Finiteness Assumptions and Intellectual
Isolation of Computer Scientists

by

Robert A. Wagner

Technical Report

No. 70-71

August 1970

Department of Computer Science
Cornell University
Upson Hall
Ithaca, New York 14850

Finiteness Assumptions and Intellectual
Isolation of Computer Scientists

by

Robert A. Wagner*

Abstract: We investigate the consequences of assuming integer variables of algorithmic languages to be finite vs. infinite in number and/or range. We suggest that different groups of computer scientists use different postulates about algorithmic languages. This leads to difficulty in communication, since the assumptions are usually unstated.

Key words and phrases: Algol vs. FØRTRAN, finiteness assumptions, intellectual isolation, integer variable range, memory finiteness, finite word size.

CR categories: 2.2, 4.22, 5.23

*Computer Science Department
Cornell University
Upson Hall
Ithaca, New York 14850

Finiteness Assumptions and Intellectual
Isolation of Computer Scientists

Thus far, the field of Computer Science has been comparatively cohesive; at least, it has usually been simple for workers in the area, e.g., of computational linguistics, to achieve communication with compiler writers. Unfortunately, this cohesiveness seems to be diminishing. Workers in different areas are adopting distinct, incompatible (and largely unstated) postulates to describe computer languages and programs. This trend threatens communication, and possibly the growth of the science, in a way that obscure notation and jargon cannot.

An illustration of the "fragmentation" trend described above occurred recently, in connection with an Admission to Candidacy Examination, given to all prospective Ph.D. students at Cornell before they begin work on their theses. As part of this oral examination, many candidates were asked the following question:

"Are there mathematical functions which are programmable in Algol, and not in FØRTRAN ?" Interestingly enough, at the moment this question apparently has three different answers!

"No - Automata Theory"

"Yes - Programming Languages"

"Maybe - any sophisticated user or implementor of compilers."

Let us examine the underlying, unstated assumptions which workers in each of these areas use to justify their respective stands.

First, and perhaps easiest, consider the Automata Theorist. Recently Automata Theorists have begun to pay more and more attention to theoretical questions concerning higher-level languages, rather than low-level machines, such as Turing machines. Such questions as "What is the shortest FØRTRAN program capable of computing an arbitrary constant?" have been studied. However, the automata theorist usually assumes that "variables" in any high level language may take on arbitrarily large values. Similarly, he ignores, as mere implementation restriction, limits on the number of characters allowable per programmer name, and limits on the number of characters allowable in expressing a constant. In this "theoretical" context, our first question must necessarily be answered in the negative: No Algol program can compute a function not also computable by a suitable FØRTRAN program. For the entire Algol program can be encoded (as by Godel numbering) into the value of one FØRTRAN variable, and a "Universal Algol Interpreter" can be written in FØRTRAN to simulate the Algol program-evaluation machinery. Thus, any Algol-programmed function can be computed by some FØRTRAN program - the "Universal Algol Interpreter."

Many students of programming languages would disagree with the automata theorist's interpretation of the formal descriptions of Algol and Standard FØRTRAN. The "standard," [1,2] for neither language is explicit about the question, although the Revised Report on Algol 60 appears to imply the existence of a finite limit on the precision of real quantities [1,3.3.6]. In any case, we are free to assume the existence of a finite limit on the precision (and hence on the maximum value) of a variable. This limit may differ in different implementations, but the mere existence of some finite upper bound on variable precision is enough to destroy the Automata Theorist's argument. For it will no longer be possible to write a Universal Algol Interpreter in FØRTRAN, and be assured that any particular implementation of FØRTRAN could handle all Algol programs. For any given FØRTRAN implementation, and any choice of encoding of the Algol program into one FØRTRAN variable, some Algol programs exist which encode into values exceeding the maximum legal variable value. So this argument breaks down.

Furthermore, we can show that under the assumption of a finite limit on variable values, FØRTRAN and Algol are not equivalent in power. For the FØRTRAN machine is inherently a finite state, while Algol is not. The difference stems from the requirement in FØRTRAN, that all array-sizes be constants at compile time, [2, 7.2.1.1], and [7.2.1.1.2]. Furthermore,

FØRTRAN does not permit recursion. Hence, at "compile time" (before the program has seen its input), a finite upper bound on the number of variables in the program must be established in FØRTRAN. Since each variable may take on only a finite number of distinct values, only a finite number of states are available to any specific FØRTRAN program.

Algol, although subject to similar restrictions on the size of any array (since the dimensions of an array are at least limited by the maximum expression value), has no restrictions on the number of times space may be allocated to that array or variable. Algol's recursion mechanism ensures this. Thus, since Algol programs are not "finite-state machines," while FØRTRAN programs are, there must be functions programable in Algol, which are not programable in FØRTRAN. One such is presented in Appendix A.

Let us now re-examine this question from the viewpoint of a user of one or both of these languages. Inherent in our previous discussion has been the assumption that computer memory (where both FØRTRAN and Algol variables reside), is infinite in capacity. Thus, we relied on the syntactic restrictions on FØRTRAN "dimension" statements to conclude that FØRTRAN programs must pre-specify a finite set of variables which they will use. Yet, if computer memory is finite in capacity, as it bears every evidence of being, Algol too must be considered finite-state. The question, from a language user's

standpoint, reduces to a question of exact sizes. It may be that, because FØRTRAN has less need for run-time space allocation mechanism, more memory locations are allocatable as variables in FØRTRAN than in Algol. This could imply that some FØRTRAN program was equivalent to a finite-state machine with more states than the machine equivalent to any Algol program. Hence, some FØRTRAN programs could compute functions not computable by any Algol program evaluable on the same computer. Obviously, the sophisticated language user must call for more information about specific machine and compiler implementation before concluding that Algol is more powerful in this sense than FØRTRAN.

We have attempted to demonstrate that unstated assumptions, in this case particularly about finiteness of various quantities, have made communication within the field of Computer Science difficult. We have investigated a question which appears to have three different answers depending on which "nasty facts" about computers you choose to ignore. We propose that, particularly when we educate new Computer Scientists, we make a conscious attempt to examine the particular abstraction from the real that we study, and relate our abstraction, if possible, to other potentially valuable abstractions, as well as to the real thing.

Appendix A

The following Algol program is not programable in FØRTRAN, assuming finite limits on the range and precision of all arithmetic variables and expressions.

begin

comment: The following simulates a push-down automaton to accept the set of expression $\{0^k 10^k \mid \text{all } k > 0\}$. This set is not regular [4], and hence, cannot be accepted by a finite-state machine;

comment: The string to be scanned is assumed to be on channel 1. The character "E" is assumed to follow it, marking its end. The machine signifies "acceptance" by placing the character "1" on channel 2. "Rejection" is signified by "0" on channel 2;

integer array M[0: 1,1:3];

integer S;

comment: M[state, input] defines the transition-function for the machine. S holds the number representing the machine's current state;

procedure Machine (X); value X; integer X; begin

switch L: = L1, L2, L3, L4;

comment: X holds the symbol at the top
of the stack;

LR: insymbol (1, '01E', I);
goto if I > 0 then L[M[S,I]] else L4;

L1: comment: stack a 1, remain in same state;
Machine (1);
goto LR;

```

L2: comment: enter state 1;
      S: = 1;
      goto LR;

L3: comment: if stack and input match, pop
      the stack, otherwise output "0" and
      halt;
      if X = I then goto L0;

L4: comment: output "0" and halt;
      outsymbol (2, '0', 1);
      goto R;

L0: end Machine;

```

comment: The body of the main program follows.
 It establishes the values of the transition
 function, which later directs control to a
 particular "state program" within procedure
 "Machine";

M[0,1]	:=1;	M[1,1]	:=3;
M[0,2]	:=2;	M[1,2]	:=4;
M[0,3]	:=3;	M[1,3]	:=5;

comment: Establish initial state and stack content;

S: = 0', Machine (3);

comment: On normal return, "Machine" has accepted;

outsymbol(2, '1', 1);

comment: On exit to label R, "Machine" has rejected, and
 has already placed a '0' on channel 2;

R: end;

REFERENCES

1. Naur, P. (Ed.) Revised report on the algorithmic language Algol 60. CACM vol. 6, number 1 (January 1963), pp. 1-17.
2. FØRTRAN vs. Basic FØRTRAN - programming language for information processing on automatic data processing systems. CACM vol. 7, number 10 (October 1964), pp. 591-624.
3. Wegner, P. Programming languages information structures and machine organization. McGraw - Hill, New York, 1970.
4. Kohavi, Z. Switching and finite automata theory. McGraw - Hill, New York, 1970.