UNIT-3

Context Free Grammars

The definition of context free grammars (CFGs) allows us to develop a wide variety of grammars. Most of the time, some of the productions of CFGs are not useful and are redundant. This happens because the definition of CFGs does not restrict us from making these redundant productions.

By simplifying CFGs we remove all these redundant productions from a grammar , while keeping the transformed grammar equivalent to the original grammar. Two grammars are called equivalent if they produce the same language. Simplifying CFGs is necessary to later convert them into Normal forms.

Types of redundant productions and the procedure of removing them are mentioned below.

1. Useless productions – The productions that can never take part in derivation of any string , are called useless productions. Similarly , a variable that can never take part in derivation of any string is called a useless variable. For eg.

S -> abS | abA | abB

A -> cd

B -> aB

C -> dc

In the example above , production 'C -> dc' is useless because the variable 'C' will never occur in derivation of any string. The other productions are written in such a way that variable 'C' can never reached from the starting variable 'S'.

Production 'B ->aB' is also useless because there is no way it will ever terminate . If it never terminates , then it can never produce a string. Hence the production can never take part in any derivation.

To remove useless productions , we first find all the variables which will never lead to a terminal string such as variable 'B'. We then remove all the productions in which variable 'B' occurs.

So the modified grammar becomes –

S -> abS | abA

A -> cd

C -> dc

We then try to identify all the variables that can never be reached from the starting variable such as variable ‘C’. We then remove all the productions in which variable ‘C’ occurs.
The grammar below is now free of useless productions –

S -> abS | abA

A -> cd

2. λ productions – The productions of type ‘A -> λ’ are called λ productions (also called lambda productions and null productions) . These productions can only be removed from those grammars that do not generate λ (an empty string). It is possible for a grammar to contain null productions and yet not produce an empty string.

To remove null productions , we first have to find all the nullable variables. A variable ‘A’ is called nullable if λ can be derived from ‘A’. For all the productions of type ‘A -> λ’ , ‘A’ is a nullable variable. For all the productions of type ‘B -> A1A2…An ‘ , where all ’Ai’s are nullable variables , ‘B’ is also a nullable variable.

After finding all the nullable variables, we can now start to construct the null production free grammar. For all the productions in the original grammar , we add the original production as well as all the combinations of the production that can be formed by replacing the nullable variables in the production by λ. If all the variables on the RHS of the production are nullable , then we do not add ‘A -> λ’ to the new grammar. An example will make the point clear.

Consider the grammar –

S -> ABCd (1)

A -> BC (2)

B -> bB | λ (3)

C -> cC | λ (4)

Lets first find all the nullable variables. Variables ‘B’ and ‘C’ are clearly nullable because they contain ‘λ’ on the RHS of their production. Variable ‘A’ is also nullable because in (2) , both variables on the RHS are also nullable. Similarly , variable ‘S’ is also nullable. So variables ‘S’ , ‘A’ , ‘B’ and ‘C’ are nullable variables.

Lets create the new grammar. We start with the first production. Add the first production as it is. Then we create all the possible combinations that can be formed by replacing the nullable variables with λ. Therefore line (1) now becomes ‘S -> ABCd | ABd | ACd | BCd | Ad | Bd |Cd | d ’.We apply the same rule to line (2) but we do not add ‘A -> λ’ even though it is a possible combination. We remove all the productions of type ‘V -> λ’. The new grammar now becomes –

S -> ABCd | ABd | ACd | BCd | Ad | Bd |Cd | d

A -> BC | B | C

B -> bB | b

C -> cC | c

3. Unit productions – The productions of type ‘A -> B’ are called unit productions.
To create a unit production free grammar ‘Guf’ from the original grammar ‘G’ , we follow the procedure mentioned below.

First add all the non-unit productions of ‘G’ in ‘Guf’. Then for each variable ‘A’ in grammar ‘G’ , find all the variables ‘B’ such that ‘A *=> B’. Now , for all variables like ‘A ’ and ‘B’, add ‘A -> x1 | x2 | …xn’ to ‘Guf’ where ‘B -> x1 | x2 | …xn ‘ is in ‘Guf’ . None of the x1 , x2 … xn are single variables because we only added non-unit productions in ‘Guf’. Hence the resultant grammar is unit production free. For eg.

S -> Aa | B

A -> b | B

B -> A | a

Lets add all the non-unit productions of ‘G’ in ‘Guf’. ‘Guf’ now becomes –

S -> Aa

A -> b

B -> a

Now we find all the variables that satisfy ‘X *=> Z’. These are ‘S*=>B’, ‘A *=> B’ and ‘B *=> A’. For ‘A *=> B’ , we add ‘A -> a’ because ‘B ->a’ exists in ‘Guf’. ‘Guf’ now becomes

S -> Aa

A -> b | a

B -> a

For ‘B *=> A’ , we add ‘B -> b’ because ‘A -> b’ exists in ‘Guf’. The new grammar now becomes

S -> Aa

A -> b | a

B -> a | b

We follow the same step for ‘S*=>B’ and finally get the following grammar –

S -> Aa | b | a

A -> b | a

B -> a | b

Now remove B -> a|b , since it doesnt occur in the production ‘S’, then the following grammar becomes,

S->Aa|b|a

A->b|a

Note: To remove all kinds of productions mentioned above, first remove the null productions, then the unit productions and finally , remove the useless productions. Following this order is very important to get the correct result.

This article is contributed by Nitish Joshi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Ambiguity in Grammars and Languages

Suppose we have a context free grammar G with production rules:

S->aSb|bSa|SS|Ɛ

Left most derivation (LMD) and Derivation Tree: Leftmost derivation of a string from starting symbol S is done by replacing leftmost non-terminal symbol by RHS of corresponding production rule. For example: The leftmost derivation of string abab from grammar G above is done as:

S => a**S**b => ab**S**ab => abab

The symbols underlined are replaced using production rules.

Derivation tree: It tells how string is derived using production rules from S and has been shown in Figure 1.

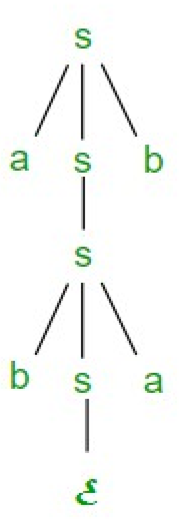
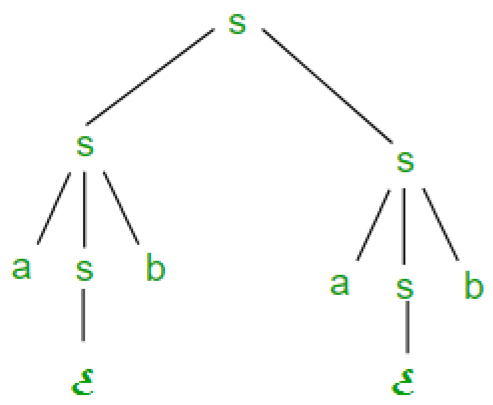


Figure 1

Right most derivation (RMD): Rightmost derivation of a string from starting symbol S is done by replacing rightmost non-terminal symbol by RHS of corresponding production rule. For Example: The rightmost derivation of string abab from grammar G above is done as:
S => S**S** => Sa**S**b => **S**ab => a**S**bab => abab

The symbols underlined are replaced using production rules. The derivation tree for abab using rightmost derivation has been shown in Figure 2.



A derivation can be either LMD or RMD or both or none. For Example:
S => aSb => abSab => abab is LMD as well as RMD

but S => SS => SaSb => Sab => aSbab => abab is RMD but not LMD.

Ambiguous Context Free Grammar: A context free grammar is called ambiguous if there exists more than one LMD or more than one RMD for a string which is generated by grammar. There will also be more than one derivation tree for a string in ambiguous grammar. The grammar described above is ambiguous because there are two derivation trees (Figure 1 and Figure 2). There can be more than one RMD for string abab which are:

S => SS => SaSb => Sab => aSbab => abab

S => aSb => abSab => abab

Ambiguous Context Free Languages: A context free language is called ambiguous if there is no unambiguous grammar to define that language and it is also called inherently ambiguous Context Free Languages.

Note:
If a context free grammar G is ambiguous, language generated by grammar L(G) may or may not be ambiguous

It is not always possible to convert ambiguous CFG to unambiguous CFG. Only some ambiguous CFG can be converted to unambiguous CFG.

There is no algorithm to convert ambiguous CFG to unambiguous CFG.

There always exist a unambiguous CFG corresponding to unambiguous CFL.

Deterministic CFL are always unambiguous.

Normal Form for Context Free Grammar

A context free grammar (CFG) is in Chomsky Normal Form (CNF) if all production rules satisfy one of the following conditions:

A non-terminal generating a terminal (e.g.; X->x)

A non-terminal generating two non-terminals (e.g.; X->YZ)

Start symbol generating ε. (e.g.; S-> ε)

Consider the following grammars,

G1 = {S->a, S->AZ, A->a, Z->z}

G2 = {S->a, S->aZ, Z->a}

The grammar G1 is in CNF as production rules satisfy the rules specified for CNF. However, the grammar G2 is not in CNF as the production rule S->aZ contains terminal followed by non-terminal which does not satisfy the rules specified for CNF.

Note –

For a given grammar, there can be more than one CNF.

CNF produces the same language as generated by CFG.

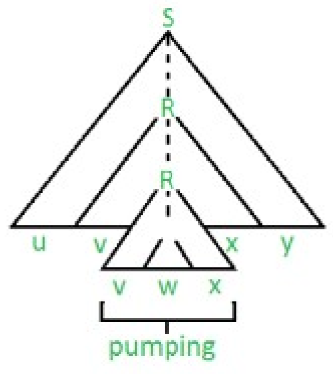
CNF is used as a preprocessing step for many algorithms for CFG like CYK(membership algo), bottom-up parsers etc.

For generating string w of length ‘n’ requires ‘2n-1’ production or steps in CNF.

Any Context free Grammar that do not have ε in it’s language has an equivalent CNF.

Pumping Lemma for Context-free Languages (CFL)
Pumping Lemma for CFL states that for any Context Free Language L, it is possible to find two substrings that can be ‘pumped’ any number of times and still be in the same language. For any language L, we break its strings into five parts and pump second and fourth substring.
Pumping Lemma, here also, is used as a tool to prove that a language is not CFL. Because, if any one string does not satisfy its conditions, then the language is not CFL.
Thus, if L is a CFL, there exists an integer n, such that for all x ∈ L with |x| ≥ n, there exists u, v, w, x, y ∈ Σ∗, such that x = uvwxy, and

(1) |vwx| ≤ n
(2) |vx| ≥ 1
(3) for all i ≥ 0: uviwxiy ∈ L



For above example, 0n1n is CFL, as any string can be the result of pumping at two places, one for 0 and other for 1.
Let us prove, L012 = {0n1n2n | n ≥ 0} is not Context-free.
Let us assume that L is Context-free, then by Pumping Lemma, the above given rules follow.
Now, let x ∈ L and |x| ≥ n. So, by Pumping Lemma, there exists u, v, w, x, y such that (1) – (3) hold.
We show that for all u, v, w, x, y (1) – (3) do not hold.

If (1) and (2) hold then x = 0n1n2n = uvwxy with |vwx| ≤ n and |vx| ≥ 1.
(1) tells us that vwx does not contain both 0 and 2. Thus, either vwx has no 0's, or vwx has no 2's. Thus, we have two cases to consider.
Suppose vwx has no 0's. By (2), vx contains a 1 or a 2. Thus uwy has 'n' 0's and uwy either has less than 'n' 1's or has less than 'n' 2's.
But (3) tells us that uwy = uv0wx0y ∈ L.
So, uwy has an equal number of 0's, 1's and 2's gives us a contradiction.
The case where vwx has no 2's is similar and also gives us a contradiction.
Thus L is not context-free.