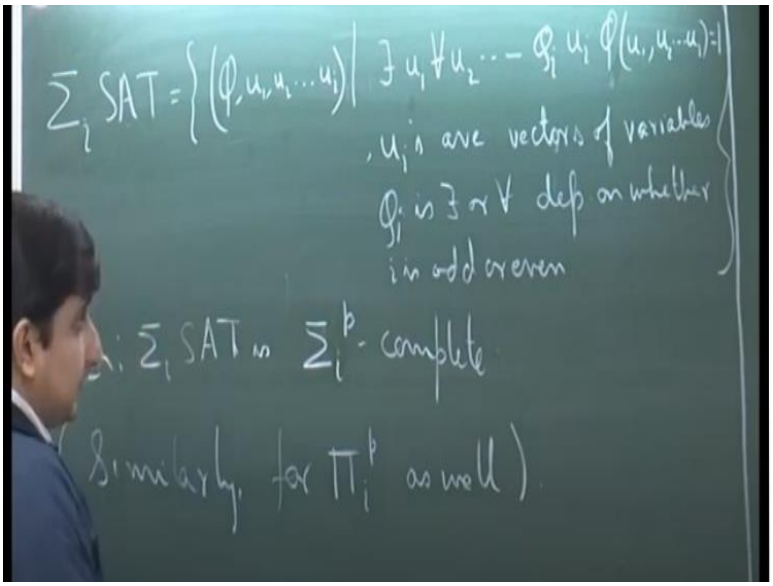
Computational Complexity Theory
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture -11
Introduction

So, we look at another collapsing result. So, just before I begin let me recall a little bit from what we did last time.

(Refer Slide Time: 00:32)



So, we defined these languages sigma i SAT which are complete problems for the various levels of the polynomial hierarchy. So, what is the definition of sigma i SAT? If I want to write it in a set theoretic manner, so these are all Boolean formulas phi such that there exist some vector of variables u 1 such that for all vectors of variables u 2 and so on up to Q i u i phi of u 1, u 2, u i is equal to 1 such that u i's are vectors of variables.

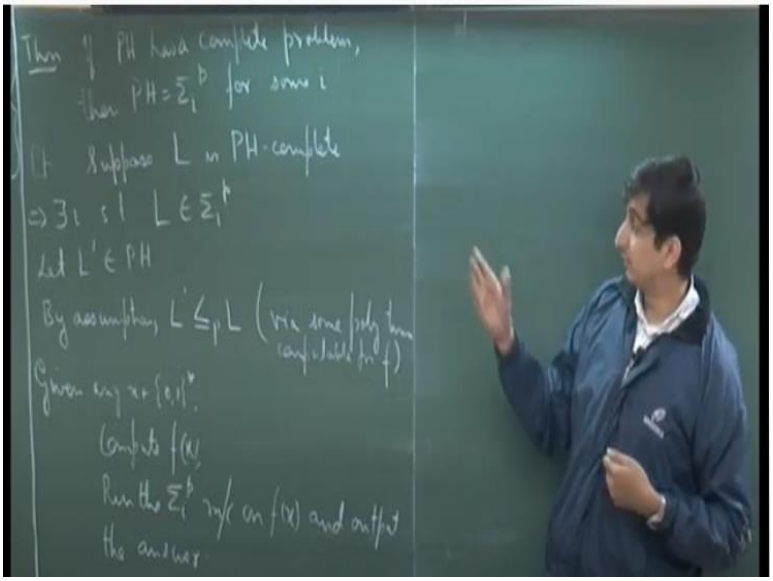
And Q i is there exist or for all depending on whether i is odd or even. So, this is not quite complete because in the definition of the problem so when I am specifying the instance, I also need to specify what these vectors are. So, for the case of SAT it is implicitly we have the only the existential quantifier attached with every variable. But when we are looking at this

generalization of SAT so every variable is either associated with a there exist quantifier or a for all quantifier not only that it has a specific place where it exists.

I mean it can only belong to exactly one of these following vectors. So, these variables I mean these vectors also need to be specified. So, I have u 1 some u i. So, this is the instance that is given to us and this is what we have to decide and we mentioned this theorem last time that sigma i SAT is sigma i p complete. So, this I left as an exercise and similarly we can show the same thing for pi i p as well.

So, we can define a language where the first quantifier is a for all quantifier and there are i alternations and pi i p is pi i SAT is complete for pi i p. So, this is what we saw last time.

(Refer Slide Time: 04:36)



So, the first result that we would see today is that if PH has a complete problem then PH is equal to sigma i p for some i. So, in other words the polynomial hierarchy will collapse to a certain level. So, again this is not very easy to I mean not very difficult to prove this chalk is. So, suppose I mean we can prove this by contradiction. So, suppose L is PH complete. Suppose there exist some complete problem then how do you argue?

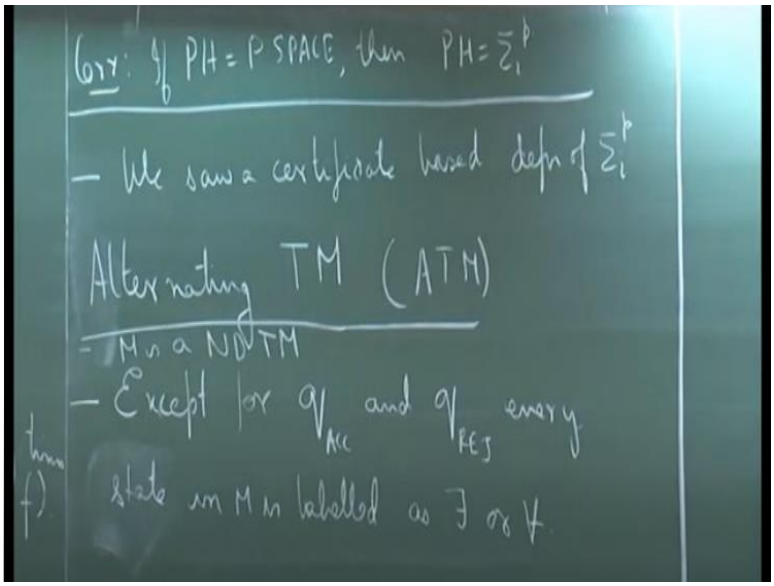
So, I mean L will belong to some level of the hierarchy. So, there exist an i such that L is in sigma i p. Since PH is nothing but the union of all the levels. And I mean without loss of

generality we can assume that i is the lowest such i such that L belongs to sigma i p. I mean if it belongs to some particular sigma i p it will also belong to all sigma j p such that j is greater than or equal to i. So, now let L prime be any language in the polynomial hierarchy. So, by our assumption L prime reduces to L. So, we are looking at polynomial time reductions here. Since it is complete and we can assume that it reduces via some function via some polynomial time computable function f.

So, then so basically, we have to show that L prime will also belong to sigma i p. So, given an instance x, so given any x, first compute f of x using this polynomial time machine. So, since f is a reduction, we know that f of x is an instance of L and we assume that L belongs to sigma i p. So, now run the sigma i p machine on f of x and output the answer. So, therefore what we have shown is that this is an algorithm to decide any language in the polynomial hierarchy in sigma I mean by using a sigma i p machine only.

So, therefore it is not believed that I mean since as I mentioned earlier that we do not believe that the polynomial hierarchy collapses. Therefore, this implies that it is quite unlikely that the polynomial hierarchy will have a complete problem.

(Refer Slide Time: 09:21)



So, I mean just as a corollary of this we can again make the following claim that if the polynomial hierarchy equals P SPACE. So, again recall that this problem is a restriction of the

TQBF language. Because in the TQBF language you can have arbitrary number of alterations. So, which implies that the polynomial hierarchy by definition is contained in. So, PSPACE suppose they were equal suppose the other way inclusion also holds then again, the polynomial hierarchy will collapse to some level.

And the argument is again the same. So, suppose if PH is equal to P SPACE then the language TQBF belongs to some level of the hierarchy and since it is hard for PH as well TQBF therefore it will collapse to sigma i p. This is the same argument. Any questions? Which part? This one? This is just look I mean this is just so first what we are doing is so basically, I want to decide L prime using a sigma i p machine.

Well I should not use the word sigma i p machine because we only have a certificate based definition for sigma i p. So, I want to decide L in sigma i p. Maybe that is a better way of expressing it. So, what I do is that given any input x, any instance x, I first compute f of x. So, this I can compute because I know that f is a polynomial time computable machine. So, I use the same thing and then I run the well I may just put it in quotes whatever the sigma i p machine means on the input f of x. So, instead of using x now I am using f of x.

So, basically what this means is that there exists some polynomial time turing machine m prime such that there exist some string u1 such that for all u 2 dot dot whatever, f of x will either be accepted by that model or not and that is what the answer that I will output for x as well. Any other question? So, what we will see next is a machine based definition of the polynomial hierarchy or in more particularly for the various levels of the polynomial hierarchy. So, what we saw so far was a certificate based definition.

So, how do I translate this to a turing machine based definition. So, for this we define these generalizations of non-deterministic turing machines which are known as alternating turing machines. So, in short, we will just refer to them as ATM. So, do not confuse this with the I mean the ATM language that you might have seen in your TOC course that is it is the language of all machines which accept every input. So, this is different and certainly not the bank ATM.

So, what is the deal with these alternating turing machines? So, what is a non-deterministic turing machine? So, one way to view a non-deterministic turing machine is that there are two transition functions. So, I can assume there are two deterministic transition functions. When I say deterministic, I mean that from a given configuration the machine I mean that transition function takes the machine to exactly one other configuration or at most one other configuration.

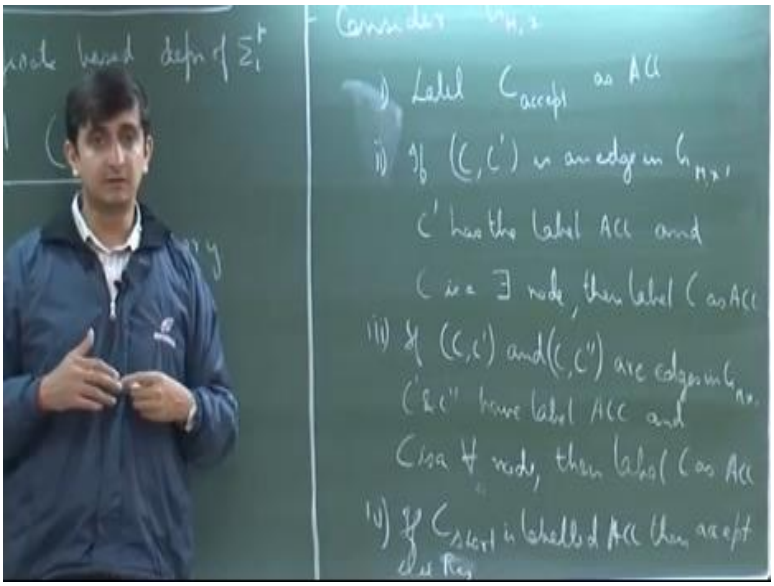
Well it can be a halting configuration in which it does not go to any other. And what the non-deterministic machine does at each step is that it non-deterministically picks one of the transition functions and decides to act according to it. And then we say that an input is accepted by this non-deterministic machine. If there is a sequence of possible choices that can be taken by this machine that is well in the first step, I will take transition function one. Maybe in the second step I will again take transition function one.

The third step the machine decides to take transition function two and so on such that it leads it to an accepting state. So, that is when I accept an input and an input is not accepted when no such sequence exists. So, what we so we generalize this notion for the case of an alternating turing machine. So, what we assume is that so except so again we will assume without loss of generality that there are two halting states. So, one is the accepting state and the other is the rejecting state.

So, we can assume this without loss of generality. So, except for and let us call them q accept and q reject. So, except for q accept and q reject every state in M is labeled as there exist or for all. So, maybe I should say this at the beginning said M is a non-deterministic turing machine. And so, in addition to it being a non-deterministic turing machine every state which has which is not the halting states.

They have either a there exist or a for all quantifier associated with it. So, therefore we can talk about configurations also in the same way that every configuration has there exist or a for all quantifier associated with it. Because basically whatever the state is corresponding to that configuration will put the label of that state. So, now how do I accept an input? So, first let me give the informal definition and maybe that will be sufficient.

(Refer Slide Time: 18:28)



So, let x be some string and consider the configuration graph of the machine on this input x. So, now we will give a way of labeling all the vertices in this configuration graph and we will do this in a recursive manner. So, first label C accept as let me use the label accept for it. Then if C, C prime is an edge in G M, x such that C prime has the label accept and C is a there exist node. So, when I say that C is a there exist node what I mean is that the state which is associated with c has the label there exist in the usual manner.

Then label C as accept and similarly if C, C prime and C, C double prime are edges in G M, x such that C prime and C double prime have label accept and C is a for all node. Then label C as accept. So, what do you think we should do next or more particularly now what do you think should be the condition by which we will accept x? Yes, so now we just check what about the start configuration. What is the label of the start configuration?

So, if C start is labeled accept then accept else reject. So, just for the time being let us forget that it is an alternating turing machine. Just let us assume that it is a regular non-deterministic machine. So, in that case all the nodes have a there exist label associated with it. So, what does this definition say or what does this say? What we are trying to do is just check if there is a path from C accept to C start. So, I am just tracing back or I mean it can be stated the other way round also. Is there a path from C start to C accept?

So, I am associating labels with the states. So, every configuration has a state associated with it. I mean there is a canonical state which is associated with every configuration. A non-deterministic machine does not say that. So, the definition of a configuration is the same whether it is a deterministic machine or non-deterministic machine. So, what is the configuration? Essentially a configuration is basically a snapshot of the machine.

I mean at any time stamp a configuration will tell you exactly what is the entire information that is being conveyed to us about the machine. That is what is its state? So, state is just one of the elements of the configuration. What are the contents of its work tape? At what position is its input head? At what position is its work tape head? And I guess that is it. That will then exactly tell you what will happen at the next configuration. So, the difference is not in whether what a configuration is.

The difference between a deterministic and a non-deterministic machine is that how a machine goes from one configuration to the other. So, if it is a non-deterministic machine then it non-deterministically goes to more than one configuration at each step. Whereas in a deterministic machine it can only go to at most one or the other configuration. So, again I mean I want to emphasize that non-deterministic machines are not a practical model of computation. I mean there is no practical way in which a non-deterministic machine can be simulated.

The reason why again this is studied is because I mean these machines kind of captures the difference between being able to compute something and just verifying whether something that has been computed is correct or not. So, suppose somebody presents some mathematical theorem to you and asks you to prove it and on the other hand somebody gives you a proof of a statement and asks you to verify whether it is a correct proof or not.

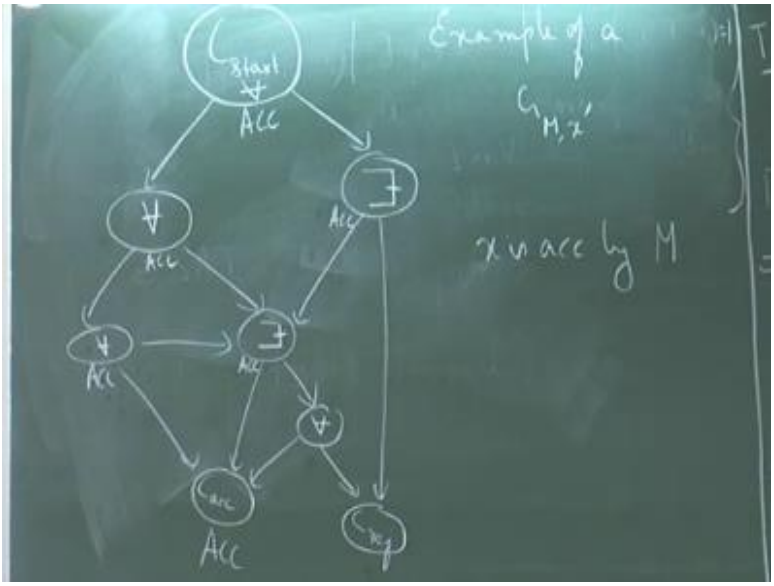
So, clearly these two problems are different problems and our intuition tells us that one is harder than the other. In fact, the former is harder. It is obviously much more difficult to come up with a proof than to verify whether proof is correct. Because to verify whether proof is correct, I just run through the steps and check that logically does a step follow from the previous step. But to

come up with a proof requires us to have some kind of ingenuity so that I know what the next step would be depending on the previous step.

So, how do we quantify this difference between these two intuitions? So, that is the reason why I mean that is one of the reasons why non-deterministic computations are studied. And I mean they capture so very nicely this difference between being able to compute something and being able to verify whether an answer is correct. So, let us get back to this alternating turing machine. So, as I was saying that now alternating turing machines are a generalization of the usual non-deterministic machine.

That is now we are not I mean it is not sufficient for us to check whether there exists a path from the start to the alternating the accepting configuration. We want that a certain sub graph of the configuration graph should exist with a certain property namely this. So, let us look at an example. Maybe that will present a better picture.

(Refer Slide Time: 28:12)



So, suppose this is we have our start configuration. And we go to two different configurations from here and let us say this is our accepting configuration. So, these are the two halting configurations. So, this is let us say C accept and this is C reject. So, now we need to put quantifiers on the various configurations to denote what label it has. So, let us say the start

configuration has label for all. This one again has label for all. This has a label there exists. This has a label for all there exists for all and that is it.

So, I am looking at the configuration graph with respect to a certain input. So, example of a G M, x so now what do you think? Will x be accepted by this machine or not? So, let us just trace the algorithm that we had. So, I have this edge. So, let us look at with respect to this. So, I have C accept so this does not yet have a label. So, let us look at with respect to this one first. So, this is for all vertex. So, I cannot label this accept because one of its thing is going to a non accepting state.

So, this gets a label accept initially. This does not get. What about this node? This does. Because it has one edge going to accepting label. Now what about this? This does. So, although this is a for all node both its children have an accepting label. So, similarly for this node also and what about this? Yes, because this is an existing, I mean this has a there exists label and it has one child which has a accepting label and same with this also.

So, therefore this x is accepted by M. Now let us just tweak a little bit with the configuration graph. So, let us consider another input. Maybe x prime which gives a for all label to this node. So, instead of this being so is everybody done with this example? I mean because every state has a label. So, we assume that we have a machine M where every state except for the accepting state and the rejecting state has a label there exist or for all. So, that is by definition. So, I am defining a machine which has this property. So, let me take a little bit time on this.

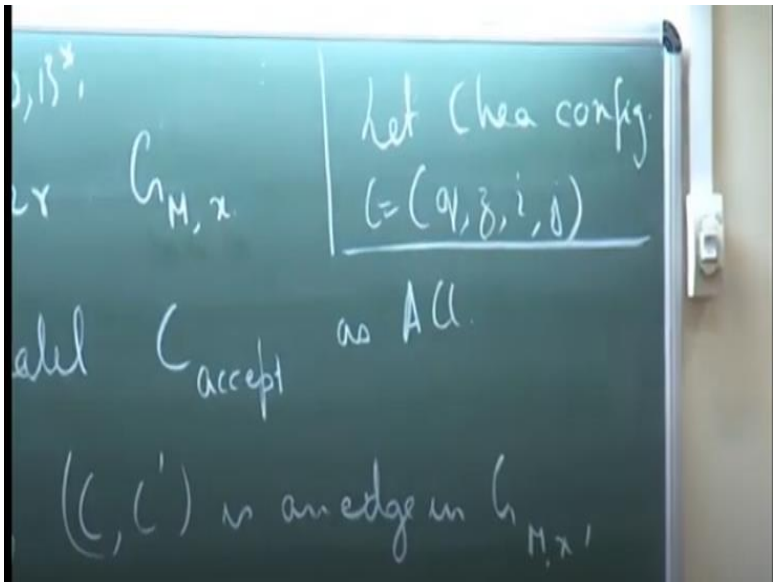
So, we are defining a model of computation. So, forget about the language, forget about strings and everything. So, what I am going to so what we are doing here is that we are defining a special kind of turing machine which I am claiming by definition is a generalization of the non-deterministic turing machine that we have seen so far. And what is the special property that this machine has is that every state of this machine must have a label there exist or for all except for the halting states. So, there is nothing to do with what input the machine accepts or not.

I mean you can also think of a general I mean of the usual non-deterministic turing machine that we have seen so far as a special case of this that every state basically has that there exist table. It is just a special case. So, now what we are saying is that so is it clear so far? So, yes let me come to that. So, far we just have a definition of a machine which has this property. Now what I am saying is that suppose you are given an input x. So, x is some string, some binary string.

So, we know that given x for a machine M whether it is a deterministic machine or a non-deterministic machine we can construct its configuration graph in a particular manner. So, that we know how to do, basically every configuration of this machine is on this particular input is a vertex and I have an edge from configuration C to C prime. If in one step of this machine on the input x I can go from C to C prime.

So, we know I mean we have seen this definition and we have been using it quite a bit. So, now what I am saying is that consider the configuration graph of M on x. So, let us look at some arbitrary configuration C.

(Refer Slide Time: 35:19)



So, let C be a configuration. So, what does it mean and what does C look like? C basically has some state q. There is some work tape content z, there is some index i which is the position at which the input head is at and there is also some position j at which the work tape head is at. So, these are kind of the elements of a configuration. So, look what is the label of this state q. So, I

know that according to this definition every state has a label. So, I just look what is the label of this state q and that is the label that I am associating with the configuration C.

So, that is what I meant when I said that there is a canonical way of associating labels to configuration depending on what state it is in. So, is this clear with everybody? And now here there can be many. I mean see there are many configurations which will have the same state. I mean maybe for the same state I can have a z and some other z prime. But that does not matter. I mean all I am saying is that forget about this portion of the configuration.

Only look at this portion of the configuration. Whatever is the label of that state I am just associating that label with the corresponding configuration. That is all I am saying. So, here of course it is input dependent. I mean depending on the input I get my configurations. But when I am defining the machine it is of course input independent. I mean the configuration graph is very much dependent on the input.

I mean so now you look at the configuration graph and this is basically the algorithm based on which you try to decide whether x is accepted by M or not. Again, this is an implicit algorithm and you do not the machine does not actually run this algorithm. I mean if this property is satisfied in other words if finally, C start gets the label accept then I would accept x otherwise I do not accept x.

In the same way when I say for the usual non-deterministic machine that if there is a path from the start configuration to the accept configuration, I accept x otherwise I do not. The certificate based definition is a there exist a certificate. And the translation basically is that path that you get from the start to the accepting configuration that path is basically the certificate that you provide to the input.

So, once you provide what that path will be together with the input you can just check with respect to the transition function whether it is accepting the input or not. So, now going to the definition of CO NP. In the CO NP we have for all labels. And the certificate based definition says that for all certificates it should accept the input. So, there the correspondence is that no

matter which path you take from the start to an accepting configuration that should accept the input.

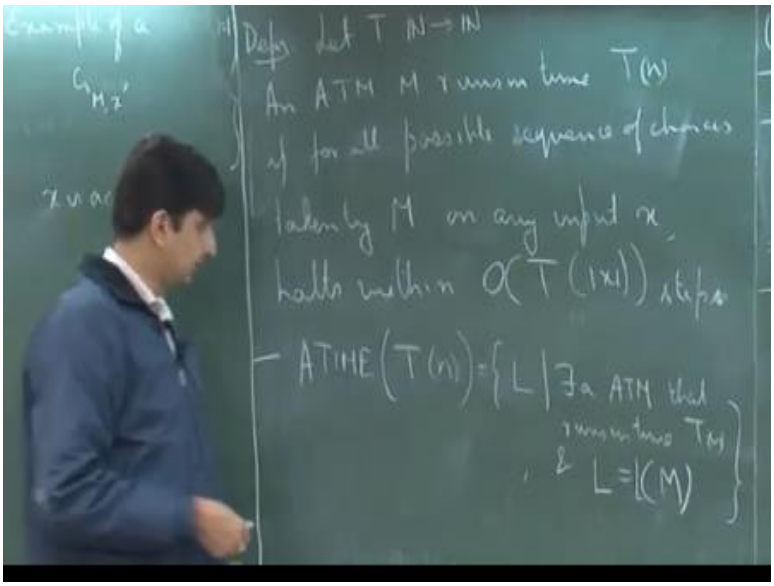
So, given any string you just check whether that corresponds to a path or not and if it corresponds to a path it should lead to an accepting path. So, that is the way of translating between this machine based definition and the certificate base definition. So, in other words certificates translate to paths in the configuration graph. Any other question? So, the other thing that I was planning is to give a non example.

So, let us consider a string x prime which gives a label let us say for all to this node. So, now I cannot any longer use these. So, now what can we say? So, this does not get a label accept since this is a for all node. Also, this also cannot get a label accept. This also cannot get label accept and therefore the start configuration also does not get a label accept. So, another way of viewing whether an input is accepted or not is to look at the configuration graph and check if there is a subgraph of this configuration graph which is constructed as follows.

So, for a for all node you include both its children into the sub graph. For a there exist node you include only one of its children into the subgraph. So, now if this subgraph has the property that the subgraph contains C start and C accept but not C reject, then you accept x otherwise not. So, that is an alternative way I mean that is the same thing. I mean you can prove that. So, it is the same thing as going through this recursive algorithm.

So, let us give some definitions standard definitions about alternating turing machines. So, is this clear to everybody what an alternating turing machine is and how is it a generalization of the standard non-deterministic turing machine? So, what do we have next?

(Refer Slide Time: 42:18)



So, we will define our time classes now. So, let T, be some function from N to N. We say that an alternating turing machine M runs in time T n. If for all possible sequence of choices taken by M on any input x halts within order T of x steps. So, no matter which sequence of steps you take for all possible sequence of choices on any input it must halt within T of x number of steps. So, if you look at the configuration graph what does that mean? Exactly.

The longest path in the configuration graph must be having length order T of x. Because any path corresponds to a possible sequence of choices. And now we can define the time class A TIME. So, we had D TIME, we had N TIME. Similarly, we can define A TIME T of n. So, it is the class of all languages L such that there exist an alternating turing machine that runs in time T of n and L is the language accepted by that machine.

So, for every string that is in the language the machine would accept it and for every string that is not in the language the machine rejects it. So, I will stop here today. We will look a little more about these classes and then we will move on next time.