

# COMPUTER RECREATIONS

*A computer trap for the busy beaver, the hardest-working Turing machine*

by A. K. Dewdney

With the possible exception of bees, beavers are the busiest animals alive. All day they ply quiet northern waters bringing twigs and branches to their dam. It was undoubtedly this behavior that led Tibor Rado of Ohio State University to name a certain Turing-machine problem the Busy Beaver Game. In the early 1960's Rado wondered how many 1's a Turing machine could be made to print before it halted. Specifically, if a Turing machine with  $n$  possible states begins work on a tape filled with 0's, what is the largest number of 1's it can print on the tape before coming to a stop? The answer is known for  $n = 1$ ,  $n = 2$ ,  $n = 3$  and  $n = 4$  but not for  $n = 5$  or for any value of  $n$  greater than 5.

Last year a contest was held in Dortmund, West Germany, to see who could discover the busiest beaver with five states. In the year preceding the contest, programs were written to generate candidate Turing machines, and hardware was developed to test the machines. In the course of this work a number of strangely behaved beavers were discovered, and the genus *Castor* had to be expanded to include several species hitherto unknown to zoologists.

The nature of the Turing machine and its place in computer science have recently been discussed in these pages by John E. Hopcroft of Cornell University [see "Turing Machines," SCIENTIFIC AMERICAN, May]. A Turing machine consists of an infinite tape, a head for reading and writing symbols on the tape and a control unit with a finite number of internal states [see top illustration on opposite page]. These components can be thought of as the hardware part of the device, whereas the contents of the control unit are the software—the Turing-machine program. It is the program that distinguishes one Turing machine from another. The program is a table the machine consults to determine what action to take next. For each possible state of the control unit and for each possible symbol at the current position

of the tape head an entry in the table tells the machine what symbol to print on the tape, in which direction to move the head and what state to enter next. All the Turing machines discussed here begin in state 1.

The actions of a Turing machine can be traced by writing down the state of the control unit and the symbols marked on the tape (or a region of it) at successive moments; one should also indicate which square of the tape is currently being scanned. The bottom illustration on the opposite page is a trace of the Turing machine shown above it. Each line in the sequence is an "instantaneous description" of the machine. The format of the description is different from Hopcroft's, but the information is the same. I have also made the tape infinite in both directions, and I have allowed a symbol to be printed in the course of the machine's final transition (as it enters the halted state), contrary to the conventions adopted in Hopcroft's article. These differences do not change what a Turing machine can or cannot do. The format chosen here for the instantaneous description is compatible with the one used in the busy-beaver contest.

A busy beaver with  $n$  states is an  $n$ -state Turing machine that meets two conditions. First, when it is started on a tape filled with 0's, it eventually halts; second, it writes at least as many 1's as any other  $n$ -state machine that halts. Busy beavers with one and three states are shown in the top illustration on page 12. Each Turing machine is represented by a state-transition diagram, in which a state is a numbered circle and a transition between states is an arrow. The labels on the arrows describe the action of the Turing machine. For example, suppose the three-state busy beaver is in state 1 and it reads a 0 on the tape. The arrow followed under these circumstances is labeled "0,1,R" and leads to state 2. Hence the machine, having read a 0, writes a 1 on the tape, moves the head one square to the right and enters state 2.

The maximum number of 1's that can

be produced by an  $n$ -state Turing machine that halts is denoted  $\Sigma(n)$ . As is indicated above, the value of  $\Sigma(n)$  is known only for the first four values of  $n$ . The one-state busy beaver writes a single 1 before it halts; in other words,  $\Sigma(1)$  is equal to 1. A two-state busy beaver produces a sequence of four 1's. Can readers devise such a machine? A three-state busy beaver writes six 1's; one three-state beaver is the machine whose program and sequence of instantaneous descriptions are shown in the illustrations on the opposite page and whose state-transition diagram is given in the top illustration on page 12. The three-state beaver was discovered in 1962 by Rado and by Shen Lin of AT&T Bell Laboratories. In 1973 Bruno Weimann of the University of Bonn found a four-state busy beaver, whose output consists of 13 consecutive 1's. Since then theorists have been searching for a five-state busy beaver.

The busy-beaver contest was organized by Frank Wankmuller and held in January, 1983, at the University of Dortmund during a conference on theoretical computer science. Some 133 five-state Turing machines were entered. Uwe Schult of Hamburg won with a machine that produced 501 1's before halting. The state-transition diagram of the winning machine is shown in the bottom illustration on page 12. The runner-up was Jochen Ludewig of the Brown Boveri Research Center in Baden, whose Turing machine printed 240 1's.

Is Schult's Turing machine a busy beaver? Schult, along with Wankmuller and Ludewig, conjectures that it is. In other words, he suspects that no Turing machine with five states can produce more than 501 1's before halting. How could such a claim be proved? The answer lies in exhaustive search by computer, a search of the kind that Schult used to find his champion Turing machine in the first place. Before describing Schult's attempt to trap the five-state busy beaver in his computer, I should like to take a closer look at the function  $\Sigma(n)$  to get some insight into why the Busy Beaver Game is so hard to play, even with the aid of a computer.

The function  $\Sigma(n)$  has an extraordinary property: it is not computable. It simply grows too fast. From the first four values of  $\Sigma(n)$ —namely 1, 4, 6 and 13—it might seem that the rate of growth is only moderate. If 501 is indeed the maximum number of 1's for a five-state machine, the increase in  $\Sigma(n)$  would still appear to be no faster than that of an exponential function. Schult has found a six-state Turing machine that produces 2,075 1's, which again suggests a quite tractable rate of growth. On the other hand, Schult has also found a 12-state machine that generates so many 1's that the number must be ex-

pressed by the following mind-boggling formula:

$$4096^{4096^{4096^{6 \times 4096}}}$$

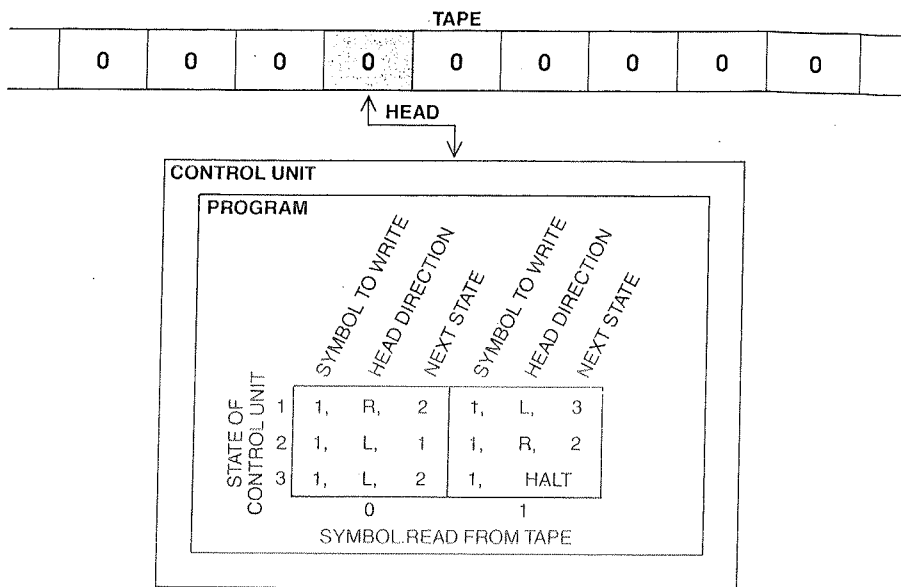
The number 4,096 appears 166 times in the formula, 162 times in the "twilight zone" represented by the three dots. The formula can be evaluated from the top down: first raise 4,096 to the fourth power, then raise 4,096 to the power of the resulting number, then raise 4,096 to the power of *that* number, and so on. When you reach the bottom, multiply by 6.

Anyone whose mind does not boggle when confronted by a string of 1's that long is welcome to construct an even bigger number. Write down any formula you like in which numbers are multiplied or raised to a power; you may even replace any of the numbers with *n*. No matter what formula you devise, for some value of *n* that is large enough the *n*-state busy beaver will produce more 1's than the formula specifies. It follows that  $\Sigma(n)$  cannot be calculated for arbitrarily large values of *n*. The best one can do is to calculate  $\Sigma(n)$  for some small, fixed value of *n*.

It is hardly surprising that the Busy Beaver Game is most often played with the aid of a computer. The essential method is to examine systematically all Turing machines with *n* states. Each time a new machine is generated its behavior on a tape filled with 0's is simulated. If the machine halts after no more than a specified number of steps, the number of 1's it printed is compared with the score of the "busiest" Turing machine found so far. From time to time a new champion is discovered.

This method of searching for the *n*-state busy beaver has two major flaws. First, the number of Turing machines to be generated is immense; for example, there are 63,403,380,965,376 five-state machines. Second, it is not known how long one should wait for a machine to halt; the maximum number of transitions an *n*-state machine can undergo (and still eventually halt), a function denoted  $s(n)$ , is itself a noncomputable number. Obviously  $s(n)$  grows even faster than  $\Sigma(n)$ , since a Turing machine must make a state transition each time it prints a 1. As Hopcroft pointed out, computing  $s(n)$  is equivalent to solving the halting problem for Turing machines, one of the first problems shown by Turing to be undecidable.

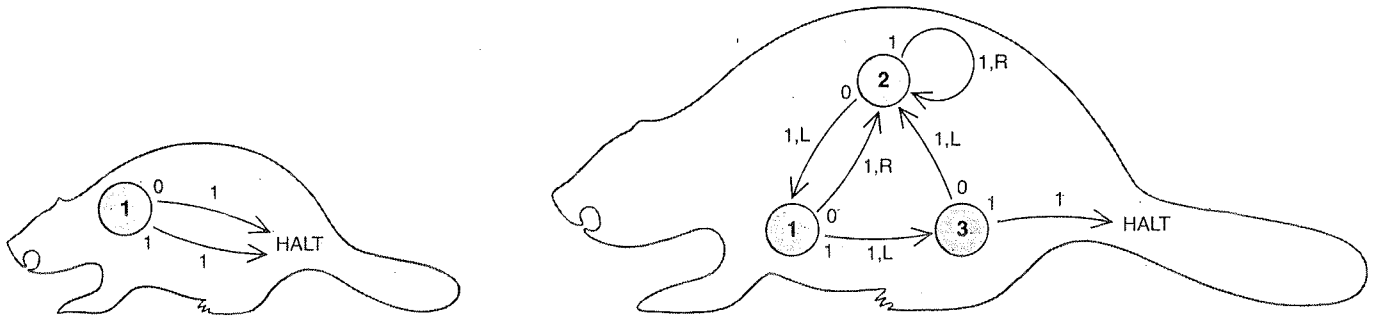
In 1982 Schult converted his Apple II personal computer into a busy-beaver trap. He augmented the computer's



A Turing machine and its program

STATE	TAPE								
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	1	0	0	0
3	0	0	0	0	1	1	0	0	0
2	0	0	0	1	1	1	0	0	0
1	0	0	1	1	1	1	0	0	0
2	0	1	1	1	1	1	0	0	0
2	0	1	1	1	1	1	0	0	0
2	0	1	1	1	1	1	0	0	0
2	0	1	1	1	1	1	0	0	0
1	0	1	1	1	1	1	1	0	0
3	0	1	1	1	1	1	1	0	0
HALT	0	1	1	1	1	1	1	0	0

"Instantaneous descriptions" trace the operation of the Turing machine



Busy beavers with one state and three states

original central processor with a circuit board bearing a Motorola 6809 microprocessor; he wrote his search program in the machine language of the auxiliary processor. To test the vast numbers of Turing machines generated by the program Schult built an actual hardware Turing machine out of standard electronic components mounted on another circuit board that plugs into the Apple II. The device provides a simulated tape of 4,096 squares as well as registers for storing the program and the current state and head position of the Turing machine. Schult estimates that without such specialized hardware his search would have taken 20 months of computer time. Even with the hardware extensions the Apple II took 803 hours to find the winning Turing machine.

In designing the necessary software Schult also gained by making the search program and the Turing-machine hardware interact closely. The program systematically filled in the transition table for a five-state Turing machine in all possible ways. Even before a table was completed it was submitted to the Turing-machine hardware for testing. In many cases an incomplete table was found to specify a machine that ran out of time or space before any of the undefined entries was reached. Thus the in-

complete table and all possible completions of it could be rejected.

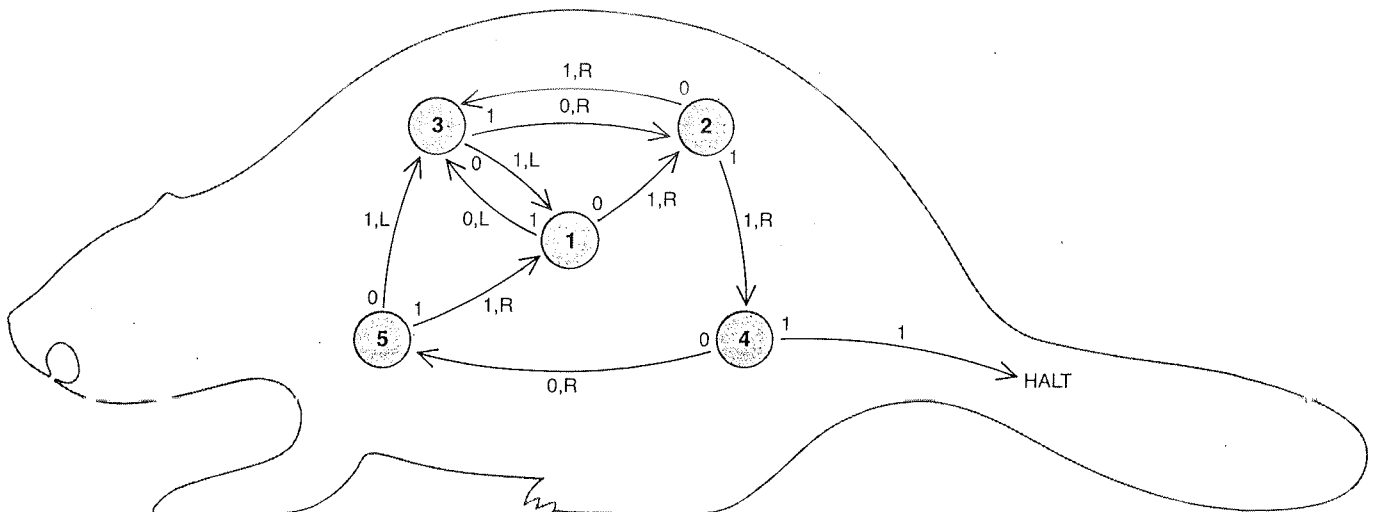
Although Schult in large measure overcame the problem of managing multitudes of Turing machines, his approach to the halting problem for five-state busy beavers is not watertight, so to speak. In the absence of exact information about  $s(5)$ —the maximum number of transitions a five-state halting Turing machine can make—the number must be guessed. Schult set the limit at 500,000 transitions; in other words, he adopted the working hypothesis that if a machine had not stopped after 500,000 transitions, it never would. Of necessity he also imposed space limitations on his candidate busy beavers; since the simulated tape had only 4,096 squares and since his Turing machines always started at the middle of this finite tape, a candidate was considered a "runner" if it moved more than 2,048 squares from its initial position. A runner is a Turing machine that not only fails to halt but also continues indefinitely to visit new tape squares.

Of the 133 Turing machines entered in the Dortmund contest, only four produced more than 100 1's. The operation of each Turing machine was simulated with a Siemens 7.748 computer.

More than an hour of processor time was needed to determine the winner.

Ludewig, the runner-up, wrote his busy-beaver search program in the Pascal programming language and ran it on a large minicomputer, the VAX, made by the Digital Equipment Corporation. In spite of a more sophisticated analysis of candidate Turing machines, 1,647 hours of central-processor time were spent in discovering his entry—the Turing machine that produced 240 1's. Schult, not surprisingly, also found Ludewig's machine; of equal interest, he found no machines between Ludewig's and his own. Apparently any halting five-state Turing machine that prints more than 240 1's must print at least 501.

Ludewig, in the course of his investigations, discovered a number of strange Turing machines with beaverlike behavior. Besides printing 1's there are other ways for a beaver to keep busy. For example, without printing many 1's a Turing machine may move a considerable distance from its starting square and then halt. Alternatively, without printing many 1's or even moving very far, it may go through a great many transitions before it halts. Among the machines tested at Dortmund, Schult's won in all three categories. On the other hand, Ludewig discovered three beavers that



Uwe Schult's candidate for a five-state busy beaver

generate no 1's at all but nonetheless either explore a wide territory or waste much time in profitless activity [see illustration below]. Accordingly three new species of beaver have been named:

*Castor ministerialis* (common name, civil-servant beaver). This enterprising creature seeks to advance itself as far as possible without producing anything.

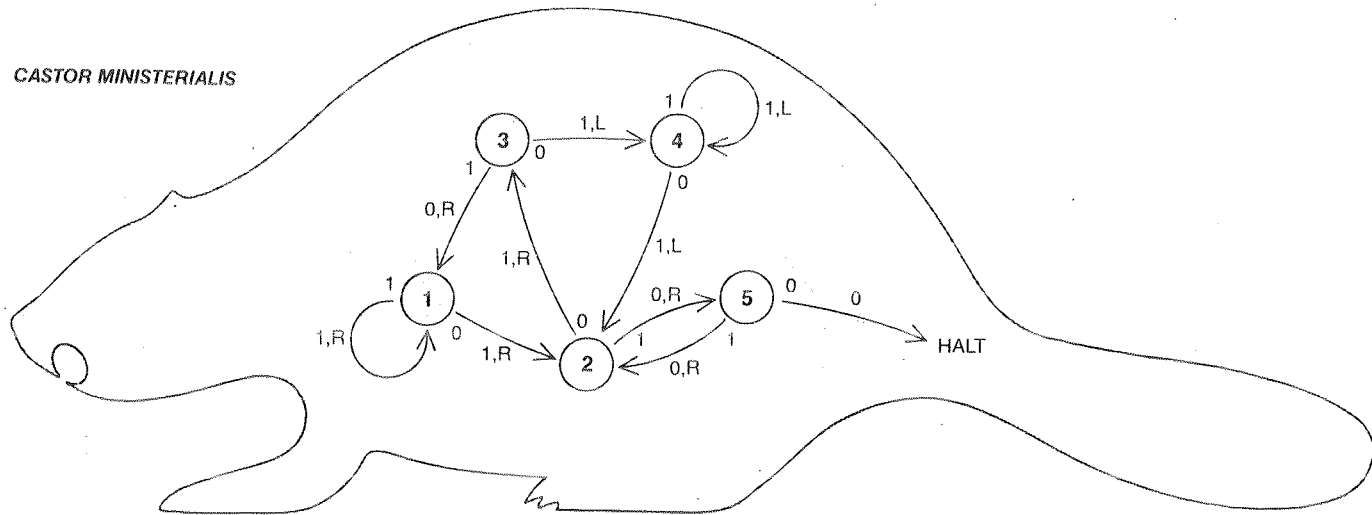
The type specimen is a five-state beaver that produces no 1's and moves 11 squares from its starting position.

*Castor scientificus* (common name, scientist beaver). Again without actually producing anything, this animal seeks to maximize its total activity, perhaps in an effort to attract grants. A five-state member of the species has been

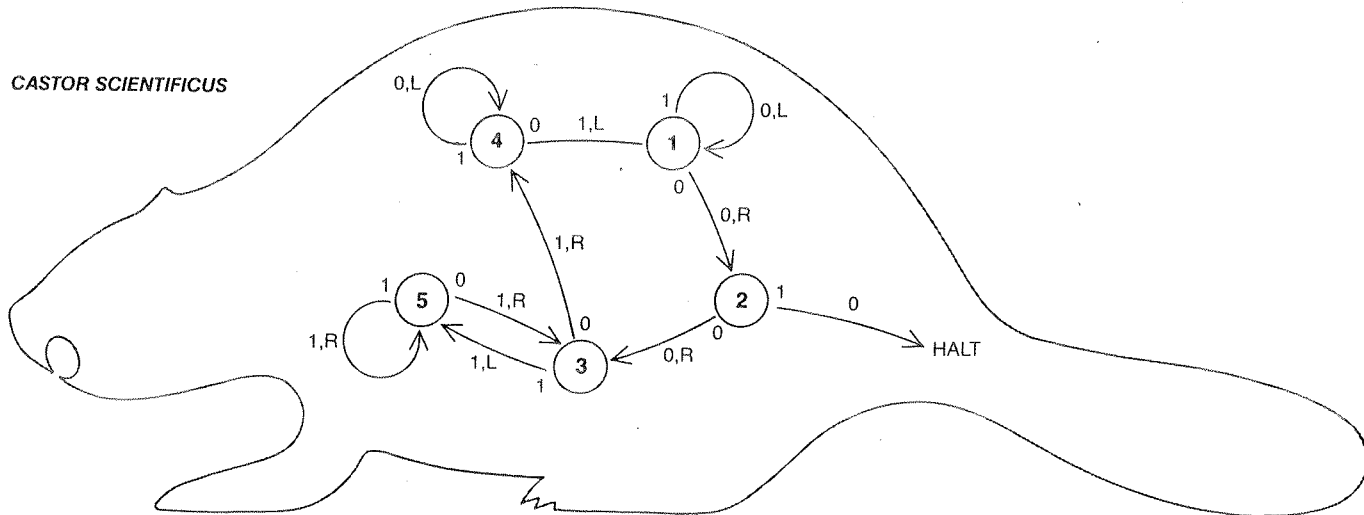
observed to make 187 transitions without writing a single 1.

*Castor circuitus* (common name, dizzy beaver). The dizzy beaver produces nothing and goes nowhere, but in the process it generates a maximum amount of activity. As the state-transition diagram suggests, it tends to spend a lot of time spinning its wheels. The busiest

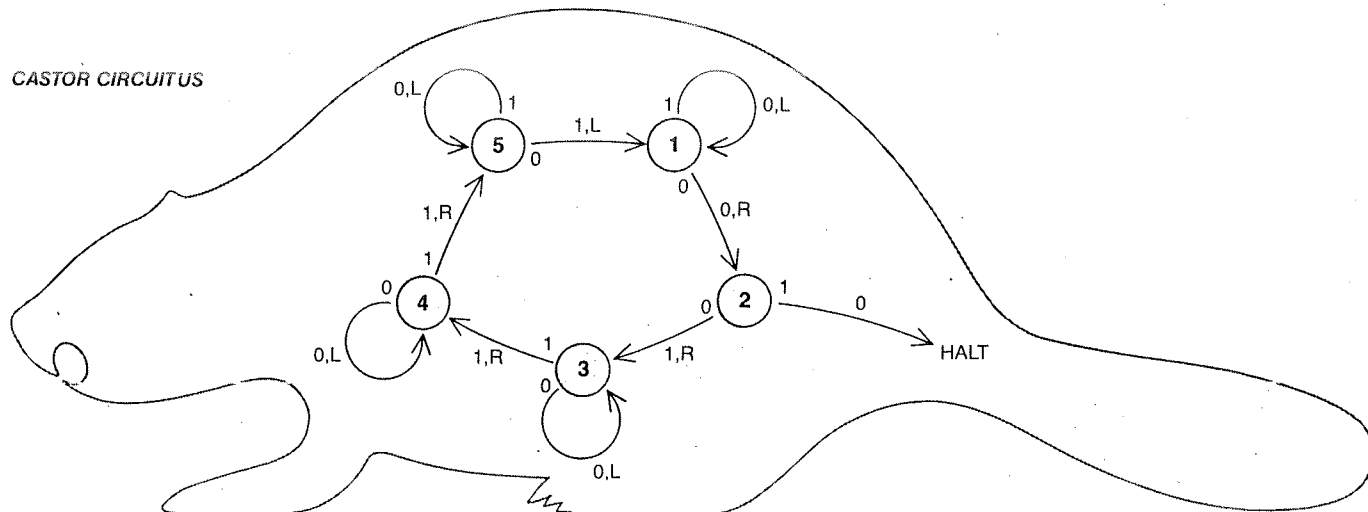
CASTOR MINISTERIALIS



CASTOR SCIENTIFICUS



CASTOR CIRCUITUS



Three new species of beaver that after much activity leave no 1's on the tape

five-state specimen found so far undergoes 67 transitions before it finally halts exactly where it started.

It would be interesting to see some three-state examples of these odd beavers. Any attempt to find them would certainly benefit from the use of a computer (personal or otherwise), even if only to test Turing-machine programs devised in one's mind.

A Turing-machine simulator is easy to write. Use a one-dimensional array to represent the tape; the contents of the array, which consist exclusively of 0's and 1's, can be shown on the computer's display screen. The display is most informative if the position of the head is indicated. For example, the machine's current state might be displayed directly below the symbol being scanned.

A two-dimensional array is needed to represent the Turing-machine program. Each element of the array is a set of instructions for the machine; instructions must be provided for each state of the control unit and for each possible tape symbol. For a three-state Turing machine the array has three rows and two columns; its structure is exactly that of the program shown in the top illustration on page 11. The state of the machine specifies a row in the array, and the symbol under the tape head specifies a column; the instructions found at the intersection of the designated row and column define the Turing machine's next action.

Suppose the machine is in state 1 and the symbol on the tape is a 0. Consulting row 1 and column 0 of the array, the simulator finds the instructions "1,R,2." Hence the machine is to write a 1 on the tape, move the head to the right one square and enter state 2. One way of implementing such instructions is to define three variables, say STATE, HEAD and SYMBOL. At the beginning of a cycle the values of STATE and SYMBOL determine where in the table the machine looks for its next instructions. The first component of the instruction found there (in this case a 1) is written on the tape; the second component (R) becomes the new value of HEAD, and the third component (2) becomes the value of STATE. The head is then moved (in the direction indicated by the value of HEAD) and the symbol found at the new position is made the value of SYMBOL. The cycle then begins anew.

Various strategies can be adopted to make the programming of such a scheme easier and more efficient. For example, the letters L and R can be replaced by numbers, which are generally easier to manipulate in the computer. Moreover, the transition that leads to the halted state demands special treatment in the program.

A Turing-machine simulator could be used to test your answers to the follow-

ing little puzzles, but it is by no means necessary to their solution.

Imagine you have bought a supply of used Turing-machine tapes at your local computer store. Before turning your busy beaver loose on them, the tapes must be cleaned up: any 1's on them must be changed back to 0's. Instead of cleaning the tapes yourself, you decide to devise a simple Turing machine to do the job for you.

One of the tapes has a single 1 on it but is otherwise filled with 0's. You must create a Turing machine that finds the 1, erases it (by changing it to a 0) and then halts. Naturally the fewer states your tape-cleaning machine has, the more elegant it will be. The tape cleaner in the illustration on this page is extremely elegant. Unfortunately it only works half of the time!

The remaining tapes are just like the first one except they have more 1's on them, although in each case the number of 1's is known to be finite. Can you construct a tape cleaner that changes all the 1's back to 0's? Of course, it will never halt.

Responses to the May column on Core War ranged from simple requests for the supplementary guidelines on the game to descriptions of complete Core War systems already in operation. In between were numerous anecdotes about Creeper-like programs inhabiting real systems (including worms in Apples), discussions of programs as genes and speculations about defensive and offensive strategy. Only a few important developments can be mentioned here; others will have to wait for a future column on the subject, which I hope will appear before the end of the year.

I have been told by Douglas B. McIlroy of AT&T Bell Laboratories that it was not he but Victor A. Vyssotsky of the same institution who invented the game Darwin. McIlroy did, however, invent an unkillable organism.

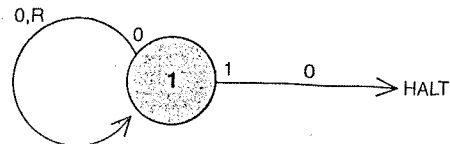
What happens when an Imp runs into a Dwarf? One possibility was explained in the May column: Dwarf transfers control to Imp's code and becomes a second Imp endlessly chasing the first one. Another possible outcome has the opposite effect. Suppose Dwarf has just jumped back to its first instruction when Imp copies itself over Dwarf's data location. The situation is then as follows:

```

Imp → MOV    0    1
Dwarf → ADD  #5  -1
        MOV  #0  @-2
        JMP  -2

```

Since it is Dwarf's turn to execute, it adds 5 to Imp's code, turning it into MOV 0 6. Then Imp executes, copying itself six spaces ahead, well clear of Dwarf, which then bombs its next address (specified by the numerical code



A tape-cleaning machine

corresponding to MOV 0 6). On Imp's next turn something curious happens: it executes the first line of Dwarf's program, so that for a time the game is played by a "double dwarf" pointlessly shooting up the core array while the object of its attack inhabits its own body and does exactly the same thing!

David Menconi of Milpitas, Calif., a game designer at Atari, Inc., has suggested making this very phenomenon a regular feature of Core War by allowing each battle program to execute in two places at once. Thus even if a program loses one "self," a second self might be able to repair the damage. Edsel Worrell of Bethesda, Md., suggests the somewhat more general scheme of  $n$  selves, all executing the same program at different addresses.

Robert Peraino of George Mason University wrote a Core War system for the Apple II+ computer, compensating for the machine's small word size by using a two-dimensional array of 2,000 by two bytes. Bill Dornfield of AMF, Inc., wrote a complete Core War system in extended BASIC on a Hewlett-Packard 9816/26 desktop computer.

The most impressive system to date was constructed by three graduate students: Gordon J. Goetsch and Michael L. Mauldin of Carnegie-Mellon University and Paul G. Milazzo of Rice University. Mauldin demonstrated the program on a VAX computer in my department at the University of Western Ontario. In an impressive screen display the entire core array is shown, with the position of each contending program marked by a capital letter and the areas affected by the program marked by the corresponding lowercase letter.

Mauldin has invented a battle program called Mortar that operates like Dwarf except that its bombs are directed according to the sequence of Fibonacci numbers (1, 1, 2, 3, 5 and so on, each number being the sum of its two predecessors). Oddly enough, Dwarf beats Mortar 60 percent of the time, but Mortar invariably kills a three-part self-repairing program called Voter. On the other hand, Voter survives attacks by Dwarf and regularly defeats it.

Goetsch, Mauldin and Milazzo have analyzed Mortar and conclude that if a battle program is longer than 10 instructions, it must be self-repairing in order to defeat Mortar. No program longer than 141 instructions, however, can repair itself fast enough to survive an attack by Mortar.

when we examine the Gray-code numbers beyond the 21st. Each of these represents a possible configuration in the rings puzzle, and the very last (corresponding to the binary number 11111) is 10000, the configuration in which only the last ring is on the loop. This implies that if you want someone to work harder at the rings puzzle, present it with all rings but the last one removed. Here the number of moves to solve the  $n$ -ring puzzle is  $2^n - 1$ , precisely as in the  $n$ -disk puzzle.

A very informative book on the Chinese rings has recently been written by Sydney N. Afriat, professor of economics and mathematics at the University of Ottawa. Called *The Ring of Linked Rings*, it is published by Gerald Duckworth & Co. Ltd., The Old Piano Factory, 43 Gloucester Crescent, London NW1, England. I am indebted to Afriat for the idea of tying the Chinese-rings puzzle to the yin and yang notion of duality. Although he suspects a Chinese origin for the Chinese-rings puzzle, he is aware of definite references only as far back as 1550. Afriat's book also describes the "Gros code," a 19th-century anticipation of the Gray code by the French mathematician Louis A. Gros, who published a treatise on the puzzle in 1872. The French, incidentally, call this puzzle Le Baguénodier and the English call it The Tiring Irons.

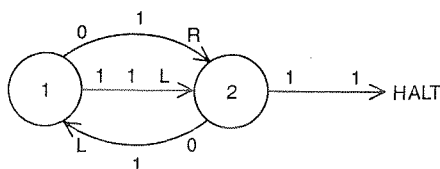
There are many other facets to both puzzles that I do not have the space to explore here. For example, Leroy J. Dickey of the department of pure mathematics at the University of Waterloo in Ontario reminded me that solving the Chinese-rings puzzle is also equivalent to traversing the edges of an  $n$ -dimensional hypercube.

It might be observed that the  $n$ -disk tower puzzle can be solved in  $2^n - 1$  moves if three pegs are used, whereas the use of  $n + 1$  pegs shortens the number of necessary moves to a considerably smaller number,  $2n - 1$ . What happens between three pegs and  $n + 1$  pegs? How does the minimum number of moves in a solution change as one changes the number of available pegs? It would be interesting to pursue such questions in a later column. Even as I write this there are people puzzling over the four-peg problem, as well as over what Martin Gardner calls a "fiendish version of the Tower," a Japanese puzzle that has been marketed under the name PANEX.

PANEX can be obtained (along with enough perplexity to while away any number of rainy afternoons) by writing to Tricks Limited, 71 Shinozakicho, 7-chome, Edogawa-ku, Tokyo 133, Japan.

The three busy-beaver puzzles posed in the August column were solved by Martin J. Maney of Palatine, Ill. His

two-state busy beaver is shown below. Starting with a blank tape, it produces four 1's before halting.



Maney's solutions are best summarized in words. A Turing machine that erases a single 1 from an otherwise blank tape uses two 1's as markers. At each stage it shuttles from one 1 to the other, checking just beyond each to see whether the 1 to be erased lies there. If it does, the machine erases all three 1's and halts. If it does not, the machine moves that marker one square outward and shuttles back to the other marker. The multiple 1's tape cleaner works similarly, except that it can never halt. How could it when some as yet unexplored region of the tape may contain a 1? Two-state busy beavers were found by Peter J. Marineau of Troy, N.Y., and Dave Kaplan of Deer Park, N.Y. Marineau also solved the tape-cleaner problem, describing his movable 1's as "brooms" that sweep the tape.

Raphael M. Robinson of Berkeley, Calif., wrote a Turing-machine simulation program for his IBM PC. As he watched Uwe Schult's conjectured busy beaver writing out its 501 1's, Robinson noticed that before halting it produced a recurring and ever lengthening pattern of alternating 0's and 1's. Starting with a blank tape, the successive lengths of this pattern were 0, 6, 13, 28, 48, 78, 121,

190, 289, 442 and 667. The last pattern contained 501 1's. It occurred to Robinson to investigate the behavior of Schult's machine, beginning not with a blank tape but with one of the alternating patterns. Starting on a pattern of length 9 (containing five 1's), the machine halted after 12,870,233 steps, having produced a new pattern containing 4,911 1's. This required three times the space and 25 times the number of steps performed by Schult's machine when started on a blank tape. That such a modest change in the input tape should produce such extravagant behavior disturbed Robinson. "It seems to me," writes Robinson, "that these results throw serious doubts on Schult's space and time restrictions."

Apparently Bruno Weimann of the University of Bonn was not the first to discover a four-state busy beaver. Allen H. Brady, now at the University of Nevada at Reno, discovered his own a decade before Weimann. At the time Brady was at Oregon State University. The school mascot is the beaver, and the computer in Brady's research was at nearby Beaverton. Brady shares Robinson's skepticism. "I know from solving the four-state problem that the five-state problem is far from decided. The crux of the matter is deciding that each alleged runaway machine will in fact never halt... As the machines become more complex this decision will become more and more difficult, eventually encompassing very profound unsolved mathematical problems... The blank tape halting problems of individual machines at some point become essentially individual mathematical theorems."

	BINARY CODE	GRAY CODE	BINARY CODE	GRAY CODE	
0	0 0 0 0 0	0 0 0 0 0	11	0 1 0 1 1	0 1 1 1 0
1	0 0 0 0 1	0 0 0 0 1	12	0 1 1 0 0	0 1 0 1 0
2	0 0 0 1 0	0 0 0 1 1	13	0 1 1 0 1	0 1 0 1 1
3	0 0 0 1 1	0 0 0 1 0	14	0 1 1 1 0	0 1 0 0 1
4	0 0 1 0 0	0 0 1 1 0	15	0 1 1 1 1	0 1 0 0 0
5	0 0 1 0 1	0 0 1 1 1	16	1 0 0 0 0	1 1 0 0 0
6	0 0 1 1 0	0 0 1 0 1	17	1 0 0 0 1	1 1 0 0 1
7	0 0 1 1 1	0 0 1 0 0	18	1 0 0 1 0	1 1 0 1 1
8	0 1 0 0 0	0 1 1 0 0	19	1 0 0 1 1	1 1 0 1 0
9	0 1 0 0 1	0 1 1 0 1	20	1 0 1 0 0	1 1 1 1 0
10	0 1 0 1 0	0 1 1 1 1	21	1 0 1 0 1	1 1 1 1 1

Each five-digit Gray-code number is obtained from its corresponding binary number by a simple rule: numbering the digits in left-to-right order, the first Gray-code digit is always the same as the first binary digit. Thereafter each Gray digit is a 1 if the corresponding binary digit differs from its successor; otherwise it is a 0.

Gray codes for the first 22 binary numbers

abundant oceanic plankton. Lemon made this feature explicit by placing plankton at every point not occupied by a shark or a fish. Plankton breed into otherwise empty spots and have the same relation to fish that fish have to sharks. Eternal populations exist here as well.

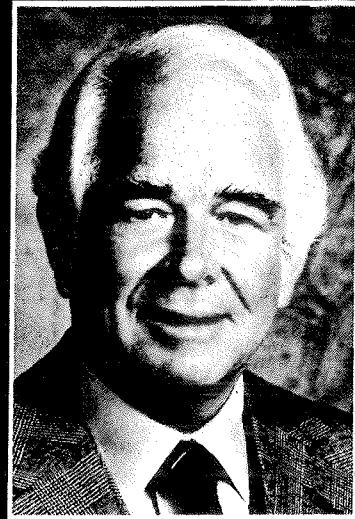
Couda's sharks gain or lose points of life-force depending on how well they eat. They can thus survive much longer without food than the primitive sharks of the standard Wa-Tor can. Couda sent plots (as did many of the other Wa-Tor programmers) that are remarkably similar to the Hudson's Bay Company data.

Connett uses two species of fish. One is the standard Wa-Tor variety; the other always breeds into any empty point to the south or east. Because of its mobile tendency, the second species often outlasts the first. Rudy Iwasko of Sacramento, Calif., proposed that sharks and fish be given characteristics of size, speed and agility. These were to be under genetic control. Berggren wrote his system, called EVOLVE, two years ago. It resembles WATER except that it lets the animals evolve according to environmental pressures. In this way, Berggren reasoned, populations would arrive at an equilibrium favoring long-term survival.

No one succeeded in solving the toroidal pursuit problem. I shall now reveal one half of the solution so as not to deny readers the pleasure of finding the other half. Remember that at each turn the fish moves and then the two sharks move. As in Wa-Tor, standing still is not allowed. Imagine four rays emanating from the lone fish. Each ray follows a diagonal and twists around the torus, sooner or later rejoining itself. Once both sharks occupy a pair of opposite rays, it does not matter which way the fish moves; one shark pursues at a constant distance and the other shark closes in. The fish is doomed. I leave it to readers to discover how sharks hunt the rays, so to speak.

A new candidate for the five-state busy beaver was discovered on December 21, 1984, by George Uhing of Bronx, N.Y. Uhing's Turing machine starts on a blank tape and prints 1,915 1's before halting. The result was independently confirmed by Allen H. Brady of the University of Nevada and by Raphael M. Robinson of the University of California at Berkeley. Described by Brady as "astounding," Uhing's machine seems to justify the skepticism both mathematicians had expressed that Uwe Schult's machine (reported in "Computer Recreations," August, 1984) was the five-state busy beaver. It produced only 501 1's.

Donald M. Kendall,  
Chairman of the Board  
and CEO, PepsiCo Inc.



## ASK YOUR RETIRED EXECUTIVES TO WORK HARD IN A STRANGE PLACE WITH NO PAY.

I'm a volunteer supporter of the International Executive Service Corps, a not-for-profit organization with a vital mission:

We send retired U.S. executives to help companies in developing countries. The executives are volunteers. We pay their expenses, but they receive no salary.

Our main purpose is to help developing countries succeed in business. But the benefit doesn't stop there. These countries consume about 40 percent of U.S. exports. So the work we do helps to create jobs and

earnings right here in America.

The International Executive Service Corps has completed 8,500 projects in 72 countries. I think you should seriously consider supporting this effort with funds and personnel. You would be in good company. Over 800 U.S. companies have supported us. Our Board of Directors and Advisory Council include the chief executive officers of many of America's most important corporations.

When you think about corporate giving, think about doing good business, as well as doing good.



### International Executive Service Corps

It's more than doing good. It's doing good business.



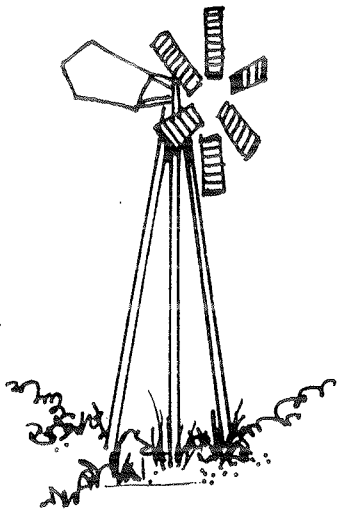
Join me in helping businesses in developing countries—by sending your retired executives through the International Executive Service Corps. For more information, write to Donald M. Kendall, Chairman of the Board and CEO, PepsiCo Inc., at 8 Stamford Forum, P.O. Box 10005, Stamford, CT 06904-2005. Or simply call this number: **(203) 967-6000.**

Name \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

# INVEST YOURSELF



A windmill to pump water for "salt farming" in India. More efficient woodburning stoves for the Sahel. Photovoltaic irrigation pumps for the Somali refugee camps

All these are solutions to technical problems in developing countries. Devising such solutions is no simple task. To apply the most advanced results of modern science to the problems of developing areas in a form that can be adopted by the people requires the skills of the best scientists, engineers, farmers, businessmen—people whose jobs may involve creating solid state systems or farming 1000 acres, but who can also design a solar still appropriate to Mauritania or an acacia-fueled methane digester for Nicaragua.

Such are the professionals who volunteer their spare time to Volunteers in Technical Assistance (VITA), a 20 year old private, non-profit organization dedicated to helping solve development problems for people world-wide.

Four thousand VITA Volunteers from 82 countries donate their expertise and time to respond to the over 2500 inquiries received annually. Volunteers also review technical documents, assist in writing VITA's publications and bulletins, serve on technical panels, and undertake short-term consultancies.

Past volunteer responses have resulted in new designs for solar hot water heaters and grain dryers, low-cost housing, the windmill shown above and many others. Join us in the challenge of developing even more innovative technologies for the future.

**VITA** Putting Resources to Work for People

1815 North Lynn Street, Arlington, Virginia 22209-2079, USA

digits of  $X$ ; the two digits determine how many times the algorithm will loop and which of the next 10 steps to jump to. Each of these 10 steps embodies a distinct method for calculating a new random number from an old one. It seemed plausible to Knuth before he tested his algorithm that "it would produce at least an infinite supply of unbelievably random numbers." To his astonishment, "when this algorithm was first put onto a computer, it almost immediately converged to the 10-digit value 6065038420, which—by an extraordinary coincidence—is transformed into itself by the algorithm." Knuth's moral is simple if not intuitively obvious: "Random numbers should not be generated with a method chosen at random. Some theory should be used."

If numerical methods for generating random numbers fail, one can always fall back on a suggestion once made by Alan M. Turing. Turing proposed that a random-number generator could be based on a source of radioactivity. Readers might enjoy the explorations of this idea in "The Computer Scientist: Random Numbers," by Forrest M. Mims III, in the November 1984 issue of *Computers and Electronics*.

Since the appearance of the January column about RACTER there have been more stories about the lettuce-craving program than there are by it. I have now conversed often enough with RACTER to understand that the program is even more seriously unbalanced than I had first thought.

John D. Owens of Staten Island, N.Y., distributes RACTER. He writes that one copy was ordered for the training of psychiatric interns who interview schizophrenic patients. Another customer was almost unhinged by RACTER. Describing RACTER as a "madman," this man failed in all attempts to halt the program. RACTER insisted on discussing the proposition that St. Peter was an atheist. I sympathize with the man. I too have run the gamut from laughter to boredom and even anger. But why react this way to RACTER? It is only a program.

To the Owens family RACTER is something more than a program. A customer lightheartedly wrote to ask whether RACTER could be "transmogrified" onto an eight-inch diskette. RACTER was consulted: "If 'We can transmogrify me onto eight-inch diskette' occurred to a pessimist, he might think it was pessimism." It later turned out that the Owens' facilities were incapable of the transmogrification.

On another occasion Terry Owens asked RACTER how much money to budget for advertising. "Very much

money because, Terry, people will believe it," RACTER replied.

At the end of the March column I mentioned a new candidate for the five-state busy beaver [see "Computer Recreations," *SCIENTIFIC AMERICAN*; August, 1984]. In December, George Uhing of Bronx, N.Y., found a five-state Turing machine that prints 1,915 1's before halting. The Uhing machine is reproduced in the table below.

To discover what the machine will do in state  $B$ , for example, examine the row bearing that label. The row is subdivided into an upper and a lower portion listing the machine's responses to a 0 or a 1 respectively. If the machine reads a 1 on its tape, it enters state  $D$ , prints a 0 on the tape and then moves one cell to the left. In the table  $H$  means that the machine halts.

Uhing, who programs for a Manhattan optical company, decided to search for the five-state busy beaver after reading this column last August. He used a Z-80 microprocessor running an assembly-language program to oversee a second machine: a Turing-machine simulator that cost Uhing less than \$100 to build. It goes through seven million Turing-machine transitions per second. Each transition amounts to a simple lookup in a table like the one below. Uhing seems determined to find the five-state busy beaver. Does the present machine qualify? It showed up after Uhing's computer had been running for a month. As far as I know it is still running.

Allen H. Brady of the University of Nevada at Reno describes Uhing's machine as "astounding." What are the chances we shall discover a six-state busy beaver? "Absolutely out of the question," Brady says.

Lee Sallows' computer-pangram challenge has now been met by several more readers. Those wishing to read Sallows' own stirring account of his analogue/digital pangram adventure, "In Quest of a Pangram," will find it in the spring issue of *Abacus*.

STATE	INPUT	NEXT STATE	OUTPUT	DIRECTION
A	0	B	1	RIGHT
	1	C	1	LEFT
B	0	A	0	LEFT
	1	D	0	LEFT
C	0	A	1	LEFT
	1	H	1	LEFT
D	0	B	1	LEFT
	1	E	1	RIGHT
E	0	D	0	RIGHT
	1	B	0	RIGHT

A five-state busy beaver?