Grammars and Finite-State Automata

The mathematics of computer science (informatics) is a rich source of problems for competitions at many levels. The aim of this paper is to provide a brief introduction to a few of these ideas.

Let's begin with a problem.

Problem 1: Consider the following method for creating strings of 1's. Begin with the symbol x, and apply the following replacement rules as many times as desired and in any order:

$$\begin{aligned} x &\mapsto 111x \quad (1)\\ x &\mapsto 11111x \quad (2)\\ x &\mapsto 111 \quad (3) \end{aligned}$$

The process ends when only ones remain. For example, we may obtain a string of 16 ones using rules 2, 1, 2, and 3, in that order:

$$x \overset{2}{\mapsto} 11111x \overset{1}{\mapsto} 11111111x \overset{2}{\mapsto} 1111111111111x \overset{3}{\mapsto} 1111111111111111.$$

How many strings of one or more 1's CANNOT be obtained by applying these rules?

(A) 4 (B) 6 (C) 8 (D) 10 (E) 15

Clearly, only strings of ones may be formed. The set of rules which generates these strings is called a **grammar,** while the set of all strings which may be created using these rules is called the **language** produced by that grammar.

The symbol "x" is called a **non-terminal symbol,** since as long as there is an x present anywhere in the string, we must continue applying rules until only 1's remain. Similarly, the symbol "1" is called a **terminal symbol.**

This informal definition of a grammar will suffice for this paper. Keep in mind that in writing contest problems, it is desirable to keep the problem statement as brief as possible. In addition, as the above problem shows, it

is possible to give the contestant an informal feel for what a grammar is without including details of a formal definition.

How does one approach Problem 1? Of course one may begin by simply writing out several strings in the language, looking for patterns. However, rules (1) and (2) are suggestive of the postage stamp problem: given an infinite supply of three- and five-cent stamps, what denominations of postage are possible?

It is well known that the maximum number of ones which cannot be produced by (1) and (2) only is $(3-1)(5-1)-1=7$, so that a quick inspection shows that only strings of 1, 2, 4, or 7 ones cannot be obtained by using (1) and (2). Since the process ends with an application of (3), strings of 4, 5, 7, or 10 ones cannot be obtained. Further, since we *must* end by using (3), the strings 1 and 11 are also impossible to generate.

Thus, only strings of 1, 2, 4, 5, 7, or 10 ones cannot be produced by using the above rules. Hence there are six such strings.

Of course it is not necessary to be familiar with the postage stamp problem in order to be able to solve this problem. But hopefully the solution to this problem serves as an illustration of the application of mathematical ideas to the analysis of languages produced by grammars.

One beautiful aspect of working with grammars is that they are very easy to write – simply jot down any set of rules which comes to mind, and explore the grammar produced. This is the easy part! Sometimes, it is difficult to determine whether the rules can produce any strings at all. By way of example, consider the following two rules:

$$\begin{array}{ll} a \mapsto 1a1a1 & (1) \\ 11a \mapsto 1 & (2) \end{array}$$

As before, we begin with the non-terminal a and apply rules until only 1's remain. A first thought is that it is not possible to ever eliminate all the non-terminal a's – since any time you used rule (1), the second a produced could never be removed because in order to do so, two 1's would need to precede it in order to use rule (2).

This turns out to be incorrect. A simple attempt at creating a string in the language yields

$$a \overset{1}{\mapsto} 1a1a1 \overset{1}{\mapsto} 11a1a11a1 \overset{2}{\mapsto} 11a11a1 \overset{2}{\mapsto} 111a1 \overset{2}{\mapsto} 111.$$

Thus, in applying rule (1) more than once, the additional 1's needed to use rule (2) are thereby produced.

So now the problem becomes more interesting! In general, rules which allow strings to shrink, such as rule (2), can allow for some very unusual behavior. What language is produced by this grammar? The curious reader may think for a few moments before continuing to the solution.

It should be clear that applying rule (1) increases the length of the string by 4, while applying rule (2) decreases the length of the string by 2. Since we begin with a – a string of length 1 – any string produced by this grammar must have odd length.

But we can never produce a single 1, since in order to do so, we would have to obtain the string $11a$ immediately before. It is clear that after the initial a, applying either of the rules results in a string ending in 1. Thus $11a$ can never be produced, and thus neither can a string consisting of a single 1.

Can strings of all odd lengths greater than one be obtained? It turns out that this is indeed possible. We use a standard proof technique here: derive additional rules which may be used to give the desired result. In particular, we will show that the following rules may be derived from the original two:

$$a \mapsto 111a1 \quad (3)$$
$$a \mapsto 11111a1 \quad (4)$$

Certainly rule (3) is easy to see, since we have already derived in on our way to producing 111 earlier. Rule (4) may be similarly derived by applying rule (1) three times, and then using rule (2) three times. Note that a fourth application of rule (2) produces the string 11111, and so suggests a pattern: to produce a string of $2n-1$ ones ($n > 1$), apply rule (1) n times, followed by rule (2) $n+1$ times.

The reader is welcome to attempt an induction proof using this observation – but it is not so easy to guarantee that when applying rule (2), there are *always* two 1's just preceding the occurrence of an a.

However, using rules (3) and (4) allow for a direct proof. For it is clear that applying rule (3) p times (where $p > 1$) followed by rule (2) results in a string of length $4p - 1$. Moreover, applying rule (4) once, rule (3) q times, and finally rule (2) results in a string of length

$$6 + 4q - 1 = 4q + 5.$$

Note that q may be 0 here, as we may use rule (2) immediately after rule (4). It is clear that allowing p and q to vary over their ranges will produce strings of ones of all odd lengths greater than 1.

Now there is no reason that we are limited to either one terminal symbol or one non-terminal symbol. The complexity of a problem can increase significantly with multiple terminal and non-terminal symbols.

Problem 2: Consider the following method for creating strings of 0's and 1's. Begin with the symbol x, and apply the following replacement rules as many times as desired and in any order.

$$\begin{array}{llll} x \mapsto 1 & (1) & y \mapsto 1x & (5) \\ x \mapsto 0x & (2) & z \mapsto 0 & (6) \\ x \mapsto 1y & (3) & z \mapsto 0y & (7) \\ y \mapsto 0z & (4) & z \mapsto 1z & (8) \end{array}$$

The process ends when only 0's and 1's remain. For example,

$$x \overset{3}{\mapsto} 1y \overset{4}{\mapsto} 10z \overset{8}{\mapsto} 101z \overset{6}{\mapsto} 1010.$$

Which of the following numbers, interpreted as a binary string, can be obtained using the above rules?

(A) 5^{2011} (B) 6^{2011} (C) 7^{2011} (D) 8^{2011} (E) 9^{2011}

How does one approach a problem like this? The key lies in examining what it means for a string to end in x, y, or z. The rules make sense if we interpret a string ending in x to mean "the string created so far, interpreted as a binary number, is congruent to 0 mod 3." Similarly, that a string ends in y means "the string created so far, interpreted as a binary number, is congruent to 1 mod 3," and that a string ends in z means "the string created so far, interpreted as a binary number, is congruent to 2 mod 3."

Each rule is consistent with this interpretation. For example, (5) indicates that if the string so far ends in y (the string is congruent to 1 mod 3), adding a 1 on the end multiplies the number by 2 and adds 1, leaving a string which is congruent to $2 \cdot 1 + 1 \equiv 0 \bmod 3$, so that it must now end in x.

Now notice that (3) and (7) leave the string ending in y, so that after applying one of these rules, the string so far may be interpreted as a binary number which is equivalent to 1 mod 3. Since (1) and (6) are similar to (3) and (7) except for adding the trailing "y," we see that when the last x, y, or z is replaced by a 0 or a 1, a binary number congruent to 1 mod 3 remains.

With the same interpretation of the above rules, it is not difficult to see that given a binary number congruent to 1 mod 3, a sequence of rules may be found which produce this number. For example, we obtain 10011 as follows:

$$x \overset{3}{\mapsto} 1y \overset{4}{\mapsto} 10z \overset{7}{\mapsto} 100y \overset{5}{\mapsto} 1001x \overset{1}{\mapsto} 10011.$$

It is not difficult to make the above arguments rigorous by using induction on the length of the binary strings produced. Thus, the rules produce precisely those binary strings which, when interpreted as binary numbers, are congruent to 1 mod 3. This immediately gives the correct answer to the multiple-choice question.

This solution technique, while differing substantially from the first, is typical in analyzing grammars. We may consider x, y and z as intermediate "states" in the construction of our string, and interpret rule (2) as meaning "after reading in a 0, we are in state x." This leads to the construction of what is called a **finite-state machine** for this grammar; however, given limitations of space, this topic cannot be addressed in any depth. Also closely related is a discussion of **regular expressions,** but again, the interested reader will necessarily need to explore that topic on his or her own.

It should be clear that number theory is playing a rather prominent role in our discussion so far. However, problems involving grammars allow for non-traditional and novel applications of number theory – so that as far as contest problems are concerned, something new may be added to the repertoire. The following is an example of an open-ended problem requiring a written solution. While not involving number theory in a deep way, the problem is certainly more involved than the postage stamp problem we began with.

Problem 3: Consider the following method for creating strings of 0's and 1's. Begin with the symbol x, and apply the following replacement rules as many times as desired and in any order.

$$\begin{array}{ll} x \mapsto 11x & (1) \\ x \mapsto y & (2) \\ y \mapsto y000 & (3) \\ 111y00 \mapsto y & (4) \\ y \mapsto 10 & (5) \end{array}$$

For example,

$$x \overset{1}{\mapsto} 11x \overset{2}{\mapsto} 11y \overset{3}{\mapsto} 11y000 \overset{5}{\mapsto} 1110000.$$

Describe all strings which can be produced using these rules.

Solution: Note that the strings accepted by this language consist of some number of 1's followed by some number of 0's. Given that a string must end with rule (5), there must be at least one of each symbol. We will show that given $m, n \geq 1$, the string of m 1's followed by n 0's belongs to this language.

Let R_i represent the number of times rule (i) is applied. Then it is evident that there are $2R_1 - 3R_4 + 1$ 1's in the string followed by $3R_3 - 2R_4 + 1$ 0's, with the "+1" terms coming from rule (5). Thus, given m and n, we must simultaneoulsly solve

$$2R_1 - 3R_4 + 1 = m, \quad 3R_3 - 2R_4 + 1 = n.$$

This system may be solved in terms of R_4, giving

$$R_1 = \frac{3R_4 + m - 1}{2}, \quad R_3 = \frac{2R_4 + n - 1}{3}.$$

By choosing R_4 to be a positive integer simultaneously satisfying

$$R_4 \equiv (m-1) \bmod 2, \quad R_4 \equiv (n-1) \bmod 3,$$

then R_1 and R_3 are seen to be positive integers. But of course this is always possible since 2 and 3 are relatively prime.

Note that since $m, n \geq 1$, we always have

$$R_1 \geq \frac{3}{2}R_4, \quad R_3 \geq \frac{2}{3}R_4,$$

so that there will always be enough 0's or 1's to apply Rule (4) as many times as is necessary – presuming that we begin by applying rule (1) R_1 times, then apply rule (2), and then apply rule (3) R_3 times.

Thus, the strings produced by these rules consist of those strings consisting of one or more 1's followed by one or more 0's.

Now for our last problem.

Problem 4: Consider the following method for creating strings of 0's and 1's. Begin with the symbol a, and apply the following replacement rules as many times as desired and in any order.

$$\begin{array}{ll} a \mapsto 1a1 & (1) \\ a \mapsto 11a & (2) \\ a \mapsto 1 & (3) \\ 11a1 \mapsto b0 & (4) \\ b \mapsto b0 & (5) \\ b000 \mapsto a & (6) \end{array}$$

The process ends when only 0's and 1's remain. For example, the string 11 may be obtained as follows:

$$a \overset{1}{\mapsto} 1a1 \overset{2}{\mapsto} 111a1 \overset{4}{\mapsto} 1b0 \overset{5}{\mapsto} 1b00 \overset{5}{\mapsto} 1b000 \overset{6}{\mapsto} 1a \overset{3}{\mapsto} 11.$$

Describe all binary strings obtained by these rules.

Solution: First, notice that it is not possible using these rules to produce a string beginning with a 0. To be completely rigorous, this may be established by a brief induction argument on the length of strings containing the symbols a, b, 0, and 1.

It is, however, possible to produce any string beginning with a 1. First note that it is possible to replace the symbol a with $a1$ as follows:

$$a \overset{1}{\mapsto} 1a1 \overset{1}{\mapsto} 11a11 \overset{4}{\mapsto} b01 \overset{5}{\mapsto} b001 \overset{5}{\mapsto} b0001 \overset{6}{\mapsto} a1.$$

It is also possible to replace the symbol a with $a0$, as follows:

$$a \overset{2}{\mapsto} 11a \overset{1}{\mapsto} 111a1 \overset{4}{\mapsto} 1b0 \overset{5}{\mapsto} 1b00 \overset{5}{\mapsto} 1b000 \overset{5}{\mapsto} 1b0000$$
$$\overset{6}{\mapsto} 1a0 \overset{1}{\mapsto} 11a10 \overset{4}{\mapsto} b00 \overset{5}{\mapsto} b000 \overset{5}{\mapsto} b0000 \overset{6}{\mapsto} a0.$$

Thus, we may repeatedly add either a 0 or 1 at the end of any string. A final application of (3) finishes the process and adds a 1 to the beginning of the string. Thus, any string beginning with a 1 can be produced. (Note that a single 1 may be produced just by an application of (3).)

Certainly no number theory is involved here, but it is not at all obvious what language is produced by this grammar just by glancing at the rules.

It is my hope that this brief introduction into grammars (and the related subjects of finite-state machines and regular expressions) gives the reader a feel for a class of interesting contest problems requiring a minimum of background knowledge. Moreover, while there are a few standard techniques for solving such problems, it should be evident that a slight change in the rules of a grammar might require entirely different solution methods. Finally, it bears repeating that it is very easy to generate open-ended questions regarding the language produced by a particular grammar. It is not so easy to create *interesting* problems.

As a final remark, there is a very famous puzzle closely related to grammars – the MU puzzle in Douglas Hofstadter's *Gödel, Escher, Bach.* Hofstadter weaves a discussion of this problem throughout his book, which is an excellent introduction to mathematical thinking, number theory, and logic. It

bears witness to the simple elegance which may be associated with problems involving the mathematics of computer science.

References.

Hofstaster, Douglas R., *Gödel, Escher, Bach: An Eternal Golden Braid,* Random House, 1980.

Hopcroft, Motwani, Ullman, *Introduction to Automata Theory, Languages and Computation,* Addison-Wesley, 2000.

Sipser, Michael, *Introduction to the Theory of Computation,* Wadsworth Publishing Co., 1997, 2005.