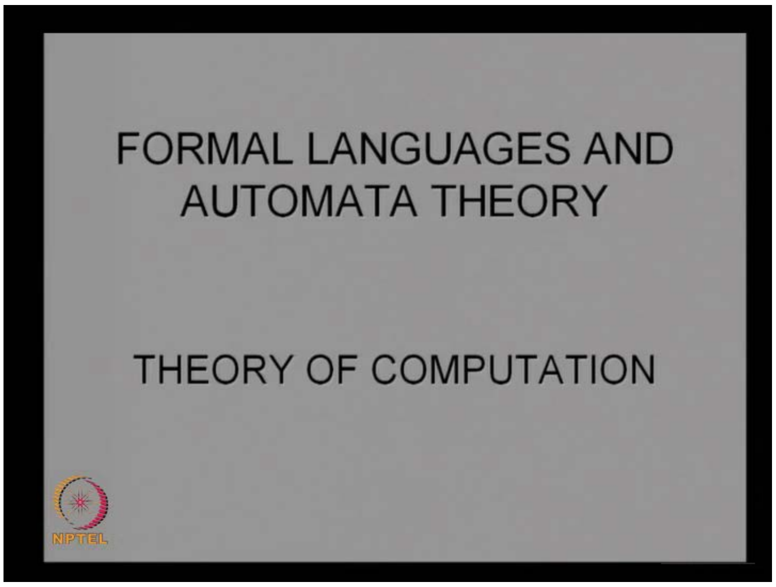
Theory of Computation

Prof. Kamala Krithivasan
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture No. # 01
Grammars and Natural Language Processing

(Refer Slide Time: 00:11)

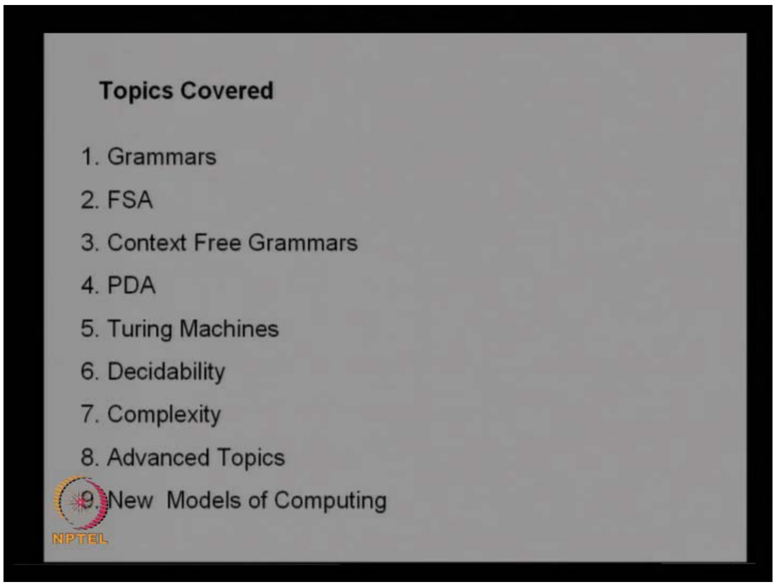


FORMAL LANGUAGES AND AUTOMATA THEORY

THEORY OF COMPUTATION

The subject we are going to learn is called Formal languages and Automata theory. It is also called theory of computation. It is a core course for B Tech or BE in computer science and engineering in many of the universities. In some universities it is also a course for B Tech IT, in some places in mathematics department also it is a course for M SC mathematics.

(Refer Slide Time: 00:43)

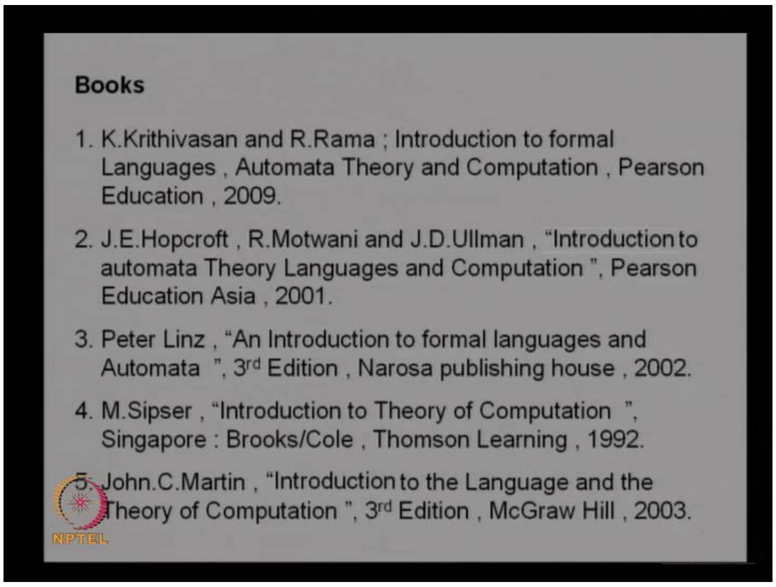


Topics Covered

1. Grammars
2. FSA
3. Context Free Grammars
4. PDA
5. Turing Machines
6. Decidability
7. Complexity
8. Advanced Topics
9. New Models of Computing

The topics that will be covered in this course are as follows, first we shall start with grammars, the four types of grammars some examples ambiguity, and so on. Next we considered the finite state automaton which is the simplest machine model, then we go on to context free grammars, study in detail some of the properties of context free grammars. Next we study about the pushdown automaton which is the machine model for context free grammars. Then we go on to the most general type of the machine - truing machines and we study about the decidability properties and also complexity in truing machines. We shall cover some of the advanced topics like grammar systems, regulatory writing, L systems and so on. And finally, we will end up with two new models of computation the DNA computing and the Membrane computing.

(Refer Slide Time: 01:54)

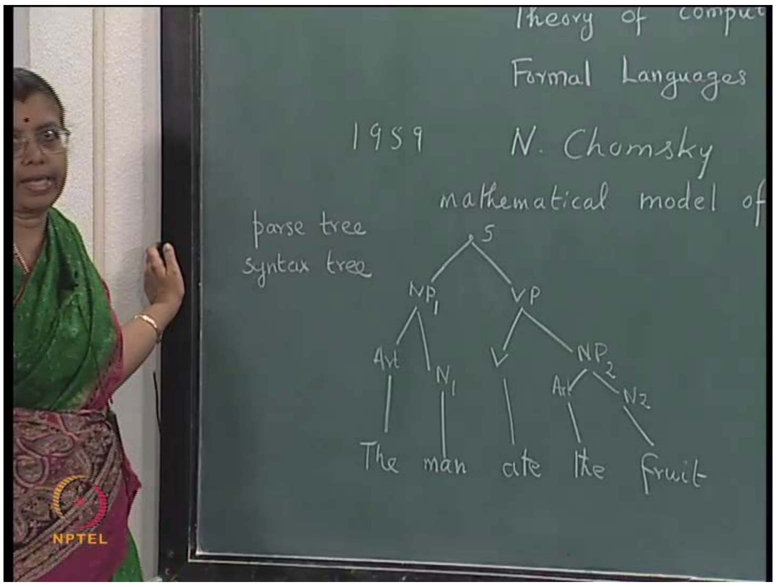


Books

1. K.Krithivasan and R.Rama ; Introduction to formal Languages , Automata Theory and Computation , Pearson Education , 2009.
2. J.E.Hopcroft , R.Motwani and J.D.Ullman , "Introduction to automata Theory Languages and Computation ", Pearson Education Asia , 2001.
3. Peter Linz , "An Introduction to formal languages and Automata ", 3rd Edition , Narosa publishing house , 2002.
4. M.Sipser , "Introduction to Theory of Computation ", Singapore : Brooks/Cole , Thomson Learning , 1992.
5. John.C.Martin , "Introduction to the Language and the Theory of Computation ", 3rd Edition , McGraw Hill , 2003.

The books which could be followed for this course are as follows, the first book is written by me and Rama. Introduction to formal languages Automata theory and Computation; published by Pearson education in 2009. Hopcroft, Motwani and Ullman; Introduction to Automata Theory Languages and Computation, that is also Pearson education. Peter Linz, An introduction to formal languages and Automata. M.Sipper introduction to theory of computation. John Martin, Introduction to the Languages and Theory of Computation. These are some of the books there could be many more books which you could follow, but as a textbook you could follow the first one and the second one.

(Refer Slide Time: 02:56)



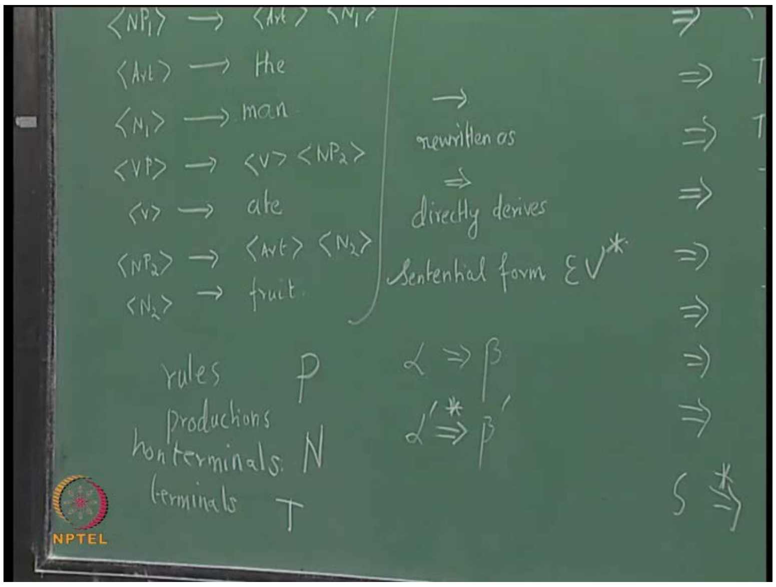
This subject formal languages in Automata theory or you can also call it as theory of computation its started in the year 1959 with the formal definition given by Noam Chomsky. He tried to define what is a grammar? What is a mathematical model of a grammar? The motivation was to study parsing in natural languages like English, so he tried to look at the Parse trees of the natural language sentences and then tried to define what is a grammar.

So, let us take some sentences in English and see how they can be parsed, take a simple sentence first. The man ate the fruit. How can this sentence be parsed? Starting with the sentence this can be considered as a noun phrase and the verb phrase, I will call it as nounphrase1 and verb phrase, then the Verb Phrase itself splits into Verb and Noun Phrase 2.

Noun Phrase 1 splits into Article and Noun1, Article is the, N 1 is man Noun1,Verb is ate, Noun Phrase 2 itself can be splits as Article and Noun 2, Article is the, Noun 2 is fruit.

So, this is the parse tree for this sentence. The man ate the fruit, the parse tree are syntax tree for the grammar. We also call it as a derivation tree, if you go from sentence to the sentence symbol to the sentence it is a derivation, if you go from the sentence to the sentence symbol trying to construct the tree in the bottom of manner, it is parsing.

(Refer Slide Time: 06:02)



$\langle NP_1 \rangle \rightarrow \langle Art \rangle \langle N_1 \rangle$

$\langle Art \rangle \rightarrow$ the

$\langle N_1 \rangle \rightarrow$ man

$\langle VP \rangle \rightarrow \langle V \rangle \langle NP_2 \rangle$

$\langle V \rangle \rightarrow$ ate

$\langle NP_2 \rangle \rightarrow \langle Art \rangle \langle N_2 \rangle$

$\langle N_2 \rangle \rightarrow$ fruit

$\rightarrow$ rewritten as

$\Rightarrow$ directly derives

Sentential form $\in V^*$

$\alpha \Rightarrow \beta$

$\alpha' \overset{*}{\Rightarrow} \beta'$

rules P

productions

non terminals N

terminals T

$S \overset{*}{\Rightarrow}$

So, let us look at the rules here, the rules will be of this form. The sentence symbol goes to Noun Phrase1and Verb Phrase right? The sentence symbol goes to Noun Phrase 1 and Verb Phrase, similarly for each one of that we will write the rules. Then Noun Phrase 1goes to Article and Noun1, Article goes to the, Noun1 goes to man and Verb Phrase goes to Verb and Noun Phrase 2. Verb goes to ate, Noun phrase 2 goes to Article and Noun 2. Again Article goes to the which is already there, Noun 2 goes to fruit. So, these are called rules or productions which are sometimes called production rules. And using these rule s you can derive the sentence, the man ate the fruit from sand not distinguishing between this t and this t, as an article we are taking it as the.

Now, how is the derivation done? Starting with start symbol S, now in the derivation I am applying a rule that is written by a double arrow. A rule there is a left hand side and there is a right hand side, right? And the left hand side is rewritten as the right hand side. So, this S is rewritten as Noun Phrase 1 and Verb Phrase, this double arrow represents directly. So, S directly derives Noun Phrase 1, and Verb Phrase and Noun Phrase 1 directly derives Article and Noun, this directly derives this. So, when you get something from something else this is called a sentential form. This verb phrase remains as it is. So, this is a Sentential form, in this sentential form a Noun Phrase 1 is rewritten as Article and Noun1.

So, this sentential form directly derives this sentential form that is written as a double arrow. Next you have to use a single rule, the rule of article can be rewritten. The Noun1 and Verb phrase they are remaining as they are, then here Noun 1 is rewritten as man, you are applying the rule Noun 1 goes to man. So, Noun 1 is rewritten as man, Verb phrase remains as it is. Please note that I am using only one rule at any step, in the next step the man, the Verb Phrase is rewritten as Verb and Noun phrase 2. So, it is rewritten as Verb and Noun Phrase 2 you seeing this rule? Then the man this is a sentential form, in this sentential form I am applying a one rule for Verb, what is a rule for Verb? Verb goes to ate.

So, Verb goes to ate Noun Phrase 2 remains as it is. The man ate, now Noun Phrase 2 is rewritten as Article and Noun 2 using this rule, then the next step Article is rewritten as the and the last step Noun 2 is rewritten as fruit using this rule, this is a called a derivation, this double arrow is really a relationship between Sentential forms.

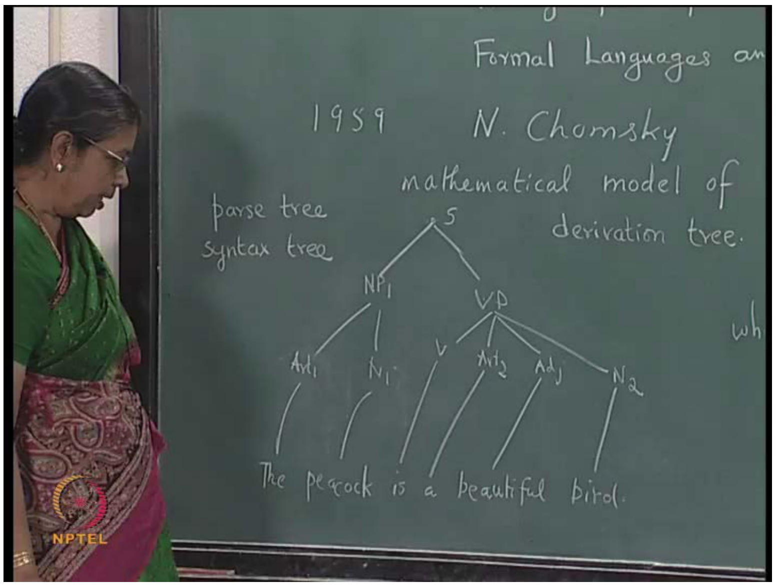
So, if you have a Sentential form Alpha you derive Beta from that, what does that mean? You get beta from Alpha by the application of single rule. So, it represents a relation between Sentential forms, Sentential forms are actually strings over Non-Terminals and Terminals, I will mention what is a Non-Terminal in a moment.

Here you note that there is a distinction between these words and the syntactic categories, these are syntactic categories right Noun Verb Phrase etcetera. So, in these Noun are Verb Phrase is rewritten as something else, so they are called Non-Terminals. Whereas, once you get to the leaves of this tree, the, man, ate, the, fruit, they cannot be rewritten as something else, the derivation terminates there, so these are called Terminals. Usually we denote by capital N the set of Non-Terminals by capitals T the set of Terminals. So, if you consider the alphabet V, this is called the Total alphabet. V is N union T and S is the start symbol or sentence symbol it is one of the Non-Terminals S belongs to N.

And what is a sentential form? It is a string belonging to V star, we know what is meant by V star is not it? We have already considered set operations on sigma star, sigmas and alphabet, here we are taking the alphabet as V right? So, if you look at this double arrow as a relationship between Alpha and Beta. What does the double arrow star denote? And this is a relation what does double arrow star denote, if you say Alpha dash goes to Beta dash what does that mean? Beta dash is obtained from Alpha dash in 0 or more steps, here Beta is obtained from Alpha in one step by the application of a single rule, you keep on applying rules. Finally, from S you are deriving the sentence. The man ate the fruit. So, usually what is this called star? It is called the Reflexive transitive closure, usually this is the reflexive transitive closure. So, double arrow star, Alpha dash double arrow star Beta dash means, beta dash is obtained from Alpha dash in 0 are more strings.

So, a grammar mainly consist of a set of Non-Terminals, a set of Terminals, a set of Production rules, usually denote by P and a start symbol S. So, a grammar consist of a G is equal to N T P S is a 4 duple N T P S. G is equal to N T P S, where N is a finite set of Non-terminals, T is a finite set of Terminals, P is a finite set of Productions, are production rules or rules, S belonging to N is the start symbol. Here you must realize that N intersection T they are disjoint, alphabet non-terminals and terminals are disjoint. This is the definition of a grammar which Chomsky gave, now there is something vague about P, P is something like the left hand side is rewritten as a right hand side. In this example it, so happens that the left hand side is a single Non-Terminal, it is a single syntactic category in this example, right hand side is a string. In general what can be the form of a Production rule? A much general form is possible.

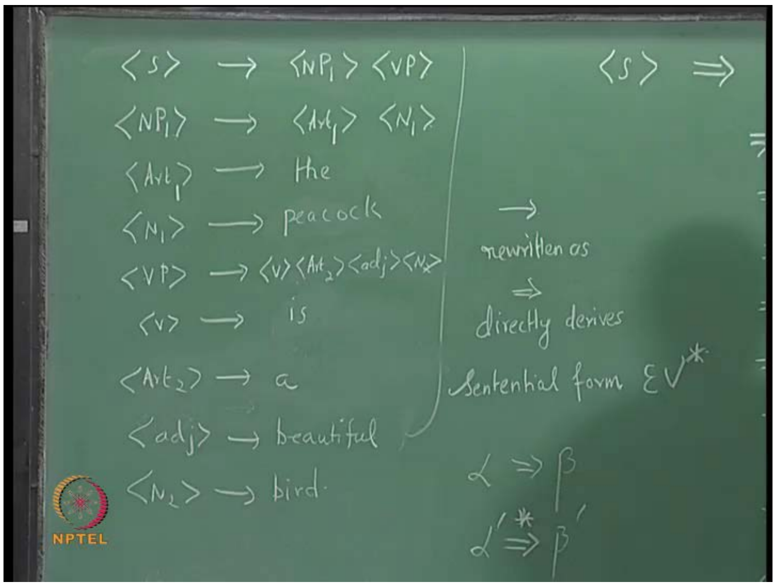
(Refer Slide Time: 21:21)



We will come to that in a moment, before that I would try to take another example of a sentence to make it more convincing. Take this sentence the peacock is a beautiful bird here again the sentence can be rewritten as Noun Phrase 1 and Verb Phrase, Noun phrase 1 is rewritten as Article 1 and Noun 1. Article1 is the, Noun1 is Peacock, Verb Phrase is written as Verb Article 2, Adjective Noun 2. So, Verb is, Article 2 is a, Adjective is beautiful, Noun 2 is bird, this is the parse tree or the derivation tree or the syntax tree for this sentence.

Now, how do the rules look like, the first one is like this, the second one is Noun Phrase 1is Article 1, there are two Articles the and a here, Noun Phrase 1 goes to article 1 and Noun1, Article 1 goes to the, Noun1 goes to peacock.

(Refer Slide Time: 23:10)



$\langle S \rangle \rightarrow \langle NP_1 \rangle \langle VP \rangle$

$\langle NP_1 \rangle \rightarrow \langle Art_1 \rangle \langle N_1 \rangle$

$\langle Art_1 \rangle \rightarrow$ the

$\langle N_1 \rangle \rightarrow$ peacock

$\langle VP \rangle \rightarrow \langle V \rangle \langle Art_2 \rangle \langle adj \rangle \langle N_2 \rangle$

$\langle V \rangle \rightarrow$ is

$\langle Art_2 \rangle \rightarrow$ a

$\langle adj \rangle \rightarrow$ beautiful

$\langle N_2 \rangle \rightarrow$ bird.

$\langle S \rangle \Rightarrow$

$\rightarrow$ rewritten as

$\Rightarrow$ directly derives

Sentential form $\in V^*$

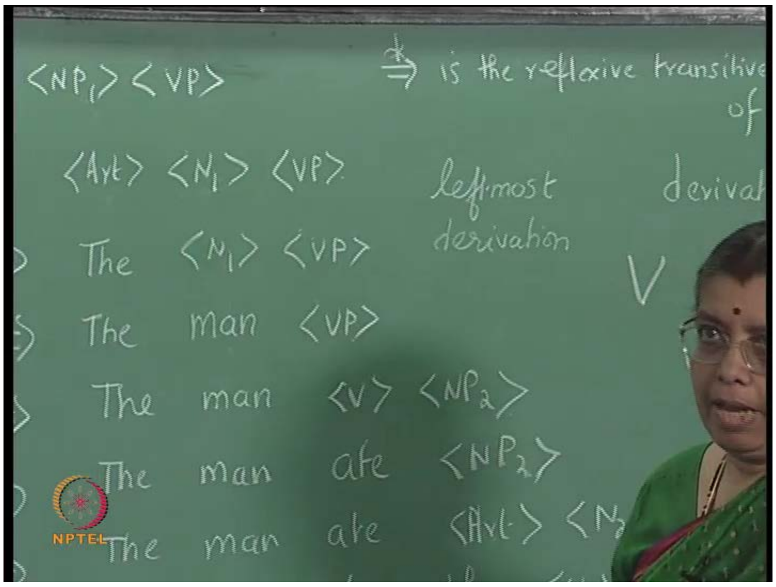
$\alpha \Rightarrow \beta$

$\alpha' \overset{*}{\Rightarrow} \beta'$

And Verb Phrase goes to Verb, Article 2, Adjective Noun 2 right? And Verb goes to is, Article 2 goes to a, Adjective goes to beautiful and Noun 2 goes to bird, these are the production rules. In a similar manner you can derive something like this, I will write the derivation for this also, but before that I want you to look at one of the things. In this derivation, sentence has been rewritten as Noun Phrase 1 and Verb Phrase 2, at the next step I have replaced the Noun phrase 1 as Article and Noun1, there is no necessary that this should have been done. You could have been replaced the Verb Phrase also, that could also have been done, the order does not matter finally, you have to end up with this sentence, but in this derivation I have specifically taken in such a manner that I always replace the leftmost Non-Terminal.

The left most is a sentential form there are 2 Non-Terminals here and I have replaced the left most Non-Terminal here, here again I have replaced the left most Non-Terminal, then the next step left most Non-Terminal, then the left most Non Terminal and so on. In each one of the steps I have replaced the left most Non-Terminal, it is not necessary you could have replaced any one of the Non-Terminal that does not matter.

(Refer Slide Time: 25:32)



$\langle NP_1 \rangle \langle VP \rangle$

$\langle Art \rangle \langle N_1 \rangle \langle VP \rangle$

$\Rightarrow$ The $\langle N_1 \rangle \langle VP \rangle$

$\Rightarrow$ The man $\langle VP \rangle$

$\Rightarrow$ The man $\langle V \rangle \langle NP_2 \rangle$

$\Rightarrow$ The man ate $\langle NP_2 \rangle$

$\Rightarrow$ The man ate $\langle Art \rangle \langle N_2$

$\overset{*}{\Rightarrow}$ is the reflexive transitive of

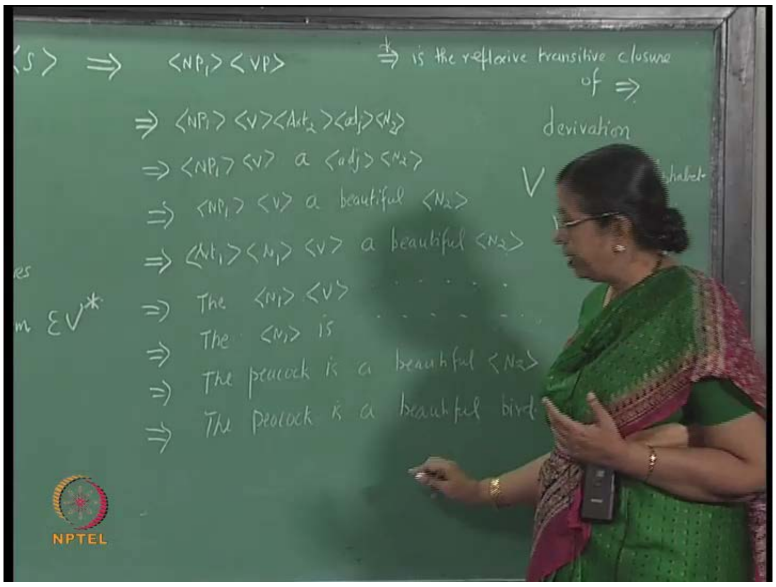
left most derivation

derivat

V

If you always replace the left most Non-Terminal such a derivation is called a left most derivation, this is a left most derivation. In contrast to that I will take the second example and give another derivation which is not left most.

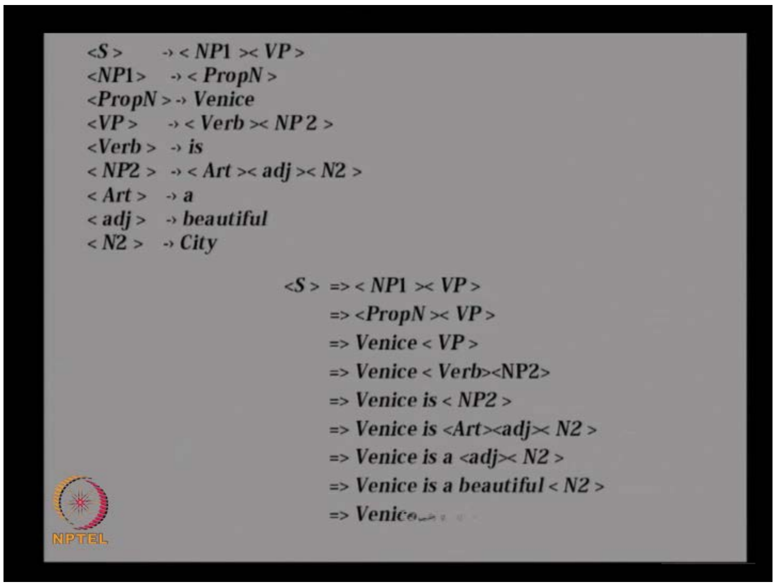
(Refer Slide Time: 26:14)


$\langle S \rangle \Rightarrow \langle NP_1 \rangle \langle VP \rangle$
$\Rightarrow \langle NP_1 \rangle \langle V \rangle \langle Art_2 \rangle \langle adj \rangle \langle N_2 \rangle$
$\Rightarrow \langle NP_1 \rangle \langle V \rangle$ a $\langle adj \rangle \langle N_2 \rangle$
$\Rightarrow \langle NP_1 \rangle \langle V \rangle$ a beautiful $\langle N_2 \rangle$
$\Rightarrow \langle Art_1 \rangle \langle N_1 \rangle \langle V \rangle$ a beautiful $\langle N_2 \rangle$
$\Rightarrow$ The $\langle N_1 \rangle \langle V \rangle$
$\Rightarrow$ The $\langle N_1 \rangle$ is
$\Rightarrow$ The peacock is a beautiful $\langle N_2 \rangle$
$\Rightarrow$ The peacock is a beautiful bird
$\overset{*}{\Rightarrow}$ is the reflexive transitive closure of $\Rightarrow$
derivation
V
$\in V^*$
NPTEL

Now, Noun Phrase 1 and Verb Phrase I am keeping the Noun Phrase 1 as it is and replacing the Verb Phrase by Verb Article 2 Adjective Noun 2. Now I will replace article 2 Noun Phrase 1 Verb, what is Article 2? It goes to a then Adjective, but each step I am using only one rule, then Noun Phrase 1 Verb a Adjective goes to beautiful, using the rule Adjective goes to beautiful and Noun 2. Now, I can replace N Noun Phrase 1 Noun Phrase 1 goes two Article 1and Noun 1, verb, a, beautiful Noun 2. Now, Article 1 can be replaced by the, Noun 1 Verb, this then the next step you can replace Verb as is and next step you can replace the Noun1 as the peacock, is, a, beautiful Noun2.

And the last step the peacock is a beautiful Noun 2 is rewritten as bird, here in sometimes I replace the left most sometimes rightmost, sometimes anything, the order does not matter, someway each step you have replaced one Non-Terminal by the corresponding right hand side. Now, you could also keep to the right most Non-Terminal, always replace the right most Non-Terminal, such a thing is also possible, such that it is called the right most derivation. This is neither a left most derivation, nor right most derivation and if you always replace the right most Non-Terminal, such a derivation is called right most derivation, this is neither a leftmost, nor a rightmost derivation. These two types of derivation rightmost and leftmost are very important, because later on when you learn about compilers you will realize that for top down parsing you use leftmost derivation and for bottom up parsing use right most derivation.

(Refer Slide Time: 29:37)

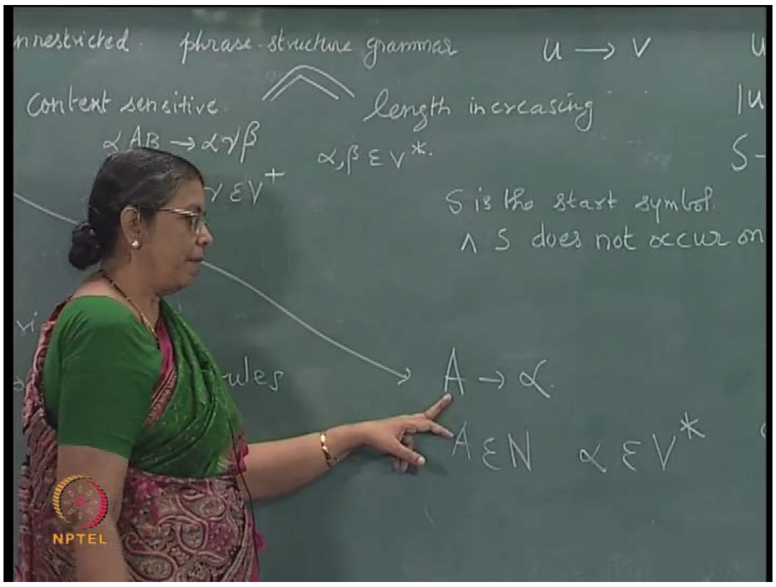


<S> → <NP1><VP>
<NP1> → <PropN>
<PropN> → Venice
<VP> → <Verb><NP2>
<Verb> → is
<NP2> → <Art><adj><N2>
<Art> → a
<adj> → beautiful
<N2> → City

<S> => <NP1><VP>
=> <PropN><VP>
=> Venice <VP>
=> Venice <Verb><NP2>
=> Venice is <NP2>
=> Venice is <Art><adj><N2>
=> Venice is a <adj><N2>
=> Venice is a beautiful <N2>
=> Venice

We consider one more example, which will generate the sentence “Venice is a beautiful city”, the rules for this are given as follows. With the rules given we shall generate the sentence like this, the first rule we get this then using the second rule Noun Phrase 1 is rewritten as proper Noun. Verb Phrase remains as it is, Proper Noun is rewritten as Venice, Verb phrase is rewritten as Verb and Noun Phrase 2, then Verb is rewritten as is, then Noun Phrase 2 is rewritten as Article Adjective. And Noun 2, Article is rewritten as a, Adjective is rewritten as beautiful, Noun 2 is rewritten as city.

(Refer Slide Time: 31:10)



So, the grammar generate the sentence Venice is a beautiful city. Now, we have been vague about the type of production rules, Chomsky defined 4 types of grammar. He defined type 0, type 1, type 2, and type 3 grammars. Type 0 is the most general one, sometimes that is called unrestricted grammar or phrase structure grammar, this is the most general type of a grammar. There the rules are of the form goes to V, where u belongs to V star NV star, it is a string of Non-Terminals and Terminals, but it should have at least 1 Non-Terminal, u should have at least 1 Non-Terminal and it is a string of Non-Terminals and Terminals, that is why it is written as u belongs to V star NV star. If it is fully Terminal you cannot rewrite it as something else, the derivation terminates there, so you should have at least 1 Non-Terminal. V can be any string V belongs to v star, please note that V is N union V is the total alphabet. Now, type 1 is also of this form , but the restriction is the length of the left hand side should be less than or equal to the length of the right hand side, length of the right hands side should be greater than or equal to the length of the left hand side.

This is also called length increasing grammar, this is also equivalent to what is known as a context sensitive grammar, but the definition of a context sensitive grammar is like this, the rules are of the form Alpha A Beta goes to Alpha Gamma Beta, where A is a Non-Terminal, Alpha and Beta are any strings. A is a Non-Terminal and Gamma belongs to V plus. That is A is rewritten as Gamma in the context of Alpha and Beta that is why such a rule is called a context sensitive rule. But these definitions are equivalent definitions, you can easily see that, but a formal proof also I will try to give after a few lectures sometimes later, this type of a grammar generally we take this one this is called type 1 grammar.

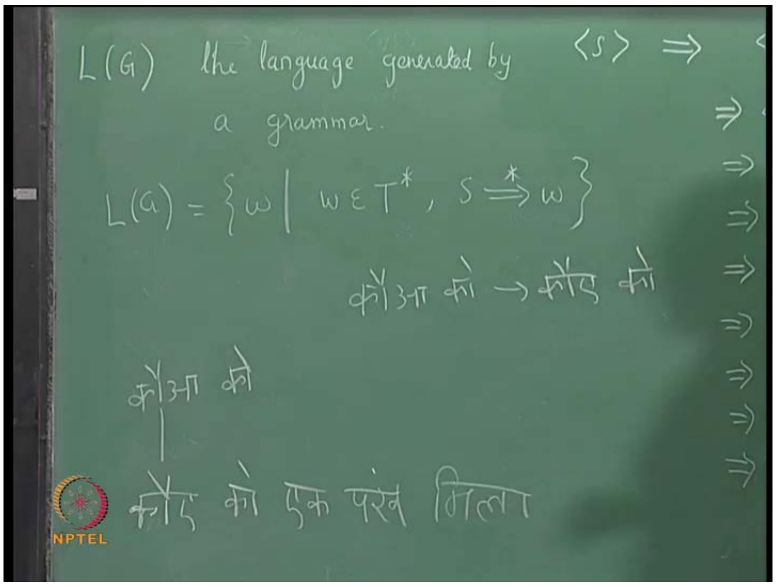
Now, with this restriction you see that you cannot have the empty string Epsilon on the right hand side isn't it? Epsilon cannot occur on the right hand side in a type 1 grammar, but sometimes for some purpose you may want to allow Epsilon. If you want to allow Epsilon you just have the rule S goes to Epsilon and if you want to include the empty string in the language, add this rule S goes to Epsilon, where S is the start symbol and in this case you must make sure that S does not occur on the right hand side of any rule. S is the start symbol, that is if you want to have Epsilon also, you must do it only in this way, S is the start symbol and you can add the rule S goes to epsilon, but you should make sure that S does not occur on the right hand side of any protection.

Now, type 2 rules are of the form A goes to Alpha, where A is a Non-Terminal and Alpha belongs to V star, left hand side you have a Non-Terminal and the right hand side you can have any string, such a rule is called a context free rule. You can see that the two examples which we consider the rules are such that on the left hand side you have a Non-Terminal and the right hand side you have a string, they are all context free rules. Both the examples we consider are context free rules, mostly European languages they will come under natural languages, it is easy to give a context free grammar for them whereas, Indian languages it is slightly difficult to give a context free rules.

So, you see that we have taken a general one, then we have given some restrictions on this we are putting restrictions on the form of production. More and more restrictions we are putting on the form of production rules. So, we put this restriction on type 0 that the length of the right and side is greater than the length of the left hand side we get type 1, then we make restriction that the left hand side can be only a Non-Terminal, then we get a type 2 or context free rules. Most of this arithmetic expressions etcetera you can get by context free rules for examples e goes to e expression e plus e, e goes to e star e, e goes to left parenthesis e, right parenthesis e goes to identifier, this will generate all arithmetic expressions involving plus and star. If you want to use other apparatus minus, division you can use, this is a context free grammar generating all arithmetic expressions involving plus and star and wherever you want to put parenthesis you can put parenthesis using this.

Then identify (( )) can be anything you can just chose (( )) a b c d whatever it is. So, this is an example of a context free grammar which generates all arithmetic expressions. The last one type 3 we put more restrictions the rules are of the form A goes to b, where A and B are Non-Terminals and a is a Terminal, b can be Terminal or Epsilon. In fact, a slightly more general definition also can be given, this is called actually a regular grammar, there is something called right linear, left linear, slight extension of it, but they are equivalent in power. So, we will deal with that in the next class or so…

(Refer Slide Time: 42:01)



L(G) the language generated by a grammar.

$L(G) = \{ w \mid w \in T^*, S \overset{*}{\Rightarrow} w \}$

कौआ को → कौए को

कौआ को

कौए को एक पंख मिला

$\langle S \rangle \Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

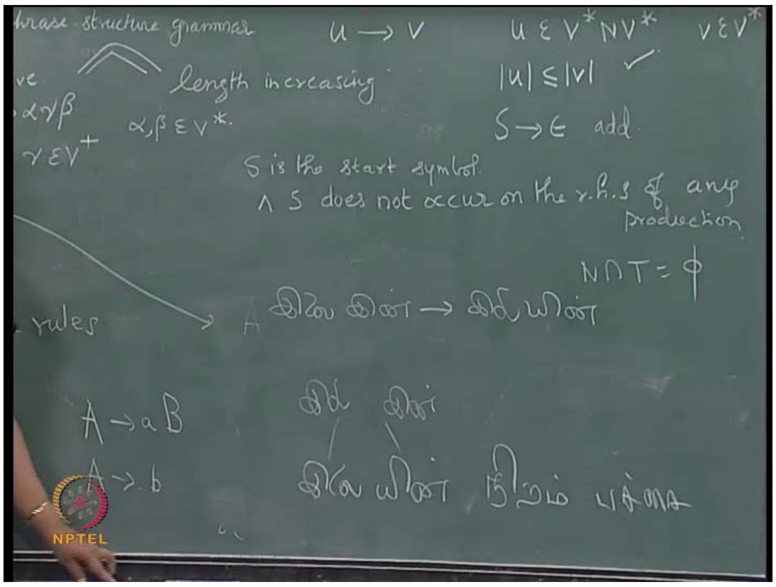
$\Rightarrow$

$\Rightarrow$

Now, having defined what is a sentential form? What is a derivation? What is double arrow star and soon? What is the language defined by a grammar? If the is a grammar what is the language generated by a grammar? It is denoted by LG and it is defined as the set of Terminal strings, w belongs to T star and you can derive w from S, starting from S the set of terminal strings which you can derive, define the language generated by the grammar. So, the two examples which we took in the first example the language consist of only 1 string, the man ate the fruit, the second example the language consist of only 1 string, the peacock is a beautiful bird. In general it will not be like that, for example, here in this one this is the start symbol, E is the only start symbol and the only Non-Terminal, plus star left parenthesis, right parenthesis, identifier, they are all terminal symbol. So, from this you can derive something like (( )), you can derive many things from here right? How will you derive this here? E goes to E star E, E goes to identifier, E goes to left parenthesis ,E right parenthesis E plus Ei d, i d this is the derivation tree, we can write a derivation using this in that manner.

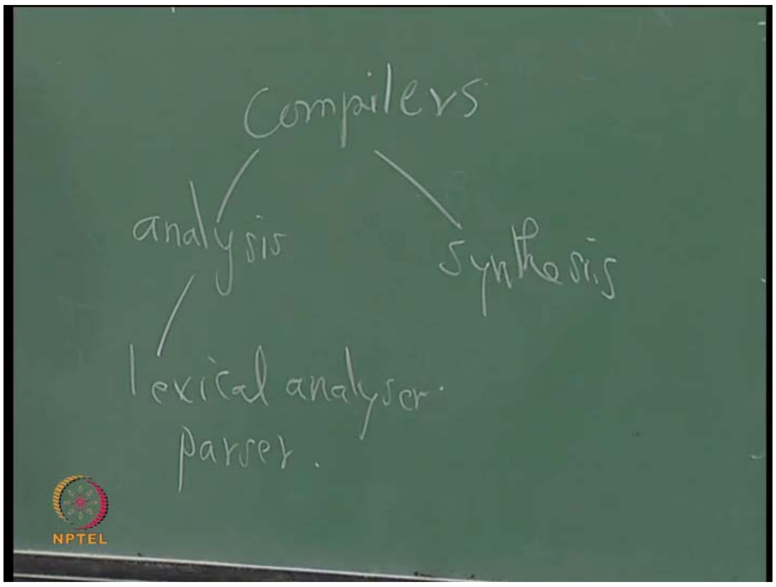
Now, as I told you the Indian languages they may not permit context free rules alone, I mean you cannot describe them using context free rules alone, you may require context sensitive rules. We will take one example from Hindi and one example from Tamil, now look at this sentence I will not draw the whole tree, but only a portion of it the previous step is. And this is a case ending and the rule applied here is. So, this changes into in the context of the case ending. So, such a rule is called a context sensitive rule, a similar thing can occur in other Indian languages also, if you are familiar with the Tamil.

(Refer Slide Time: 45:57)



So, the previous step here will be and the rule applied is. So, this IN is a case ending it changes to IN the presence of the Noun, only in the presence of the No unit changes, the case ending changes, the form of the case ending changes. So, this changes into this in the context of the Noun. So, such a rule is called a context sensitive rule, type 1 is called type 1. The use of this grammars we will learn more about grammars more examples in the next class.

(Refer Slide Time: 47:25)

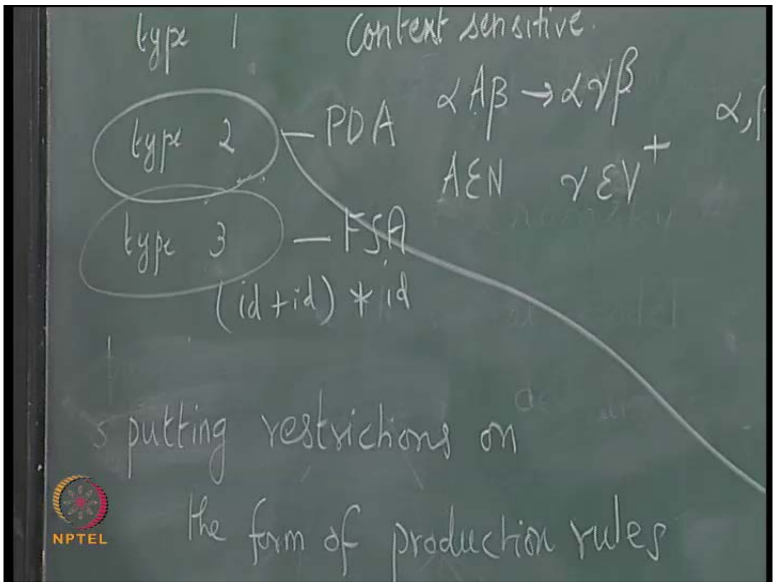


But before that use of this is mainly from compilers of course, though they were different from natural language point of view it had lot of application compilers, it has an analysis part and the synthesis part. The analysis part we have the lexical analyzer and parser, which in 1960when ALGOL 60was defined, it is a block structured language. People found that it could be described by what is known as Backus normal form B N F. And B N F was nothing, but type 2 grammar, context free grammar, but it has been known that you can give the rules in context free, but you may not be able to capture all the features. Pascal has a very beautiful grammar simple grammar, which is type 2. In fact, a subclass sub type 2 L L1grammar, it can be the compiler for a Pascal is top down one recursive descent parser you have.

Whereas, languages like Fortran and PL1 which were considered during the 60. All features of those languages cannot be captured using context free rules and all there were some context sensitive aspects, but when you have consider a context sensitive grammar, the parse tree or syntax tree cannot be drawn easily, it is not very easy to have a parse tree with a type 1 grammar, only type 2 grammar, something goes into something you can have a tree like structure.

So, how do you capture the context sensitive features of such languages like Fortran or P L 1 or later also many languages. The features are not merely context free, some other higher features are involved, we will learn about them in the due course. So, for that what they did is? The rules they maintained as context free, because that is helpful in parsing, but then some additional features like attributes or some other controlled mechanisms were involved, so that the other properties of the language are also taken care of. So, as we go along the course we will learn about them. So, there are other ways of putting restriction on the manner of applying the rules, here the four grammar of Chomsky we obtained by taking the most general form, unrestricted grammar then put some restriction, then we got type 1, put more restriction we got type 2, put more restriction we got type 3.

(Refer Slide Time: 50:50)



type 1 Context sensitive
$\alpha A \beta \to \alpha \gamma \beta$
type 2 — PDA
$A \in N \quad \gamma \in V^+$
type 3 — FSA
$(id + id) * id$
putting restrictions on
the form of production rules

So, we were putting restrictions on the form of the production rules. So, our concentration will be mainly on type 2 grammars and type 3, these grammars are called generative devices, they generate the language in contrast to that you have recognition devices also. So, a language can also be accepted by an automata. So, Formal Language Theory is more about grammars, Automata theory is more about Automata, they go together Formal Language Theory Automata Theory they go together and it is called Formal Language and Automata Theory.

So, whenever a language is defined you can define it by means of a generative device, which is known as a grammar. At the same time you can also have an automata recognizing that language, the simplest Automaton is known as the Finite State Automaton which corresponds to a type 3 grammar, Finite State Automaton. And for type 2 you have a Pushdown Automaton, which is has a stack and that stack is the major idea in parsing. In type 1 you have what is known as a Linear Bounded Automata and for type 0 you have the corresponding reorganization defined as Truing machines. And one of the thing I should mention is the truing machine was defined in 1936 itself and even 15 years before the first computer was built. And at the time the person who defined A MT ruing he was able to foresee, what is possible with the computer and what is not possible with the computer, 15 years before the first computer was built and that has stood the test of time. Even now what he defined as computer built it still stands up as the definition of truing machine and is taken as the model of computation till today. So, we will consider more examples in the next class.