particularly appropriate to point out that one advantage of approximation via iteration is that the accuracy of the approximation is limited only by the arithmetic of the machine doing the work. This is in marked contrast to polynomial or rational approximations, for example, which must usually be completely reconstructed in order to improve accuracy.

REFERENCES

1. Kendall E. Atkinson, An Introduction to Numerical Analysis, Wiley, New York, 1978.
2. J. Hart, et al., Computer Approximations, Wiley, New York, 1968.
3. Fortran Version 5 Common Library Mathematical Routines Reference Manual, Control Data Corporation, 1979.
4. Walter Gander, On Halley's iteration method, this MONTHLY (92) 131–134.
5. George H. Brown, Jr., On Halley's variation of Newton's method, this MONTHLY (84) 726–728.
6. J. M. Borwein and P. B. Borwein, The arithmetic-geometric mean and fast computation of elementary functions, *SIAM Rev.*, 26, 351–366.
7. ______, Pi and the AGM, Wiley-Interscience, New York, 1987.

There Are No Safe Virus Tests

WILLIAM F. DOWLING
Department of Mathematics and Computer Science, Drexel University, Philadelphia, PA 19104

This note gives a proof that no program can both test its input for the presence of a virus and simultaneously be guaranteed not to spread a virus itself. (You may define "virus" any way you please, as long as the definition is extensional.) This immediate corollary of Rice's Theorem [**1**] is proved by a direct diagonalization and offered as an antidote (not a vaccine) to boredom in the elementary computability course during the presentation of the halting problem.

Programs running on modern computers, unlike the executions of Turing machine programs, as usually conceived, run "in an environment." This is to say that when a program is executed, it is run as a subprogram of the logically independent program, the operating system, which is responsible for such bookkeeping chores as primary and secondary memory management, process management, recording statistics, and so on. A program that, when run, alters the code of the operating system, is called a *virus*. (When a new version of the operating system is written legitimately, it replaces, not alters, the former operating system.) This is a somewhat less restrictive definition of virus than others have proposed [**2**], in that no particular behavior is required of the modified operating system. For instance, it is frequently required that a virus have the effect of inserting its own code into other executable programs. Such restrictions are unnecessary for the result we seek.

It would be nice if we could detect automatically which programs are viruses and which are not by submitting them to a filter program, thus avoiding the expense and inconvenience of unwittingly and possibly harmfully altering our operating system. We now show there can be no program that does this correctly for every possible input, while guaranteed not to spread a virus itself.

We begin by fixing an operating system OS, and making a definition.

DEFINITION 1. *Program P spreads a virus on input x if running P under operating system OS on input x alters OS. Otherwise it is safe on input x. A program is safe if it is safe for all inputs.*

We also make the assumption that there exist viruses for OS, otherwise there would be no necessity for our test. Now for the sake of contradiction, let us assume there is some safe program *IS-SAFE* that decides the safety of running an arbitrary program P on arbitrary input x. Thus

$$IS\text{-}SAFE(P, x) = \begin{cases} \text{yes} & \text{if } P \text{ is safe on input } x \\ \text{no} & \text{otherwise.} \end{cases}$$

Given such a program and our assumption that there exist viruses, it is easy to write a program $D(\,)$ of one argument that has the following behavior:

$$D(P) = \begin{cases} \text{Write ``Have a nice day''} & \text{if } IS\text{-}SAFE(P, P) = \text{no} \\ \text{alter OS} & \text{otherwise.} \end{cases}$$

We can now show that *IS-SAFE* cannot be both safe and correct by examining the behavior of D on input D. If D is safe on input D this can only be because it has not executed the otherwise clause, that is, because $IS\text{-}SAFE(D, D) =$ "no," thus showing that *IS-SAFE* is not correct. On the other hand, if D alters OS on input D, there are two possibilities. On one hand, the call to $IS\text{-}SAFE(D, D)$ may be returning "yes" so the otherwise clause in D is being executed, in which case *IS-SAFE* is not correct. On the other hand, if *IS-SAFE* returns "no" (so D simply prints "Have a nice day") the assumption that D is unsafe on input D means that the call to *IS-SAFE* must be the culprit, that is, *IS-SAFE* is not safe. We conclude that the assumption of the existence of a safe, correct program that runs on *OS* and checks the safety of its input must be incorrect; there can be no such program *IS-SAFE*.

REFERENCES

1. H. Rogers, Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.
2. I. Witten, Computer (in)security: infiltrating open systems, *Abacus*, 4 (Summer 1987) 7–25.