"shortint," and 32767 replaced by 127, aborts at the statement $i := i + 1$, with an "out of range" run-time error. If the language modeled in the program verifier does not know all that, and cannot take account of it, that verifier will be relatively useless because many programming errors result from such language features.

Howard E. Tompkins
19 Congress Terrace
Milford, MA 01757

AN ERROR IN ERROR DETECTION?

I am writing to comment about a very common, but unfortunately widely repeated, error in the basic conception of coding theory, as recently found in Neal R. Wagner and Paul S. Putter's article, "Error Detecting Decimal Digits," (*Communications*, Jan. 1989, pp. 106–110). The erroneous statement (near the beginning of the article) is: ". . . the theory only works over a field."

In fact, error correcting schemes work for *any* integer greater than 2 by decomposing that integer into a product of prime products for different primes and then working separately over each prime. In the case treated by Wagner and Putter, we have the integer 10, which must be treated as 5 times 2. The coding schemes are to be worked out separately for 5 and for 2, and the resulting codes combined.

An example should suffice. We have the decimal number 5632078149 (say), to which a check digit is to be appended. The first step is to take each digit in the above decimal number and divide it by 5, collecting the quotients in one place and the remainders in another. In this case we obtain the base-2 number 1100011001 and the base-5 number 0132023144. (Thus, for example, 5, the *first* digit in 5632078149, divided by 5, gives quotient 1, the *first* digit in 1100011001, and remainder 0, the *first* digit in 0132023144. In general, the *K*th digit of the original number, divided by 5, gives the *K*th digit of each of the new numbers from the calculated quotient and remainder.)

Now apply whatever theory exists over a field of order 2 to obtain check digits for 1100011001. Suppose these check digits are 101. Now apply theory over a field of order 5, to obtain check digits for 0132023144. Suppose these check digits are 324. Now combine the two check digit sequences, 101 and 324, by the reverse of the process outlined above; that is, in this case,

$$101 \times 5 + 324 = 829$$

is the resulting check digit sequence, and, in the general case, the check digit sequence is $5b + q$ where b is the computed binary check digit sequence and q is the computed base-5 check digit sequence.

It should be obvious that if there is a transposition of two unequal digits, there will be a corresponding transposition in at least one of the two numbers obtained in this way. Thus, if the operator transposes the 5 and the 6, obtaining 6532078149, the resulting base-5 number will be different (1032023144), and specifically different by a transposition. The resulting check digit sequence q' will be different from q, and therefore $5b + q'$ will be different from $5b + q$.

On the other hand, if the operator transposes the 4 and the 9, obtaining 5632078194, then the resulting base-5 number will be the same, but the base-2 number will be different (1100011010), and again, different because of a transposition. If the resulting check bit sequence b' is different from b, then $5b' + q$ will be different from $5b + q$. This theory is immediately extensible to the case which we need x check bits and y check digits modulo 5, where $x \neq y$; these may be combined into a single decimal check integer from 0 through $2^x5^y - 1$.

W.D. Maurer
The George Washington University
School of Engineering and Applied Science
Washington, D.C. 20052

The article, "Error Detecting Decimal Digits," by N.R. Wagner and P.S. Putter (*Communications*, Jan. 1989, pp. 106–110) brought to mind the following scheme for generating a series of self-checking decimal numbers. We have used this method at Rhode Island Hospital for many years; I have not seen it described in the literature. The algorithm is . . .

Use only those numbers which are multiples of thirteen.

So, the series would begin with 0, 13, 26, 39, . . . , and include such sequences as 2405, 2418, 2431, 2444, . . . , 239343, 239356, 239369, etc.

As simple as this algorithm is, it can detect all single digit substitution errors, all adjacent and jump transpositions, and most (greater than 90 percent) of the other common kinds of errors: twins, jump twins, and dropped or added digits. Its ability to detect single-digit substitution errors follows from the fact that any such error alters the value of the number by an integer multiple of an integer power of ten, the prime factors of which cannot include 13. Similar considerations can validate its other error-detecting properties.

In addition to being trivially easy to generate and to check in any programming language or environment, a consecutive sequence of these self-checking numbers has *uniformly distributed terminal digits*, an important consideration when the associated records are physically arranged according to terminal digit(s).

A disadvantage of the scheme is that it cannot be used to append numeric check digits onto an existing set of numbers.

One useful generalization should be pointed out: the remainder mod 13 did not necessarily have to be defined as zero; *any* set of numbers having *identical* remainders mod 13 with have error-detecting properties. So there are actually *thirteen different interleaved sets* of self-checking numbers. An organization could use the 0-remainder set for employee numbers, the 1-

remainder set for customer numbers, the 2-remainder set for inventory numbers, and so on.

John C. Pezzullo
Center for Information Technology
Rhode Island Hospital
593 Eddy Street
Providence, RI 02903

AUTHORS' RESPONSE

We agree that Maurer's proposal would allow the power of coding theory to be applied to decimal error detection and correction. Our article was not primarily about coding theory, but about practical solutions to problems of detecting decimal errors. Maurer introduces a theoretical approach without giving any specific practical instance. In fact, his method does not make efficient use of decimal check digits because it catches or corrects base-5 and base-2 errors independently of one another, while actual errors occur in *decimal* digits.

Consider single decimal error correction. Maurer's method requires single-error correction separately for integers 5 and 2. If one uses the Hamming code over a field with q elements and with r check digits, then there are up to $(q^r - 1)/(q - 1)$ digits altogether. (See Hill's book [1] for a lucid discussion of these codes.) Two base-5 check digits yield only four data digits, so one needs at least three base-5 check digits for a practical application. Then one gets 31 digits and 28 data digits. By coincidence, five base-2 check digits will also give 31 digits, with 26 data digits. Combining these gives a code with up to 26 decimal data digits and with a check integer from 0 to $2^5 5^3 - 1 = 3999$. Now one sees a difficulty. If only four decimal digits are used for this check integer, then at least one of them must contain two of the base-2 check digits. Thus, without five decimal check digits there would exist uncorrectable single-decimal errors. In general, the base-2 part of Maurer's code would require a separate check decimal digit for each base-2 check bit. A more serious weakness is the correction of far more errors than just single decimal errors. Any pair of decimal errors where one error makes only a base-2 change and the other error makes only a base-5 change would also be corrected. Thus, the method corrects a number of special double errors, but only a fraction of all double errors.

If one is willing to use a decimal check integer, which itself is not completely protected against single decimal errors, then a better choice for single-error correction would be a Hamming base-11 code with three check digits (combined into a check integer from 0 to $11^3 - 1 = 1330$). This method would use the same four decimal check digits of the previous example and would yield up to 130 decimal data digits rather than 26.

Pezzullo presents an elegant and simple reformulation of a standard approach. His method is equivalent to appending a base-13 check digit on the right of an account number, using powers of ten as the weights in the check equation, and throwing out account numbers for which the check digit has value 10, 11, or 12 (about 23 percent). Thus, the sequence of all account numbers 0, 1, 2, 3, 4, 5, . . . would yield, with the appended base-13 check digit shown in parentheses: 0(0), 1(3), 2(6), 3(9), 4(12), 5(2), 6(5), 7(8), 8(11), 9(1), 10(4), 11(7), 12(10), 13(0), 14(3), ... Throwing out entries with a check digit greater than 9 yields just the successive multiples of 13 that Pezzullo uses. This method works with multiples of any prime greater than 10, though powers of 10 are not necessarily the best choice for the weights. For example, if one wants to detect as many jump transpositions as possible, successive integers for weights would be better.

Neal R. Wagner
University of Texas at San Antonio
Division of Mathematics, Computer Science, and Statistics
San Antonio, TX 78285

Paul S. Putter
AT&T Bell Laboratories
190 River Road
Summit, NJ 07901

REFERENCES

1. Hill, R. *A First Course in Coding Theory*. Clarendon Press, Oxford, 1986.
2. Wagner, N.R., and Putter, P.S. Error detecting decimal digits. *Commun. ACM 32*, 1 (Jan. 1989), 106–110.

A BRILLIANT EXAMPLE

My letter concerns the article, "A Sample of Brilliance" by Jon Bentley and Robert Floyd, which appeared in September *Communications*, 1987, pp. 754–757. I think that because Floyd has been a pioneer in the area of programs' verification, his algorithm F1 of page 755:

```
function Sample(M, N)
  if M = 0 then
      return the empty set
    else
      S := Sample(M − 1, N − 1)
      T := RandInt(1, N)
      if T is not in S then
          insert T in S
        else
          insert N in S
      return S
```

is well worth the proof. It should consist—in my opinion—of a sequence of obvious steps, making it easy to verify *evidence* of its conclusion. Unfortunately, the clarity of the reasoning quoted on page 756 leaves something to be desired. In particular, it was much easier for me to find more obvious evidence of the correctness of F1 than to check the proof presented in the paper. Here are the details.

The totality of F1 is obvious. For its correctness, it suffices to demonstrate that for each set U of M natural numbers not greater than N the probability that F1 will return U is equal to

$$\binom{N}{M}^{-1}.$$

Since F1 is evidently correct for $M = 0$, we will restrict the proof to the case of $M > 0$.

Let π be a deterministic procedure, ϕ be a formula, and $\diamond_\pi \phi$ be the weakest precondition for ϕ with respect to π (i.e., any initial state $\mathcal{S}$ satisfies $\diamond_\pi \phi$ iff π halts on $\mathcal{S}$ with a terminal state $\mathcal{T}$ satisfying ϕ). The probability