Conversion of a Context-Free Grammar to Chomsky Normal Form
Robert M. Keller
14 April 2014

Chomsky Normal Form is a special case of context-free grammar (CFG) that has a variety of uses in algorithms for using grammars.

**Definition:** A CFG is in **Chomsky Normal Form (CNF)** provided that every production has one of these forms:

- $S \rightarrow \epsilon$, where $S$ is the start symbol and $S$ does not appear on the right-hand side of any production
- $A \rightarrow BC$, where $B$ and $C$ are non-terminal
- $A \rightarrow \sigma$, where $\sigma$ is terminal

In particular, the following are not allowed in CNF, among others

- $A \rightarrow \epsilon$, where $A$ is not the start symbol
- $A \rightarrow B$, where $B$ is non-terminal
- $A \rightarrow BCD$, where $B$, $C$, and $D$ are non-terminal
- $A \rightarrow Bo$, where $B$ is non-terminal and $\sigma$ is terminal

**Why is CNF useful?**

For one thing, it is useful to know at a glance whether or not $\epsilon$ is in the language generated. With CNF, it is trivial to find this out, as it can only happen if there is a production $S \rightarrow \epsilon$. A second reason is that is helpful if every production applied in a derivation makes progress toward ultimately generating a terminal string.

While it is possible to have a non-terminal in CNF that does not yield such a string, an additional simplification can eliminate such non-terminals. Once this simplification is performed, every application of a production in CNF either immediately yields a terminal symbol, or increases the length of the ultimate string by one symbol. Also, once the simplification is performed, we can tell by inspection whether a grammar yields any terminal strings at all.
**CNF Conversion Algorithm**

We present a five-step algorithm, essentially that in Sipser’s book, for converting an arbitrary CFG into a CNF one. At each step, the resulting grammar will be seen to be equivalent to the one before, and thus to the original.

**Step 1: Isolate the start symbol.**
Create a new start symbol $S_0$ and add the production $S_0 \rightarrow S$.
The purpose of this step is to prevent the start symbol from being a part of any cycles.

**Step 2: Remove $\varepsilon$-productions, i.e. productions of the form $A \rightarrow \varepsilon$**, with the possible exception of $S_0 \rightarrow \varepsilon$.
This may require multiple sub-steps. In order to remove $A \rightarrow \varepsilon$, we need to add productions that preserve the effect of $A \rightarrow \varepsilon$. If there is a production $B \rightarrow uAv$, where $u$ and $v$ are strings (possibly empty), then we need to add a production $B \rightarrow uv$. We need to add a new production for every combination of $A$’s that occur on the right-hand side of any other production. For example, if we originally had $B \rightarrow aAbAc$, then we need to add all of these: $B \rightarrow abAc$, $B \rightarrow abAc$, and $B \rightarrow abc$, because any of the $A$’s on the right-hand side could yield $\varepsilon$ in a derivation.

**Step 3: Remove unit-productions, i.e. productions of the form $A \rightarrow B$, where $B$ is non-terminal.**
As with step 2, multiple sub-steps may be required. For each non-terminal $A$ in turn, including the start symbol, and each production $A \rightarrow B$, identify all productions of the form $B \rightarrow u$ where $u$ is a string, and add the production $A \rightarrow u$, unless $A \rightarrow u$ is a unit rule previously removed, or $u$ is just $A$ (in which case we don’t add $A \rightarrow A$ as there is no point to that), then remove $A \rightarrow B$.

At this point, there is no production, other than possibly $S_0 \rightarrow \varepsilon$, with a right-hand side shorter than 2.

**Step 4: Replace any production with a right-hand side of length $n \geq 3$ with $n-1$ productions, each having a right-hand side of length 2.**
Suppose that $A \rightarrow u_1 u_2 \ldots u_n$, $n \geq 3$ is such a production. Add $n-1$ new non-terminals $A_2, \ldots, A_n$, and the following additional productions:

- $A \rightarrow u_1 A_2$
- $A_2 \rightarrow u_2 A_3$
- $\ldots$
- $A_{n-1} \rightarrow u_{n-1} u_n$
For example, if $A \rightarrow BCDE$, then $(n = 4)$, add $A_2, A_3, A_4$, and replace the production with new productions

$$A \rightarrow B A_2$$
$$A_2 \rightarrow C A_3$$
$$A_3 \rightarrow D E$$

**Step 5:** For any production with a right-hand side of length 2, if there is a terminal $\sigma$ on the right-hand side, replace $\sigma$ with a new non-terminal $B$ and add a production $B \rightarrow \sigma$.

For example, if we have a production $A \rightarrow bc$, where both right-hand side symbols are terminal, we would replace it with $A \rightarrow BC$, $B \rightarrow b$, and $C \rightarrow c$.

**Complete CNF Conversion Example**

Consider the grammar with start symbol $S$ and $\Sigma = \{', '\}$:

$$S \rightarrow (L)$$
$$L \rightarrow \varepsilon$$
$$L \rightarrow SL$$

Which productions violate CNF?

**Step 1:** New start symbol $S_0$:

$$S_0 \rightarrow S$$
$$S \rightarrow (L)$$
$$L \rightarrow \varepsilon$$
$$L \rightarrow SL$$

**Step 2:** Remove $\varepsilon$ productions:

Replace $L \rightarrow \varepsilon$ by adding:

$$S \rightarrow ()$$
$$L \rightarrow S$$

So we now have:

$$S_0 \rightarrow S$$
$$S \rightarrow (L)$$
$$S \rightarrow ()$$
$$L \rightarrow SL$$
$$L \rightarrow S$$

**Step 3:** Remove unit productions:

Replace $S_0 \rightarrow S$ with $S_0 \rightarrow (L)$ and $S_0 \rightarrow ()$.

Replace $L \rightarrow S$ with $L \rightarrow (L)$ and $L \rightarrow ()$.

Now we have:

$$S_0 \rightarrow (L)$$
$$S_0 \rightarrow ()$$
$$S \rightarrow (L)$$
$$S \rightarrow ()$$
L → SL
L → (L)
L → {}

Step 4: Introduce new non-terminal A for the productions with right-hand side (L):

S₀ → (A
A → L)
S₀ → {}
S → (A
S → {}
L → SL
L → (A
L → {}

Step 5: Introduce new non-terminals B and C for the terminals ( and ):

S₀ → BA
S₀ → BC
A → LC
S → BA
S → BC
L → SL
L → BA
L → BC

This is our CNF result.

Checking the Result

The result of conversion can be checked using Prolog’s grammar facility. See cfg2cnf.pro on the website. We checked all strings of lengths 0 to 8 to see which could be generated, and both grammars agreed, producing these 9 strings:

(), (()), (()()), ((())), (())(), (()(()), ((())()), (((()))), (((())))

Exercise

Show how to perform the extra step of determining the set of symbols that yield no terminal string, the “useless” symbols. Such symbols and productions containing them, can be removed from the grammar without changing the language generated.