Additional Example to Practice with Regular Grammars
Martha Kosa

In the lesson, you learned the format for a regular grammar and developed a grammar for simple arithmetic expressions. Now you can practice with another example (or a few).

In your earlier schooling, you learned simple divisibility tests for numbers. For example, you learned that an integer is divisible by 5 if its last digit is a 0 or 5 and that an integer is even if its last digit is a 0, 2, 4, 6, or 8.

Let us first design a grammar to generate all strings of decimal digits such that the represented integer is divisible by 5.

Every grammar needs a start symbol/variable. The convention is to call it S. What are the special digits that we need to consider? They are **0** and **5**, because if the string ends with one of those two symbols, it is a valid string. So we need to treat the other digits, namely **1, 2, 3, 4, 6, 7, 8,** and **9** differently, because a valid string cannot end with those digits. We can repeat these digits as long as we want. So this suggests 8 recursive right-linear rules involving S. Remember that a **recursive** grammar rule has the same variable appearing on both the left and right-hand sides of the rule.

Try It!

Start JFLAP and click the **Grammar** button. Enter the 8 recursive rules discussed previously.

Questions to Think About:

1. As the grammar currently exists, can any terminal strings be generated? Why or why not?
2. With the current grammar, what happens if a **0** or **5** appears in a string?

Try It! (continued)

Now we need to deal with the **0** and **5** digits that must appear at the end of any valid string. We should have a new variable. You can call it by any letter that you wish, but a good mnemonic letter would be **E**. We need two new right-linear rules with S on the left-hand side. Enter these two rules.

Questions to Think About:

1. As the grammar currently exists, can any terminal strings be generated? Why or why not?
2. With the current grammar, how many times can a **0** or a **5** appear in a string?

Try It! (continued)

Now we need to finish the grammar. If a valid string must end in a 0 or 5, what is the effect of having multiple 0's and/or 5's? Add two recursive right-linear rules to handle all possibilities. Add one rule to properly stop the recursion. Remember that only one 0 or 5 is required at the end of any valid string. Save your grammar using a descriptive file name, and remember to save when you make changes.

Questions to Think About:

1. Can the string 123450500 be generated by your grammar? Why or why not?
2. Can the string 1234567895 be generated by your grammar? Why or why not?

Try It! (continued)

We are at the final step. You need to add 8 more right-regular rules to be finished to allow multiple clusters of digits that are neither 0's nor 5's. Save your grammar file with a descriptive name. Run some test strings. Check its type to make sure it is right-linear. Make sure only valid strings are generated by selecting ***Input > Generate Language*** and then entering 1 and pressing the **String Length** button. Do the same task for strings of lengths 2 and 3, respectively. What happens when you attempt to generate strings of length 4? Your grammar should look similar to the grammar from the file **DivisibleBy5.RG.jflap.**

As a challenge, modify your grammar so that **0** is the only string starting with **0** that can be generated.

You may have noticed that it might have been easier to design a left-linear grammar since we only want to generate strings that end with a **0** or a **5**. Design a left-linear grammar to generate the same set of strings that your original right-linear grammar does. As an additional challenge, modify your grammar so that **0** is the only string starting with **0** that can be generated.

Questions to Think About:

1. How would you modify your grammars to allow negative integers to be generated, too?
2. How would a regular grammar generating strings of decimal digits representing **even** numbers differ from the grammar you just designed?
3. How would a regular grammar generating strings of decimal digits representing **odd** numbers differ from the grammar generating even numbers?
4. An integer is divisible by 4 if the integer represented by its rightmost two digits is divisible by 4. Design a regular grammar generating strings of decimal digits representing numbers divisible by 4.
5. An integer is divisible by 3 if the sum of all its digits is divisible by 3. Design a regular grammar generating strings of decimal digits representing numbers divisible by 3. As a hint, review the Quotient-Remainder Theorem typically studied in discrete mathematics. Also, think about what happens to the value of a number when a digit is appended to its right-hand side. For example, what is the relationship between 123 and 1234?