THE DIGITAL PHOENIX

How Computers Are Changing Philosophy

Edited by

TERRELL WARD BYNUM and JAMES H. MOOR

Published in cooperation with the Committee on Philosophy and Computers of the American Philosophical Association and also with the journal *Metaphilosophy*

BLACKWELL Publishers

© 1998

© Copyright Blackwell Publishers, Ltd and
The Metaphilosophy Foundation 1998

First published 1998
Revised edition 2000

Blackwell Publishers Ltd
108 Cowley Road
Oxford, OX4 1JF
UK

and

Blackwell Publishers Inc.
350 Main Street
Malden, MA 02148
USA

All rights reserved. Except for the quotation of short passages for the purposes of criticism and review, no part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of The Metaphilosophy Foundation and the publisher.

Except in the United States of America, this book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, resold, hired out or otherwise circulated without a similar condition including this condition being imposed on the subsequent purchaser.

British Library Cataloguing in Publication Data has been applied for.

Library of Congress Cataloguing-in-Publication Data has been applied for.

ISBN 0-631-203524

Typeset in 10 pt Book Antiqua

Printed in Great Britain by Whitstable Litho Ltd, Kent

CONTENTS

Rogers, Hartley. (1967) *Theory of Recursive Functions and Effective Computability*, New York, NY: McGraw-Hill.

Searle, John. (1992) *The Rediscovery of the Mind*, Cambridge, MA: MIT Press.

Siegelmann, Hava. (1995) "Computation Beyond the Turing Limit," *Science* 268: 545-548.

Siegelmann, Hava and E. D. Sontag (1994) "Analog Computation Via Neural Nets," *Theoretical Computer Science* 131: 331-360.

Turing, Alan. (1950) "On Computable Numbers with Applications to the Entscheidungsproblem," *Proceedings of the London Mathematical Society* 42: 230-265.

PHILOSOPHY AND COMPUTER SCIENCE: REFLECTIONS ON THE PROGRAM VERIFICATION DEBATE

JAMES H. FETZER

During 1986-87, four other philosophers and I participated in a special fifteen-month, post-doctoral program offered by Wright State University for Ph.D.s in linguistics and philosophy who wanted to study computer science and artificial intelligence.[1] The program extended over five quarters, with courses distributed over the first four and thesis work toward an M.S. degree the following summer. Among the courses for the fall quarter was one on programming languages, which was taught by Professor Al Sanders.

The course requirements included a term paper on one or more articles listed in the references for the course text by Michael Marcotty and Henry F. Ledgard, *Programming Language Landscape* [1986]. The most intriguing item I noticed was "Social Processes and Proofs of Theorems and Programs" by Richard DeMillo, Richard Lipton and Alan Perlis, which had appeared in *Communications of the ACM* in 1979. It had generated several letters and an authors' responses and looked as if it might be of philosophical interest.

When I located the paper itself and had the chance to read it through, I was fairly astonished. The authors were appraising the prospects for using formal methods to enhance our confidence in the reliability of software in computer systems. The advocates of this approach, who are an influential group within computer science, maintain that computer science ought to be modeled on mathematics as its paradigm, a conception that DeMillo, Lipton and Perlis on various grounds were intent to reject as an unattainable ideal.

The analogy embraced by advocates of formal methods takes the following form. In mathematics, proofs begin by identifying the propositions to be proven and proceed by deriving those propositions from premises ("axioms") as conclusions ("theorems") that follow from them by employing exclusively deductive reasoning. In computer science, proofs may begin by identifying the proposition to be proven (in this case, specifications of desired program performance), where deductive reasoning applied to the text of a program might show it satisfies those specifications and thereby prove it is "correct".

DeMillo, Lipton and Perlis sought to undermine the force of this analogy by discussing several respects in which "proofs" of mathematical theorems differ from "verifications" of program correctness. Their most important argument focused upon the role of social processes in evaluating proofs of theorems, where mathematicians consult other mathematicians to secure agreement that proofs are valid. In their view, the complexity of program verifications means that no comparable social processing ever takes place.

They found words to express what they wanted to convey in vivid and forceful prose. Supporting their position, for example, they remarked that, the verification of even a puny program can run into dozens of pages, and there's not a light moment or a spark of wit on any of those pages. Nobody is going to run into a friend's office with a program verification. Nobody is going to sketch a verification out on a paper napkin. Nobody is going to buttonhole a colleague into listening to a verification. Nobody is ever going to read it. One can feel one's eyes glaze over at the thought. [DeMillo, Lipton, and Perlis 1979, 276] Thus, according to DeMillo, Lipton and Perlis, the absence of social mechanisms in the program-verification community parallel to those found in the theorem-proving community destroys the comparison with mathematics.

As a student in search of a thesis, I had stumbled upon what appeared to be a philosopher's bonanza. While the authors described themselves as appraising the prospects for using formal methods to enhance confidence in the reliability of software, I sensed the aim of program verification was to guarantee the performance of programs when they are executed by machine. Having come of age intellectually in the logical empiricist tradition, I was confident that formal methods alone could not attain that objective.

Since DeMillo, Lipton and Perlis were not very specific on this point, I had some homework to do, which led me to publications by C. A. R. Hoare of Oxford, who was among the leading figures in the program verification movement. Imagine my feelings at finding the following passage in which Hoare had articulated what I had conjectured to be the implicit conception:

> Computer programming is an exact science in that all the properties of a program and all 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, 576]

It was immediately apparent to me that this conception, which asserts that purely formal methods can guarantee the performance of a computer when it executes a program, implies the existence of synthetic *a priori* knowledge.

Although I was already convinced that the program verification movement was predicated upon a misconception about the scope and limits of formal methods, I was not inclined to argue the case in my paper by denying the existence of synthetic *a priori* knowledge. An approach of this kind would have compelled the introduction of the analytic/synthetic distinction within this context, even though many philosophers reject this framework. And while I was confident that their reasons for its rejection were not well-founded, I was reluctant to base my critique upon such a disputed premise.[2]

As a consequence, I introduced other distinctions that I suspected would be easier to convey to computer scientists and that could not be derailed on the basis of purely philosophical concerns. I therefore argued that a distinction had to be drawn between *algorithms* as effective solutions to problems and *programs* as causal models of those algorithms, where the latter but not the former possess the capacity to exercise causal influence over computers when they execute a program. I emphasized the differences between pure and applied mathematics and between abstract models and causal systems.

Computer programming, of course, is ordinarily conducted by means of (what are called) *high-level* programming languages, such as Pascal, LISP, and Prolog, where there is a one-to-many relationship between commands in programs and instructions executed by a machine. Assemby language, by comparison, provides a *low-level* language, where something closer to a one-to-one relationship between commands obtains. Digital machines operate on the basis of strings of zeros and ones (or of high and low voltage), which would be difficult if not impossible to program directly. The causal connection between programs in high-level languages and target machines is therefore effected by interpreters and compilers, which translate them.

This means that programs are ordinarily written for *virtual machines*, which may or may not have physical counterparts. A mini-language CORE introduced as a pedagogical device by Marcotty and Ledgard [1986, Ch. 2] to illustrate the elements of programming languages but for which there is no interpreter or compiler afforded a perfect illustration, because "proofs" of program correctness in CORE could be constructed in relation to virtual machines that were guaranteed to execute them as definitional properties of those machines. These machines are abstract models of CORE machines.

Mistakes could still be made in programming, of course. Marcotty and Ledgard [1986, pp. 45-46] identified various sources of error, such as undefined value errors, overflow errors, negative value errors, and so forth, which could lead to the abnormal termination of a program. My point was not that programs written in CORE could never be imperfect, but rather that the performance of computers executing pro

grams written in CORE could be guaranteed in the sense that, assuming there were no programming errors, there could be no possible failure of a machine to execute a CORE program.

The crucial difference thus becomes that between virtual machines for which there are no physical counterparts and virtual machines for which there are physical counterparts. *Virtual machines* only exist as abstract entities beyond space/time for which definitional properties but no causal relations can obtain. *Physical machines*, by comparison, are causal systems in space/time, which can exert causal influence upon other things in space/time. While it may be possible to prove the correctness of a program using purely deductive reasoning, I urged, those formal methods cannot possibly guarantee what will happen when a target machine executes that program.

The analysis, in other words, was an elaboration of the epistemological ramifications that accompany the ontological difference between abstract entities and causal systems. Thus, I argued that the conclusive verification of a virtual machine was logically possible when it had no physical counterpart, because its behavior could be definitively established on the basis of purely deductive reasoning from stipulated axioms. The behavior of a target machine, however, might deviate arbitrarily from those axioms and, as a consequence, could never be definitively established by purely deductive reasoning. The behavior of causal systems must be established inductively.

The term paper I submitted thus suggested that, in contending that program verification could not guarantee what happens when a computer executes a program, DeMillo, Lipton and Perlis had arrived at the right general conclusion but for the wrong specific reasons. The problem was not rooted in the social processing of proofs but in the causal character of computers. When Al Sanders subsequently returned our papers, a lively discussion ensued, during which he reported that he had found mine to be "fascinating!" This enthusiastic reception moved me to send it to *Communications of the ACM*, which had published the original paper by DeMillo, Lipton and Perlis.

The paper was submitted 26 November 1986 with the title, "Social Processes and Causal Models of Logical Structures", which led James Maurer, the Executive Editor, to send it to Rob Kling, who was the area editor for social aspects of computing. Over the next 18 months, he would ask me to revise it four times in order to insure that my arguments would be accessible to readers of the magazine. Having published a 500-page book manuscript without having to change even one word [Fetzer, 1981], this was not something I expected. Each time that he asked me to make further revisions, I became more and more disenchanted with our progress.

In the meanwhile, I had been hired as full professor by the University of Minnesota in Duluth, and my colleagues were aware of my

research on this topic. Sometime between the forth and fifth drafts, David Cole showed me a list of "I/O Statements" from the IBM PC manual for Microsoft BASIC, which included the following commands and their expected consequences:

Statement	Action
BEEP	Beeps the speaker.
CIRCLE (x, y) r	Draws a circle with center x, y and radius r.
COLOR b, p	In graphics mode, sets background color and palette of foreground colors.
LOCATE row, col	Positions the cursor.
PLAY string	Plays music as specified by string.

While David had intended these findings as counterexamples to my thesis, I was euphoric, because they were perfect illustrations: there was no way formal proofs could guarantee *a speaker would beep* or *music would play*!

No doubt in part because I incorporated these examples into the fifth draft of my text, the paper was finally acceptable to Rob Kling, and on 13 June 1988, I received a formal acceptance from James Maurer. Since my discussion was no longer primarily criticism of DeMillo, Lipton and Perlis [1979] but a general critique of the limits of formal methods in computer science, Kling suggested I provide a new title for the paper. I responded with "Program Verification: The Very Idea". Since it had taken so long to reach this point, I was surprised when it came out three months later.

The cover of that issue of *Communications* featured a head extending from an enormous pile of computer printouts crying for help. My greatest fear at this point was that my article had finally appeared but no one would even notice or, worse yet, that it would be greeted with yawns as belaboring the obvious. I was therefore pleasantly surprised when my good friend, Chuck Dunlop, with whom I had participated in the program at Wright State, sent me a copy of a message from the Risks Forum, an electronic bulletin board devoted to issues related to computer reliability.

In this posting of 5 October 1988, Brian Randell explained that he had just finished my article "with great interest and enjoyment" and affirmed,

> In my opinion it is a very careful and lucid analysis of the dispute between, e.g., DeMillo, Lipton and Perlis on the one

hand, and Hoare on the other, regarding the nature of programming and the significance of program verification. [Randell 5 Oct 88 9:56:39 WET DST]

He included the text of the abstract and ended his message by quoting the last lines of the paper, where I suggest these matters are not only important theoretically for computer science but practically for the human race.

Chuck advised me that many other messages were now appearing over the nets, which I found reassuring. My worst fears had been allayed: the article was not being ignored, and the initial response had been positive. Shortly thereafter, moreover, I received a letter from Robert Ashenhurst, who edits the Forum for the magazine, dated 1 November 1988. Included were copies of six Letters to the Editor of *Communications*, which Ashenhurst thought were appropriate for publication. He invited me to reply. I agreed with his judgment concerning five and submitted my response.

The letters reflected a variety of attitudes, ranging from a complaint [by James Pleasant] that I had traded upon ambiguity by failing to distinguish exactly what I had in mind by the term "program", to a thoughtful critique [by William Bevier, Michael Smith, and William Young], who suggested that, at the level of logic gates, the difference between abstract entities and causal systems virtually disappears, to additional arguments [by Stephen Savitzky] that supplied further grounds supporting my position in the case of useful programs that are not merely unverifiable but even verifiably incorrect, where the most important requirements of programs may be ones that are not formalizable, etc. [Pleasant, et al., 1989].

Pleasant posed no problem (since I had been entirely explicit on this point); Bevier, Smith and Young could be disarmed (since the difference at stake does not disappear); and Savitzky had come to my defense. The potentially most damaging letter in the set, however, was from Lawrence Paulson, Avra Cohn, and Michael Gordon of Cambridge University, who castigated me for contending that programs must work perfectly, for asserting that verification is useless because it cannot guarantee perfection, and for condemning a subject of which I knew nothing. Since I had not made the claims they attributed to me, I was invulnerable to their criticisms, which were based on drastic oversimplifications of my arguments [Fetzer, 1989b]. But it was beginning to dawn that I might have touched a sensitive nerve.

The worst was yet to come. In correspondence dated 12 December 19-88, Robert Ashenhurst sent along four more letters. Three of these [by Harald Muller, by Christopher Holt, and by Arron Watters] were not unexpected. Muller suggested that I was implicitly drawing a (Platonic) distinction between *the world of the pure* (abstract entities)

and *the world of the real* (causal systems), whereas computers (as constructed artifacts) instead fall in between. Holt suggested that the issue was the correctness of the implementation of the language in which a program is written in the hardware, which most verificationists assume as an *axiom*. And Watters maintained that the truly important issues were not those discussed in my article but those previously raised by DeMillo, Lipton and Perlis [Muller et al., 1989].

What I liked about Muller's letter was not his Platonic framework but the introduction of artificially contrived machines. As I explained in reply, two modes of operation are available. Either the machine is created in accordance with the design (axioms) or the design (axioms) is created in accordance with the machine. *Either way, however, it is necessary to discover precisely how the machine behaves in order to determine whether or not it is in accordance with the design* [Fetzer, 1989c, 511; original emphasis]. While it may be possible to determine the properties of virtual machines by stipulation as a matter of definition (for abstract entities), it is only possible to discover the properties of target machines by the use of induction (for causal systems), thereby reasserting the basic elements of my position.

There was much about Holt's letter with which I completely agreed, so I sought to accent the subtle points of disagreement. While we are all entitled to assume whatever we want for the sake of hypothetical reasoning, there is an important difference between *assuming something to be the case* and *its being the case*. President Reagan, I observed, presumably assumed that we could lower taxes, increase spending, and nevertheless balance the budget. The issue is not whether or not assumptions can be made but under what conditions an assumption is justified, warranted or true.

Arguments based upon hypothetical "axioms" may be valid but are not therefore also sound. Since Hoare had made observations that appeared to conform to Holt's conception, I offered an illustration that is by no means unproblematic:

> When the correctness of a program, its compiler, and the hardware of the computer have all been established with mathematical certainty, it will be possible to place great reliance on the results of the program and predict their performance with a confidence limited only by the reliability of the electronics. [Hoare, 1969, 579]

The catch, I remarked, is that this conditional has an antecedent that may be incapable of satisfaction, since the correctness of the program, its compiler, and the hardware can *never* be established "with mathematical certainty" — unless it happens to be an abstract rather than a physical machine!

If the first three letters were not surprising, the fourth was something else entirely. A scathing diatribe of the likes of which I had never seen before, this letter not only raked me across the coals for misrepresenting the goals and methods of program verification but damned the editors as well:

> by publishing the ill-informed, irresponsible, and dangerous article by Fetzer, the editors of *Communications* have abrogated their responsibility, to both the ACM membership and to the public at large, to engage in serious inquiry into techniques that may justify the practice of computer science as a socially responsible engineering practice. The article is ill-informed and irresponsible because it attacks a parody of both the intent and the practice of formal verification. It is dangerous because its pretentious and ponderous style may lead the uniformed to take it seriously. [Ardis, Basili, et al., 1989, 287]

The letter was signed by ten prominent members of the program verification community. In a handwritten note, Robert Ashenhurst penned that he had just learned that this letter had been forwarded to Peter Denning, the Editor-in-Chief, who planned to respond regarding the review process.

I was stunned. Were this letter, which was receiving special treatment, to appear without a response from me, whether or not I might have forceful and convincing replies to the charges they had raised would not matter: My reputation would have already endured serious, irretrievable damage. In his cover letter, Ashenhurst also mentioned that "the original package" had apparently been "bumped" from January to February. This gave me hope that perhaps something could be done. I called the Executive Editor to plead for the opportunity to provide a response of my own that would appear in *Communications* at the same time as this extraordinary letter.

James Maurer listened patiently as I explained that, while I greatly appreciated the fact that Peter Denning was going to respond on behalf of the editors concerning the review process, it was essential for me to have an opportunity to respond on my own behalf concurrent with the publication of this letter. I observed that it was my name attached to this article and that it was my reputation at stake. I was enormously relieved when, after extensive consultation, the Executive Editor and the Editor-in-Chief decided that I was entitled to respond at the time of its publication. This meant in turn that the "package" would be moved from February to March.

Under considerable pressure, I began to systematically disentangle the objections the authors had raised and consider my replies. There appeared to be at least six issues involved here, some of which were far more serious than others. They asserted, for example, that there are no published claims to "conclusive absolute verification", a phrase that I had used; that assembly language programs *are* amenable to verification procedures, a prospect they contended I had denied; and that I seemed to be "totally unaware" of a large body of work applying formal procedures to compilers and hardware, which indicated to me that these authors had apparently not understood my views.

The absence of the phrase "conclusive absolute verification" from the literature, of course, did not mean that the concept was not present, and formal methods could do no more for compilers or for hardware than they could for programs. The point about assembly language was bothersome, but the only sentence supporting their interpretation concerned the control mechanism of missile systems processing real-time streams of data, "where, to attain rapid and compact processing, their avionics portions are programmed in assembly language—a kind of processing that does not lend itself to the construction of program verifications" [Fetzer, 1988, 1062]. Having originally discussed this point with Chuck Dunlop, I thought that I ought to call him once again.

While we were going though the program, Chuck and I had considered many of these issues before, but perhaps never with such beneficial effect. While it was true in the example I used that the programming was done in assembly language and that programs written in assembly language could indeed be subjected to program verifications, the circumstances of this case precluded that, since these missiles were able to reprogram themselves in flight. We speculated about possible conditions under which such programs could be verified and laughed at our thoughts. I slept very well that night.

My published response, which appeared immediately following the letter in the March issue of *Communications*, began with the following observation:

> The ancient practice of killing the messenger when you do not like the message receives its latest incarnation in the unfortunate letter from Ardis, Basili, et al. ("The Gang of Ten"). The authors allege (a) that I have misrepresented the goal of program verification (thereby attacking a "straw man"); (b) that I have misunderstood the role of mathematics in any engineering endeavor (especially within computer science); and (c) that my conclusion, if it were true, would undermine research in vast areas of computer science (including most theoretical work). [Fetzer, 1989a, 288]

In rebuttal, I observed (a) that, since I was attacking Hoare's position, as I had clearly explained, I was attacking a "straw man" only if Hoare's position was a "straw man"; (b) that the role of mathematics in engineering qualifies as *applied mathematics* when used to describe physical structures and as *pure mathematics* when used to describe abstract structures; and (c) that it was hard to believe these authors could seriously maintain that computer science could benefit from misrepresenting the certainty of its findings.

I remarked that the authors had "misdescribed" my conclusion, since it was not my position that program verification was useless or even harmful because it provides no certainty. My point, on the contrary, was that "since program verification cannot guarantee the performance of any program, it should not be pursued in the false belief that it can—which, indeed, might be entertained in turn as the 'ill-informed, irresponsible and dangerous dogma' that my paper was intended to expose" [Fetzer, 1989a, 288].

My favorite passage, however, was one inspired by my discussion with Chuck. While acknowledging that the sentence I used in describing assembly-language programming as a type that did not lend itself to program verifications might have been misleading, I reaffirmed the force of my example involving real-time transmission of streams of data from sensors to processors:

> the specific avionics example that I was discussing . . . reflects a special type of programming that can be found in cruise missiles and other sophisticated systems with the capacity to reprogram themselves en route to their targets. The only technique that would permit the verification of these programs as they are generated in flight would be procedures permitting the correctness of these programs to be established as they are constructed. Perhaps the authors of this letter could volunteer to accompany these missiles on future flights in order to demonstrate that this is a type of programming that actually does lend itself to the construction of program verifications, after all. [Fetzer , 1989a, 288]

I find it impossible to reread this passage without smiling even to this day.

Having discovered that the *Journal of Automated Reasoning* was about to publish a new paper by Avra Cohn entitled, "The Notion of Proof in Hardware Verification", in which she acknowledged that verification "involves a pair of *models* that bear an uncheckable and possibly imperfect relation to the intended design and to the actual device" [Cohn, 1989, 132], I cited it as "display[ing] a great deal of sensitivity to the basic issues at stake in my article", in spite of the fact that "she is one of three Cambridge scholars who reject my analysis on peripheral grounds elsewhere" in this magazine.

I concluded my reply by praising the efforts expended on my behalf by the editors and staff of *Communications*, especially Rob Kling, "in providing sympathetic criticism and in enforcing high standards". I finished with observations about the tone and quality of the letter from the Gang of Ten:

> In its inexcusable intolerance and insufferable self-righteousness, this letter exemplifies the attitudes and behavior ordinarily expected from religious zealots and ideological fanatics, whose degrees of conviction invariably exceed the strength of the evidence. [Fetzer, 1989a, 289]

I remain enormously indebted to James Maurer and to Peter Denning for granting my plea to publish my response at the same time as their letter.

The March 1989 issue of *Communications* thus began with that letter, which took up a full-page of the Forum, followed by my reply. Peter Denning, the Editor-in-Chief, true to his word, offered a powerful defense of the editorial process, observing that the paper had been put through four rounds of revision and noting that "the article was subjected to a review more rigorous than is required by ACM policy, and that the review process was fair and professionally sound. I stand fully behind my editors." [Denning, 1989, 289]. The letters from Pleasant, the three Cambridge scholars, et al. appeared with my replies as "Technical Correspondence".

The April 1989 issue published the letters from Muller, Holt, and Watters, which was something that I expected, together with an OP/ED piece entitled, "Program Verification: Public Image and Private Reality", which I had not [Dobson and Randell, 1989]. The authors were John Dobson and Brian Randell, the same "Brian Randell" who had posted a favorable notice of my article on the Risks Forum. I was therefore somewhat distressed to read that, while their initial opinion had been quite favorable in viewing it as "an interesting, and unusually literate, contribution to the literature on the theoretical limitations of program verification", they had now reached the conclusion that it was "undoubtedly . . . misconceived" but perhaps useful in correcting overselling of their product by the verification community.

Thus, although I was alleged to have "failed to give this topic the careful scrutiny it so clearly needs, they said the program verification community had not done so either. Their principal objections to my analysis of the theoretical limitations of program verification, how-

ever, were two in number: (1) "the (unfortunately justifiable) fear that [my] paper will be misinterpreted by laymen, particularly those involved in funding" [Dobson and Randell, 1989, 420] and, (2) their belief that I had mistakenly entertained proofs of program correctness as intended to provide explanatory rather than evidential reasons:

> *it seems to us that Fetzer thinks the verificationists have been representing their work as providing explanatory reasons for program correctness, whereas they claim their work provides merely evidential reasons* [Dobson and Randell, 1989, 421, original emphasis].

I was acutely disappointed that an author who had been so positive about my work had now withdrawn his support as I began to draft my response.

Before they raised the issue, I had not considered—even remotely—whether or not an analysis of the scope and limits of formal methods in computer science had any financial ramifications. The "fear" that my paper might be "misinterpreted" by those who fund verification projects and that they might be "persuaded" to reallocate their resources for more promising activities, of course, is a classic example of the informal fallacy known as "the appeal to pity", where the unfortunate consequences that might follow if a certain position were accepted as true are treated as though they counted as evidence that it is false. But this was no reason to suppose I was wrong.

Their first objection, therefore, was simply fallacious, although I could appreciate why those whose profession, promotions, or tenure depend on funding of this kind might have been unsettled by my analysis. In asserting that I had mistaken "evidential" for "explanatory" reasoning, however, they raised issues of a different caliber entirely. Their second objection, which implied that I had attacked a straw man by portraying the function of proofs of program correctness as explanatory in relation to executions when they should be understood as evidential, after all, represented a set of issues which to any philosopher of science would be extremely familiar.

There is in fact a fundamental distinction between *evidential reasoning* and *explanatory reasoning*, as Carl G. Hempel, some time ago, explained by distinguishing "explanation-seeking" questions (such as, "Why is it the case that p?") from "reason-seeking" questions (say, "What grounds are there for believing that p?"). While adequate answers to explanation-seeking questions also provide potential answers to corresponding reason-seeking questions, the converse is not the case [Hempel, 1965, 335]. To that extent, of course, I thought that Dobson and Randell had drawn a relevant distinction.

It was fascinating to observe them illustrating the difference that they wanted to invoke with two examples regarding the height of a tree, which might be *explained* by considerations such as, "that it is a

tree of such-and-such a sort, growing in this sort of climate in this sort of soil, and that under such conditions trees of that type can sustain a height of about 100 feet", on the one hand, or established *evidentially* by considerations such as, "that it casts a shadow of 50 feet at a time of day when a 10-foot pole casts a shadow of 5 feet" [Dobson and Randell, 1989, pp. 420-421]. The parallel with Sylvain Bromberger's flagpole example was both obvious and remarkable.

When Dobson and Randell claimed that mathematical logic provides "explanatory reasons", however, they committed a blunder. Logic specifically and formal methods generally are context independent: they are applicable for deriving conclusions from premises without any concern for the purpose of the arguments thereby constructed. Indeed, the flagpole case—and their own example!—clearly indicate that the ability to deduce a conclusion (about the height of a flagpole, for example) is not the same as the ability to explain that phenomenon (why the flagpole has that height). No student of the symmetry thesis would have been prone to commit this mistake [Fetzer, 1974].

Since I had drawn a distinction between *algorithms* as logical structures and *programs* as causal models of those structures, they properly noticed that I focused on the causal contribution of programs to the performance of computers. An important ramification of my position is the claim that the outcome of executing a program "obviously depends upon various different causal factors, including the characteristics of the compiler, the processor, the printer, the paper and every other component whose specific features influence the execution of such a program" [Fetzer, 1988, 1057].

Since programs are only partial causes of the effects of their execution, if I were taking formal proofs to be explanatory, when their proponents only intend them to be evidential, then I would be holding them at fault for failing to guarantee what a computer will do when it executes a program when their reasoning is merely evidential. This position, however, not only contradicts Dobson and Randell's assumption that formal proofs are always explanatory but also overlooks the consideration that differences between virtual machines and causal systems still remain whether program verifications are viewed as evidential *or* as explanatory reasoning.

Even when we overlook the inconsistency in their position by supposing that program verifications as formal proofs of correctness are merely intended to provide evidential reasons for believing that a computer will perform correctly when that program is executed, the difference between *conclusive* and *inconclusive* evidence for a conclusion also persists. Thus, the fundamental objective of my analysis was to establish that, for target machines as opposed to virtual machines, the kind of evidence which program verifications can appropriately provide is uncertain rather than certain evidence, which not only does

not contradict the possibility that program verifications are evidential but even implies it [Fetzer, 1989e, 920].

When I submitted my response to Dobson and Randell [1989], however, Peter Denning balked. I asked for equal time by having it run as an OP/ED piece, which is technically known there as a "Viewpoint", but he wanted to edit it drastically and to print it as a Letter to the Editor instead. It finally appeared in the August issue [Fetzer, 1989e]. In the meanwhile, the magazine published a letter in which I extended the distinction between algorithms and programs to the problem of patent and copyright protection in the June issue [Fetzer, 1989d] and another set of Letters to the Editor regarding the program verification debate in the July issue [Hill et al., 1989].

These events were insignificant by contrast with the next development, however, which occurred when Jon Barwise devoted his column in *Notices of the AMS* [American Mathematical Society] to the program verification controversy. He began by discussing the celebrated debates that occurred early in the 20th century over the foundations of mathematics which, after extended deliberations that seemed to generate more heat than light, led to the formulation of various positions about the nature of mathematics, such as Platonism, logicism, formalism and intuitionism, which have contributed to our understanding of the subject and have "kept the wolf from the door".

Barwise drew an explicit comparison between the debates over the foundations of mathematics and the program verification debate, which he described as concerning the relation of mathematics "to the rest of the world":

> Today a similar controversy about the nature of mathematics and its relation to the rest of the world is raging out of sight of most mathematicians in the pages of *CACM*, the *Communications of the Association for Computing Machinery*. The debate is almost as exciting and at least as acrimonious. . . . The present debate swirls around an article called "Program Verification: The Very Idea", written by the philosopher James Fetzer. [Barwise, 1989, 844]

Barwise suggested that the program verification debate might contribute to our understanding of the nature of *applied* mathematics as the earlier debates contributed to our understanding of the nature of *pure* mathematics.

I was enormously flattered. Barwise followed Bevier, Smith and Young in characterizing my position by means of three key contentions as follow:

> (1) The purpose of program verification is to provide a mathematical method for guaranteeing the performance of a program.
> (2) This is possible for algorithms, which cannot be executed by a machine, but not possible for programs, which can be executed by a machine.
> (3) There is little to be gained and much to be lost though fruitless effort to guarantee the reliability of programs when no guarantees are to be had. [Barwise, 1989, 845]

And he reported that I accepted this summary as perfectly reasonable, "so long as the first premises is intended as a reflection of the position that is – implicitly or explicitly – endorsed by the proponents of program verification."

In my estimation, the discussion by Barwise was a valuable contribution to the debate, partially because, in general, he endorsed my position. But I also took exception to some of his arguments in a letter that subsequently appeared in his column. In particular, he alleged that I had committed the Fallacy of Identification by viewing "proofs" as purely syntactical entities, an objection to which I took exception on the grounds that this conception of "proof" was the relevant one within this context; and he claimed that I had failed to sufficiently differentiate "programs" as (abstract) types and "programs" as (causal) instances, a conception fundamental to my position.

I therefore responded to this objection by distinguishing programs-as-texts (unloaded) from programs-as-causes (loaded), where (human) verification involves the application of deductive methods to programs-as-texts:

> Hoare and I both assume that programs-as-causes are represented by programs-as-texts. The difference is that Hoare assumes that programs-as-causes are always *appropriately* represented by programs-as-texts, an assumption that I challenge. [Fetzer, 1989f, 1353]

Thus, programming languages themselves function as models of (actual or virtual) machines, where the degree of correspondence between them is a matter that, in the case of target machines, unlike that of virtual machines, cannot possibly be ascertained on the basis of purely deductive reasoning.

There have been several interesting developments since then. In 1990, for example, the Association for Computing Machinery invited me to participate in a formal debate on the scope and limits of formal methods in software engineering. My opponents were Mark Ardis and David Gries, two of "The Gang of Ten". More significantly, in 1991, I

received an invitation to contribute a 10,000 word entry on "program verification" from Allen Kent and James G. Williams, the editors of the *Encyclopedia of Microprocessors*, an entry that they also included in their *Encyclopedia of Computer Science and Technology*, both published by Marcel Dekker [Fetzer, 1993a and 1994].

In my more recent work on this subject, I have emphasized the differences between formal systems, scientific theories, and computer programs. While mathematical proofs, scientific theories and computer programs qualify as syntactical entities, scientific theories and computer programs have a semantic significance (for the physical world) that proofs (in pure mathematics) do not possess. And computer programs possess causal capability that even scientific theories do not enjoy, which reflects the fact that each of them can be subjected to different methods of evaluation [Fetzer, 1991].

I have also collaborated with Tim Colburn and Terry Rankin in editing a collection of classic and contemporary articles on program verification, which includes papers by John McCarthy, Peter Naur, Robert Floyd, C. A. R. Hoare, William Scherlis and Dana Scott, Bertrand Meyer, Bruce Blum, Christiane Floyd, Brian Smith, and other papers I have discussed [Colburn et al., 1993]. In my capacity as the editor of *Minds and Machines*, I have sought to nurture the fledgling field of "the philosophy of computer science", as I think of it, and have had the opportunity to publish several important articles by Colburn [1991] and David Nelson [1992 and 1994], among others.

Opinions appear to differ over whether or not the program verification debate has had any impact on the computer science community. A book entitled *Fatal Defect* by Ivars Peterson will appear next month [Peterson, 1995]. He discusses a series of mishaps and accidents involving computer systems and reviews contributions by many experts in the field, including four of "The Gang of Ten". He describes the program verification debate, accurately quoting passages from "Program Verification: The Very Idea":

> "The limitations involved here are not merely practical: *they are rooted in the very character of causal systems themselves*", Fetzer emphasized. "From a methodological point of view, it might be said that programs are conjectures, while executions are attempted—and all too frequently successful—refutations." [Peterson, 1995, 181]

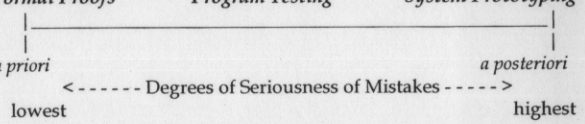
Nevertheless, he concludes that, "In the end, the debate didn't settle much of anything, and Fetzer's arguments did not derail program verification", although its proponents are perhaps "less extravagant" in their claims and "less casual" in their use of the language of "proof" [Peterson, 1995, 183].

During a visit to England last November, however, I had the pleasure of presenting a lecture at King's College of the University of London, during which I reviewed many of these matters. During the discussion, I was fascinated to learn from a member of the audience that he had been in Japan recently and that, while waiting at a train station outside Tokyo, he had encountered C. A. R. Hoare. He took the opportunity to ask Hoare about something he had heard of but did not then understand, the program verification debate. Hoare almost immediately launched into an explanation of how the use of formal methods may detect problems in programs at an early stage and thereby yield better average performance under statistical controls.[4]

Whether or not philosophy has contributed to computer science in this instance, there appears to be an important interaction phenomenon here that deserves to be considered. As I previously explained, I deliberately chose to pursue these issues without appealing to the analytic/synthetic distinction, first, because it would have required explanation for the benefit of computer scientists, and, second, because many philosophers deny it. Even though I myself have never believed there were appropriate grounds for its rejection, I was reluctant to rest my case on such a disputed premise.

The more I have examine these problems, however, the more convinced I have become of the tenability—*and even vitality*—of that distinction. The differences between virtual and physical machines, between abstract entities and causal systems, between algorithms and programs, between pure and applied mathematics, and between validity and soundness upon which these crucial issues depend are diverse manifestations of the underlying difference between kinds of knowledge that are analytic and *a priori* and kinds of knowledge that are synthetic and *a posteriori*. To those who remain skeptical of this distinction, I ask you to consider the significance of the program verification debate as a form of vindication of the distinction.

My studies of computer systems have also convinced me that the more serious the consequences of making mistakes, the greater our obligation to insure that they are not made. This obligation in turn implies that purely formal methods must give way to program testing and system prototyping as the degree of seriousness of making mistakes increases, as follows:



Thus, this appears to me to be one situation in which philosophy makes a (non-trivial) difference to important issues of public policy [Fetzer, 1996].

ACKNOWLEDGMENTS

I am deeply indebted to Chuck Dunlop, Al Sanders, Rob Kling, David Cole, David Nelson and Tim Colburn. For further discussion, see Colburn [1993].

NOTES

[1] The program had been inspired by David Hemmendinger, a philosophy Ph.D. who had acquired an M.S. in computer science and had joined the faculty at Wright State. The other participants were Charles E. M. Dunlop, an old friend who convinced me that I should join him in taking advantage of this program while it lasted; Ken Ray, a former student of his and of mine when we were visiting faculty at the University of Cincinnati during 1978-80; Adam Drozdek, who recently received tenure at Duquesne University; and Joe Sartorelli, who was then on leave from Arkansas State University.

[2] My grounds for rejecting Quine's critique of the analytic/synthetic distinction are elaborated, for example, in Fetzer [1990, 105-110], and in Fetzer [1993b, 16-21]. Indeed, my first written assignment in graduate school at Indiana University in 1966 was composing a critique of "Two Dogmas". While I disagreed with Quine even then regarding the first "dogma", I agreed with him regarding reductionism; see Fetzer [1993b, 51-55].

[3] If the paper had begun with this title, however, it might not have been published. Years later, I was advised that the area editor for dependable computing, John Rushby, had said that, if it had come to him for editorial review, he would have killed it. Such are the vicissitudes of publication.

[4] The audience member turned out to be the philosopher Donald Gillies.

REFERENCES

Ardis, M., V. Basili et al. (1989) "Editorial Process Verification", *Communications of the ACM,* March 1989: 287-288.

Barwise, J. (1989) "Mathematical Proofs of Computer System Correctness", *Notices of the AMS.* September 1989: 844-851.

Cohn, A. (1989) "The Notion of Proof in Hardware Verification", *Journal of Automated Reasoning* **5**: 127-139.

Colburn, T. R. (1991) "Program Verification, Defensible Reasoning, and Two Conceptions of Computer Science", *Minds and Machines* February 1991: 97-116.

Colburn, T. R. (1993) "Computer Science and Philosophy", in T. R. Colburn, J. H. Fetzer, and T. L. Rankin, Eds., *Program Verification* Dordrecht, The Netherlands: Kluwer Academic Publishers, 3-31.

Colburn, T. R., J. H. Fetzer, and T. L. Rankin, Eeds. (1993) *Program Verification.* Dordrecht, The Netherlands: Kluwer Academic Publishers.

DeMillo, R., R. Lipton and A. Perlis. (1979) "Social Processes and Proofs of Theorems and Programs", *Communications of the ACM,* May 1979: 271-280.

Denning. (1989) "Reply from the Editor in Chief", *Communications of the ACM,* March 1989: 289-290.

Dobson, J. and B. Randell. (1989) "Program Verification: Public Image and Private Reality", *Communications of the ACM,* April 1989: 420-422.

Fetzer, J. H. (1974) "Grunbaum's 'Defense' of the Symmetry Thesis", *Philosophical Studies,* April 1974: 173-187.

Fetzer, J. H. (1981) *Scientific Knowledge.* (Dordrecht, The Netherlands: D. Reidel.

Fetzer, J. H. (1988) "Program Verification: The Very Idea", *Communications of the ACM,* September 1988: 1048-1063.

Fetzer, J. H. (1989a) "Response from the Author", *Communications of the ACM,* March 1989: 288-289.

Fetzer, J. H. (1989b) "Program Verification Reprise: The Author's Response", *Communications of the ACM,* March 1989: 377-381.

Fetzer, J. H. (1989c) "Author's Response", *Communications of the ACM,* April 1989: 510-512.

Fetzer, J. H. (1989d) "Patents and Programs", *Communications of the ACM*, June 1989: 675-676.

Fetzer, J. H. (1989e) "Another Point of View", *Communications of the ACM*, August 1989: 920-921.

Fetzer, J. H. (1989f) "Mathematical Proofs of Computer System Correctness: A Response", *Notices of the AMS*, December 1989: 1352-1353.

Fetzer, J. H. (1990) *Artificial Intelligence: Its Scope and Limits*, Dordrecht, The Netherlands: Kluwer Academic Publishers.

Fetzer, J. H. (1991) "Philosophical Aspects of Program Verification", *Minds and Machines*, May 1991: 197-216.

Fetzer, J. H. (1993a) "Program Verification", *Encyclopedia of Computer Science and Technology*, Vol. 28. New York, NY: Marcel Dekker, 237-254.

Fetzer, J. H. (1993b) *Philosophy of Science*. New York, NY: Paragon House Publishers.

Fetzer, J. H. (1994) "Program Verification", *Encyclopedia of Microprocessors*, Vol. 14. New York, NY: Marcel Dekker, 47-64.

Fetzer, J. H. (1996) "Computer Reliability and Public Policy: Limits of Knowledge of Computer-Based Systems", *Social Philosophy & Policy*, 13: 229-266.

Hempel, C. G. (1965) *Aspects of Scientific Explanation*. New York, NY: The Free Press.

Hill, R., R. Conte et al. (1989) "More on Verification", *Communications of the ACM*, July 1989: 790-792.

Hoare, C. A. R. (1969) "An Axiomatic Basis for Computer Programming", *Communications of the ACM*, October 1969: 576-583.

Marcotty, M and H. E. Ledgard. (1986) *Programming Language Landscape*, 2nd ed. Chicago, IL: Science Research Associates, 1986.

Muller, H., C. Holt, and A. Watters. (1989) "More on the Very Idea", *Communications of the ACM*, April 1989: 506-510.

Nelson, D. (1992) "Deductive Program Verification (A Practitioner's Commentary)", *Minds and Machines*, August 1992: 283-307.

Nelson, D. (1994) Discussion Review of Robert S. Boyer and J Strother Moore, *A Computational Handbook*, and J Strother Moore (ed.), "Special Issue on System Verification", *Journal of Automated Reasoning*, December 1989, *Minds and Machines*, February 1994, 93-101.

Peterson, I. (1995) *Fatal Defect: Chasing Killer Computer Bugs*. New York: Random House/Times Books.

Pleasant, J., L. Paulson et al. (1989) "The Very Idea", *Communications of the ACM*, March 1989, 374-377.