Brit. J. Phil. Sci. **29** (1978), 213–222 *Printed in Great Britain*

Three Myths of Computer Science

by JAMES H. MOOR

In this paper I will discuss a basic concept of computer science—the concept of a computer program—and three related distinctions of computer science—software *vs.* hardware, digital *vs.* analogue, and model *vs.* theory. I believe all of these notions are important, and if properly understood, provide an important part of the conceptual framework of computer science. But, when these notions are misunderstood, certain myths tend to evolve. With regard to artificial intelligence work these myths perpetuate sloppy research, and with regard to philosophy these myths promote misconceptions related to the mind-body problem and scientific methodology.

1 COMPUTER PROGRAMS

Computers can be understood on two levels. Computers can be analysed on the physical level just as any physical system can. If the computer is electronic, then the laws of physics concerning electronics can explain the computer activity. But, it is the understanding on the symbolic level which makes computers calculating devices, for it is under this kind of interpretation that various structures or processes of the computer are understood as symbols. Similarly, a computer program can be understood on two levels. Physically, computer programs may be a series of punched cards, configurations on magnetic tape, or in any number of other forms. Symbolically, computer programs are understood as instructions to a computer.

Often, proposed definitions of 'computer program' are misleading. The following definition from a dictionary of computers is a typical example: A computer program is 'a set of instructions composed for solving a given problem by a computer' (Chandor [1970], p. 303). Initially, this definition seems reasonable enough, but a little reflection shows that it is both too narrow and too broad. First, consider computer programs which are

Received 28 *September* 1977

excluded by the definition. Computers can be and sometimes are programmed to produce other computer programs. The resulting programs are not necessarily *composed* for solving given problems by computer. In the simplest case the computer can generate random lists of computer instructions some of which turn out to be computer programs although the resulting programs were not designed for any given purpose. Furthermore, although computers do a lot of problem-solving, it is very misleading to characterise all computer activity in this way unless 'problem-solving' is taken in the broadest possible sense. Computers can perform all sorts of symbol manipulations. Sometimes computer programs do no more than instruct the computer to print certain characters. Secondly, consider items which are not generally considered to be computer programs but which are included by the definition above. If I stand in front of a computer and say, 'Calculate my tax return and report the results to the government', I would be giving a set of instructions composed for solving a given problem by computer, but in normal circumstances I would not be giving a computer program. Flow-charts often give sets of instructions composed for solving problems by computers, but not all such flowcharts are computer programs. One might be tempted to improve this definition by insisting that the instructions be at least be machine readable; but the problem with this suggestion is that often computer programs are printed on paper, scribbled on blackboards, expressed orally, or given in any number of other forms which are in general not machine readable.

I believe we can avoid these problems and still capture our intuitive notion of a computer program if we recast the definition as follows: a computer program is a set of instructions which a computer can follow (or at least there is an acknowledged effective procedure for putting them into a form which the computer can follow) to perform an activity. For instance, if for some instructions scribbled on a blackboard, one would have only to type them into a computer and the computer could follow them to perform an activity, then the instructions on the blackboard make up a computer program. Some flowcharts are computer programs but some are not. Those which are not may provide insight into the logic of an operation, but they are not given in a specific enough way so that an acknowledged effective procedure could put them into a form which a computer could use. Finally, the reason the oral instructions 'calculate my tax return and report the results to the government' fail to be a computer program is not because they are oral. Some computers today can execute a limited number of oral commands. Moreover, the instructions could always be typed into a computer. The problem is that no computer today could follow them. Of course, this is not to deny that there could be situations in the future in which such instructions could become a computer program, *e.g.*, a situation in which a computer is able to understand the instructions, has access to records of income, can communicate with the government tax office, *etc.* In principle virtually any physical entity or process could be understood symbolically as a computer instruction if one imagines an appropriate computer which could understand and execute the instruction. But as a practical matter what we regard as computer instructions, and consequently what we regard as computer programs, is determined by the computers available.

2 SOFTWARE/HARDWARE

Computer programs are often considered to be part of the software of a computer as opposed to the hardware. Unfortunately, computer hardware is frequently characterised as 'the physical units making up a computer system' (Chandor [1970], p. 179). By contrast this seems to suggest that software is not part of the computer system or even worse that it is not physical. It is important to remember that computer programs can be understood on the physical level as well as the symbolic level. The programming of early digital computers was commonly done by plugging in wires and throwing switches. Some analogue computers are still programmed in this way. The resulting programs are clearly as physical and as much a part of the computer system as any other part. Today digital machines usually store a program internally to speed up the execution of the program. A program in such a form is certainly physical and part of the computer system.

Furthermore, since programming can occur on many levels, it is useful to understand the software/hardware dichotomy as a pragmatic distinction. For a given person and computer system the software will be those programs which can be run on the computer system and which contain instructions the person can change, and the hardware will be that part of the computer system which is not software. At one extreme if at the factory a person who replaces circuits in the computer understands the activity as giving instructions, then for him a considerable portion of the computer may be software. For the systems programmer who programs the computer in machine language much of the circuitry will be hardware. For the average user who programs in an applications language, such as Fortran, Basic, or Algol, the machine language programs become hardware. For the person running an applications program an even larger portion of the computer is hardware. This pragmatic view of the software/hardware distinction makes the distinction both understandable and useful·

A myth concerning the software/hardware distinction can arise, however, if the distinction is understood and taken to have more ontological significance than it has. With regard to work being done in cognitive simulation, consider a distinction by Keith Gunderson between program-receptive features of mentality and program-resistant features of mentality ([1971], p. 70 f.). Gunderson says that program-receptive features include most kinds of problem-solving such as game-playing and theorem-proving; whereas, program-resistant features, Gunderson argues, are task-oriented and something clearly counts as success and failure in accomplishing the task. If a subject is problem-solving, he can provide protocol statements to report on how he is attempting to solve the problem and often the problem-solving abilities acquired by doing a few problems can be projected to new situations. Gunderson states that these properties generally do not apply to program-resistant features. The program-receptive *vs.* the program-resistant distinction seems plausible if Gunderson is simply suggesting that some mental abilities may be easier to program than others. But Gunderson makes it clear that he means much more than this. Gunderson ([1971], p. 73) says, 'It would be a methodological howler to attempt to simulate pains, feelings, and emotions with a machine simply by expanding current programming techniques in the sense of defining new routines. It may, of course, prove possible to produce machine analogues of these features through a development in hardware.' Gunderson ([1971], pp. 73–4) concludes, 'Thus, some psychological predicates may be viewed primarily as software predicates; some primarily as hardware predicates.' Consequently, Gunderson believes that those who do cognitive simulation work which attempts to program feelings, emotions, or various personality traits are making a basic methodological mistake involving the software/hardware distinction.

Gunderson's recommendations are generated by reading more into the software/hardware distinction than is there. What is considered hardware by one person can be regarded as software by another. Again, if at the factory the installation of certain circuits into a computer is interpreted on the symbolic level as giving instructions, then in that situation the resulting programs can be regarded as software.

Furthermore, even if one fixes the boundary between software and hardware at a particular level, *e.g.*, on the applications language level, there is still no reason to believe that it is a methodological mistake to attempt to simulate "program-resistant" features. For if one can simulate the features on the hardware of a digital machine, then this activity can be described in terms of a computable function. Such a function could be understood in terms of a Turing machine which can certainly be simulated through programming on the suggested software level.

Wherever the boundary between software and hardware is drawn, additional hardware is often helpful; for it is likely to increase software capabilities. In cognitive simulation there can be some very special, practical reasons for wanting more and somewhat unique hardware. For example, in simulating certain activities like perception some researchers believe it is desirable to have machines which have parallel processing capabilities to reduce the processing time and to make the programming somewhat easier. Other researchers have built robots with mechanical arms and television eyes. Watching robots controlled by the computer can make the simulation of certain cognitive activity much more realistic. But, the point is that capabilities should be cast into hardware only to the extent that there is no longer concern with changing them. For instance, to make a simulation of feelings more realistic one might build a piece of hardware which emitted an 'ow' sound when stimulated. We understand that an 'ow' sound should be omitted sometimes; but what we don't understand is what leads up to it and in general how the affective system is integrated with the cognitive system. It is investigation into these areas which requires software because we want to manipulate such capabilities in doing research. Thus, Gunderson is mistaken in claiming that programming is an inappropriate approach to simulating "program-resistant" features of mentality. On the contrary, in development of computer simulations of "program-resistant" features of mentality, programming is essential.

3 DIGITAL/ANALOGUE

Computers are often classified as one of two types—digital or analogue. There are also some hybrid computers which include both digital and analogue components. The essential difference between digital and analogue computers is generally taken to be in the type of information processing. In a digital computer information is represented by discrete elements and the computer progresses through a series of discrete states. In an analogue computer information is represented by continuous quantities and the computer processes information continuously. I believe this is a worthwhile distinction as long as it is understood that it is a distinction made on the symbolic level and not on the physical level of understanding the computer. Taken as strict physical descriptions the digital and analogue characterisations would be very misleading. Alan Turing puts the point nicely concerning discrete state machines.

[Discrete state machines] are the machines which move by sudden jumps or clicks from one quite definite state to another. These states are sufficiently different for the possibility of confusion between them to be ignored. Strictly speaking there are no such machines. Everything really moves continuously. But there are many kinds of machines which can profitably be *thought of* as being discrete state machines. (Turing [1950], p. 439.)

There is also a discrepancy which accompanies the analogue characterisation if it is understood as a practical physical description. Theoretically, a dense continuum of values may be represented by a continuous physical quantity, but in practice the computer's sensitivity and a user's ability to discriminate these values are always limited. In practical terms the analogue computer seems capable of only a finite number of discrete states! A simple example is an ordinary slide rule, often regarded as an analogue computer, which is accurate only to a few places.

Such discrepancies are not important because the digital and analogue characterisations are not, or at least should not, be given as complete and accurate physical descriptions of the computer. Rather the digital or analogue characterisation is an interpretation on the symbolic level. The relevant physical feature will be abstracted. If it is a digital interpretation, continuities will be ignored; and if it is an analogue interpretation, discontinuities will be ignored. Undoubtedly, some physical systems are more easily interpreted in one way than the other. Nevertheless, in principle most, if not all, physical systems which might be considered to be computers could be interpreted either in digital or in analogue terms. For example, consider an early computer by Pascal which performed simple calculations by the movement of cogged wheels. Is Pascal's computer a digital or analogue machine? If we interpret the cogs in the gears as digits and understand the completed movements of the gears as discrete states, then Pascal's device is a digital computer. On the other hand, if we interpret the gears as representing a continuum of values and focus on the continuous movement, then Pascal's device is an analogue computer.

A myth about the digital/analogue distinction can arise if the distinction is given more ontological significance than it has, *i.e.*, if one believes that there are intrinsic physical properties which divide computers into one of these two classes. This myth can generate bogus questions such as 'Is the brain *really* a digital or an analogue device?' The brain can be interpreted in either way. If one treats neurons as being either in a firing or a non-firing state, then the brain can be understood as digital; and if one emphasises the ongoing chemical processes, then the brain can be understood as analogue. The "correct" interpretation will depend upon one's purposes.

Hubert Dreyfus ([1972], pp. 71–4) argues that researchers in cognitive simulation and artificial intelligence are misguided in their extensive use of digital machines because intelligent human behaviour stems from the brain which is most likely an analogue device. Thus, Dreyfus attempts to use the digital/analogue distinction to support his general conclusion that further significant progress in research in cognitive simulation or artificial intelligence is 'extremely unlikely' (Dreyfus [1972], p. 197). It is not sufficient in replying to Dreyfus to point out that digital computers can calculate whatever analogue computers can, for Dreyfus agrees that digital computers could simulate analogue computers at least with regard to the end result (Dreyfus [1972], p. 233). Dreyfus' concern is that digital computers will never be able to represent fully analogue information processing.

Dreyfus' argument is most forceful if one believes that the digital/analogue distinction ontologically separates computers into two classes, for this suggests that researchers in cognitive simulation and artificial intelligence are manipulating the wrong material. To use Dreyfus' analogy it is similar to the attempts of alchemists to turn lead into gold. Once it is realised that the digital/analogue distinction is not based upon intrinsic physical properties but is a matter of interpretation, then Dreyfus' argument loses much of its force.

Dreyfus might reformulate his view as follows: The conceptual scheme given by the analogue interpretation is more useful in characterising those information processes which lead to intelligent behaviour than that given by the digital interpretation. Now the issue becomes pragmatic. The claim might be put: Just as it is easier to solve certain differential equations on an analogue computer; it is easier to handle information processing leading to intelligent behaviour on an analogue computer. But, for Dreyfus' purposes, there are two clear shortcomings with such a position. First, the position as a criticism against cognitive simulation and artificial intelligence seems at best only hypothetical since so little is known about how knowledge, for instance, might be represented in a computer under any interpretation. Secondly, even if the claim could be established, it would not support Dreyfus' original, pessimistic conclusion that further significant progress in this area of research is extremely unlikely, but only that workers in cognitive simulation and artificial intelligence who use a digital analysis are labouring under a clumsy interpretation. Just as digital computers can be used to solve differential equations; it would seem that, however clumsily, processes leading to intelligent behaviour could be represented as accurately as one desired in a digital form.

4 MODEL/THEORY

A major activity in computer science is modelling. Computer models are constructed to simulate everything from the contraction of the heart muscle to the growth of rice crops. Of course, scientists have utilized models for centuries to help understand and test theories, and it is certainly appropriate for them to use the recently developed computer technology to continue to do this. However, computer scientists often speak as if there is no distinction among programs, models, and theories; and discussions slide easily from programs to models and from models to theories. For instance, Geoffrey Clarkson ([1963], p. 347) describes work on a model of the trust investment process as follows: 'Our model, written as a computer program, simulates the procedures used in choosing investment policies for particular accounts, in evaluating alternatives presented by the market, and in selecting the required portfolios'. A few lines later Clarkson says, 'Since our model is a theory of individual decision-making behaviour, the method of analysis is based on the theory of human problem-solving [by Newell, Shaw, and Simon].' There is nothing wrong with such talk as long as it is understood that is a *façon de parler* and it is realised that there are important distinctions among programs, models, and theories.

One can have a theory, *i.e.*, a set of laws used to explain and predict a set of events, without having a model except for the subject matter itself. Also, one can have a model of a given subject matter, *i.e.*, a set of objects or processes which have an isomorphism with some portion of the subject matter, without having a theory about the subject matter. For example, soap film on irregularly shaped wires is sometimes regarded as an analogue computer which gives solutions to minimisation problems (Isenberg [1976]). Although the soap film on the irregularly shaped wire can serve as a model, it does not generate a theory about minimisation. Today there is no mathematical theory which can provide a general existence proof for such solutions let alone an analytical approach for producing them. Such a mathematical theory might be developed some day, but of course the theory could be produced independently of models like the soap film model.

Further, a computer model is usually much more than just a computer program, for a computer program is a set of computer instructions in the sense defined earlier. For the computer model to be understood as a model the isomorphism has to be pointed out between the computer activity (as produced by the program, perhaps) and the subject matter in question. In general, one must give the computer activity a symbolic interpretation which goes beyond the standard symbolic understanding of the program in order to make it a model.

The model/theory myth occurs in computer science when the model/theory distinction is blurred so that programming a computer to generate a model of a given subject matter is taken as tantamount to producing a theory about the subject matter or at the very least an embodiment of a theory. As John Loehlin ([1968], p. 5) expresses it, 'A computer model of a system is a concrete embodiment of a theory about the operation of that system, and running it in a computer is a way of determining what the theory predicts under specified sets of conditions.' Such a claim is extremely misleading since a computer model does not automatically embody a theory in the important sense that one knows what the theory is. Understanding the isomorphism between the model and the subject matter does not necessarily reveal the theoretical connections which will allow explanations and predictions. The theory must be statable independently of the computer model. Only then can it be said in an interesting sense that the model embodies the theory, for only then can one view the subject matter or the model in terms of the theory.

This last point may seem unimportant since it might be thought that if somebody programs a computer to behave in a certain way, then he *must* have a theory about the subject matter which is being modeled. But, such a theory need not be at hand for the model may be produced by a number of *ad hoc* programming techniques. For instance, Joseph Weizenbaum [1965] has created a program, ELIZA, which by using scripts can direct the computer to carry on conversations. If ELIZA is given the psychiatrist script, the program is called "DOCTOR" and can guide the computer in interviewing a patient. When the computer is running ELIZA, it can be understood as a model of a language user holding a conversation. As Weizenbaum ([1976], pp. 6–7) points out, ELIZA has been successful in that some people become so emotionally involved that they converse with the computer as if it were a person. Weizenbaum himself wishes to argue that ELIZA does not contain a general solution to the problem of computer understanding of natural language. ELIZA works primarily by a somewhat superficial analysis of semantic and syntactic cues and certainly does not provide a theory of human conversation. Weizenbaum ([1965], p. 143) states, 'ELIZA in its use so far has had as one of its principal objectives the *concealment* of its lack of understanding'.

Patrick Winston states a common view about programs and theories. Winston ([1977], p. 258) says, 'Occasionally, after seeing what a program can do, someone will ask for a specification of the theory behind it. Often the correct response is that the program *is* [Winston's italics] the theory.' My claim is that this is rarely, if ever, the correct response. Even if there is some theory behind a model, it cannot be obtained by simply examining the computer program. The program will be a collection of instructions which are not true or false, but the theory will be a collection of statements which are true or false. Thus, the program must be interpreted in order to generate a theory. In the process of interpreting, it is likely that some of the program will be discarded as irrelevant since it will be devoted to the technicalities of making the program acceptable to the computer. Moreover, the remaining parts of the program must be organised in some coherent fashion with perhaps large blocks of the computer program taken to represent specific processes. Abstracting a theory from the program is not a simple matter for different groupings of the program can generate different theories. Therefore, to the extent that a program, understood as a model, embodies one theory it may well embody many theories.

I do not wish to minimise the importance of computer modelling, and I am not suggesting that the theory always must be fully worked out before the modelling can begin, for one of the purposes of constructing models is to work out a better theoretical understanding. But at some point the theory must be statable independently of the computer model. Otherwise gimmicky programming can pass for good research.

In conclusion, I would like to emphasise that I am not claiming that the distinctions which I have discussed are myths, for indeed the distinctions are very important to computer science. But when they are misunderstood, they can generate myths. Under such misunderstandings the hardware/software distinction and the digital/analogue distinction are made to do more work than they can do properly, and the model/theory distinction is not made to do enough work. When these distinctions are conceptually spindled, folded, or mutilated, they don't compute.

Dartmouth College

REFERENCES

CHANDOR, A., GRAHAM, J. and WILLIAMSON, R. [1970]: *A Dictionary of Computers.* Baltimore: Penguin Books.

CLARKSON, G. P. E. [1963]: 'A Model of the Trust Investment Process', in Feigenbaum, E. A. and Feldman, J. (*eds.*): *Computers and Thought.* New York: McGraw-Hill Book Company, pp. 347–71.

DREYFUS, H. L. [1972]: *What Computers Can't Do.* New York: Harper & Row.

GUNDERSON, K. [1971]: *Mentality and Machines.* Garden City, New York: Doubleday & Company, Inc.

ISENBERG, C. [1976]: 'The Soap Film: An Analogue Computer', *American Scientist*, **64**, pp. 514–18.

LOEHLIN, J. C. [1968]: *Computer Models of Personality.* New York: Random House, Inc.

MOOR, J. [1976]: 'Analysis of the Turing Test', *Philosophical Studies*, **30**, pp. 249–57.

TURING, A. [1950]: 'Computing Machinery and Intelligence', *Mind*, **59**, pp. 433–60.

WEIZENBAUM, J. [1965]: 'ELIZA—A Computer Program for the Study of Natural Language Communication Between Man and Machine', *Communications of the Association for Computing Machinery*, **9**, pp. 36–45.

WEIZENBAUM, J. [1976]: *Computer Power and Human Reason.* San Francisco: W. H. Freeman and Company.

WINSTON, P. H. [1977]: *Artificial Intelligence.* Reading, Massachusetts: Addison–Wesley. Publishing Company.