4CS4-6: Theory of Computation (Chomsky NF, and Grebach Normal Forms)

Lecture 16: March, 23 & 25, 2020

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

16.1 Normal Forms of CFG

The CFGs have no restrictions on the right hand side part of every production $A \to \alpha$, where $\alpha \in (V \cup \Sigma)^*$. The normal forms on these grammars state that all the context-free grammars are equivalent to grammars with some restrictions on the right side of each production. Many proofs can be simplified if the right-hand (α) are bounded to be of length two; then no grammar tree ever has more than binary branching.

Definition 16.1 binary standard form. *Let $G = (V, \Sigma, S, P)$ be a context-free grammar, the G is said to be binary standard form if each production rule is of the form,*

$$A \to BC \mid a$$

where $A \in V$; $B, C \in V - \{S\}$.

Theorem 16.2 *For each context-free grammar $G = (V, \Sigma, S, P)$ without null productions, useless productions, and unit productions, there is a grammar $G' = (V', \Sigma, S, P')$ such that $L(G) = L(G')$, where G' is in Chomsky Normal Form (CNF).*

Proof: Let G be a CFG, which generates strings of context-free language L, and $\varepsilon \notin L$. Also, we assume that G does not have unit productions and useless productions. In case this does not hold, one can always convert a grammar with ε-productions, useless productions, and unit productions, into an equivalent grammar that does not have these. For proofs, see the theorems discussed above.

Now consider the productions P. If a production has single symbol on the right hand side, this will be a terminal symbol because unit productions do not exist in G. Hence this production is already a CNF. Next consider a production in P of the form $A \to X_1X_2 \dots X_n$, where $n \geq 2$ and $X_i \in (V \cup \Sigma)$. If X_i is a terminal symbol, say a, then substitute X_i by C_i and introduce a new production $C_i \to a$. After making these changes for all the productions, let the new set of variables be V' and new set of productions be P'. Let us call this CFG as $G' = (V', \Sigma, S, P')$. However, G' is still not CNF as it has production of the form:

$$A \to a$$
$$A \to C_1C_2 \dots C_n$$

where $A, C_i \in V'$, and $a \in \Sigma$. However, we have $L(G) = L(G')$, as due to above changes the generating power of remains same as that of G.

In second step, additional variables are introduced, and productions of the form $A \rightarrow C_1C_2 \ldots C_n$ are reduced to $A \rightarrow BC$.

The productions of the form $A \rightarrow C_1C_2$ (already a CNF) are retained unmodified in G' along with their variables. Next, for the productions of the form $A \rightarrow C_1C_2 \ldots C_n$, for $n > 2$, new variables are introduced and productions are modified in G' as follows.

$$\begin{aligned} A &\rightarrow C_1D_1 \\ D_1 &\rightarrow C_2D_2 \\ ...D_{n-3} &\rightarrow C_{n-2}D_{n-2} \\ D_{n-2} &\rightarrow C_{n-1}C_n \end{aligned}$$

The above productions are as per the CNF, hence added into the P', and all corresponding variables are added into V'.

If above process is repeated for all the productions in P', these productions get transformed into CNF in G'. This proves that

$$L(G') = L(G).$$

Example 16.3 *Convert the CFG $G = (V, \Sigma, S, P)$, with productions given below, into Chomsky normal form.*

$S \rightarrow aA \mid bB$

$A \rightarrow aaA \mid b$

$B \rightarrow bbB \mid a$

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

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

S is start symbol.

First considering productions of the form $A \rightarrow a$, these are two only. Hence,

$$P' = \{A \rightarrow b, B \rightarrow a\}.$$

Next consider the productions of the form $A \rightarrow X_1X_2 \ldots X_n$, where $X_i \in (\Sigma \cup V)$ and $n \geq 2$. These are,

$S \rightarrow aA$

$S \rightarrow bB$

$A \rightarrow aaA$

$B \rightarrow bbB$

Convert these to the form $A \to C_1C_2 \dots C_n$. Hence, the above are transformed as,

$S \to C_1A$, $C_1 \to a$

$S \to C_2B$, $C_2 \to b$

$A \to C_3C_4A$, $C_3 \to a$, $C_4 \to a$

$B \to C_5C_6B$, $C_5 \to b$, $C_6 \to b$

Add these productions to P'.

Now, pickup the productions of the form $A \to C_1C_2 \dots C_n$ in and convert them to the form $A \to C_1D_1$, $D_1 \to C_2D_2$, etc. These productions in P' are:

$A \to C_3C_4A$

$B \to C_5C_6B$

Convert these productions into the form given below, and add them into P'.

$A \to C_3D_1$

$D_1 \to C_4A$

$B \to C_5D_2$

$D_2 \to C_6B$

Add rest of productions also into P'. The final P' and V' are,

$$\begin{aligned} P' = \{&A \to b, B \to a, S \to C_1A, C_1 \to a, S \to C_2B, \\ &C_2 \to b, A \to C_3D_1, D_1 \to C_4A, C_3 \to a, \\ &C_4 \to a, B \to C_5D_2, D_2 \to C_6B, C_5 \to b, C_6 \to b\}. \end{aligned}$$

$V = \{S, A, B, C_1, C_2, C_3, C_4, C_5, C_6, D_1, D_2\}$.

A CNF grammar, equal to the original given grammar is, therefore

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

16.2 Invertible Grammars

The motivation for studying this class of grammar comes from the theory of *bottom-up parsing*. Crudely speaking, bottom-up parsing consists of successively finding the phrases and reducing them to their parents. Invertible grammars allow reduction decision to be made simpler.

Definition 16.4 Invertible Grammar. *A context-free grammar $G = (V, \Sigma, S, P)$ is said to be invertible if no two production rules are the same of right hand side of the productions, i.e., $A \to \alpha$ and $B \to \alpha$ in P implies $A = B$, for $A, B \in V$, and $\alpha \in (\Sigma \cup V)^*$.*

Thus invertible grammars have unique right-hand sides of the productions and the reduction phase of the parsing becomes a matter of table lookup. The reason for this name is that P^{-1} is a function exactly when G is invertible.

16.3 Greibach Normal Forms

Several algorithms have been proposed utilizing some forms of pushdown store machines for syntactic analysis of natural or artificial languages, which place some restriction on context-free grammars. In order to work efficiently, these algorithms forbid infinite left-going structures resulting from generations like $A \Rightarrow^* A\varepsilon$. It has been hypothesized that natural languages do not permit infinite loops of this kind. One can avoid such structures in designing the syntax of programming languages. It has been proved that every context-free grammar is strongly equivalent to one that does not contain such structures. The Greibach normal form (GNF) shows that every context-free grammar is strongly equivalent to one that does not contain such structures.

In a given analysis of a given input string, the machine (PDA) scans the input strictly from left to right. At each operation it,

1. scans a new input symbol,
2. scans and erases the top symbol on the push-down store,
3. adds a (possibly null) string of symbols to the push-down store,
4. prints an output.

Example 16.5 *Give algorithms to decide the following: "Is $L(G)$ finite, for a given CFG G?*

Convert G to Greibach normal form (without useless symbols), that is, all productions are of the form $A \to a\gamma$, where $\gamma \in V^*$. Build a directed graph with vertices V (i.e., variables), and add an edge $A \to B$ if B appears in γ. $L(G)$ is finite if and only if the graph does not contain cycles.

Theorem 16.6 *For each context-free grammar $G = (V, \Sigma, S, P)$ with production P defined as,*

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m$$

where $\alpha_i \neq \varepsilon$ for all i, $1 \leq i \leq n$, be all the rules with variables A on the left such that the left-most symbol of the right hand side of the rule is A. The remaining rule is,

$$A \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

,

then there exists a CFG $G' = (V, \Sigma, S, P')$ such that $L(G) = L(G')$, where

$$V' = V \cup \{B\}$$

and, P' is given as,

$$\begin{aligned} A &\to \beta_1 B \mid \cdots \mid B_n B \mid \beta_1 \cdots \mid \beta_n \\ B &\to \alpha_1 B \mid \cdots \mid \alpha_m B \mid \alpha_1 \mid \cdots \mid \alpha_m. \end{aligned}$$

In other words, for each grammar with left recursions in the productions, there exists an equivalent grammar without left recursions.

Proof: The effect of this construction is to eliminate the left recursion in the variable A. In each place we have a new right recursive variable B. We note that none of the new A-rules are directly left recursive because none of the β_i begin with A. In addition, with $\beta_i \neq \varepsilon$ one cannot get the left recursion on A by going through B. Also, we note that B is not the left recursive variable because $\alpha_i \neq \varepsilon$ for all i and because none of the α_i begin with B.

Let us consider a derivation using G:

$$\begin{aligned} A &\Rightarrow A\alpha_{i_1} \Rightarrow A\alpha_{i_2}\alpha_{i_1} \\ &\Rightarrow \cdots \Rightarrow A\alpha_{i_p}\alpha_{i_{p-1}} \ldots \alpha_{i_1} \\ &\Rightarrow \beta_j \alpha_{i_p}\alpha_{i_{p-1}} \ldots \alpha_{i_1} \end{aligned}$$

The above derivation in G can be obtained using G' as,

$$\begin{aligned} A &\Rightarrow \beta_j B \Rightarrow \beta_j \alpha_{i_p} B \Rightarrow \beta_j \alpha_{i_{p-1}} B \\ &\Rightarrow \cdots \Rightarrow \beta_j \alpha_{i_p}\alpha_{i_{p-1}} \ldots \alpha_{i_2} B \\ &\Rightarrow \beta_j \alpha_{i_p}\alpha_{i_{p-1}} \ldots \alpha_{i_1}. \end{aligned}$$

This proves that $L(G) = L(G')$. ■

A context-free phrase-structure generator is in standard form if and only if all of its rules are of the form $Z \to aY_1Y_2 \ldots Y_m$, where $Z, Y_i \in V$ and $a \in \Sigma$. This format will help in processing one input symbols at each step. A standard form is always convenient for computer manipulation of context-free languages. This standard from is called *Greibach normal form.* The proof of this theorem (refer bibliography for details) states that every context-free phrase-structure generator is strongly equivalent to one in GNF.

In *recursive descent* parsers, presence of left recursion causes the device to go into an infinite loop. Thus elimination of left recursion is of practical importance in such parsers.

A CFG $G = (V, \Sigma, S, P)$ is in Greibach normal form if and only if all of the rules of P are of the forms:

$$\begin{aligned}
&A \to a \\
&A \to aB_1B_2 \dots B_n, \text{ for } n \geq 1, a \in \Sigma, \text{ and } B_i \in V.
\end{aligned}$$

Example 16.7 *Convert the following CFG into GNF:*

$$\begin{aligned}
&S \to BC \\
&B \to bB \mid aC \mid a \\
&C \to a
\end{aligned}$$

The above CFG is not GNF, but applying the theorem (substitution 1) these productions can be transformed into an equivalence grammar, which is GNF.

$$\begin{aligned}
&S \to bBC \mid aCC \mid aC \\
&B \to bB \mid aC \mid a \\
&C \to a
\end{aligned}$$

which is GNF.