SOCIAL EPISTEMOLOGY, 1990, VOL. 4, NO. 2, 135–154

Open peer commentary

Computationalism

ERIC DIETRICH

1 *Introduction*

Computationalism is the hypothesis that cognition is the computation of functions. If computationalism is correct, then scientific theories of cognition will explain it as the computation of functions. The job for the computationalist is to determine which functions cognition is, i.e. which specific functions explain specific cognitive phenomena.

A particular kind of computationalism – cognitivism – is the backbone of our current cognitive science (Haugeland 1978; Cummins, 1983). And, though we should, and will, come to eschew cognitivism as cognitive science advances (or so I claim), we will still embrace computationalism. In fact, I am inclined to think that even when cognitive science is broadly construed, computationalism is still its foundational assumption. If we consider cognitive science to be the attempt to explain phenomena ranging from recognizing food or a conspecific to proving Fermat's Last Theorem, and concerned with systems ranging from paramecia to societies of humans, we find that computationalism is still at its foundation. And, though it is not foundational in disciplines such as developmental neurobiology, and molecular biology, ecology, economics, and neuropsychology and developmental psychology, computationalism is part of their methodological repertoire.

Computationalism is only foundational and methodological. By itself, it makes no claims about which functions are computed, except to say that they are all Turing-computable (computationalists accept the Church–Turing thesis), nor does it make any specific claims as to how they got computed, except to say that the functions are systematic, productive, and interpretable in a certain way (see Section 2).[1] Computationalism, therefore, makes no specific predictions about any aspect of cognition beyond those implied by 'functionhood', nor does it explain any specific aspect of cognition. In fact, the computationalist thesis is compatible with a variety of more detailed theories of cognitive behavior which are themselves incompatible.[2]

Author: Eric Dietrich, Program in Philosophy and Computer & Systems Science, State University of New York, Binghamton, NY 13901, USA. I would like to thank Robin Hill for reading and commenting on an earlier draft of this paper. I have benefited from discussing the ideas presented here with the members of a New Mexico State graduate seminar on the methodologies for AI and the members of the 1988 History & Philosophy of Science Conference at Boulder, Colorado. Finally, I wish to thank Chris Fields for years of discussing these topics with me.

0269–1728/90 $3.00

Computationalism is an empirical thesis. It could be false – the only way to tell is to vigorously pursue a computationalist research strategy and see if we make any theoretical and empirical headway. But before this can happen, computationalism must be taken seriously. Taking computationalism seriously requires embracing, at least provisionally, all its consequences, and computationalism has some rather severe consequences. The purpose of this paper is to discuss two of these consequences. Specifically, I will argue that computationalism is incompatible with the notion that humans have any special semantical properties in virtue of which their thoughts are about things in their environment, and that the thesis is incompatible with the notion that humans make decisions. I am not, here, primarily concerned with whether computationalism is true or false, or explanatorily adequate or inadequate; that can only be determined by using it, and seeing if our cognitive science is better off with it than without it.

The two consequences just mentioned should be elaborated slightly. Computationalism claims that humans have no *special* semantic properties which attach them to their environment. Humans do indeed have such semantical properties but so do computers. More succinctly, if computationalism is correct, both humans and computers have *intentionality*. In fact, intentionality will turn out to be almost a commonplace. The second consequence is that humans have no special ability to make decisions, at least not when this term is given its ordinary meaning – as the capacity for willfully selecting among alternatives. Decisions are not made by any special exercise of the will.

As I formulate them, these two consequences have an interesting asymmetry. It is widely believed that humans possess intentionality but that computers do not (Searle 1980; Fodor 1981; Dretske 1985; Sayre 1986). It is also widely assumed that both computers and humans make decisions. If computationalism is correct, this is almost exactly backwards: both humans and computers possess intentionality, but neither make decisions. Intentionality is, therefore, not rare, and exercising one's will is so rare as to be non-existent.

2 *The nature of computationalism*

In this section I shall describe computationalism in some detail and distinguish it from two other theses about cognition.

2.1 Properties of computational explanations

Computationalists claim that explanatorily adequate theories of cognition will use computational explanations. Such an explanation has the following properties. First, it is an explanation of an ability or capacity of a system to exhibit certain behavior. Thus, a computational explanation differs from a causal law which describes the causal state changes of a system (for more details, see Haugeland (1978) and Cummins (1983)). Because the observed behavior is regular, the underlying capacity can be described as the capacity to compute a mathematical function from inputs to outputs. For many systems, we forego describing the relation between inputs and outputs as a mathematical function, opting instead for descriptions such as 'the system plays chess' or 'understands English'. Nevertheless, to be amenable to computational explanation,

a system's behavior must be describable as the computation of Turing-computable, mathematical function. Conversely, attributing a function to a particular system is seeing that system as doing something regular.

The requirement that the observed behavior be regular and due to a certain ability may seem too weak because *any* system which changes states (i.e. everything) can be described as computing a function – the function that describes its behavior. Yet we clearly do not want computation explanations to apply to everything; computationalism cannot replace physics. For example, we could see an object and a volume of water as a system which computes Archimedes's principle when the object is placed in the water. Or we could see a bicycle pump, a flat tire, and a pumping cyclist as a system which computes the ideal-gas law. But this seems wrong. We can explain why the object displaces the amount of water that it does by using Archimedes's principle, but we do not see the object–water system as actually computing a function. There are proposed solutions to this problem, but it is not clear any of them work. This problem need not concern us, though. There are clear, unproblematical cases where we explain a system's behavior by saying it computes a function. We will stick to such cases.[3]

The first property of computational explanations guarantees that capacities for exhibiting certain behavior are understood (by the explainers) as capacities for computing certain functions, and that exhibiting the behavior is seen as computing the function. The function so attributed to the system is 'system-sized'; it describes the behavior of the whole system.

The second property of computational explanations begins where the first leaves off. A computational explanation must by systematic, i.e. it must exhibit the system in question as, indeed, a system. To do this, the explanation must posit interdependent functions which interact to produce the output from the input. In other words, the explanation must analyze the system-sized function into subfunctions and show how the different subfunctions interact to produce the output (the behavior) in question (cf. Haugeland 1978). Moreover, the subfunctions must constitute a rather fixed set of functions out of which the larger functions are built. Procedures and standard computer programs are systematic in just this sense. In fact, one way of phrasing the second property is to say that, in order for an explanation to be computational, it must allow theorists to see the system's state changes as the execution of a procedure. (Computer programs, i.e. procedures implemented in some computer language, are also systematic, but they are not explanations in the sense used here; they are not theoretical explanations of behavior. It is easy to see this. Many different programs can implement the same procedure. If programs were theoretical explanations, then we would have an embarrassment of theories for each capacity we wished to theorize about, and we would have no reason for selecting one of these 'theories' over the other (see Dietrich (in press b)). Of course, all procedures must be couched in some language or other. But some languages are more scientifically useful than others. Computationalism needs a scientifically useful language, not a software engineering language. (Logics of various sorts and statistical mechanics are the currently preferred languages of computationalism.)

This is a good place to introduce the notion of *control flow*. The specific ordering arrangement of functions which are computed is called the control flow of the system's computation. In Section 4, this notion will be quite important. For now, I just want to note that the flow of control is fixed by the functions which the systems can compute and by the initial state of the system, i.e. its initial input. In other words, once a

procedure is specified (by a programmer or by evolution) then flow of control is determined (except for random events) by the initial state of the system.

The first and the second property together mean that the ability of the system to produce or exhibit some particular kind of behavior on a regular basis is the ability to execute certain procedures, and whenever the system exhibits the behavior in question it is executing the relevant procedure.

Third, a computational explanation is interpretive. This property of computational explanations is logically entailed by the first. The capacity for a certain behavior is manifested as a characteristic part of the actual behavior. In order to see (or describe) a part of behavior as the computation of a certain function, F, we, as theorists, must see the system generate, over time, output equal to F(input). However, in order to see this, we must be able to interpret the initial state of the system as being the requisite input for F, and the final state of the system as being the requisite output, F(input). So, computational explanations are inherently interpretive.

The three properties of computational explanations may be summed up as follows. Explaining a system's behavior as computing some function requires interpreting the initial and final states of the system as inputs and outputs for the proposed function, and then analyzing the proposed function into a sequence of subfunctions each of which carries with it its own interpretation of its initial and final states. Finally, whenever the system exhibits the behavior in question, it is computing the relevant sequence of subfunctions, i.e. it is executing a procedure. (All of the preceding is relatively well-known thanks to the work of Haugeland (1978) and Cummins (1983); see also Fodor (1965).)

The three properties just discussed do not fully capture the notion of a computational explanation. One other property is required. I call this property *productivity*: the functions whose computation is attributed to the system in question must be productive, a property which can be understood by contrasting it with non-productive procedures.

Consider two devices which take numbers as input and output their product. The question is: are both devices multipliers? Suppose the first device uses the well-known iterative-sum procedure: given 4×7, it totals up four sevens. Suppose the second device stores a two-dimensional, $n \times n$ matrix as in Figure 1. When the second device is given 4×7 it merely finds the product which is already present in the matrix.

Only the first device can potentially handle any posed multiplication problem because only it executes a multiplication procedure, hence only the first device is a multiplier. The second device is not a multiplier because it does not execute a multiplication procedure. In order for the second device to be useful, the products must be built in, and in order to do this they must already be known or computed. Hence, to build the second device, one requires the first device (or some other device that uses a productive multiplication procedure).

Of course, the second device is a computer, albeit a simple one, because it does compute functions: it computes matrix-look-up functions and it does this by executing matrix-look-up procedures. Hence, we can produce computational explanations of the second device because the procedure attributed to it is productive: it continues to work as the size of the matrix, is increased. But it is not a multiplier, and computational explanations attributing computing the multiplication function to it are wrong.

Some are tempted to say here the second device computes the multiplication function, but not by executing the iterative-sum procedure. Here is the argument. Both devices have a finite capacity for multiplying numbers. The first device only has a finite

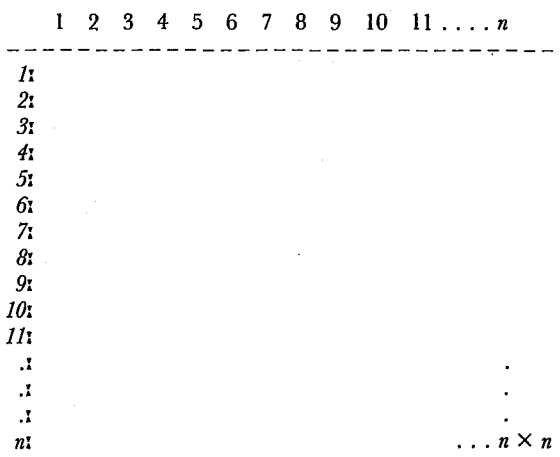

FIGURE 1. The n × n *two-dimensional matrix for the second device.*

(random-access) memory. Consequently, it can multiply only a finite set of numbers. The second device only has a finitely large matrix. Suppose that the behavior of the two devices is indistinguishable. That is, suppose that the second device's matrix ls large enough to accommodate any multiplication problem which the first device can handle. On this supposition, we can attribute the computation of the multiplication function to both devices, but only the first device is executing the iterative-sum procedure.

The temptation to argue this way must be resisted because we have little reason to attribute the computation of the multiplication function to the second device unless we have some idea of *how* it is computing it, and this requires determining how the multiplication function is analyzed into subfunctions. We must see the second device as executing some multiplication procedure, i.e. some procedure which productively multiplies numbers. Of course, we might *use* the device to compute the function just as we might use a calculator as a doorstop, but this does not mean calculators are doorstops. In general, we have little reason to attribute the computation of a function F to a system unless we can see the computation of F as systematic and productive (the second and fourth properties), i.e. unless we see the system as something which can compute F. An infinite number of functions can be attributed to a system if we are simply shown a finite sequence of its input–output behavior. Though this is also true when we systematically analyze the computation of F into component functions, we are nevertheless justified in settling on F as the function the system computes (instead of, say, G) once an understandable procedure begins to emerge which satisfactorily shows us that F explains the system's behavior.

The last claim commits me to the view that computational explanations are relative to what theorists understand and find satisfactory. But all scientific explanations, in fact, *all* explanations, are relativized this way. This relativization means that, in certain circumstances, we might view both devices as computing the multiplication function. If the devices' behavior really is indistinguishable (i.e. if they multiply exactly the same set of numbers in exactly the same amount of time, giving off exactly the same amount of heat, and we cannot open them up to run further tests) then we would find it irresistable to attribute the multiplication function to both, and it seems reasonable to do so. If this suits our other explanatory goals, then such an attribution is all the more

reasonable. But such a case would be extraordinary, and we should not reject the productivity property because of it.

We now, finally, have all the properties of computational explanations. The four properties work together to form an explanatory strategy which is scientifically respectable and robust. And now, we can be a little more rigorous.

The goal of the computationalist is to attribute to a physical system, S, the computation of a certain function, F. To do this, it must be determined (or assumed) that S computes F, and it must be determined how S computes F. Attributing the computation of F to S is successful when it is possible to explain S's computation of F in terms of a sequence of functions $\langle g_1, \ldots, g_n \rangle$ $(n>1)$ such that: (1) $F = g_n \circ g_{n-1} \circ \ldots \circ g_1$; (2) the sequence $\langle g_1, \ldots, g_n \rangle$ is productive; (3) S passes through a sequence of states each of which corresponds via an interpretation function I to either the domain or range of one of the g_i's, and each state between the first and final states is the range of some g_i and the domain of some g_{i+1}; and (4) we antecedently understand the g_i's. Succinctly, a computational explanation has the form described in Figure 2. When $F = g_n \circ g_{n-1} \circ \ldots \circ g_i$'s and the g_i's are non-trivial, it is natural to say that the sequence of functions $\langle g_1, \ldots, g_n \rangle$ *analyzes* the computation of F by S and explains the capacity of S to compute F (cf. Haugeland (1978) and Cummins (1983: 28–44)).[4]

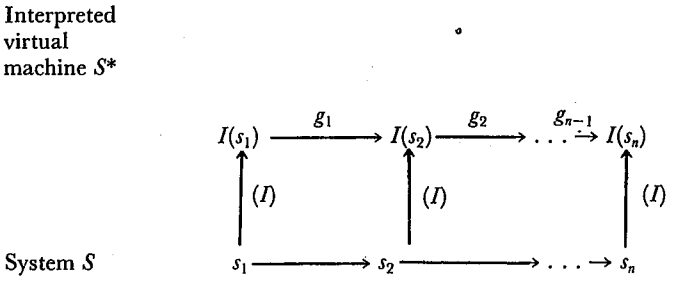

FIGURE 2. Interpreting the state transitions of S *as computing the sequence of functions* $\langle g_1, \ldots g_n \rangle$.

2.2 *Computationalism versus other '-isms'*

Computationalism is distinct from both 'computerism' and cognitivism. 'Computerism' is the thesis that explanations of cognition will look like procedures for our current (late 20th century) computers. Computerism is thus tied to a specific computer hardware architecture, in particular a serial architecture. Note that the computerist is interested in more than the functions which get computed; she is interested in how they are computed. It is not clear that anyone actually believes computerism, but frequently, when computationalism, artificial intelligence, and computational psychology are attacked, it is 'computerist' notions that are attacked instead (see, for example, Carello, *et al.* (1984) and Reeke and Edelman (1988).

Almost all computers today are Von Neumann machines, meaning that they compute functions serially, executing one program instruction at a time. To claim that some machine is a Von Neumann machine is to make a claim about its *architecture*, and, hence about the kinds of procedures which can be written *directly* for that architecture (i.e. what kinds of procedures can be written in the *assembly language* of that architecture).

Specifically, assembly language procedures must be constructed using combinations of these three kinds of control: looping, branching, and sequencing (this is true of

almost all programming languages, also.[5] These three kinds of control, in turn, limit the kinds of assembly language instructions which may be used: the instructions must be simple, deterministic, and discrete. So, the computerist believes that the procedures relevant to explaining cognition use sequences of simple, deterministic instructions. Hence, for the computerist, the g_i in Figure 2 would be instructions like those we find in assembly language computer programs which exist today: simple, deterministic in structions such as REP MOVS DEST, SRCE (move what's in SRCE to location DEST) and STOS DISPLAY (store what is in a specific register at DISPLAY) which are instructions for the Intel 8088 microprocessor (Intel 1983).

Computerism is false on empirical grounds: the brain is not a Von Neumann machine, not even approximately. In fact, when seen as attacks on computerism, the works by Dreyfus, Carello, etc., are compelling. The mathematical functions which the computationalist believes explain cognition can only be tortuously described by simple, deterministic instructions, at best. Put simply, computerism adopts the wrong kind of description language for the phenomena it is supposed to describe because it assumes an impoverished explanatory ontology.

Computationalism as depicted in Figure 2 is neutral on the Von Neumann–non-Von Neumann issue. Computationalism is not tied to any specific architecture, nor is it committed to the view that procedures which are composed of simple, deterministic instructions explain cognition. Computationalism is compatible with massively parallel architectures such as those studied by the connectionists (see Bechtel (1988) and Smolensky, (1988)), and those studied by neural modelers such as Grossberg (1987) (see Dietrich and Fields (1988)).

Cognitivism is the thesis that the functions which explain cognition are rational functions defined over proposition (or sentences). Rational functions relate propositions in an epistemologically appropriate way. Thus, for cognitivists, the objects which are computationally manipulated are propositions (or sentences), and it is propositions which are the inputs and outputs of such systems (this is why some cognitivists do not consider early perception part of cognition). Cognition is the production of output propositions which are rationally related to input propositions. For the cognitivist, inference is the paradigmatic cognitive function. For example, Cummins (1983) says: '. . . cognitive capacities are inferentially characterized capacities . . .: the transition law specifying a cognitive capacity is a rule of inference' (p. 53). Cognitivism is far away the most prevalent, general hypothesis about cognition. (Cognitivism and its prospects has been discussed in detail by Haugeland, (1978, 1981); see also, Cummins (1983) and Cummins and Schwarz, (1988).)

Computationalism is not cognitivism. Computationalists are not commited to the claim that cognition is inferentially processing propositional or sentential structures. Nor are they committed to the claim that rationality or epistemological adequacy is the fundamental relationship between inputs and outputs of cognitive systems. Rationality plays a major role in cognitivism, but it plays only a minor role in computationalism, and even then, it is a special type of rationality (see Section 4).

Cognitivism will be false if it turns out that the most important cognitive functions are not inferences. Yet in such a case, computationalism could still be true. This would happen if thinking turned out to be, for example, manipulating algebras, vectors, or automata of various types. In such cases, the functions explaining cognition would be algebraic morphisms, operation on vectors, or operations on automata, respectively. This is not mere hand waving, either. It is plausible that at least one kind of creative thought – the 'a-ha' experience – is *not* an inference at all, and in fact could not happen

in a system only capable of inferences (Dietrich and Fields 1986; Fields and Dietrich 1987b). In general, theories of cognition couched in *mathematical* languages will be compatible with computationalism, but not compatible with cognitivism. Mathematics is not the science of inferences; 1 + 1 = 2 is not a rational inference.

Most philosophers balk at this, utterly. For example, Cummins says

> What makes a [cognitive] capacity [cognitive] is surely that its exercises are epistemologically assessable, and this commits one to the view that such exercises are amenable to sentential interpretation. . . . Talk of cognition in the absence of actual or sentential interpretation of inputs and outputs is mere hand waving. (Cummins 1983: 198–199).

The limits of a philosopher's imagination are frequently where he or she uses the word 'surely'. The view that thinking is basically inferencing is a vestige of logical positivism – one of philosophy's dark ages. Computationalism offers ontological and methodological riches unimagined by the cognitivist, which we will squander as long as we believe thinking is inferencing.

Computerism and cognitivism differ from computationalism along two different dimensions. Computationalism, again, is the claim that cognition is the computation of certain Turing-computable functions which are to be determined by cognitive *science*. Cognitivism is a further claim about *which* functions are computed. It is the claim that the functions are inferences of one sort of another, that the objects which the functions process are sentence-like propositions, and that rationality is a fundamental property of cognition. Computerism, however, is a claim about *how* the functions are computed, i.e. it is a claim about what the architecture of the brain (or maybe the mind) is. All three claims are empirical and are based on various combinations of evidence, theoretical assumptions, 'what-else-could-it-be' arguments, wishful thinking, and lack of imagination, and all three could be false (i.e. the three do not exhaust the space of cognitive theories). Only computationalism manages to keep its claims modest by tying them to the evidence, and to avoid premature commitments to theories and research strategies. It does this by strictly adhering to the notion of computation found in computer science, and letting evidence determine both which functions are computed and how they are computed. In spite of its modesty, however, computationalism has (at least) two strong consequences which are explained in the following two sections.

3 *Intentionality*

Intentionality is sometimes defined as the property of mental states to be about things (Haugeland 1978, 1981: 32; Sayre 1986). The things need not exist (for example, one can have thoughts about the Grail), nor do the things need to be logically possible (one can have thoughts about round squares, for example – one can think they do not exist). When understood in this way, intentionally is a semantic notion: intentionality is the psychological property of having semantic content.

At other times, intentionality is defined as the property of a system to understand its own representations (Searl 1980; Fodor 1981; Dreyfus 1982; Follesdal 1982; Cummins 1983; Dretske 1985; Haugeland (1981) also discusses this definition of intentionality under the term 'original intentionality'). This notion of intentionality is, or is very close to, the notion of a system consciously understanding the world around it.

However it is defined, intentionality is regarded as a crucial aspect of cognition (but see Stich (1983)), and, therefore, it must be explained if we are to understand cognition

thoroughly. Since computationalism purports to be a theoretical framework for cognitive theories, the question naturally arises as to how computationalism should deal with intentionality. This question is a matter of some importance, since computationalism is widely regarded as being *incapable* of supporting any explanations of intentionality. Arguments purporting to demonstrate computationalism's deficiency in this matter have been offered for a variety of intentional psychological phenomena. Examples include understanding a language (Searl 1980), perceiving an environment (Sayre 1986), and, finally, thinking itself (Searle 1980; Dretske 1985). In short, computationalism is considered to be incapable of producing explanations of the phenomena we intuitively regard as genuinely mental, and is, therefore, rejected.

3.1 Intentionality as a semantic property

In this section I consider how computationalism copes with intentionality defined as a semantic property of mental states. If computationalism is true, mental states have semantic content. In fact, providing a computational explanation of a system's behavior *requires* attributing semantic contents to the system's states. Recall that computational explanations require interpreting the inputs, outputs, and states of a system as elements in the domains and ranges of a series of functions. In fact, computationalism provides us with a *strategy* for semantic interpretation. I call this strategy the *computational strategy*, and the way in which it works is described briefly below (for more details, see Dietrich in press a).

Recall Figure 2, when S passes from state s_i to s_j, function g_i is computed and its output is then input for succeeding function g_j. We (as theorists) understand the state transition of S from s_i to s_j by seeing the transition as the execution of $g_j i$. Doing this is just interpreting the states of S because it is treating the states of S as symbols which are transformed. When we do this, we see S not merely as a physical system, but as an interpreted virtual machine, i.e. as a system S^* that computes F by passing through a sequence of virtual states which are the inputs and outputs of the g_i terms (for the notion of a virtual machine, see Tanenbaum (1984). The correspondence between the S states and the g_i is made precise by the interpretation function, I, which maps states of S on to the g_i. Once we can view S as an interpreted virtual machine, we can switch between this view and the physical state transition view (cf. Stabler 1983).[6]

There are five pertinent consequences of using the computational strategy in order to attribute semantics to systems. The two most important consequences concern psychology, computer science, and philosophy of mind. First, if psychology is to embrace the computational paradigm, it must (*contra* Stich (1983)) ascribe contents to mental states and processes because ascribing contents is necessary for understanding which function is being computed by the psychological processes in question, and understanding this is necessary for understanding the behavioral and psychological capacities of the system. Secondly, we cannot view computers as merely formal symbol manipulators and understand their behavior. That is, we cannot view computers as merely syntax machines performing their computations on the basis of the syntactic properties of the symbols they manipulate. In order to understand their behavior we must interpret their states (this is well-known to computer scientists, see Wulf *et al* (1981: chap. 5) and Stoy, (1977)). This is quite contrary to the received dogma. The view that computers *are* formal symbol manipulators and that we can understand them as such is the prevalent view. This view has allowed philosophers to divorce semantics

from computational explanations. Semantic content, then, becomes something one adds to computational explanations to get psychological explanations (Searle, 1980). Other philosophers have claimed that we can get by without semantics at all (Stich 1983). If computationalism is correct, both these views are wrong (for more on these two consequences, see Dietrich (forthcoming, a)).

A third consequence is that there are no arbitrary or extra-theoretical restrictions placed on the contents that can be attributed to a system. The only general requirement is that the attributed contents explain observed behavior and hypothesized cognitive capacities. This renders otiose some well-known relations that are supposed to be important to semantic contents. For example, the causal connections of referring terms have no role in the computational strategy. Hence, we can simply avoid issues such as what the causally correct referent of a referring term is. Which function a system is computing is the only matter of importance because it is this that determines the contents of its states.

Causation, in general, is relegated to a supporting role from the computationalist's view. This is a welcome result because causation cannot do the work some philosophers think it can. Some philosophers seem to think that computers are not causally connected to the world, or at least not causally connected in the 'right way'. (This is one of the things Sayre taxed us with missing in our commentary on his paper. See Sayre, (1986, 1987) and Fields and Dietrich (1987a).) The 'right way' is frequently expressed in terms of information theory, but this just will not work. The information computers get from the world when described by information theory is as real as the information our perceptual systems obtain. Computers are as 'causally embedded' in the world as humans are. So, as far as information theory is concerned, humans are not causally special. Computationalists can accommodate this result easily: the role of causation is to describe the physical-state changes and properties of the system S (see Figure 2).

A fourth consequence is that attributing a particular content to a particular mental state (or state of a system) is *not* paramount, as it is in other strategies for semantic attribution. The computational strategist wants to understand systems. Semantic contents are thus viewed in the context of entire systems. On the computational strategy, no mental state, indeed no symbol whatsoever, is (usefully) interpreted in isolation. Rather, whole systems of states must be ascribed contents so that a cogent explanation results.

A fifth consequence is that the attributing semantics via the explanatory strategy is not a folk art or a matter for causal speculation. One must be intimate with the systems under study in order to attribute contents that are scientifically useful. This is just another way of saying that understanding and attribution are achieved concommitantly.

Finally, note that when intentionality is defined as a semantic notion it is nearly ubiquitous. If a computational explanation explains a system's behavior, then that system's states have semantic content. All kinds of systems, from humans to the lowly and oft-maligned thermostat, will have contentful, computational states.

3.2 Intentionality as a system's understanding of its own representations

I will now consider how computationalism copes with the definition of intentionality as the capacity of a system to understand its own representations. (In this section I use the word 'intentionality' to mean only the notion of a system understanding its own

symbols.) This is the most well-known notion of intentionality, and it is the notion discussed by Cummins (1983) and used by Searle (1980) and Dretske (1985) in their arguments that machines cannot think.

In his famous and important paper on intentionality, Searle says:

> ... the programmed computer does not do "information processing". Rather, what it does is manipulate formal symbols. (Searle 1980: 303).

For the computationalist, the phrase 'formal symbol' is an oxymoron. However, for Searle and others of his ilk, a formal symbol is something which is a contentful symbol to us, but a contentless object to a computer or other syntax machine. In his equally stimulating paper, Dretske (1985) states

> To understand *what* a system is doing when it manipulates symbols, it is necessary to know, not just what these symbols mean, what interpretation they have been, or can be, *assigned*, but what they mean to the system performing the operations. (Dretske's emphasis; Dretske 1985: 27).

Similarly, Cummins says when a system has intentionality

> the representations in question are representations for (or to) the system that has them, and not merely for (or to) a user or theorist ... (Cummins 1983: 76).

Intentionality is thus the unification of meaning and manipulation: the symbols being manipulated mean something to the system doing the manipulations.

What is intentionality for? What good is it? Nearly everyone writing about intentionality assumes that intentionality is crucial for cognition. Humans and other genuine cognitive agents do not have formal symbols. A human's mental symbols have meaning not only to theorists such as psychologists, but to it, itself, and it is because of this, apparently, that humans can think. But no one has actually shown that intentionality is crucial for cognition. Searle certainly has not. At best, he has shown that the Chinese Room lacks intentionality. But his argument depends on the behavior of the room being indistinguishable from the behavior of a genuine Chinese speaker. Searle has succeeded in showing that intentionality is useless.[7]

Before proceeding, I want to point out that intentionality is supposed to be different from consciousness or conscious understanding (see Searle (1980) and Dretske (1985: esp. p. 30)). This is important for what follows because intentionality is problematic exactly to the extent it is thought to be different from consciousness. If intentionality were just another word for consciousness, most of us could at least agree that it exists, though we still would not know what it is, nor what it is for. I also want to mention again that computers are supposed to lack intentionality, i.e. their symbols do not mean anything to them.

We can question the very cogency of the notion of a system understanding its own symbols or representations. First, it is a notion that is supposed to apply to me as a thinking creature. But I can assure you that I do not understand my own symbols. I understand the symbols I am currently writing now, but these are not mine; they are on this page and are public symbols. I can introspect, but intentionality is supposed to be more general and ubiquitous than introspection, and, anyway, computers can introspect. So, it is not clear that intentionality is a notion that applies to me while not applying to computers.

Secondly, it seems as if no system could have intentionality on pain of generating an

infinite regress. If understanding involves symbols, then if a system understands its own symbols it is using symbols. Does it understand these 'second-order' symbols? If not, then the system – the whole system – lacks intentionality because it does not understand its own symbols, some of them, anyway. If it does understand its 'second-order' symbols, then it must understand its 'third-order' symbols, or, again, it lacks intentionality. It follows that a system must understand an infinite hierarchy of symbols. However, it is not obvious that any sort of physical system can do this.

An obvious objection to my argument is that a system need not understand all its symbols; it need only understand a few of them at a time – the ones on which its attention is currently focused, for example. This objection is rather plausible, but it has two problems for believers in intentionality. First, computers can generate hierarchies of symbols as well as processes (Dietrich 1985). So, once again, intentionality applies equally to humans and computers. The believers in intentionality would counter by insisting that intentionality is more than the capacity to generate hierarchies of symbols. It involves the notion of special, internal understanding, and computers lack such an understanding. However, now it seems as if the notion of consciousness, not a separate notion of intentionality is doing the real work in this objection. In fact, it seems to me as if this latter claim relies on the notion of *self*. Believers in intentionality seem to be saying that a system must have a concept of itself as an enduring whole in order to have intentionality. This also seems plausible to me, but is clearly depends on notions which philosophers such as Searle and Dretske regard as completely independent of intentionality.

Let us get our bearings. It seems as if intentionality either does not apply to humans, or applies equally to computers and humans, or is a hodgepodge of notions such as consciousness and the self masquerading as a single concept. I think that intentionality is really a masquerading hodgepodge, and the hodgepodge comprises the concepts we should really be interested in. Here is my argument. We can dismiss any notion that intentionality does not apply to humans. This leaves the second and third options. There is a robust notion of 'understanding your own symbols' which applies to humans and computers (I will show this shortly). However, believers in intentionality will reject this as not being what they mean by intentionality. Hence, consciousness, etc., must be the real notions. I will now show that there are processes within operating computers to which internal symbols have meaning. If I am right, then computers are not merely formal symbol manipulators, but computationalism can easily accommodate part of the notion of a system understanding some of its internal symbols.

It seems to me that the ordinary notions of a variable, (variable binding and variable substitution) suffice to establish the claim that computers are not merely formal symbol manipulators. In an obvious sense of the term, to manipulate symbols in a purely formal manner is to manipulate them without regard to what they refer to, mean, or denote, nor must the manipulations depend on the fact that a symbol has a meaning or denotation. For example, manipulating a symbol solely by virtue of whether it is a token of some numeral, letter, or part of speech is one way to treat a symbol purely formally.

Consider the notion of a variable in this instance of the Lisp function '+': (+ x 1). Loosely speaking, this function adds 1 to whatever x is bound to. However, for this argument, it is important that we be more precise about what the entities under discussion are. A computer running a Lisp interpreter (the Lisp execution program) defines a virtual machine called the Lisp virtual machine (LVM). The LVM operates solely in terms of Lisp expressions, the syntax for which can be specified by a grammar;

no numbers or other 'external' objects are involved. (I will mention Lisp expressions by placing single quotes around them. The primary reason for couching the argument in terms of an LVM is that it will be easier to understand. Nothing turns on this. The same argument could be made at the bit level, though at this level the argument would be all but lost in the detail.)

The LVM takes as input expressions such as '(+ *x* 1)', evaluates them, and returns expressions as outputs. We interpret the inputs, outputs, and intermediate expressions as, for example, computing (an instance of) the function plus, and producing the number 7 as its value. We also attribute the semantic content 1 or 'representing the number 1' to Lisp expressions such as '1'. These interpretations are enhanced by making the syntactic form of the Lisp expressions look like expression in languages we already know.

Evaluating the expression '(+ *x* 1)' requires the LVM to determine the value of the variable '*x*', which is some other Lisp expression, say '6'. If the LVM could not do this, the expression would be syntactically ill-formed: '+' is not defined for expressions we interpret as non-variable letters. But note that the LVM itself, in treating '*x*' as a variable, regardless of our interpretations (which are, in fact, quite different), is treating '*x*' as denoting the expression '6'. It follows from this that the LVM treats '*x*' as having a meaning. Hence, the operation of the LVM depends on '*x*' having a meaning for the LVM, and not just for us. Of course, the meaning '*x*' has for the LVM (*viz.* '6') is not the meaning '*x*' has for us (we typically interpret '*x*' as representing the number 6, not the Lisp expression '6'), and the LVM is not conscious of the meaning '*x*' has. Nevertheless, the LVM's manipulations depend on the fact that '*x*' has a meaning, and, indeed, on the meaning that it has. This is enough to make false the claim that computers are formal symbol manipulators, at least on the straightforward interpretation of this claim I have assumed.

To sum up, I have shown, that: (1) computers are not merely formal symbol manipulators because they 'look up' the values of variables, and anything capable of doing this is also not merely a formal symbol manipulator; (2) since computers are not formal symbol manipulators and since computational explanations explain the behavior of computers as *genuine* symbol manipulators, computational explanations can explain the behavior of systems which are more than mere formal symbol manipulators; and therefore, (3) computational explanations can explain much more of human behavior than is commonly believed. If looking up the values of variables captured the notion of intentionality satisfactorily, then we could see why intentionality would be important for cognition. A cognitive system cannot have a function for every situation which might arise in its environment, so a few functions must have wide applicability. This is accomplished by having variables and variable binding and look-up procedures.

I want to close this section by returning to consciousness and related notions. As I said, I suspect that the notion of intentionality studied by Searle, Dretske, and others is a hodgepodge of other notions and intuitions, some of which we want to maintain. Consciousness is certainly one of these notions. Humans, but not computers, are conscious; we are aware of some of the states and contents of our own minds. Personally, I like Nagel's notion of consciousness (1974): there is something it is like to be a human, but being a computer *seems* as if it would be like being an intelligent rock. Consciousness, whatever it is, certainly needs explaining. Of course, computational explanations would be entirely appropriate (see Dietrich (1985)). Another notion is intelligence. Humans are simply smarter than computers. In fact, computers occupy a

new class in the intelligence hierarchy; they are a new class of idiot savants. Expert systems demonstrate this clearly. They are less intelligent than snails on virtually every dimension we care about, yet are smarter than most humans on a few special dimensions (for more on this, see Harnad (1989)). Finally, in a fascinating paper, Haugeland (1979) argued that computers will not succeed in understanding natural language until they can be given (or otherwise develop) a sense of the world they inhabit, the creatures they interact with, and, most importantly, a sense of themselves as enduring wholes. Those that think computers are merely formal symbol manipulators (and, hence, that computationalism is too weak) may in fact be noticing that computers are not conscious, not very intelligent, and are not selves.

If intentionality is only a semantic property, it is virtually ubiquitous. If intentionality is variable binding and look-up, then it is quite common. If intentionality is consciousness, etc., then it is quite rare. The literature on intentionality defines it as one of the first two notions, but reading between the lines, consciousness and related notions are the real phenomena of interest.

4 *Making decisions*

Computationalism is incompatible with our ordinary, day-to-day view that humans make decisions. Hence, for a computationalist, decision-making is not a cognitive capacity. In fact, if humans really do make decisions in the way we ordinarily think decisions are made, then computationalism is false.

On the ordinary view, humans and other intelligent systems frequently decide to take a certain course of action or to form a certain intention. Of course, not all of our actions or goals are arrived at by deciding, but some are. A few months ago, I decided to write this paper; I decided to work on it today. But as I type this paragraph I am not deciding to breathe or to maintain tonus; these are done automatically.

The computationalist wants to, and should want to, maintain the distinction between the two kinds of action just mentioned. However, whereas both our folk psychology and our current cognitive psychology couch the distinction in terms of deciding and not deciding, the computationalist couches it terms of the kinds of procedures executed. This makes all the difference in the word. I will describe this class of procedures shortly; for now, let us consider ordinary deciding.

Deciding is, I think, most naturally seen as the exercise of the will. Typically, the system has three or four options before it, and it willfully chooses the one which has the highest score provided by a process of evaluating the options along some dimension (or dimensions). Generally speaking, the dimensions measure the desirability of the outcomes produced or their ability to satisfy some previously set goal or established intention. For example, I can continue to write or I can go to watch football on television. I ponder over these options. I would rather go to watch football; it is relaxing and fun. Writing this paper is not relaxing. However, writing this paper is fulfilling in a way that watching football is not. I continue to ponder. Aesthetics, relaxation, and enjoyment turn out to be secondary. I have a duty to write this paper: a duty to myself (for my career and philosophical integrity), to specific others (those I have made promises to regarding this paper), and unspecified others (the philosophical and cognitive science communities at large). I sum everything up and 'see' that my duties outweigh my desires for fun and relaxation, so I choose to work on my paper. My will enters here in the last step. Once I see which option has the highest score, I am not

thereby destined or forced to work on my paper. I must willfully choose to work on my paper. That this is true can perhaps be seen more readily if we suppose that I am one of those who generally ranks duties below enjoyment no matter what my duty. In this case, the fact that I must willfully choose to work on my paper is more apparent.

As I just mentioned, will can be exercised with varying intensity. Suppose I see that my duties are not all that strong. Suppose I already have plenty of research grants and publications, my promises were 'weak' promises or promises with unspecified dates, and the philosophical and psychological communities at large already accept the scientific bankruptcy of the notion of deciding. In such a case, my desire for fun might get the highest score, so I choose to watch football. Even in this case, I exercise my will. For example, I might be a workaholic and have to make an effort to relax. But even if I am one of those who is disposed to relax and have fun when I have no other pressing duties, I must still exercise my will in order to watch football on television, though in this case, I need not exercise my will very much. If I do not exercise my will, I will just sit here in front of my monitor maintaining tonus and breathing, typing nothing and doing nothing. What is the will? No one knows (though not from want of trying). Perhaps one day psychologists will discover what it is. Perhaps one day artificial-intelligence researchers will be able to program will into a computer. Perhaps we will never know (cf. Fodor (1983)). But humans clearly have wills, and they exercise them frequently in the course of their daily lives.

This, I submit, is the ordinary view of human decision-making – and, it is the view which gets extended to cover all other intelligent systems from ants, to cockroaches, to dolphins, to chimpanzees, and to computers.

Computationalists have a different view of deciding. Their view is that decisions are the computation of branching functions. Branching functions map expressions constructed using computable, boolean functions (which are called *conditions*) onto some other computable function. Thus if F is set of computable functions, we have:

$$B\text{: } \{\text{conditions}\} - - - \rangle F.$$

The action of B is then

$$B(\text{condition}) = \begin{cases} f_1, \text{ if condition is true} \\ f_2, \text{ if condition is false} \end{cases}$$

where both f_1 and f_2 are elements of F. In computer science, branching functions are typically rendered as IF statements (which are procedures)[8]

$$\text{IF } \langle \text{condition} \rangle \text{ THEN } f_1$$

$$\text{ELSE } f_2^8.$$

At this point, we need to recall the notion of *control flow*. As mentioned in Section 2, the specific sequence of functions which get computed is called the control flow of the system's computation. In *sequential control flow*, no branching functions are executed: computing the function f_i is always a sufficient condition for computing the function f_{i+1}. (Here, we must assume that the system is working properly and that it is not a stochastic system, i.e. that its state changes are not probabilistic relative to our explanatory goals.) Described at the state level, we can say that being in state s_1 is always sufficient for entering state s_2 (again, assuming the system is working correctly and is not stochastic). However, branching functions can change the flow of control in a system's procedure execution. Computing a branching function B does not always

result in next computing a specific function. Depending on the values of the relevant variables, computing B could result in computing f_1 next or f_2 next. However, as also noted in Section 2, once a procedure is specified, flow of control is determined by the initial state of the system, and this is true even if the procedure contains branching statements. In fact, given a branching function B, its condition C, and c's input i, $B(c(i))$ uniquely determines an output function f which is then simply executed automatically. Computationalism is incompatible with the notion that cognitive agents such as humans decide precisely because there is nothing for the will to do. The automatic nature of control flow and the notion of branching functions capture everything there is to the notion of deciding.

I now return to my decision to work on this paper instead of watching football on television. To the computationalist, I computed some branching function which determined that my duties outweighed my desire for fun and relaxation. That is I executed this procedure:

> IF (duties outweight desire for fun),
> THEN work on paper,
> ELSE watch football.

Given that my duties do outweigh my desire for fun, I simply compute next the work-on-paper function or, more precisely, the first function in the sequence of functions which constitutes my work-on-paper function and which explains my working on my paper as I do. Notice that there is no need for will. Once the branching function computes the boolean expression which makes up the condition ⟨duties outweigh desire for fun⟩ I compute the next function automatically, just as the computer does.

But what about the case where my duties rank lower than my desire for fun, yet I choose to work on my paper instead? Will seems required to explain this. In the computationalist view, however, if the procedure mentioned above adequately explains my behavior and I work on my paper instead of watching football, then I have not, in fact, ranked my duties lower than my desire for fun. However, does this mean that computationalism cannot take seriously the distinction underlying the above question? Some decisions do seem harder to make than others. Computationalism either ought to account for this or to show that the supposed phenomenon is an illusion. I believe that computationalists will be able to explain this phenomenon. Of course, the best way to argue this point would be to produce such an explanation here. But something less will do, too – I need only show that computationalism *can* explain this phenomenon which I can do by producing a plausible explanation.

Some function computations are *goal-driven*. A goal-driven computation is just like an ordinary computation (as in Figure 2) except for the way in which the sequence of functions came into existence. For goal-driven computations, the system itself builds the sequence (actually some subsystem builds the sequence, but we can be relatively sloppy about this point). The constraint on building a sequence is that the sequence should, when executed, result in achieving the goal. (Goal-driven computations are well-known in artificial intelligence (see Rich (1983: 57 ff.) for an introduction. Goal-directed computation is also known as top-down processing, expectation-driven processing, and backward reasoning, etc.)

In general, in a goal-driven computation, whether or not the goal should be achieved is not open to debate. The goal results in a sequence of functions, control is passed to the first function in the sequence, and the goal is achieved (assuming the sequence is in fact capable of producing the goal). However, sometimes a system has 'dueling goals', i.e.

different goals which compete against each other for current resources. Suppose there are two kinds of dueling goals: candidate goals and interfering goals. In both kinds of goals, goals compete against each other. The difference is in how genteel the competition is. Candidate goals are goals which, as it were, agree to abide the decision of a branching function – rather like polite political candidates, or factions in a legal dispute. Interfering goals, however, are not 'willing' to abide by the decision of a branching function. They, as it were, influence the voting. Such goals are like heads of rival criminal organizations of corrupt political adversaries. Specifically, interfering goals create looping branching functions.

Returning again to my decision to work on this paper. Suppose that I have dueling goals: i.e. to work on this paper or to watch football on television. In the case where I fairly easily decide to work on this paper, my two goals are candidate goals. My *goal adjudication system* takes the two goals as input and constructs the following branching procedure (which is quite similar to the one described above):

IF (duties outweight desire for fun),
THEN Activate: work-on-paper,
ELSE Activate: watch-football.

This branching procedure is executed, and the winning function, 'Activate: work on paper', say, activates my goal to work on my paper. This, in turn, spawns the appropriate sequence of functions, and I do indeed work on my paper.

However, suppose that my goals are interfering goals. Their interference causes my goal adjudication system to create a looping branching procedure:

IF (duties outweigh desire for fun),
THEN decrease the importance of my duties
  and re-execute this procedure,
ELSE increase the importance of my duties
  and re-execute this procedure.

The watching-football goal is responsible for the THEN clause, and the work-on-paper goal is responsible for the ELSE clause. The difficulty of the decision depends on the intensity of the increase/decrease war implicit in the branching procedure. In the worst case, every decrease could be met by an increase of exactly the same amount, and an infinite loop could result. Then my *emergency looping branching procedure repair system* would have to be called (the calling condition would be something like 'branching function has looped more than 10^6 times'), and it would be responsible for taking control away from the looping branching procedure, and trying to restore order. In the very worst cases, my emergency looping branching procedure repair system might simply have to 'flip a coin' and pass control to the winner while actively preventing control from being usurped by the loser. When viewed from my conscious level, I could very plausibly describe all this as 'agonizing over my decision whether to work on my paper or to watch football', and as 'exercising my will to work on my paper'.

We can now, perhaps, finally abandon the notion of willful decision-making. With the demise of willful decision-making goes any robust notion of a person scanning an array of options and choosing one. Humans do not choose, they merely compute. The procedures we execute are extraordinarily complex and quite plastic, but like any computational mechanism, we decide to do some things and not others entirely on the basis of our initial state and the branching procedures we execute.[9]

5 *Conclusion*

In this paper, I have defined computationalism, shown that computationalism makes any distinct notion of intentionality widely applicable, and I have shown that computationalism is incompatible with our ordinary notion of willful decision-making. From the computationalist perspective, humans are very different from what we thought. Thus, computationalism is incompatible with folk psychology. Given the batting average of folk theories, this a point in the favor of computationalism.

Notes

1. A brief discussion of algorithms, procedures, functions, and the Church–Turing thesis is required. Computer scientists distinguish between *effective procedures* and *algorithms* (e.g. see Brainerd and Landweber (1974). An effective procedure ('procedure' for short) is a finite, unambiguous description of a finite set of effective operations. An operation is effective if there is a strictly mechanical method for executing it. (This is as precise as the definition can be made, which is why the Church–Turing thesis cannot be proved). All computer programs are (effective) procedures.

   Depending on what input they are given, procedures will halt in a 'yes' or 'no' state in a finite amount of time or go into an infinite loop. If a procedure always halts no matter what input it is given, then it is called an *algorithm*. In other words, an algorithm never enters an infinite loop; instead, it always produces a definitive answer to an input question, though it need not produce the right answer. Only some computer programs are algorithms. In this paper, 'procedure' will be used most of the time because we have no empirical evidence that any cognitive procedure always halts.

   The Church–Turing thesis says that Turing machines can compute any function for which an effective procedure exists. The converse of this statement is known to be true. Computationalists are committed only to the claim that cognition is the execution of procedures. This is the weakest claim compatible with computationalism. It is this claim and the Church-Turing thesis that commits them to the further claim that all cognitive functions are Turing computable. Computationalists need not, and should not, commit to the stronger claim that cognition is the execution of algorithms because we have no evidence that every procedure which humans execute always produces a definite answer.

   Computer scientists frequently distinguish between computing a function and executing a procedure because every procedure realizes exactly one function, but each function can be realized in several different procedures. For example, the function *2*x* which doubles any number can be realized as an procedure which adds x to itself, or as an procedure which multiplies x by 2. Another example, is the function *sort A* which sorts an array of items from the lowest to the highest (e.g. if the items are character strings, *sort A* sorts them into alphabetical order). *Sort A* could be realized using the bubble sort procedure or the selection sort procedure. However, this distinction is not too important in computationalism because computationalists must *analyze* the functions they attribute to systems into subfunctions (see Section 2). Doing this forces them to view functions as built from certain subfunctions in certain ways, hence they are forced to view functions as procedures. Therefore, I shall use the words 'function' and 'procedure' more or less interchangeably.
2. For example, there are competing theories of how humans make analogies. One kind of theory claims that analogies are made by accessing pairs of representations (or data structures) in memory which denote relations such as causal relations (see, e.g., Schank (1982)). Another kind of theory claims that the accessing strategy is rather wanton, accessing almost any representation it can regardless of what it denotes (see, e.g., Dietrich and Fields (1986), Gentner and Landers (1985) and Gick and Holyoak (1983)). On the former theory, when required to do so, humans should by and large produce only a few plausible analogies from which a 'best' analogy is selected. On the latter theory, when required to do so, humans should produce a large number of candidate analogies many of which will be spurious. These theories make incompatible predictions, yet both are compatible with the computationalist thesis. Indeed, both kinds of theory are couched in computationalist language.
3. This is not a cop out. All sciences are beset by the problem of how to carve the world in order to get systems for which good explanations are forthcoming. The current problem is merely computationalism's version of this. Presumably, computationalism can handle it as well as physics, for example.
4. Completing steps 1–4 (i.e. determining that S computes F and that $F = g_n \circ g_{n-1} \circ \ldots \circ g_1$, where we understand the g_1 terms) is generally quite difficult and typically requires creativity and insight. A theory of how steps 1–4 are accomplished would, therefore, require a theory of how humans come to see systems as executing F instead of E, and why F, say, provides a more satisfying explanation of the behavior of S than E does. To date, very little is known about this phenomenon.
5. Looping is executing a sequence of instructions over and over again. Branching is jumping from the

current position in the program to some other place in the program; this is typically accomplished by using a test statement, e.g. 'IF × ⟩ 5 GOTO STMT 100'. Sequencing is executing the next in a sequence of instructions unconditionally.

6. In computer science, I is typically the composition of two functions ($I = I_2 \circ I_1$). I_1 maps states of S onto instructions in some programming language. This function is left largely implicit, and is realized by engineers who design computers, beginning with those who design computer chips and ending with those who design operating systems and applications software. I_2 is provided by, e.g. a denotational semantics (or some other kind of semantical function) which provides semantic valuation functions mapping syntactic contructs in the programming language onto the abstract values they denote (see Stoy 1977). I_2 is also often left implicit.
7. Dretske (1981, 1985) has argued that intentionality is crucial to learning. If he is right, then he has shown why intentionality is crucial to cognition. But computers can learn, and they are supposed to lack intentionality.
8. An important manifestation of branching functions involves the nesting of conditions. Such nestings have this form (I shall use the procedure notation since it is more familiar):

IF ⟨condition 1⟩ THEN f_1
ELSEIF ⟨condition 2⟩ THEN f_2
.
.
.
ELSEIF ⟨condition $n-1$⟩ THEN f_{n-1}
ELSE f_n.

In such nestings, the first condition to be evaluated to be true determines which of the functions f_1 to f_n becomes executed. There are always a finite (and typically small) number of conditions to test for, and one of the conditions must be chosen: the ELSE clause (the final clause) is executed in case no other clause is; the ELSE clause is thus the trap clause.

9. Some philosophers have suggested that I have only succeeded in producing another argument for determinism. The attitude seems to be that my argument can be met by trotting out some argument for freewill. But this completely misses point. The object of the game is to explain human cognition scientifically, not to save some cherished notion of human agency. If the notion of will can be made scientifically respectable and we discover that our theories of cognition are inadequate without this revised notion, then will the concept of will take its proper place among other scientifically respectable entities such as mass, energy, the proton, the quark, DNA, etc. In fact, such a scenario is compatible with adopting computationalism as the foundation of cognitive science. But I suspect that any notion of will with a scientific foundation will not be considered the real thing. So, if we can explain human cognition without the will, then we ought to do so.

References

Bechtel, W., 'Connectionism and the philosophy of mind: an overview'. *The Southern Journal of Philosophy* XXVI (Suppl.) 2: 17–41 (1988).

Brainerd, W. and Landweber, L., *Theory of Computation.* Wiley, New York (1974).

Carello, C., Turvey, M., Kugler, P. and Shaw, R., 'Inadequacies of the computer metaphor', in M. Gazzaniga (ed.), *Handbook of Cognitive Neuroscience.* Plenum Press, New York, (1984). pp. 231–248.

Cummins, R., *The Nature of Psychological Explanation.* MIT/Bradford, Cambridge, MA (1983).

Cummins, R. and Schwarz, G., 'Radical Connectionism', in Horgan, T. and Tieson, J. (eds) *The Southern Journal of Philosophy* XXVI (Suppl.): 43–61 (1988).

Dietrich, E., *Computer Thought: Propositional Attitudes and Metaknowledge.* Doctoral Dissertation, University of Arizona, Tucson, Arizona (1985).

Dietrich, E., 'Semantics and the computational paradigm in cognitive psychology'. *Synthese* (in press a).

Dietrich, E., 'Programs in the search for intelligent machines: the mistaken foundation of AI', in D. Partridge and Y. Wilks (eds). *The Foundations of Artificial Intelligence.* Cambridge University Press, Cambridge (in press b).

Dietrich, E., 'Computers, intentionality, and the new dualism'. *Computers and Philosophy Newsletter,* Carnegie Mellon University, Pittsburgh, PA (in press c).

Dietrich, E. and Fields, C., 'Creative problem solving using the wanton inference strategy', in *Proceedings of the first Annual Rocky Mountain Conference on Artificial Intelligence.* University of Colorado/Breit, Boulder, CO (1986), pp. 31–41.

Dietrich, E. and Fields, C., 'Some assumptions underlying Smolensky's treatment of connectionism'. *Behavioral and Brain Sciences* 11: 29–31 (1988).

Dretske, F., *Knowledge and Flow of Information.* MIT/Bradford, Cambridge, MA (1981).

DRETSKE, F., Machines and the mental'. *Proceedings and Addresses of the American Philosophical Association* 59, (1): 23–33 (1985).
DREYFUS, H. (ed.), *Husserl, Intentionality, and Cognitive Science.* MIT/Bradford, Cambridge, MA (1982).
FIELDS, C. and DIETRICH, E., 'Intentionality is a red herring'. *Behavioral and Brain Sciences* 10: 756–757 (1987a).
FIELDS, C. and DIETRICH, E., 'Multi-domain problem solving: a test case for computational theories of intelligence', in *Proceedings of the Second Annual Rocky Mountain Conference on Artificial Intelligence,* University of Colorado/Colorado Institute for AI, Boulder, CO (1987b) pp. 205–223.
FODOR, J., 'Explanation in psychology', in M. Black (ed.) *Philosophy in America* Cornell University Press, Ithaca, NY (1965).
FODOR, J., *Representations.* MIT/Bradford, Cambridge, MA (1981).
FODOR, J., *The Modularity of Mind: an Essay on Faculty Psychology.* MIT/Bradford, Cambridge, MA (1983).
FOLLESDAL, D., 'Husserl's notion of noema' In Dreyfus (ed.), pp. 73–80.
GENTNER, D. and LANDERS, R., 'Analogical reminding: a good match is hard to find', *Proceedings of the International Conference on Systems, Man, and Cybernetics,* Tucson, AZ (1985).
GICK, M. and HOLYOAK, K., 'Schema induction and analogical transfer', *Cognitive Psychology* 12: 306–355 (1983).
GROSSBERG, S., 'Competitive learning: from interactive activation to adaptive resonance'. *Cognitive Science.* 11: 23–63 (1987).
HARNAD (1989).'Minds, Machines, and Searle'. *Journal of Experimental and theoretical Artificial Intelligence* 1: 5–25 (1989).
HAUGELAND, J., 'Semantic engines: an introduction of mind design', in J. Haugeland (ed.) *Mind Design.* Montgomery, VT, Bradford (1981), pp. 1–34.
HAUGELAND, J., The nature and plausibility of cognitivism'. *Behavioral and Brain Sciences* 1: 215–226 (1978).
HAUGELAND, J., 'Understanding natural language'. *Journal of Philosophy* 76: 619–632. (1979).
INTEL, *Introduction to iAPX 88.* Reston, Reston, VA (1983).
NAGEL, T., 'What is it like to be a bat?' *The Philosophical Review,* October (1974).
REEKE, G. and EDELMAN, G., 'Real brains and artificial intelligence'. *Daedalus* 117 (1): 143–173 (1988).
RICH, E., *Artificial Intelligence.* McGraw-Hill, New York (1983).
SAYRE, K., 'Intentionality and information processing: an alternative view'. *Behavioral and Brain Sciences,* 9: 121–166 (1986).
SAYRE, K., 'Various senses of 'intentional systems'. *Behavioral and Brain Sciences* 10: 760–765 (1987).
SCHANK, R., *Dynamic Memory.* Cambridge University Press, New York (1982).
SEARLE, J., 'Minds, brains, and programs'. *Behavioral and Brain Sciences* 3: 417–457 (1980).
SMOLENSKY, P., 'On the proper treatment of connectionism'. *Behavioral and Brain Sciences* 11: 1–23 (1988).
STABLER, E., 'How are grammars represented?' *Behavioral and Brain Sciences* 3: 391–402 (1983).
STICH, S., *From Folk Psychology to Cognitive Science: the Case Against Belief.* MIT/Bradford, Cambridge, MA (1983).
STOY, J., *Denotational Semantics: the Scott–Strachey Approach to Programming Languages.* MIT, Cambridge, MA (1977).
TENENBAUM, A., *Structured Computer Organization.* Prentice-Hall, Englewood Cliffs, NJ (1984).
WULF, W., SHAW, M., HILFINGER, P. and FLON, L., *Fundamental Structures of Computer Science.* Addison-Wesley, Reading, MA (1981).