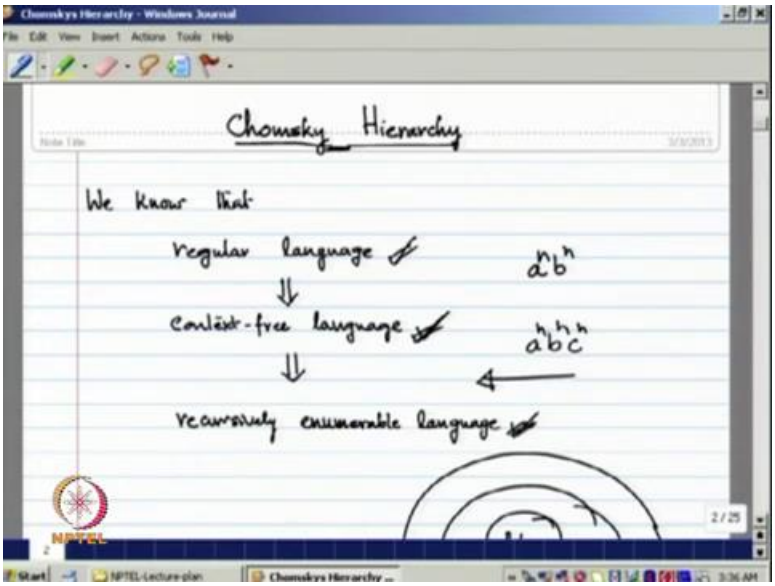
Formal Languages and Automata Theory
Prof. Diganta Goswami
Department of Computer and Engineering
Indian Institute of Technology, Guwahati

Module - 15
Chomsky Hierarchy
Lecture - 01
Chomsky Hierarchy

In today's lecture, we will discuss Chomsky Hierarchy.

(Refer Slide Time 00:34)



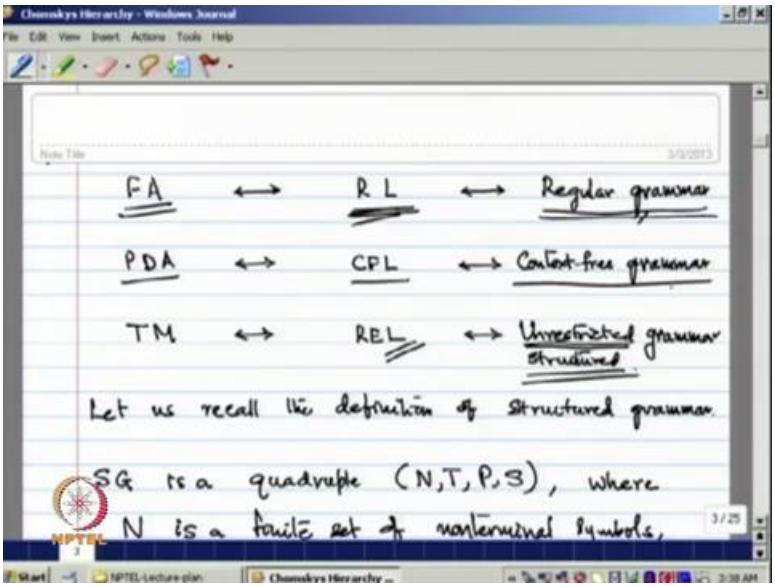
Already, we have seen a few classes of the languages: a regular language context free language and recursive language, and then recursively enumerable language. We have character, characterized all these class of languages grammatically, and also using automata. We have found that a regular language is a proposes set of context free language, because every regular language can be generated by using context free grammar. But there is a language like say, a to the n b to the n and greater than or equal to 0, which is not regular that we have already proved, but this is a context free language.

Similarly, every context free language is recursively enumerable language because every context free language can be generated by using some arms structure grammar and is accept by some turning machine, but there is a language a to the n, b to the n, c to the n, which is recursively enumerable. But not context free, so that way regular language is

properly contained within context free language, context free language is properly contained within recursively enumerable language.

Then, we will introduce some more clause in between, which is called context on language these four classes. These four classes of languages are said to be Chomsky Hierarchy, his name doctor the famous language known Chomsky, who proposed these languages as natural model for natural languages. So, this is confinement, we can show it regular language, it's properly contained within context free language, context free language properly contained within recursively enumerable language.

(Refer Slide Time 02:45)

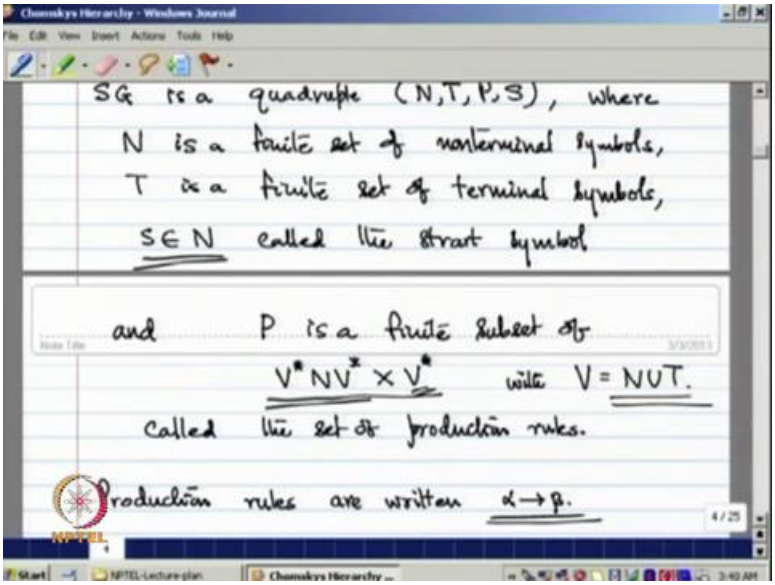


And we have introduced that regular right linear grammar generates regular language. That means, the regular language generated by regular language and final number accepts regular language and finite automata, and regular grammar. These are equivalent that we have already shown. Similarly push down automata and context free language context free grammar they are equivalent. So, p d a accepts context free language and a context free grammar generates a context free language.

Similarly, a turning machine accepts recursively enumerable language. And similarly, a recursive language enumerable language is generated by an unrestricted grammar or structure grammar, same thing. Now, let us recall that the definition of structure grammar, a structure grammar is a quadruple N T P S, where n is a finite set of non-

terminal symbols T is a finite set of terminal symbols, and S which is a non-terminal. It is called a start symbol…

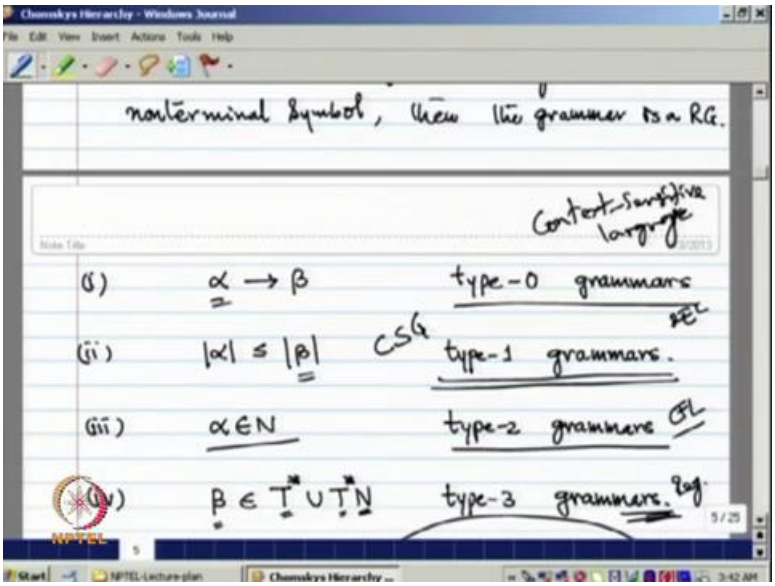
(Refer Slide Time 04:12)



And P is a finite subset of V n, V star n, V star cross V star, where V is basically union of non-terminal terminals is called sub set of production rules. That means production rules is written, basically by this notation, alpha goes to be beta over alpha is from is V star n V star, beta is from v star that means beta maybe epsilon. But the left hand side alpha must contain at least one non-terminal symbol. Now, if left hand side of each production is a non-terminal.

Simply, then we say that the grammar is a C F G. So, this structure grammar is said to be c f g, if the left hand side is a single non-terminal symbol in addition, if the right hand side of each production rule is a terminal string 1 by at most one non-terminal symbol. Then, the grammar is said to be a regular grammar.

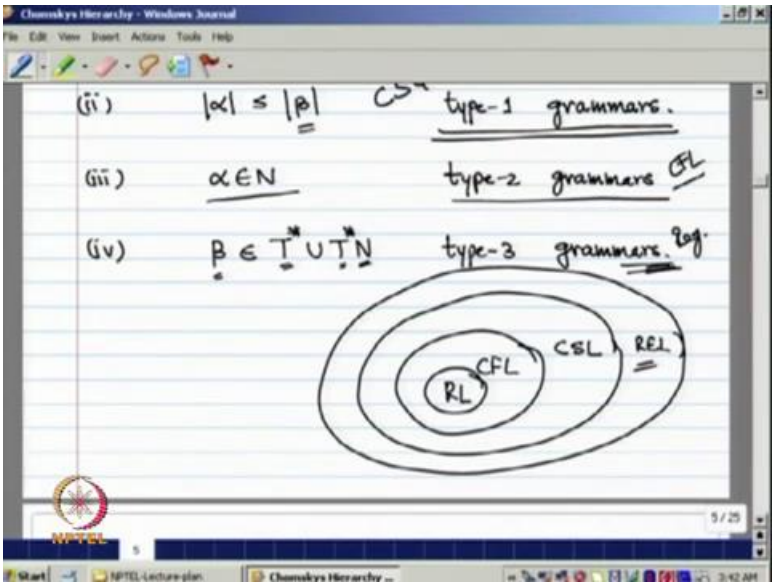
(Refer Slide Time 05:09)



So, based on this we can have a classification of grammar, so alpha goes to beta, if there is no restriction other than the rule that you have already given is said to be type 0 grammars, if we keep on imposing some restriction on both sides. Then, we get some other kind of grammar, if the length of alpha left hand side is less than or equal to the length of the right hand side.

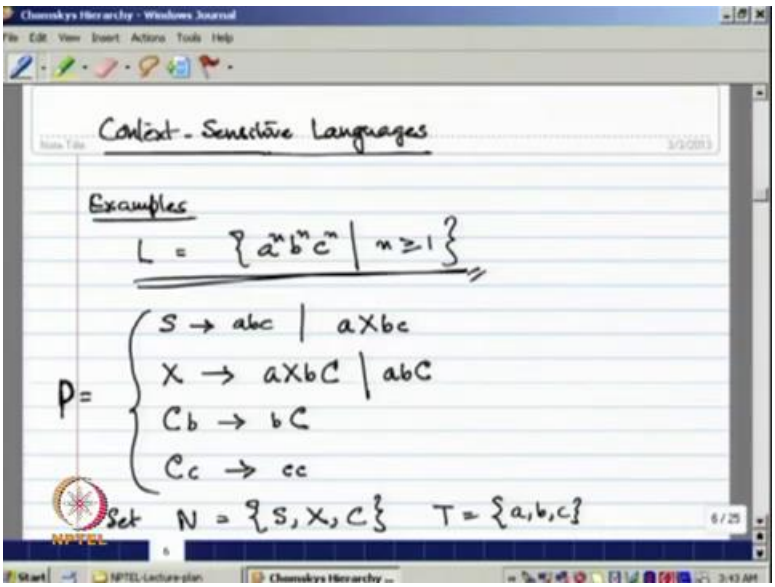
Then, we say that this grammar is type 1 grammar, and if the left hand side is single 1 terminal, it is called type 2 grammars or context free grammar, if beside this, if the right hand side beta is either the sequence of terminal symbols or a sequence of terminal symbols followed by a non-terminal, then we say that it is type 3 grammar. So, already we know that the type 3 grammar generates regular language, type 2 grammar generates context free language, type 1, type 0 grammar generates recursively enumerable language. But we had written mention, so far these type 1 grammar, we see that the type 1 grammar is nothing but it is called context sensitive grammar, and it generates context sensitive language context sensitive language. So, we will first introduce this context sensitive language. We will characterized this context sensitive language.

(Refer Slide Time 07:01)



Then, we will show that context sensitive language properly contain within this recursively enumerable language, in fact recursive language and there at least one context sensitive language, which is not context free. So, this shows the proper containment of C S L within recursive language, and C F L within context sensitive language.

(Refer Slide Time 07:24)

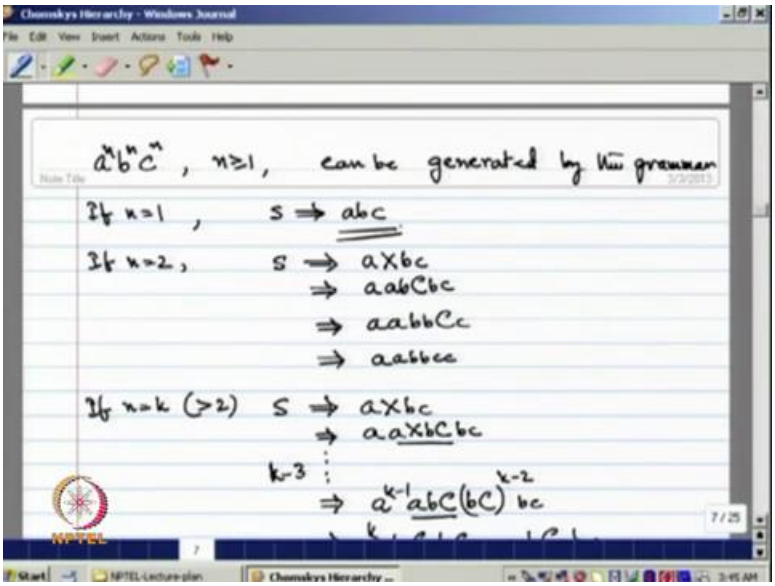


Just consider an example of context sensitive language say, this is the language L equal to, equal number of a 4 by number a 4 by same number of b 4 by same number of this,

where n is greater than or equal to 1. That means n equal to a to the n, b to the n, c to the n, n greater than or equal to 1, now we can generate this grammar using generate this language using the following production rules is a set of production rules. For the first one says that S goes to a b c S, is start symbol or a X, b c x is a non-terminal x goes to a x b capital C, capital C is non-terminal or a b capital C.

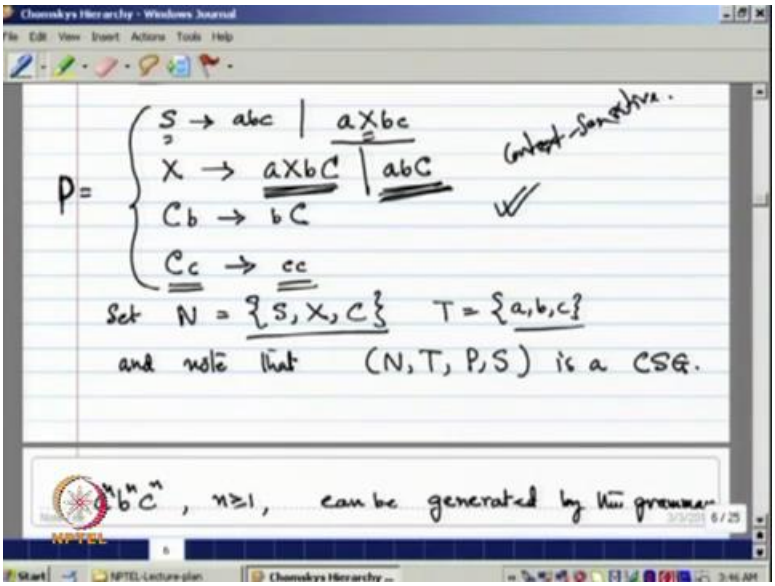
Then, C b goes to b C and C c goes to c c, now we can show that this grammar can generate this language, we know that this language is not a C F L, but this production rule set of production rules, which follows the restriction imposed by its context free language in no production rules. The left hand side length of left hand side is more than the length of right hand side. It is at most equal at this point C b equal goes to b C, but in other cases it is less than the length of the right hand side. So, it is basically the type one grammar, which is a context sensitive grammar, it is context sensitive grammar now. Therefore it is set of non-terminal S is S X C and a set of terminal symbol is a b c. So, it is quite clear that this is a context sensitive grammar C F G.

(Refer Slide Time 09:21)



Just have a look at how can derive the string of the form a to the n, b to the n, c to the n for some n greater than or equal to 1. So, if n equal to 1, we simply generate here is a first rule S goes to a b c to get a b c, if n equal to.

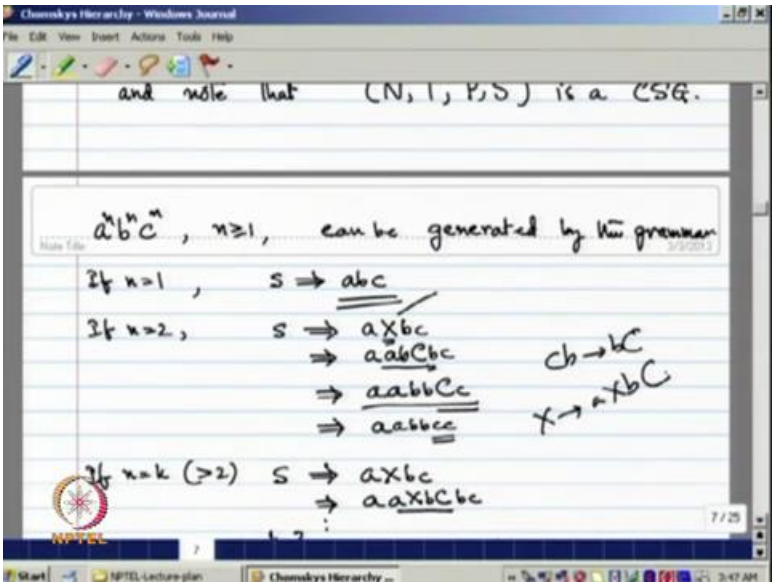
(Refer Slide Time 09:42)



We first use the second rule for S that means S goes to a X b c, and then this X will be terminated by one of these. We can be terminated by one of these, if we keep on continuing with terminal using this one. We will keep on generating more and more a S b, a S b and c and whenever we use this one, we stop there generating anymore a, and once we used this production S goes to a b capital C. Then, we keep on shifting the b, which is there to the right hand side. We will keep on shifting towards the left hand side.

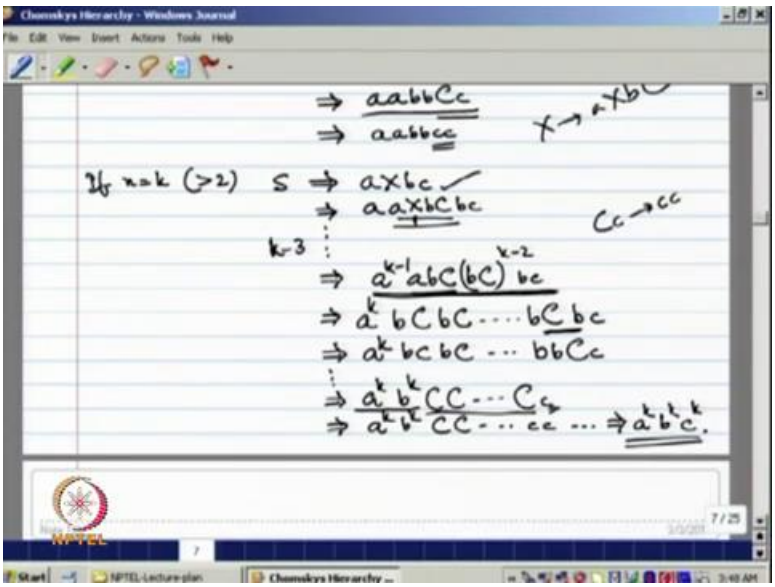
So, that all b appear immediately after the group of a, and eventually all capital C will go towards the, towards the n excrement. Eventually all capital C is using kind of rule capital C goes to C, capital C is C goes to C c small c c. We can use or we can have, we have converted all capital C is to small c.

(Refer Slide Time 10:49)



So, this is what you have done for n equal to a X b c, so this X is replaced by a X a b capital C. Now, using the rule c b goes to b c, we have shifted this b towards the left hand side by one symbol. And we have got this string and eventually capital C C goes to small c c, we have used the last step to get the c. Similarly, if you keep on continuing to use the production X goes to a X b c.

(Refer Slide Time 11:36)

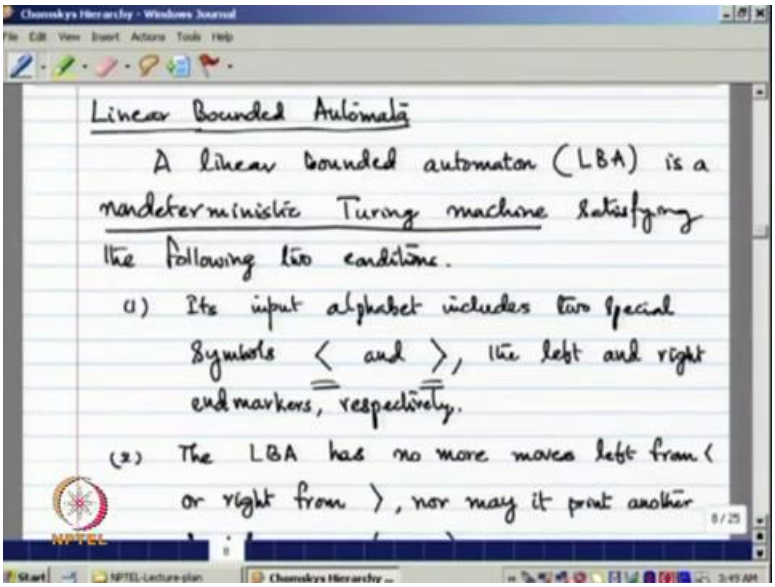


Then, we will have this kind of derivation S derives a X b c, then replace this x by a X b c keep on continuing to replace this X by a X b c. Then, after large number of steps, we

will have this kind of string a quadruple k minus 1 a b C b C quadruple k minus 2. Then, b C and eventually shifting this b in X in right towards the left hand side by using the rule C b goes to b C b capital C number of times.

We can shift all the b towards the left hand side getting a power k b power k. And then k numbers of k minus 1 numbers of capital C and small c at excrement, and eventually using the rule capital C small c goes to small c, small c k numbers of step k numbers of times. We eventually get this string a quadruple k b quadruple k c quadruple k. So, this grammar generates the language a to the n, b to the n, c to the n.

(Refer Slide Time 12:53)

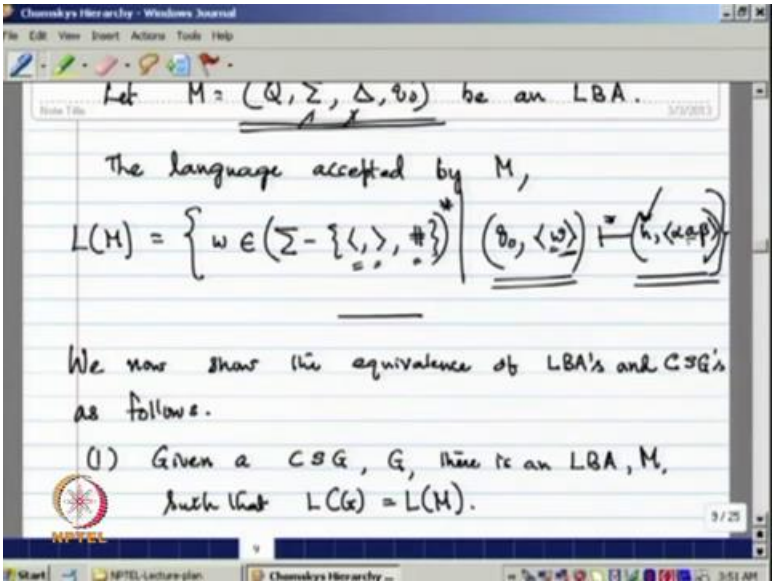


So, this is an example of context free grammar, we will introduce kind of automata called linear bounded automata. And we will show that the linear bounded automata accept context free languages. Now, linear bounded automaton is a nondeterministic turing machine satisfying the following conditions: the first one is that its input alphabet includes two special symbols: one is left angular bracket. And the other is right angular bracket the left and right.

There are left and right end markers respectively, the L B A has no moves, no more moves left from the left end marker and or right from the right towards the right end marker nor may it print another symbol over left end marker, right end marker. That means left end marker and right end marker cannot be erased or cannot be replaced by

any other symbol, and head readied head cannot move left towards left end marker. And right towards the right end marker.

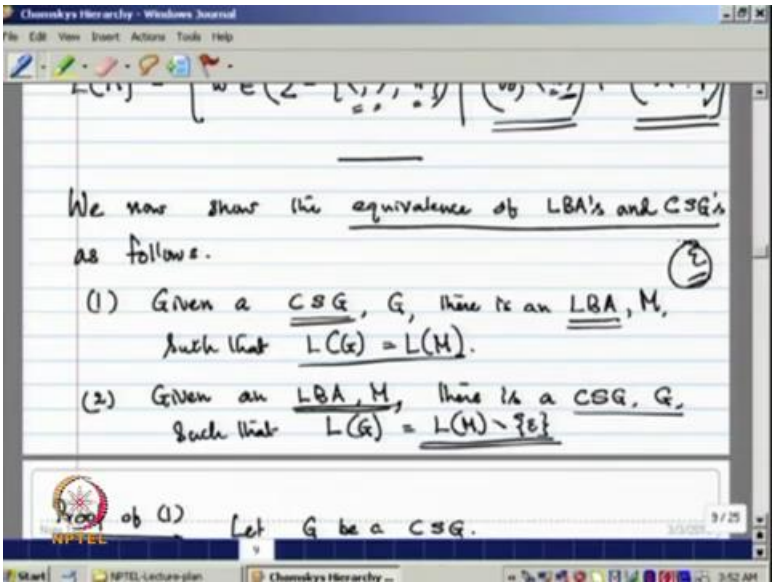
(Refer Slide Time 14:17)



Now, consider a linear bounded a linear bounded automata automaton M, where Q is a set of states and input alphabet transition moves and q 0 is the start state. Say, this is an L B A linear bounded automaton, the language accepted by this linear bounded automaton M. The set of all strings w over sigma other than, what which does not contain left end marker, right end marker or head symbol such that it starts with this configuration initially. That means it starts with the start symbol q naught. The input w is placed between the left end marker and right end marker and the head readied head is reading the right end marker.

Initially, eventually from this, if it moves to this configuration, where it enters in a state to its halting state. And in that case that outcome maybe anything, it maybe may have any content within this left angular bracket and right angular bracket alphabet at any strings over string of terminals or any non-terminals. Then, it reading some symbol a since it has eventually entered the halt state for this string will be w will be accepted. We will now show that the L B A linear bounded automaton automata and context sensitive grammars are equivalent. This equivalence is shown as follows…

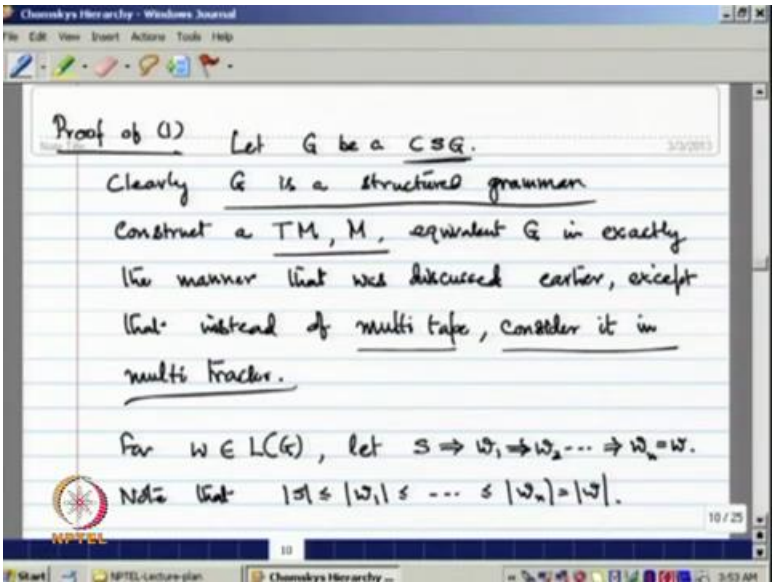
(Refer Slide Time 16:15)



Given a C S G context free grammar context sensitive grammar G. There is an L B A linear bounded automaton a such that L G equal to L M. Similarly, given an linear bounded automaton M, there is a C S G, G such that L G equal to L M minus epsilon. So, here since the L B A might accept epsilon, but since in the C S G the left hand side of the grammar is less than length of left hand side of grammar is less than or equal to the right hand side.

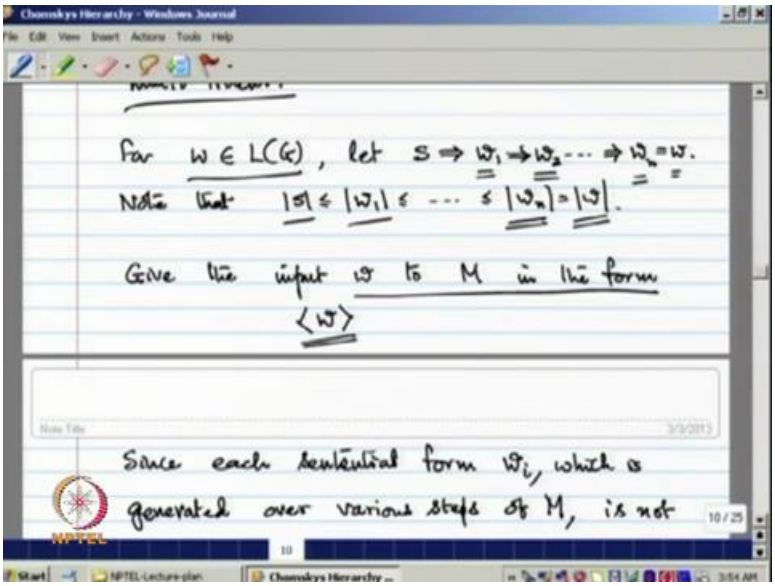
Therefore, this context sensitive grammar cannot generate epsilon, so epsilon cannot be generated by context sensitive grammar. Therefore, it will be equivalent in that sense that it generates any string accepts by the linear bounded automaton except for this string epsilon.

(Refer Slide Time 17:28)



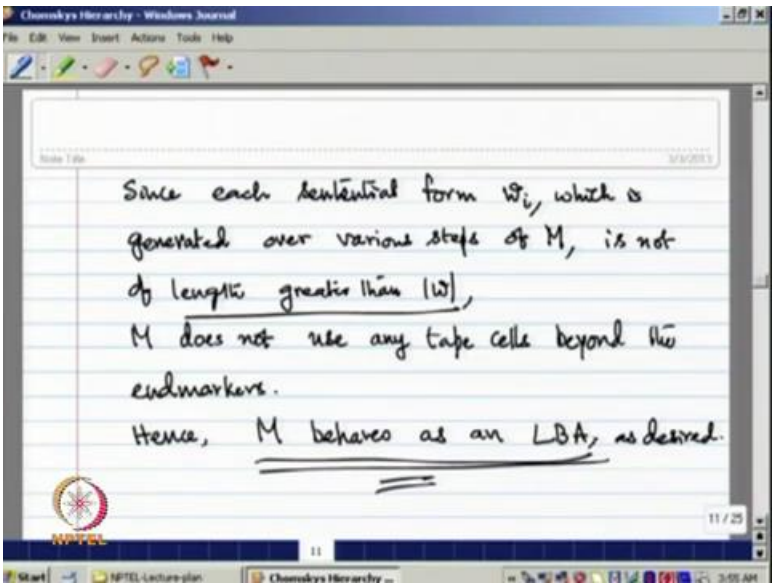
Now, we will first show that, if there is a C S G, there is equivalent linear bounded automaton M accepting the same language generated by the grammar G. Now, clearly G is a structure grammar special case of structure grammar. Now, we will construct turing machine M equivalent to G in exactly the manner. That was discussed earlier except that earlier that in the in the context of showing that a turing machine and structure grammar are equivalent already. We have shown that turing machine and structure grammar are equivalent in that context whatever construction we used. We will use a same construction except that instead of multi tape; consider it, in the multi track, in such a case.

(Refer Slide Time 28:30)



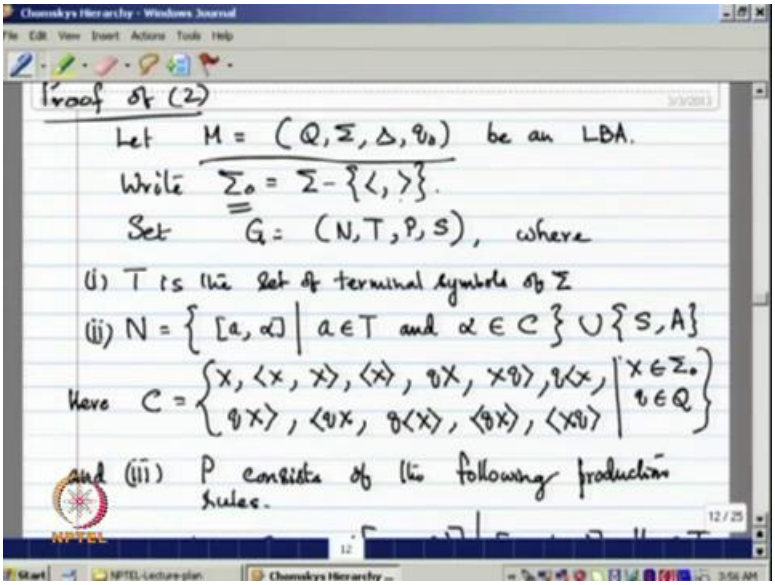
Suppose, w belongs to L of G, then there is a derivation like this, S derives w 1, w 1 derives w 2 eventually w n minus 1 derives w n, which is equal to w. And in every steps since, we are using the production rules of the context, of the context sensitive grammar. Therefore, the length of S must be less than or equal to length of w 1 because of the restriction that we have for context sensitive grammar and length of w n is greater than length w n minus 1 and length of w n is basically w. Now, give the input w to linear bounded automata in the form w, within this left end marker and the right end marker.

(Refer Slide Time 19:20)



Since each sentential form w i, which is generated over various steps of M is not of length greater than w, that greater than length of w M does not use any tape cells beyond the end marker. So, this is for sure because in every step the length of w or w i, which not greater than the length of the final string that will be generated. Therefore, at in no step the deterministic, we will use any cell towards the left side of the end marker or towards the left side of left end marker or right side of the eight end marker. Therefore, M behaves as a linear bounded automaton as desired, therefore M is an L B A, which accepts the language generated by the C S G, G. Let us now see, the other derivation that means, if there is an L B A…

(Refer Slide Time 20:22)



M accepting some language, say L M. There has to be an equivalent C S G except accepting the same generated in a same language. And that is your L G that means L M equal to L G, consider that the linear bounded automaton is M with the elements Q sigma delta and q naught. We write sigma naught to be sigma minus, these two symbols that means sigma naught does not contain that 2 end markers. Now, construct grammar, which is, which is a required context sensitive grammar from this L B A.

You will see that T is a set of terminal symbols of sigma, whatever terminal symbols is there in sigma will have in T and N, is a set of non-terminal S. We have 2 special non 2 non-terminal S S and a beside that we will have, we use some composite symbols to represent the non-terminal S A composite symbols of the form a alpha is a pair.

Basically, within square bracket a alpha, where a is a terminal symbol, and alpha is from this set C. So, it maybe X for X in sigma 0 or it may contain angular bracket, left angular bracket.

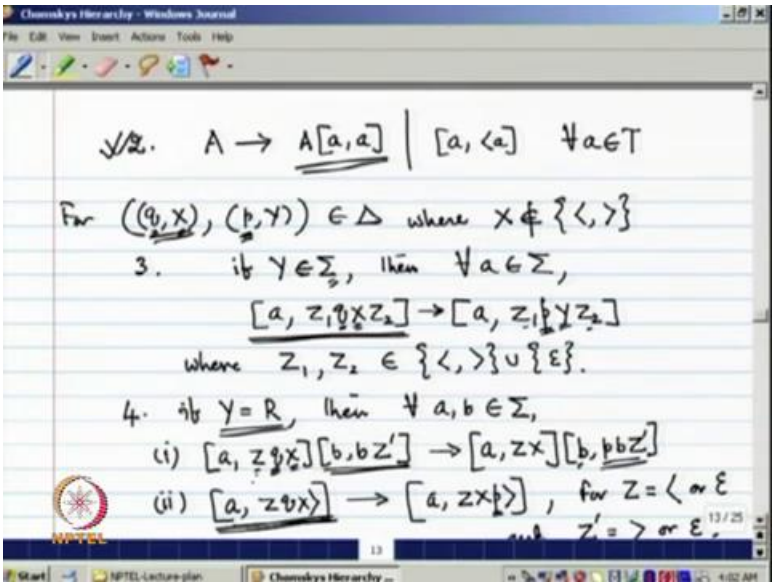
Then, X or X left angular bracket or it may contain both the angular bracket. And X, X maybe any symbol from sigma naught or it may contain some states symbol q q belongs to capital Q and it may contain both states symbol, and angular bracket along with this X. So, C maybe anyone from this set alpha maybe anyone from this set C and P consists of the set of the production rule, consists of the following productions. We will now see what the production rules to be used?

So, first production rule is that S goes to capital A and then this composite symbol a a q naught, a comma a q naught or S goes to the single symbol non-terminal. That is a comma within bracket with bracket a q naught. Here, q is a terminal symbol and this a for I can generate this kind strings A a, a and a left bracket a, in fact using this 1 and 2, we can generate the input string.

Here, once we generate a string using S, and a the first component, if you collect the first component from each of the composite symbols. That will give us the exactly input string and the second component, basically represents the tape con component, and while deriving during derivation this grammar will simply simulate the moves of that turing machine linear bounded automaton M. Suppose, that the automaton contains A move like this delta q x equal to p Y. It is in state q, if it reach a symbol X, then it goes to state p, and Y may be a print or Y maybe lefts left move or Y may be a right move, if Y is a print from some symbol from input alphabet a prints.

Then, what is done? If, currently suppose that turing machine or linear bounded automaton are using symbol x, it was in state q reading symbol x state. That is, what given by this move. Then, it simply sends the state from q to p and x is sends to Y. That means, it prints a new symbol Y. Here Z 1 and Z 2 maybe any symbol, it maybe epsilon or it may be left angular bracket or a right angular bracket. Therefore, Z 1 and Z 2 will remain same.

(Refer Slide Time 25:21)

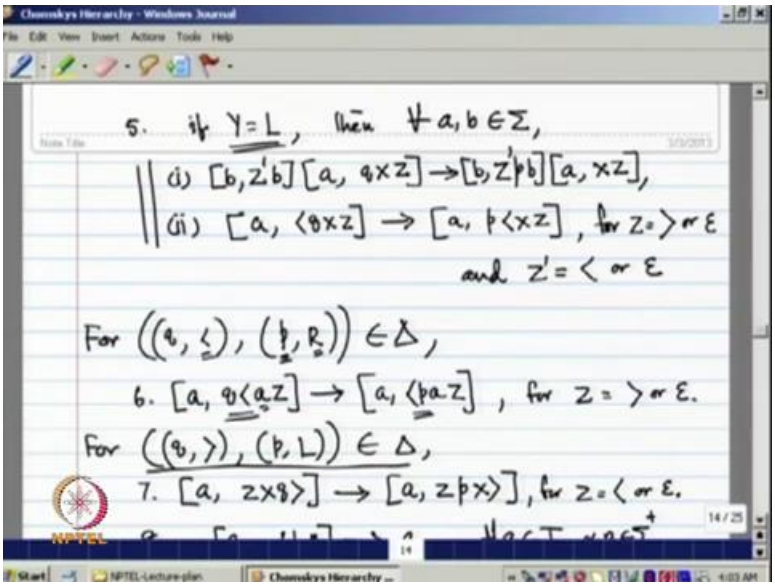


Now, if suppose that turing machine or the linear bounded automaton in state q reading X goes to state p. And it moves towards the right hand side, head is move towards the right hand side that means Y equal to R in such a case.

Suppose, this is a last this represents, this is a last symbol; last non-terminal in such a case, this angular bracket will appear at the end. Therefore, it will simply reading q reading terminal symbol X at state q. It will simply go to the right side of the symbol X. That means, it is now reading the right end marker p. Otherwise, if this is not the last symbol, then there will be some other symbol towards the right hand side.

So, it is in state q read symbol X, then it will simply move over the symbol X. That means, it will read the next symbol. Therefore, it will go to the symbol b, so from a Z q X, next symbol is b Z dash, where Z maybe any right angular bracket or epsilon, it may be empty also, just as symbol q. It will move towards the a right side to indicate that it is now reading the next symbol b. So, it is P b Z dash Z dash maybe empty, and here this Z maybe the left angular bracket or it may be empty as well. Similarly, you can see the moves when it goes towards the left side. These are exactly similar to the previous 2, and a special case, when it is reading the left end marker.

(Refer Slide Time 27:22)

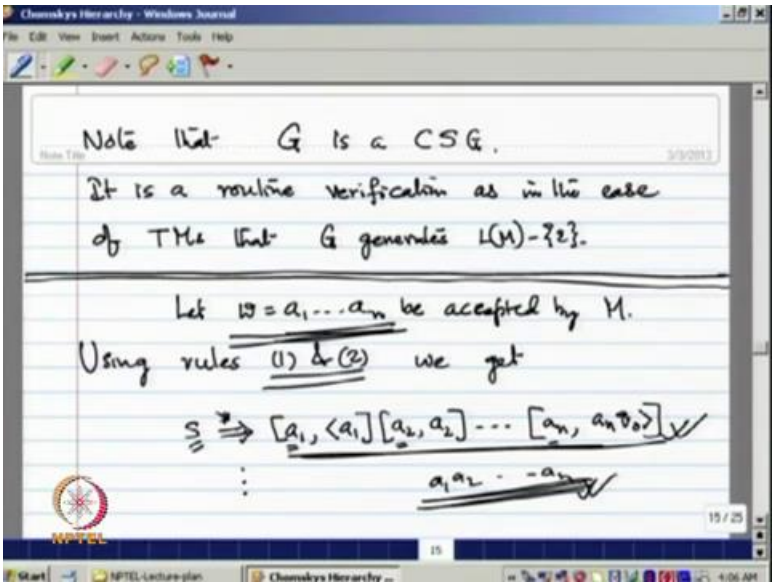


It simply goes towards the right side, it moves towards right side, so this is right side. And it maybe changes a state to p. So, a q left angular bracket a Z, it is reading the left end marker. So, there is no move towards the left side, it has to move towards the right side, so that can now read the symbol a. Similarly, we can write the rule for, when the L B A is reading the right end marker in such a case. Of course, there will be no right move, so it has to go towards the left side to read the symbol, which is their towards the left of right end marker.

Finally, when the L B A enters in a state, which is a halting state is h, is a halting state. Then, simply we reduce this composite symbol to a, where a is a symbol that is there towards the right side. Then, from then onward we, can reduce this combination some composite symbol and terminal symbol to a b. Say, a comma alpha b will be; now a b taking the first terminal symbol and these are the terminal symbols.

Similarly, b a alpha will be converted to b a, so we see that each of the production is follows the rules of a context rules of context sensitive grammar. That means type 1 grammar. So, this grammar that we have constructed from the moves of turing machine is exactly is a context sensitive grammar. Now, it is a routine verification as in the case of turing machines that this grammar G generates the language accept by linear bounded automaton except for epsilon.

(Refer Slide Time 29:17)

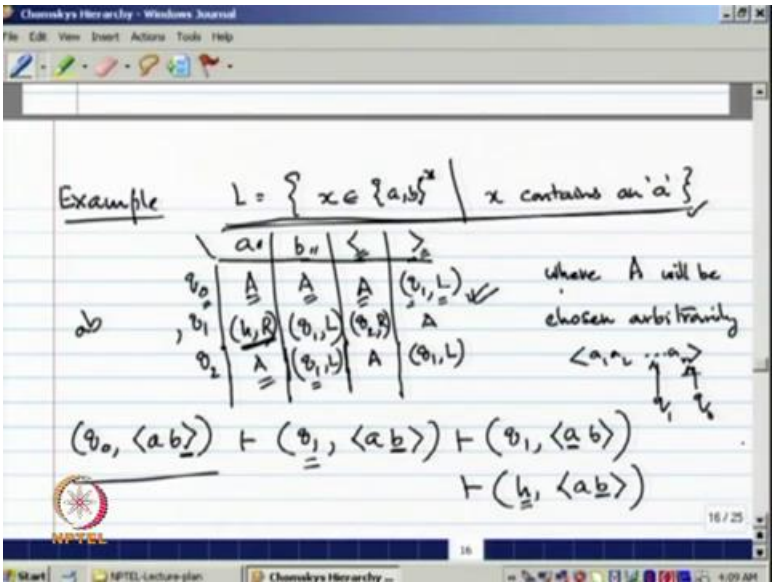


Suppose, W is a string which is a 1 a 2 up to a n accepted by the automaton M. So, using rules 1 and 2 we can simply generate in a few steps square bracket a 1 comma left angular bracket a 1. Then, square bracket a 2, a 2 square bracket a 3, a 3. Finally, square bracket a n comma a n q naught left angular bracket.

You see that, if you collect the first symbols from every composite symbol, it constitutes the string a 1 a 2 up to a n, which is the input string a 1 to a n. And then we can use the other symbols following the moves of the grammar, we can use other rules of the grammar eventually to generate the string eventually; this can be converted or transformed to this string.

So, we can transform this string from side, we will get generate this string. Eventually, we can transform it to this string provided that turing machine M accepts this particular string. That means we can have, we have some moves to accept this string by linear bounded automaton, we will illustrate this by an example.

(Refer Slide Time 30:44)



Say, this is an linear bounded automaton, which generates accepts all string containing an a. Any string containing a will be accepted by this linear bounded automaton, which is shown in that table the moves of the linear bounded automaton is shown in that table. So, q 0 q 1 q 2 order states of the linear bounded automaton a b are terminal symbols. And left angular bracket and right angular bracket are two special symbols that we have at left end marker, and right end marker.

Here, in this capital a shows some arbitrary moves, you can choose anything there, because that situation does not arise since q naught in the start state. It always read the left right angular bracket, so in such a case we will enter in a state q 1, and go to the left side because the string will be given like this a 1 a 2 up to say, a n within this. We will be reading this left end marker at state q 0.

Therefore, first we have to move towards the left side. That means we will move towards this side and enter in state q 1. Now, we will read the symbol a n that is the first move and once we at there in the state q 1, we will keep on skipping the symbols until we see n a. Once we see n a, if at q 1 we see n a, we enter in the halting state. We move towards the right side or left side it does not matter, but if you see any other symbol b, other symbol b, then we are in the same state q 1, but moves towards the left side.

We keep on skipping symbols, but if we see the left angular bracket, then we enter in a state q 2 and move towards the right. And in q 2 again, if we see n a, we enter in a

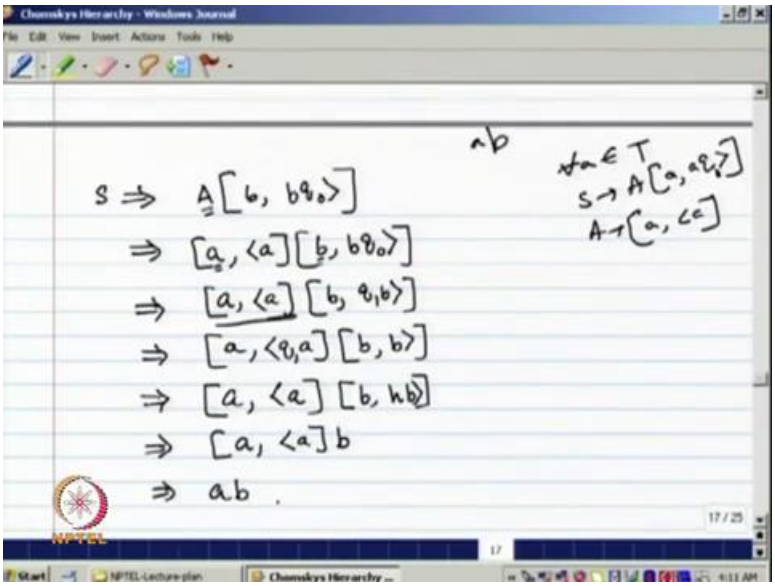
arbitrary state because that situation does not arise. Otherwise we have to enter in a halting state earlier in the state q 1, and if in state q 2. We see, a b you enter in a state q 1 and move towards the left side, therefore at that point it will keep oscillating.

So, just consider a move of turing machine, it starts input string is a b whether the turing machines are indeed, which in L B A whether it accepts a b or not. So, q naught a b is that 2 end markers currently reading. This indicates the currently reading symbol right angular bracket. So, in the next move following this one, q 0 right angular brackets goes to q 1 L. So, it will enter in a state q 1 and it will move towards the left hand side, head will towards the left hand side, so it is now reading b.

So, on q 1 b, q 1 b it will again skip and move towards the left hand side. The head moves towards the left hand side, startle remains same. Now, q 1 a, q 1 a it will enter in a halting state, it halting state, it will move towards the right hand side. So, since now it has not entered in a halting state, this string a b is accepted by the linear bounded automaton. Now, let us see whether this C S G, which is constructed using the rules of grammar, generates the string a b or not.

So, the if we apply the first rule, so S derives A b b q naught for every for all a belong to T. We have this kind of rule S derives a S goes to A small a a q naught. This is rule, we can apply and then this a goes to a a, this rule can be used in the next step to terminate the non-terminal A. So, once we have this, we see that the first the symbol a b represent the input string a b, and the others the second quart can be used to simulate the tape of the L B A.

(Refer Slide Time 35:13)



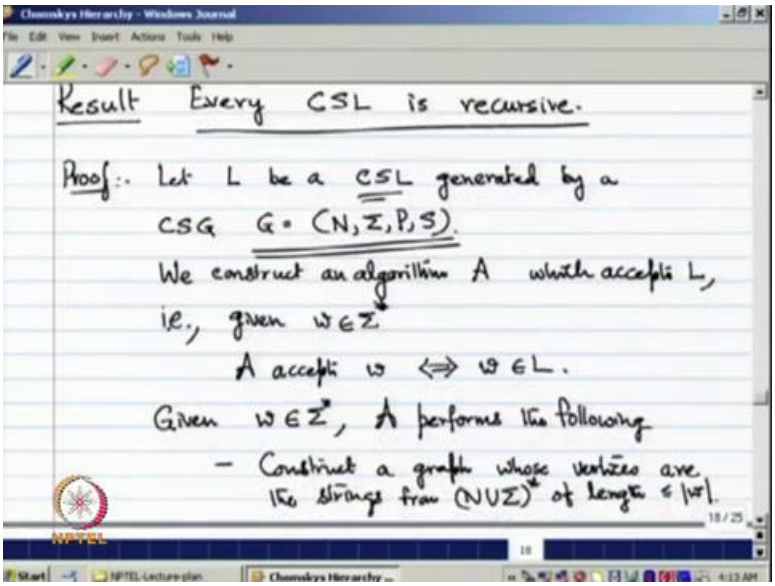
Now, this q 0 right angular brackets, in this case you simply move towards the left hand side following our rules of the move of the turing machine q 0 right angular bracket. It will move towards the left hand side. So, we will use the rules of the grammar corresponding to the rules of the grammar, so this is b q 0. And then q 1 b in this step since q 1 b, we have this rule q 1 b, it goes to towards the left side. Therefore and it remains a same set q 1.

So, in this case, so since there is no way to go, here in the left side. So, this q 1 will move towards the left this symbols, towards the left side composite symbol towards left side. So, to have to read the symbol a that means it will be now left angular bracket q 1 a. And similarly, now q 1 a since we have the move q 1 a, q 1 a is halting state in r. Therefore, in the corresponding rule in the grammar C S G will be according to our construction, so q 1 a will go to the right side and to read out symbol b, and on the halting state, so this will, this will a new station.

So, since it has entered in the halting state. So, this now will reduce to b according to the rule, this rule that we have so according to the rule X, rule on the context free context sensitive grammar. So, whenever we have halting state, it will reduce to the symbol that is there towards, which is the left component. Therefore, in this case, so this will reduce to b. So, that is how we have used over here, and now b allows this. Now, we will reduce to a b, so this is a b. Therefore, S derives a b, so this is how this corresponding grammar

generates the strings. Therefore, we have shown that context sensitive language context, sensitive grammar and linear bounded automaton automata are equivalent. And the language generated by context sensitive grammar, is said to be context sensitive language.

(Refer Slide Time 37:47)



Now, we will show that context sensitive language is properly contain within recursive language first. We will show that every context sensitive language is recursive. Let L be a context sensitive language is generated by the grammar context sensitive grammar G. We will construct an algorithm A, which accepts L that is given any string w from sigma star A accepts w, if and only if w belongs to the language L.

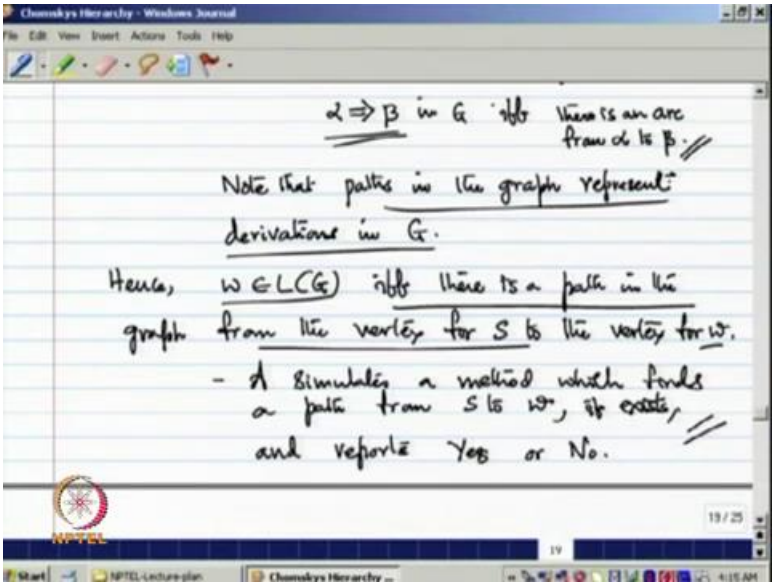
So, it goes to like this given w belongs to sigma star the algorithm performs the following steps. It constructs a graph whose vertices are the strings from n union sigma star whose length is less than or equal to w, length is less than w. So, here the vertices represents the strings over n union sigma star, but the length will be must be less than or equal to the length of the string w.

Now, for vertices say, alpha and beta are vertices of the graph, it can show. We know that alpha; there is beta in the grammar G, if and only if there is an arc from alpha to beta. So, this is quite easy or you can just see that alpha. Now, A simulates the method, which finds the path from S to w, if exists and there algorithms, we can do it easily to find out the path from one vertex to another vertex in a graph. And it reports yes, if there

exists any such path to indicate that S derives w over means w is actually generated by the grammar G.

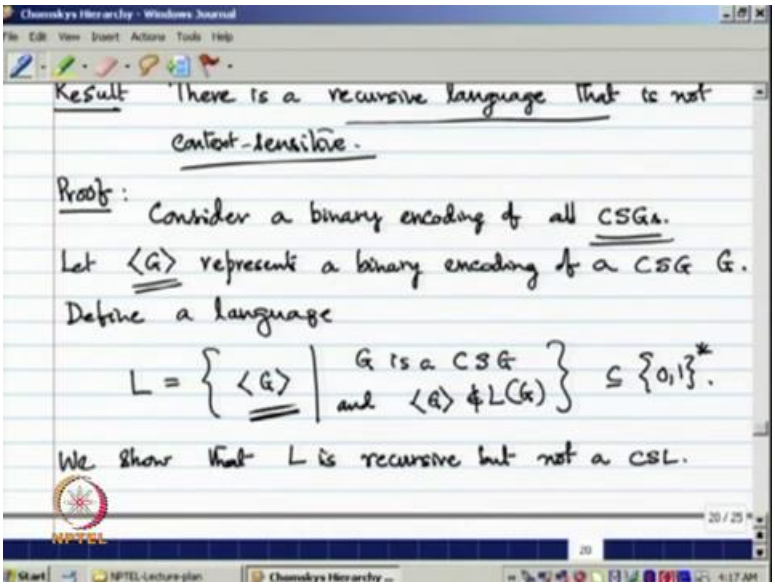
There is beta, if and only if there is an arc. There is an arc from alpha to beta, now that the paths in the graph represent derivation in G. Therefore, w belongs to l of G, if and only if there is a path in the graph from vertex S to the vertex for w. So, if you can find somehow find out a path from the vertex representing S to the vertex representing w. Then, we know that S derives w.

(Refer Slide Time 40:27)



Otherwise, it says that no it is exist. Therefore, we know that there exists such an algorithm and hence every C S L can be shown to be recursive, now we will show that there is a recursively language. That is not context sensitive that show that proper containment, whose shows that there is a recursively language that is not context sensitive.

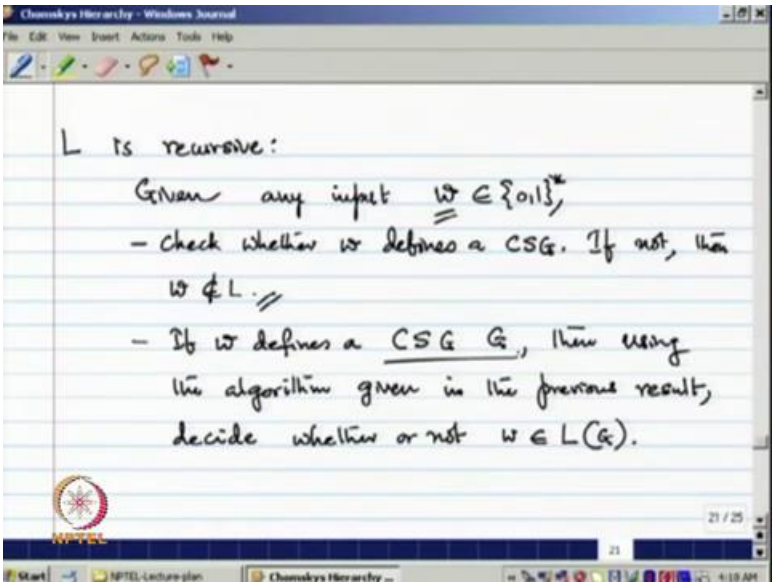
(Refer Slide Time 41:02)



To show this, we will consider a binary encoding binary encodings of all C S G context sensitive grammars, we will just consider some binary encoding. Suppose, this G in angular bracket represents the binary encoding of a C S G, now we define the language L like this L is a set of all binary encodings of all the C S Gs within G. We will make it a binary encoding binary encodings of all the C S G s such C S G. And this encoding is not accepted or not generated by the grammar G, now this L. Of course, it belongs to 0 1 star because this is a binary encoding.

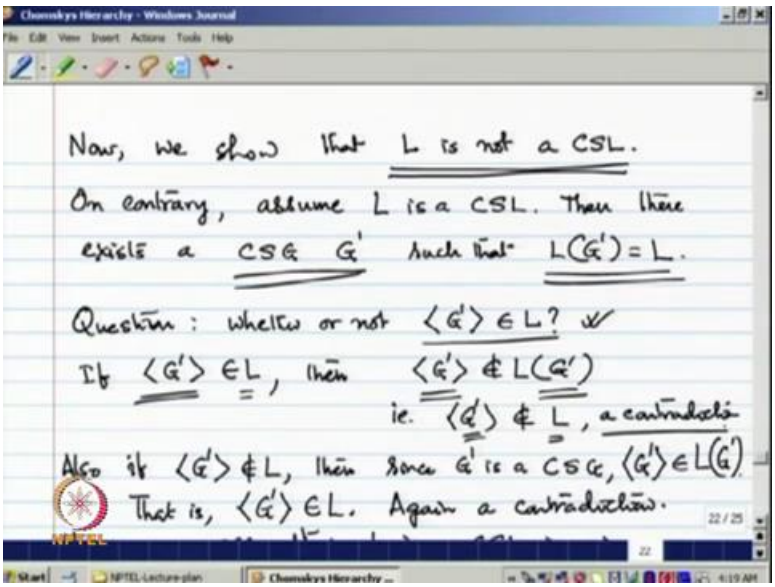
So, here G must be a C S G, and this is the binary encoding and this is not accepted by or generated by the grammar G that means this encoding does not belong to L G. Now, we will show that L is recursive this language, L is recursive, but it is not a context sensitive language.

(Refer Slide Time 42:31)



To show that L is recursive, if any inputs is given. Say, any w belongs to say, 0 1 star is given, we will see whether w defines the C S G, if it does not define C S G. Obviously, w does not belong to L that is quite clear now, if w defines the C S G, then using the algorithm just you have described above in the given the previous result. We decide whether or not w belongs to L of G. So, that we can always do by constructing the graph, and then looking for a path, if there exists. Therefore, we can show that L is recursive because we can decide whether or not w belongs to L G.

(Refer Slide Time 43:22)

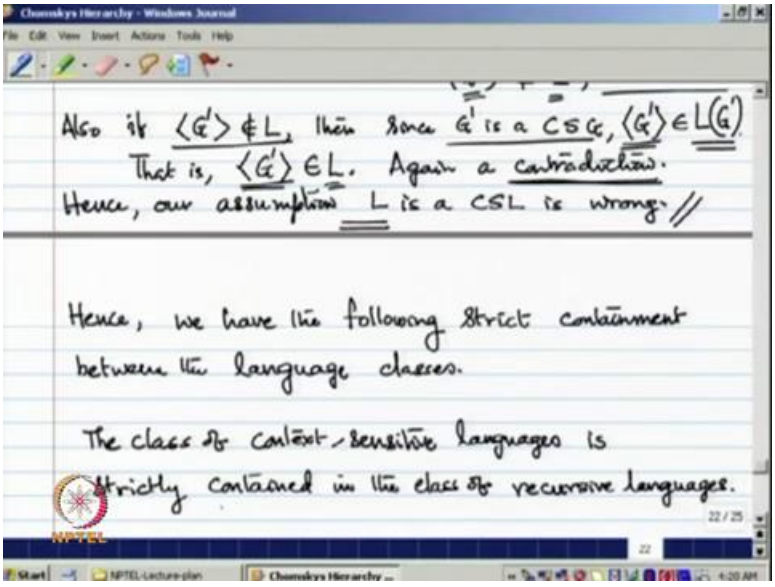


Now, we show that L is not a context sensitive language. So, assume for contradiction that L is context sensitive language. Then, there exists a context sensitive grammar. Say, G dash such that L G dash equal to l. There must be some context free grammar, now we pose a question equation whether the encoding of that grammar G dash belongs to l. This is a question that we asked.

Suppose, that this encoding belongs to l, then according to our definition the this encoding does not belong to L of G dash. Therefore, this encoding does not belong to L, so this is a contradiction that we have got. Similarly, if we assume that this encoding belongs to l, encoding of this grammar that we have assumed belongs to l. Then, since G dash is a C S G, then encoding of this belongs to L of G. That is encoding of this belongs to L, again we have arrived at contradiction by this argument.

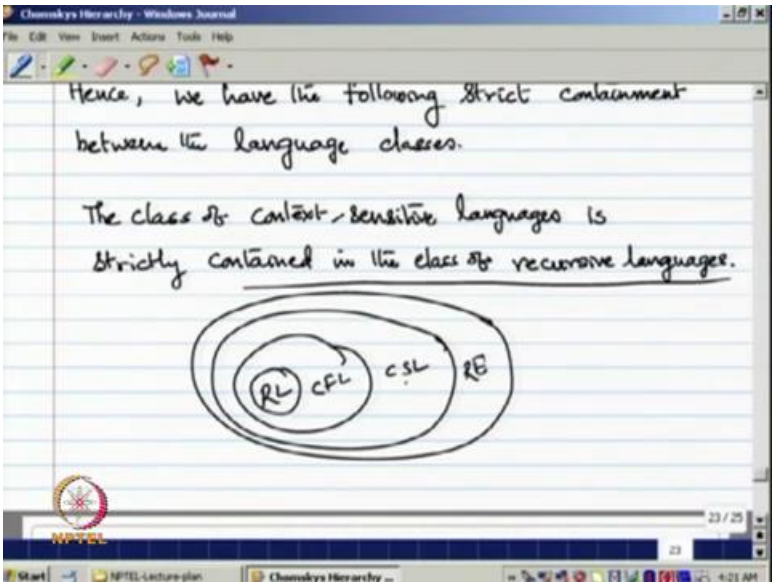
Therefore, our original assumption that L is a C S L must be wrong. Therefore, such a grammar context sensitive grammar generating L does not exists. Therefore, it is not context sensitive language. So, hence l is a there exists, I mean to say, L is a L is a recursive, but it is not context sensitive.

(Refer Slide Time 45:10)



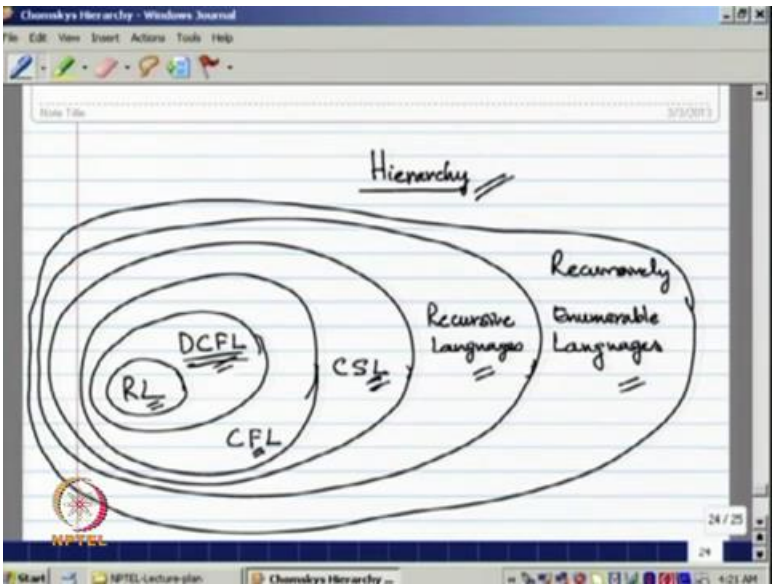
So, hence we have the following strict containment between the, between the language classes. The class of context sensitive language is strictly contained in the class of recursive languages and hence we have got this hierarchy. In fact, we have some other classes. Say, so the first step I have got this…

(Refer Slide Time 45:35)



Containment say, regular language is properly contained within context free language, context sensitive language is properly contained. And C F L is properly contained in the context sensitive language, and C S L is properly contained within recursively enumerable language. So, in fact we have some other classes.

(Refer Slide Time 46:00)



For example, this C F L deterministic context free language, so regular language is properly contained within deterministic context free language that results. Also we have shown earlier to determine context free language purposes set of context free language,

context free language purposes set of context sensitive language, and context sensitive grammar is purposes set of recursive language. In fact recursive language is purposes set of recursively enumerable language. So, these are hierarchy, which is called Chomsky Hierarchy.