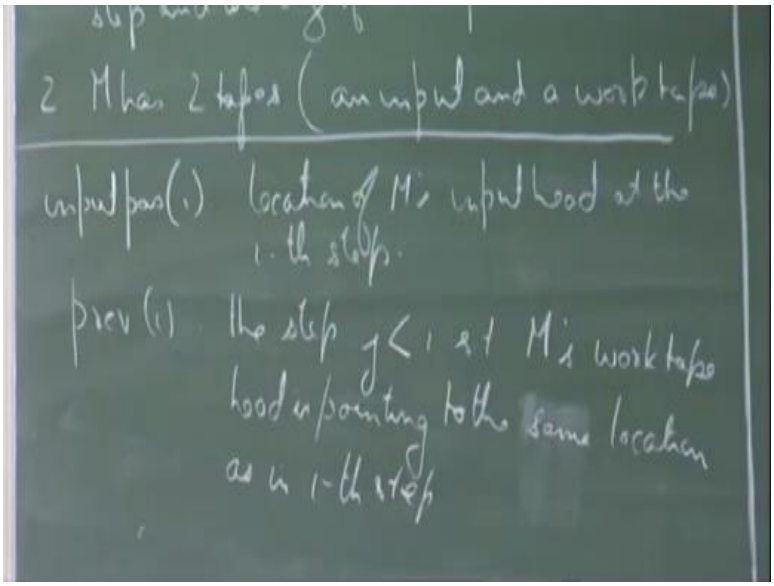
Computational Complexity Theory
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture -03
Introduction

Good morning everybody, so we were looking at the proof that SAT is NP complete last time and we saw that we can make certain assumptions about the turing machine that we have. So we take so basically the what we saw last time was that we start with an NP complete language L, which means that there exist some deterministic polynomial time turing machine m which given some certificate you can verify whether a given instance x together with the certificate u gets accepted or not.

Depending on that we will construct an instance of satisfiability phi such that phi will be satisfiable, if x is in the language and phi is not satisfiable, if x does not belong to the language. So the 2 assumptions that we so let me just state that again.

(Refer Slide Time: 01:29)



So firstly we assume that M is oblivious, what that means is the head movement on M's tapes depends only on the current step and the size of the input. So in other words suppose you are at the i th step and so the machine has if you recall 2 tapes, there is an input tape and there is a work

tape. So there are two input heads each pointing to one of the two tapes and the movement of the head in both the tapes only depends on which step it is, that is the particular i.

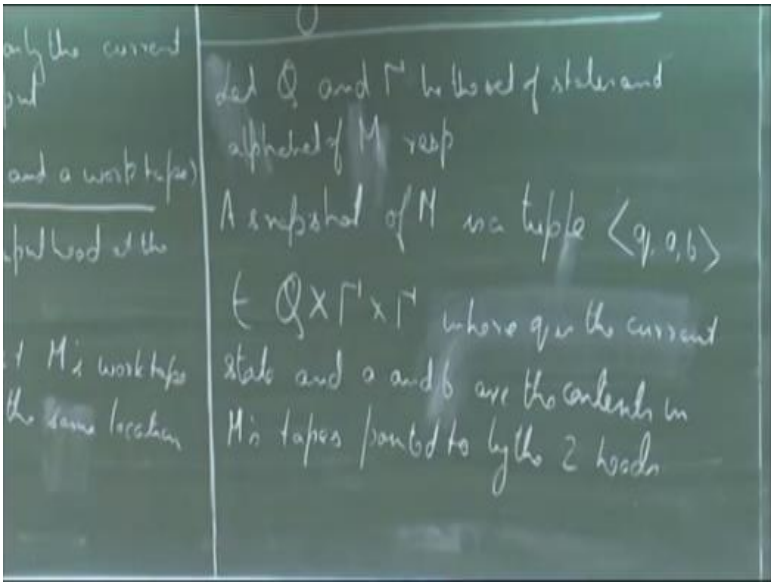
And whatever is the size of the input. In other words it does not depend on the actual content that belongs or that lies in the cell to which that head is pointing to. So that is what it means to say that M is oblivious and the second assumption is which i just said that M has 2 tapes an input and a work tape. So now the question is why do we assume? So why do we make these two assumptions?

So let me motivate that. So we use the fact that M is oblivious to make certain computations. So in other words, let us define two functions, let me define a function input position so this stands for the input position at the i th step. So the value of this function is that given an argument i what is the location of M's head in the input tape at the i th step. Location of M's input head at the i th step.

And i define another function previous of i this is also takes as an argument this is the step before i so let me say just say the step j strictly less than i such that M's work tape head is pointing to the same location as in the i th step. So whatever is the location that the work tape was pointing in the i th step previous of i is just the step prior to i when it was pointing at that same step whatever that value j is.

And now the thing is that suppose i want to determine these two values for all possible i's so we are given an input x and so note that M takes 2.

(Refer Slide Time: 06:45)



The machine takes 2 strings as arguments it takes a string x and it also assumes that there exist some string u. So what we can do is we can take the all 0 string so u 0 that has length some P of x so there is some polynomial P and u has length p of the size of x, so we can take the all 0 strings for u. Suppose this is the string 0 to the power P of x and i just simulate the machine on this input that is x and 0 to the power P of x.

And that will allow me to get all possible values of input position of i and previous of i for all possible i's. Because M is oblivious it does not matter what u is, it will always for every possibility it will have the same values. So this is where we use the fact that M is oblivious and we use the fact that it has only two tapes just to simplify our computation. I mean we do not need to consider multiple such functions each corresponding to a different.

Any questions any clarifications regarding this, notion of a step, so you have so suppose you have a turing machine so it has a input on its input tape and now depending on what the transition function is it takes one step each time so it reads let us say the first input of the input tape and then depending on what the transition function dictates it possibly changes its state it writes something on to its work tape it moves its input head.

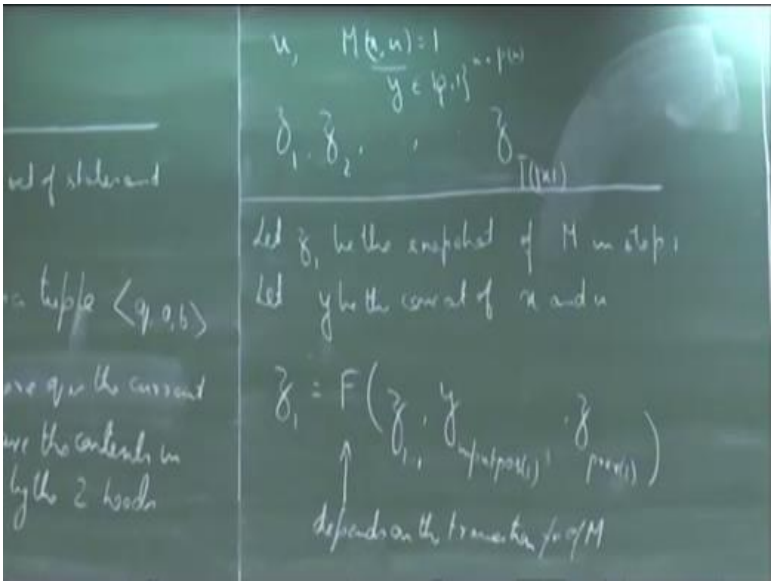
So each such execution of the transition function is basically a step of a turing machine. So if you look at the section on turing machines I think it is in the first chapter, so there they formally

define what a step is and all these things. Any other questions? So now let me define a snapshot of the turing machine. So snapshot is the word which this text uses but alternatively you can also think of it as a configuration if you comfortable with configuration of the turing machine.

So a so let Q and gamma be the set of states and alphabet of M respectively then a snapshot of M is a tuple some q, a, b belonging to Q cross gamma cross gamma corresponding where q is the current state and a and b are the contents in M's pointed to by the 2 heads. So whatever is the current state and whatever are the contents of the 2 cells that is being pointed two by the 2 heads of M states.

So now let us have; so now let us see what do we exactly need to answer this? And answer the following question.

(Refer Slide Time: 13:03)



So suppose somebody presents to you with a string u suppose somebody gives you a string u which has the property that M, x, u is equal to 1. In other words suppose somebody gives a correct certificate and as an evidence that it is a correct certificate he gives a sequence of snapshots. So let me denote a snapshot by the symbol z so he gives the first snapshot and he gives the second snapshot.

And since M has polynomial running time and that also can be figured out and what is the running time of M on a particular string x together with a given certificate that also can be figured out from the fact that M is oblivious, so you run it on the all 0 input and whatever total time it is taking let us just denote that by t of the size of x. So somebody presents you that many snapshots as an evidence.

How can you check whether this is a correct sequence of snapshots which determines the fact that M, x, u is equal to 1. So firstly you need to check whether the first snapshot corresponds to the initial snapshot or the initial configuration that is it contains the start state as q and it contains the first two symbols of the input tape. And the work tape then you must verify whether given a snapshot z i or let say given a snapshot z i plus one is it a correct snapshot from z i mean since this is a deterministic turing machine.

Since M is a deterministic turing machine there is only one way of going from one snapshot to the next snapshot. So given a particular snapshot is the next snapshot the correct one depending on the transition function of M. And there is one more thing that needs to be pointed out here is that since M is a deterministic turing machine only a constant number of bits get modified in each of the snapshots of M. Suppose if you want to go from particular z i to z i plus 1, maybe you make a change of position in the input tape maybe you make a change of position in the work tape and maybe there is a change of state.

And with all these three changes; constantly only a constant number of bits get modified. So the next thing is to check if z i plus 1 is a correct snapshot depending on what the previous z i's were and thirdly you need to check whether z, T of x is an accepting snapshot. In other words is it a snapshot which halts at an accepting state. Anything else? Well we need to also check if the input is correct or not so basically you are given this string x, u so let us denote this as some y which has length some n + P of n.

So we just need to check if the first n bits of y correctly encode x or not because u is something that is being presented to you and as an evidence you are given these snapshots but whether the first n bits of y correctly encodes x or not. So we need to be able to perform all the following

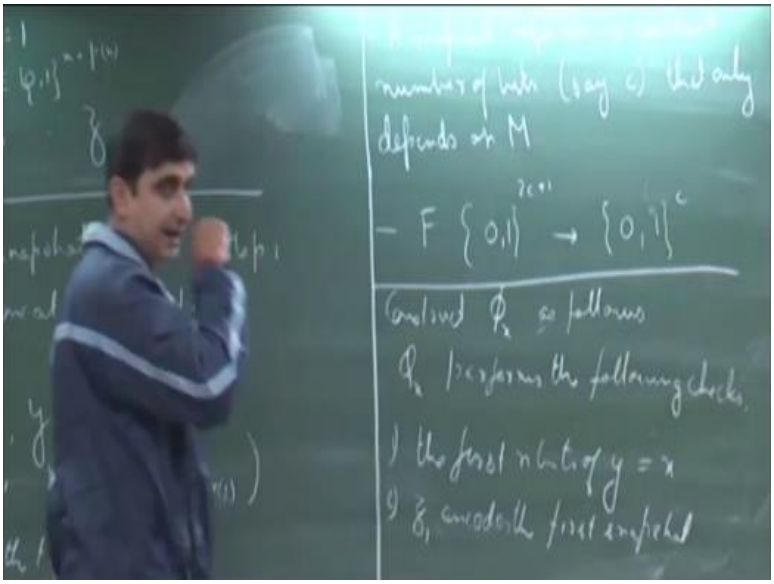
checks. And the claim is that all these checks can be performed very efficiently. I mean we can encode all these checks as a Boolean function which uses only constant number of bits.

And now by using the claim that we saw last time we can convert it to a CNF formula. So let me write down what I mentioned so let z i be the snapshot of M in step i and let y be the concatenation of x and u we will just denote it by y. So then the claim is that there exists a function F capital F such that z of i depends on the following it depends on what the previous snapshot is.

What is the content of the cell at which the input head is pointing to in the ith step that is y input position of i and we also need to know what is the content of the work tape and for that it is enough to look at the snapshot in this particular step. Because after this step if you look at the definition of previous of i after this step the work tape was not at all modified this was the last step prior to i in which the work tape was modified.

So I just need to look at that particular snapshot as well z previous of i. So given these three arguments I can easily compute a Boolean function that depends on the transition function of M so f depends on the transition function of M and I can check whether it correctly encodes z i or not. So what is the size of this argument that the function takes.

(Refer Slide Time: 20:54)



So suppose, before we look into what is the size of this argument let us look at what is the size of a snapshot. Suppose if you look at a particular snapshot how many bits does it require, so basically it requires a log of Q plus some 2 log gamma so in other words a snapshot requires a constant number of bits say c that only depends on the machine M in other words it does not depend on the input size.

So now what is the size of this argument as a function of c, so F is a function which takes an argument from set of strings of length 2c +1 so you need c bits for the previous snapshot c bits for this snapshot, the snapshot where the same cell was visited in the workday and one bit for what is the content of the input tape at that particular step and it maps it to strings of length c. So now, we have all the tools that we require we just need to go ahead and compute or define what the Boolean formula would be corresponding to x.

So now construct Boolean formula phi of x as follows, so phi of x performs the following checks so firstly it checks whether the first n bits of y is equal to x or not. So here is a good exercise for you guys to try out. So note that we want so basically what we are trying to do is we are trying to construct a Boolean formula which should be satisfiable if certain conditions are met. And here the first condition that I am stating is checking equality of two boolean strings.

Or the question is how do I check equality based on the three operands of Boolean formula that is and or and not. Suppose you are given a string x and a string y let us say both having n bits how do you check if these three these two strings are equal or not by constructing a Boolean formula based on these two strings. So as he said what you can do is I just would not write it here you can construct a Boolean formula which is x or not of y and with not of x or y and if this boolean function evaluates to true then the string x and y are the same otherwise they are not.

So you can just check that as an exercise. So, that allows us to make this claim that we can check whether the first n bits of y are the same as x or not. Secondly we need to check whether z 1 encodes the first snapshot.

(Refer Slide Time: 27:01)

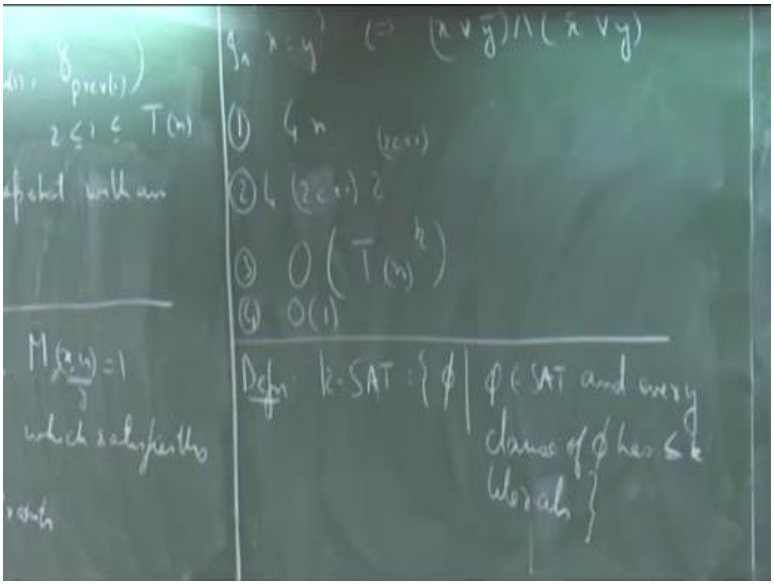
Thirdly what you need to check is that given snapshots z i -1 did I define F, the bit y input position of i and z previous of i does the function F of this argument correctly gives the i th snapshot or not, Again this is checking a certain equality of certain boolean strings and finally we need to check whether z, T of n encodes a snapshot with an accepting configuration with an accepting state.

So we construct a Boolean formula phi of x which encodes all the following checks. So now we need to check 2 things we need to check correctness, if there is a Boolean formula which encodes all these four things then it is a correct reduction and secondly we need to check whether this is polynomial size and whether this reduction is a polynomial time reduction or not. So let us check correctness first.

So if x were to be in l so x is in l if and only if there exists, a string u such that M of x, u is equal to 1 and this is true if and only if there exist a sequence of snapshots of this machine. So again I think of x, u as the string y so this is true if and only if there exist some sequence of snapshots z1 through zd of n which satisfies the above four constraints. This is by definition so if there is a if there is such a u as I argued earlier then you will find a sequence of T n snapshots which satisfies these 4 conditions.

And if there is no such u then you will not be able to find and what about the size of phi of x. So let us just divide that into the following parts, how big a formula do you need to check the first condition?

(Refer Slide Time: 31:13)



So suppose I want to check equality of two strings x and y so is x equal to y the Boolean formula that your friend said just now is, i mean so this checking basically corresponds to the Boolean formula x or not of y and not of x or y. So how large is this in terms of the size of x and y. It is about 4 n roughly, so for the first step you need a formula of length about 4 n what about the second step?

So again you basically it is also checking equality and it is checking equality for a particular instantiation of this function. So therefore by that earlier claim, claim 1 of last class we can create a formula of length c times no it is not c what is the length it is 2 c + 1 times 2 raised to 2 c +1 times, sum 4, so essentially its some constant. It does not depend on the input length again. For the third step basically you need a something larger so here also you are checking equality but you are doing it for all possible i.

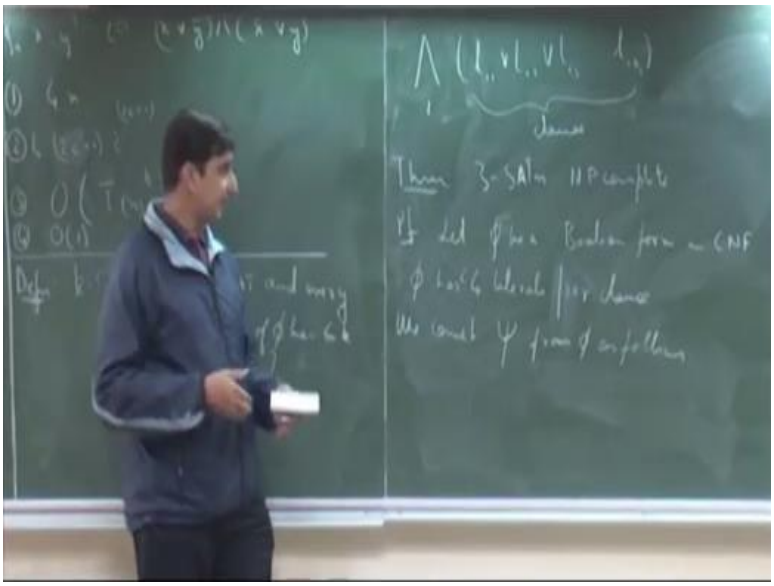
So basically, i greater than or equal to 2 and less than or equal to T of n. So here you need some let me just skip the details and just let me write this as some big O of some T n to the power some constant and again for the fourth step you need some constant size formula. So now your

final CNF formula is just an end of all these four formulas. And as you can see that this can be constructed in polynomial time as well as it has size polynomial.

So fourth step is also some order of let me just keep it as. So now we have seen the construction of the first NP complete language and so this is very interesting because not only does this say that there exist i mean all this theory of NP completeness not only does this give an existential proof of NP complete languages but it also gives a very natural problem so satisfiability i mean the problem of satisfiability is a very naturally occurring problem.

So let me just go ahead and mention something more, maybe then I will come back to this discussion so let us see one more reduction using Boolean formulas. So k-SAT so we saw what the language SAT is so k-SAT is the set of all boolean formulas phi such that phi belongs to SAT and every clause of phi has at most k literals. So did I mention last time what do we mean by clause and literals? Maybe I did not. So if you look at a CNF formula what is the structure that a CNF formula has in general.

(Refer Slide Time: 36:30)



So every CNF formula is basically an and. So I can write it as a global and of some i clauses, so each clause is basically and or of certain literals so i can have a literal l i1 so let me write it this way an and of literals l i1 or l i2, l i3 so 1 up to let say some l ik subscript type. So, each of these

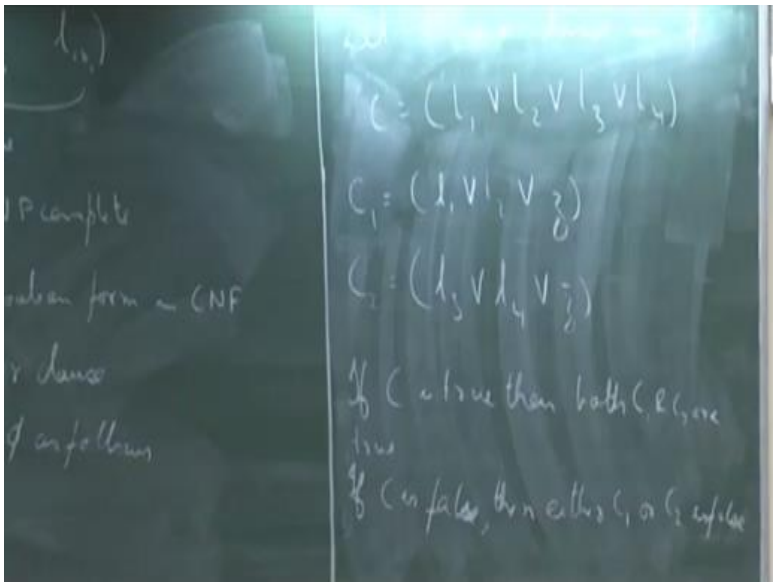
formulas is known as a clause. So a clause is basically an, or of literals and this together I mean when you look at the and of all these clauses that gives you a CNF formula.

Now a k SAT is basically a CNF formula that is satisfiable and every clause of that formula has at most k literals. So as it turns out what we can show is that 3-SAT is also NP complete. In other words we can take a arbitrary Boolean formula in CNF form and we can reduce it to a Boolean formula that has 3 literals per clause. So how to prove this so let phi be a Boolean formula in CNF in conjunctive normal form.

So I will make one assumption about 5 for the time being later, we will see that how that assumption can actually be generalized. So we will assume that phi has 4 literals or it has at most 4 literals per clause. Now let us reduce phi to some other Boolean formula that is an instance of 3- SAT. So we construct a formula psi from phi as follows. So any idea how we can reduce this?

So let us just assume that for the time being. So in general you are it can have arbitrary number of literals but let us just look at the case when it has at most 4 literals. We will look at the general case later on. So the idea is that we go clause by clause so we pick each clause in phi and we reduce it to a set of clauses that belong to 3-SAT or that are instances of 3-SAT.

(Refer Slide Time: 40:50)



So let C be a clause in phi, so by our assumption C has the form sum l 1 or some l 2 or l 3 or l 4.

How can we reduce to a 3-SAT instance? Maybe by adding extra variables if necessary; so the idea is that if this is satisfiable then whatever we are constructing should be satisfiable and if this is not then the reduced formula also should not be satisfiable. So let me give the answer to that, so what we do is that given C we construct 3 clauses as follows.

So we construct a clause C 1 which is equal to l 1 or l 2 or we introduce an additional variable let us call it z so you construct a clause c 1 that is l 1 or l 2 or z and we construct another clause C2 which is l 3 or l 4 or the negation of z. So when we say we have a literal so literal by definition is either a variable in the formula phi or its negation so that is the definition of a literal. So these are the two clauses that we construct from C and note that both these clauses have 3 three literals in them.

So now if C were to be satisfiable that is if C evaluates to true, if C is true then what can we say about C 1 and C 2. At least one of them can we say something more, both why both because we will take, so if the clause c is true then it means that there is at least one literal in these amongst these four that is true. So let that literal belong to C 1 without loss of generality then we pick z to be 0 that also makes C 2 true and if that literal belong to C 2, then we set z to 1 which makes C 1 true as well.

So if C 1 is true then both sorry if C is true then both C 1 and C 2 are true and if C evaluates to false then what so if C evaluates to false what it means is that all these 4 literals evaluate to false. It means that no matter what value we give to z at least one of C1 and C2 have to be false. Then either C1 or C2 is false. Now if we take a; and of all these clauses C1, C2 for every possible clause C in phi, we get a reduction from at least in this case from 4-SAT to 3-SAT.

And now you can see how this can be generalized. So you start with any k-SAT you reduce it to k minus 1-SAT. So that introduces how many new variables, so how many so how many variables do you need in this case suppose you are reducing 4-SAT to 3-SAT at most the number of clauses because you cannot reuse the same variable. So you introduce m number of variables and then you go from k minus 1 to k minus 2 and so on you can come down to 3.

So can you come down any further can you reduce 3-SAT to 2-SAT by this approach no. I mean clearly the way we are designing the clauses C 1 and C 2, I mean this does not give us a way of reducing 3-SAT to 2-SAT. So at least this approach has the bottleneck that we will stop at 3-SAT and actually that is in some sense what is expected because 2-SAT as it turns out has a complexity which is much lower than that of 3-SAT.

So 2-SAT is not only in polynomial time may be in a couple of lectures we will see that it lies in a complexity class which is even smaller than polynomial time. So this is a very surprising structure about again Boolean formulas that the moment you come down so for all k that is greater than or equal to 3, they have the same complexity as we can see by these sequence of reductions but the moment you come down from 3 to 2 there is a well significant drop in the complexity of Boolean formula or at least that is what our current understanding tells us.

So we do not know so it might turn out that they are equal as well but that is not what people believe.