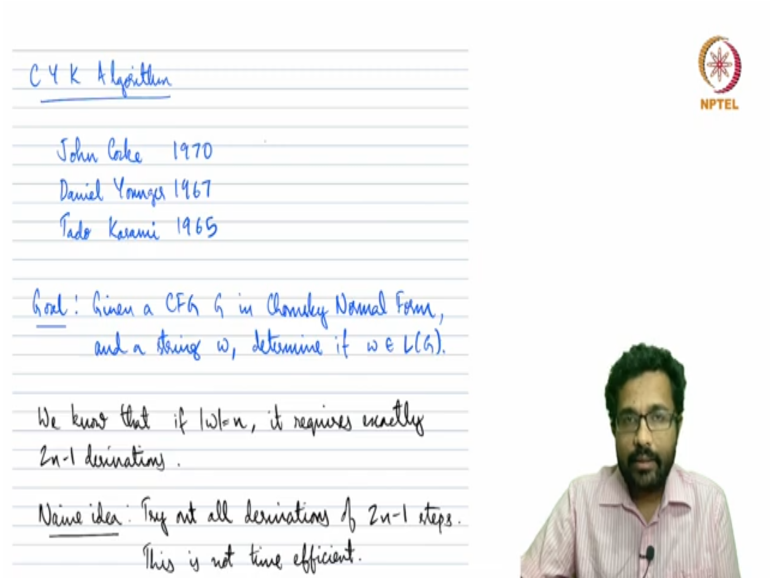
Theory of Computation

Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

CYK Algorithm – Part 1

(Refer Slide Time: 00:16)



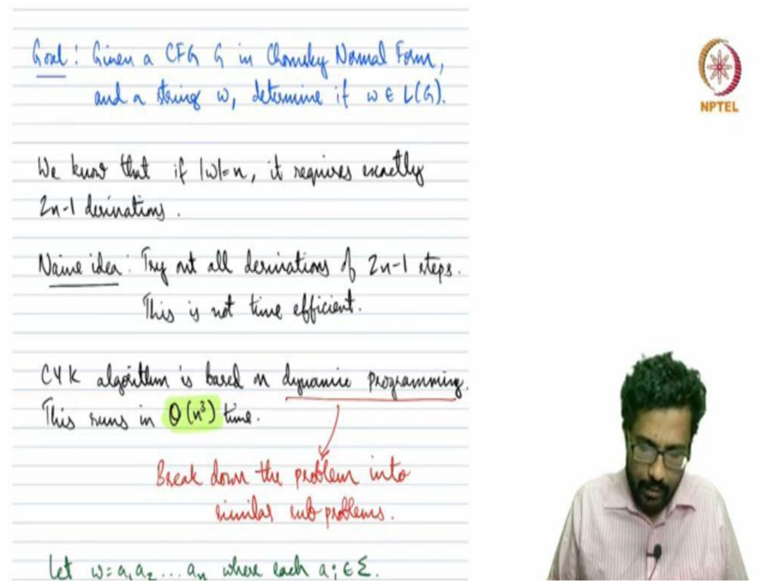
CYK Algorithm

John Cocke 1970
Daniel Younger 1967
Tado Kasami 1965

Goal: Given a CFG G in Chomsky Normal Form, and a string w, determine if $w \in L(G)$.

We know that if $|w| = n$, it requires exactly $2n-1$ derivations.

Naive idea: Try out all derivations of $2n-1$ steps. This is not time efficient.



Goal: Given a CFG G in Chomsky Normal Form, and a string w, determine if $w \in L(G)$.

We know that if $|w| = n$, it requires exactly $2n-1$ derivations.

Naive idea: Try out all derivations of $2n-1$ steps. This is not time efficient.

CYK algorithm is based on dynamic programming. This runs in $O(n^3)$ time.

Break down the problem into similar sub-problems.

Let $w = a_1 a_2 \ldots a_n$ where each $a_i \in \Sigma$.

Hello and welcome to lecture 18 of the course theory of computation. In the previous lecture, we saw Chomsky normal form, and how the structure of Chomsky normal form makes it easy for us to verify whether a string is derived from a certain grammar. So, one of the things that we mentioned was that it, there is an algorithm there is an efficient algorithm, if the given grammar is in the Chomsky normal form. We can, there is an efficient algorithm to test whether a given string can be derived from the grammar. So, that algorithm is called, one of the algorithms that does this is called the CYK algorithm, and which is what we are going to see in this lecture. So, it is CYK algorithm.

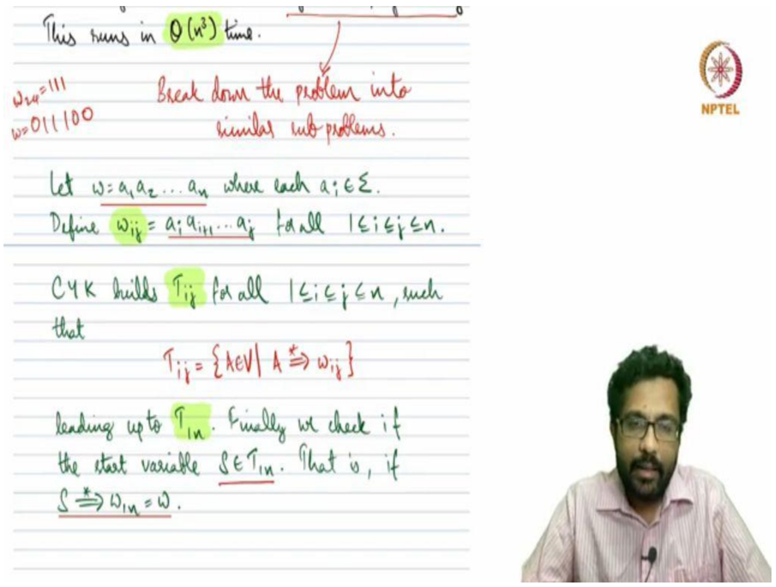
It was independently discovered by three people Cocke, Younger and Kasami in the respective year; so, around late 60s to 70. So, they all, the algorithm does the following. It addresses the question that I just said, given a grammar in Chomsky normal form, and given a string; it determines whether the string can be derived from the grammar. So, in the previous lecture, we saw that if the string is of length n and if the grammar is in Chomsky normal form, we require exactly 2n - 1 derivations; so, exactly 2n - 1 derivations. So, one possibility is let us say if you get a string of length 5, we know it requires 9 derivations. We could just try out the grammar and try out all possible derivations that require 9 derivations; and see whether the string is produced.

But, there is a fairly inefficient way; so you will have to. It is kind of like a brute force approach, but the only intelligence that we are using is that we are limiting ourselves to 90 derivations; so, this is not very efficient. It kind of quickly blows up because of the number of rules. So, the CYK algorithm is different, and is a more efficient approach for answering this. This is an approach based on dynamic programming. So, dynamic programming, as you may know, basically you break down the problem into similar sub problems. So, for those of you who have learned algorithms or basic undergraduate algorithms or learning algorithms, this may seem familiar.

So, basically what we do is this, I am talking about dynamic programming in general; so, you have a certain problem. And basically, you break down the problem into small small pieces, and you use the same approach to build on this; so, you solve the problem in the small pieces. And then you use those solutions to build on and for the, and to get the solutions for the larger pieces. So, that is the general approach in dynamic programming. For those of you who have seen dynamic programming, this particular algorithm may seem; it may be easier to understand this algorithm, because there is a standard structure and approach to dynamic programming; and then you may be able to relate to that.

But, for those of you who have not seen dynamic programming, also you do not really need to worry; because I am going to explain the step by step procedure that we are going to follow. So, it is fairly straightforward and simple. So, we are going to see dynamic programming based algorithms and this runs in $\theta(n^3)$ time; so, when n is the length of the input string. So, the typical assumption is that the grammar is there and you are given a string. So, you are given different strings and one needs to understand whether this string can be derived from the grammar. So, let us see the algorithm. So, before that, some setting up of some basic notation which we will be requiring in the algorithm.

(Refer Slide Time: 04:40)



This runs in $\Theta(n^3)$ time.

$w_{24} = 111$

$w = 011100$

Break down the problem into similar sub problems.

Let $w = a_1 a_2 \ldots a_n$ where each $a_i \in \Sigma$.

Define $w_{ij} = a_i a_{i+1} \ldots a_j$ for all $1 \leq i \leq j \leq n$.

CYK builds T_{ij} for all $1 \leq i \leq j \leq n$, such that

$$T_{ij} = \{A \in V \mid A \overset{*}{\Rightarrow} w_{ij}\}$$

leading up to T_{1n}. Finally we check if the start variable $S \in T_{1n}$. That is, if $S \overset{*}{\Rightarrow} w_{1n} = w$.

So, let the string be w, the input string be W; and let W be of length n. And let a_1, a_2 et-cetera be the up to and be the n individual symbols of the string. So, a_1, a_2 are the individual symbols and now we are going to define something called Wij, which is the substring from the ith substring to the i th symbol to the j th symbol. So, for instance, if the string is 011100, this is W; then W, let us say 24 is 111. Because the second symbol is the second symbol to the fourth symbol; so second, third, fourth symbol, which is 111 here; and maybe W56 is 00. So, now you understand what Wij is.

It is a substring from the ith symbol till the j th symbol, and it is defined whenever i is less than or equal to j. Of course, both of them lie between 1 and n. So, what the CYK algorithm does is? It builds this set called T_{ij}, for all such i and j. Whenever i is less than or equal to j, it builds at T_{ij}. And what is Tij? T_{ij} is a set of all a where is a variable; maybe I will just write that. So, for all maybe I just write that for A is a variable. So, A in V means A is a variable, where A yields or A derives the substring W_{ij}. So, which is the set of variables from which we can derive that substring? So, I said here W24 is 111.

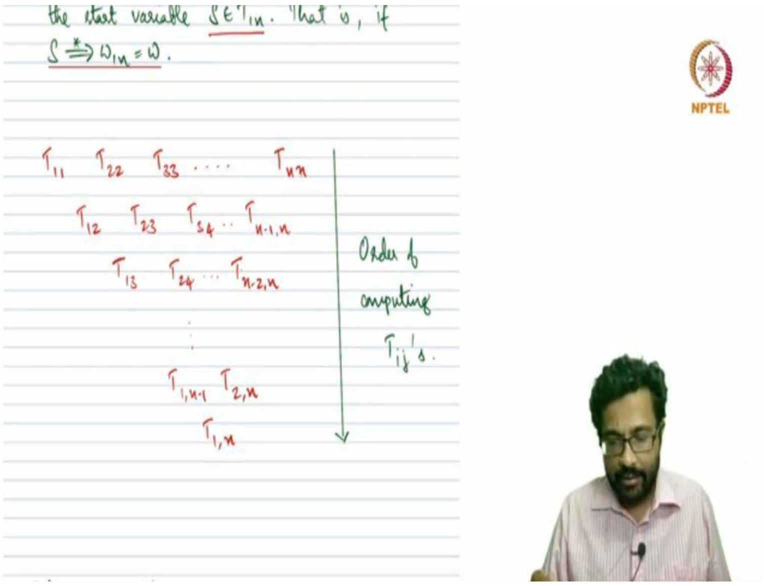
So, T_{24} will be the set of all variables which will just, which will be able to derive this substring 111. So, when I say derive means that variable alone should derive this and nothing extra should be there, exactly this. So, what we do is, we will we want to, the CYK algorithm will build these sets. So, if there are multiple such variables that derive Wij will, that set will contain all of this; and then we will build all the Ti's, all the sets T_{ij}. And finally, we will also build; so, we will also build the set T_{1n}. T_{1n} is the set of all variables that derive W_{1n}; but W_{1n} is a substring from the first symbol to the n symbol, which is actually the entire string. So, which are the variables that derive the entire string? And then we see if the start variable is such a variable.

In other words, we check if the start variable is in set T1n, which is basically asking for the start variable; sorry, a start variable is set from which we can derive the entire string. So, this is pretty much the definition of whether the string is derived by the grammar. So, if the grammar generates the string, what does it mean to say the grammar generates the string? The grammar generate the string if the start variable can derive the string. So, we build all these sets T_{ij} leading up to the set T1n; and then we finally we once we have built T_{1n}, we just check whether the start variable is there. If the start variable is there, that means the start variable can derive the string W.

If the start variable can derive the string W that is the same as saying that the grammar W is generated by the grammar. If the start variable is there, we say yes, W is generated by the grammar. If not, we said W is not generated by the grammar; and so that is the high level picture. So, what do we do? We have W which is a1 to an individual symbol. We defined Wij which is the substring from the ith symbol to the jth symbol, where i is less than or equal to j. Corresponding to Wij, we build sets Tij, which are the set of variables from which the substring Wij can be derived. And we use the smaller we use the smaller Tijs to build a bigger Tijs; whereas, a smaller Tijs, I mean Tijs corresponding to the smaller substrings to build the Tijs corresponding to the bigger substrings.

And so on and so on till we get to the entire string which is T1n or which is W1n. And we in the set T1n which corresponds to the entire string; we checked whether the start variable is a member.

(Refer Slide Time: 10:00)



the start variable $S \in T_{1n}$. That is, if $S \overset{*}{\Rightarrow} w_{1n} = w$.

T_{11} T_{22} T_{33} .... T_{nn}

T_{12} T_{23} T_{34} .. $T_{n-1,n}$

T_{13} T_{24} ... $T_{n-2,n}$

⋮

$T_{1,n-1}$ $T_{2,n}$

$T_{1,n}$

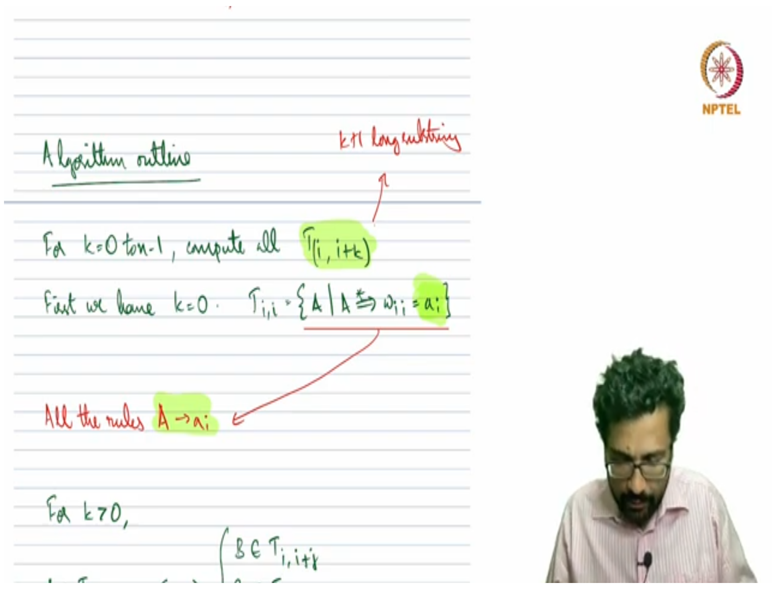
Order of computing T_{ij}'s.

So, this is the order in which we are going to compute these sets. So, we are going to compute, initially we are going to compute the first row that is written here T_{11}, T_{22} and so on up to T_{nn}. So, which is this, so T11 corresponds to W11; W11 means it is a substring from the first symbol to the first symbol; which means the substring consisting of only one symbol. T22 is the substring consisting only of only the second symbol, just again one symbol and so on. So, the first row consists of all the sets, where the substrings correspond to one symbol. Second row corresponds to all the sets, where the substring corresponds to two symbols. T12 means a1, a2; T23 means a2, a3; so, the substring corresponds to or a substring is of length 2.

In the third row the substring is of length 3; so, the set corresponds to the substring of length 3. And this is the order in which we proceed. We first compute the sets in the first row T11, T22 and so on; then, T12, T23, T34 and so on. So, when we compute T12, we will make use of the sets T11, T22. When we compute T13 the third row, we will make use of the previously constructed sets; we will make use of T11, T23, T12 and so on. And finally, we come to the last set which is T_{1n} which is the substring that is only a substring of length n; and then we check whether the start variable is part of this set. So, this is the order in which from top to bottom, this is how we are going to compute the Tijs. So, now, let me just explain it a bit further.

So now, I have told you the order; now let us see how in this order, how are we actually going to compute the sets. So, this is the order: first we T1 first substrings of length one, then substring of length two and so on. So, the order is of increasing length of the substring.

(Refer Slide Time: 12:16)



Algorithm outline

k+1 long substring

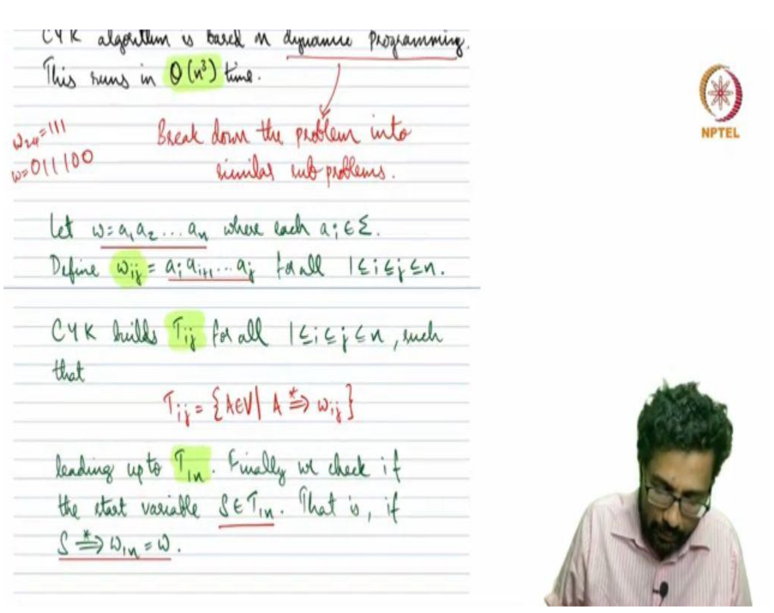
For $k = 0$ to $n-1$, compute all $T(i, i+k)$

First we have $k = 0$. $T_{i,i} = \{A \mid A \overset{*}{\Rightarrow} w_{ii} = a_i\}$

All the rules $A \rightarrow a_i$

For $k > 0$,

$B \in T_{i,i+j}$



CYK algorithm is based on dynamic programming.
This runs in $O(n^3)$ time.

$w_{24} = 111$
$w = 011100$

Break down the problem into similar sub problems.

Let $w = a_1 a_2 \dots a_n$ where each $a_i \in \Sigma$.
Define $w_{ij} = a_i a_{i+1} \cdots a_j$ for all $1 \leq i \leq j \leq n$.

CYK builds T_{ij} for all $1 \leq i \leq j \leq n$, such that

$$T_{ij} = \{A \in V \mid A \overset{*}{\Rightarrow} w_{ij}\}$$

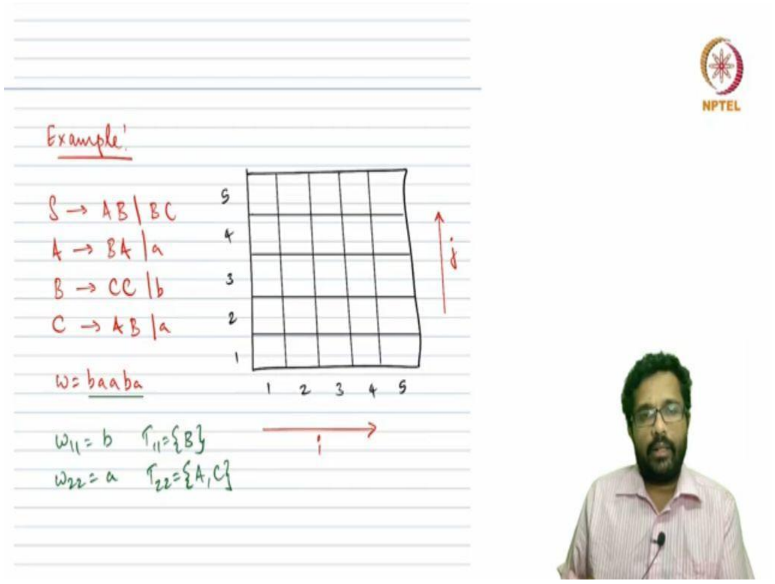
leading up to T_{1n}. Finally we check if the start variable $S \in T_{1n}$. That is, if $S \overset{*}{\Rightarrow} w_{1n} = w$.

So, what we are going to do is we are going to compute all $T_{i,i+k}$; so, where the substring is of length actually, so ith symbol to i plus k symbol. So, substring is of length, substrings of length k plus 1; so k plus 1 long substring. $T_{i,i+k}$ correspond to k + 1 length substring. So, we compute this from for k equal to 0, sorry, to n - 1; so, we from k equals 0 to n - 1, we compute all set substrings. First, we take the k equal to 0 case, which means the substring is of length one; so, substrings of length one means T i, i; k is 0.

So, we are asking whether there are variables A from which Wi, i can be derived. So Wi, i if you remember is a single is from the ith substring, ith symbol to the symbol; so it is basically just the symbol ai. So, we are asking whether there are variables, we generate ai, we generate ai. So, we know from the structure of the grammar, the only way to generate single variables is if there is a rule of this type. Some A, some variable A directly yields a variable ai, because that is pretty much the only way; because, if you have, if you use more steps, the length of the resulting string is going to get bigger.

So, the way to check whether the variables that are going to derive a_i; so the only way a variable can derive ai is if there is a direct rule. Sorry, is there is a direct rule A gives a_i? If there is no such direct rule, you cannot produce this. So, basically you go through all the rules and check whether such rules are there. Whenever such rules are there, you include this. So, maybe we will work out an example.

(Refer Slide Time: 14:48)



Example:

$S \rightarrow AB \mid BC$

$A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$

$C \rightarrow AB \mid a$

$w = \underline{baaba}$

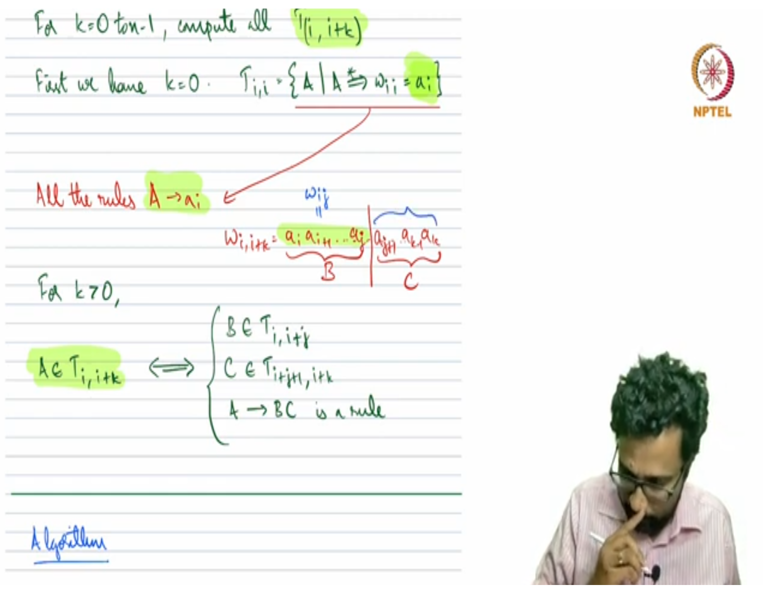
$w_{11} = b \quad T_{11} = \{B\}$

$w_{22} = a \quad T_{22} = \{A, C\}$

So, suppose this is a grammar into Chomsky normal form. So, notice that in all the rules, the start variable does not come in the left hand side; there is no epsilon rule. And all the rules on the right hand side is such that there are either two variables or a single terminal. And so, let us just see, so, which are the variables that derive let say T. So, the string that that we are that is of interest is b a a b a; so, W11 is equal to just b. So, which are the variables that derive b? So, the T_{11} is basically the set just capital B; because capital B derives b. But, A and C, A C S do not derive B.

And just to complete W22 is the symbol a; and T22 is capital A and capital C, capital A and capital C derive a. So, we say this is capital A and this is capital C. So, for k equals 0, it is just that easy. We just basically look at the rules and see if there is such a rule that a particular variable directly gives that symbol.

(Refer Slide Time: 16:15)



For $k=0$ to $n-1$, compute all $T(i, i+k)$

First we have $k=0$. $T_{i,i} = \{A \mid A \overset{*}{\Rightarrow} w_{ii} = a_i\}$

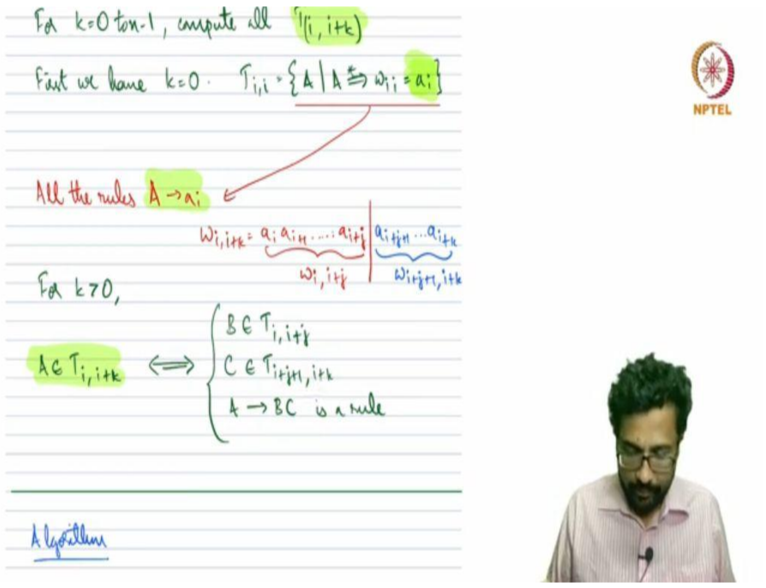
All the rules $A \rightarrow a_i$

$$w_{i,i+k} = \underbrace{\overset{w_{ij}}{a_i a_{i+1} \dots a_j}}_{B} \Big| \underbrace{\overbrace{a_{j+1} \dots a_{k-1} a_k}}_{C}$$

For $k>0$,

$$A \in T_{i,i+k} \iff \begin{cases} B \in T_{i,i+j} \\ C \in T_{i+j+1,i+k} \\ A \rightarrow BC \text{ is a rule} \end{cases}$$

Algorithm



For $k=0$ to $n-1$, compute all $T(i, i+k)$

First we have $k=0$. $T_{i,i} = \{A \mid A \overset{*}{\Rightarrow} w_{i,i} = a_i\}$

All the rules $A \to a_i$

$w_{i,i+k} = \underbrace{a_i a_{i+1} \cdots a_{i+j}}_{w_{i,i+j}} \mid \underbrace{a_{i+j+1} \cdots a_{i+k}}_{w_{i+j+1,i+k}}$

For $k>0$,

$$A \in T_{i,i+k} \iff \begin{cases} B \in T_{i,i+j} \\ C \in T_{i+j+1,i+k} \\ A \to BC \text{ is a rule} \end{cases}$$

Algorithm

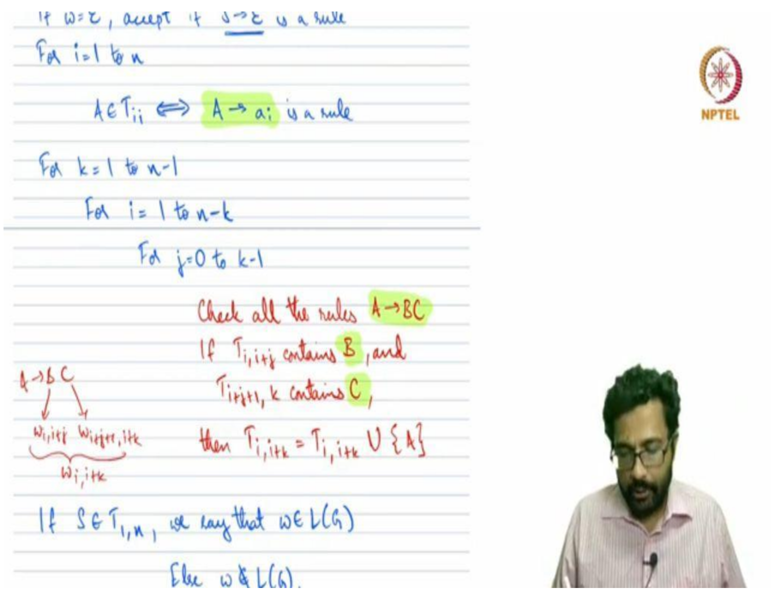
For k greater than 0, it becomes more interesting; so, this is when k equals 0. Remember k equals 0 means of length one; so k greater than 0 means length two and higher. So, when does a certain variable, when is a certain variable A part of the set $T_{i,i+k}$. So, the way it works out is the following. So, A has to derive the string W i, i plus k; so, A has you derive the string. So, W i, i plus k which is ai, ai plus 1 and so on; I am just splitting it like this ak. So, ak minus 1 and so on; so, this is how W i, i plus k is written. And now why can a variable capital A derive the string? The way it would be derived is basically think of k as a slightly bigger number like maybe 5 or 10 or something, not 1 or something.

So, it needs to go through multiple steps. So, the first step has to be; so there are two types of rules allowed in the Chomsky normal form. So, we are also using the structure, we need to use a structure. So, one variable A gives a symbol, but then that would mean the symbol will be of single length; then we are already at a terminal. The other rule A gives B C. Suppose we are using the rule A gives B C, A gives B C. So, which means now B should generate some part of the string. So, let us say B generates till its part aj; so B generates the string till. So this, this highlighted part is generated by B and sorry; and the next symbol onwards aj plus 1 onwards, this is generated by C; and A gives BC.

So, when can A be, when can A generate the substring Wi to i plus K? That happens when A gives B C is a rule, which is what I have written here; and B generates the string ai to aj. So, what is ai to aj? ai to aj is nothing but this is equal to Wij, and aj plus 1 to ak. I will just change it a bit; so, instead of ij, I want to say it a bit differently. So, I will say a i, i + j; maybe I will use a different color for the right side, ai to i plus k. So, repeating this, so this part is Wi, i plus j and this part, sorry; this part is actually Wi plus j plus 1 to i plus k. And we want B to generate the part in red Wi to i plus j; and C to generate the part in blue which is i plus j plus 1 to i plus k.

So, A gives a is in this set $T_{i,i+k}$. If A gives B C is a rule, where B C can generate some part i to i plus j, and C can generate the remaining part i + j + 1 to i + k. So, this is the idea. So, now let us formally view the algorithm; and then once we formally view the algorithm it is just about like maybe we work out an example, and that should make things clear. So, this is the same thing; I am just writing it in a structured form.

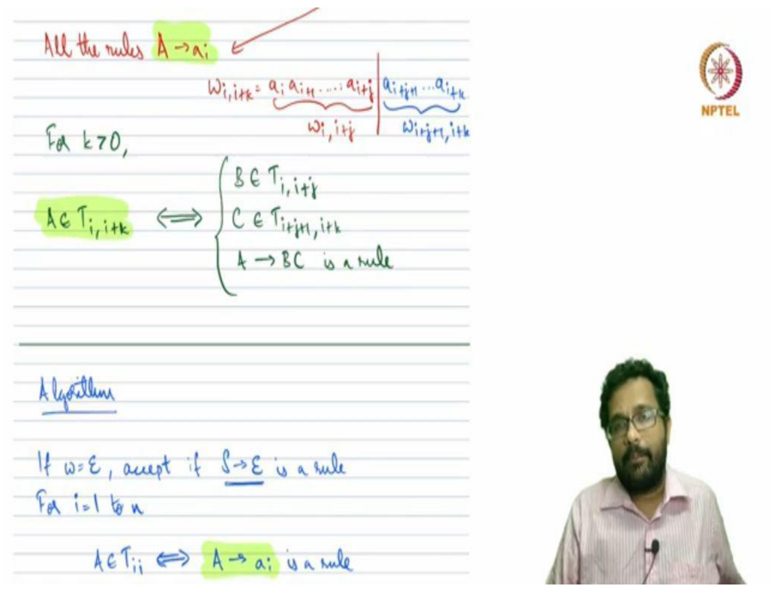
(Refer Slide Time: 21:27)



```
If $w = \varepsilon$, accept if $\underline{S \to \varepsilon}$ is a rule
For i=1 to n
        $A \in T_{ii} \iff A \to a_i$ is a rule
For k=1 to n-1
    For i=1 to n-k
        For j=0 to k-1
            Check all the rules $A \to BC$
            If $T_{i,i+j}$ contains B, and
               $T_{i+j+1,k}$ contains C,
            then $T_{i,i+k} = T_{i,i+k} \cup \{A\}$
If $S \in T_{1,n}$, we say that $w \in L(G)$
        Else $w \notin L(G)$.
```

$$\begin{tikzcd}[column sep=small] & A \to B \arrow[dl] & C \arrow[d] \\ w_{i,i+j} & & w_{i+j+1,i+k} \end{tikzcd}$$

$\underbrace{w_{i,i+j} \quad w_{i+j+1,i+k}}_{w_{i,i+k}}$



All the rules $A \to a_i$

$w_{i,i+k} = \underbrace{a_i a_{i+1} \cdots a_{i+j}}_{w_{i,i+j}} \Big| \underbrace{a_{i+j+1} \cdots a_{i+k}}_{w_{i+j+1,i+k}}$

For $k > 0$,

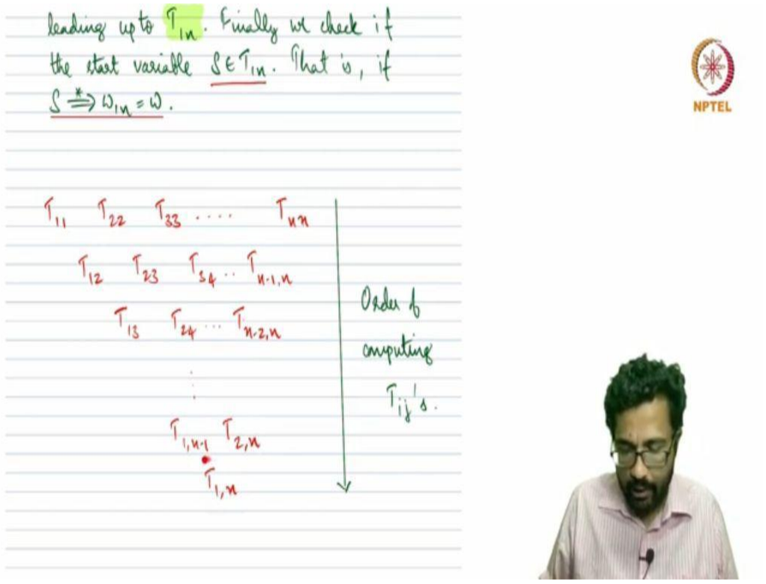
$$A \in T_{i,i+k} \iff \begin{cases} B \in T_{i,i+j} \\ C \in T_{i+j+1,i+k} \\ A \to BC \text{ is a rule} \end{cases}$$

Algorithm

If $w = \varepsilon$, accept if $\underline{S \to \varepsilon}$ is a rule

For i=1 to n

$A \in T_{ii} \iff A \to a_i$ is a rule


leading up to T_{1n}. Finally we check if
the start variable $S \in T_{1n}$. That is, if
$S \overset{*}{\Rightarrow} w_{1n} = w$.
T_{11} T_{22} T_{33} .... T_{nn}
T_{12} T_{23} T_{34} .. $T_{n-1,n}$
T_{13} T_{24} ... $T_{n-2,n}$
$T_{1,n-1}$ $T_{2,n}$
$T_{1,n}$
Order of computing T_{ij}'s.
NPTEL

So, let us see, let us go through the algorithm again; it should be fairly easy. If the string is empty, so we have to, we are given a grammar in Chomsky normal form; and we are given a string. The goal is to determine whether the string is generated by the grammar. So, if the string is empty, Basically, the only way an empty string can be generated in the Chomsky normal form is, if we have a rule of this type S gives the empty string; so, you just check that. If the string is of length 1, or if the string is not the empty string; sorry, not of length one; basically, we start building these sets.

So, as I said before, we build these sets for the smaller length substrings length 1 and so on till length n.

So, first, let us see how to build substring sets for the substrings; sorry sets T_{ii} which correspond to the substring of length 1. So, we built A, we build the sets T_{ii}, where A is in Tii, if there is a rule where A gives the symbol ai. For bigger substrings for substrings of length 2 and above, we have this process. So, k is equal to 1 to n - 1. So, k equal to 1 to n - 1 means; so this corresponds to the substrings, each the length of the substring, i equal to 1 to n - k, it corresponds to the position of the substring. So, i equal to 1 means substring starting from the first location; i equal 2 means substring starting from the second location and so on.

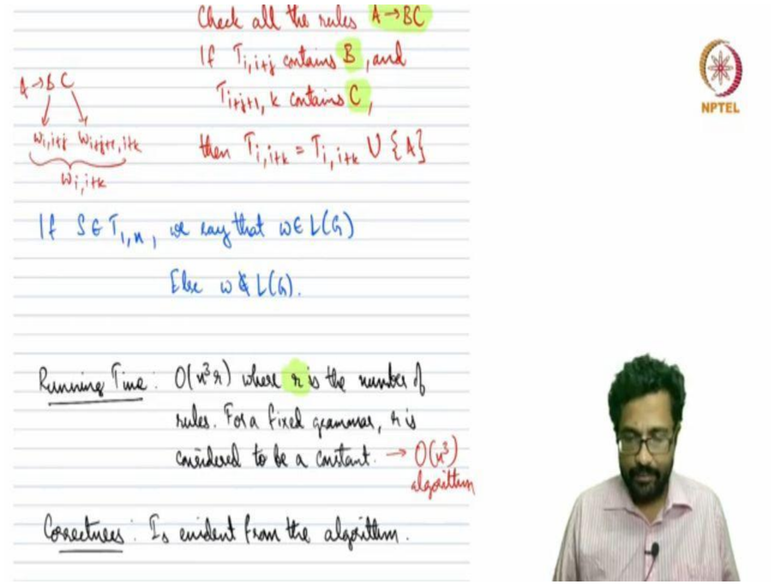
And it can only go till up to n minus k, because the substrings under consideration; ak is already fixed we are on this, we are on the second loop. So, we cannot start beyond n - k. And j equal to

0 to k - 1 indicates where we are going to split. So, here so if you recall this point, this is exactly what it is, k denotes the length of the substring, k equal to 0 means substring of length 1, i denotes the point where the substring begins and j denotes the point where the first split happens. So, for i, j, k as defined, so outermost loop is k, then i, then j; so, for all this, the lengths of a, for all the substring lengths starting from the smallest. Then, we look at it from the leftmost and then we look at all the possible splits. For a specific such split we check all the rules A B C, A gives B C.

And we check if $T_{i,i+j}$ contains the variable B; and $T_{i+j+1,k}$ contains a variable C. If this happens, we know that A gives BC and the first part of the substring is generated by B, and the second part of the substring is generated by C. So, basically it is like this. So, if this happens, A gives B C; and this generates $W_{i,i+j}$, and this generates $W_{i+j+1,k}$. And together they equal $T_{i,k}$; this is what we want. If that is the case, so not i k, i plus k. If that is the case, then the variable A can be included into this set $T_{i,i+k}$; and like that we keep building. And at the end of the loop, we would have built this set; we start with T11, T22 and so on.

And basically we follow this from top to bottom. Finally, we would have built the set $T_{1,n-1}$, $T_{2,n}$ and finally T1, n. So, we would have built $T_{1,n}$ which corresponds to all the variables that derive the entire string, and then we check whether the start variable is in that set. If the start variable is indeed in that said, we say that w is generated by this language; if it is not, we say that W is not generated in this language, sorry by this grammar. So, this is the high level picture; things will be clear once we work out an example.

(Refer Slide Time: 26:24)



$A \to BC$

$\underbrace{w_{i,i+j} \quad w_{i+j+1,i+k}}_{w_{i,i+k}}$

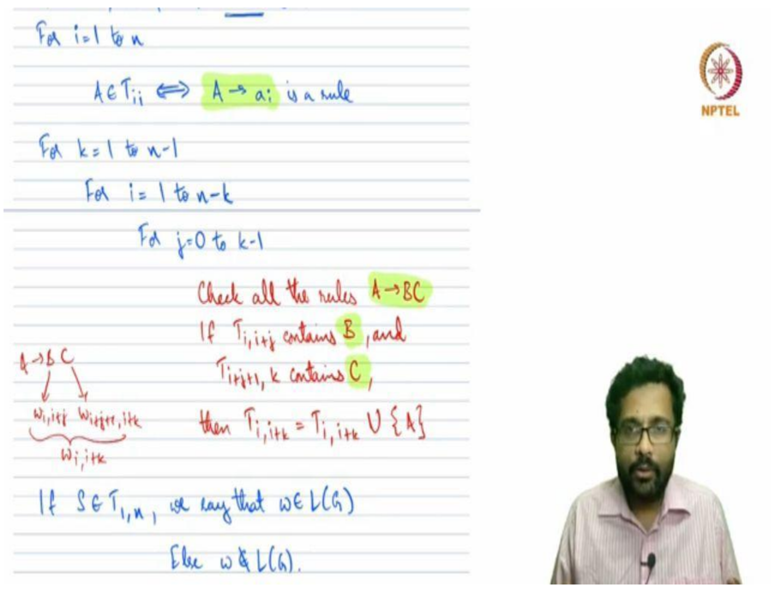
```
Check all the rules $A \to BC$
If $T_{i,i+j}$ contains $B$, and
$T_{i+j+1,k}$ contains $C$,
then $T_{i,i+k} = T_{i,i+k} \cup \{A\}$
```

If $S \in T_{1,n}$, we say that $w \in L(G)$

Else $w \notin L(G)$.

Running Time: $O(n^3 r)$ where r is the number of rules. For a fixed grammar, r is considered to be a constant. $\to O(n^3)$ algorithm

Correctness: Is evident from the algorithm.



```
For i=1 to n
    $A \in T_{ii} \iff A \to a_i$ is a rule
For k=1 to n-1
    For i=1 to n-k
        For j=0 to k-1
            Check all the rules $A \to BC$
            If $T_{i,i+j}$ contains $B$, and
            $T_{i+j+1,k}$ contains $C$,
            then $T_{i,i+k} = T_{i,i+k} \cup \{A\}$
```

$A \to BC$

$\underbrace{w_{i,i+j} \quad w_{i+j+1,i+k}}_{w_{i,i+k}}$

If $S \in T_{1,n}$, we say that $w \in L(G)$

Else $w \notin L(G)$.

So, the running time is $\theta(n^3 r)$, where n is the length of the string that is for which we need to check; and r is the number of rules, so r is the number of rules in the grammar. Why is it n cubed r? Because the major is the time consuming part. Basically, we have to, we have this three nested for loops; each of which can go upper bounded by length n. So, there maybe some improvement, but it is still upper bounded by length n. So, it is n times n times n which is n cubed. And over here, we need to check all the rules. So, if there are r rules, so there is a further multiplicative factor of r; so that is how we get n^3 r.

But, the model that we would like to think of is that we have the grammar already there. And now the input is just the string that is coming in. So, r being a parameter of the grammar it can take, we can think of it as a constant; because the input string could be a varying length, so, that is a real variable. So, r is considered to be a constant. So, we say that we usually say there is an order n cubed algorithm; because r is a parameter that corresponds to the grammar itself. And the correctness is fairly evident from the procedure in which we constructed, the correctness is evident.