George Boolos, "A New Proof of the Godel Incompleteness Theorem,"
Notices of the American Mathematical Society, 36.4, April 1989, 388-390.

bells ring and cursors blink. The questions are: How can we find the resources to make computers work for all who could profit from them? Failing that, how do we weigh conflicting demands for computational resources among different kinds of users? And how do we teach today's users how to put the computers that are available to the best possible uses? These questions do not have easy answers, but until we answer them, many of us will be like teachers without books, or teachers who have books but are barely literate.*

This Month's Column

This month's column is an interesting potpourri of items variously related to computers and mathematics. It contains three reviews of mathematical software:

- a review by Raymond F. Smith of *MathView Professional*, a package of numerical routines for the Macintosh;
- a review by Gustaf Gripenberg of *MINPAC1-LIB*, a collection of FORTRAN routines for solving nonlinear systems of equations and nonlinear least-squares problems on the IBM-PC; and
- a review by Mark Sand of *ZG*, a freeware program for data analysis, also for the IBM-PC.

The column also contains two freeware offerings, a couple of letters reacting to previous articles in the column, and a very lovely proof of Gödel's Incompleteness Theorem, probably the deepest single result about the relationship between computers and mathematics, as well as having played an important (if slightly ironic) role in the development of computers, as I have discussed earlier. I am pleased to be able to include in this column the most straightforward proof of this result that I have ever seen.

If you have comments on or suggestions for this column, please get in touch. And if you have suggestions for software you would like to see reviewed, send me the name and address of the distributor.

Professor Jon Barwise
Center for the Study of Language and Information
Ventura Hall
Stanford University
Stanford, CA 94305

* The slant of this editorial was influenced by many conversations over the years with Brian C. Smith of Xerox PARC on the nature of computation.

A New Proof of the Gödel Incompleteness Theorem

*George Boolos**
Massachusetts Institute of Technology

Many theorems have many proofs. After having given the fundamental theorem of algebra its first rigorous proof, Gauss gave it three more; a number of others have since been found. The Pythagorean theorem, older and easier than the FTA, has hundreds of proofs by now. Is there a great theorem with only one proof?

In this note we shall give an easy new proof** of the Gödel Incompleteness Theorem in the following form: *There is no algorithm whose output contains all true statements of arithmetic and no false ones.* Our proof is quite different in character from the usual ones and presupposes only a slight acquaintance with formal mathematical logic. It is perfectly complete, except for a certain technical fact whose demonstration we will outline.

Our proof exploits *Berry's paradox*. In a number of writings Bertrand Russell attributed to G. G. Berry, a librarian at Oxford University, the paradox of *the least integer not nameable in fewer than nineteen syllables*. The paradox, of course, is that that integer has just been named in eighteen syllables. Of Berry's paradox, Russell once said, "It has the merit of not going outside finite numbers".***

Before we begin, we must say a word about algorithms and "statements of arithmetic", and about what "true" and "false" mean in the present context. Let's begin with "statements of arithmetic".

The *language of arithmetic* contains signs $+$ and $\times$ for addition and multiplication, a name 0 for zero, and a sign s for successor (plus-one). It also contains the equals sign $=$, as well as the usual logical signs $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (if ... then ...), $\leftrightarrow$ (... if and only if ...), $\forall$ (for all), and $\exists$ (for some), and parentheses. The variables of the language of arithmetic are the expressions $x, x', x'', \ldots$ built up from the symbols x and $'$: they are assumed to have

*George Boolos is Professor of Philosophy at MIT. His email address is Boolos@cogito.mit.edu.

** Saul Kripke has informed me that he noticed a proof somewhat similar to the present one in the early 1960s.

*** Bertrand Russell, "On 'Insolubilia' and their solution by symbolic logic," in Bertrand Russell, *Essays in Analysis*, ed. Douglas Lackey, George Braziller, New York, 1973, p. 210.

the natural numbers (0,1,2,...) as their values. We'll abbreviate variables by single letters: y, z, etc.

We now understand sufficiently well what truth and falsity mean in the language of arithmetic; for example, $\forall x \exists y x = sy$ is a *false* statement, because it's not the case that every natural number x is the successor of a natural number y. (Zero is a counterexample: it is not the successor of a *natural* number.) On the other hand, $\forall x \exists y (x = (y + y) \vee x = s(y + y))$ is a true statement: for every natural number x there is a natural number y such that either $x = 2y$ or $x = 2y + 1$. We also see that many notions can be expressed in the language of arithmetic, e.g., less-than: $x < y$ can be defined: $\exists z(sz + x) = y$ (for some natural number z, the successor of z plus x equals y). And, you now see that $\forall x \forall y[(ss0 \times (x \times x)) = (y \times y) \rightarrow x = 0]$ is – well, test yourself, is it true or false? (Big hint: $\sqrt{2}$ is irrational.)

For our purposes, it's not really necessary to be more formal than we have been about the syntax and semantics of the language of arithmetic.

By an *algorithm*, we mean a computational (automatic, effective, mechanical) procedure or routine of the usual sort, e.g., a program in a computer language like C, Basic, Lisp, ..., a Turing machine, register machine, Markov algorithm, ... a formal system like Peano or Robinson Arithmetic, ..., or whatever. We assume that an algorithm has an *output*, the set of things it "prints out" in the course of computation. (Of course an algorithm might have a *null* output.) If the algorithm is a formal system, then its output is just the set of statements that are provable in the system.

Although the language of arithmetic contains only the operation symbols $s, +$, and $\times$, it turns out that many statements of mathematics can be reformulated as statements in the language of arithmetic, including such famous unproved propositions as Fermat's last theorem, Goldbach's conjecture, the Riemann hypothesis, and the widely held belief that $P \neq NP$. Thus if there were an algorithm that printed out all and only the true statements of arithmetic—as Gödel's theorem tells us there is not—we would have a way of finding out whether each of these as yet unproved propositions is true or not, and indeed a way of finding out whether or not any statement that can be formulated as a statement S of arithmetic is true: start the algorithm, and simply wait to see which of S and its negation $\neg S$ the algorithm prints out. (It must eventually print out exactly one of S and $\neg S$ if it prints out all truths and no falsehoods, for, certainly, exactly one of S and $\neg S$ is true.) But alas, there is no worry that the algorithm might take too long to come up with an answer to a question that interests us, for there is, as we shall now show, no algorithm to do the job, not even an infeasibly slow one.

To show that there is no algorithm whose output contains all true statements of arithmetic and no false ones, we suppose that M is an algorithm whose output contains no false statements of arithmetic. We shall show how to find a true statement of arithmetic that is not in M's output, which will prove the theorem.

For any natural number n, we let $[n]$ be the expression consisting of 0 preceded by n successor symbols s. For example, [3] is $sss0$. Notice that the expression $[n]$ stands for the number n.

We need one further definition: we say that a formula $F(x)$ *names* the (natural) number n if the following statement is in the output of M: $\forall(F(x) \leftrightarrow x = [n])$. (Observe that the definition of 'names' contains a reference to the algorithm M.) Thus, for example, if $\forall x(x + x = ssss0 \leftrightarrow x = ss0)$ is in the output of M, then the formula $x + x = sss0$ names the number 2.

No formula can name two different numbers. For if both of $\forall x(F(x) \leftrightarrow x = [n])$ and $\forall x(F(x) \leftrightarrow x = [p])$ are true, then so are $\forall x(x = [n] \leftrightarrow x = [p])$ and $[n] = [p]$, and the number n must equal the number p. Moreover, for each number i, there are only finitely many different formulas that contain i symbols. (Since there are 16 primitive symbols of the language of arithmetic, there are at most 16^i formulas containing i symbols.) Thus for each i, there are only finitely many numbers named by formulas containing i symbols. For every m, then, only finitely many (indeed, $\leq 16^{m-1} + \cdots + 16^1 + 16^0$) numbers are named by formulas containing fewer than m symbols; some number is not named by any formula containing fewer than m symbols; and therefore there is a least number not named by any formula containing fewer than m symbols.

Let $C(x, z)$ be a formula of the language of arithmetic that says that x is a number that is named by some formula containing z symbols. The technical fact mentioned above that we need is that whatever sort of algorithm M may be, there is some such formula $C(x, z)$. We sketch the construction of $C(x, z)$ below, in 3).

Now let $B(x, y)$ be the formula $\exists z(z < y \wedge C(x, z))$. $B(x, y)$ says that x is named by some formula containing fewer than y symbols.

Let $A(x, y)$ be the formula $(\neg B(x, y) \wedge \forall a(a < x \rightarrow B(a, y)))$. $A(x, y)$ says that x is the least number not named by any formula containing fewer than y symbols.

Let k be the number of symbols in $A(x,y)$. $k > 3$.

Finally, let $F(x)$ be the formula $\exists y(y = ([10] \times [k]) \wedge A(x,y))$. $F(x)$ says x is the least number not named by any formula containing fewer than $10k$ symbols.

How many symbols does F contain? Well, [10] contains 11 symbols, $[k]$ contains $k + 1$, $A(x,y)$ contains k, and there are 12 others (since y is x'): so $2k + 24$ in all. Since $k > 3$, $2k + 24 < 10k$, and $F(x)$ contains fewer than $10k$ symbols.

We saw above that for every m, there is a least number not named by any formula containing fewer than m symbols. Let n be the least such number for $m = 10k$. Then n is not named by $F(x)$; in other words, $\forall x(F(x) \leftrightarrow x = [n])$ is not in the output of M.

But $\forall x(F(x) \leftrightarrow x = [n])$ is a true statement, since n *is* the least number not named by any formula containing fewer than $10k$ symbols! Thus we have found a true statement that is not in the output of M, namely, $\forall x(F(x) \leftrightarrow x = [n])$. Q.E.D.

Some comments about the proof:

1. In our proof, symbols are the "syllables", and just as 'nineteen' contains $2 << 19$ syllables, so the term $([10]x[k])$ contains $k + 15 << 10k$ symbols.
2. In his memoir of Kurt Gödel,* Georg Kreisel reports that Gödel attributed his success not so much to mathematical invention as to attention to philosophical distinctions. Gregory Chaitin once commented that one of his own incompleteness proofs resembled Berry's paradox rather than Epimenides' paradox of the liar ("What I am now saying is not true").** Chaitin's proofs make use of the notion of the *complexity* of a natural number, i.e., the minimum number of instructions in the machine table of any Turing machine that prints out that number, and of various information-theoretic notions. None of these notions are found in our proof, for which the remarks of Kreisel and Chaitin, which the author read at more or less the same time, provided the impetus.
3. Let us now sketch the construction of a formula $C(x,z)$ that says that x is a number named by a formula containing z symbols. The main points are that algorithms like M can be regarded as operating on "expressions", i.e., finite sequences of symbols; that, in a manner reminiscent of ASCII codes, symbols can be assigned code numbers (logicians often call these code numbers *Gödel* numbers); that certain tricks of number theory enable one to code expressions as numbers and operations on expressions as operations on the numbers that code them; and that these numerical operations can all be defined in terms of addition, multiplication, and the notions of logic. Discussion of symbols, expressions (and finite sequences of expressions, etc.) can therefore be coded in the language of arithmetic as discussion of the natural numbers that code them. To construct a formula saying that n is named by some formula containing i symbols, one writes a formula saying that there is a sequence of operations of the algorithm M (which operates on expressions) that generates the expression consisting of $\forall, x, (,$ the i symbols of some formula $F(x)$ of the language of arithmetic, $\leftrightarrow, x, =, n$ consecutive successor symbols s, 0, *and*). Gödel numbering and tricks of number theory then allow all such talk of symbols, sequences, and the operations of M to be coded into formulas of arithmetic.
4. Both our proof and the standard one make use of Gödel numbering. Moreover, the unprovable truths in our proof and in the standard one can both be seen as obtained by the substitution of a name for a number in a certain crucial formula. There is, however, an important distinction between the two proofs. In the usual proof, the number whose name is substituted is the code for the formula into which it is substituted; in ours it is the unique number of which the formula is *true*. In view of this distinction, it seems justified to say that our proof, unlike the usual one, does not involve *diagonalization*.

* Georg Kreisel, "Kurt Gödel, 28 April 1906–14 January 1978." *Biographical memoirs of Fellows of the Royal Society* 26 (1980), p. 150.

** Cf. Martin Davis, "What is a computation?" in *Mathematics Today*, ed. Lynn Arthur Steen, Vintage Books, New York, 1980, pp. 241–267, especially pp. 263–267, for an exposition of Chaitin's proof of incompleteness. Chaitin's observation is found in Chaitin, Gregory, "Computational complexity and Gödel's incompleteness theorem," (Abstract) *AMS Notices* 17 (1970), p. 672.

Correspondence

A Letter from Bob Fisch and David Griffeath

As the authors of *Graphical Aids for Stochastic Processes (GASP)*, we were delighted to see our software product reviewed in your column (February 1989). However, there is one point the reviewer brought up which we would like to clarify.

NOTICES
OF THE

AMERICAN MATHEMATICAL SOCIETY

ARTICLES

FEATURE COLUMNS

DEPARTMENTS