A Characterization of Context-Free Languages

As alluded to in the title of this current chapter, context-free grammars "place parentheses." When writing grammars for languages like $\{a^i b^i c^j d^j : i, j \geq 0\}$, it helps to recognize the pattern of "parentheses" in the language: Every "opening" a is matched by a "closing" b, and all of this is then followed by opening cs matched by closing ds. The type of production that is at the core of a grammar for such matching characters is

$$T \rightarrow a\ T\ b$$

which results in a path of Ts in a derivation tree with matching as and bs being generated on the left and right of the path. A complete grammar for the above language is

$$\begin{aligned} S &\rightarrow T\ T' \\ T &\rightarrow a\ T\ b \mid \lambda \\ T' &\rightarrow c\ T'\ d \mid \lambda \end{aligned}$$

On the other hand, a non-nested pattern of matching characters as in $\{a^i b^j c^i d^j : i, j \geq 0\}$ is beyond the power of context-free grammars (as can be shown with a pumping argument).

It is easy to see that any depth of nesting with any set of parentheses-like characters is within the reach of context-free grammars. The grammar

$$\begin{aligned} S &\rightarrow S\ S \mid \lambda \\ S &\rightarrow (\ S\ ) \\ S &\rightarrow [\ S\ ] \\ S &\rightarrow \{\ S\ \} \\ S &\rightarrow a \mid b \end{aligned}$$

generates all properly nested strings of these n types of parentheses with letters $a_1 \ldots a_m$ interspersed. Let's call this language $PAR_n(T)$. Furthermore, for every set A of characters and every language L, let $Erase_A(L)$ be the set of all strings in L from with all occurrences of characters in A have been deleted.

Theorem 20 *Every context-free language is equal to*

$$Erase_A(R \cap PAR_n(T))$$

for some sets A and T of characters, some regular language R, and some integer $n \geq 0$.

What this theorem says is that every context-free language can be built by starting with a context-free grammar that produces essentially nothing but nested parentheses, and intersecting this language with a regular language and finally throwing away all occurrences of certain characters. The only context-free step in this construction is the initial generation of $PAR_n(T)$. This supports the contention that the only aspect of a context-free language that could possibly require a context-free grammar is some form of placement of parentheses and conversely, any placement of matching characters that does not follow the nesting pattern of parentheses is bound to be beyond the power of context-free grammars. [A proof of the above theorem can be found in *Davis and Weyuker*.]

Exercises

Ex. 1. Which of the following languages are context-free?

1. $L = \{a^n b^m : n, m \geq 1$ and both even $\}$
2. $L = \{a^n b^m c^q : n, m, q \geq 1$ and at least two of the three numbers are equal $\}$
3. $L = \{a^n b^m c^q : n, m, q \geq 1$ and at least two of the three numbers are different $\}$

Ex. 2. In the dynamic programming algorithm for parsing, if all the entries in the top row are $\emptyset$, does this imply that the string is not in the language? What if one of them is $\emptyset$? What if all the entries in the second row are $\emptyset$? The third row? Fourth?

Ex. 3. Is it decidable if a context-free language, given by a context-free grammar

1. ... contains a string of length less than 100?
2. ... contains a string of length more than 100?
3. ... is finite?

Ex. 4. Is it decidable if the set of all outputs of a program forms a context-free language? Explain briefly.

Pumping Lemma

Ex. 5. Consider the language $L = \{\alpha\#\beta : \alpha, \beta \in \{0,1\}^*$, α and β have the same length, and the bitwise AND between α and β is not all zeroes $\}$. Examples of strings in L are 01#11, 010000#111111, 010101#101011.

Prove that L is *not* context-free.

Ex. 6. Which of the following languages are context-free?

1. $L_{10} = \{a^i b^j a^j b^i : i, j \geq 0\}$
2. $L_{11} = \{a^i b^j a^i b^j : i, j \geq 0\}$
3. $L_{12} = \{\alpha_1\#\alpha_2\#\cdots\#\alpha_p\$\beta_1\#\beta_2\#\cdots\#\beta_q : p, q \geq 0,$
$\alpha_i, \beta_i \in \{0,1\}^\star,$ and $\{\alpha_1, \alpha_2, \cdots, \alpha_p\} \supseteq \{\beta_1, \beta_2, \cdots, \beta_q\}\}$

(L_{12} captures the properties of some programming languages that variables need to be declared before they are used.) Prove your answer either by giving a grammar or by providing an argument that the language is not context-free.

Ex. 7. Even if A and B are both context-free, $A \cap B$ may not be. Prove this by giving an example of two context-free languages whose intersection is not context-free.

Ex. 8. The proof of the Pumping Lemma did not show that the pumped strings v and y could not both be empty. This might happen when the grammar has what is known as "unit productions" — productions that merely replace one variable by another. (An example of a unit production is $E \to T$ in one of our grammars for arithmetic expressions.)

(a) The mere presence of unit productions does not yet make it possible for both v and y to be empty. For example, our grammar for arithmetic expressions (the version with variables E, F, and T) has unit productions but does not have derivation trees with v and y both empty. Something else must be true about the unit productions. What is that?

(b) There are two ways to correct this deficiency in the proof. One is to show that unit productions can be removed can be removed from any grammar without changing the language — this involves suitable changes to the other productions. The other is to rephrase the proof of the Pumping Lemma to work around the fact that some nodes of a parse tree might only have one child. The latter approach entails a redefined notion of "height" of a parse tree. Usually, the height of a tree is defined to be 1 plus the maximum height of its subtrees. What would be a suitable modification of this definition?

Ex. 9. Any finite-state machine for a language L can be turned into a context-free grammar for L. If the machine has a transition labeled c from state A to state B, then the grammar has a production $A \to cB$.

(a) Carry out this translation from machine to grammar for the machine that accepted binary numbers divisible by 3.

(b) What other productions need to be added to the grammar to make sure it derives the same strings that the machine accepts? (Give an answer for the specific example and also for the general case.)