Lecture 8

Context Free Grammars

In this lecture we start our second unit of the course, in which we will study a new class of languages: *context-free languages.* We will start by defining the notion of a *context-free grammar*.

8.1 Definitions

A context-free grammar is a way of representing languages using variables together with rules for how those variables can be converted into alphabet symbols. One motivation for it comes from linguistics (or from programming languages), as you might infer from the name "grammar". The idea is that in linguistics, you might have variables like S denoting a sentence, N denoting a noun, V denoting a verb, and A denoting an adjective, together with rules for how a sentence might decompose into these constituent parts of speech. For example, you might have a rule $S \to NV$ to denote that a sentence might decompose into a noun followed by a verb. You could have another rule like $S \to ANV$ to denote that a sentence might instead decompose into an adjective, noun, verb sequence. You could then also have rules that allow the variables A, N, and V to turn into real words: for example, $N \to \langle\text{John}\rangle$ might be a rule (we will treat $\langle\text{John}\rangle$ as a single symbol).

Such collections of variables and rules are called "context-free grammars". They are sometimes used in linguistics as well as in programming languages, where one needs to formally define the syntax of a language (so that the compiler would be able to tell you if there is a syntax error, for example).

In this course, our context-free grammars (CFGs) might look something like this:

$$\begin{aligned} S &\to XY \\ X &\to 0X1 \\ Y &\to 2Y \\ X &\to \epsilon \\ Y &\to 3. \end{aligned}$$

The interpretation of this will be as follows. The capital letters S, X, and Y are called *variables.* One variable, usually denoted S, is special: it is called the *start variable.* The expressions such as $X \to 0X1$ are called *rules*, and tell us how we are allowed to turn the variable on the left (in this case X) into a new string of variables and alphabet symbols. To generate a string from a context-free grammar, we start with the start variable S, and then use any number of rules (in any order) to replace the variables with other strings, until we get a string that has only alphabet symbols and no variables; such a string is said to be *generated* by the context-free grammar.

For example, in the CFG above, we can do the following:

$$S \to XY \to 0X1Y \to 00X11Y \to 0011Y \to 00112Y \to 001123.$$

The string 001123 is therefore generated by the CFG. Note what happened: we kept taking a variable and replacing it with a string according to one of the rules of the CFG (when the variable X disappears, it is because we used the rule $X \to \epsilon$). Each time we do that, we keep the order of the rest of the symbols fixed; the variables are each replaced with a string that is placed in the same position the variable was, with no movement. Note that one context-free grammar can generate a lot of strings; in this case, our CFG generates the language $\{0^n 1^n 2^m 3 : n, m \in \mathbb{N}\}$.

Let us now give a formal mathematical definition of a context-free grammar.

Definition 8.1. *A context-free grammar is a* 4*-tuple*

$$G = (V, \Sigma, R, S),$$

where

1. *V is a finite, non-empty set (we call its elements "variables"),*
2. *Σ is an alphabet (a finite, non-empty set of symbols) which must be disjoint from V,*
3. *R is a set of* rules*; each rule is a string of the form $A \to w$, where $A \in V$ is a variable, $\to$ is a special symbol not in V or Σ, and w is a string in $(V \cup \Sigma)^*$,*
4. *S is a variable in V, called the start variable.*

We will use ϵ as shorthand for the empty string. Technically, in the above definition, the rule $X \to \epsilon$ will be written "$X \to$", but we will write $X \to \epsilon$ to clarify that the empty string is meant to be there. (To avoid confusion, we will never use ϵ as a variable in V or symbol in Σ.)

We will introduce some further notation that will help us formally define the language generated by a context-free grammar G.

Definition 8.2. *For a context-free grammar $G = (V, \Sigma, R, S)$, we define the* yields relation *$\Rightarrow_G$ to relate two strings over $V \cup \Sigma$. For $x, y \in (V \cup \Sigma)^*$, we say that $x \Rightarrow_G y$ if we can write $x = uAv$ and $y = uwv$ such that $A \in V$, $u, v, w \in (V \cup \Sigma)^*$, and such that $A \to w$ is a rule in R.*

Less formally, the yields relation relates some pairs of strings over $(V \cup \Sigma)^*$ to each other; two such strings x and y are related if we can get from x to y using a single rule of G. If x and y a related that way, we write $x \Rightarrow_G y$, and we say that x yields y. Note that formally, a relation is just a set of ordered pairs, so $\Rightarrow_G$ can be viewed as a subset of $(V \cup \Sigma)^* \times (V \cup \Sigma)^*$.

We have now formally defined what it means for one string in $(V \cup \Sigma)^*$ to yield another string in $(V \cup \Sigma)^*$ (remember that there are multiple options for what a string in $(V \cup \Sigma)^*$ can yield). This yields relation corresponds to applying a single rule of G. We will now define the transitive closure of this operation, to get a relation that corresponds to all the strings we can derive using any number of rules.

Definition 8.3. *For a context-free grammar $G = (V, \Sigma, R, S)$, we define the relation $\Rightarrow_G^*$ over pairs of strings in $(V \cup \Sigma)^*$ as follows. For $x, y \in (V \cup \Sigma)^*$, we say that $x \Rightarrow_G^* y$ a finite sequence of m strings $z_1, z_2, \ldots, z_m \in (V \cup \Sigma)^*$ such that $z_1 = x$, $z_m = y$, and for each $i = 1, 2, \ldots, m-1$, we have $z_i \Rightarrow_G z_{i+1}$.*

In other words, we have $x \Rightarrow_G^* y$ if, starting with x, we can apply a finite number of rules of G to get to the string y.

Finally, using this definition, we can formally define the language generated by a context-free grammar.

Definition 8.4. *Let $G = (V, \Sigma, R, S)$ be a context-free grammar. We define the language generated by G, denoted $L(G)$, to be*

$$L(G) := \{x \in \Sigma^* : S \Rightarrow_G^* x\}.$$

In other words, $L(G)$ is the set of all strings in Σ^* that we can derive by starting from S and using a finite number of rules of G. Note that only strings in Σ^* are included in $L(G)$; strings over $(V \cup \Sigma)^*$ that have variables inside of them do not count as strings in $L(G)$.

If $G = (V, \Sigma, R, S)$ is a context-free grammar and $x \in L(G)$ is a string, then using the definitions above, we know that $S \Rightarrow_G^* x$, and so there is a finite sequence of strings $z_1, z_2, \ldots, z_m \in (V \cup \Sigma)^*$ such that $S = z_1 \Rightarrow_G z_2 \Rightarrow_G \cdots \Rightarrow_G z_m = x$. The sequence $(z_1, z_2, \ldots, z_m)$ is said to be a *derivation* of x in G. There may be multiple derivations for each string in $L(G)$. A string $x \in \Sigma^*$ is in $L(G)$ if and only if it has a derivation in G.

Finally, we define the notion of a context-free language.

Definition 8.5. *A language $A \subseteq \Sigma^*$ is called* context-free *if there exists a context-free grammar $G = (V, \Sigma, R, S)$ such that $L(G) = A$.*

We will abbreviate "context-free language" by CFL, and "context-free grammar" by CFG. Note that CFLs and CFGs are not quite the same thing: CFL refers to the *language* (that is, a set of strings), while CFG refers to the *grammar* (that is, a 4-tuple (V, Σ, R, S)). Note that each CFG G has a unique CFL $L(G)$, but each CFL might have multiple CFGs.

8.2 Examples of context-free grammars

Now that we have formally defined context-free grammars and their languages, let's go through some examples. Before we do so, however, we'll introduce a small simplification of the notation: if we have multiple rules with the same variable on the left hand side (for example, $X \to 0X1$ and $X \to \epsilon$), we will often write them in the same line using a vertical bar to separate them. That is, instead of writing

$$\begin{aligned} X &\to 0X1 \\ X &\to \epsilon, \end{aligned}$$

we will write

$$X \to 0X1|\epsilon.$$

For example, the grammar we gave at the beginning of this lecture can then be written

$$\begin{aligned} S &\to XY \\ X &\to 0X1|\epsilon \\ Y &\to 2Y|3. \end{aligned}$$

Example 1. Is the language $\{0^n 1^n : n \in \mathbb{N}\}$ context-free?

The answer is yes: we can generate it using the CFG $S \to 0S1|\epsilon$. Let's now argue that this CFG generates the right language. First, for each string of the form $0^n 1^n$, we can write a derivation for it of the form $S \to 0S1 \to 00S11 \to \cdots \to 0^n S 1^n \to 0^n 1^n$. That is, our derivation sequence will be $(z_1, z_2, \dots, z_m)$ with $m = n+2$, $z_i = 0^i S 1^i$ for $i = 1, 2, \dots, n+1$, and $z_{n+1} = 0^n 1^n$. It is clear that this is a valid derivation for $0^n 1^n$, because each string in the sequence can be derived from the previous one using a rule of the CFG. Hence each string in $\{0^n 1^n : n \in \mathbb{N}\}$ is generated by the CFG.

We are not done! We also need to show the reverse direction, to ensure that the CFG doesn't generate any strings not in the desired language. To this end, let $x \in \{0,1\}^*$ be any string generated by the CFG, and let $(z_1, z_2, \dots, z_m)$ be its derivation, so that $z_1 = S$, $z_m = x$, and z_{i+1} can be reached from z_i by following a rule of the CFG. Since each rule of the CFG replaces S with a string that has at most one variable, each z_i can have at most one variable; moreover, since no more strings can be derived when we have no variables left, we conclude that z_m is the only string with no variables, and all other z_i have exactly one variable – and hence exactly one copy of S inside of them. This means that except for the transition from z_{m-1} to z_m, all other transitions must use the first rule of the CFG, that is, $S \to 0S1$. It therefore follows that $z_i = 0^{i-1} S 1^{i-1}$ for all $i \leq m-1$, and hence $x = z_m = 0^{m-1} 1^{m-1}$, which is in the language. This completes the proof.

Example 2. What is the language generated by the CFG whose only rule is $S \to SS$?

In this CFG, the only rule is $S \to SS$, a rule that can never lead to any string in Σ^* (since the number of variables never decreases). This means that no string has a derivation in this CFG, so the language of this CFG is $\varnothing$.

Example 3. Let $P = \{w \in \{0,1\}^* : w = w^R\}$ be the language of all palindromes over the alphabet $\{0,1\}$. Is P context-free?

Yes, P is a CFL, because we can construct a context-free grammar for it: $S \to 0S0|1S1|0|1|\epsilon$. One would have to argue that this grammar indeed generates P, but this is not hard to do and we leave it for the reader.

Example 4. Let B be the language of all balanced parentheses over the alphabet $\{(,)\}$. That is, B consists of strings such as $(()())(())$ or $((()))$ or ϵ, which are strings over the alphabet $\{(,)\}$ such that the left-hand brackets and the right-hand brackets can be correctly matched. Formally, we say that a string over this alphabet is balanced if repeatedly deleting the substring $()$ can yield the empty string; B then contains all balanced strings of parentheses. Give a context-free grammar for B.

One answer is $S \to SS|(S)|\epsilon$. The idea is that each non-empty balanced string of parentheses can be decomposed in one of two ways: either it looks like $w_1 w_2$ where w_1 and w_2 are balanced strings of parentheses, or else it looks like (w) where w is a balanced string of parentheses. Once we have such a decomposition, the context-free grammar follows easily. Formally, to show that this context-free grammar generates B, we need to show both directions.

First, we show that if a string is generated by the grammar, it is balanced. To see this, suppose by contradiction that it was false, and consider the smallest counterexample w. This counterexample cannot be the empty string. Consider its shortest derivation in the grammar. This derivation starts with either $S \to SS$ or $S \to (S)$. If it starts with $S \to SS$, then let w_1 and w_2 be the strings in $\{(,)\}*$ that the two S variables turn into, respectively. We must have $w_1 \neq \epsilon$ and $w_2 \neq \epsilon$, since otherwise there would be a shorter derivation of $w = w_1 w_2$ in the grammar. Therefore, w_1 and w_2 are shorter than w and are generated by the grammar, so they are in B. This means we can repeatedly delete the substring () from each to get to ϵ, and using the same strategy, we can repeatedly delete () from w to get to ϵ. Thus $w \in B$, which gives a contradiction. On the other hand, if the derivation starts with $S \to (S)$, then we have $w = (w_1)$, where w_1 is generated by the grammar and is shorter than w. This means that $w_1 \in B$, so we can repeatedly delete the substring () from w_1 to get to ϵ. Using this strategy, we can repeatedly delete () from w to get to (), and then we can delete () again to get to ϵ, so $w \in B$, giving a contradiction.

Now let's show the other direction: if a string is in B, then it is generated by the grammar. Once again, we'll assume by contradiction that this is false, and consider the smallest counterexample $w \in B$. We must have $w \neq \epsilon$. Since $w \in B$, we can repeatedly delete () from w to get to ϵ. Consider the last pair of brackets that are deleted from w in this way, right before reaching ϵ. All the previous () substrings deleted must be either strictly between these two brackets, or else to the left of both brackets, or else to the right. From this, it follows that we can write $w = w_1(w_2)w_3$, where w_1, w_2, w_3 are balanced strings (we can delete () from them repeatedly to get to ϵ) and the two written brackets around w_2 are the last to be deleted from w. Hence $w_1, w_2, w_3 \in B$, and they are all shorter than w, so they can be generated by the grammar. Now, we can write

$$S \Rightarrow_G SS \Rightarrow_G SSS \Rightarrow_G S(S)S \Rightarrow_G^* w_1(w_2)w_3,$$

which shows that w can be generated by the grammar as well, giving a contradiction.

Example 5. Let $A = \{w \in \{0,1\}^* : |w|_0 = |w|_1\}$ be the language of all strings with the same number of 0s and 1s (we use $|w|_0$ to denote the number of 0s in w, and similarly for $|w|_1$). Show that A is context-free.

This is a bit tricky. Here is one solution: $S \to 0S1S|1S0S|\epsilon$. It is clear that each string generated by this grammar has an equal number of 0s and 1s; it is less clear that this grammar can generate all such strings. Let's prove this.

As in the previous example, we will proceed by contradiction, and consider the shortest string w that has an equal number of 0s and 1s but which is not generated by the grammar. (This trick, of considering the smallest counterexample and then later using the property that all smaller strings are not counterexamples, is equivalent to a proof by induction on the length of the string.)

If $w = \epsilon$, it can clearly be generated by the grammar, so we can assume $w \neq \epsilon$; let's say w has n zeroes and n ones, with $n \geq 1$. We can assume without loss of generality that w starts with 0 (a completely analogous argument will work if w starts with 1; just flip the symbols 0 and 1). At each position i of the string w, let ℓ_i be the number of zeroes in $w_1w_2\dots w_i$ minus the number of ones in $w_1w_2\dots w_i$. Then $\ell_1 = 1$, since $w_1 = 0$. Moreover, $\ell_{2n} = 0$, since w has length $2n$ and has the same number of 0s and 1s. Consider the smallest $j \geq 1$ such that $\ell_j = 0$; this is the shortest non-empty prefix of w that has the same number of 0s and 1s. Note that when i increases by 1, the number ℓ_i must either increase by 1 or decrease by 1. Since $\ell_1 = 1 > 0$ and $\ell_j = 0$, and since j is the first time ℓ_i reaches 0, we must have $\ell_{j-1} = 1$. Since ℓ_j is one smaller than ℓ_{j-1}, we must have $w_j = 1$. Moreover, since $\ell_j = 0$ and $w_1 = 0$, $w_j = 1$, the string $w_2w_3\dots w_{j-1}$ must have the same number of 0s and 1s. Hence we can write $w = 0w_11w_2$, where w_1 has the same number of 0s and 1s. Since w also has an equal number of 0s and 1s, the string w_2 must also have this property. Thus w_1 and w_2 are in A and are shorter than w, so they have derivations in the grammar. We now write

$$S \Rightarrow_G 0S1S \Rightarrow_G^* 0w_11w_2,$$

which gives a derivation of w in the grammar, giving a contradiction.

Example 6. Let's try a more difficult example. Consider the language $C = \{x\#y : x, y \in \{0,1\}^*, x \neq y\}$ over the alphabet $\{0, 1, \#\}$. Find a context-free grammar for C.

The solution to this one is rather complicated. To start, consider how to generate strings $x\#y$ where x and y have different lengths. If we want x to be longer, we could generate such strings using a variable Z, with the rules $Z \to XZX|XY\#$, where X is a variable that generates one bit (with rules $X \to 0|1$) and Y is a variable that generates an arbitrary string in $\{0,1\}^*$ (with rules $Y \to XY|\epsilon$). If we want Z to also generate strings $x\#y$ with $|y| > |x|$, we could add another rule for Z to get $Z \to XZX|XY\#|\#YX$.

So far, we know how to generate all the strings of the form $x\#y$ such that $|x| \neq |y|$. What remains is to generate strings of the form $x\#y$ such that $|x| = |y|$ but $x \neq y$. For such strings, there will be some n such that $x_n \neq y_n$. To generate this, it will suffice to generate strings of the form $x_1 0 x_2 \# y_1 1 y_2$ with $|x_1| = |y_1|$ (we don't need to specify that $|x_2| = |y_2|$, because it's OK if that isn't true; the two strings x and y will still not be equal, and it's OK to re-generate some of the strings $x\#y$ with $|x| \neq |y|$). We will similarly want to generate strings of the form $x_1 1 x_2 \# y_1 0 y_2$ with $|x_1| = |y_1|$.

To generate strings $x_1 0 x_2 \# y_1 1 y_2$ with $|x_1| = |y_1|$, we can use $W_0 1 Y$ (here Y generates an arbitrary string corresponding to y_2) with the rule $W_0 \to XW_0X|0Y\#$. To generate strings $x_1 1 x_2 \# y_1 0 y_2$ with $|x_1| = |y_1|$, we can use $W_1 0 Y$ with the rule $W_1 \to XW_1X|1Y\#$. Putting all these rules together (and using an additional start variable), we get the grammar

$$\begin{aligned}
S &\to Z \mid W_0 1 Y \mid W_1 0 Y \\
Z &\to XZX \mid XY\# \mid \#YX \\
W_0 &\to XW_0X \mid 0Y\# \\
W_1 &\to XW_1X \mid 1Y\# \\
Y &\to XY \mid \epsilon \\
X &\to 0 \mid 1.
\end{aligned}$$

Note that we did not yet prove this grammar correct; we only motivated how we came up with it. We won't go through the proof here (it's a bit tedious and not very enlightening).