Chomsky Normal Form

Why do we need a normal form in the first place. We want to compile. There may be different grammars using different rules. To make compiler design easier, it is often helpful to transform the original rules into a simpler form. This simplified description is called a *normal form.*

What is Chomsky normal form. In the Chomsky Normal Form (CNF), only three types of rules are allowed:

- rules of the type $S \to \varepsilon$, where S is the starting variable;
- rules of the type $V \to a$, where V is a variable and a is a terminal symbol; and
- rules of the type $V \to AB$, where V, A, and B are variables.

Who is Chomsky. Noam Chomsky is a great linguist who invented many of the concepts we use in this class. His political views are somewhat unusual: whatever happens in the world, he thinks that US is to blame. But from the scientific viewpoint, he is clearly a genius.

How to transform a grammar into the Chomsky normal form. The algorithm consists of a preliminary step and four main steps. For each step:

- first, we will explain what it does;
- then, we will provide an algorithm for this step; the algorithm will be in italics.

After that, we will illustrate all these steps on the example of the following grammar:

$$S \to \varepsilon; \quad S \to aAb; \quad A \to bSa; \quad A \to \varepsilon; \quad A \to S.$$

Preliminary step: description. *Introduce a new starting variable S_0 and a rule $S_0 \to S$, where S is the starting variable of the original grammar.*

Preliminary step: example. In the above example, we add the new rule $S_0 \to S$. So, the grammar takes the following form:

$$S \to \varepsilon; \quad S \to aAb; \quad A \to bSa; \quad A \to \varepsilon; \quad A \to S; \quad \underline{S_0 \to S}.$$

For clarity, on each step, the newly added rules are underlined.

Comment. This addition of a new state is similar to the preliminary step of a transition from a finite automaton to the corresponding regular expression:

- There, in the preliminary step, we introduced a new starting *state* with a jump from the new starting state to the old starting state.
- Here, similarly, we introduce a new starting *variable* S_0 and a direct transition (equivalent of jump) from the new starting variable S_0 to the old starting variable.

It should be mentioned that:

- In the transition from a finite automaton to a regular expression we also introduced a new final state and a jump from each original final state to the new final state.
- There is no equivalent procedure here, since context-free grammars have a starting variable, but they do not have any "final" variables: final symbols in the derivation are terminal symbols, not variables.

Step 0: motivations. First, we deal with rules in which the right-hand side has length 0: we make sure that all these rules are in Chomsky normal form.

The length of the word is simply the number of symbols in this word. So, length 0 means that the right-hand side has no symbol, i.e., that it is the empty string ε. In other words, we talk about the rules of the type $V \to \varepsilon$.

According to the definition of Chomsky normal form, the only rule of this type which is allowed is the rule in which V is the starting variable of the grammar (in our example, it is the variable S_0). So, all other rules need to be eliminated.

Step 0: algorithm. *If the grammar has the rule $A \to \varepsilon$, where A is not the starting variable, then:*

- *we delete this rule from the grammar, and*
- *for each rule that has A in the right-hand side, we add another rule in which this symbol A is deleted;*
- *if the right-hand side has several occurrences of this symbol A, then we add new rules in which each of these occurrences is deleted, add new rules in which two occurrences are deleted, etc.*

Step 0: motivations (cont-d). We want to keep exactly the same words derived in this grammar. How is the rule $A \to \varepsilon$ used in derivations? Since A is not a starting variable, it has to be generated by some rule, i.e., we must have a rule that has letter A in the right-hand side. If after that, we apply the rule $A \to \varepsilon$, this is equivalent to deleting A from that right-hand side.

Step 0: example. In our example, we need to eliminate two rules: $S \to \varepsilon$ and $A \to \varepsilon$.

To eliminate the rule $S \to \varepsilon$, we find all the rules that have S in the right-hand side; there are three such rules: $A \to bSa$, $A \to S$, and $S_0 \to S$.

- For the rule $A \to bSa$, if we delete the letter S from the right-hand side, we get the rule $A \to ba$ that we add to our grammar.
- For the rule $A \to S$, if we delete the letter S from the right-hand side, we get the rule $A \to \varepsilon$. This rule is already in the grammar, so we do not need to add it to the grammar.
- For the rule $S_0 \to S$, if we delete the letter S from the right-hand side, we get the rule $S_0 \to \varepsilon$ that we add to our grammar.

After we delete the rule $S \to \varepsilon$ and add the new rules $A \to ba$ and $S_0 \to \varepsilon$, we get the following grammar:

$$S \to aAb; \quad A \to bSa; \quad A \to \varepsilon; \quad A \to S; \quad S_0 \to S; \quad \underline{A \to ba}; \quad \underline{S_0 \to \varepsilon}.$$

To complete Step 0, we need to eliminate the rule $A \to \varepsilon$. In the above grammar, there is only one rule with letter A in the right-hand side: the rule $S \to aAb$. If we delete the letter A from the right-hand side, we get the new rule $S \to ab$ that add to our grammar. After we delete the rule $A \to \varepsilon$ and add the new rule $S \to ab$, we get the following grammar:

$$S \to aAb; \quad A \to bSa; \quad A \to S; \quad S_0 \to S; \quad A \to ba; \quad S_0 \to \varepsilon; \quad \underline{S \to ab}.$$

In this grammar, we no longer have rules of the type $V \to \varepsilon$ for non-starting variables V, so Step 0 is completed.

Step 1: motivations. Now, we deal with rules in which the right-hand side has length 1, i.e., in which the right-hand side is just one symbol. We make sure that all these rules are in Chomsky normal form.

The symbol in the right-hand side can be either a variable or a terminal symbol. According to the definition of Chomsky normal form, the only rule of this type which is allowed is the rule in which the right-hand side is a terminal symbol. This means that all the rules of the type $A \to B$, where A and B are both variables, need to be eliminated.

Step 1: algorithm. *If we have a rule $A \to B$, then:*

- *we delete this rule from our grammar, and*
- *for each rule $B \to w$ that has the variable B is the left-hand side (for any right-hand side w), we add a rule $A \to w$.*

Note. On Step 0, we looked for the variable A in the *right*-hand sides of the rules, now we look for a variable in the *left*-hand sides.

Step 1: motivations (cont-d). As a result of the derivation, we get a word consisting of only terminal symbols. Thus, when we get a letter B by applying the rule $A \to B$, we have to somehow delete B. The only way to do it is to use some rule $B \to w$. Thus, the use of these two rules is equivalent to applying a single rule $A \to w$.

Step 1: example. In the current grammar, we have two rules of the type $A \to B$: the rule $A \to S$ and the rule $S_0 \to S$.

We have two rules with S in the left-hand side: $S \to aAb$ and $S \to ab$. So, once we eliminate the rule $A \to S$, we have to add rules $A \to aAb$ and $A \to ab$. As a result, we get the following grammar:

$$S \to aAb;\ \ A \to bSa;\ \ S_0 \to S;\ \ A \to ba;\ \ S_0 \to \varepsilon;\ \ S \to ab;\ \ \underline{A \to aAb};\ \ \underline{A \to ab}.$$

Similarly, we eliminate the rule $S_0 \to S$ and add two new rules $S_0 \to aAb$ and $S_0 \to ab$:

$$S \to aAb;\quad A \to bSa;\quad A \to ba;\quad S_0 \to \varepsilon;\quad S \to ab;\quad A \to aAb;\quad A \to ab;$$

$$\underline{S_0 \to aAb};\quad \underline{S_0 \to ab}.$$

Step 2: motivations. Now, we deal with rules in which the right-hand side has length 2 (or more). Each symbol in the right-hand side can be either a variable or a terminal symbol. According to the definition of Chomsky normal form, the only rule of this type which is allowed is the rule in which the right-hand side consists of two variables. This means that all other rules of this type need to be eliminated.

Step 2: algorithm.

- *For each terminal symbol a, we introduce an auxiliary variable V_a and a rule $V_a \to a$.*
- *Then, in each rule in which the right-hand side has 2 or more symbols and at least one of them is a terminal symbol, we replace each terminal symbol with the corresponding variable.*

Step 2: example. In our grammar, we have two terminal symbols a and b. So, we introduce two new variables V_a and V_b and two new rules $V_a \to a$ and $V_b \to b$.

In the rule $S \to aAb$, we replace a with V_a and b with V_b, and get the new rule $S \to V_aSV_b$. We do the same replacement with all other rules in which the right-hand side has 2 or more symbols and at least one of them is a terminal symbol. As a result, we get the following grammar:

$$\underline{S \to V_aAV_b};\ \ \underline{A \to V_bSV_a};\ \ \underline{A \to V_bV_a};\ \ S_0 \to \varepsilon;\ \ \underline{S \to V_aV_b};\ \ \underline{A \to V_aAV_b};$$

$$\underline{A \to V_aV_b};\ \ \underline{S_0 \to V_aAV_b};\ \ \underline{S_0 \to V_aV_b};\ \ \underline{V_a \to a};\ \ \underline{V_b \to b}.$$

Step 3: motivations. Now, we deal with rules in which the right-hand side has length 3 or more. Chomsky normal allows at most two symbols in the right-hand side, so all such rules must be eliminated.

Step 3: algorithm.

- *We replace each rule of the type $V \to ABC$ with two rules: $V_{AB} \to AB$ for a new variable V_{AB} and $V \to V_{AB}C$.*
- *We replace each rule of the type $V \to ABCD$ with three rules: $V_{AB} \to AB$ for a new variable V_{AB}, $V_{ABC} \to V_{AB}C$ for a new variable V_{ABC}, and $V \to V_{ABC}D$; etc.*

Step 3: motivations. The notation AB is similar to the usual notation for multiplication. This analogy explains the above algorithm. Suppose that we need to multiply three numbers A, B, and C, i.e., to find the value $V = ABC$. The computer's Central Processing Unit can only multiply two numbers at a time. So, what the computer will do is:

- first multiply A and B, and get the intermediate result $V_{AB} = AB$; and
- then multiply the result V_{AB} by C, producing $V = V_{AB}C$.

This is exactly what we are doing here.

Step 3: example. According to the algorithm, the rule $S \to V_a A V_b$ is replaced by two rules: $V_{aA} \to V_a A$ and $S \to V_{aA} V_b$. After we perform the same replacement for all other rules that have three or more symbols in the right-hand side, we get the following grammar:

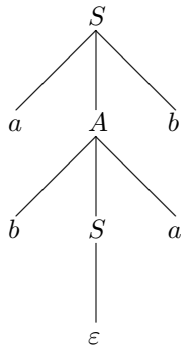
$$\underline{V_{aA} \to V_a A};\ \ \underline{S \to V_{aA} V_b};\ \ \underline{V_{bS} \to V_b S};\ \ \underline{A \to V_{bS} V_a};\ \ A \to V_b V_a;\ \ S_0 \to \varepsilon;$$

$$S \to V_a V_b;\ \ \underline{A \to V_{aA} V_b};\ \ A \to V_a V_b;\ \ \underline{S_0 \to V_{aA} V_b};\ \ S_0 \to V_a V_b;$$

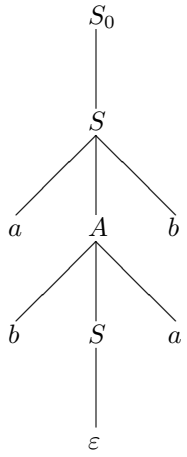
$$V_a \to a;\ \ V_b \to b.$$

One can easily check that this grammar is now in Chomsky normal form.

How this transformation changes the derivations. Let us illustrate this on the example of the word *abab* whose derivation in the original grammar has the following form:

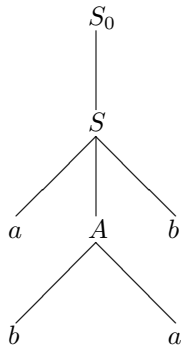


In the preliminary step, we add a new starting state S_0 and a new rule $S_0 \to S$. In terms of derivation, this means that on the top we have S_0, and then an additional transition from S_0 to S:



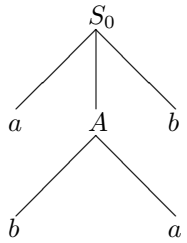
As a result, the complexity of the derivation practically does not change.

This derivation uses the rule $S \to \varepsilon$ that is eliminated on Step 0. We get the variable S that gets transformed into the empty string from the rule $A \to bSa$. On Step 0, this sequence of two rules $A \to b\underline{S}a \to ba$ is replaced by a single rule $A \to ba$:



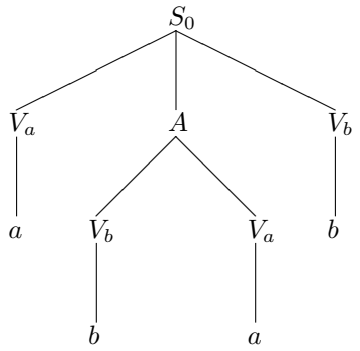
Interestingly, the derivation becomes shorter and simpler!

This derivation uses the rule $S_0 \to S$ that is eliminated on Step 1. On this step, we replace a sequence $S_0 \to \underline{S} \to aAb$ with a single rule $S_0 \to aAb$:



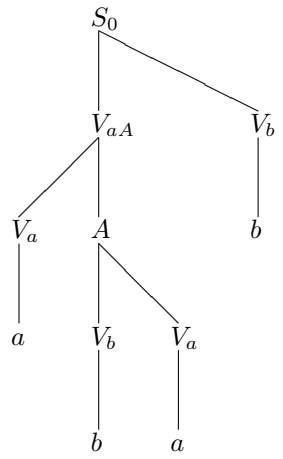
Again, the derivation becomes simplified.

On Step 2, we replace each letter a in the right-hand sides with the variable V_a, and b with V_b. These variables need to be converted in a and b by using the auxiliary rules $V_a \to a$ and $V_b \to b$:



The derivation becomes more complicated.

It will become even more complicated when we apply Step 3, when, instead of using the rule $S \to V_a A V_b$ with three symbols in the right-hand side, we need to use a sequence of two rules $S \to \underline{V_{aA}} V_b \to V_a A V_b$:



This is the derivation of the word *abab* in the Chomsky-form grammar.