4CS4-6: Theory of Computation (Normal Forms of Context-Free Grammars)

Lecture 15: March, 12, 2019

Prof. K.R. Chowdhary *: Professor of CS*

15.1 Introduction

This lecture note will show some normal formats of context-free grammars that are capable to generate all the context-free languages. Accordingly, we will be interested in transformation of any grammar into these normal forms. Definition of CFG does not impose any restrictions on the right hand side of the productions. However, there are ways to impose restrictions on the productions without affecting the generative power of the grammar. If G is a CFG, its corresponding context-free language $L(G)$ can be generated with following restrictions on G:

1. Each variable and each terminal symbol of G appears in the derivation of some word of $L(G)$.
2. There are no productions of the form $A \to B$, called unit production, where A and B are variables.
3. If $\varepsilon \notin L(G)$, then there is no need of productions of the form $A \to \varepsilon$, where A is a variable symbol.

In addition, there are some *normal forms*, if applied on the grammars, generate the same language as the original grammar.

Definition 15.1 Chomsky Normal Form. *A context-free grammar $G = (V, \Sigma, S, P)$ is in Chomsky normal form (CNF) if each production is one of these forms:*

$$A \to BC \mid a, \tag{15.1}$$

where $A, B, C \in V$, and $a \in \Sigma$.

Definition 15.2 Greibach Normal Form. *For a CFG $G = (V, \Sigma, S, P)$ with $\varepsilon \notin L(G)$ is Greibach normal form (GNF) if all the production rules are of the form,*

$$A \to a\alpha. \tag{15.2}$$

where $\alpha \in V^*$ and $a \in \Sigma$.

If a CFG is in normal form then it cannot have infinite left-going structure. In addition, using normal-form we can give particularly elegant proofs of the connection between context-free languages and, pushdown-store machines.

15.2 Transformation of Grammars

15.2.1 ε-Productions

The productions of the form $A \to \varepsilon$ are called ε-productions (null-productions). If $L(G)$ contains the ε-strings, certainly all the productions of the form $A \to \varepsilon$ cannot be eliminated from a CFG G. However, if there are no null-strings in $L(G)$, it is possible to eliminate all the $\varepsilon-$ productions from the grammar. However, if $\varepsilon \in L(G)$, then we could eliminate all the null productions (null rules) and get a grammar for $L(G) - \varepsilon$.

The method for removing the null productions consists finding a variables A for which $A \Rightarrow^* \varepsilon$ holds. In that case, the variable A is called *nullable*. In such cases, each production of the form $B \to X_1X_2 \dots X_n$ can be replaced by production formed by eliminating some subsets of X_i that are nullable. But if there is a production like $B \to \varepsilon$, it is not considered nullable even if all X_i in $B \to X_1X_2 \dots X_n$ are nullable.

Let L be a context-free language and $G = (V, \Sigma, S, P)$ be the corresponding context-free grammar with $\{\varepsilon\} \subseteq L(G)$. It is possible to obtain $G' = (V', \Sigma, S', P')$ using G, with no ε-productions such that G' will generate the same language L such that $L(G') = L(G) - \{\varepsilon\}$.

Definition 15.3 *Let $G = (V, \Sigma, S, P)$ be a context-free grammar, and let $A, B \in V$. Then B is said to be reducible from A if $A \Rightarrow^* B$ for $\alpha, \beta \in V^*$.*

Example 15.4 *Suggest an algorithm to find which symbols are nullable (derives ε)?*

Initially, mark all variables A with production $A \to \varepsilon$. If there is a production $A \to B_1B_2 \dots B_m$, where all B_i's are nullable, then mark A as nullable. This can be implemented in linear time (for each production of the form $A \to B_1B_2 \dots B_m$, assign an integer denoting the number of nullable variable on the right hand side, which is zero initially; for each variable A, store a list of pointers to all productions whose right hand side containing A, and when A becomes nullable, update the corresponding counters).

15.2.2 Useless Symbols in Productions

Certain symbols from the grammar can be eliminated without affecting the generating capability of the grammar. At the same time it simplifies the grammar. Let $G = (V, \Sigma, S, P)$ be the CFG. A symbol $X \in V$ is called a useful symbol if there is a derivation like

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w,$$

where $\alpha, \beta \in (V \cup \Sigma)^*$. The X is useful because it appears at least in one derivation from S to a word $w \in L(G)$. If this is not the case, then X is *useless symbol*. The justification for this is that a terminal string must be derivable through X.

Another condition for X to be useful is that it should derive a string $w \in \Sigma^*$, i.e., $X \Rightarrow^* w$. These two conditions are necessary, but they are not sufficient. However, the usefulness of a symbol has to be tested in two steps above.

In the productions given below variable B is not reachable through derivations from S, hence it is useless. The production $B \to bB$ is called *useless production.*

$$S \to aSb \mid \varepsilon,$$
$$B \to bB.$$

When productions that do not take part in the derivation process are removed, it results to a simplified grammar.

Consider another set of productions P for a grammar as given below.

$$S \to aSb \mid A \mid \varepsilon,$$
$$A \to aA.$$

Here, the production $S \to A$ has no role because A cannot be eliminated in the sentential form when generating terminal string. Hence, productions $S \to A$, $A \to aA$ both can be removed. The simplified grammar for above is therefore,

$$S \to aSb \mid \varepsilon.$$

Example 15.5 *Consider the following productions in a grammar G,*

$$S \to A$$
$$A \to aA \mid \varepsilon$$
$$B \to bA$$

where S is starts symbol. We note that variable B does not appear in the sentential form during any derivation string. Though a derivation from B leads to primary string but this is of no use when B does not appear in any derivation string $w \in L(G)$, where $S \Rightarrow^* w$. Therefore, variable B and production $B \to bA$ can be eliminated without affecting the generating power of grammar G.

Example 15.6 *Remove the useless symbols and productions from context-free grammar $G = (V, \Sigma, S, P)$, where*

$V = \{S, A, B, C\}$

$\Sigma = \{a, b\}$

S is start symbol, and

$P = \{S \to aS \mid A \mid B, A \to a, C \to bb, B \to aBa\}$.

Let us first identify the productions that can lead to terminal string. These are,

$$A \to a$$
$$C \to bb$$
$$S \to A.$$

Thus, variables that can lead to terminal strings are A, C, S. We note that variable B does not lead to terminal string. Hence it can be removed. Let $G' = (V', \Sigma, S, P')$ be a new grammar after variable B and its corresponding productions have been removed. Similarly, the variable C cannot be reached from the start symbol, hence C and its productions can be removed. The tuples of G can be defined as follows.

$$V' = \{S, A, C\}$$
$$\Sigma = \{a, b\}$$
$$P' = \{S \to aS \mid A, A \to a\}.$$

Consequently, the simplified grammar is given as

$$G' = (\{S, A\}, \{a\}, S, P').$$

Example 15.7 *Let $G = (V, \Sigma, S, P)$ be a CFG where,*

$$V = \{A, B\}$$
$$\Sigma = \{a, b, c\}$$
$$S = \{A\}$$
$$P = \{A \to a \mid aaA \mid abBc, B \to abba \mid b\}.$$

Simplify the grammar G.

Let us assume that using the substitution process given in the theorem **??** above, we get an equivalent grammar

$$G = (V', \Sigma, S, P')$$

where $V' = \{A\}$, and P' is defined comprising of following productions:

$$A \to a \mid aaA \mid ababbac \mid abbc$$

It can be easily verified that every string that is generated by G can also be generated by G'. Hence it proves that $L(G) = L(G')$.

Theorem 15.8 *Given a CFG $G = (V, \Sigma, S, P)$ with $L(G) \neq \phi$, there exists an equivalent CFG $G' = (V', \Sigma, S, P')$ such that for every variable $A \in V'$ there is derivation $S \Rightarrow^* \alpha A \beta \Rightarrow^* w$, for $w \in \Sigma^*$ where $\alpha, \beta \in (\Sigma \cup V')^*$. In conclusion, the equivalent CFG G' does not contain any useless symbols or productions.*

Proof: As we discussion above, a useless symbol is one which does not appear in any derivation $S \Rightarrow^* w$ of, $w \in L(G)$. Hence, useless symbol is one, which is not reachable.

Assume that initially, V' and P' are null. First we consider variables $A_i \in V$ such that $A_i \to w \in P$. For this $A \Rightarrow^* w$ holds, hence we move A_i to V' and $A_i \to w$ to P'.

Next consider $A_j \to X_1 X_2 \dots X_n \in P$, where each X_i is either a terminal or variable. In case of variable $X_i = A_i$, it is already placed in V'. Hence, a terminal string can be derived with a derivation beginning with $A_j \Rightarrow X_1 X_2 \dots X_n$. The above situation is illustrated in productions tree (parse tree) shown in Fig. 15.1.

Consider the level k in this tree. Since $A_i \to ab$, therefore this production is moved into P' and variable A_i is made part of V' by,

$$P' = P' \cup A_i \to ab$$
$$V' = V' \cup \{A_i\}.$$

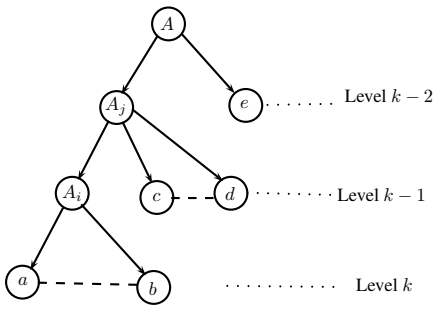

Figure 15.1: Derivation-tree

Similar process is repeated for all the variables at this level. Next we move to a lower level in the tree, say, at $k-1$. The symbols A_i are already in V' (see Fig. 15.1). Now, same process is repeated at $k-1$ level to move the terminal symbols (A_j) into V' and productions ($A_j \to A_i cd$) into P'. And so on, it is repeated up to level 1 in the parse-tree.

This results to generation of CFG G'. It should be noted that variables and productions that do not participate in derivation when started from the root S, gets automatically eliminated.

An algorithm, which iteratively computes V' and P' is given as algorithm 1. The algorithm cannot terminate while there remains variables A_j in the V having production like $A_j \to X_1 X_2 \dots X_n \in P$ where $X_i \in (\Sigma \cup V')$.

In the next phase, we construct a variable dependency graph for G' to find out all the variables and terminals that cannot be reached from S. Next, delete all such non-reachable variables and terminals, and obtain $G' = (V', \Sigma, S, P')$. Therefore, $V' \subseteq V$, $P' \subseteq P$ and $G' \subseteq G$. However, G' will generate the same strings that could be generated using G, because the useless symbols and productions were not contributing in generation of strings earlier also. Therefore, $L(G') = L(G)$, and G' and $L(G)$ are simplified versions of G and $L(G)$, respectively. ■

Algorithm 1 Construction of an equivalent simplified grammar.

1: $V' = \{A_i \mid A_i \to w, w \in \Sigma^*\}$
2: $V = V - V'$
3: $P' = \{A \to w\}$
4: $P = P - P'$
5: **while** there is $A_j \to X_1X_2 \dots X_n \in P$ for $X_{(i=1,n)} \in \Sigma \cup V')$ **do**
6: $\quad V' = V' \cup A_j$
7: $\quad P' = P' \cup \{A_j \to X_1X_2 \dots X_n\}$
8: $\quad V = V - \{A_j\}$
9: $\quad P = P - \{A_j \to X_1X_2 \dots X_n\}$
10: **end while**

15.2.3 Removing unit Productions

In simplification of context-free grammars another important step is to make the given context-free grammar as *unit-rule free*. This is essential for the applications in compilers. As the compiler spends time on each rule used in parsing by generating semantic routines, having unnecessary unit-rules will increase compiler's time.

Lemma 15.9 *Let G be a CFG, then there exists a CFG G' without unit rules such that $L(G) = L(G')$.*

Elimination of unit productions is somewhat similar to elimination of ε-productions. When unit productions are eliminated we must be sure that same strings can be generated by the grammar as earlier. Sometimes, there is requirement to add a production when a unit production is removed. Consider the following productions.

$$A \to B$$
$$B \to bB \mid c$$

The unit of production $A \to B$ can be deleted and we add production:

$$A \to bB \mid c$$

For this we need a systematic view to find all the pair of variable (A, B) such that $A \Rightarrow^* B$. If there are no null productions in G, such pairs can be easily located by a sequence of unit productions. But, if these are null productions, then finding unit productions is slightly difficult, because $A \Rightarrow^* B$ may be derived through $A \Rightarrow BC \Rightarrow B$, where $C \to \varepsilon$. Therefore, before removing the unit productions, it is necessary that all the ε-productions be removed. Thus, first G' is obtained from G so that G' is free from ε-productions, and $L(G) = L(G')$.

If G (or G') has no ε-productions, we can find for each variable A, all variables B such that there is a derivation $A \Rightarrow^* B$. This can be achieved by drawing dependency graph with an edge (C, D) whenever the grammar has a unit production $C \to D$. Hence, $A \Rightarrow^* B$ holds whenever there is walk-through in the graph from A to B.

Consider that $G = (V, \Sigma, S, P)$ be a grammar with some unit productions but no ε-productions, and $G' = (V', \Sigma, S, P')$ be an equivalent grammar without unit productions. The G' can be constructed as follows.

1. First add into P' all non-unit productions of P,
2. Next, for all A and B satisfying $A \Rightarrow^* B$, we add into P' the production $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$,

where, $B \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$ is already in P'. Since this is already in P', none of the α_i are single variables because only the non-unit productions were added into P' in advance. Therefore, no unit productions are created in P'. The same process can be used for adding the remaining productions from P to P'. The context-free languages $L(G)$ and $L(G')$ are equal as per the following theorem.