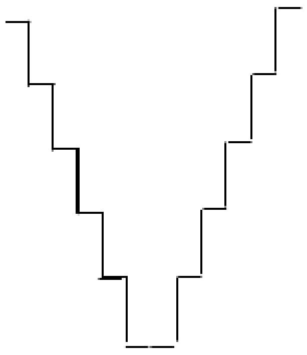
Additional Example to Practice with Context-Free Grammars
Martha Kosa

In the module, you learned the essential parts of a context-free grammar. Now you can practice some more.

The study of architecture involves lots of patterns, both symmetric and asymmetric, for designing buildings, stairs, and many other structures. The rules of context-free grammars describe patterns (productions) for valid strings so that valid strings can be generated from the start symbol and sample strings can be checked for validity by parsing (you will learn about several parsing algorithms as well).

The production rules allow for all the basic building blocks of problem solving (sequencing, selection, and repetition) to be used. Sequencing is implied by the order of symbols on the right-hand sides of production rules. Selection (for choice or union, when sets of strings are generated) is allowed when there are multiple production rules with the same variable on the left-hand side. Repetition is allowed when there are recursive production rules. A directly recursive rule is a production rule with the same variable appearing on both sides of the rule. The rules $A \rightarrow xA$ and $A \rightarrow y$ generate 0 more more copies of x followed by y. Recursion can also be indirect; with the rules $A \rightarrow aB$ and $B \rightarrow cA$, A derives acA after two steps.

Why don't we use context-free grammars to generate some simple architectural structures? These structures could also be drawn in computer games. Suppose we wanted to draw pictures like the following.



What alphabet could we use for our instructions? We could use **u, d, r** for **up, down,** and **right**, respectively. What pattern do you see? First we repeat the pattern **right** followed by **down**, then have one **right**, and finally repeat the pattern **right** followed by **up** the same number of times. There is one extra **right** at the end. This pattern may be used for going down into a fountain or amphitheater area and then up again.

How can you use the ideas from the grammar for the earlier example $\{0^n1^n \mid n \geq 0\}$ to develop a grammar for valid step patterns?

Questions to Think About:

1. What is the string corresponding to the smallest possible valid step pattern?
2. What is the string corresponding to the picture above?
3. Suppose we want to draw a pattern corresponding to **n** down steps followed by **n** up steps? How many characters will be in the corresponding string?

Try It!

1. If JFLAP is not already active, start JFLAP and click the **Grammar** button.
2. Enter the rule **S → Tr**.
3. Determine the appropriate productions with **T** on the left-hand side. Assume that you must have at least one step.
4. Save your grammar using a descriptive file name.
5. Verify that your grammar is a context-free grammar by selecting ***Test***. If it is not, fix it and resave.
6. Verify that your grammar generates strings corresponding to patterns with 1, 2, 3, and 4 down steps followed by the same number of up steps. Select ***Input > Brute Force Parse*** to do this.
7. Verify that your grammar does not generate strings corresponding to patterns with no down steps and no up steps.
8. Open the grammar file **UpAndDownSteps.CFG.jflap**. Your grammar should be similar.

Questions to Think About:

1. In some countries, people read and write right to left instead of left to right, as is done in the USA. Perhaps the steps would be drawn from right to left. What are the necessary changes in your grammar to draw the same patterns?
2. What would happen if you had forgotten the non-recursive rule with T on the left-hand side?

Let's try another pattern. Suppose that we wanted to draw an outline of a simple building (without windows). We would need to have the same number of **up** steps as **down** steps, and at least one of each. What would we have in between? We would have at least one **right**, but the number of **right** steps can be arbitrarily large (subject to real-world resource constraints, of course).

Try It!

1. If JFLAP is not already active, tart JFLAP and click the **Grammar** button.
2. Enter 2 rules to generate the basic outline. One will be recursive to make sure each **up** step has a corresponding **down** step. The other will have a new variable on the right-hand side, and it needs to ensure that there is at least one **up** step and at least one **down** step.
3. The next two rules will generate the width of the building. One of the rules will be recursive, and the other will make sure that there is at least one **right** step.
4. Save your grammar using a descriptive file name.
5. Verify that your grammar is a context-free grammar by selecting ***Test > Test for Grammar Type***. If it is not, fix it and resave.
6. Verify that your grammar generates strings corresponding to valid building patterns with heights up to 3 and widths up to 3. How may such strings are there?
7. Verify that your grammar does not generate a selection of strings corresponding to invalid

building patterns.

8. Open the grammar file **SimpleBuilding.CFG.jflap**. Your grammar should be similar.
9. Rename that start variable S to be another unused capital letter. Rename S wherever it occurs in your grammar.
10. Add two rules with S on the left-hand side to generate one or more simple buildings for a whole street. Make sure that two neighboring buildings are separated by a single **right** step. They will be packed close together. One of these rules will be recursive.
11. Save your updated grammar to another file with a descriptive name.
12. Verify that your grammar is a context-free grammar by selecting ***Test > Test for Grammar Type***. If it is not, fix it and resave.
13. Test your grammar as before to make sure it can still generate a single building.
14. Test your grammar as before to make sure it can generate two buildings.
15. Test your grammar as before to make sure it can generate three buildings.
16. Open the grammar file **SimpleStreet.CFG.jflap**. Your grammar should be similar.

A good challenge for you would be to develop some grammars for some other patterns, perhaps colors for tiling or musical notes. Have fun!