

Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Chomsky Normal Form

(Refer Slide Time: 00:16)

Chomsky Normal Form

→ A standardized form for a CFG.

→ One of the main goals is to be able to check if string w is generated by grammar G .

→ What we want: Simple rules that are easy to check.

→ What we don't want: loops, empty derivations
 Unless rules.

Consider $S \rightarrow OS1 \mid SOS1 \mid T$
 $T \rightarrow \epsilon \mid \epsilon$

→ One of the main goals is to be able to check if string w is generated by grammar G .

→ What we want: Simple rules that are easy to check.

→ What we don't want: loops, empty derivations
 Unless rules.

Consider $S \rightarrow OS1 \mid SOS1 \mid T$ 1010000
 $T \rightarrow S \mid \epsilon$ $10S10T00$
 $10S1000$
 101000

Def 2.8: A context-free grammar is in Chomsky



Hello and welcome to lecture 17 of the course theory of computation. In the previous week, week-3, in the last lecture of week-3, we started seeing context-free grammars. So, we saw the definition. We saw some examples, and we even saw the definition of ambiguity like what it means for a grammar to be ambiguous. In this lecture, we are going to see a specific form or a

structure for context-free grammar, which is called the Chomsky normal form. So, this was invented by Noam Chomsky in I think late fifties; so, this is it.

This gives a certain structure to the Chomsky context-free grammar. So, context-free grammar as we know, it is a set of rules where each variable gives rise to multiple derivations. So, it is a bunch of rules, so, where and we want to know what are the strings that we can form out of this grammar. So, let us see why we need such a structured set of rules, which is the Chomsky normal form. So, as I said, this is the standardized form. So, already the definition for the context-free grammar gives some standard. So, here we are going to be imposing more structure than what is insisted by the definition of the context-free grammar. So, what is, so let us to make it easier to motivate this; let us see what is the main goal of a context-free grammar.

So, or what is the one of the things that we need to do with it? So, one of the things to think of grammar generating all strings, which are like the in the syntax of a programming language; and the computer has to actually read the string and interpret it. So, it has to understand what, it has to breakdown the structure and try to interpret the structure of the string, and see what this really means that one line of the code or whatever. So, it has to be able to understand how the string was derived by the grammar. So, this is the one of the main goals of what we want to do, or what one should be aiming to do with a context-free grammar. So, they should be able to understand how like whether a string w is generated by a grammar; and if it is elevated, how is the generation? How exactly is it derived?

So, this is what one of the things that we want to do. And the rules should be in such a way that it should enable these kinds of questions. So, it should be easy to check whether the string w is derived from the grammar; and if so, how it is derived. So, we will briefly illustrate how in general context-free grammar it may not be that easy, and how the Chomsky form Chomsky normal form makes it somewhat easier. So, consider this particular the context-free grammar that is there in the bottom of the screen. $S \rightarrow 0S1$ and $1S0S1$ and T ; three different rules for S , and t gives S or epsilon. So, there are a couple of issues here. So, let us say we get a string; let say 1010000; so it is a string of seven.

So, now one possibility of trying to check whether this string is derived from this language or this grammar is to actually try out all possible derivations of a certain length. So, you try out like you basically try out all the rules, and kind of it may come out as like a tree; and see whether

what are the strings of length two, length three, length four and so on. And once you reach length seven, you see whether this particular string is 1010000; whether this string is derived from this particular grammar or not. And basically, you may also have other strings which may have 10S01T; some other strings may there of that out of length seven which have variables in them.

But, the point is that it is enough to consider all strings up to length seven. But, then the issue is that it would have been fine if, if the length of the strings that are derived do not become smaller. If the, when you apply rules if the length of the string only keeps getting bigger, I can stop at a certain length which is the length of the string that we want. But, in this particular grammar that is not the case, because we have a certain rule here; we have a rule that $T \rightarrow \epsilon$. So, potentially, I could have a string like this 10S10T00. And in the next step, this T could go to epsilon; and I could get this 10S100. And even this S can go to T, and then T can go to epsilon; so, the next step I could get this, so the string can become smaller.

So, this is why this empties the rules that derive empty strings; so, the rules like this T gives epsilon, this is not desirable. Another issue is that in a program in any program, we do not want, we do not like loops; loops, meaning a gives b, and b goes back to a. So, some kind of infinite loops and that sort of thing is happening here. If you see there is $S \rightarrow T$, and $T \rightarrow S$. So, when you are trying to structurally bring down all possible derivations, this can possibly create problems. So, even now I am not really explaining a proper algorithm, but I am explaining issues that are likely to arise when we try to develop an algorithm.

So, $S \rightarrow T$ and $T \rightarrow S$ can result in problems. So, empty derivations I already explained. And this is a loop and one way to remove loops is to not allow single derivations; meaning derivations like one variable giving another. So, this is why one variable gives another, we do not want; and there is also this thing called useless rules. So, some useless meaning: what if some rule is never actually used in generating a string? Some rules are there but it never is; if you try to use that rule, it never ends in the, we never get an actual string by applying such rules. So, these are kind of undesirable things.

And what we want is a grammar that enables the creation of an algorithm; so that is the brief motivation of why we would like more structure. And with this brief example, I hope it is clear why in general the structure of context-free grammar is not strong enough to ensure such an algorithm. So, that is the motivation for Chomsky normal form.

(Refer Slide Time: 07:35)

10 1000

Def 2.8: A context-free grammar is in Chomsky Normal Form if every rule is of the form

$$\begin{array}{ll} A \rightarrow BC & A \rightarrow BC \rightarrow cBC \\ A \rightarrow a & \rightarrow cBC \\ & \rightarrow cBC \\ & \rightarrow cBC \end{array}$$

where a is a terminal, A, B, C are variables and B and C are not the start variable. Also, we allow $S \rightarrow \epsilon$.

One of the advantages of the Chomsky Normal Form is the "predictability" in the derivation of a string. Any string of length n requires exactly $2n-1$ steps in its derivation.

and B and C are not the start variable. Also, we allow $S \rightarrow \epsilon$.

One of the advantages of the Chomsky Normal Form is the "predictability" in the derivation of a string. Any string of length n requires exactly $2n-1$ steps in its derivation.

Also Chomsky NF results in an efficient algorithm for checking if w is generated by G .

Natural question at this stage. What if there is no equivalent CFG that is in the Chomsky Normal Form?



So, what is Chomsky's normal form? It is a grammar and the grammar is said to be in Chomsky normal form, if every rule is in one of the following types. So, what are the following types? One is A gives BC , and the other is A gives small a . So, A gives BC means one variable giving rise to deriving two different two variables, so $A \rightarrow BC$. And the second is that one variable giving rise to a single terminal, so here is a single terminal; and the two more things that we want. We do not want the start variable to appear on the right side of any rule. So, B and C should not be the start variable.

We also do not want empty rules; so any rule of the form $A \rightarrow \text{empty string}$ is not allowed. The only exception is that if we never allow $A \rightarrow \text{an empty string}$, what if the language that we are trying to create also has the empty string? So we want to somehow account for that. So, if the language that we are trying to create has an empty string, we allow one exception for this empty string rule. We allow the start variable giving an empty string, but not for any other variables. So, basically, the entire CFG is such that of the rules of this type, $A \rightarrow BC$ or $A \rightarrow a$; so, B and C cannot be the start variable. And epsilon does not feature in the right hand side of any rule, except possibly start gives empty string.

So, this is the, this is the structure; $A \rightarrow BC$, A gives small. And where A is a variable, B and C are variables which are not the start variable. And if needed, we allow start gives an empty string; we do not allow another variable keeping empty string. So, if the entire context-free grammar consists of only rules like this, of this type, we say that the context-free grammar is in the Chomsky normal form. So, one of the advantages is that now you can see that say, you can see that every variable now we apply the first type of rule; A gives BC; the length of the resulting string increases by 1. So, maybe you have things like A gives BC, now maybe $B \rightarrow CD$ is there.

So, every time you apply the rule of the first type, the length of the resulting string increases. If you apply the rule of the second type, let us say C gives c or something. The length does not change and it is not that difficult to see that to get from A. So, now maybe see the D gives a; and C gives c. So, now if you can see from the start variable A to get a string of length three, we use one, two, three, four, five, steps one, two, three, four, five. So, this is something that is easy to see, which I tried to, I will ask you to show as an exercise that any string requires exactly $2n$ minus 1 steps for the derivation. So, a string of length n, so maybe the time is to write of a string of length n requires exactly $(2n-1)$ steps in its derivation.

It is easy to see because first from the start variable, to go to a string of variables of length n, we need to do n - 1 steps. And then we have a variable of, there is a string of n variables and each one needs to be replaced to another n steps. So, that is pretty much the proof; but I leave it to you to formally reason it. So, there is a predictability. The point is that it is not like if you come a certain way or a certain string can be delivered in three steps. But, another string of the same

length requires 10 steps. So, now we know, if there is a string of length 5, it requires exactly 9 steps, whichever way you take. In a string of length 10 requires exactly 19 steps.

So, this is some kind of structure and order that we are getting as a result of the structure imposed by the Chomsky normal form. And another thing that we will see very shortly, maybe not, maybe in the next lecture is that, because of the structure, we are able to have a gated algorithm that is able to check whether a string is generated by a grammar. So, there is an efficient algorithm; this is called the CYK algorithm, which we will see in the next lecture. So, this is all good about the Chomsky normal form. So, the natural question that one may ask at this stage is okay. So, it is, I agree things are good that we have all this structure; but what if I have a grammar that cannot be put in this structure?

So, is a structure too rigid that it is not general; that is a natural question that one may ask. The answer is interestingly that it is not too rigid; and that if you give me any context-free grammar, I can convert it into a grammar into Chomsky normal form. So, any context-free language can be written as a context, represented by a context-free grammar in the Chomsky normal form. Any context-free grammar or any context-free grammar or language is generated by a grammar in the Chomsky normal form. So, so we will get into the proof.

(Refer Slide Time: 13:43)

CFG in Chomsky normal form.

Proof: We provide a process that demonstrates how to convert any CFG into Chomsky NF.

Optional Step: Remove useless symbols and productions

$S \rightarrow AB \mid a$
 $A \rightarrow a$

$\rightarrow B$ is a useless var.

1. Add new start variable S_0 .

$S_0 \rightarrow S$



10.510.00
10.1000

$1 \rightarrow S | \epsilon$

Def 2.8: A context-free grammar is in Chomsky Normal Form if every rule is of the form

| | |
|--------------------|------------------------------------|
| $A \rightarrow BC$ | $A \rightarrow BC \rightarrow CBC$ |
| $A \rightarrow a$ | $\rightarrow cDC$ |
| | $\rightarrow c \& C$ |
| | $\rightarrow caC$ |

where a is a terminal, A, B, C are variables and B and C are not the start variable. Also, we allow $S \rightarrow \epsilon$.

One of the advantages of the Chomsky Normal Form is the "predictability" in the derivation of a string. Any



The proof is fairly direct in the sense that it gives us a process by which we can convert any context-free grammar into the Chomsky normal form. So, briefly go over the process, and instead of just explaining the process, I will, we will go through an example. So, what is a process? There is an optional step which will not, which will just briefly touch upon. So, first is to remove useless symbols in productions. So, what do I mean by that? So, suppose this is the grammar. The grammar is this, the one in the box. So, this grammar if you see it is S gives AB and a ; and A gives a . So, if you see that the variable b is never used here, so, there is nothing defined for what the variable b yields.

So, if you use the rule A and use AB , we cannot, there is no real next step to go. So, the B is a useless variable here, so B is a useless variable. So we, so we may as well remove the rule as well, S gives AB . So, this is, so this is an example of a very simple grammar where there is a useless rule or a useless variable. Anyway, so this is one example. So, the first step is to remove useless rules and variables; this is an optional step. This is not required for the rest of the process, so that is why I am saying it is an optional step. So, just to remind ourselves, what is Chomsky normal form? Every rule should be of the form A gives BC , where B and C are not the start variable; or A gives small a , where a is a terminal.

And we are not allowed to have Epsilon features in the right hand side of any rule, except the start variable gives epsilon. So, the first thing to tackle is maybe. So, maybe, I have the proof written here; but I will not go through the proof. Instead, we will work as an example.

(Refer Slide Time: 16:13)

Example: $S \rightarrow TST \mid aB$
 $T \rightarrow B \mid S$
 $B \rightarrow b \mid \epsilon$



1. Add S_0 as new start variable.

$S_0 \rightarrow S$

$S \rightarrow TST \mid aB$
 $T \rightarrow B \mid S$
 $B \rightarrow b \mid \epsilon$



Consider $S \rightarrow OSI \mid IOSI \mid T$ 1010000
 $T \rightarrow S \mid \epsilon$ $10S10T00$
 $10S1000$
 101000



Def 2.8: A context-free grammar is in Chomsky

Normal Form if every rule is of the form

$A \rightarrow BC$ $A \rightarrow BC \rightarrow CBC$
 $A \rightarrow a$ $\rightarrow cBC$
 $\rightarrow caC$
 $\rightarrow caC$

where a is a terminal, A, B, C are variables and B and C are not the start variable. Also, we allow $S \rightarrow \epsilon$.



So, suppose this is maybe more illustrative. Let us suppose this is a, this is the CFG; S gives $TST \mid aB$, T gives $B \mid S$, and B gives $b \mid \epsilon$. So, there are three variables: capital S , capital T and capital B ; two terminals small a and small b . So, there are many violations here. So, first of all, there are rules like this, TS like the rules where the right hand side has three variables. The right hand side has a variable and a terminal, a comma B , small a , capital B . There are rules T gives B which is the rule where the right hand side is a single variable. There are epsilon rules, where B gives epsilon. And the right hand side also has the start variable S gives TST , and T gives S ; so there are many kinds of violations.

So, we will see how to fix all of this. So, we want all the rules to be of the form $S \text{ gives } A \text{ gives } BC$, where B and C are variables, and A gives small a . So, the first thing to do is so we do not want the start variable on the right side. So, the simplest way to handle that is to just introduce a new start variable; so we will add S_0 as a new start variable. And in order to now connect that with the rest of the grammar, we will just add a new rule $S_0 \text{ gives } S$. So, now we have modified the grammar by adding a new start variable, but the grammar is equivalent. So, every, so if you see, we will keep the grammar equivalent; we will modify it, but will ensure that the set of strings generated by the grammar does not change.

So, we will keep restructuring it, but we will ensure equivalence at all stages; so, we added this. So, now we have the other violations still, we have rules where three variables are on the right side; we have one variable on the right side, variable terminal combined on the right side. But, we do not have a start variable on the right side; so that is one thing fixed. Now, let us try to eliminate the epsilon roots. So, we have one rule here: $B \text{ gives an empty string}$; so, first let us remove that. So, I have not written it down, but let us try to work it out.

(Refer Slide Time: 19:09)

2. Remove $B \rightarrow \epsilon$

$$S_0 \rightarrow S$$

$$S \rightarrow TST \mid aB \mid a$$

$$T \rightarrow B \mid \underline{\epsilon} \mid S$$

$$B \rightarrow b$$

$T \rightarrow \epsilon$



So, this is the rule. So, what we will do is whenever B appears in the right side, so we have the rule $S_0 \text{ gives } S$, let that B ; and whenever B appears in the right side, for instance here, we will also include what will happen if. So, there are two places where B appears, so here as well as here. So, we will also by eliminating this epsilon, the epsilon over here for eliminating this, we

will also try to ensure that the grammar does not, the set of strings does not change. So, let us see what I mean by that. So, I have the rule $S \rightarrow TST$ which is unchanged. Now, $S \rightarrow a$, B is there; but we know that B could have gone to epsilon, but now we are removing that. So, which means we account for it here by allowing; so what would happen if B goes to epsilon?

So, we take that in this stage itself. So, if B went to epsilon, this a , B would have been just small a . So, we just add a rule small a to kind of take into account the effect of what would happen if B goes to epsilon. Similarly, we have the rule $T \rightarrow B$ and S . So now, if B went to epsilon, this entire thing would also go to epsilon; so we add $T \rightarrow \epsilon$ and S remains unchanged. And then B goes to, now B goes to epsilon can be removed; because wherever B appeared in the right hand side, we already took into account; took care of what would happen if B went to epsilon and we have updated. So, this is the updated set of rule.

So, here we have a , B in the hand side and we the possibilities when B went to epsilon, where this had to be replaced by small a ; and here we had B itself. So, B has to be written and then we also added; when B goes to epsilon, this has to be epsilon. So, now we have successfully removed $B \rightarrow \epsilon$. But, we have now generated or we have created a new issue; we have this epsilon now. We have created a new epsilon. So, so now that would mean that I have to take into account; I have to remove this. And there is only one place where T appears in the right hand side, which is this TST . And I have to modify this rule so that this epsilon that appears in the right hand side has to be; so, we want to remove this $T \rightarrow \epsilon$.

So, what are the possibilities when T goes to epsilon? So, this particular rule has two T s in the right hand side T . And we have to account for all the possibilities; what if both of them go to epsilon? What if one of them goes to epsilon? So, the modification that we need to do is, so we need to remove.

(Refer Slide Time: 22:30)

$$T \rightarrow B \mid \epsilon \mid S$$
$$B \rightarrow b$$
$$T \rightarrow \epsilon$$
$$S \rightarrow S$$
$$S \rightarrow TST \mid ST \mid TS \mid S \mid aB \mid a$$
$$T \rightarrow B \mid S$$
$$B \rightarrow b$$

3. Unit Rules



We are going to remove this $T \rightarrow \epsilon$. So, $S \rightarrow S$ gives S which is standard, and then S gives; maybe I will just move it. S gives, so we have TST . Now, we want to, we are removing $T \rightarrow \epsilon$. So, multiple things are possible; the first T could go to epsilon, which would give us ST . The second T could go to epsilon which would give us TS ; maybe both these could go to epsilon which would give us just S . Then, we have aB and then we have a , and then T gives $B \mid S$, and then B gives b . So, now we have taken into account, we have **removed $T \rightarrow \epsilon$** . And whatever impact the grammar had because of that removal that we have taken into account by modifying the rules for S . So, we added new rules here in order to absorb the impact of the removal of $T \rightarrow \epsilon$; so this is good now.

We have now removed epsilon rules; so we have now two things that are taken care of. We do not have a start variable on the right side; we also do not have any epsilon rules. There are other things that are still there. For instance, we have this unit rule; we have $T \rightarrow B$ which we do not want. We have small rules like this $T \rightarrow B$; we have long rules like this, $S \rightarrow TST$. And we also have rules like $S \rightarrow aB$, $S \rightarrow a$, $S \rightarrow B$ which are not in the structure. So, next goal is to address these anomalies.

(Refer Slide Time: 24:32)

$$\begin{array}{l} T \rightarrow B | S \\ B \rightarrow b \end{array}$$

↓
redundant rule
can be removed.



3. Unit Rules

$$S_0 \rightarrow S$$
$$S_0 \rightarrow TST | TS | ST | aB | a$$
$$S \rightarrow TST | TS | ST | aB | a$$
$$T \rightarrow B | S$$
$$B \rightarrow b$$



$$T \rightarrow \epsilon$$
$$S_0 \rightarrow S$$
$$S \rightarrow TST | ST | TS | \cancel{S} | aB | a$$
$$T \rightarrow B | S$$
$$B \rightarrow b$$

↓
redundant rule
can be removed.



3. Unit Rules

$$S_0 \rightarrow S$$
$$S_0 \rightarrow TST | TS | ST | aB | a$$



So, what are the unit rules? Let us see. So, maybe the first thing I have already filled out. So, the first unit rule is S_0 gives S ; this is the unit rule which is not allowed as per the rules of the context, sorry Chomsky normal form. So what we do is, S_0 gives S , and then S could give all these things like TST , ST , TS etcetera. Before getting there, let me just make one more statement. So notice that we have one kind of redundant rule here; we have S gives S here. So, this is a redundant rule, because it does not do anything to any grammar. S gives S , it is not even a valid like it is there is no change; so we will remove that, can be removed. Now, coming back to the removal of unit rule, so S_0 gives S is what we have to we want to remove.

So, when we want to remove this; so instead what we will do is whatever S can yield, we will just add that to the productions of S0. So, we will remove S0 gives S; but, S could give T S T, S T, T S, a and a B and a. So, now that is what we are doing here. S0 also now gives TST, TS, ST; order is changed ST and TS, but it is the same, a B and small a. S also gives the same things, T gives B and S; and B gives small b. So, while we removed S0 → S whatever we could get as a result of that, whatever S could produce that we put at the right hand side of S0.

(Refer Slide Time: 26:37)

3. Unit Rules

$S_0 \rightarrow S$

$S_0 \rightarrow TST \mid TS \mid ST \mid aB \mid a$

$S \rightarrow TST \mid TS \mid ST \mid aB \mid a$

$T \rightarrow B \mid S$

$B \rightarrow b$

$T \rightarrow B$

$T \rightarrow b \mid S$

$T \rightarrow S$

$S \rightarrow S$

$T \rightarrow B$

$T \rightarrow b \mid S$

$T \rightarrow S$

$S_0 \rightarrow TST \mid TS \mid ST \mid aB \mid a$

$S \rightarrow TST \mid TS \mid ST \mid aB \mid a$

$T \rightarrow TST \mid TS \mid ST \mid aB \mid a \mid b$

$B \rightarrow b$

4. Restructure rules with more than one symbol in the R.H.S.



So, next thing is T gives B. So, T gives B here, now what would what it would mean? What would it mean? T gives B. So, now what are the things that capital B produces? So, that rule

what we will do is we will replace this rule. We will replace this rule with whatever B can small b can produce which is that small b, will remove T gives capital B and we get this. But then, the next thing that needs to be addressed is T gives S. So, what all T gives S, S can yield, S can yield all this T S, T S T, a B all that; so we need to add that as well. So, now we modify this into T gives. So, let me list down all the outcomes of what T can give. What S can give into T? T S T, T S, S T, a B, a.

And this B also we need to include this particular B, so we include that. So, now that is the result of two. So, now this rule has to be replaced by this everything else is done; so, now let me just. So, this is the entire set of grammar, S0 gives TST, TS, ST, aB and a. S gives TST, TS, ST, aB and a; T gives this one, TST, TS, ST, aB, a and b. Maybe what I will do is I will just; sorry, I will just erase and write the whole thing again. Maybe it is easier if we see the rules at one place; S0 gives TST, TS, ST, aB and a; S gives the same thing. And finally, we have B gives small b; so this is the entire grammar, S0 gives this. So, now we have taken care of start variable not appearing in the right side. We have taken care of rules that are designing the empty string.

We have also taken care of rules where of the type a gives b, where a and b are variables. So, exactly one variable is produced. Now, the only other issue that we have are one which are rules which are producing more than two variables or not of the proper form; so one variable and one terminal things like that. This what we do is now this is also not in the structure of the Chomsky normal form. So, what we will do is we will introduce new variables. So, whenever three variables are produced like T gives T S T we will add an intermediate step, so that it will first, it will derive two variables and then derive three variables. So, by introducing new variables, we will restructure this. So this is what we have; So let us see what we do.

(Refer Slide Time: 30:50)

↓

$$\begin{aligned} T &\rightarrow S \\ S_0 &\rightarrow TST \mid TS \mid ST \mid aB \mid a \\ S &\rightarrow TST \mid TS \mid ST \mid aB \mid a \\ T &\rightarrow TST \mid TS \mid ST \mid aB \mid a \mid b \\ B &\rightarrow b \end{aligned}$$

4. Restructure rules with more than one symbol in the RHS.

let $TS = \underline{Z}$, $a = \underline{A}$.

$$B \rightarrow b$$

4. Restructure rules with more than one symbol in the RHS.

let $TS = \underline{Z}$, $a = \underline{A}$.

$$\begin{aligned} S_0 &\rightarrow ZT \mid TS \mid ST \mid AB \mid a \\ S &\rightarrow ZT \mid TS \mid ST \mid AB \mid a \\ T &\rightarrow ZT \mid TS \mid ST \mid AB \mid a \mid b \\ Z &\rightarrow TS \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$


4. Restructure rules with long R.H.S.



Example: $S \rightarrow TST \mid aB$
 $T \rightarrow B \mid S$
 $B \rightarrow b \mid \epsilon$

1. Add S_0 as new start variable.

$S_0 \rightarrow S$

$S \rightarrow TST \mid aB$
 $T \rightarrow B \mid S$



So, we will introduce two new variables; one is Z, which we want to stand in for T S; and then capital A, which we want to stand in for small a. Because, basically, we want to; this is something that we have problem with T S T; and this is something that we have problem with a, B. So, let us see how we fix that; so, S gives. So, instead of T S T, I will say that Z T; T S will remain as it is, because it is in the form; S T will also remain as it is. Now, small a, B I will write it as A B, and small a is also allowed; so this is what we do. And similarly, so this is S_0 ; similarly, S also we do the same thing. T also we do the same thing; for T, we have one more rule.

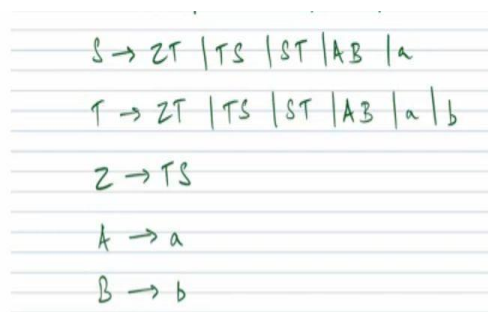
And now, this we have to formalize, this particular thing that I said here T S gives Z. Or so, we want Z to stand in for this T S. So, we have this new rule Z gives T S, and we have this other rule that A gives small a. So, now you see that now, whatever modifications we did, S_0 gives T S T et-cetera. Now, everything is still drivable from this with this change. The only thing that is not include is B gives small b; so, we have written that. And now that completes the transformation. So, now if you see, there are no epsilon rules, start variable is not in the right hand side. There are no unit rules, meaning rules of the type S gives T or something like that.

There are no rules where other structures appear long, like all the rules of the form A gives B C, or T gives a B. One variable gives two different variables or two variables, or of the form A gives small a, where one variable gives a terminal. And you can verify that; you can go through

the changes, go through the modification and you can indeed verify that whatever string that was produced by the original grammar which was this. All of this are still produced by the transformed grammar, and whatever was not produced are still not produced. We are not whatever the resulting grammar that we have is an equivalent grammar.

It is kind of evident in the way we went about transforming this grammar. But, perhaps if you are confused or if you are unsure, you can still verify this. Basically, whenever we modified a rule or whenever we removed a rule, we ensure that whatever impact the removal have is causing, that is absorbed by some other modification of some other rules.

(Refer Slide Time: 34:24)


$$\begin{aligned} S &\rightarrow ZT \mid TS \mid ST \mid AB \mid a \\ T &\rightarrow ZT \mid TS \mid ST \mid AB \mid a \mid b \\ Z &\rightarrow TS \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

Now the above CFG is in Chomsky Normal Form and by construction, we have ensured that it is equivalent to the original grammar.



Chomsky Normal Form

→ A standardized form for a CFG.

→ One of the main goals is to be able to check if string w is generated by grammar G .

→ What we want: Simple rules that are easy to check.

→ What we don't want: loops, empty derivations
Useless rules.



$T \rightarrow s | \epsilon$ 10810T00
1081000
101000



Def 2.8: A context-free grammar is in Chomsky

Normal Form if every rule is of the form

$A \rightarrow BC$ $A \rightarrow BC \rightarrow CDC$
 $A \rightarrow a$ $\rightarrow cDC$
 $\rightarrow cAC$
 $\rightarrow cac$

where a is a terminal, A, B, C are variables and B and C are not the start variable. Also, we allow $S \rightarrow \epsilon$.



One of the advantages of the Chomsky Normal Form

and B and C are not the start variable. Also, we allow $S \rightarrow \epsilon$.



One of the advantages of the Chomsky Normal Form is the "predictability" in the derivation of a string. Any string of length n requires exactly $2n-1$ steps in its derivation.

↓
EXERCISE!

Also Chomsky NF results in an efficient algorithm for checking if w is generated by G .



Natural question at this stage. What if there is no equivalent CFG that is in the Chomsky Normal Form?

So, now the grammar is in Chomsky normal form, the above CFG is in Chomsky normal form; and by construction, we have ensured that it is equivalent to the original grammar. So, that is basically the kind of sort of constructive proof. I will also share these notes where verifies in the proof, I actually explained how to do this. But, instead of going through the proof, we just went through the constructor procedure. So, that is it for this lecture. Basically, we introduced Chomsky normal form, which is a standardized form, where the rules are of this two types; sorry these two types. A gives $B C$, where B and C are variables or A gives small a , where small a is a terminal.

And in addition, we allow S gives empty string and start variable should not appear on the right hand side. And we showed that this is still general, in the sense that any grammar, any context-free grammar can be converted into Chomsky normal form without an equivalent, as an equivalent context-free grammar that satisfies these structures. And then the advantage was that this gives a structure, and we promised that we will get an efficient algorithm for deciding whether a string is generated by a grammar, if the grammar is in Chomsky normal form. So, we explained the transformation of how to convert any general grammar into Chomsky normal form.

And we said that there is an efficient grammar, efficient algorithm sorry; this algorithm will be covered in the next lecture. It is called CYK algorithm. It is an algorithm that base that uses dynamic programming so we will see that in the next lecture. Thank you.