magnetization which may be mistaken for a clock pulse on playback. This is guaranteed during Pass 3 by magnetizing the clock track, after the last clock reversal of the leading address, in the erase direction and ensuring that an even number of reversals takes place up to and including the last clock reversal of the next leading address.

The block address itself consists of five reversals of magnetization on each clock track. Thus when the address of the first block is written its last reversal will leave the tape magnetized contrary to erase unless an odd number of reversals precedes the first address. For this reason an isolated reversal of magnetization occurs before the first block address, causing there to be six reversals of magnetization between the write current being first turned on at the beginning of Pass 3 whilst the tape is still on leader, and the region of tape between the leading address of the first block and the data.

It is also important to check for the presence of this isolated reversal, and facilities are provided for this during playback in Pass 4. Having provided a system for writing and checking an isolated reversal it is used in every interblock gap during Pass 3, and the write current is kept in the erase direction at all times except whilst writing the isolated reversal and the leading block address.

A final check of the direction of magnetization of every block is carried out in Pass 7 as data are written to each block in the conventional way.

Correspondence

An impossible program**

To the Editor,
The Computer Journal.

Sir,

I must admit that my previous acquaintance with *reductio ad absurdum* arguments has been confined to those which state the hypothesis which it is their intention to disprove; in the light of Mr. Lunnon and Mr. Outred's letter,* however, I now understand that Mr. Strachey intended his proof† to commence:

> Suppose that there exists a Boolean function, which we will call $T[R]$, taking a routine (or program) R with no formal or free variables as its argument, such that, for all R, $T[R]$ = **True** if R terminates if run and $T[R]$ = **False** if R does not terminate.

I was obtuse in taking Mr. Strachey's proof to consist of definition and counter-example; however, I feel that my refutations‡ require but little amendment to meet the *reductio ad absurdum* argument, and I hope that you will allow me to rephrase them in that context.

In all three of these arguments I equate "program" with "program capable of being run". My understanding is that Mr. Strachey would concur. If, however, he does not—if he considers that a set of instructions not capable of being run still qualifies for the title "program"—then I will immediately agree that $T[R]$ does not exist (in the sense of "is not defined") for all programs R; the proof is then trivial.

(i) I asserted that the function T exists by definition. Let us define it now.

> Let R be a program (with no formal or free variables). Either R terminates if run or it does not. Define $T[R]$ = **True** if R terminates if run; $T[R]$ = **False** if R does not terminate if run. $T[R]$ is, then, defined for all R.

It should be noted that for certain R the value of $T[R]$ is unknown—I agree with Mr. Higman* in lamenting the fact that we do not know $T[A]$, where A is the example program he gives. For other R the value of $T[R]$ may be unknowable—in the light of Gödel's Theorem it should be possible to prove this. But the fact that the value of a function is in some cases unknown or unknowable does not affect the question of the existence of the function.

(ii) I asserted that P is not a program. My original argument is certainly invalid in the light of the fact that Mr. Strachey's argument is a *reductio ad absurdum* one. May I replace it by another?

(iia) Consider the set of instructions P':

> **Set of instructions** P'
> §L: **if** Y **go to** L
> **Return** §

In working through this set of instructions we arrive at the question: what is the value of Y? Is Y **true** or is it **false**? If the value of Y is known, then we can insert this value and, so, proceed with the execution of the set of instructions. But if the value of Y can not be determined until after the execution of P' has been proceeded with beyond this point (and clearly P' can not be proceeded with until after the value of Y has been determined) then we are at an impasse: P' can not be executed, and, so, is not a program. A particular case occurs when Y is $T[P']$; in this case P' is P. P is not, then, a program.

(iii) I asserted that Mr. Strachey had drawn too restricted a conclusion from his argument. In any *reductio ad absurdum* argument one has not just one hypothesis but several (or, if you like, one compound hypothesis). Arrival at a contradiction by a valid argument demonstrates that (at least) one of the original hypotheses is false. But it does not indicate *which* of the original hypotheses is false.

In the case of Mr. Strachey's argument a second hypothesis is (implicitly) introduced: that P is a program. The contradiction at which he arrives by valid argument demonstrates that one of the hypotheses is false. I maintain that his conclusion can be no more than "either T does not exist or P is not a program".

In conclusion, may I apologize to you and Mr. Strachey for the style of my previous letter. At the time of writing it I could not entirely dismiss the suspicion that Mr. Strachey had his tongue in his cheek, and I was attempting to guard against being taken in by a hoax; but that does not excuse its somewhat impertinent tone.

Yours faithfully,

27 August 1965 H. G. APSIMON

* This Journal, July 1965, p. 175.
† *Ibid.*, January 1965, p. 313.
‡ *Ibid.*, April 1965, p. 72.

** *The first three of the letters that follow were received before the October issue went to press, but unfortunately they had to be held over for lack of space.* ED.

To the Editor,
The Computer Journal.

Sir,

I have just come across Strachey's letter (*The Computer Journal*, Jan. 1965; reprinted in *Computing Reviews*, July 1965) on the impossibility of writing a program which "can examine any other program and tell, in every case, whether it will terminate or get into a closed loop when it is run." The letter was of particular interest to me because I had, several months ago, proved that it is indeed possible to write such a program, at least in the case of finite memory. It may be of interest to compare my approach with Strachey's (and Prof. Turing's) to observe why the results are not in fact contradictory.

A computer with finite memory has a finite number of states b^m, where b is the number of values which each memory element can take (two, for a binary computer) and m is the number of memory elements. Let us say that a routine terminates if and only if it comes to an instruction which transfers to itself; i.e., does not change the state of the computer. Then a program terminates if and only if the computer eventually reaches a state such that it is the same as the next state. Specifically, let M be the memory of the computer, which is a finite set (including all registers and the location counter); let B be the set of values which each memory element can take ($B = [0, 1]$ for a binary computer), let $\mathbf{S}$ be the set of all maps $S: M \to B$, that is, all states (or instantaneous descriptions) of the computer, and let $I: \mathbf{S} \to \mathbf{S}$ be the map which determines, for each state of the computer (including the value in the location counter, of course) the next state of the computer. A program is now a particular state S of the computer. (A program may, of course, be represented by various states S, each of which has the same values in that subset of M in which the program is stored; but this point is not essential to the argument.) To determine whether the program S terminates, one simply calculates $I(S), I^2(S), \ldots$, until a power $I^{i+j}(S)$ is found which is equal to $I^i(S)$. The program S terminates if and only if $j = 1$. The various states $I^k(S)$ may be kept in a finite memory M' which is disjoint from M; the process will always terminate, since $\mathbf{S}$ is finite, and since each $I^k(S)$ has a finite representation, the memory M' may likewise be taken as finite. Thus the theorem is proved.

It is interesting to note that Strachey's disproof does not seem to involve memory; it is applicable to programs running in finite memory, and itself uses a finite procedure which does not use recursion or pushdown storage. The difficulty seems to be that what was actually proved above is the following: *Given any program in a finite memory M, there exists a program in a finite memory M'* (whose cardinality depends on that of M) *which will determine whether the original program terminates or not.* Strachey's arguments do not contradict this fact. If Strachey's program P is imbedded in M, and his $T(R)$ (which determines whether R, and in particular P, terminates or not) is imbedded in M', then P calls T, so that P is in fact imbedded in $U \cup M'$, and thus the conditions of the statement are violated. In general, M' must be much larger than M.

Sincerely yours,

W. D. MAURER

Room 813,
545 Technology Square,
Cambridge, Mass.
27 August 1965.

To the Editor,
The Computer Journal.

Sir,

The point made by Phillips and Irish on Strachey's proof is worth underlining.

The proof still admits of there being sets of routines other than the class of *all* routines for which the function *can* be written. It is easy to find particular cases (admittedly generally trivial) of such sets. For example the set of routines consisting of a sequence of assignment statements turned into a closed loop by a simple **go to.** Such a routine can arise from incorrect labelling or the accidental omission of a conditional branch instruction.

That such sets exist is hardly surprising, for determining whether or not a program gets into a closed loop is something programmers are doing every day. It would be very odd if some of the tests and intuition they use in doing this could not be turned into worthwhile compiler diagnostics. Writers of these in the world of practical application should not let Strachey's formidable piece of generality frighten them off!

Yours faithfully,

P. J. H. KING

University College of Wales,
Aberystwyth.
3 September 1965.

To the Editor,
The Computer Journal.

Sir,

Stripped of its technicalities, is not the essence of this question the same as Bertrand Russell's query about "the class of all classes which are not members of themselves"?

Many of your correspondents subsume the validity of *reductio ad absurdum* and of the law of the excluded middle on which it normally depends. Relevant to this is Brouwer's contention that the idea that a thing must have a certain property or not have a certain property is legitimate only when applied to *finite* sets.

Yours faithfully,

C. H. R. MORRIS

National Coal Board,
Ocean Buildings,
Bute Docks,
Cardiff.
22 October 1965.