PHILOSOPHICAL ASPECTS OF PROGRAM VERIFICATION

Not least among the fascinating issues confronted by computer science is the extent to which purely formal methods are sufficient to secure the goals of the discipline. There are those, such as C. A. R. Hoare and Edsger Dijkstra, who maintain that, in order to attain the standing of a science rather than of a craft, computer science should model itself after mathematics. Others, including Richard DeMillo, Richard Lipton, and Alan Perlis, however, deny that the goals of the discipline can be gained by means of purely formal methods.

Much of the debate between adherents to these diverse positions has revolved about the extent to which purely formal methods can provide a guarantee of computer system performance. Yet the ramifications of this dispute extend beyond the boundaries of the discipline itself. The deeper questions that lie beneath this controversy concern the paradigm most appropriate to computer science. The issue not only influences the way in which agencies disburse funding but also the way in which the public views this discipline.

Some of the most important issues that arise within this context concern questions of a philosophical character. These involve 'ontic' (or ontological) questions about the kinds of things computer and programs are as well as 'epistemic' (or epistemological) questions about the kind of knowledge we can possess about things of these kinds. They also involve questions about crucial differences between 'pure' and 'applied' mathematics and whether the performance of a system when it executes a program can be guaranteed.

The purpose of this essay is to explore the similarities and differences between mathematical proofs, scientific theories, and computer programs. The argument that emerges here suggests that, although they all have the character of syntactic entities, scientific theories and computer programs possess semantic significance not possessed by mathematical proofs. Moreover, the causal capabilities of computer programs distinguish them from scientific theories, especially with respect to the ways they can be tested.
1. THE FORMAL APPROACH TO PROGRAM VERIFICATION

The phrase 'program verification' occurs in two different senses, one of which is broad, the other narrow. In its broad sense, 'program verification' refers to any methods, techniques, or procedures that can be employed for the purpose of assessing software reliability. These methods include testing programs by attempting to execute them and constructing prototypes of the systems on which they are intended to be run in an attempt to discover possible errors, mistakes, or 'bugs' in those programs that need to be corrected.

In its narrow sense, 'program verification' refers specifically to formal methods, techniques, or procedures that can be employed for the same purpose, especially to 'proofs' of program correctness. This approach seeks to insure software reliability by utilizing the techniques of deductive logic and pure mathematics, where the lines that constitute the text of a program are subjected to formal scrutiny. This approach has inspired many members of the community (cf. Linger et al., 1979; Gries, 1979; and Berg et al., 1982).

Thus, while 'program verification' in its broad sense includes both formal and non-formal methods for evaluating reliability, in its narrow sense 'program verification' is restricted to formal methods exclusively. The use of these methods tends to be driven by the desire to put computer science on a sound footing by means of greater reliance on mathematics in order to "define transformations upon strings of symbols that constitute a program, the result of which will enable us to predict how a given computer would behave when under the control of that program" (Berg et al., 1982, p. 1).

The conception of programming as a mathematical activity has been eloquently championed by Hoare, among others, as the following reflects:

Computer programming is an exact science in that all of the properties of a program and all of the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning. (Hoare 1969, p. 576)

Thus, if this position is well-founded, programming ought to be viewed as a mathematical activity and computer science as a branch of mathematics. If it is not well-founded, however, some other paradigm may be required.