Converting CFGs to CNF (Chomsky Normal Form)

Richard Cole

October 17, 2007

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 $S \rightarrow \epsilon$, if ϵ 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.

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

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:

$$S \rightarrow ASA \mid U_aB;\ A \rightarrow B \mid S;\ B \rightarrow b \mid \epsilon;\ U_a \rightarrow a$$

Step 2 For each rule with 3 or more variables on the righthand 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:

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

. 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 ϵ.

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

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

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:

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

Step 4 This step removes rules of the form $A \rightarrow \epsilon$, as follows. First, we determine all variables that can generate ϵ in one or more moves. We explain how to do this two paragraphs down. Then for each such variable A, for each occurrence of A in a 2-variable right hand side, we create a new rule with the A omitted; i.e. if there is a rule $C \rightarrow AB$ we create the new rule $C \rightarrow B$, and if there is a rule $C \rightarrow DA$ we create the new rule $C \rightarrow D$ (if there is a rule $C \rightarrow AA$, we create the new rule $C \rightarrow A$). Then we remove all rules of the form $A \rightarrow \epsilon$, apart from $S \rightarrow \epsilon$, if present (i.e. we keep rule $S \rightarrow \epsilon$, if present).

The new rules serve to shortcut a previously generatable instances of ϵ, i.e. if previously we had used a rule $A \rightarrow BC$, and then in a series of steps had generated ϵ from B, which has the net effect of generating C from A, we could instead do this directly by applying the new rule $A \rightarrow C$. Consequently, the generated language is unchanged.

To find the variables that can generate ϵ, we use an iterative rule reduction procedure. First, we make a copy of all the rules. We then modify the rules by removing from the right hand sides all instances of variables A for which there is a rule $A \rightarrow \epsilon$. We keep iterating this procedure so long as it creates new reduced rules with ϵ on the right hand side. (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 ϵ 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.)

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

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

Step 5 This step removes rules of the form $A \to B$, which we call *unit rules.* We form the directed graph defined by these rules, i.e. for each rule $A \to B$, we create a directed edge (A, B). For each strong component in this graph, we replace the variables it contains with a single one of these variables in all the rules in which these variables occur. So if $U_1, U_2, \cdots, U_k$ form a strong component (and so any one of these variables can be replaced, in a sequence of applications of unit rules, by any other of these variables) then we replace every occurrence of U_i, $2 \le i \le k$ with U_1 in every rule in which they occur.

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

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

We can remove the unnecessary rule $X \to X$ also.

Next, we traverse the resulting acyclic graph, in reverse topological order (i.e. starting at nodes with no outedges and working back from these nodes); for each traversed edge (E, F), which corresponds to a rule $E \to F$, for each rule $F \to CD$, we add the rule $E \to CD$, and then remove the rule $E \to F$. Any derivation which had used the rules $E \to F$ and $F \to CD$ in turn can now use the rule $E \to CD$ instead. So the same strings are derived with the new set of rules. (This can be implemented via a depth first search on the acyclic graph.)

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

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

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