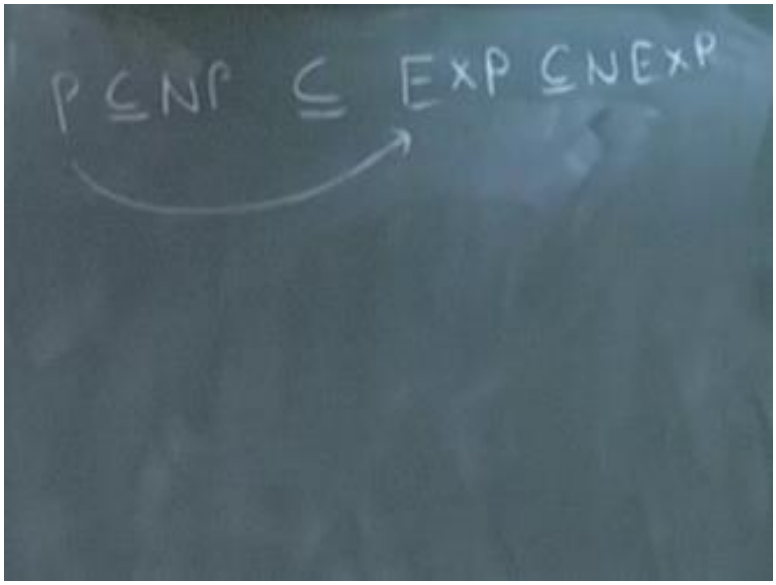


So this is also an open problem, whether NP is equal to Co NP or not. Basically resolving this question amongs to coming up with a polynomial learn certificate for SAT bar. Can I come up with one particular SAT certificate which is able to verify that a formula is unsatisfied or not and that kind of if you just think about it for a while you will realize why it is so difficult because there are exponentially so many possible formulas and it is not easy to come up with such a logic.

In fact, it is believed that it is the other way also. It is the general belief is that these two classes are probably not the same. So, let me define some more things and then we will stop. So, we can define an exponential time classes in a similar manner. So EXP is the class of all languages which are the union of the languages contained in D TIME, 2 to the power n to the power c. So recall that polynomial time the class P was defined as the union of all the languages which are the union of D times n to the power c.

Here we have the exponential term and similarly N EXP is union of all languages in NTIME 2 to the power n to the power c. So what are the known relations between these classes?

(Refer Slide Time: 42:07)



So by definition, we know that P is contained in NP and again by the definition of EXP and NEXP, we know that EXP is contained in NEXP as well. Also from the definition we know that P is contained in EXP as well. So we can given EXP algorithm for SAT as follows, just run through all possible strings of length n. Suppose, it says SAT on it is a formula on n variables so just run through all possible assignment of values to those n variables.

There are 2 to the power n many of them and keep checking. So, if you find even one such string which makes the formula satisfiable then accept otherwise reject. So that is the deterministic algorithm in exponential time which shows that NP is content in EXP as well. So we will stop here today next time we will see a little bit more about these classes.