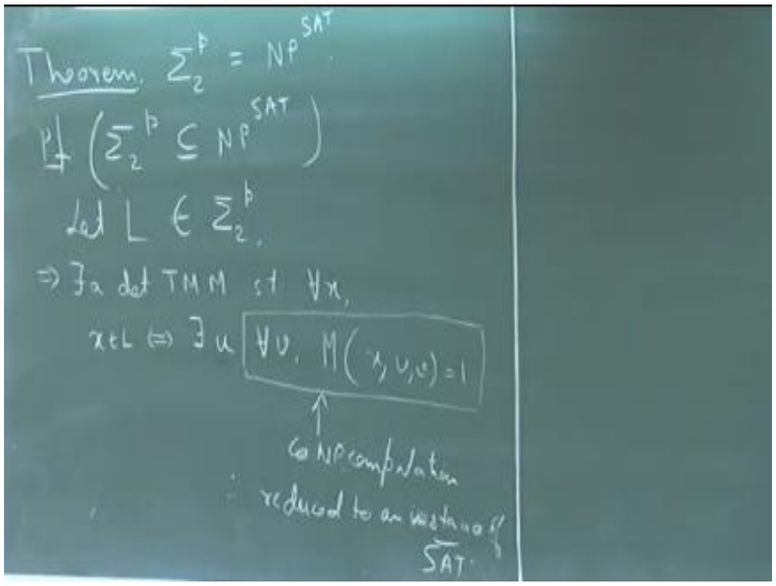
Computational Complexity Theory
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture -13
Introduction

So, we were looking at various ways of defining the polynomial hierarchy. So, one of the ways that we saw was we can look at a turing machine model as a way of defining the polynomial hierarchy. And we also ended by noting that we can get an oracle-based definition again for defining the polynomial hierarchy. So, let us look at the proof of that approach. So, that is where I think we ended last time.

(Refer Slide Time: 00:50)



Theorem. $\Sigma_2^p = NP^{SAT}$.

Pf ($\Sigma_2^p \subseteq NP^{SAT}$)

Let $L \in \Sigma_2^p$.

$\Rightarrow \exists$ a det TM M s.t. $\forall x$,

$x \in L \Leftrightarrow \exists u \boxed{\forall v.\ M(x,u,v)=1}$

$\uparrow$ co NP computation

$\therefore$ reduced to an instance of $\overline{SAT}$.

So, what I am going to prove today is that sigma 2 p is NP to the SAT and I will just leave the generalization to you. You can see that this can be generalized for any i for any i sigma i p is NP to the complete problem for sigma i minus 1 p which is sigma i minus 1 SAT in particular. So, is everybody clear about the oracle model? What does it mean to say when we have an oracle language with respect to a particular class?

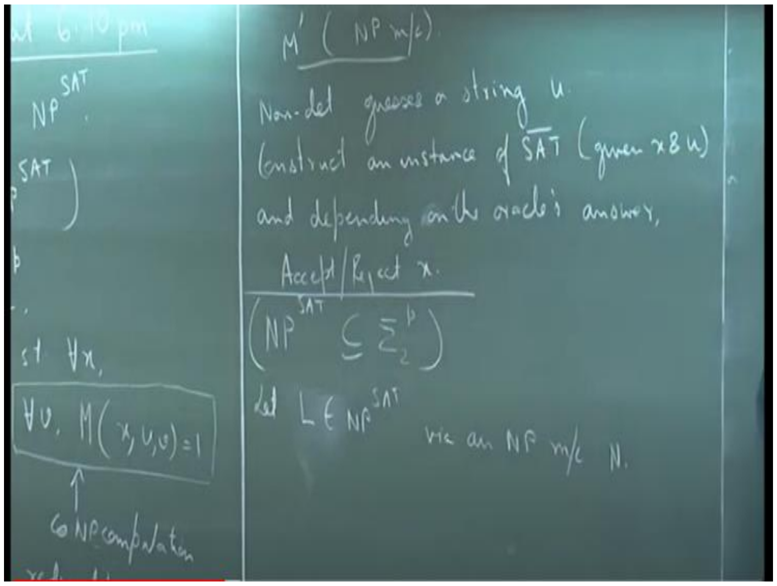
So, let us look at the proof of this. So, first let us show that sigma 2 p is contained in NP to the SAT. So, this is what we will show first. So, let L belong to so let L be a language in sigma 2 p. So, what does that mean? It means that there exist a deterministic turing machine M such that for all x, x belongs to L if and only if there exists some string u such that for all v M given x, u and v accepts.

So, I just so when I say there exist u and v I mean that polynomially bounded strings u and v. So, what does this statement mean? I mean so the goal is to show that such a computation can be simulated using an NP machine with an oracle query to a SAT instance. So, how do we show that? So, well note the following thing. So, let us note just this part of the computation. So, suppose of course x is provided to us and suppose somebody also provides us with a u.

So, let us assume that we are provided with an x and with a u then what does this mean I mean what is happening over here? What kind of a computation is this? Well not I mean you can reduce it to SAT bar. I mean L is any arbitrary language but as I suggested this is basically a CO NP computation. You are checking the for all strings v is this machine given x u and v equal to 1. So, in essence this is a CO NP computation and you can think that this is happening via summation M prime and as you suggested this can be reduced to an instance of SAT bar.

So, now how do we prove this inclusion? What do you think? So, what we have to show is given an x I want to decide if x is in L or not using an oracle to SAT. So, basically the point to be noted here is that whether we have an oracle for SAT or SAT bar it is the same thing. I mean I just flip the answer if it is an oracle for SAT and I am making a query for a SAT bar instance whatever answer I get I just flip it. I mean so that is the good thing about having an oracle. So, as he suggested what we do is we construct an NP machine as follows.

(Refer Slide Time: 06:11)


M' (NP m/c)
Non-det guesses a string u.
Construct an instance of $\overline{SAT}$ (given x & u)
and depending on the oracle's answer,
Accept/Reject x.
$(NP^{SAT} \subseteq \Sigma_2^p)$
Let $L \in NP^{SAT}$ via an NP m/c M.
NP^{SAT}
$\forall x$,
$\forall v, M(x, u, v) = 1$
Co NP computation

So, let us say I construct an NP machine M prime which non-deterministically guesses a string u of whatever length it is. I mean there is some associated polynomial together with this machine. So, whatever is the value of that polynomial I guess a string of that particular length. So, when I say guess I mean that at every step I non-deterministically decide what is going to be the next bit of u whether it is 0 or 1. And depending on my sequence of guesses I land up with a string u.

So, now together with this so earlier what I had assumed is that somebody provides me with nu. But now as a result of this guess I have a string u. So, together with the x and the u I make a query to my oracle SAT bar and whatever answer it gives me I just output that. So, construct an instance of SAT bar. Given x and u and depending on the oracle's answer accept or reject. So, whatever basically this machine M was doing just simulate the machine M.

So, this side was comparatively easier. I mean you have the entire oracle first SAT sitting out there to help you and in essence you are just making one query to that oracle. So, is this clear to everybody? And we of course have an input x given to us and we are assuming we have some u. So, but we do not have a formula as such. So, where do we get the formula from? Exactly. So, basically use Kooks reduction.

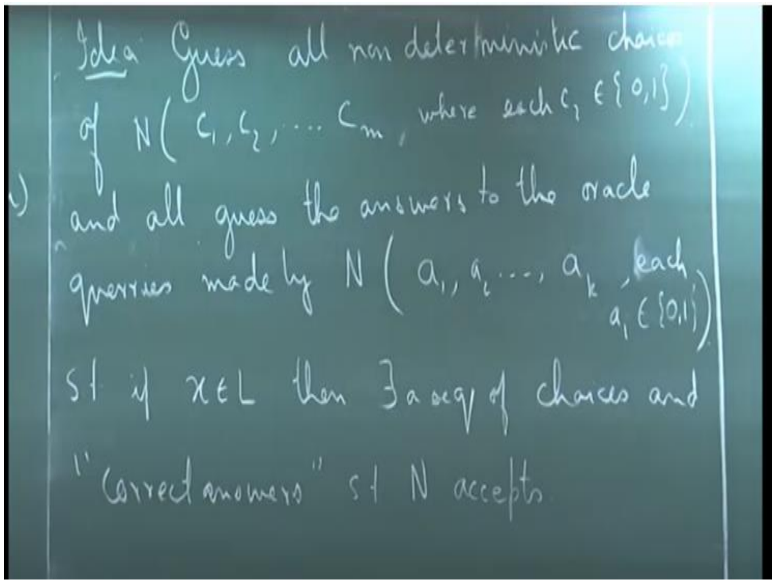
So, that is so given any NP machine we can always reduce it to some formula. And the same thing happens for a CO NP machine also. Given any CO NP machine I can reduce it to a formula which is never satisfiable. So, that is the same thing. So, this basically I mean what I am doing here to get the SAT formula is just implementing Cook's reduction. So, that was implicitly mentioned over here. So, that is one direction. What about the other direction?

So, this is more harder. Because see here we do not know how many queries the base machine that is the NP machine is going to make to its oracle. I mean potentially it can make polynomially many queries. Since it has a polynomial runtime in each, I mean in each step it can make a query to a set oracle and then depending on its answer it proceeds. So, how do we show that it can be simulated by a sigma 2 p kind of a machine?

So, here the idea is that what we do is we so let us so let L belong to NP to the SAT via an NP machine and let us say. So, the base machine is N. So, what we do is we guess two things. So, we guess all the non-deterministic choices that N makes. So, whenever it means makes a nondeterministic choice, I guess all those choices. And I also guess the answers to all the queries that N makes to it SAT oracle.

Of course, I have to verify those answers. Because when a machine I mean when N makes a query to a SAT oracle it gets a right answer. So, if I am guessing that what that answer is going to be, I must have some means of actually verifying whether it is a correct guess. So, therefore the idea is actually not again very difficult.

(Refer Slide Time: 12:21)



Idea: Guess all non deterministic choices of N ($c_1, c_2, \ldots c_m$, where each $c_i \in \{0,1\}$) and all guess the answers to the oracle queries made by N ($a_1, a_2, \ldots, a_k$, each $a_i \in \{0,1\}$) s.t. if $x \in L$ then $\exists$ a seq of choices and "correct answers" s.t. N accepts

So, the idea is guess all non-deterministic choices of N. So, let us say it has choices. It makes some choices C 1, C 2 and so, on. Maybe some m choices where each C i is some binary value. And also guess the answers to the oracle queries made by N. So, I denote the answers as some a 1 up to some a k. So, let us say it makes k oracle query. So, k can be any polynomial in the length of the input.

So, each of these a i's again is some Boolean value such that if x belongs to the language then there exists a sequence of choices. And correct answers this is very important such that N accepts. So, let us look at the so, before we give a sigma 2 p algorithm for L let us look at this NP to the SAT machine for L. So, what this machine is doing is that at each step if it is, I mean if it is a non-deterministic step then it makes some non-deterministic choice.

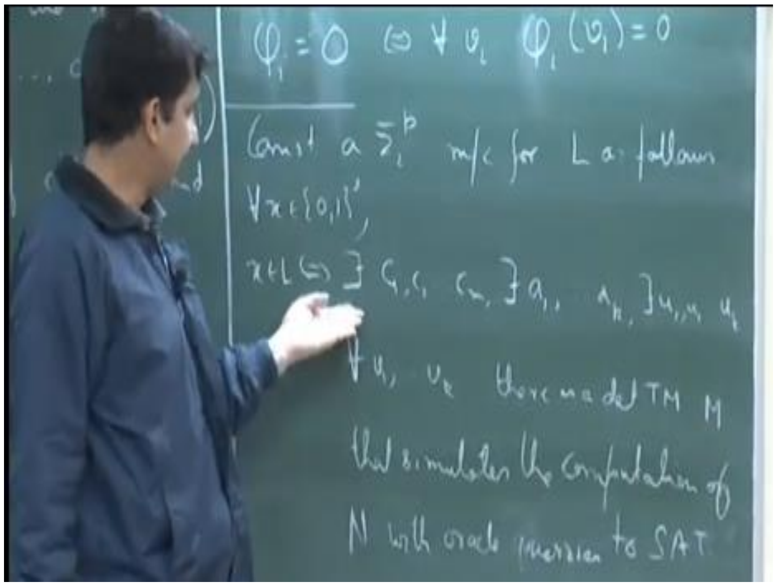
So, that can be simulated by these bits if it is the non-deterministic step, I use the bit C i. And if it makes a query to an oracle let us say it makes some ith query then I use the answer a i. But I also need to ensure that it is the correct answer. Because what is the definition of an oracle and by definition the oracle always returns the correct answer. So, at any step if the machine N produces a query for SAT, we assume that in the very next step it will get the correct answer whether that instance is a yes instance of SAT or not.

So, we need to be able to actually check whether the answers that I am guessing are indeed correct answers. Each of these queries can happen in different branches. So, that can happen in different branches. So, no but depending on which branch you are in. So, basically what this means is that so suppose if I have these choices. Suppose if I simulate N and at each point when N tries to make a non-deterministic choice, I will use the next non-deterministic bit that is there in this sequence.

And when make and when N makes an oracle query I will use that particular answer that the oracle will produce. Yes, so this is with respect to one branch. So, I am not saying which particular branch it is. I mean the branch will be determined on what these sequences are. But suppose if you fix a sequence C 1 to C m so that will fix. So, is that clear? And then when does N accept?

So, N will accept if there is at least one choice of these numbers C 1 through C m. I mean there is some string C 1 through C m and N gets the correct answer along that particular branch. And it leads up to an accepting state. So, that is when N will accept. So, let us see how to do this.

(Refer Slide Time: 18:30)



$\varphi_1 = 0 \Leftrightarrow \forall v_1 \; \varphi_1(v_1) = 0$

Const a Σ_2^p m/c for L as follows

$\forall x \in \{0,1\}^*$,

$x \in L \Leftrightarrow \exists\, c_1, c_2, \ldots c_m, \exists\, a_1, \ldots a_k, \exists\, u_1, \ldots u_k$

$\forall v_1, \ldots v_k$ there is a det TM M

that simulates the computation of

N with oracle queries to SAT

So, let phi i be the ith oracle query. So, it makes some k queries and let phi i be the ith query. So, what does it mean to say that phi i is equal to 1. So, N is phi i 1. Yes, when this formula is satisfiable in other words if and only if there exist some assignment to the variables of phi i. So, there is some string u i such that phi i given u i is equal to 1. So, when I say that I mean so this is kind of an abuse of notation.

Because here when I am saying phi i is equal to 1 what I mean is that phi i is satisfiable. But here when I am saying that phi i given u i is equal to 1 what I mean is that this formula given this assignment to its variables evaluates to true. So, just I hope you understand that. The other cases when phi i is 0 and what does this mean? Means that for all let us say strings v i phi i of v i is not satisfiable.

So, this is what it means to say that a query the answer to a query is 1 or 0. So, now how do I construct a sigma 2 machine? So, now the sigma 2 machine is not very difficult. So, we do the following. So, we construct a sigma 2 machine for L as follows. So, basically what are the CO NP parts? So, basically when phi i is equal to 0 and you have all these strings. So, but the NP part is slightly more longer because you need to guess these bits as well as the answers as well.

So, let me just write that down and then it will be clearer. So, for all x so we say that x belongs to this language. So, if I just simulate this definition of n if there exist first of all a sequence of choices some C 1, C 2 up to C m, there exists answers a 1 through a k. And we are also able to verify that these answers are correct. So, there exist strings u 1, u 2 up to u k such that for all v1 through v k there is a deterministic turing machine.

Let me call it M that simulates the computation of N with oracle queries to SAT. So, we say that the language is in sigma 2 p if there is some deterministic machine M such that there exist some string and for all some other string this deterministic machine does some polynomial time computation. So, what is the polynomial time computation that this machine is doing? So, whenever so it basically simulates N.

So, whenever N makes a non-deterministic choice it just goes and looks what is that particular value of C i. And depending on what that C i is it proceeds deterministically. And whenever N makes a query to the oracle it again checks what is that particular answer. So, if that answer is 1 then it checks that for that particular u i if phi i given u i as an assignment to its variable evaluates to 1 or not.

And the same thing with when a i is 0. So, the main idea here I mean the main idea behind this proof is that although there are, I mean the oracle is capable I mean the machine is capable of making many queries to the oracle we can in a preemptive manner guess all those answers and also verify them in an appropriate manner. So, that is the basic idea. It is not sufficient so the point is that it is not sufficient for us to just guess the answers.

Because we always assume that the oracle gives us correct answers, we should also be able to verify whether those answers are correct. And because of the nature of what that oracle is since it is a SAT oracle, I can use the power of a sigma 2 p machine to actually verify the correctness of all those answers. So, is this clear? Is this part clear to everybody what is happening? So, here so one question is so here I am using I mean here I am assuming that u i and a v i is given for every possible k. So, what if a i is equal to 1?

That is what if we are using only the u i for that particular i what happens to that vi? So, let us say so let us look at we can look at that case also. So, let us say that all the answers are 0. So, what do we do in that case? I mean even if they are not empty, we can just ignore their existence and we can just skip through them. No, but you have to be little careful because when you I mean what is the definition of a sigma 2p.

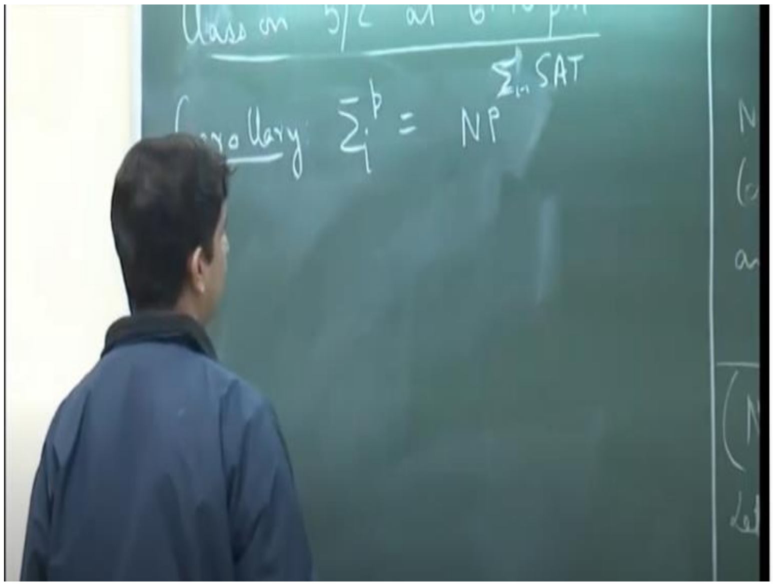
So, there exists some machine M and some polynomial q such that this string is bounded by that polynomial or not actually not bounded that string has length equal to that polynomial. And the string also has length equal to that polynomial. So, we have to be careful. So, but as you said we can just pad it up with zeros or some bits that we do not use. It can be the zero polynomial but, in our case, we actually need a some non trivial polynomial. Because these strings are some non trivial strings.

So, depending on again what N is so of course we are given this machine N. So, when we say that a language belongs to NP to the SAT it is with respect to a machine N. So, we know what that particular case. So, that was my question that what do I do if I have a particular a i which is either 1 or 0. That is what I was saying that I do not modify this notation. I just keep k strings here and k strings here but depending on what that a i is I ignore one of those u i or v i.

I just move to the next one. So, if an a i is 1 I just choose that u i and ignore the v i and vice versa. That is what I was saying. Because we do not know I mean you are we do not know what this a i's are going to be. So, that is why we cannot assume beforehand how many u i's we need and how many v i's we need. So, now as a generalization let me just write this as a corollary. I would assume it is something like a function call.

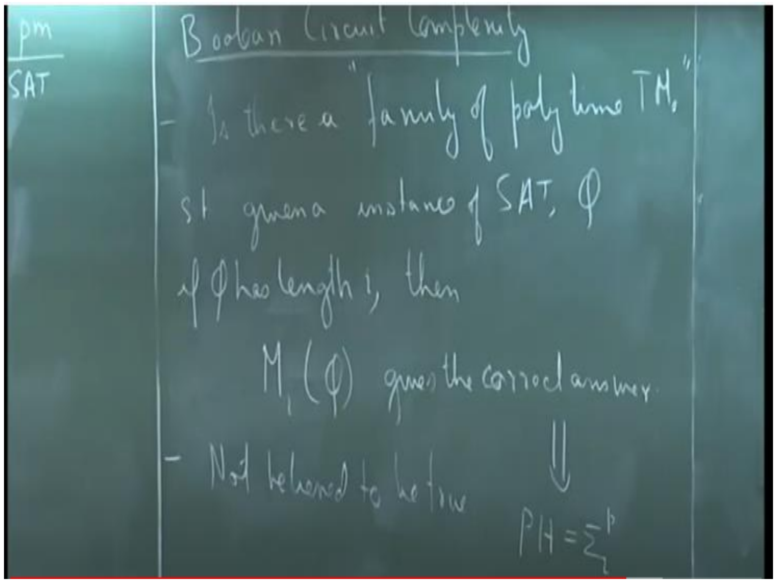
I mean if you are writing let us say you are in a programming language setting and you are making a call to let us say some function that is probably the closest analog to an oracle. So, where do you assume that the function will always return you an answer, I mean a correct answer.

(Refer Slide Time: 28:41)



So, what we can say is sigma i p is NP to the sigma i – 1 SAT. So, I do not think this p is used here. So, what we saw so far were three different ways of looking at the polynomial hierarchy and various properties about the hierarchy. So, let us move on.

(Refer Slide Time: 29:44)



pm

SAT

Boolean Circuit Complexity

- Is there a "family of poly time TM,"

s.t. given a instance of SAT, ϕ

if ϕ has length i, then

$M_i(\phi)$ gives the correct answer.

- Not believed to be true $\Downarrow$ $PH = \Sigma_2^p$

So, the next topic that we would discuss is a Boolean circuit complexity. So, what does it mean so before looking at what we mean by circuit complexity, again let me try to motivate as to why this might be interesting. So, let us go back to the P versus NP question. So, what does that question ask? I mean let us say I want to get a positive answer for that question. What that amounts to asking is there a polynomial time algorithm for SAT or for that matter any other NP complete problem.

So, now suppose if I kind of relax this question so instead of asking whether there is a polynomial time algorithm for SAT. What if I ask that is there a family of polynomial time algorithms? So, one for every different length of the given instance. So, whatever SAT instance that is given to you look at that the length of that instance and depending on what that length is you put it to that particular turing machine.

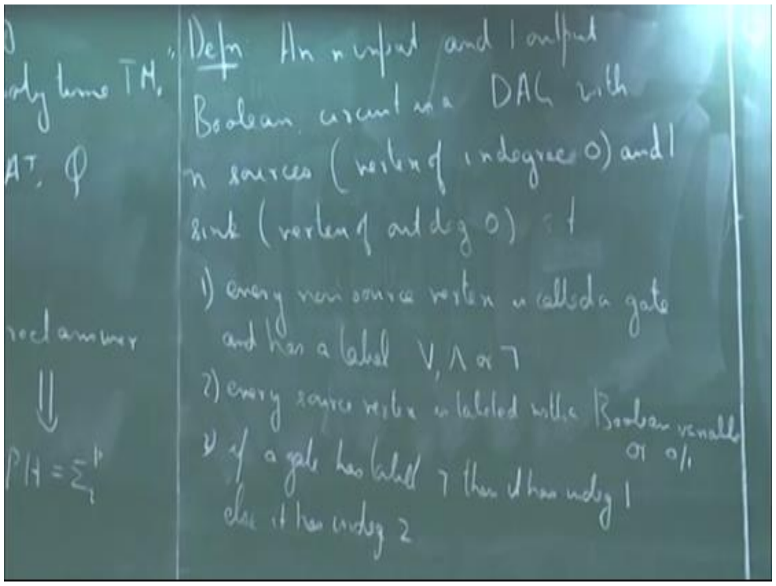
So, is there a family of polynomial time turing machines? So, when I say family what this means is it is an infinite class of turing machines such that given a instance of SAT, let us say phi. If phi has length i then M i given phi gives the correct answer, any questions? So, of course this is an easier question because now no longer we have the constraint that only one turing machine should solve the SAT problem.

What I am asking is that is there a family of turing machines an infinite family so that depending on what the instances. If that instance has length i the machine M phi would correctly decide whether phi is satisfiable or not. So, this is again the question which was asked by people immediately after showing that SAT is NP complete again in the late 70s that whether we can prove that such a relaxed version is indeed possible.

And as it turns out that even this is hard so in the next couple of lectures, we will show that why this is also hard. So, I will just mention it here so this is also not believed to be true. But this was the motivation for I mean this was probably the earliest motivation why people started looking at what is known as circuit complexity. No, so actually we cannot say NP is equal to P. What we can say is something little less thoughtful. We can say that NP is equal to sigma 2 p.

No, I am sorry NP is of course contained in sigma 2 p. We can say that the polynomial hierarchy is equal to sigma 2 p. So, that is not believed to be true. So, maybe I can mention this that if this is true then it would imply that the polynomial hierarchy is equal to sigma 2 p. And this is probably the main reason why this is not believed to be true. So, what is the relation between this question and circuit complexity? So, to see that let us look at what circuits are. So, what is the circuit? We have an informal idea. But we have to be little careful.

(Refer Slide Time: 35:43)



oly time TM,

AT, ϕ

rect answer

$\Downarrow$

$PH = \Sigma_2^p$

Defn An n input and 1 output Boolean circuit is a DAG with n sources (vertex of indegree 0) and 1 sink (vertex of outdeg 0) s.t.

1) every non source vertex is called a gate and has a label $\vee, \wedge$ or $\neg$

2) every source vertex is labeled with a Boolean variable or 0/1

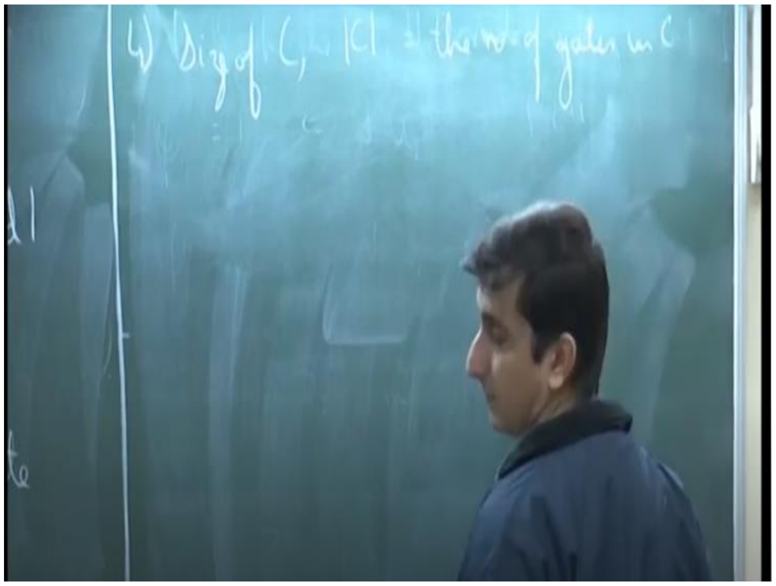
3) if a gate has label $\neg$ then it has indeg 1 else it has indeg 2

So, let me give the proper definition. So, an n input and 1 output Boolean circuit is a directed acyclic graph with n sources. So, a source is a vertex of in degree 0 and 1 sink such that we have the following conditions. So, what is to be noted here is that it is a directed acyclic graph. So, it is a directed graph and it cannot have cycles. So, we will come to that little later. Every non-source vertex is called a gate and has a label OR, AND or NOT.

Every source vertex is labeled with a Boolean variable, some Boolean variable. It can be 0 or 1 as well. So, actually yes, I mean there are many generalizations which are possible. I mean you can also have a circuit with more than one output. So, if it has multiple sinks you can think of it as outputting many bits. And as you said that you can also have sources that are labeled with constants that is 0 or 1. I can include that as well. I guess or 0 or 1.

So, what else? If a gate has label NOT then it has in degree 1 else it has in degree 2. So, we assume that OR and AND gates have in degree 2. So, again this also can be generalized. And finally let us say the size. So, let us call it Boolean circuit c.

(Refer Slide Time: 40:25)



Size of c which is denoted as follows is equal to the number of gates in c. So, again some texts refer even to the source vertices as gates. They are called input gates. But this book only refers to the non-source vertices as gates. So, these are all matters of terminology. So, this is what the definition of a circuit is. And now how is the circuit evaluated? So, it is evaluated in a very natural manner.

So, if you have a source vertex so every source vertex is labeled with either the constant 0 or 1 or with some variable x i. And if you have a non-source vertex then the value at that vertex depends on what is the input to that node. So, if it is a NOT gate then the value at that vertex is the negation of its input. And if it is an OR under or an AND gate its value depends on what that operation is.

So, it is defined in the usual manner and the value of the entire circuit is equal to the value at this gate at the sink whatever is the value. So, now given a string x let us say you have a string of length n you can check what is the value of the circuit if that string is provided to the input gates of that circuit. So, I will stop here. You are running out of time. We will look a little more into the definition tomorrow and we will continue from there on.

.