ERIC DIETRICH

SEMANTICS AND THE COMPUTATIONAL PARADIGM IN COGNITIVE PSYCHOLOGY

ABSTRACT. There is a prevalent notion among cognitive scientists and philosophers of mind that computers are merely formal symbol manipulators, performing the actions they do solely on the basis of the syntactic properties of the symbols they manipulate. This view of computers has allowed some philosophers to divorce semantics from computational explanations. Semantic content, then, becomes something one adds to computational explanations to get psychological explanations. Other philosophers, such as Stephen Stich, have taken a stronger view, advocating doing away with semantics entirely. This paper argues that a correct account of computation requires us to attribute content to computational processes in order to explain which functions are being computed. This entails that computational psychology must countenance mental representations. Since anti-semantic positions are incompatible with computational psychology thus construed, they ought to be rejected. Lastly, I argue that in an important sense, computers are not formal symbol manipulators.

1. INTRODUCTION

In *From Folk Psychology to Cognitive Science* (1983), Stephen Stich argues that there is no scientifically respectable method for attributing semantic content to psychological states. Because of this, he advocates the radical position of rejecting semantics and semantically interpreted states (such as propositional attitudes) from cognitive psychology. Cognitive psychologists, on his view, ought to construe cognitive processes purely syntactically. If Stich is correct, future cognitive scientists will have theories that do not countenance mental states with content, i.e., theories that do not countenance mental representations. For philosophers such as Fodor and Pylyshyn, who advocate representational theories of the mind, this would be catastrophic.[1]

Stich's view, however, is much more sweeping than it first appears. If he is correct, then not only will representational theories have to be rejected, but, I claim, all of *computationalism* will have to be rejected as well, because computationalism – the thesis that cognitive capacities are best explained in terms of capacities to compute certain functions – requires us to posit and semantically interpret mental states and mental processes. A success for Stich is thus much more than a disaster for Fodor and Pylyshyn. Rejecting computationalism would affect *all* of cognitive psychology. Psychologists in fields as diverse as

Synthese **79**: 119–141, 1989.

problem solving, perception, imagery, memory, language understanding, and developmental psychology would have to completely change their methodology.[2]

In this paper I will argue that computationalism does in fact require us to posit and interpret mental states and structures, and I will present a strategy for doing this, the "computational strategy". This strategy is used in both the computational and cognitive sciences. In practice, therefore, computer and cognitive scientists attribute semantic content in the course of producing scientific explanations. (In my opinion, this is a good prima facie reason for being skeptical about arguments such as Stich's.) I will argue that the computational strategy is used because it provides cognitive scientists with a scientifically respectable method for attributing contents of computational, psychological states. If my argument is correct, then there are good reasons for embracing semantics and remaining computationalists, and we may reject current arguments to the contrary.

My view, however, is also more sweeping than it first appears. The necessity of semantic attribution within the computational paradigm is not generally acknowledged. The main reason for this, I think, is that computers themselves are widely regarded as formal symbol manipulators or syntax machines, manipulating symbols independently of their content. If my argument is correct, then to understand computers we must ascribe contents to their internal functioning. This means that computers cannot be understood as formal symbol manipulators. Therefore, in my view, not only is Stich wrong in claiming that there is no scientifically respectable method for assigning contents to mental states, but the dominant view of computers is also in error.[3]

In Section 2, I present two of the problems Stich sees for countenancing representational mental states. My primary concern is not these problems per se, but rather the general intuition they are intended to foster that semantic attribution is scientifically suspect. My arguments that semantic attribution is scientifically respectable and necessary for computationalism is presented in Section 3. Finally, in Section 4, I discuss the consequences of my argument for the view that computers are formal symbol manipulators.

2.

Stich sees two fundamental problems with attributing semantics to mental states. The first is that a scientific psychology must generalize

over contents ascribed to mental states. This requires picking out identical contents across individual subjects or systems. But according to Stich, there simply is no viable notion of content identity; rather, contents are more or less *similar* (1983, pp. 85–87). Moreover, Stich claims that judging contents to be similar essentially involves the person doing the judging and the context in which the judgments are made. This makes such judgments so vague and context sensitive that they will not support scientific theorizing (pp. 128–148). Hence, we ought to forego ascribing semantic contents to subjects.

The second problem Stich sees is that in the case of a particular subject or system, the referential components of the semantic contents attributed to it will depend on causal relations between it and external referents. Changing external referents changes the content ascribed to the subject. However, such changes need not (and typically will not) change the states and processes of the subject that are relevant to psychology. Hence, it cannot be the case that referential content is important to psychology. Again therefore, we ought to forego ascribing semantic contents to subjects (1983, pp. 164–170).[4]

Each of these problems can be construed as an argument that a certain strategy for semantic attribution is inadequate for scientific psychology. I will call the strategy associated with the first problem the *similarity strategy*, and the strategy associated with the second problem the *causal strategy*.

A. *Problems with the Similarity Strategy*

The similarity strategy is the strategy Stich claims we use in our folk psychology and everyday ascriptions of content (1983, pp. 73–110). On Stich's view, when we ordinarily attribute content to the mental states of others we compare them to ourselves, exercising a sort of 'empathic' capacity. This comparison amounts to the claim that another is in a mental state similar to one which would normally be the central cause of our assertion of the content sentence. We would think that someone else is in a mental state similar to one we might have if we judge that person's circumstances, networks of beliefs and desires, and specific behavior in the circumstances to be similar to our own. Thus if someone's situation and mental life seems similar to our own, then we can successfully ascribe contents to his or her mental states by comparing them to ones we might have were we in his or her situation (1983, and 1982).[5]

To attribute content in this way we have to understand, at least partially, the other individual and his or her situation. Hence, we cannot consider subjects nor their mental states in isolation. Instead, we must consider the entire subject, its network of interconnected mental states, the set of circumstances in which it finds itself, and the context in which we are making the attribution. Stich discusses this understanding at length (1982, 1983). He argues that it resolves into various kinds of *similarity judgments*. These judgements assess the similarity between the mental contents of the subject and the ascriber along certain dimensions, the main ones being functional similarity, or similarity of causal potential, ideological similarity, and similarity of reference (pp. 88–90).[6]

According to Stich, the dependence of content ascription on similarity judgments (and 'empathic' or 'projective' understanding, in general) explains an interesting phenomenon of our everyday ascriptions of mental content: as subjects become less and less like us, or as the situations of subjects become less and less like those of our everyday experience, we become less and less able to attribute beliefs or other content-full mental states to them (1983). Stich provides several interesting examples of humans who are so different from us that we cannot comfortably ascribe contents to their mental states using the similarity strategy. This does indeed suggest that in our ordinary life we use something like the similarity strategy to ascribe contents to mental states. Unfortunately, the similarity strategy cannot be used to scientifically ascribe content, and hence should not be used in psychology, or so Stich argues (1983).

Consider the nature of similarity judgments. Judging that X is similar to Y is always *context dependent* and *relative* to the one who is doing the judging. According to Stich, context dependency and observer relativity make similarity judgments inappropriate for psychology because whether a given psychological generalization applies should not depend on the theorist doing the research or on the setting the theorist is in (1983, footnote p. 139). If psychologists were to use the similarity strategy then they would be hindered in their efforts to develop psychological theories of very young children, primitive peoples, and victims of strokes, retardation, and schizophrenia, none of whom are similar enough to the psychologists themselves for semantic content to be successfully ascribed. The *science* of psychology, as opposed to our folk psychology, should apply to all humans, not just ones like us.

The conclusion is that the similarity strategy for ascribing contents to mental states is inadequate for use in scientific psychology. Though the strategy may be used in our folk psychology and day-to-day judgments, it cannot work in our scientific psychology.

B. *Problems with the Causal Strategy*

Stich's second argument is that some components of semantic contents (e.g., the referential ones) essentially depend on causal relationships disallowed by the autonomy principle. According to Stich, the autonomy principle is so fundamental that it cannot be rejected, hence semantics must be rejected, along with all psychological theories that ascribe semantic content to mental states.

The autonomy principle claims that "states and processes that ought to be of concern to the psychologist are those that supervene [only] on the current, internal, physical state of the organism" (1983, p. 164).[7] This entails that "any differences between organisms which do not manifest themselves as differences in their current, internal, physical states ought to be ignored by a psychological theory" (p. 164). Consequently, if some aspect of a subject's environment on history might have been different without affecting its internal states, then that aspect is psychologically irrelevant (pp. 164–165).

Evidently, referential components in semantic interpretations of mental states are just the kind of environmental aspect rendered psychologically irrelevant by the autonomy principle. To make this plausible, consider this example (due to Stich, 1978). Imagine that I had an exact duplicate made of me yesterday. According to the autonomy principle, all psychologically adequate theories will posit properties and relations that are true of both me and my duplicate, since we are physically identical. Moreover, only properties and relations true of the both of us are genuine, explanatorily useful psychological properties and relations. (Stich says, "If the principle is to be observed, then the only properties and relations that may legitimately play a role in explanatory psychological theories are the properties and relations that a subject and its replica will share" (1978).)

By definition, my duplicate and I share physical descriptions of our internal mental states. However, we do not share semantic descriptions of our internal states. I have a mental state referring to my deceased great-grandmother because (assuming a causal theory of

reference, as Stich does) I have had the right kind of causal relation to her. However, my duplicate (let us suppose) has not had the right sort of causal relation. My duplicate, therefore, has no mental states referring to her. Hence, our mental states differ semantically (precisely in our referential components). It is clear to Stich, however, that this difference doesn't make a difference, psychologically; whatever behaviors I have, my duplicate will also have. Sharing physical descriptions of our internal states but not semantical descriptions means that the semantic descriptions do not manifest themselves as differences in our physical, internal states. It follows from this and the autonomy principle that semantic descriptions of mental states are otiose from the perspective of cognitive psychology.

Since the causal strategy depends on psychologically irrelevant causal relations, the attributions themselves must be psychologically irrelevant. As cognitive scientists, we simply ought to avoid making them.

C. *The Inadequacy of Semantic Attribution*

Taken together, the problems with the two strategies make questionable the goal of scientifically attributing semantics to mental states. On the one hand, we cannot use the strategy from our folk psychology, because when using this strategy, we search for similarities between ourselves and others, and this prevents us from being able to successfully attribute content to those significantly different from us. On the other hand, one of the best theory-driven approaches to semantic attribution (the causal strategy) will not do either because the underlying theory (the causal theory of reference) postulates relationships between subjects and the world that are irrelevant to psychology.[8] Hence, even if our subjects are identical replicas of us, we still have no guarantee that we will be able to attribute semantic contents because our replicas may not be embedded in the world the way we are.

All this suggests that semantic attribution is successful only when applied in ordinary settings to subjects like us. But a scientific enterprise should not be so idiosyncratic. Hence, semantic attribution is scientifically disreputable. This is not to claim that semantic attribution is completely disreputable. It does have a role in our ordinary, day-to-day lives because it helps us understand (though not

scientifically) most of the humans we come in contact with. Semantic content, and semantic attribution emerge, then, as heuristic methods we use in our ordinary lives to get along with ordinary people like us. As Stich says: "As I see it, the notion of 'content' . . . despite all its utility in the workaday business of dealing with our fellow creatures, is simply out of place when our goal is the construction of a scientific theory about the mechanisms underlying behavior" (1983, pp. 5–6).

3.

The failure of the two strategies discussed in Section 2 suggests that semantic content itself is scientifically otiose. Neither strategy is used in the cognitive sciences, yet our psychological theories often explain our subjects' behavior. Hence, content (together with semantic attribution) must not be required for scientific psychology (cf., Stich 1983, p. 208).[9]

We cannot accept this conclusion without giving up computationalism, the most successful paradigm cognitive psychology has ever embraced. Such a costly methodological shift could be avoided if we could find a strategy for semantic attribution that allowed us to attribute contents to subjects very different from us, and one that tied semantic descriptions of subjects' internal states to physical descriptions of their states (thus adhering to the autonomy principle). Meeting both of these objectives would make semantic attribution scientifically respectable because it would free semantic attribution from the idiosyncrasies of the ascribers (though not necessarily from their scientific goals), and make the attribution depend on the subjects and their relation to their environment.

Such a strategy exists, and in fact is used today in the computational and cognitive sciences. I call this strategy the *computational strategy*. The computational strategy allows one to ascribe contents to another's mental states based on what function would *explain* the other's behavior and psychological capacities. This strategy relies on the fact that explanatory pressures license the ascription of certain contents and not others, just as such pressures license scientific explanations in general.

The computational strategy doesn't require us, as theorists, to be similar in any way to those to whom we are ascribing contents (this is why it is successfully used by computer scientists, for example). It also

respects semantical differences between subjects precisely where the autonomy principle dictates it should. Finally, if the computational strategy is not used when attributing semantic content to a subject, then we will not be able to devise any computational theory for that subject or subjects of its type. This last point is strong. It entails first that an explanation of a cognitive capacity cannot be a *computational* explanation if is does not posit representations and processes internal to the subject, and attributes content to them. Second, it entails that we must use the computational strategy if we are to understand a subject as computing functions. The rest of this section is devoted to defending these claims.

A. *The Computational Strategy*

The goal of the computational strategy is to attribute to a subject or physical system, S, the computation of a certain function, F. To do this, it must be determined (or assumed) that S executes F, and it must be determined how S executes F. The computational strategy is successful when it is possible to explain S's computation of F in terms of a sequence of functions $\langle g_1, \ldots, g_n \rangle$ such that (1) $F = g_n \circ g_{n-1} \circ \cdots \circ g_1$; (2) S passes through a sequence of states where each state corresponds 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 (3) we antecedently understand the individual g_i's (see Figure 1). When $F = g_n \circ g_{n-1} \circ \cdots \circ g_1$ and the g_i's are nontrivial it is natural to say that the sequence of functions $\langle g_1, \ldots, g_n \rangle$ *analyzes* the capacity of S to compute F (see Cummins 1975 and 1983, pp. 28–44).[10]

Completing steps 1, 2, and 3 (i.e., determining that S computes F and that $F = g_n \circ g_{n-1} \circ \cdots \circ g_1$ where we antecedently understand the g_i's) is generally quite difficult and always a matter of creative problem solving. A theory of how steps 1, 2, and 3 are accomplished would therefore require a theory of how humans creatively solve problems, i.e., how humans come to see systems as executing F instead of E, and why F, say, provides a better explanation than E. To date, there are no theories of creative problem solving that can adequately address these issues (but see Dietrich and Fields, 1986). Something can, however, be said about steps 1, 2, and 3.

F is a "system-sized" function. Fixing F for a particular system is seeing S as doing something regular. Therefore, fixing F grossly, but not trivially, answers the question "What is S doing"?

The necessity of semantic attribution begins with the specification of F. If we choose, we could refuse to interpret F's inputs and outputs, and instead specify F at the level of physical regularities. But doing this would leave the specification of F relatively useless. In fact, we would merely have a description of the state changes of the system S.

Consider a transducer such as a telephone receiver that takes electrical power and produces compression waves in the surrounding air. If we notice regularities on the compression-wave side that covary with regularities on the electrical side (e.g., respective frequencies), we can describe the transducer as computing the function T:

> T: ⟨specification of electrical regularities⟩→⟨specification of compression-wave regularities⟩

However, interpreting T's input and output (or, what is the same: producing another function T^*) not only gives us a better understanding of what the transducer does, but an understanding that is crucially different from the specification of T above. For example, if we interpret the electrical regularities as representing voice patterns and the compression waves as acoustic signals representing voice patterns then we will be in a position to detect *errors* in our transducer. Indeed, in general, the only way we can know that a certain regularity is an inappropriate output given a certain input is by using some global interpretation within which we expect the given input to produce an output other than what it did. Take an extreme example. If our transducer outputs an acoustic signal corresponding to 'goodbye' every time it received an electrical signal corresponding to 'hello', this would be an error (i.e., unwanted). But we could recognize such an error only if we had interpreted the corresponding physical regularities.[11]

Once F has been specified, the analysis of F into the sequence $\langle g_1, \ldots, g_n \rangle$ is possible. Here is where semantic attribution has its greatest payoff. The analysis of F is a real step forward when we antecedently understand the individual g_i's. This and the correspondence between the states S can pass through and the g_i's is what gives the explanation of S's state changes as the computation of F its power, for it is this analysis that connects S's state changes with something we understand, viz., the g_i's.[12]

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 understand S's state transition from s_i to s_j by seeing the transition as the execution of g_i. Doing this is interpreting the states of S because it is treating the states of S as symbols which get 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's (see Figure 1).[13] Once we can view S as an interpreted virtual machine, we can switch between this view and the physical state transition view. At this stage, the computational strategy is complete, and we have our explanation of what S does and how it does it (see Cummins 1983, p. 39).

The correspondence between the S's states and the g_i's can be made precise. On the assumption that S computes F by computing $g_n \circ g_{n-1} \circ \cdots \circ g_1$ we can provde an 'interpretation function' that maps states of S onto the g_i's. Letting I represent the interpretation function, then we get the following picture (see Stabler 1983, and Pylshyn 1984, pp. 54–59).[14]

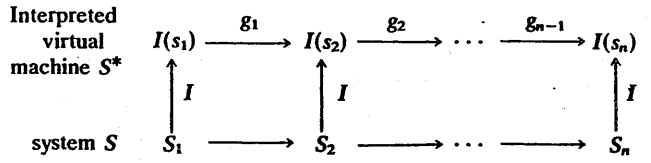

Figure 1. Interpreting the state transitions of S as computing the sequence of functions $\langle g_1, \ldots, g_n \rangle$.

B. *An Illustration of Semantic Attribution using the Computational Strategy*

The use of the computational strategy is good scientific procedure. A concrete example that exhibits this is perhaps useful at this point. Consider something very different from us: an ordinary, four-function calculator, a device that can add, subtract, multiply, and divide. For our purposes, assume that we are given a four-function calculator

which uses a brand new architecture, and that our task is to figure out what this new architecture is.[15]

When we test the new calculator, we notice that it sometimes seems to add, subtract, multiply, and divide, and that sometimes it does nothing. Further experimentation reveals that the order of the inputs makes a difference. If we input "2+2", the calculator does nothing (other than echo the final "2"). But if we input "2 2+", the calculator outputs "4". We hypothesize that, for the four, two-argument arithmetical operations, the operation must be specified last. We generalize this, and hypothesize that the operation must always be input last. To test this we input "1 2 3 4+". The calculator outputs "7", which doesn't accord with our prediction. The calculator seems to only "remember" the last two numbers input. Perhaps it can only accept two numbers at a time, with the operation last, for example we observe that, "1 2+3 4+" outputs first "3" then "7". However, as we continue experimenting with different orders of input, we discover that "1 2+3+4+" gives the output "10". We interpret this behavior this way. All four operations only accept two numbers at a time. However, the result of an operation is somehow 'remembered', and is used as one of the arguments for succeeding operations, provided that it is not 'forgotten' which happens if more than one number is input next.

This initial explanation, we may imagine, stands up against further tests. Now, how are we going to explain how our calculator works? Why does the input have to have the argument and operations in the order that it does (post-fix order as opposed to the more natural infix form)? Specifically, we want to explain the way our calculator seems to 'remember' and 'forget' numbers, and that when it gives the output we want it seems to 'remember' two numbers, operate on them, and 'remember' the result, which can then be used as one of the arguments for the very next operation.

Naturally enough, we hypothesize an internal representation that stores the inputs in a certain way, and we hypothesize that the calculator executes an algorithm that must have its information stored that way. There is a data structure that nicely fits our needs: the stack. Our hypothesis, then, is that our calculator is a stack-based machine, i.e., our calculator uses a stack as its primary data structure.[16]

Stacks have two associated operations: push and pop. Data items are pushed onto the stack as they are stored, and popped off from the 'top' of the stack as they are used. We hypothesize that our calculator

pushes numbers onto its stack as they are input, and pops them when an operation is input. If "2 2+" is the input, the first "2" is pushed onto the stack, then the second "2" is pushed. When "+" is received, the two "2"'s are popped off the stack, added together, and "4" is pushed back onto the stack.

Here is how the calculator works if we input "1 2 + 3 + 4 + ". Let T be the stack in the calculator, and assume that $T = ( )$, i.e., assume that T is empty at the beginning of the computation. First, "1" and "2" are pushed onto the stack in the order in which they are received. Now $T = (2\ 1)$. (Note that the top of the stack is on the left.) Next, "+" is input. Now "1" and "2" are popped off the stack, added, and the result is pushed back on the stack. Now, $T = (3)$. Next, "3" is received as input. Now $T = (3\ 3)$. Next, "+" is received. The two "3"'s are popped, added, and "6" is pushed onto the stack. $T = (6)$. When "4" and "+" are received the same thing happens, with the result that finally $T = (10)$. If we assume that whatever is on the top of the stack is also displayed as output, then 10 is displayed, and we have our explanation of why the order of input to our calculator has to be the way that it does.[17]

Compare the features of this example with the definition of the computational strategy given above. First, we assumed that the calculator added. (This is indeed an assumption because we could have assumed that the infix "2 + 2 = null" was the interesting function rather than the post-fix "2 2+ = 4". This assumption is dependent on the context that in our culture calculators are used to compute (instances of) functions like addition, not functions that map inputs onto the null set.) Second, we explained how the calculator added by analyzing its addition into a series of antecedently understood stack operations (involving push and pop). Third, our explanation requires a correspondence between states of the calculator and states of the stack. We interpreted (a subset of) the states of the calculator as being states of the stack. We also interpreted changes in the stack as computations of the functions push and pop. Both of these steps essentially involved semantic attribution, and both were essential to understanding what our calculator did and how it did it. Finally, the success of our explanation, i.e., its utility, vindicated our original assumption that the calculator added.

I am now in a position to present an argument that positing and interpreting structures and processes is necessary if an explanation is

to be a *computational* explanation. A process is a computation if it is usefully described as computing a function F, i.e., if over time, the process generates output equal to F(input). But, in order to see a certain process as computing a function F, we must interpret the initial state of the process (or system) as being the requisite input for F, and the final state of the process as being the requisite output, namely F(input).

C. *The Computational Strategy as a Strategy for Semantic Attribution*

It is worth emphasizing the main points so far. First, the goal of the computational strategy is ascribing the computation of a certain function F to a certain system, S. Second, in order to do this one must *interpret* states of S as entities over which F (and each of the g_i's) is defined. That is, we cannot understand state transitions of a system as adding 1 and 2 unless we are prepared to interpret states of S as representing 1 and 2.[18]

Though similar points have been made before (see Cummins 1983, Smith 1982), they have not been seen as constituting a strategy for semantic attribution. Furthermore, the consequences of using this strategy have not been appreciated. The first consequence is that though attributions of semantic content do depend on the context in which the attributions are made, the contexts are completely independent of any similarity measures between subjects and theorists. For example, if an aboriginal tribe found a calculator in the desert and did not understand addition, then they would devise theories of it that differed significantly from ones we would devise. Such context dependence relativizes our scientific explanations to our goals and views of the world. This is not methodologically suspect as Stich suggests (1983, footnote p. 139); it is merely a feature of scientific explanation. It is hard to imagine a methodology that could do otherwise. Indeed, this sort of context dependence appears in all our sciences. Even in physics, explanatory goals affect theories: if we wanted to, we could describe the earth as being at the center of the solar system with all the planets revolving around it in epicycles of perfect circles. Moreover, if we observe the practices of theorists in the computational and cognitive sciences, we see that the computational strategy is used successfully relative to our goals (see

Cummins, 1983; Haugeland, 1981; Smith, 1982; Wulf, 1981, esp. Chap. 5 and part 2; and Stoy 1977).

The second consequence of using the computational strategy is that there are no arbitrary or extra-theoretical restrictions placed on the contents that can be attributed. The only general requirement is that the attributed contents explain observed behavior and hypothesized cognitive capacities. For example, causal connections underlying reference have no role in the computational strategy, but not because such connections are incompatible with semantic attribution. Rather, such connections are not useful for explaining cognition. Whereas, on the causal strategy, we had to pick between attributing contents or having ("scientific") explanations, using the computational strategy allows us to attribute contents in an explanatorily useful way. For example, in the case of my duplicate and I, both of us have thoughts about my great grandmother. Presumably, only I was causally related to her, but that is not important as far as scientific explanation is concerned. What is important is which function we are computing, and hence the (attributed) contents of our states. Since my duplicate and I compute the same function, the content of our states must be identical.[19]

Since referential causal connections are not important to the computational strategy, we can simply avoid such issues as what the causally correct referent of a referring term is, while respecting the autonomy principle. The computational strategy allows us to attribute contents and fix referents in a way that is relevant to psychological explanation because contents are assigned on the basis of what explains the current physical, computational state of the system: this is what the function *I* does.[20]

This brings us to the third consequence. In the computational strategy, attributing a particular content to a particular mental state (or state of the system) is *not* paramount, as it is for example in the causal strategy. 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. In discussing this point with respect to ciphers, Cummins says:

> In a cipher any numeral can, taken independently of the others, be assigned any significance whatever. It is only a definite context which places any constraint on the significance to be assigned to an individual numeral, the requirement being that when each numeral is assigned a meaning by a determinate rule, a coherent message should result. (1983, p. 37)

The fourth consequence is that attributing semantics via the computational strategy is not a folk art. 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, the view that computational explanations essentially involve semantic attribution has two major consequences. First, it entails that if psychology is to embrace the computational paradigm, it must 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. Second, it entails that computers cannot be understood as merely formal symbol manipulators. In the next section, I will defend this latter claim.

4.

Most philosophers view computers as formal symbol manipulators. A formal symbol manipulator is a *syntax machine*, performing the actions it does on the basis of the syntactic properties of the symbols it manipulates, rather than on the basis of what the symbols symbolize. This view of computers has allowed some philosophers to divorce semantics from computational explanations. Semantic content, then, becomes something one adds to computational explanations to get psychological explanations (see Haugeland, 1981; and Pylyshyn, 1984). Other philosophers, such as Stich (1983), take the more radical view that semantics is completely otiose. If I can at least raise some doubts that computers are or can be understood as formal symbol manipulators, then this basis for current anti-semantic sentiments will be weakened.

The importance of semantically interpreting structures and processes in computational explanations is well known to computer scientists.

The field of *denotational semantics* is an example of the study of the semantics of computational processes. In denotational semantics, computer scientists study the semantics of programming languages (see, e.g., Stoy, 1977). This approach to semantics involves providing valuation functions for the syntactic constructs of a programming language which are used to write programs in that language. The valuation functions map the constructs onto the abstract entities they denote: numbers, truth values, data structures, etc. The important point for our purposes is to note that the mapping is designed in such a way that behavior of the constructs is explained by the behavior of the abstract entities, which can be studied independently. Since any program is made up of syntactic constructs of a given programming language, explaining the constructs is necessary for explaining what the program does, or, said another way, explaining the behavior of the virtual machine the program instantiates.[21] Within denotational semantics, therefore, machines are thought of both formally and semantically. This is an important point: for computer scientists, semantics is not something that is informal or essentially dependent on folk notions.

Semantically interpreting processes and structures using the computational strategy is more than a field of study within computer science, however. Brian Smith has argued that explanatory semantic attribution lies at the core of the notion of computation (Smith, 1982). In 'Semantic Attribution and the Formality Condition' (1982), Smith says "... what distinguishes an abacus, a calculator, and even a full scale computer, from other rule-governed complex artifacts like steam plants and food processors, is that the best explanation of [their] behavior is formulated in the domain of *interpretation*, not in the domain of the uninterpreted signs ... [Smith's emphasis]". He goes on to say: "The more complex the computer, the more important the interpreted account becomes to our understanding, and the more variegated the kinds of interpretation: once you move past simple calculating devices into full programming languages, you find not just simple names, but quotation and internal reference, complex function designators, and even intensional contexts". Thus, on Smith's view of computation, something is a computer just in case we are required to semantically interpret its input/output behavior, its structures, and its internal processes in an effort to explain its most salient capacities. His own summary of this view is this: "Computers ... are ... *just those devices whose functional architecture we understand in terms of external semantical attribution* [Smith's emphasis]".

Though Smith's view has not achieved the status of received doctrine within computer science, it is fair to say that he has articulated what many computer scientists have long believed: interpreting the constituents of a process as standing for or representing other things is essential to understanding that process as a computation.[22] Using the language from Section 3, if we can usefully view a system as an interpreted virtual machine, it is a computer.

So far, my argument has been 'epistemological'. I have argued that in order to understand computers (i.e., in order to know what they are doing), we must view them as interpreted virtual machines. In some sense, a computer might be a formal symbol manipulator, but we could never understand it as such. Unless we attribute contents to the computer's processing we could never build one nor could we ever debug one (recall the discussion of the transducer in Section 3).[23]

To me, the epistemological argument is compelling. Worries about whether computers are really formal symbol manipulators seem beside the point: to understand them, we must view them as semantic engines. However, I think an argument can be made that, independently of our understanding them, computers are *not* formal symbol manipulators.

In the 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. This means that the manipulations must not depend on a symbol having a meaning or denotation. For example, manipulating a symbol solely in virtue of whether it is a token of some numeral, letter, or part of speech is one way to treat a symbol purely formally. If this is correct, then computers are not formal symbol manipulators.

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 precise about what the entities under discussion are. A computer running a Lisp interpreter (the Lisp execution program) defines a virtual machine we will call a Lisp virtual machine (LVM). An LVM operates solely in

terms of Lisp expressions, the syntax for which can be specified by a grammar. (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.[24])

An 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 content of 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.

A particular LVM evaluating the expression '(+ x 1)' is required to look up the value of the variable 'x', which is some other Lisp expression, let us say '6'. If the LVM could not perform this look-up, the expression would be syntactically ill-formed: '+' is not defined for expressions we interpret as non-variable letters. But now notice, the LVM itself in treating 'x' as a variable, 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*, 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'). Indeed, '(+ x 1)' means something different to us than it does to the LVM. Nevertheless, the LVM's manipulations depend on the fact that 'x' has a meaning (and indeed, on the meaning it has). This is enough to render false the claim that computers are formal symbol manipulators, at least on the straightforward interpretation of this claim I have assumed. (This same result appears to follow assuming a causal theory of reference. When a computer internally fixes the binding of a variable, it is determining the referent of that variable causally.)

I suspect that the notion of a formal symbol manipulator is a hodgepodge of other notions and intuitions, some of which we want to maintain. For example, in a very interesting paper, Haugeland (1979), argues 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 may in fact be noticing that computers are not 'understanders', at least not today. Another way of stating my point, then, is though computers may not be understanders, they are not merely formal symbol manipulators.

5. CONCLUSION

I have shown, I believe, that computers are not formal symbol manipulators. But whether they are or not, our understanding of computers inescapably involves attributing semantics to their processes. Moreover, I have argued that, in general, computational explanations, either of humans or computers, crucially involve attributing semantic contents. If we are going to utilize computational explanations in our psychological theories, then our psychological theories will crucially involve semantic content.[25]

NOTES

[1] Fodor, for example, wishes to explain propositional attitudes in terms of operations defined over mental representations (Fodor, 1975, 1981). He also wants to explain the intentionality of propositional attitudes, their semantic properties, in virtue of mental representations (Fodor, 1981 and 1984b). Obviously, if a mature cognitive science does not countenance propositional attitudes and mental representations, Fodor's program is doomed.

[2] Computationalism is the theoretical and methodological backbone of modern cognitive psychology in the sense that some of our best theories are explicitly computational (see Fodor, 1975, Cummins 1983, and Pylyshyn 1984). There are computational theories of language processing and language acquisition (e.g., Fodor, Bever, and Garrett 1974, and Anderson 1976), learning (e.g., Anderson 1976, 1983), imagery (e.g., Pylyshyn 1984, and Block 1981), memory (e.g., Norman and Rumelhart 1975, Anderson 1976, 1983), problem solving (e.g., Newell and Simon 1972), cognitive development (e.g., Moore 1973), and vision (Marr 1982).

[3] Rejecting the formal symbol manipulator view of computers will also have far-ranging affects. For example, Searle's (1980) argument that computers aren't intentional, Fodor's (1980) argument for adopting methodological solipsism as a research strategy in cognitive psychology, and Dretske's (1985) argument that computers can't add (let alone think) will all be undermined since they all rest on this view of computers.

[4] Stich uses both of these problems in arguing that cognitive psychologists ought to reject the notion of belief (1983).

[5] An essential part of Stich's theory is attributing content to ourselves. Notice that this cannot involve any sort of empathic comparison. Though he doesn't explicitly say, apparently we can attribute content to ourselves merely by introspecting, discovering

which contents are 'there', and attributing those to ourselves. In any case, I am going to assume that introspection is involved when we attribute content to ourselves.

[6] Stich does not think that these similarity judgments are consciously a part of content ascription; rather, the judgments are made unconsciously (1982).

[7] Fodor (1980) has advocated a like principle called 'methodological solipsism' a term Putnam coined for the 'assumption that no psychological state, properly so-called, presupposes the existence of any individual other than the subject to whom the state is ascribed' (Putnam, 1975).

[8] The causal strategy is not a folk strategy. It is a theory-driven strategy because contents are attributed in accordance with referents specified by the underlying theory of reference. Moreover, the causal strategy is regarded by many as being at least partially correct. Stich endorses it (see 1983, pp. 60ff, 89ff, and 165). See also, Kripke (1972), Putnam (1975), Devitt (1981), Dretske (1981), Fodor (1984).

[9] Though the failure of the two strategies is not sufficient for claiming that scientific psychology is incompatible with ascribing semantics, this claim can be bolstered by developing a semantic-free methodology for psychology. This is the strategy Stich attempts with his syntactic theory of mind (1983).

[10] In computer science, the sequence of g_i's are typically expressed in some general formal language like the lambda calculus or in a programming language like Lisp. The sequence of g_i's constitute an *algorithm* expressed in the formal language. For completeness, note that in the degenerate case, $F = g_k$ for some $(1 \leq k \leq n)$, and that all the other g_i's are the identity function, i.e., the "do nothing" function.

[11] Of course, we can detect errors in devices such as transducers at the regularity level if the manufacturer tells us what the T should be, but this is completely artifactual: we don't have manufacturer's specifications for our eyes, for example. For natural transducers, we must interpret the given physical regularities, e.g., interpreting neuronal activity as representing edges, shadows, or locations of color patches. See Marr (1982).

[12] What understanding a function amounts to is also something that eludes cognitive scientists. It surely has something to do with knowing how to use it, how to recognize it, and how to transform it. In this regard, understanding a function is similar to understanding a natural language sentence. Presumably, all types of understanding are similar and will be subsumed under one theory; until we have a much more robust theory of the mind, however, we are going to have to be content with couching understanding in terms of use.

[13] For an introduction to the notion of a virtual machine, see Tanenbaum (1984).

[14] In computer science, I is the composition of two functions ($I = I_2 \circ I_1$). It is I_1 that allows us to see the state transitions of S as causally related and law-like. In physics, a function similar to I_1 allows us to see S's state transitions as being governed by physical (transition) laws (see Cummins 1983, esp. pp. 1–2). However, in computer science, I_1 maps states of S onto objects (or states) that are themselves interpreted, i.e., I_1 maps the states of S onto instructions in some programming language. I_1 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 picks up where I_1 leaves off, providing a further interpretation of the states of S. I_2 is provided by a denotational semantics which provides semantic valuation functions mapping syntactic constructs in the programming language onto the abstract values they denote (see Stoy 1977, and Pylyshyn 1984, pp. 59–62).

[15] This is sometimes called 'reverse engineering'. It is a procedure used in industrial espionage.

[16] Data structures are methods for organizing information inside a computer. A stack is merely a kind of data structure where individual items of information are added to or deleted from only one end of the stack called its 'top'. A stack of dinner plates works like a stack data structure: plates are added to the top of the stack when stored, and they are removed from the top when they are needed.

[17] This is not a complete explanation of how our calculator works; there are, for example, other data structures that are involved when the calculator actually adds (subtracts, etc.) the two numbers from the stack. A complete explanation would also include an account of how the calculator's hardware supports the computational explanation involving data structures and processes. Explanations at the hardware level still require semantic attribution. A good example of this sort of enterprise is provided in the appendix to Cummins (1983).

[18] It is possible to describe systems (such as our calculator) at the micro-component level. Yet as noted in Note 17, even at this level, components are interpreted as computing functions, and inputs and outputs to these components are interpreted as either numbers or truth values. So even at the microcomponent level, semantic interpretation is crucial. I will return to this point in Section 4.

[19] My duplicate and I of course do not have identical physical states; we have similar physical states (because of quantum mechanical effects and the like). But this difference will only make a difference when our behaviors become divergent enough to warrant assigning different functions to our cognitive processes.

Notice by the way that the causal strategy may not be as bad as Stich claims. The causal relations between my duplicate and I (the ones figuring in the construction of my duplicate) are important to (and may be sufficient for) attributing thoughts about my great-grandmother to my duplicate.

[20] Care must be taken here not to think of physical states as states described in one of the languages of physics such as quantum mechanics, for at this level there aren't any computers, or any computational devices whatsoever. There are only state transitions and probabilities assigned to the state transitions. No functions get computed by any system described at this level unless one is willing to see, e.g., clusters of particles as numbers, but of course if one does this, one is interpreting state transitions.

[21] There are other methods of studying the semantics of programming languages, (for a good introduction see Wulf, et al., 1981). Note the similarity between denotational semantics and model theory. Valuation functions in denotational semantics resemble valulation functions in logic that assign terms in a formal language L to an interpretation of L.

[22] Smith no longer believes that the semantic properties of computers are merely attributed. On his new view, computers are not formal symbol manipulators, in any sense of this phrase. Thus his view now is stronger than the one he espouses in his 1982 paper (personal communication).

[23] Two colleagues of mine were recently testing and debugging an English language parser. The parser produced incorrect parses for sentences such as "The best ingredients for pizza are ground beef and onions". The error turned out to be that the parser took 'beef' to be one conjugation of the verb "to be". In diagnosing and repairing this bug the content "beef is related to the verb 'to be'" was actually

attributed to the parser. This attribution was crucial to finding and fixing the bug.

[24] Sometimes the bit level description of computers is thought to be the 'real' level of description, often because the bit level is thought to be the level of causal interactions. This is completely mistaken. First, causal interactions can be and are described as occurring at many different levels, e.g., the Lisp level. Second, bits are as symbolic as Lisp expressions.

[25] This paper has benefitted greatly from the comments of an anonymous reviewer – thank you. I also thank Chris Fields, Robin Hill, Alan Strudler, Dan Fass, and Jordan Pollack for invaluable discussions on the various philosophical and computational issues I've raised here, and Stephen Stich, Robert Cummins, Mike Harnish, and Myles Brand for commenting on earlier versions of this paper.

REFERENCES

Anderson, J. R.: 1976, *Language, Memory, and Thought*, Lawrence Erlbaum, Hillsdale, N.J.

Anderson, J. R.: 1983, *The Architecture of Cognition*, Harvard University Press, Cambridge, MA.

Block, N. J. (ed.): 1981, *Imagery*, Bradford/MIT, Cambridge, MA.

Cummins, Robert: 1975, 'Functional Analysis', *The Journal of Philosophy* **72**, 741–760.

Cummins, Robert: 1983, *The Nature of Psychological Explanation* Bradford/MIT, Cambridge, MA.

Devitt, M.: 1981, *Designation*. Columbia University Press, New York.

Dietrich, E. and C. Fields: 1986, 'Creative Problem Solving Using the Wanton Inference Strategy', in *Proceedings of the First Annual Rocky Mountain Conference on Artificial Intelligence*.

Dretske, Fred: 1981, *Knowledge and the Flow of Information*, Bradford/MIT, Cambridge, MA.

Dretske, Fred: 1985, 'Machines and the Mental', *Proceedings and Addresses of the American Philosophical Association*, Vol. 59(1), Sept. 1985.

Fodor, Jerry: 1975, *The Language of Thought*, Cornell, New York.

Fodor, Jerry: 1980, 'Methodological Solipsism Considered as a Research Strategy in Cognitive Psychology', *The Behavioral and Brain Sciences* **3**.

Fodor, Jerry: 1981, *Representations*, Bradford/MIT, Cambridge, MA.

Fodor, Jerry: 1984, 'Semantics Wisconsin Style', *Synthese* **59**, 231–50.

Fodor, J., T. Bever and M. Garrett: 1974, *The Psychology of Language*, McGraw-Hill, New York.

Haugeland, John: 1979, 'Understanding Natural Language', *Journal of Philosophy* **76**, 619–32.

Haugeland, John: 1981, 'Semantic Engines: An Introduction to Mind Design', in J. Haugeland (ed.), *Mind Design*, Bradford Press, Montgomery, VT.

Kripke, S.: 1972, 'Naming and Necessity', in D. Davidson and G. Harman (eds.), *Semantics of Natural Language*, Reidel, Dordrecht.

Marr, David: 1982, *Vision*, W. H. Freeman, San Franscisco.

Moore, T. E. (ed.): 1973, *Cognitive Development and the Acquisition of Language*, Academic Press, New York.

Newell, A. and H. Simon: 1972, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, N.J.

Norman, D. and D. Rumelhart (eds.): 1975, *Explorations in Cognition*, W. H. Freeman, San Francisco.

Putnam, Hilary: 1975, 'The Meaning of 'Meaning'', in K. Gunderson (ed.), *Minnesota Studies in the Philosophy of Science*, University of Minnesota **7**, 131–193.

Pylyshyn, Zenon: 1984, *Computation and Cognition*, Bradford/MIT, Cambridge, MA.

Searle, John: 1980, 'Minds, Brains, and Programs', *The Behavioral and Brain Sciences* **3**, 417–457.

Smith, Brian: 1982, 'Semantic Attribution and the Formality Constraint', unpublished report, Xerox PARC.

Stabler, Edward: 1983, 'How are Grammars Represented?', *The Behavioral and Brain Sciences* **3**, 391–402.

Stich, Stephen: 1978, 'Autonomous Psychology and the Belief-Desire Thesis', *The Monist* **61**, 573–91.

Stich, Stephen: 1982, 'On the Ascription of Content', in A. Woodfield (ed.), *Thought and Object*, Oxford University Press, Oxford.

Stich, Stephen: 1983, *From Folk Psychology to Cognitive Science: The Case Against Belief*, Bradford/MIT, Cambridge, MA.

Stoy, J.: 1977, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT, Cambridge, MA.

Tanenbaum, A. S.: 1984, *Structured Computer Organization*, Prentice-Hall, Englewood Cliffs, NJ.

Wulf, W., M. Shaw, P. Hilfinger and L. Flon: 1981, *Fundamental Structures of Computer Science*, Addison-Wesley, Reading, MA.

Computing Research Laboratory
New Mexico State University
La Cruces, NM 88003
U.S.A.