FINITE-STATE MACHINES

Frederick Gass
Miami University
Oxford, OH 45056

Modern algebra is a topic in which students benefit from early, informal exposure. If the idea and some examples of groups are mentioned some time during the high school years, then abstract algebra seems more familiar and concrete when encountered in college. The examples presented in this article belong to a new breed of algebraic system whose place in the mathematics curriculum is due largely to the rise of computer science. Because of their relationship with computers, these systems are often described as – and even called – "machines". Working with specific machines affords the same pleasure as solving puzzles, with the added challenge of composing precise descriptions. And just beyond that level lie questions that lead a capable student to more abstract reasoning and to consideration of the machine concept as a model.

Picture a newspaper-vending machine that requires 35 cents in the form of a dime and a quarter, inserted in either order. Its current *state* is always one of these:

S0: Locked, requires a quarter and a dime.
S1: Locked, requires only a quarter.
S2: Locked, requires only a dime.
S3: Unlocked.

S0 is called the *initial state*, because one is expected to begin there. The events capable of changing the state of the system are

Figure 1

a: insert a quarter.
b: insert a dime.
c: attempt to open and close door. (Don't forget your paper !)

All this information is conveniently pictured in the *state diagram* of Figure 2 and also in the table of Figure 3, which has less visual appeal but is reminiscent of the operation table for a group. (Actually, it's even more like the scalar multiplication table for a small vector space, since the two categories of state and event are something like the categories of vector and scalar.)

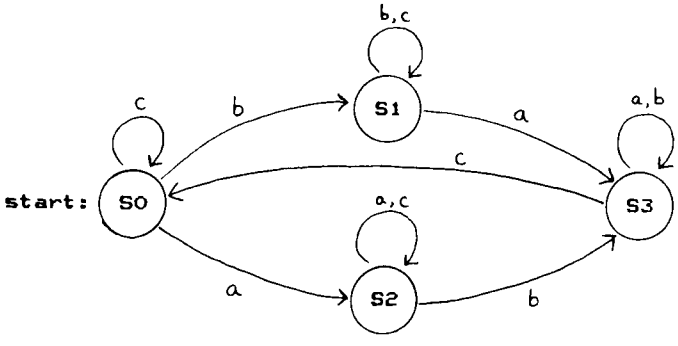

Figure 2

Strictly speaking, the finite-state machine in this example is embodied in the table of Figure 3 and is not really the physical vending machine itself. Without necessarily mentioning this philosophical point to a class, a teacher might invite the students to come up with another situation that gives rise to a diagram structured just like Figure 2; then comment that in a sense one has really arrived at the *same* finite-state machine as before.

	s0	s1	s2	s3
a	s2	s3	s2	s3
b	s1	s1	s3	s3
c	s0	s1	s2	s0

Figure 3

Gotta Dance

Most people at a ballroom dance seem to spend their nondancing time either socializing or watching intently from the edge of the floor, evidently trying to pick up some tips. One such couple operates like this: Whenever a foxtrot is played, they fall into conversation with friends; when a tango is played, they stop dancing to watch, or if not dancing, just continue to watch or socialize. All other numbers, they dance – waltz, rhumba, swing, samba, you name it! What has this got to do with finite-state machines? Look for clues in Figure 4 (a case of two left feet). It's fun to look for finite-state machines hidden around us.

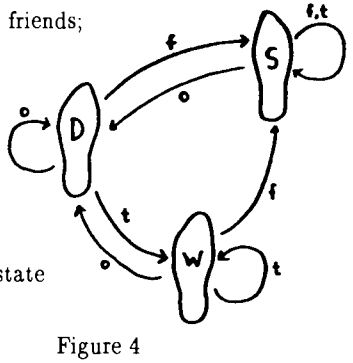

Figure 4

Computer Science

An example in the spirit of computer science is the parity-checker, which is to say a program that reads through a finite sequence of 1's and 0's to determine whether the number of 1's is even or odd. (Checks similar to this are part of programs that search for errors in coded information. This particular example is adapted from one in the interesting Computer Recreations column of *Scientific American* by Brian Hayes.) Let's suppose that an error-free sequence must be of even parity, so that a sequence is not acceptable unless it contains an even number of 1's. In the state diagram of Figure 5, notice how the occurrence of a 1 causes the machine to change states but a 0 does not. By toggling between EVEN and ODD

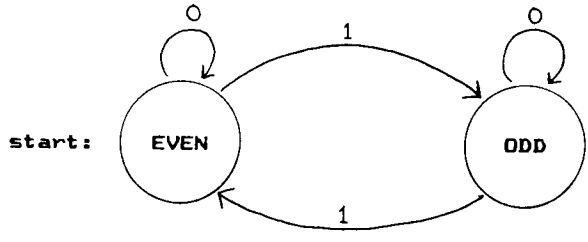

Figure 5

with each occurrence of a 1, the machine keeps track of the parity of the sequence up to the current point. EVEN is the starting state, because initially the checker will have read an even number (0) of 1's. It is also the one and only *accepting state*, because a sequence is acceptable if and only if the machine is in EVEN at the end of the sequence.

Finite-state machines that distinguish between acceptable and unacceptable sequences of symbols play an important role in the processing of computer programs by a compiler. The statements in a program are translated by the compiler program into a more elemental language that can be directly implemented by the computer. Before actual translation, however, part of the compiler acts as a finite-state machine that reads through each statement to decide whether it is grammatically acceptable.

Language Recognizers

If a finite-state machine M is intended to read finite sequences of symbols and identify the acceptable ones, then certain of its states are designated as accepting states, and the collection of all sequences that leave M in an accepting

state is called "the language recognized by M". (Incidentally, we shall assume that all sequences are nonempty.) In the next example and succeeding ones, we follow the convention of simply numbering the states 0 , 1 , 2 ... , understanding that 0 is the starting state and double-circled states are the accepting ones. Figure 6 is the state diagram of a machine M having three states, two of which are accepting. M reads sequences of x's and y's and recognizes a certain language that contains the sequence yxxyy, among others.

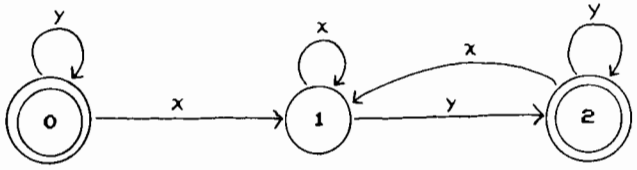

Figure 6

To check that claim about yxxyy, start in state 0 and read the sequence from left to right, using it as a set of directions to guide you among the states of M in Figure 6. The initial y leaves M in state 0 , the first x sends M to state 1, and second x leaves M in state 1 and the next y sends M to state 2. The final y leaves M in 2, which is an accepting state, and so the sequence yxxyy is accepted. As one can verify, the language recognized by M consists of all sequences that end with a y.

To illustrate a way of recording the progress of M as it reads a sequence, consider xyyyx as the input sequence. The idea is simply to record below each symbol, and slightly to the left of it, what state M is in when it encounters that symbol. Since the initial state is 0 , put "0" at the beginning of the state list. While in state 0, the machine reads the initial symbol x and switches to state 1. Keeping track of the computation so far, we have this:

sequence x y y y x
states 0 1

The rest of the computation creates this record:

sequence x y y y x
states 0 1 2 2 2 1 (final state not accepting)

Students enjoy the challenge of puzzling out the language recognized by a finite-state machine, and they like to create examples of their own. Alongside the

fun of discovery is the highly valuable exercise of writing a precise description of the recognized language. As an interesting example, determine the language accepted by the machine in Figure 7.

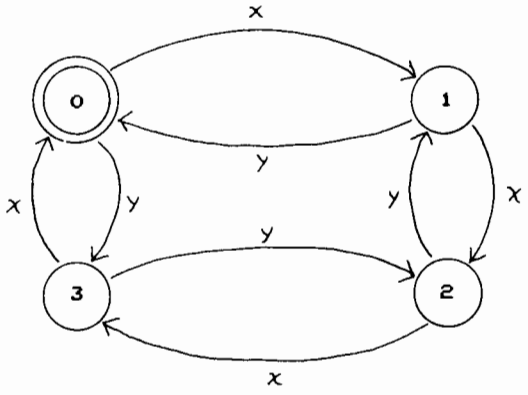

Figure 7

Reference

Hayes, B. "Computer Recreations, On the Finite-State Machine, a Minimal Model of Mousetraps, Ribosomes and the Human Soul". *Scientific American*, 249(6), 1983, 19–28.

Once you have the necessary items, plug the RS–232 connector on the back side of the Mini Modem 2400 into the RS–232C connector on your computer, printer or terminal, then screw up. – *Installation instructions for a modem made in Taiwan.*

Don't worry, we will!

From *The New Yorker*
October 28, 1991, p. 84