1 Converting CFGs to Chomsky Normal Form (CNF)

A CNF grammar is a CFG with rules restricted as follows.

The right-hand side of a rule consists of:

i. Either a single terminal, e.g. $A \rightarrow a$.
ii. Or two variables, e.g. $A \rightarrow BC$.
iii. Or the rule $S \rightarrow \lambda$, if $\lambda$ is in the language.
iv. The start symbol $S$ may appear only on the left-hand side of rules.

Given a CFG $G$, we show how to convert it to a CNF grammar $G'$ generating the same language.

We use a grammar $G$ with the following rules as a running example.

$$S \rightarrow ASA \mid aB;\ A \rightarrow B \mid S;\ B \rightarrow b \mid \lambda$$

We proceed in a series of steps which gradually enforce the above CNF criteria; each step leaves the generated language unchanged.

Step 1 For each terminal $a$, we introduce a new variable, $U_a$ say, add a rule $U_a \rightarrow a$, and for each occurrence of $a$ in a string of length 2 or more on the right-hand side of a rule, replace $a$ by $U_a$. Clearly, the generated language is unchanged.

Example: If we have the rule $A \rightarrow Ba$, this is replaced by $U_a \rightarrow a, A \rightarrow BU_a$.

This ensures that terminals on the right-hand sides of rules obey criteria (i) above.

This step changes our example grammar $G$ to have the rules:

$$S \rightarrow ASA \mid U_aB;\ A \rightarrow B \mid S;\ B \rightarrow b \mid \lambda;\ U_a \rightarrow a$$
Step 2  For each rule with 3 or more variables on the right-hand side, we replace it with a new collection of rules obeying criteria (ii) above. Suppose there is a rule $U \rightarrow W_1 W_2 \cdots W_k$, for some $k \geq 3$. Then we create new variables $X_2, X_3, \cdots, X_{k-1}$, and replace the prior rule with the rules:

$$U \rightarrow W_1 X_2; \quad X_2 \rightarrow W_2 X_3; \quad \cdots; \quad X_{k-2} \rightarrow W_{k-2} X_{k-1}; \quad X_{k-1} \rightarrow W_{k-1} W_k$$

Clearly, the use of the new rules one after another, which is the only way they can be used, has the same effect as using the old rule $U \rightarrow W_1 W_2 \cdots W_k$. Thus the generated language is unchanged.

This ensures, for criteria (ii) above, that no right-hand side has more than 2 variables. We have yet to eliminate right-hand sides of one variable or of the form $\lambda$.

This step changes our example grammar $G$ to have the rules:

$$S \rightarrow AX \mid U_a B; \quad X \rightarrow SA; \quad A \rightarrow B \mid S; \quad B \rightarrow b \mid \lambda; \quad U_a \rightarrow a$$

Step 3  We replace each occurrence of the start symbol $S$ with the variable $S'$ and add the rule $S \rightarrow S'$. This ensures criteria (iv) above.

This step changes our example grammar $G$ to have the rules:

$$S \rightarrow S'; \quad S' \rightarrow AX \mid U_a B; \quad X \rightarrow S'A; \quad A \rightarrow B \mid S' \mid S; \quad B \rightarrow b \mid \lambda; \quad U_a \rightarrow a$$

Step 4  This step removes rules of the form $A \rightarrow \lambda$.

To understand what needs to be done it is helpful to consider a derivation tree for a string $w$. If the tree use a rule of the form $A \rightarrow \lambda$, we label the resulting leaf with $\lambda$. We will be focussing on subtrees in which all the leaves have $\lambda$-labels; we call such subtrees $\lambda$-subtrees. Now imagine pruning all the $\lambda$-subtrees, creating a reduced derivation tree for $w$. Our goal is to create a modified grammar which can form the reduced derivation tree. A derivation tree, and its reduced form is shown in Figure 1.

We need to change the grammar as follows. Whenever there is a rule $A \rightarrow BC$ and $B$ can generate $\lambda$, we need to add the rule $A \rightarrow C$ to the grammar (note that this does not allow any new strings to be generated); similarly, if there is is rule $A \rightarrow DE$ and $E$ can generate $\lambda$, we need to add the rule $A \rightarrow D$; likewise, if there is a rule $A \rightarrow BB$ and $B$ can generate $\lambda$, we need to add the rule $A \rightarrow B$.

Next, we remove all rules of the form $A \rightarrow \lambda$. We argue that any previously generatable string $w \neq \lambda$ remains generatable. For given a derivation tree for $w$ using the old rules, using the new rules we can create the reduced derivation tree, which is a derivation tree for $w$ in the new grammar.

Finally, we take care of the case that, under the old rules, $S$ can generate $\lambda$. In this situation, we simply add the rule $S \rightarrow \lambda$, which then allows $\lambda$ to be generated by the new rules also.
To find the variables that can generate $\lambda$, we use an iterative rule reduction procedure. First, we make a copy of all the rules. We then modify (reduce) the rules by removing from the right-hand sides all instances of variables $A$ for which there is a rule $A \rightarrow \lambda$. We keep iterating this procedure so long as it creates new reduced rules with $\lambda$ on the right-hand side.

For our example grammar we start with the rules

$$S \rightarrow S'; S' \rightarrow AX \mid U_a B; X \rightarrow S'A; A \rightarrow B \mid S'; B \rightarrow b \mid \lambda; U_a \rightarrow a$$

As $B \rightarrow \lambda$ is a rule, we obtain the reduced rules

$$S \rightarrow S'; S' \rightarrow AX \mid ; U_a; X \rightarrow S'; A \rightarrow \lambda \mid S'; B \rightarrow \lambda; U_a \rightarrow a$$

As $A \rightarrow \lambda$ is now a rule, we next obtain

$$S \rightarrow S'; S' \rightarrow X \mid U_a; X \rightarrow S'; A \rightarrow \lambda \mid S'; B \rightarrow \lambda; U_a \rightarrow a$$

There are no new rules with $\lambda$ on the right-hand side. So the procedure is now complete and this yields the new collection of rules:

$$S \rightarrow S'; S' \rightarrow AX \mid X \mid U_a B \mid U_a; X \rightarrow S'A \mid S'; A \rightarrow B \mid S'; B \rightarrow b; U_a \rightarrow a$$

An efficient implementation keeps track of the lengths of each right-hand side, and a list of the locations of each variable; the new rules with $\lambda$ on the right-hand side
are those which have newly obtained length 0. It is not hard to have this procedure run in time linear in the sum of the lengths of the rules.

Step 5 This step removes rules of the form $A \rightarrow B$, which we call unit rules.

What is needed is to replace derivations of the form $A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_k \Rightarrow BC$ with a new rule $A \Rightarrow BC$. We proceed in two substeps.

Substep 5.1. This substep identifies variables that are equivalent, i.e. collections $B_1, B_2, \cdots, B_l$ such that for each pair $B_i$ and $B_j$, $1 \leq i < j \leq l$, $B_i$ can generate $B_j$, and $B_j$ can generate $B_i$. We then replace all of $B_1, B_2, \cdots, B_l$ with a single variable, $B_1$ say. Clearly, this does not change the language that is generated.

To do this we form a directed graph based on the unit rules. For each variable, we create a vertex in the graph, and for each unit rule $A \rightarrow B$ we create an edge $(A, B)$. Figure 2(a) shows the graph for our example grammar. The vertices in each strong component of the graph correspond to a collection of equivalent variables.

For the example grammar, the one non-trivial strong component contains the variables $\{S', X\}$. We replace $S'$ with $X$ yielding the rules:

$S \rightarrow X; \ X \rightarrow AX \mid X \mid U_a B \mid U_a; \ X \rightarrow XA \mid X; \ A \rightarrow B \mid X; \ B \rightarrow b; \ U_a \rightarrow a$

We can remove the useless rule $X \rightarrow X$ also.

Substep 5.2. In this substep, we add rules $A \rightarrow BC$, as described above, so as to shortcut derivations that were using unit rules.

To this end, we use the graph formed from the unit rules remaining after Substep 5.1, which we call the reduced graph. It is readily seen that this is an acyclic graph.

In processing $A \rightarrow B$, we will add appropriate non-unit rules that allow the shortcutting of all uses of $A \rightarrow B$, and hence allow the rule $A \rightarrow B$ to be discarded. If there are no unit rules with $B$ on the left-hand side it suffices to add a rule $A \rightarrow CD$ for each rule $B \rightarrow CD$. 

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{unit_rules_graph.png}
\caption{(a) Graph showing the unit rules. (b) The reduced graph.}
\end{figure}
To be able to do this, we just have to process the unit rules in a suitable order. Recall that each unit rule is associated with a distinct edge in the reduced graph. As this graph will be used to determine the order in which to process the unit rules, it will be convenient to write “processing an edge” when we mean “processing the associated rule”. It suffices to ensure that each edge is processed only after any descendant edges have been processed. So it suffices to start at vertices with no outedges and to work backward through the graph. This is called a reverse topological traversal. (This traversal can be implemented via a depth first search on the acyclic reduced graph.)

For each traversed edge \((E, F)\), which corresponds to a rule \(E \rightarrow F\), for each rule \(F \rightarrow CD\), we add the rule \(E \rightarrow CD\), and then remove the rule \(E \rightarrow F\). Any derivation which had used the rules \(E \rightarrow F\) and \(F \rightarrow CD\) in turn can now use the rule \(E \rightarrow CD\) instead. So the same strings are derived with the new set of rules.

This step changes our example grammar \(G\) as follows (see Figure 2(b)):

First, we traverse edge \((A, B)\). This changes the rules as follows:
- Add \(A \rightarrow b\)
- Remove \(A \rightarrow B\).

Next, we traverse edge \((A, X)\). This changes the rules as follows:
- Add \(A \rightarrow AX |XA |U_aB |U_a\).
- Remove \(A \rightarrow X\).

Finally, we traverse edge \((S, X)\). This changes the rules as follows:
- Add \(S \rightarrow AX |XA |U_aB |U_a\).
- Remove \(S \rightarrow X\).

The net effect is that our grammar now has the rules

\[
S \rightarrow AX |U_aB |XA |U_a; X \rightarrow AX |U_aB |XA |U_a; A \rightarrow b | AX |U_aB |XA |U_a; B \rightarrow b; U_a \rightarrow a
\]

Steps 4 and 5 complete the attainment of criteria (ii), and thereby create a CNF grammar generating the same language as the original grammar.