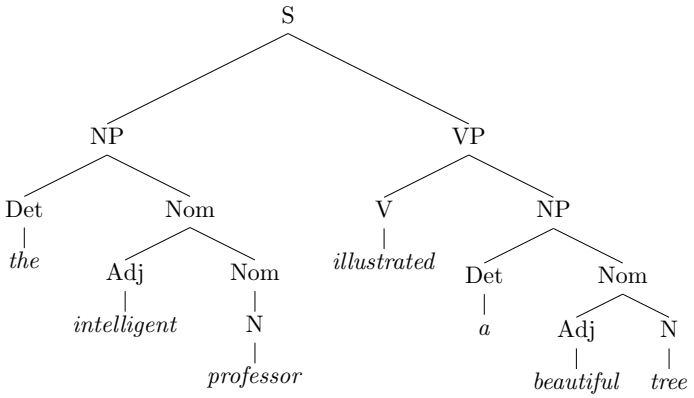
1.3 Ambiguity

If we are given a derivation of a word for some context-free grammar, we need not always represent it in a linear fashion like we did in the previous examples. We could alternatively represent it as a tree structure, where the root of the tree corresponds to the start nonterminal S and each branch of the tree adds a new nonterminal or terminal symbol. We refer to such trees as *parse trees*.

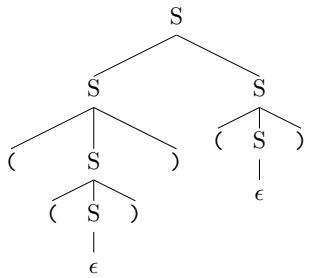
Parse trees are nothing new to linguists; the idea is used all the time to break down sentences or phrases into their constituent components, like nouns, verbs, and so on. In doing so, linguists are able to study the structures of sentences in different languages. For example, consider the following parse tree for an English sentence:



Here, the sentence (S) is broken down into a noun phrase (NP) and a verb phrase (VP); the noun phrase is broken down further into a determiner (Det) and a nominal (Nom); and so on. There are all kinds of rules specifying exactly how we can break down English sentences in this way.

In every parse tree, the root of the tree is the start nonterminal S, the leaves of the tree contain terminal symbols from Σ (or ϵ), and all other vertices of the tree contain nonterminal symbols from V. If a parse tree contains an internal (non-leaf) vertex A, and all the children of the vertex A are labelled $a_1, a_2, \ldots, a_n$, then the underlying grammar's rule set must contain a rule of the form $A \to a_1 a_2 \ldots a_n$.

Example 4. Recall our derivation of the word `(())()` from the language of words with balanced parentheses in Example 3. We can represent the derivation of that word by the following parse tree:



For most grammars we deal with, there exists exactly one way to generate any given word in the language of the grammar. However, this is not always the case. There are some grammars that allow us to generate the same word in more than one way.

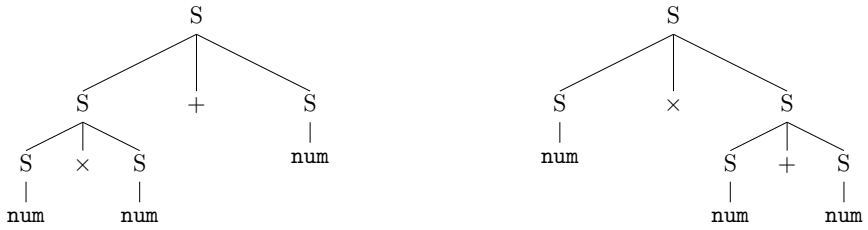
Perhaps one of the most well-known examples is the grammar generating the language of arithmetic expressions. If you recall grade school mathematics, you'll remember that there is an order of operations that specify the order in which we should apply arithmetic operations in a given expression.[2] We first evaluate expressions in parentheses, then exponents, then multiplications and divisions, and finally additions and subtractions.

[2]Sometimes referred to using the mnemonics BEDMAS, BIDMAS, BODMAS, or PEMDAS, depending on where you went to school.

Let's consider a simplified set of operations, where we only use parentheses, addition, and multiplication. The grammar generating the language of arithmetic expressions using these three operators together with the standard set of numbers is as follows:

$$S \to (S)$$
$$S \to S + S$$
$$S \to S \times S$$
$$S \to \texttt{num}$$

If we consider the expression $\texttt{num} \times \texttt{num} + \texttt{num}$, we discover that there exists more than one way to generate this expression, depending on whether we apply the $+$ rule or the $\times$ rule first. This is evidenced by the fact that there exist two parse trees for the same expression:

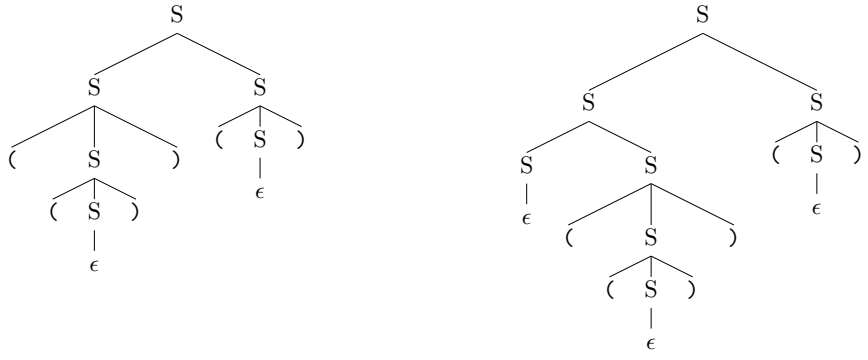


We don't need to do anything tricky in order to obtain these parse trees. In fact, both parse trees can be obtained simply by applying rules to each nonterminal from left to right; that is, at some level of the parse tree where there exists two nonterminals, we apply a rule to the first (left) nonterminal before the second (right) nonterminal. This process is known as a *leftmost derivation*.

If there exists some word in the language of a grammar for which there is more than one leftmost derivation of that word, then we say that the word is derived *ambiguously*. Likewise, the grammar producing that word is itself ambiguous.

Definition 5 (Ambiguous context-free grammar)**.** A context-free grammar G is ambiguous if there exists some word $w \in L(G)$ that can be derived ambiguously.

Example 6. The grammar from Example 3 generating our language of words with balanced parentheses is ambiguous. Consider again the word `(())()`. There exist two different parse trees corresponding to leftmost derivations of this word; the left parse tree was given in Example 4 and the right parse tree is new:



Removing Ambiguity

In some cases, if we have an ambiguous context-free grammar, we can create an equivalent context-free grammar with no ambiguity. For example, recall our three-operation arithmetic grammar from earlier:

$$\begin{aligned} S &\to \texttt{(}S\texttt{)} \\ S &\to S + S \\ S &\to S \times S \\ S &\to \texttt{num} \end{aligned}$$

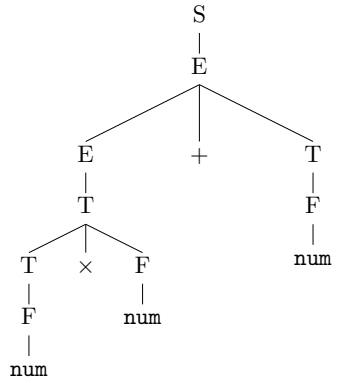
Nothing in this grammar forces us to use one rule before another, so we end up being able to derive the same word via different sequences of rule applications. However, we can construct an unambiguous grammar simply by adding a little more structure to our rules—specifically, by adding a few more nonterminals:

$$\begin{aligned} S &\to E \\ E &\to E + T \mid T \\ T &\to T \times F \mid F \\ F &\to \texttt{(}E\texttt{)} \mid \texttt{num} \end{aligned}$$

Now, each nonterminal plays a particular role. The nonterminal S, as usual, serves as our starting point and produces an expression, E. Each expression consists of subexpressions E or additive terms T. Likewise, each term consists of subterms T or multiplicative factors F. Finally, each factor can be either a `num` or a parenthesized subexpression, starting the whole process over again.

We won't verify here that this grammar is in fact equivalent to our original one, though we can intuit that they each generate the same language. Suffice it to say that, with this revised grammar, we're able to ensure that the addition rule is always applied before the multiplication rule.

Our revised grammar now allows us to draw one unambiguous parse tree for the expression $\texttt{num} \times \texttt{num} + \texttt{num}$:



Inherent Ambiguity

Unfortunately, there is no general procedure or algorithm for removing ambiguity from a context-free grammar; indeed, it isn't even possible to remove ambiguity in some cases. Some context-free languages are *inherently ambiguous*, meaning that any grammar generating the language will have some unavoidable ambiguous component to it.

Example 7. Consider the language $L_{\text{twoequal}} = \{\texttt{a}^i\texttt{b}^j\texttt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } j = k\}$ over the alphabet $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$. This language contains all words that have either the same number of `a`s and `b`s or the same number of `b`s and `c`s.

We can generate this language using the following grammar:

$$\begin{aligned} S &\to S_1 \mid S_2 \\ S_1 &\to S_1\mathtt{c} \mid A \\ A &\to \mathtt{a}A\mathtt{b} \mid \epsilon \\ S_2 &\to \mathtt{a}S_2 \mid B \\ B &\to \mathtt{b}B\mathtt{c} \mid \epsilon \end{aligned}$$

The rules S_1 and A generate words of the form $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m$ and the rules S_2 and B generate words of the form $\mathtt{a}^m\mathtt{b}^n\mathtt{c}^n$, each where $m, n \geq 0$.

Now, consider words of the form $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$, where $n \geq 0$. All words of this form belong to the language L_{twoequal}, but each such word has two distinct derivations in this grammar: it can be generated either by the rules S_1 and A, or by the rules S_2 and B.

While the formal proof that this language is inherently ambiguous is quite long, we can intuitively see that (for instance) any grammar generating this language must have rules similar to S_1 and A to produce balanced pairs of as and bs followed by some number of cs. We can make a similar argument for the rules S_2 and B. Thus, any grammar for this language will include some degree of ambiguity.

1.4 Normal Forms

Up to now, we've imposed no restrictions on the form of each rule in our context-free grammars. As long as each rule of our context-free grammar looked like $A \to \alpha$, where A is a nonterminal symbol and α is a combination of terminal and nonterminal symbols, we were happy.

However, computers (and, by extension, the people who program computers) like having structure. For instance, a compiler for a programming language usually incorporates a context-free grammar into its workflow at some point during the compilation of a program, and having a highly-structured grammar makes the compiler's job of determining which rule to apply both easier and faster.

Therefore, at times, we might like to transform a context-free grammar into a *normal form*; that is, to modify the grammar in such a way that each rule of the grammar follows a canonical form or template. There are many normal forms to choose from, and each one comes with its own benefits. However, here we will focus our attention on arguably the most popular normal form.

Chomsky Normal Form

The *Chomsky normal form*, as the name suggests, was first studied by Noam Chomsky in the late 1950s as he attempted to develop a model of natural language using grammars. A grammar in Chomsky normal form is one where each rule either has two nonterminal symbols or one terminal symbol on the right-hand side.

Definition 8 (Chomsky normal form)**.** A context-free grammar is in Chomsky normal form if every rule in the grammar is of one of the two following forms:

1. $A \to BC$ for $A, B, C \in V$ with $B, C \neq S$; or
2. $A \to a$ for $A \in V$ and $a \in \Sigma$.

Additionally, we may allow the rule $S \to \epsilon$.

The main benefit of converting a grammar into Chomsky normal form comes in how we can represent and store derivations of words in memory. Since each rule derives either two nonterminal symbols or one terminal symbol, every parse tree will have a branching factor of either 2 or 1. This fact allows us to use efficient data structures for representing binary trees in memory, as well as to apply efficient algorithms to process parse trees and derivations. Moreover, the number of steps in a derivation using a grammar in Chomsky normal form is easy to bound: if the grammar generates a word w, then the derivation of w will contain $|w| - 1$ applications of a rule of the first form and $|w|$ applications of a rule of the second form.

Example 9. Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, and consider the following two grammars. Each grammar generates words consisting of one b surrounded on either side by zero or more as. The grammar on the left is not in Chomsky normal form. The grammar on the right is in Chomsky normal form, and it is equivalent to the grammar on the left.

$$\begin{aligned} S &\to A\mathtt{b}A \\ A &\to A\mathtt{a} \mid \epsilon \end{aligned} \qquad\qquad \begin{aligned} S_0 &\to TA \mid BA \mid AB \mid \mathtt{b} \\ T &\to AB \\ A &\to AC \mid \mathtt{a} \\ B &\to \mathtt{b} \\ C &\to \mathtt{a} \end{aligned}$$

Every context-free grammar can be converted into a context-free grammar in Chomsky normal form, and the conversion process consists of four steps:

1. **START**: Replace the start nonterminal.

   Add a new start nonterminal S_0 together with a new rule $S_0 \to S$, where S is the start nonterminal of the original grammar. This guarantees that S_0 will not occur on the right-hand side of any rule.

2. **EPS**: Remove epsilon rules.

   Remove all rules of the form $A \to \epsilon$, where $A \neq S$. For each occurrence of A on the right-hand side of a rule in the original grammar, add a new rule with that occurrence of A removed from the right-hand side; that is, convert all rules of the form $X \to \alpha A \beta$ to $X \to \alpha\beta$, where α and β are combinations of nonterminal and terminal symbols. Note that this applies to each *occurrence* of A, so a rule of the form $X \to \alpha A \beta A \gamma$ would contribute three new rules: $X \to \alpha\beta A\gamma$, $X \to \alpha A \beta\gamma$, and $X \to \alpha\beta\gamma$.

   If there exists a rule of the form $X \to A$, then add a new rule of the form $X \to \epsilon$ unless we previously removed a rule of that form.

   Repeat until all epsilon rules not involving the original start nonterminal are removed.

3. **UNIT**: Remove unit rules.

   Unit rules are rules of the form $A \to B$, where A and B are nonterminal symbols. Remove all rules of this form.

   If there exists a rule of the form $B \to \alpha$, where α is a combination of nonterminal and terminal symbols, then add a new rule $A \to \alpha$ unless we previously removed a unit rule of that form.

   Repeat until all unit rules are removed.

4. **BIN**: Remove rules with more than two nonterminal or terminal symbols on the right-hand side.

   Replace each rule of the form $A \to \alpha_1\alpha_2 \ldots \alpha_k$, where $k \geq 3$ and each α_i is either a nonterminal or terminal symbol, with a series of new rules $A \to \alpha_1 A_1$, $A_1 \to \alpha_2 A_2$, $\ldots$, $A_{k-2} \to \alpha_{k-1}\alpha_k$. Each A_i is a new nonterminal symbol.

   If α_i is a terminal symbol in one of our new rules $A_{i-1} \to \alpha_i A_i$, then remove this rule, replace α_i with a new nonterminal symbol B_i, and add two new rules $B_i \to \alpha_i$ and $A_{i-1} \to B_i A_i$.

Example 10. Consider the following grammar not in Chomsky normal form:

$$\begin{aligned} S &\to ASB \\ A &\to \mathtt{a}AS \mid \mathtt{a} \mid \epsilon \\ B &\to S\mathtt{b}S \mid A \mid \mathtt{bb} \end{aligned}$$

We will convert this grammar to an equivalent grammar in Chomsky normal form.

1. **START**: Replace the start nonterminal.

   Adding a new start nonterminal and the rule $S_0 \to S$ gives us the following:

$$\begin{aligned} S_0 &\to S \\ S &\to ASB \\ A &\to \mathtt{a}AS \mid \mathtt{a} \mid \epsilon \\ B &\to S\mathtt{b}S \mid A \mid \mathtt{bb} \end{aligned}$$

2. **EPS**: Remove epsilon rules.

   The first epsilon rule we will remove is $A \to \epsilon$. Since A occurs on the right-hand side of two other rules ($S \to ASB$ and $A \to \mathtt{a}AS$), removing this rule means we must add two new rules $S \to SB$ and $A \to \mathtt{a}S$. Additionally, since we have a rule $B \to A$, we must add a new rule $B \to \epsilon$. This gives the following:

$$\begin{aligned} S_0 &\to S \\ S &\to ASB \mid SB \\ A &\to \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \\ B &\to S\mathtt{b}S \mid A \mid \mathtt{bb} \mid \epsilon \end{aligned}$$

   Next, we remove the epsilon rule $B \to \epsilon$ that we just added. Since B occurs on the right-hand side of two other rules ($S \to ASB$ and $S \to SB$), removing this rule means we must add two new rules $S \to AS$ and $S \to S$. This gives the following:

$$\begin{aligned} S_0 &\to S \\ S &\to ASB \mid SB \mid AS \mid S \\ A &\to \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \\ B &\to S\mathtt{b}S \mid A \mid \mathtt{bb} \end{aligned}$$

3. **UNIT**: Remove unit rules.

   We first remove the unit rule $B \to A$. We do so by replacing A on the right-hand side of the rule with everything that can be produced by a rule of the form $A \to \alpha$:

$$\begin{aligned} S_0 &\to S \\ S &\to ASB \mid SB \mid AS \mid S \\ A &\to \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \\ B &\to S\mathtt{b}S \mid \mathtt{bb} \mid \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \end{aligned}$$

   Next, we remove the unit rule $S \to S$. This change is more straightforward, as we don't need to modify the S rule in any other way:

$$\begin{aligned} S_0 &\to S \\ S &\to ASB \mid SB \mid AS \\ A &\to \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \\ B &\to S\mathtt{b}S \mid \mathtt{bb} \mid \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \end{aligned}$$

Lastly, we remove the unit rule $S_0 \to S$. Again, we replace S on the right-hand side of the rule with everything that can be produced by a rule of the form $S \to \alpha$:

$$\begin{aligned} S_0 &\to ASB \mid SB \mid AS \\ S &\to ASB \mid SB \mid AS \\ A &\to \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \\ B &\to S\mathtt{b}S \mid \mathtt{bb} \mid \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \end{aligned}$$

4. **BIN**: Remove rules with more than two nonterminal or terminal symbols on the right-hand side.

   Starting from the top, we replace the rule $S_0 \to ASB$ with the rules $S_0 \to AU_1$ and $U_1 \to SB$:

$$\begin{aligned} S_0 &\to AU_1 \mid SB \mid AS \qquad & U_1 &\to SB \\ S &\to ASB \mid SB \mid AS \\ A &\to \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \\ B &\to S\mathtt{b}S \mid \mathtt{bb} \mid \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \end{aligned}$$

   Similarly, we replace the rule $S \to ASB$ with the rules $S \to AU_2$ and $U_2 \to SB$:

$$\begin{aligned} S_0 &\to AU_1 \mid SB \mid AS \qquad & U_1 &\to SB \\ S &\to AU_2 \mid SB \mid AS \qquad & U_2 &\to SB \\ A &\to \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \\ B &\to S\mathtt{b}S \mid \mathtt{bb} \mid \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \end{aligned}$$

   Next, we replace the rule $A \to \mathtt{a}AS$ with the rules $A \to \mathtt{a}U_3$ and $U_3 \to AS$:

$$\begin{aligned} S_0 &\to AU_1 \mid SB \mid AS \qquad & U_1 &\to SB \\ S &\to AU_2 \mid SB \mid AS \qquad & U_2 &\to SB \\ A &\to \mathtt{a}U_3 \mid \mathtt{a} \mid \mathtt{a}S \qquad & U_3 &\to AS \\ B &\to S\mathtt{b}S \mid \mathtt{bb} \mid \mathtt{a}AS \mid \mathtt{a} \mid \mathtt{a}S \end{aligned}$$

   Moving along, we replace the rules $B \to S\mathtt{b}S$ and $B \to \mathtt{a}AS$ with the set of rules $B \to SU_4$, $B \to \mathtt{a}U_5$, $U_4 \to \mathtt{b}S$, and $U_5 \to AS$:

$$\begin{aligned} S_0 &\to AU_1 \mid SB \mid AS \qquad & U_1 &\to SB \\ S &\to AU_2 \mid SB \mid AS \qquad & U_2 &\to SB \\ A &\to \mathtt{a}U_3 \mid \mathtt{a} \mid \mathtt{a}S \qquad & U_3 &\to AS \\ B &\to SU_4 \mid \mathtt{bb} \mid \mathtt{a}U_5 \mid \mathtt{a} \mid \mathtt{a}S \qquad & U_4 &\to \mathtt{b}S \\ & & U_5 &\to AS \end{aligned}$$

   Lastly, we replace all rules containing one terminal and one nonterminal symbol on the right-hand side. We introduce two new rules $V_1 \to \mathtt{a}$ and $V_2 \to \mathtt{b}$ and make the appropriate changes to other rules:

$$\begin{aligned} S_0 &\to AU_1 \mid SB \mid AS \qquad & U_1 &\to SB \qquad & V_1 &\to \mathtt{a} \\ S &\to AU_2 \mid SB \mid AS \qquad & U_2 &\to SB \qquad & V_2 &\to \mathtt{b} \\ A &\to V_1U_3 \mid \mathtt{a} \mid V_1S \qquad & U_3 &\to AS \\ B &\to SU_4 \mid V_2V_2 \mid V_1U_5 \mid \mathtt{a} \mid V_1S \qquad & U_4 &\to V_2S \\ & & U_5 &\to AS \end{aligned}$$