Context-Free Grammars: Normal Forms

Seungjin Choi

Department of Computer Science and Engineering
Pohang University of Science and Technology
77 Cheongam-ro, Nam-gu, Pohang 37673, Korea
seungjin@postech.ac.kr
Outline

- **Simplification of CFG**: Transform an arbitrary CFG into an equivalent form that satisfies certain restrictions on its form
  - Substitution rule
  - Removing useless productions
  - Removing $\epsilon$-productions
  - Removing unit-productions
- **Normal forms**
  - Chomsky normal form
  - Greibach normal form
Throughout the lecture, we only consider languages that do not contain the empty string $\epsilon$, since it plays a rather singular role in many theorems and proofs.

In doing so, we do not lose generality.

Let $G$ be a CFG for $L - \{\epsilon\}$.
Then we add a new variable $S_0$ to $V$ and add productions to $P$:

$$S_0 \rightarrow S | \epsilon.$$  

Any non-trivial conclusion that we can make for $L - \{\epsilon\}$ will almost certainly transfer to $L$. Given any CFG $G$ there is a method of obtaining $\hat{G}$ such that $L(\hat{G}) = L(G) - \{\epsilon\}$. 
Substitution Rule

Theorem
Let $G = (V, T, S, P)$ be a CFG. Suppose that $P$ contains a production of the form

$$A \rightarrow x_1Bx_2.$$ 

Assume that $A$ and $B$ are different variables and that

$$B \rightarrow y_1 \mid y_2 \mid \cdots \mid y_n$$

is the set of all productions in $P$ which have $B$ as the left side. Let $\hat{G} = (V, T, S, \hat{P})$ be the grammar in which $\hat{P}$ is constructed by deleting $A \rightarrow x_1Bx_2$ from $P$ and adding to it

$$A \rightarrow x_1y_1x_2 \mid x_1y_2x_2 \mid \cdots \mid x_1y_nx_2.$$ 

Then $L(\hat{G}) = L(G)$. 
Proof. What you need to show is that

- if $w \in L(G)$ then $w \in L(\hat{G})$
- if $w \in L(\hat{G})$ then $w \in L(G)$
Suppose that $w \in L(G)$, i.e., $S \xrightarrow{G}^* w$.

If this derivation does not involve the production $A \rightarrow x_1 B x_2$, then obviously

$$S \xrightarrow{G}^* w.$$  \[ \]

If it does, then look at the derivation the first time $A \rightarrow x_1 B x_2$ is used. The $B$ so introduced eventually has to be replaced; we lose nothing by assuming that this is done immediately. Thus,

$$S \xrightarrow{G} u_1 A u_2 \Rightarrow u_1 x_1 B x_2 u_2 \Rightarrow u_1 x_1 y x_2 u_2.$$  \[ \]

But with grammar $\hat{G}$ we can get

$$S \xrightarrow{\hat{G}} u_1 A u_2 \Rightarrow u_1 x_1 y x_2 u_2.$$  \[ \]

Thus we can reach the same sentential form with $G$ and $\hat{G}$. If $A \rightarrow x_1 B x_2$ is used again later, we can repeat the argument. If follows then, by induction on the number of times the production is applied, that

$$S \xrightarrow{\hat{G}}^* w.$$  \[ \]

Therefore, if $w \in L(G)$ then $w \in L(\hat{G})$.

By similar reasoning, we can show that if $w \in L(\hat{G})$ then $w \in L(G)$, which completes the proof.  \[ \]
**Example:** Consider $G = (\{A, B\}, \{a, b, c\}, A, P)$ with productions

\[
\begin{align*}
A & \rightarrow a | aaA | abBc, \\
B & \rightarrow abBA | b.
\end{align*}
\]

Substitutions for the variable $B$, lead to the grammar $\hat{G}$ with productions

\[
\begin{align*}
A & \rightarrow a | aaA | ababbAc | abbc, \\
B & \rightarrow abBA | b. \quad \text{(unnecessary)}
\end{align*}
\]

The string $aaabbc$ has the derivation

\[
A \Rightarrow aaA \Rightarrow aaabBc \Rightarrow aaabbc \quad \text{in } G
\]

and

\[
A \Rightarrow aaA \Rightarrow aaabbc \quad \text{in } \hat{G}.
\]
Useful?

Definition
Let $G = (V, T, S, P)$ be a CFG. A variable $A \in V$ is said to be **useful** if and only if there is at least one $w \in L(G)$ such that

$$S \Rightarrow^* xA y \Rightarrow^* w$$

with $x, y \in (V \cup T)^*$.  
In other words, a variable is useful if and only if it occurs in at least one derivation.

Two requirements to be "useful":

- Can derive a terminal string?: A symbol $X \in V \cup T$ is **generating** if $X \Rightarrow^* w$ for some $w \in T^*$.
- Can be reached from the start variable?: A symbol $X \in V \cup T$ is **reachable** if there is a derivation $S \Rightarrow^* \alpha X \beta$ for some $\alpha$ and $\beta$.  

Removing Useless Productions

- A variable that is not useful is called **useless**.
- A production is **useless** if it involves any useless variable.
- Eliminate the symbols that are not generating first, and then eliminate from the remaining grammars those symbols that are not reachable. Note that the other way around does not work!

**Example:** Eliminate useless variables and productions from $G$ where $V = \{S, A, B, C\}$ and $T = \{a, b\}$ with

\[
\begin{align*}
S & \rightarrow aS | A | C, \\
A & \rightarrow a, \\
B & \rightarrow aa, \\
C & \rightarrow aCb.
\end{align*}
\]
First, identify the set of variables that can lead to a terminal string

\[ A \rightarrow a, \text{ (OK)} \]
\[ B \rightarrow aa, \text{ (OK)} \]
\[ S \Rightarrow A \Rightarrow a, \text{ (OK)} \]
\[ C \text{ (No!)} \]

Remove \( C \), then we have \( V_1 = \{ S, A, B \} \) and \( T = \{ a \} \) with

\[ S \rightarrow aS | A, \]
\[ A \rightarrow a, \]
\[ B \rightarrow aa. \]
Next, eliminate the variables that cannot be reached from the start variable. To this end we draw a dependency graph where its vertices are labeled with variables and an edge between $C$ and $D$ is connected if and only if there is a production form $C \rightarrow xDy$.

The variable $B$ is useless. Therefore, $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ where $\hat{V} = \{S, A\}$ and $\hat{T} = \{a\}$ with

$$
S \rightarrow aS \mid A, \\
A \rightarrow a.
$$
Theorem
Let \( G = (V, T, S, P) \) be a CFG. Then there exists an equivalent grammar \( \hat{G} = (\hat{V}, \hat{T}, S, \hat{P}) \) that does not contain any useless variables or productions.

Proof. The algorithm \( \hat{G} \) can be generated from \( G \) by an algorithm consisting of two parts. In the first part we construct an intermediate grammar \( G_1 = (V_1, T_1, S, P_1) \) such that \( V_1 \) contain only variables \( A \) for which \( A \Rightarrow^* w \in T^* \) is possible.

1. Set \( V_1 \) to \( \phi \).
2. Repeat the following step until no more variables are added to \( V_1 \).

   For every \( A \in V \) for which \( P \) has a production of the form

   \[
   A \rightarrow x_1x_2 \cdots x_n \quad \text{with all } x_i \in V_1 \cup T,
   \]

   add \( A \) to \( V_1 \).
3. Take \( P_1 \) as all the productions in \( P \) whose symbols are all in \( V_1 \cup T \).
Clearly this procedure terminates. It is equally clear that if $A \in V_1$ then $A \Rightarrow^* w \in T^*$ is a possible derivation with $G_1$. The remaining issue is whether every $A$ for which $A \Rightarrow^* w$ is added to $V_1$ before the procedure terminates.

Consider a parse tree
At level $k$, there are only terminals, so every variable $A_i$ at level $k - 1$ will be added to $V_1$ on the first pass through step 2. Any variable at level $k - 2$ will then be added to $V_1$ on the second pass through step 2. The third time through step 2, all variables at level $k - 2$ will be added, and so on. The algorithm cannot terminate while there are variable in the tree that are not yet in $V_1$. Hence $A$ will be eventually added to $V_1$.

In the second part of the construction, we get the final answer $\hat{G}$ from $G_1$. We draw the variable dependency graph for $G_1$ and from it find all variables that cannot be reached from $S$. We can also eliminate any terminal that does not occur in some useful productions. The result is the grammar $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$.

Because of the construction, $\hat{G}$ does not contain any useless symbols or productions. Also, for each $w \in L(G)$ we have a derivation

$$S \Rightarrow xAy \Rightarrow w.$$
Since the construction of $\hat{G}$ retains $A$ and all associated productions, we have everything needed to make the derivation

$$S \xrightarrow{\ast} xAy \xrightarrow{\ast} w.$$ 

The grammar $\hat{G}$ is constructed from $G$ by removal of productions, implying $\hat{P} \subseteq P$, which gives

$$L(\hat{G}) \subseteq L(G).$$

Putting the two results together, we see that $G$ and $\hat{G}$ are equivalent. ■
Removing $\epsilon$-Productions

**Definition**
Any production of a CFG that is of the form

$$A \rightarrow \epsilon$$

is called $\epsilon$-production.

**Definition**
Any variable $A$ for which the derivation

$$A \Rightarrow^* \epsilon$$

is possible, is called nullable.
Example: Consider the CFG

\[ S \rightarrow aS_1b, \]
\[ S_1 \rightarrow aS_1b | \epsilon. \]

This leads to \( L = \{a^n b^n \mid n \geq 1\} \).

The \( \epsilon \)-production, \( S_1 \rightarrow \epsilon \), can be removed by the substituting \( \epsilon \) for \( S_1 \), leading to

\[ S \rightarrow aS_1b | ab, \]
\[ S_1 \rightarrow aS_1b | ab. \]

which also generate \( L = \{a^n b^n \mid n \geq 1\} \).
Theorem
Let $G$ be any CFG with $\epsilon$ not in $L(G)$. Then there exists an equivalent grammar $\hat{G}$ having no $\epsilon$-productions.

Proof. Find the set $V_N$ of all nullable variables of $G$.

1. Find all productions $A \rightarrow \epsilon$, and put $A$ into $V_N$. 
2. Repeat the following step until no further variables are added to $V_N$.

2.1 For all productions

$$B \rightarrow A_1 A_2 \cdots A_n,$$

where $A_1, A_2, \ldots, A_n \in V_N$, put $B$ into $V_N$. 

Once the set \( V_N \) has been found, we are ready to construct \( \hat{P} \). We look at all productions in \( P \) of the form

\[ A \rightarrow x_1x_2 \cdots x_m, \quad m \geq 1, \]

where \( x_i \in V \cup T \).

For each such production of \( P \), we put into \( \hat{P} \) that productions as well as all those generated by replacing nullable variables with \( \epsilon \) in all possible combinations.

For example, if \( x_i \) and \( x_j \) are both nullable, there will be one production in \( \hat{P} \) with \( x_i \) replaced with \( \epsilon \), one in which \( x_j \) is replaced with \( \epsilon \), and one in which both \( x_i \) and \( x_j \) are replaced with \( \epsilon \). There is one exception: if all \( x_i \) are nullable, the production \( A \rightarrow \epsilon \) is not put into \( \hat{P} \). If is straightforward to see \( \hat{G} \) is equivalent to \( G \). \( \blacksquare \)
Example: Consider the CFG $G$ with

\[
\begin{align*}
S & \rightarrow ABaC, \\
A & \rightarrow BC, \\
B & \rightarrow b \mid \epsilon, \\
C & \rightarrow D \mid \epsilon, \\
D & \rightarrow d.
\end{align*}
\]

Identify nullable variables. First we recognize $V_N = \{B, C\}$ since $B \rightarrow \epsilon$ and $C \rightarrow \epsilon$. In addition, since we have $A \rightarrow BC$, the set of nullable variables is $V_N = \{A, B, C\}$. 
Then we construct $\hat{G}$ (with no $\epsilon$-productions)

\[
\begin{align*}
S & \rightarrow ABaC | BaC | AaC | ABa | aC | Aa | Ba | a, \\
A & \rightarrow BC | B | C, \\
B & \rightarrow b, \\
C & \rightarrow D, \\
D & \rightarrow d.
\end{align*}
\]
Removing Unit-Productions

Definition

Any production of a CFG of the form

\[ A \rightarrow B, \]

where \( A, B \in V \) is called a unit-production.

We use the substitution rule to remove unit-productions.
Theorem

Let $G = (V, T, S, P)$ be any CFG without $\epsilon$-productions. Then there exists a CFG $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ that does not have any unit-productions and that is equivalent to $G$.

Proof. Obviously, any unit production of the form $A \rightarrow A$ can be removed from the grammar without effect. We need only consider $A \rightarrow B$ where $A$ and $B$ are different variables.

We first find, for each $A$ all variables such that

$$A \Rightarrow^* B.$$

We can do this by drawing a dependency graph with an edge $(C, D)$ whenever the grammar has a unit production $C \rightarrow D$. $A \Rightarrow^* B$ holds whenever there is a walk between $A$ and $B$. 
The new grammar $\hat{G}$ is generated by first putting into $\hat{P}$ all non-unit productions of $P$. Next, for all $A$ and $B$ satisfying $A \Rightarrow^* B$, we add to $\hat{P}$

$$A \rightarrow y_1 \mid y_2 \mid \cdots \mid y_n,$$

where $B \rightarrow y_1 \mid y_2 \mid \cdots \mid y_n$ is the set of all rules in $P$ with $B$ on the left. Note that since $B \rightarrow y_1 \mid y_2 \mid \cdots \mid y_n$ is taken from $\hat{P}$, none of the $y_i$ can be a single variable, so that non-unit productions are created by the last step.

To show that the resulting grammar is equivalent to the original one we can follow the same line of reasoning as in Theorem 1.
Example: Remove all unit-productions from

\[ S \rightarrow Aa \mid B, \]
\[ B \rightarrow A \mid bb, \]
\[ A \rightarrow a \mid bc \mid B. \]

We first draw a dependency graph for the unit-productions.
It follows from the dependency graph that we have

\[
\begin{align*}
S^* & \Rightarrow A, \\
S^* & \Rightarrow B, \\
B^* & \Rightarrow A, \\
A^* & \Rightarrow B. \\
\end{align*}
\]

The new rules are

\[
\begin{align*}
S & \rightarrow a | bc | bb, \\
A & \rightarrow bb, \\
B & \rightarrow a | bc.
\end{align*}
\]
The original non-unit productions are

\[
\begin{align*}
S & \rightarrow Aa, \\
B & \rightarrow bb, \\
A & \rightarrow a | bc.
\end{align*}
\]

Therefore, the equivalent grammar (without unit-productions) are constructed by adding the new rules to the original non-unit productions:

\[
\begin{align*}
S & \rightarrow a | bc | bb | Aa, \\
A & \rightarrow bb | a | bc, \\
B & \rightarrow a | bc | bb.
\end{align*}
\]
Theorem

Let $L$ be CFG that does not contain $\epsilon$. Then there exists a CFG that generates $L$ and that does not have any useless productions, $\epsilon$-productions, or unit-productions.
Summary

To clean up a grammar we can

1. Eliminate $\epsilon$-productions
2. Eliminate unit-productions
3. Eliminate useless productions

in this order.
Definition
A CFG is in Chomsky normal form (CNF) if all productions are of the form

\[ A \rightarrow BC, \quad \text{or} \quad A \rightarrow a, \]

where \( A, B, C \in V \) and \( a \in T \).

- The number of symbols on the right of a production are strictly limited.
- The string on the right of a production consists of no more than two symbols.
**Example:** The grammar with the following productions

\[
S \rightarrow AS \mid a, \\
A \rightarrow SA \mid b,
\]

is in **CNF**.

The grammar with the following productions

\[
S \rightarrow AS \mid AAS, \\
A \rightarrow SA \mid aa,
\]

is not in **CNF**.
Theorem
Any CFG $G$ with $\epsilon \in L(G)$ has an equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ in CNF.
Example: Consider a CFG $G$ with productions

\[
S \rightarrow ABa, \\
A \rightarrow aab, \\
B \rightarrow Ac.
\]

Convert $G$ to CNF.
Step 1: Introduce new variables, $B_a, B_b, B_c$, for terminals $a, b, c$, respectively. Then we have

\[
\begin{align*}
S & \rightarrow ABB_a, \\
A & \rightarrow B_aB_aB_b, \\
B & \rightarrow AB_c, \\
B_a & \rightarrow a, \\
B_b & \rightarrow b, \\
B_c & \rightarrow c.
\end{align*}
\]
Step 2: Introduce additional variables to take care of the first two productions:

\[
\begin{align*}
S & \rightarrow AD_1, \\
D_1 & \rightarrow BB_a, \\
A & \rightarrow B_B D_2, \\
D_2 & \rightarrow B_a B_B, \\
B & \rightarrow AB_c, \\
B_a & \rightarrow a, \\
B_b & \rightarrow b, \\
B_c & \rightarrow c.
\end{align*}
\]
Greibach Normal Form

Definition
A CFG is said to be in Greibach normal form (GNF) if all productions have the form

\[ A \rightarrow ax, \]

where \( a \in T \) and \( x \in V^* \).

The form \( A \rightarrow ax \) is common to both Greibach normal form and s-grammars but GNF does not carry the restrictions that the pair \((A, a)\) occur at most once.

Theorem
For every CFG \( G \) with \( \epsilon \in L(G) \), there exists an equivalent \( \hat{G} \) in GNF.
From CFG to GNF

- Not trivial
- A simple idea is to eliminate the following:
  1. Front recursions: $A \rightarrow AB$
  2. Front non-terminals
  3. Non-front terminals
Consider a grammar involving the front recursion

\[ A \rightarrow AB | CD, \]

We convert this into the following equivalent grammar:

\[ A \rightarrow CDN | CD, \]
\[ N \rightarrow BN | B. \]
Consider a grammar involving the front non-terminal

\[
A \rightarrow BbC | aA, \\
B \rightarrow cDA | aE, 
\]

We convert this into the following equivalent grammar:

\[
A \rightarrow cDAbC | aEbC | aA, \\
B \rightarrow cDA | aE. 
\]
Consider a grammar involving the non-front terminal

\[ A \rightarrow cDAbC, \]

We convert this into the following equivalent grammar:

\[ A \rightarrow cDANC, \]
\[ N \rightarrow b. \]
**Example:** Consider a CFG with

\[
S \rightarrow SA | BeA, \\
A \rightarrow AS | a, \\
B \rightarrow b.
\]

Find an equivalent GNF?

*Solution.* We directly apply the technique explained in previous 3 slides. The GNF is given by

\[
S \rightarrow bEAC | bEA, \\
C \rightarrow aDC | aD | aC | a, \\
A \rightarrow aD | a, \\
D \rightarrow bEACD | bEAC | bEAD | bEA, \\
E \rightarrow e.
\]