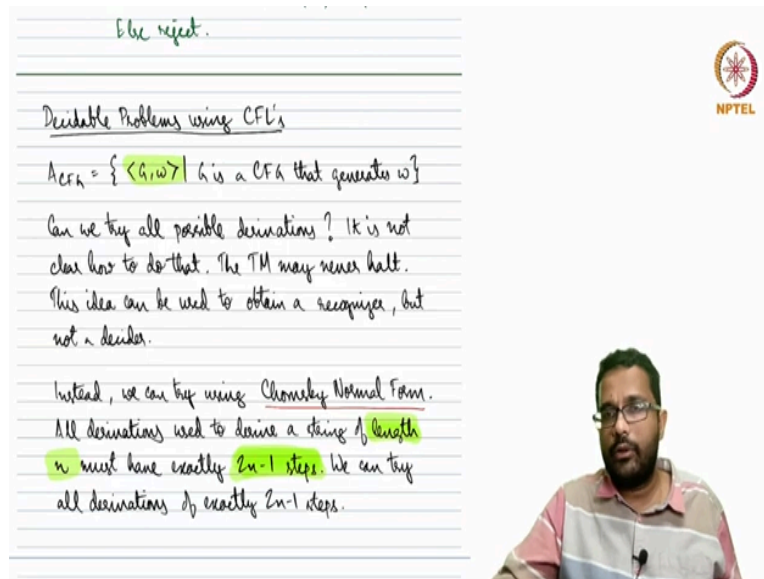


**Theory of Computation**  
**Professor Subrahmanyam Kalyanasundaram**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Lecture 39**  
**Decidable Problems Concerning Context Free Languages**

(Refer Slide Time: 00:16)




Else reject.


Decidable Problems using CFL's

$A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates } w \}$

Can we try all possible derivations? It is not clear how to do that. The TM may never halt. This idea can be used to obtain a recognizer, but not a decider.

Instead, we can try using Chomsky Normal Form. All derivations used to derive a string of length  $n$  must have exactly  $2n-1$  steps. We can try all derivations of exactly  $2n-1$  steps.





Hello and welcome to lecture number 34 of the course Theory of Computation. So, in the past lecture number thirty-three we started with decidable languages, we talked about decidable languages that were built on regular languages. And in this lecture, we will talk about decidable problems that are built on context free languages. So, just like we saw acceptance problems of regular languages, the first thing that we will see is the acceptance problem of a context free language. It is a (give) so meaning we are so, that it is  $A_{CFG}$ . So, we are given a context free grammar  $G$  and a string  $W$  and we are asked whether this context free grammar generates the string  $W$ .

Here again the naive way that one may guess. So, we have the grammar, so we have some variables, some rules, some production rules etcetera. We could try generating various possibilities. So, from the start variable, you try all kinds of possibilities and you look and you check whether the given string  $W$  is generated by the grammar, but, the problem is it is not clear when to stop because there are so many possibility there could be so many possibilities and it is not clear when you are supposed to stop and which branches you take when to stop etcetera.

So, you can try out some simple ways, then some other ways, some other ways and so on, but it is not clear when you stop. So, if you never get the string  $W$ , if by chance you happen to get the string  $W$  in one of these derivations, then you can say yes,  $G$  generates  $W$  and you can accept, but if you do not see it, then how long will you keep trying out things? So, meaning, if  $W$  is not generated by  $G$ , you will never, you will never possibly halt. So, that is also an issue. So, that is why I am saying here that this idea may result in a recognizer, but cannot provide a decider because if  $W$  is not generated by  $G$  there is no clear way to reject.

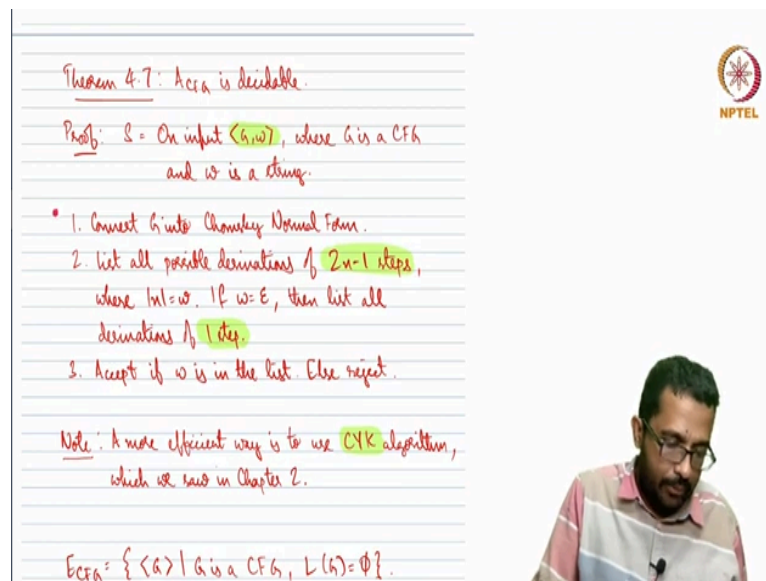
But, the problem here is that we do not know when to stop it. So, if there are some derivations that you can try and then we say, we tried many things, but now, we are sure that  $W$  is not generated by  $G$  that that should be a good thing so, is there some something that will give us a guarantee that if you try these possible derivations, these possible generations, and you never got the string  $W$ .

Now, you are never going to get  $W$  like suppose there is such a guarantee thing you try this and if you cannot get  $W$  till this point, then you are not going to get it. If that guarantee is there, then we can build a decider, but that guarantee is given to us by the Chomsky normal form. So, if you recall, we saw Chomsky normal form in second chapter and importantly, we saw it all suppose the string is of length  $n$ , then any derivation using the Chomsky normal form must take exactly  $2n - 1$  steps, any derivation of that string must take exactly  $2n - 1$  steps meaning if the string is of length ten it must exactly use nineteen steps, not more not less, exactly nineteen steps.

So, what we can do is? So, we have the grammar, we have some variables, some rules etcetera, you can simply try all possible derivations of nineteen steps. So, you start with the start variable, you try out all possible ways to replace the start variable by another using a rule then each of these possibilities you try out all possible next steps and all possible next, next steps and so on. But if you know that you only have to try out  $2n - 1$  steps. So suppose the string as I said is of length ten then we know that we only have to try out all possible derivations of nineteen steps.

So we can do that we can try out all possible derivations of nineteen steps because we have to we may have to try out all possible but we know that we can stop after nineteen steps, if you do not get  $W$  after nineteen steps, we know we are not going to get it in the future, because  $W$  is of length ten, if you do not see  $W$  in nineteen steps, we are not going to see it after twenty-one or twenty-five steps.

(Refer Slide Time: 05:16)



Theorem 4.7:  $A_{CF}$  is decidable.

Proof:  $S = \langle G, w \rangle$ , where  $G$  is a CFG and  $w$  is a string.

- 1. Convert  $G$  into Chomsky Normal Form.
- 2. List all possible derivations of  $2n-1$  steps, where  $|w|=n$ . If  $w = \epsilon$ , then list all derivations of 1 step.
- 3. Accept if  $w$  is in the list. Else reject.

Note: A more efficient way is to use CYK algorithm, which we saw in Chapter 2.

$A_{CF} = \{ \langle G \rangle \mid G \text{ is a CFG, } L(G) = \emptyset \}$ .

So, this helps us build a decider what we do is given a grammar, we can convert into Chomsky normal form, this also something that is something that we have seen in the second chapter. All grammars can be reduced and can be converted to Chomsky normal form and equivalent Chomsky normal form, which means, whatever is accepted by  $G$  will be accepted by the grammar in Chomsky normal form and whatever is not accepted will not be accepted.

And now, you list out all possible derivations of  $2n - 1$  steps, so, this is important. In the Chomsky normal form, you try out all possible derivations of length  $2n - 1$ . If the string that we like so, you are given an input  $G$  and  $W$  and you are asked whether  $G$  generates  $W$ . So, if  $W$  is an empty string, so, now,  $2n - 1$  will be minus one, so, it does not make sense, but if it is an empty string there has to be a rule to start gives empty string.

So, if it is empty string then you just check whether you check all the derivations of length 1, otherwise you check all the derivations of length  $2n - 1$  and then see whether  $W$  is generated in any of these derivations, if it is generated you accept meaning there is some rule or some set of rules applied in some sequence generating  $W$ . If  $W$  is not generated that means it is, we have tried up to  $2n - 1$  steps, it is not going to be generated later also, so, you can safely reject.

So, what you do is we convert the grammar into Chomsky normal form with all the possible derivations of  $2n$  minus 1 steps and accept if  $W$  is generated and reject if it is not generated. So, the earlier way did not give us the confidence to reject if it was not generated, because,

there is simply no guarantee that we will be done after doing this many steps and Chomsky normal form is able to provide that assurance.

And one more point is one more related point is that we have actually seen an even more efficient algorithm, which was a CYK algorithm in the second chapter. In fact, in the book the CYK algorithm appears in the seventh chapter I think, but, for the purpose of this course, I think we advanced it and covered it in the second chapter. So, in fact, given a grammar  $G$  and a string  $W$ , the grammar  $G$  being in Chomsky normal form, we saw an efficient way to check whether  $W$  is generated by this grammar. So, CYK is even more efficient than blindly trying out all possible derivations of length  $2n - 1$ .

So, it uses a very dynamic programming approach, which is much more clever and smart, you end up saving a lot of work that you would otherwise redo. So, the CYK algorithm is actually another way to decide this, but since the question is whether we can decide or whether it is not decidable. Even the algorithm that is listed here, we will do. So, we are not really at this point, we are not really optimising on the resources that complete  $A_{CFG}$ ,  $A_{CFG}$  is decidable.

(Refer Slide Time: 08:46)



Proof:  $S = \langle G, w \rangle$ , where  $G$  is a CFG and  $w$  is a string.

1. Convert  $G$  into Chomsky Normal Form.
2. List all possible derivations of  $2^{n-1}$  steps, where  $|w| = n$ . If  $w = \epsilon$ , then list all derivations of 1 step.
3. Accept if  $w$  is in the list. Else reject.

Note: A more efficient way is to use CYK algorithm, which we saw in Chapter 2.

$$E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG, } L(G) = \emptyset \}$$

Theorem 4.8:  $E_{CFG}$  is decidable.



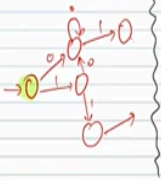
decides for DFA.

Theorem 4.4:  $E_{DFA}$  is decidable.

$$E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA, } L(A) = \emptyset \}$$



How can we check each and every string? <sup>Not</sup> clear.

Instead, we view the DFA as a graph. Is there a path from the starting state to an accepting state?



If there is a path, language is nonempty.

We can use BFS/DFS to check if an accepting



Note: A more efficient way is to use CYK algorithm, which we saw in Chapter 2.

$$E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG, } L(G) = \emptyset \}.$$

Theorem 4.8:  $E_{CFG}$  is decidable.

We cannot keep checking for each  $w$  if  $w \in L(G)$ . This will not be a decider. Instead, we use a strategy similar to the one used for  $E_{DFA}$ .

Page 1:  $\langle G \rangle$   
 Page 2: Initially list all the terminal symbols.

The next is  $E_{CFG}$ . So, given a grammar does it generate or does it generate the empty language? In other words, given a grammar is there even one string that it generates. If you recall in the previous lecture, we showed that  $E_{DFA}$  is decidable. So, we said that you cannot possibly try out all possible strings whether that string is accepted by the DFA instead we looked at the DFA as a graph. We looked at DFA as a graph and checked whether any accepting state is reachable from the starting state.

Based on this we decided, so, the in the case of  $E_{CFG}$  it is similar we can for the same reason as why we could not do  $E_{DFA}$  by brute force checking each string whether it is accepted by the DFA, for the same reason we cannot check if each string whether it is accepted or generated by the grammar.

So, we know that  $A_{CFG}$  is decidable but we cannot keep doing like the first string whether this string is generated by this grammar, second string the second string is generated with this grammar. If some string is generated then we know the language is not empty, but if the language is indeed empty, you will just keep checking soon after another after another, there are infinite strings and you will never end, so (there) this will not be a decider.

So, what we have to do is look at some other approach. So, in the case of DFA we try to see, we try to view it as a graph and check whether this state is the accept or one of the accepting states is reachable from the starting state.

(Refer Slide Time: 10:38)

This will not be a decider. Instead, we use a strategy similar to the one used for EDA.

Tape 1 | Rules G

Tape 1: (G)      Tape 2 | a b 0 1 A B E

Tape 2: Initially list all the terminals symbols.



- Repeat until no new variable is added.

$A \rightarrow a b a$   
 $B \rightarrow 0 0 A$   
 $C \rightarrow D A$   
 $E \rightarrow A B$

A is added to tape 2 if A can generate a string of terminals

For all A, variable in G:  
 If there is a rule  $A \rightarrow U_1 U_2 \dots U_k$  where all of  $U_1, U_2, \dots, U_k$  are in tape 2, then add A to tape 2.

- Check if start variable is in tape 2.

Tape 1 | Rules G

Tape 1: (G)      Tape 2 | a b 0 1 A B E

Tape 2: Initially list all the terminals symbols.

- Repeat until no new variable is added.



$A \rightarrow a b a$   
 $B \rightarrow 0 0 A$   
 $C \rightarrow D A$   
 $E \rightarrow A B$

A is added to tape 2 if A can generate a string of terminals

For all A, variable in G:  
 If there is a rule  $A \rightarrow U_1 U_2 \dots U_k$  where all of  $U_1, U_2, \dots, U_k$  are in tape 2, then add A to tape 2.

- Check if start variable is in tape 2.  
 - If it is not in tape 2, accept.  
 - Else reject.

$L(G) = \{ w \in \Sigma^* \mid \delta \Rightarrow w \}$

So, here we do something which is similar in spirit. So, you want to know whether any string is generated. So, what we do is, it may not be immediately, it is a slightly clever approach. So, what we do is, so, in the first tape, as we know multi tape is the same as single tape, you list the rules of G or you list the grammar G tape 1. And tape 2 what we do is? You initially list all the terminals it is a b 0 1 that is it, these are your terminals you list these things in tape 2.

Now, what you do is this? Now, you have this in the tape 2, tape 1 is just tape 1 will just continue to store the grammar. Now, you check if there is any variable that will now go through all the rules one by one now, suppose there is a rule that would say A gives a b a. So, now A derives a string that is entirely terminals.

$A \rightarrow aba$

So, which means the variable A is able to generate something that is entirely terminals. So, then we include A in tape 2. So, any variable that is able to generate a string of entirely terminals we include here. Now, suppose there is a rule let us say B gives 00A, now, we know that 0 and 0 are terminals and variable A is capable of deriving a string of entirely terminals. So because A derives a b a so basically what we do is we check whether there is a rule where the right hand side is entirely in the tape 2.

$B \rightarrow 00A$

So, first the right hand the tape 2 content only terminals. So now, we asked whether there is some rule where the right hand side is entirely in tape 2. So now A and 0 are there in tape 2. So now we add B to the tape 2. So, suppose there is a new rule where let us say C gives DA, then that you do not add C because D is not in tape 2.

$C \rightarrow DA$

Now, suppose there is a rule, let us say E gives AB where A and B are both variables, but both A and B are there in the tape 2. So, now you can include E also in the tape 2. So, like that we proceed. So, initially tape 2 contains only terminals, you go through all the rules and if any rule if in any rule the hand side is entirely in tape 2, you add the left hand side also the variable in the left hand side also to the to tape 2. So basically what I am saying here is if in any rule, the rule is of this form A gives U1 U2 etcetera. We will just move this slightly, A gives U1, U2 etcetera.

$E \rightarrow AB$

$A \rightarrow U1U2...$

Now, if all of U1 U2 etcetera could be either symbols either terminals or variables. Now if all of the right hand side are in the tape 2 you add A to the tape 2, otherwise you go through all the rules if one round you go through all the rules and you add no rule you are not able to add any rule to tape 2 or you are not able to add any variable to tape 2 then you stop, if at least one variable were able to add to tape 2 you do another round and in that round if you are able to add something you again do another round and you repeat till you are able to go to go through the entire round without adding any variable to tape 2.



And when you say which means at that point you cannot add any more variables so because you added you did not add anything even if you do one more round it is not going to make a change.

So, the point is tape 2 contains those variables. So, it contains terminals, it also contains those variables like A B and E, from which we can generate a string of terminals. So, if you start with A, A is able to generate a string of terminals, if you start with B, B is able to generate 00A, and where A can again generate a string of terminals. So, A B E can generate a string of terminals. So, the variables in tape 2 are those from which we can generate a string of terminals.

Now, you are done with the process, all you have to check is whether the start variable is able to generate a string of entirely terminals? So, what is the language? Language is simply the set of all strings that is  $L(G)$  for a grammar is nothing but any string that is entirely composed by terminals, which is derived from the start variable.

So, now, we are asking whether the start variable can generate a string of terminals, but then in tape 2, we are now maintaining a set of variables that can generate a string of only terminals. So, all you have to do is check if the start variable is there in the tape 2. If the start variable is there in tape 2, you know that the language is the start variable is able to generate a string of only terminals, which means the language generated by the grammar is not empty it at least has one string which means you reject, if the start variable is there in tape 2 you reject, if the start variable is not there in tape 2 you accept.

So, again, we instead of addressing it directly in the naive way by trying out one string after another, we are we constructed a clever algorithm where you kept track of those variables from which we can generate a string of only terminals and then at the end, you checked whether the start variable is in that list, if start variable is in that list, the language generated is not empty and then you reject if start variable is not in that list then you accept. So, this is an algorithm for  $E_{CFG}$  whether the grammar generates the empty language or not.

(Refer Slide Time: 17:38)



- Check if that variable is in tape 2.  
- If it is not in tape 2, accept.  
- Else reject.  $L(G) = \{w \in \Sigma^* \mid S \Rightarrow w\}$

Theorem 4.9: Every CFL is decidable.  
Read the proof. (Uses ACFG)

$EQ_{CFG} = \{ \langle G, H \rangle \mid G, H \text{ are CFG's, } L(G) = L(H) \}$

Can we use the same trick as in  $EQ_{DFA}$ ? NO.

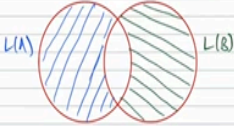
CFL's are not closed under intersection and complement. In fact,  $EQ_{CFG}$  is undecidable.



Proof: Given  $A, B$ , we can use closure properties to construct a DFA  $C$  such that



$$L(C) = \overline{L(A) \cap L(B)} \cup \overline{L(A) \cap \overline{L(B)}}$$

Symmetric difference of  $L(A)$  &  $L(B)$


$$L(A) = L(B) \iff L(C) = \emptyset.$$

On input  $A, B$ , which are DFA's.

1. Construct DFA  $C$  using closure properties.
2. Run the checker for  $C$  on  $A$ .



- If it is not in tape 2, accept.  
 - Else reject.  $L(G) = \{w \in \Sigma^* \mid \exists s \# w\}$

Theorem 4.9: Every CFL is decidable.  
 Read the proof. (Uses  $A_{CFG}$ )

$EQ_{CFG} = \{ \langle G, H \rangle \mid G, H \text{ are CFG's, } L(G) = L(H) \}$

Can we use the same trick as in  $EQ_{DFA}$ ? NO.

CFL's are not closed under intersection and complement. In fact,  $EQ_{CFG}$  is undecidable.

The next thing I want to talk about is theorem 4.9. I will just state the theorem, you can read the proof in the book, it is very similar to  $A_{CFG}$  or rather it uses  $A_{CFG}$ , it just says every context free grammar is decidable sorry!! Not context free grammar, what I meant is every context free language is decidable, every CFL is decidable, given a context free language and given a string but then you can check whether the string is in the context free language or not.

So we will close with one or two small points. So, in the case of DFAs or regular languages, we saw this, the fact that  $EQ_{DFA}$  was decidable; it is a question whether two given DFAs are equivalent whether they recognize the same language? Now, we can ask the same question for context free languages or context free grammars. Given two grammars  $G$  and  $H$ , do they generate the same language?

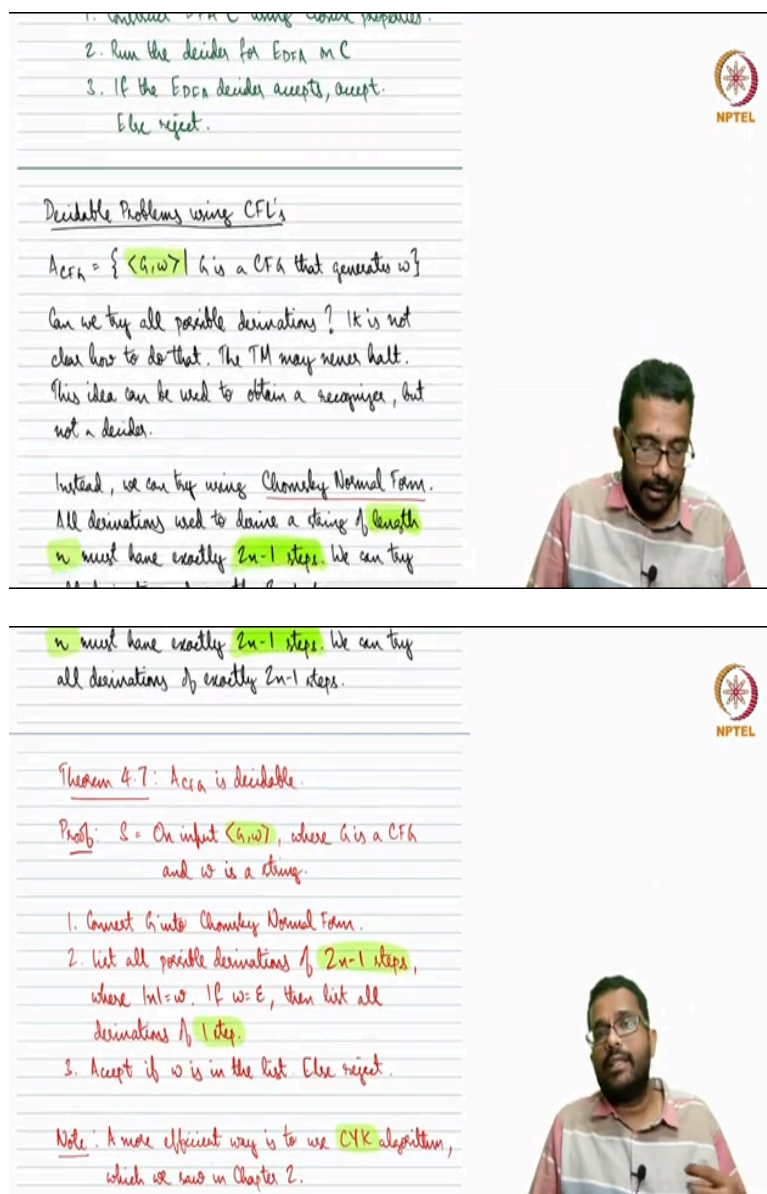
So, given two grammars  $G$  and  $H$ , do they generate the same language? Now, since we saw  $EQ_{DFA}$ , maybe the first thing that comes to mind is can we use the same thing? Can we construct a  $A_{CFG}$  which generates exactly the symmetric difference of  $L(A)$  and  $L(B)$ ? The answer is unfortunately no because the regular languages were closed under complement, intersection and union. So here, notice that we are using complement here  $L(A)$  complement, intersection for  $\overline{L(A)}$ , the intersection  $L(B)$  and over here another intersection so we are using intersection and complement and also union.

However, context-free languages are not closed under complement; they are not closed under intersection. So, there is no way to build a context free grammar in general that will generate the intersection of two context free grammars or that will generate the language which is the intersection of the two languages entered by two different context-free grammars, so, we

cannot use the same approach. So,  $EQ_{CFG}$  at least we cannot use the same approach, we will have to think of some other approach because the context free languages are not closed under intersection and complement.

And unfortunately, it turns out that this is one language which is not decidable, in fact no approach works. So, I am not telling you why I am not telling you the proof, but I am just telling you now that it is undecidable. So right now for, for now, you have to take it as a fact, maybe one of the later lectures maybe, maybe next week or the next-next week, we will see why it is undecidable.

(Refer Slide Time: 20:47)



1. Construct PDA C using above properties.  
2. Run the decider for EDA on C.  
3. If the EDA decider accepts, accept.  
Else reject.

Decidable Problems using CFL's

$A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates } w \}$

Can we try all possible derivations? It is not clear how to do that. The TM may never halt. This idea can be used to obtain a recognizer, but not a decider.

Instead, we can try using Chomsky Normal Form. All derivations used to derive a string of length  $n$  must have exactly  $2n-1$  steps. We can try

$n$  must have exactly  $2n-1$  steps. We can try all derivations of exactly  $2n-1$  steps.

Theorem 4.7:  $A_{CFG}$  is decidable.

Proof:  $S =$  On input  $\langle G, w \rangle$ , where  $G$  is a CFG and  $w$  is a string.

1. Convert  $G$  into Chomsky Normal Form.
2. List all possible derivations of  $2n-1$  steps, where  $|w|=n$ . If  $w = \epsilon$ , then list all derivations of 1 step.
3. Accept if  $w$  is in the list. Else reject.

Note: A more efficient way is to use CYK algorithm, which we saw in Chapter 2.

We cannot keep checking for each  $w$  if  $w \in L(G)$ .  
 This will not be a decider. Instead, we use a strategy similar to the one used for DFA.

Tape 1 | Rules  $G$

Tape 1:  $\langle G \rangle$       Tape 2:  $a + 01ABE$



Tape 2: Initially list all the terminal symbols.

- Repeat until no new variable is added.

$A \rightarrow aBa$   
 $B \rightarrow 00A$   
 $C \rightarrow DA$   
 $E \rightarrow AB$

$A$  is added to  
 tape 2 if  $A$   
 can generate  
 a string of  
 terminals

For all  $A$ , variable in  $G$ :  
 If there is a rule  $A \rightarrow U_1 U_2 \dots U_k$   
 where all of  $U_1, U_2, \dots, U_k$  are in  
 tape 2, then add  $A$  to tape 2.

And that completes the topics that I have in decidable problems arising out of context free languages. So, we saw  $A_{CFG}$  where we converted the grammar into Chomsky normal form and checked whether the given string is generated by the grammar. So, the Chomsky normal form gave us a bound on how many steps to check, then we saw  $E_{CFG}$  whether the given grammar generates the empty language.

So, here we maintained a set of variables which are able to generate a string of only terminals, not in one step, but maybe multiple steps. And you and then we check if the start variable is in that list, and then we use that to decide whether the grammar generates empty language or not. And we said that  $EQ_{CFG}$  cannot be we cannot use the same trick as  $EQ_{DFA}$ , and in fact, it is undecidable.

And that is all I have in the, that is all I have in the regular sorry! Decidable languages decidable problems arising out of context free languages. And now, in the next lecture, we will move to some theory that we need to have in order to show languages are undecidable. So, the next step is trying to understand undecidable things. So, before that, we need to understand certain other mathematical theories, so that we will see in the next lecture. Thank you.