The Significance of Computational Psychology

(A commentary on Charles Taylor's "Cognitive Psychology")

Brian C. Smith

1. Introduction

I am sure that full-blooded human mental life is as deep and subtle as Charles Taylor suggests. The story of significance he tells may well be true. But there is a striking contrast between the richness of his account of people, and the naivete of his sense of computation -- a naivete, I will argue, that ultimately undermines his conclusion.

I take him to argue as follows:

1. Certain essential qualities attend human mental life and human action, such as implicitness and self-interpretation -- which derive from the fact that we are in an essential way "subjects of significance".

2. These qualities must occupy a central position in any explanatory theory of human psychology. They cannot be lightly dismissed as "irrelevant" or "epiphenomenal" because they are a crucial part of the very phenomena which psychology has as its task to explain.

3. Such qualities cannot attend the "mental life" of computational processes, or be described in computational terms.

From these three premises Prof. Taylor concludes that computational psychology must inevitably fail -- that no computational account can or will illuminate the workings of the human psyche.

While he argues for the first two premises, he apparently finds the third self-evident. Typically, after identifying an essential feature or quality of the human psyche, he suggests that it is obvious that a computer could not have it. For example, in the discussion that much of human knowledge is and must be tacit, we are told that in contrast "there is nothing comparable to tacit knowledge in a machine". Similarly, after arguing for what he calls strong action in humans, Prof. Taylor claims that "one cannot easily conceive of any basis [for strong action] being found in a machine". Other such statements can be found scattered throughout his paper, but we are never given an argument from first

principles explaining exactly why it is that machines -- or computational processes -- must <;necessarily> lack these qualities so essential to human rationality.

Since I do not myself find this third step nearly so obvious, I will take it as one my tasks to cull from his remarks those which deal with computational matters, and attempt to show that he has no argument that computational notions will prove inadequate to deal <;with what I will agree are necessary components of any theory of psychology> -- tacitness, self-interpretation, "full-bloodedness", and so forth. As a consequence, I must conclude that the most we can take from his paper is a reminder of some of the facts about human cognition -- admittedly serious and difficult -- that any psychological theory is going to have to face.

I am not, in other words, primarily going to disagree with Prof. Taylor's concept of the <;human>. Rather, I will take issue with his concept of the <;computational>.

My technical criticism may be taken as illustrative of a general concern, the relevance of which goes well beyond today's particular discussion. It is surely an elementary fact that the utility of any scientific metaphor is limited by the extent to which the <;base> of the metaphor is itself understood; before we can understand what it is to apply computational metaphors in our study of human cognition, therefore, we need to understand what computation is. But, though we build computational artifacts every day, I submit that as a community, we do not yet have an explicitly articulated understanding of computation nearly adequate to withstand the kind of scrutiny to which it must necessarily be subjected by arguments about what can and cannot be accomplished under its banner.

There are of course certain aspects of computation which are relatively well understood -- such as which functions are formally computable. But discussions of the sort we are engaged in today make much stronger demands on our concept of what computation than that. We are talking today about <;strong> properties of computation: about possible reference for internal symbols, about the relationships between ingredient processes and constituted wholes, about the interplay between implementation and interpretation. The moral underlying my remarks will be that, without adequate accounts of these and a host of other similar computational concepts, arguments about the promises and limits of computational psychology must founder out of simple ignorance.

One more introductory remark: in spite of the critical posture I am adopting, I find myself in very substantial agreement with Prof.

Taylor's concerns. That "the significance feature is at the centre of human life" is a position which I thoroughly endorse, and I share with him a frustration at its apparent neglect in current theories. Although I disagree with his <;argument>, I don't necessarily disagree with his <;conclusion> -- in fact I would love to know whether it is true or not. Furthermore, I feel his paper will make a substantial contribution to those computationlists among us, if it "ups the ante" on what it is to develop an adequate theory, by making explicit facts that we should all know, but that we may all too easily forget.

2. An Argument about Computation?

My first task will be to look briefly at the comments that Prof. Taylor makes about computation. We find in essence five claims, all of which, I will argue, turn out on scrutiny either to be false, or else to seriously beg the question.

2.1: An Ingredient is not the Whole

First, consider the argument that people cannot be computational since people do not generally take themselves to be "decomposing" or "making explicit" any series of ingredient steps, whereas, by assumption, <;a computational process must do these things>. Unfortunately, this assumption is simply false of computational processes: while they may perhaps be <;constituted> of ingredient processes which make a series of explicit moves (although see the discussion of explicitness, below), those are <;underlying> processes -- they are <;ingredients in a composite whole>. This distinction is crucial in computational practice: if in implementing an AI program, I were to fail to distinguish the process as a whole from the underlying LISP interpreter, I would end up spinning in infinite circles. I will mention a number of other examples later in the discussion.

This distinction between whole and ingredient has been remarked on by commentators: indeed, this is the issue about which Dennett "cares tremendously", in apparent contrast with Fodor. But that Prof. Taylor falls to recognize the compositionality of process -- surely one of the hallmarks of computation -- is made evident at the very outset. He says it is the task of what I am calling computational psychology to explain "by some underlying process" how we perform various cognitive activities. So far so good. But then we encounter:

> "to explain the performance would then be to give an account of how <;we> compute these responses, how we

take in the data, process it" etc. [emphasis mine]

which is seriously problematical. A metaphorical story of how we perform cognitive activities, based on current computational practice, would not be a story of how <;we> take in data and process it; rather, it would show how we are constituted of processes that take in data and process it.

A "computation", according to Prof. Taylor, is not something that we are aware of doing -- not something we "could be got to avow and take responsibility for, granted undistorted self-knowledge". Fine, I say. Why should we be aware of it? Computational psychology doesn't say we should, since there is no reason to conclude, just because a computational process does something in some way, that that process itself should in any way be disposed to admit, or know, <;or have any form of direct access whatsoever>, to the manner in which it did it.

Now of course when we get to self-interpretation, Prof. Taylor is going to talk about things we <;do> have to admit to. Again, fine: by the same token, if we are going to develop an adequate computational psychology, we will have to construct computational models which do the same. What aspects of its own constitution a process -- human or computational -- can be made aware of is of course a very substantial question (which we will look at in a few moments). But we have yet to see an argument that constructing one will prove impossible.

As a footnote, I find it odd that Prof. Taylor feels that programs must somehow be capable of "introspective confessions" <;ad infinitum>, especially given his recognition of the "inescapable horizon of the implicit surrounding activity" which must, as he has realized (and, I take it, as Wittgenstein and Polanyi and Quine and others have also realized), characterize human knowledge. The point is that inescapable tacitness is a fact about <;all> symbolic systems: it is as much an ultimate truth about computation as it is an ultimate truth about people.

2.2: The Priveleged Account

A second "fact" that Prof. Taylor mentions regarding computation has to do with his claim that, for people, there is a "priveleged" description which identifies what they <;really> do, as opposed to other possible stories which are merely externally attributed, and which lack this special kind of <;authenticity>. Computers, according to Taylor, obviously lack such a priveleged description. Now while this <;may> be true, surely we have here no argument, but rather a restatement of the conclusion. For <;why> must computers lack privileged descriptions? The claim itself doesn't provide an answer. Prof. Taylor is of course right

that computers <;may> be taken under externally imposed descriptions, but so may people (as his example of lecturing taken as mantric-clicking illustrates). Thus, being subject to external description doesn't distinguish people from machines. Being causally embedded in the world can't be sufficient either, since it is plain that a robot may in actual fact screw a typewriter together. Suppose it turns out to be some kind of self-interpretation that is the special ingredient that makes descriptions priveleged. Then presumably if I construct a self-interpreting machine, it too will be subject to priveleged description.

Of course whether I can do <;that> will depend on whether machines can be self-interpreting. But the point is this: it is not incumbent on <;me> to elucidate what "subject-to-priveleged-description" comes to. Rather, Prof. Taylor must show that it involves something inherently impossible computationally (and I take it that he has not done so). Otherwise this point amounts to a claim that computers can't be full-blooded because they would have to posess some quality which only full-blooded processes can possess. The point merely begs the question, in other words, and his conclusion must depend on what other arguments he can advance.

2.3: Self-Interpretation

A third claim is that while full-blooded human activity involves self-interpretation, computational artifacts could not be self-interpreting. Once again, I must ask "Why not?". Where is the analysis of what it is to be self-interpretive, such that computational processes cannot be so? Surely this too begs the question. I would agree that no extant program posesses any substantial degree of self-interpretation, but nothing in principle follows from that observation of current practice. Furthermore, as opposed for example to emotion, about which I haven't any clues, self-interpretation is not a phenomenon about which we are in total mystery. A great deal of current work in Artificial intelligence just might illuminate such questions. I will have more to say about this later, when we look at what in fact computation comes to. But once again, all I can take away from Prof. Taylor's claim is that self-interpretation is another ingredient in, or aspect of, full-bloodedness. Nothing of computational import follows.

2.4: Tacitness

A fourth recurrent theme about computation is that human action is tacit, whereas computational action is explicit. Now this claim is a little difficult to refute in detail, because it is far from clear what notion of "explicit" is meant. In a few moments we will take this subject

up in some detail: I will suggest five definitions of "explicit", each an eminently reasonable notion in computation, but <;no two of which are equivalent.> But, as it turns out, Prof. Taylor's argument will survive under no one of these definitions. For some of them (including the "temporally composite" sense which Prof. Taylor seems at places to have in mind), it is simply false that computational processes <;are> explicit. If by "explicit" he means explicit <;from the point of view of the process as a whole>, then my denial is particularly strong: on the contrary, the vast majority of computational processes I have ever met have been most extrodinarily tacit. On the other hand, if explicit means no more than "is somehow constituted in terms of explicit ingredients", then I say "so what?"; surely the minimal fact that we are physically constituted makes us explicit too, with respect to <;that> definition. And furthermore nothing would <;follow> from this last kind of explicitness. This can be seen even in the computational case, for the theoretical achievements computer science in no way flow full-blown from the explicitness of programs.

Finally, if the definitions of "explicit" or "tacit" depend in any way on notions of how people do or do not act, then the claim is again reduced a question-begging one, for we would once again have a circular argument.

In sum, I can make no sense of his remark that because "we make no explicit definition of the problem .. there is correspondingly no break-down into sub-problems, or application of procedures". From the simple fact that <;we> don't break our activity down into subproblems, there is no valid argument that there <;is> no break-down into sub-problems. <;Computational processes don't break their activity either>. To the extent that it happens at all, <;we programmers> do it, and we do it in many different ways. Some of these many ways will be discussed in a few moments.

2.5: Theory reduction

Finally, there is the point about theory reduction: that the relationship between adequate psychology and computational accounts must be what Prof. Taylor calls "case III". But this, of course, is a summary of, or corrollary to, his conclusion, not an argument for it. The reduction will be case III just in case computational psychology is impossible. But we have, as I have attempted to demonstate, simply no argument as yet.

3. Five Principles of Computation

But look: we must examine these issues in a little more depth. For one thing, I put myself in a dangerous position, if I merely counter Prof. Taylor's sense of the facts about computation with my own sense of the facts (this is no way for a reasoned debate to proceed). Secondly, it would be nice to take away from the current discussion some facts that will enlighten future analysis. Finally, we have still not considered the obviously serious question of whether his conclusion is <;true>.

I would like, therefore, in this second portion of my commentary, to set out five basic principles about computers and computational processes which I take to be crucially relevant to discussions of the present sort. As we come to each one, we will be able to look from a more informed position at the claims which Prof. Taylor has made about computation, and see not only why they are in error, but also how a more adequate story would go. Instead of just looking at his paper on its own, in other words, I want now to subject his arguments to the light of at least a suggestive theory of computation. I will be forced to conclude that he has in serious ways failed to grasp the very notions he considers so inadequate for psychology.

Principle 1: Computers are not a Natural Kind

First, <;computers are not a pre-existing natural kind>, like some strange variety of horned owl, the properties of which we, as natural scientists, are engaged in uncovering. If there is a coherent notion of computation, it is presumably a professional consensus, residing in the heads of those people who design and use them.

It is possible to argue, of course, that this consensus delineates some cohesive, ultimate concept, analagous to that of number, that transcends our understanding, and that lives, say, in Plato's heaven. It isn't obvious, after all, that numbers are a natural kind either, but we certainly assume there is more to the concept of number than some consensus in the minds of mathematicians. Or, if you don't like transcendence, you might want to claim that the concepts of computation are somehow an inate part of our human rational endowment, of which computer science should be taken as the "rational reconstruction" (although who besides Fodor was born with the concept of a Turing machine wired in?). But views on either the transcendence or innateness of computation would have to be argued, and I have yet to hear such argument. Without them, it follows that there is no apriori reason to think that computation is now, or has ever been, a precisely defined or fixed concept (although there is presumably some agreement, since the field does seem to hang together).

The importance of raising these issues at the moment is this: we need to know wherein lies the responsibility of deciding whether something is computational or not. Suppose you and I disagree on whether some <;theory> is computational because we disagree as to what it <;is> to be computational. Do we settle our differences by conducting a physical experiment? Obviously not. Nor can we retreat to pretheoretic intuition, as we might with numbers or sets. Surely our recourse would be to "ask someone who knows". It seems likely, in other words, that the causal chain legitimizing our use of the word "computation" runs indirectly through the knowledge of experts.

And these experts of course don't necessarily have or need an explicitly articulated theory of computation. As is true in any science, the story of what constitutes their subject matter is part of the implicit background in which they work. Therefore, claims about whether the mind (or anything else, for that matter) is or is not computational are going to be difficult to defend -- and certainly aren't going to be <;obvious> -- without somebody's first undertaking the substantial task of providing an adequate definition of computation, sufficient to explain -- in a manner which reflects expert usage -- the concepts that one encounters in "the field": concepts like <;represent>, <;symbol>, <;interpret>, <;compile>, <;procedure>, <;program>, <;process>, <;code>, <;state>, and so on and so forth.

I actually have a tentative working definition of computation:

> Computation is a notion which applies first and foremost to a particular class of <;processes>, namely, to those which can be understood as <;compositionally constituted of ingredient processes interacting with abstract symbols that we, as external observers, interpret as significant>.

While I have no intention of defending the empirical adequacy of that definition, or even of spelling it out in detail, there are a few comments that can be made. First, it is tentative (I cannot over-emphasize how inchoate and preliminary I consider <;all> explicit accounts of computation to be). Second, it is <;strong>, in the sense that it defines computation in terms of the <;internal structure> of processes, not simply in terms of their surface behaviour (computationalists, in other words, are not behaviourists about their domain). Third, as formulated, it makes use of semantical terms: "symbols" and "significant". I am defining computation, in other words, with respect to concepts which may themselves bear only a "case III" reduction to electronic physiology.

Principle 2: Computation is a Special Science

This last point leads us to our second principle: there is no simple relationship between computational predicates applying to abstract processes, and physical predicates applying to the machines on which those computational processes are implemented. In particular, it is by no means type-type in Fodor's sense. An account of how any given process is implemented in hardware can be a vastly complex, time-dependent, and disjunctive, the simplest description of which is often the program for the interpreter for the language under consideration, which can often be hundreds of pages long. And this is only in a particular case -- general "law-like" relationships are as unlikely as Fodor's despaired-of law-like physical predicates for money. In other words computation, as various commentators have realized, is a "special science".

Adding to the complexity is the fact that computational processes are implemented not only in hardware, but also in terms of other processes, and the relationships between the latter aren't type-type either. One of the questions I want to raise later is what happens to such semantic notions as reference when you cross from one of these processes to the one in terms of which it is implemented. There are some non-trivial technical problems: for one thing, the identity of objects is generally not preserved across an implementation boundary: it is only "whole systems" which are implemented (they are connected by "total theories", so to speak). What counts as a unique object at one level may disappear into some disjunctive fragment of the whole organizational texture of the system-wide encoding. Further, even if one object at the lower level "is" (the reduction of) some higher level object at any given instant in time, that may be false one second later.

Let me give an example of this latter case. Consider a typical implementation of LISP, for example, and suppose that a LISP object L1 (say, an atom) at some instant is "reducible" to a machine-language object M1 (say, a location in a sequential memory). It may be possible to watch the behaviour of this machine for one minute both under its description as a LISP system, and under its description as a machine-language system (remember, this is <;one and the same artifact> -- just two ways of describing it). At the end of this minute atom L1 may now to reduce to some <;different> machine-language object M2 (suppose, for example, what is known as a "garbage collection" has revamped the mapping of objects). Now suppose that this "sequential memory" in which M1 exists is not implemented directly in hardware, but is run-length encoded in PASCAL lists, which get copied over every time a memory access is made. Not only is there now no lower-level object reducing L1, but to the extent it is encoded in a functional state of the machine, the predicate identifying it

is changing every single machine instruction.

The point is this: there may easily be half a dozen such levels of implementation between the computational process a user interacts with, and the one implemented in hardware, no one of which has special claim to be the <;real> description of what is going on. Establishing the identity of abstract symbols, by looking at hardware, would be about as possible as establishing the identity of word tokens, in terms of a predicate on air molecules.

In light of this fact we may look at Prof. Taylor's argument that, because we are not aware "at the top level" of any decomposition to our skilled performances like catching a ball, then if these processes are computational they

> "would have to be an underlying process, on a par with -- or indeed, identical with, --- the electrical discharges in brain and nervous systems".

There are of course several problems with this: we have already dealt with the false assumption that we would <;know> about the decomposition if there were any decomposition to be made. But this brief story about implementation hierarchies also denies the conclusion that, if a computational account isn't apparent to the overall process, then it must be on a par with physiology. This simply isn't true.

Principle 3: Turing Equivalence is Weak Equivalence

Our third principle is that <;Turing-Church equivalence is weak equivalence>. Of course any notion of equivalence must rely on some metric: the metric used in the proofs of equivalence between the systems of Post, Church, Turing, Markov, etc., is that they are able to compute the same functions. All that Church's thesis amounts to, therefore, is that any <;behaviour> definable on one machine can be defined on any other. No notion of "strong" plays a role in deciding whether the two machines are Turing equivalent (it is perhaps not surprising, in light of this, that the Turing test for intelligence is a test of weak equivalence). This makes sense in terms of the implementation story we just gave, since Turing equivalence is generally demonstrated by showing how each machine can be implemented in terms of the other, and, as we have just seen, strong properties aren't preserved across implementations.

Now it is clearly one of Prof. Taylor's principal views that to say of something that it is full-blooded is to say something <;strong>. Hence, arguments depending on intuitions garnered from the operations of

Turing machines will fail: it is patently obvious that no strong argument advanced about any one member of a weakly equivalent class will necessarily hold of any other.

Thus, the notion "equivalent to a Turing machine" will simply not serve as the definition of computation we so urgently need for discussions of this sort. If Turing equivalence were all there was to computation, we could eliminate the word "computer" from our vocabulary, and simply use "machine". But of course there <;is> more to computation than Turing equivalence, and not only that: both Prof. Taylor's arguments, and my own definition, deal crucially with such strong attributions. It is for this reason that I will be able, in the final portion of this talk, to begin to analyse how the requirements he sets forth might impinge on a computational model.

Principle 4: A Process is not its Intepreter

The following are not the same: a computational process, and the ingredient process which interprets the programs in terms of which the other process is defined. The latter is informally (by which I mean in computational circles) called the "interpreter" (but note that this is a different, if related, notion to the one, used in model theory and semantics, that deals with positing properties of signifying to "symbolic" objects).

Imagine saying of a car engine that <;it> decides how much gas it needs by measuring the incoming air pressure. This informal manner of speaking is literally incorrect: the engine comprises, among other things, a fuel monitoring system, and if anything "decides" how much gas is needed, it is this ingredient monitoring system. Further, you <;know> that in saying "the engine decides", you don't mean the whole engine. For if the engine sputtered because the system malfunctioned, would you admit that the engine was causing itself to run badly? Do we have self-reference from Detroit? Of course not.

It is exactly analogous, and equally incorrect, to say of a computational process that <;it> manipulates its internal representations. This we noted earlier, but we can now see what it comes to. Specifically, a (serial) process is <;constituted> of an interpretive process that examines (manipulates, etc.) representations (often called programs and data structures), and in virtue of that manipulation, the process as a whole is constituted. This distinction, which is tacitly but universally understood in computational work, is particularly serious for Prof. Taylor, because it interacts intimately with such issues as self-interpretation and consciousness. It also relates to the

interpretation hierarchy we just described, which, to avoid infinite regress, is grounded at some point by constructing a <;weakly equivalent> physical device to mimic the behaviour of the lowest interpreter in the chain.

I submit that interpretation of this sort is one of the central notions of computer science (as opposed, by the way, to <;compilation>, which is a red herring and of no import for current metaphorical theorizing). Perhaps because it is so very central, it is often tacitly assumed. I recall once reading a paper of which the first sentence went something like: "When two people engage in discourse, they wander around their respective mental maps". Now I am certain that the author didn't mean <;they> wandered around their mental maps: that just doesn't make any sense. Rather, the suggestion was that a possible explanation of discourse is being constituted of an interpreting process wandering around a mental map. You can't write programs if you are confused on this issue (the relationships between micro-code and macro-code, the distinctions between meta-circular interpreters vs. bootstrapping interpreters, and so forth, demand that one keep these separate).

It is part of my emerging theory of computational ontology that, complex though it may be, it is only by taking this chain of interpretation to be the essence of computation one can begin to develop serious definitions of the other theoretical terms of art: a "program" is then a representation, for the interpretive process, of instructions to obey in constituting the overall process; a "computer" is a physical device so constituted that its behaviour can be taken as the interpreter for a class of computational processes, etc. Much of the confusion that arose over the infamous procedural/declarative distinction turns on a failure to recognize how such questions must be relativized to a specific interpreter. But these are mere glimpses into a subject that demands a book, not a paragraph.

A confusion in Prof. Taylor's paper deriving from his failure to grasp this interpretation of a program by an ingredient interpreting process can be seen in the comment that representation must be

> "separable from the stuff which it monitors, or of which it is a symptom. If it plays any role in explaining these processes, it must be in interacting with them. Since interaction is ruled out on materialist assumptions, it cannot be allowed any explanatory role at all."

Now I must confess that I fail to understand what Prof. Taylor is getting at in saying this; perhaps I am just too rooted in my computational

background. For, by my analysis, if a computational process is <;anything>, it is a process in which a representation <;of> behaviour plays an ingredient causal role in <;effecting> that behaviour. This is just what we defined a program to be: a representation, to be inspected by the interpreting process, of what the overall behaviour is to consist in?

Principle 5: There is no single meaning of 'explicit'

Our fifth principle (I call it a "principle" mostly because of its importance) summarizes a point made earlier: there is no single, clear notion of what "explicit" means with regards to computation. Therefore, before you can possibly defend any statement like "everything of causal import in a computational process must be explicitly represented in the data structures", or even "if it is a computation it must be explicit", you must provide a definition of the kind of explicitness intended. The problem is that there are simply too many quite plausible candidates that under scrutiny turn out to be very different.

Because of the importance of this point, I am going to take the next few minutes to suggest a half-dozen or so possible kinds of "explicitness" that strike me as the most reasonable. These won't, however, be <;definitions>, but rather pointers to areas where it seems to me that a coherent notion of "explicit" is lurking. I frankly don't yet know how to tighten any of these into a real definition that would withstand scrutiny.

E1: One possible meaning of "explicit", one in fact that Prof. Taylor may have in mind, is the notion of being <;temporally composite>. This is strongly suggested by his comment that for a process "to reach an answer by computation is [for that process] to work it out in a series of explicit steps", and his further remark "you wouldn't say someone was computing, if he gave the answer straight off wihntout any analystical reflection." But it is simply not true of current practice that for a machine operation to be called a "computation" it has to be <;temporally> decomposable into identifiable constituents. For one thing, if this were true, all <;primitive> machine instructions would have to be denied the status of "computations", which would be very strange. More seriously, programmers often spent weeks designing computational artifacts so that certain particular operations or "computations" will take place in what is called "unit" time. This is, for example, standard practice in data base inversions, knowledge representation inference systems, etc. It is also one of the motivations behind the development of parallel machines executing marker-passing algorithms, such as those of Woods and Fahlman and so on.

E2: Another possible meaning of "explicit" is that we identify explicit causal relationships in our analysis of the functional structure of the overall system. It should be obvious that <;causal> and <;temporal> explicitness are not the same. I may for example identify quite a few distinct causal links between the shift lever beside my seat, and the gear my car is in, but it doesn't follow that I should expect discernable delay between my shifting that lever and the car's changing gears.

E3: Yet another possible meaning (and this is one of the most "obvious" with respect to computation, and may be what Prof. Taylor had in mind) would go something like "explicitly represented in the program. Several comments are in order about this. First, I doubt that it would be trivial to define just what this intuitive kind of explicitness came to. Are the organizational relationships among the expressions to be counted as part of the overall explicit structure, for example? How about the syntactic structuer? Or the immediate deductive or operational consequences? Or the <;possible> deductive or operational consequences? Or the meaning? Which of these properties are <;explicitly> born in the expression? Perhaps there is no real answer -- you could simply decide.

Second, because what <;program> you are talking about depends on what <;interpreter> you are refering to (remember, we just established this -- any given process may be simultaneously described as running a variety of very different programs), you would have to identify which program you mean. What is explicit for one may be implicit for the other. For example, suppose I take sides on a current debate in AI and decide that visual images are represented in terms of something very like a visual array, and define a process embodying this assumption. But then suppose that I need to implement this process, and, in order to do so efficiently, I implement the visual array by some clever coding scheme, so that I don't need to actually <;have>, in some sense, array elements for each bit. This is in fact no idle fantasy -- it is done in the software supporting many of the fancy printers and terminals that we use all the time. There are various encoding schemes that could be used: I might for example maintain descriptions of the shapes of black areas, and then if I want to know whether any bit is on, I will see if it must be part of one of these shapes. If so, it is black -- otherwise it is white. (Perhaps if I use the kind of ingenuity we were told about yesterday, I may be able to do this in linear time.) Now I have a machine which can be said to stores images either as arrays and as descriptions, depending on which description you look at it under. It doesn't do <;two> things -- it does one thing, which can be described either way.

E4: Of course yet another possible notion of explicit has to do with what is explicit <;from the point of view the process itself> -- <;introspectively available>, so to speak, for the computational process as a whole. Because of his constant reference to the fact we don't <;take> what we are doing to be computational, this may be partly what Prof. Taylor had in mind. But this, as I pointed out above, is unrelated to any of the foregoing. In fact, this kind of explicitness is an enormously difficult <;goal> in computational practice, which we rarely achieve.

E5: Another potential meaning of "explicit", again unrelated to any of the foregoing, might go something like: "understood in terms of an explicit theory". Note that one cannot identify the explicitness of programs with explicitness of theories describing those programs. To think this would be as naive as to expect to find fully-developed physics theories sitting around in the external world. Take a computational example: there has in recent years been an interesting development of understanding of the relationship between so-called <;iterative> and <;recursive> programs, in which it was discovered that various of our sort of implicit notions of recursion weren't equivalent, that one of them which was thought to be mutually exlusive with iteration was not, at least by one account of iterative. A notion of "tail recursion" was proposed, to help in explaining cases where the form of the <;program>, taken as a composite expression, was recursive, but where the control regime implied by that program, was iterative. Basically, papers were written, ideas were explored, and finally an appopriate theoretical analysis took shape. What was striking -- and this is why I raise this -- was that programs embodying all the qualities described in terms of this theoretical structure had existed well before the debate started.

In sum, if tacitness is the opposite of explicitness, then the comment that "there is nothing comparable to tacit knowledge in a machine" must simply be taken as under-determined. Further, on many plausible reconstructions of what it might mean, it must be taken as false.

Summary

These five principles, I suggest, are a standard, if tacit, part of the intellectual tool kit of every working programmer. This is not the place to delve any further into pure computational ontology, although ultimately, of course, I would like to defend this claim by analysing computational practice, and to show in addition that all these principles follow from the definition of computation I proposed. But in the meantime I must rely on your acceptance of my informal arguments. But most importantly, I hope that this much of a glimpse has demonstrated both how

complex a subject matter we have at hand, and how much needs to be understood before one can develop careful arguments about the possibility of computational psychology.

Postscript:

I must pause in this analysis to consider a potential objection that I am not being fair to Prof. Taylor in that I am not taking seriously what he means by the term "computation". But I think this derives from a confusion -- a confusion that I sometimes felt Prof. Taylor himself was making, although to be honest it is not completely sure from what he says. Because of that possibility, however, I want to mention it briefly.

He several times mentions a notion of what people call "computing", which is different (so he says, probably correctly) from what machines do, and which is consequently unlikely to be explained by mechanistic metaphor. Because of this, I take it that one way of reading his overall argument that full-blooded human action cannot be computational, would be to take him as saying that most full-blooded human action is not the kind "computing" that people do when we say that they are "computing".

But surely to take this reading seriously is simply an error. Look: no one has suggested -- and no one, I take it, could reasonably suggest -- that when we are doing what we call computing (that very explicit "problem-solving" kind of thinking), that <;that> is the basis for the computational metaphors currently being applied in cognitive psychology. Among other things, to take a psychological phenomenon -- and one we don't understand, at that -- as an explanatory principle on which to construct psychological theories, would be both circular and crazy.

And even if we were to accept this move, then what would be Prof. Taylor's argument? That for principled reasons computers cannot really compute? And therefore, since <;real computing> is what we want to ground psychology in, we shouldn't use machines as theoretical intermediaries in our research? This definition of the word "computing" would leave machines, normal thinking, and problem-solving, all unrelated.

In fact, presumably, Prof. Taylor's argument is that psychology based on what <;machines> do will prove insufficient to deal with full-bloodedness. And it is therefore <;machine computation> that we have to examine. I take it that the only relevance of human "computation" to Prof. Taylor's argument (and this he <;does> imply, in a manner that is both coherent and consistent with his general position) is that machine computation will in all likelihood <;not even be able to explain that

psychological phenomenon we call human computing, which you might expect to submit to such explanation, even if other human actions didn't>. The argument for this position, presumably, is the usual one: that human computation is full-blooded, and <;no> machine can be full-blooded.

Hence, I take it that the critique that I have given, which doesn't deal at all with human computation, but attempts to show instead that we have as yet no argument that machine computation must fail to be full-blooded, is the correct one to mount, and relates to human computation in the same sense that it relates to all full-blooded activity.

4. Self-interpreting computational systems

*** OMIT ***

I want to take the remaining few minutes to briefly explore the computational import of Taylor's account of psychology, <;completely apart from his own notions of computation>. It would be too easy, and ultimately uninteresting, simply to say that his conclusion is not supported because his conception of computation is inadequate, since the mere fact that I don't find his computational arguments compelling doesn't tell me whether the very real requirements he has set forth for an adequate theory of psychology could ever be met in a computational framework. The question of substance, in other words, is whether his conclusion is <;true>, given that we accept his account of human psychology, and given what is in fact the case about computation. I have already claimed that none of us know the <;final> answer to that question, but surely it is a question that we owe it to ourselves and to Prof. Taylor at least to look at.

The ultimate challenge, I take it, would be to analyse what it would be to construct a full-blooded comptutational artifact. But of course I can't possibly do that, since I don't yet fully know what full-bloodedness comes to. However, it is part of Prof. Taylor's argument, presumably, that, among other things, properties of tacitness and self-interpretation will be central, as will be being a subject of privileged description. I have argued that I don't understand any interesting definition of tacitness which computers don't already possess, so I will concentrate here on the other question: what it would be to construct a computational process which was self-interpretive.

For purposes of the discussion I am going to need to assume that we can construct some computational process P which, <;in the shallow sense>, can be said to <;say> something. Remember, this is <;shallow> --

I don't need you to admit to any kind of strong, full-blooded "saying" on its part.

Now before I can set the stage that I need, I must digress briefly about the subject of a computer "saying" something.

... thoughts and language (if not action) have a quality of what I will call <;primary significance>, in that they are fundamentally <;about> something, in a way that catching a ball is not. We may not know much about reference or truth or meaning or denotation, but whatever they are, they are qualities that relate first and foremost to language and thought, to <;symbolic> or <;linguistic> behaviour. ...

... What is important about this distinction can be summarized as follows: First, it is <;not> a distinction with which Prof. Taylor is at least explicitly concerned (I identify it partly in order to contrast it in a moment with the more complex notions of significance on which he does concentrate). Second, it <;is> the significance distinction on which most other current discussions of the possibility of computational psychology appear to focus. The subject of primary significance in fact seems to be rather popular these days: it is the essential subject matter of model theory and denotational semantics; it is the key concept that people like McDermott and Hayes find lacking in much of AI; and it is the feature of cognition that Fodor claims computational pschology cannot account for. In addition, as I will argue in the next sections, it is a distinction which plays an essential role in our understanding of computation, and as such it will come to the fore when we start inquiring into the possibilities of computational self-interpretation.

... It also seems clear that primary significance must play a central role in any adequate theory of psychology. For, in spite of Fodor's claim to the contrary, (and as some of the respondents to his <;Brain and Behavioural Sciences> article point out), the import of mental life's primary significance is not easy to escape. I take it, in particular, that the very concepts of <;formal> and <;syntactic> make implicit reference to the potential primary significance of their subjects, merely by avoiding it: Fodor's own working definition of syntactic as "a way of <;not> being semantic" surely betrays this. Surely you cannot claim to treat a pine cone formally or syntactically without thereby admitting that you have some notion of significance which you are temporarily setting aside.

In other words, the terms "formal" and "syntactic" may indeed mean something like "shape", but if they are ultimately going to prove useful they will surely select, out of the range of possibilities, <;just

those aspects of an object's shape in virtue of which it bears its primary significance>. Somehow, in refering to the formal properties of a token of the numeral "1" (in stating, for example, that my computer adds numbers by considering only formal properties of numerals) I wouldn't take myself to be refering to such features as the thickness of the serif <;unless I could attribute significance to that feature>.

Thus, merely be being <;formal>, an account of psychology doesn't finesse primary significance altogether. In addition, Prof. Taylor's story of why self-interpretation must be an integral part of the subject matter of psychology would apply equally to primary significance. In particular, since whether I smile at you is likely to depend on whether I think that you were the one who took my raincoat, <;some> aspects of the references of thoughts will simply have to be included in any psychological theory, if no more than to assume <;that> thoughts refer, and <;that> some thoughts are about the same referents as other thoughts.

But end of digression, and back to our putative process P. For example, suppose it ... in fact catches a ball (i.e., suppose it moves its mechanical arm in such a way that a ball ends up resting in its grip). Now according to my definition of "computational", it follows that P is constituted of an interpretive process IP manipulating a field of symbol structures that I will call S. They are symbols which <;for us> signify in the world in which P is embedded -- which is to say, we take them to signify trajectories or balls or arms or whatever. In other words, <;IP> is an ingredient process which interacts with symbols (S) that for us represent/encode/signify/whatever the actions and beliefs we (again as external observers) attribute to <;P>.

Now I take it that all of this is simply what people in AI do when they construct programs to do things. I.e., if you are disposed to challenge anything yet, then you are challenging the <;assumptions> of my argument, for we have not yet made any move to self-interpretation.

But now I will make my move. Suppose I design IP in such a way as to interact with symbols S by manipulating instead symbols S' which refer to symbols S. Now a host of questions spring up demanding clarification. First, in what sense does S' denote S? For now, suppose simply that we external observers take them to do so. I will strengthen this in a moment.

Now there is actually much current research in AI which operates just as I have described (in fact it is really tremendously popular, for lots of technical reasons I won't go into here). In most of this work, the relationship between S' and S seems to have no real causal substance. The programmer provides what he or she takes to be a model of how he or she

takes the program to work. For some purposes (such as providing what he or she wants to call an explanation, IP may "reason" with these meta-descriptive structures. Meta-circular interpreters provided in programming systems like LISP are often of this flavour -- they reflect the operation of the machine, but they are in no way causally related to that machine.

On the other hand, it is possible to design a machine in such a way as to have S' and S be very causally related. In fact, in virtue of changing, say, an element in S' saying that some description in S has been negated, then that symbol in S can in fact be made to be negated. In other words, providing a causally central self-descriptive model is technically possible. [Note: ALMOST!]

... Note quote on P. 4, also, in this machine, doing something and taking itself to be doing it, are relatively indistinguishable ...

Now what I want to ask is this: what is it to say of P (not IP) that it is taking itself to be catching a ball, when IP manipulates symbols in S'. I am not asking <;whether> P is self-interpretive, for I really have no idea. Really what I want to know is, what are we to say about such a P?

... (I will, however, avoid discussing whether such a process could be legitimately said to be either conscious or self-conscious, for two reasons: first, I have very little idea of what those notions mean, and second, I would argue that, in spite of this present speculation, our current understanding of the technical correlates of consciousness are so primitive that we would flatter ourselves to consider it.) ...

First, it seems to me that there is a lot more reason to attribute authentic reference to the relationship between S' and S than to the relationship between S and the world, for the causal chain seems so much stronger. Furthermore, <;if> we are prepared to say of P that it (perhaps shallowly) catches a ball, then it seems that we are forced to admit that it took itself to catch a ball. Of course this "taking of itself to catch a ball" is in some sense shallow also, although the strength of the S'-S reference relationship may offset that. And I suspect that "externally attributed self-interpretation" wouldn't count for much in Prof. Taylor's book. I don't know that it would count for much in my book, either, but I repeat my question: ...

Prof. Taylor says that the argument gets silly when we get towards asking whether computational processes can be conscious. This, he feels, makes it a highly suspect question. My response is that it is not silly,

it is merely confusing if one doesn't make very specific what one is asking, and if one does not have in hand a cogent theory of what computation is. That is all that is hard, I think: given those two (I have just assumed both in the previous discussion, for instance), I think such questions not only aren't silly, they even have answers.

Note that there are tremendous technical questions I haven't addressed. Suppose that instead of catching a ball process P emitted the sentence "There is a ball coming at you.", as a result of (causally) interpreting data coming in though its TV camera. We externally attribute significance to this remark. But if it happened in virtue of IP manipulating expressions we (and it?) takes to refer to that sentence, is it requisite that it represent, among other things, the <;fact> that that sentence refers to that ball? I.e. must reference relationships themselves be represented (say, named)? And if so, what is it to causally represent a reference relationship? This is just one of a whole host of thorny technical questions that arise in constructing such systems.

*** END OMISSION ***

5. Conclusion

I want to review the three themes that have been woven together in this commentary.

First, it has been my goal to show how badly arguments about what can and cannot be computational fare without a prior theory of what it is to be computational. I hope that the many examples I have mentioned will have substantiated this position. But before closing let me offer one last hint at how complex this situation is. What I will do is to identify the many categories of object or relationship which would seem to have to be explicated, before, as I understand it, anything which could be called a "theory" of computational mentalese semantics could be defended.

First, there are the symbols which the interpretive process will manipulate -- the expressions of mentalese, so to speak. Second, there are the <;external> expressions emitted and received by the program, if, say, it is designed to communicate in something like language. Third, there is the notation you use to signify these internal symbols (and, while you might not expect this, the relationship between the notation and the internal symbols is anything but trivial -- as must be obvious to anyone who has ever tried to explain to a LISP programmer why printing out a structure and reading something back in plays havoc with the two varieties of the identity predicate.) Fourth, there are the operations

which your interpreting program can effect on the structures (this, I take it, although I am no expert on such matters, is what Scott/Strachey style semantics deals with, in terms of abstract functions). Fifth, there is the denotational semantics you attribute to the internal symbols (the semantics that would be illuminated by something like a model theory for the declarative expressions in the mentalese).

Then, suppose in addition that you implement your program in terms of some underlying language. New versions of all five of these categories will be relevant for the underlying language. After identifying them, you would presumably want to clarify all the relationships across this implementation boundary. The denotational semantical ones I suspect will be paricularly tricky. I submit that we have a certain amount of homework yet to do.

A second theme that has permeated my arguments is the argument that a full understanding of the interplay between a <;computational process> and the <;ingredient processes in terms of which it is constituted> is a crucial prerequisite to any application of computational metaphors to psychological phenomena. This distinction has served as a double-edged sword in my attempt to carve a critical path through this material. First, I have argued that many of the false claims of incompatibility between tacit and explicit computations derive from among other reasons, a failure to distinguish what are fundamentally distinct processes. (I think this failure characterizes John Searle's "comments on AI", also, but we can look at that this afternoon.) In particular, what is tacit for one process may be explicitness itself to that process's interpreter. But by the same token, I would use this distinction to argue that much of what is advertised in AI to be self-interpretive behaviour fails because of exactly the same confusion. With this latter claim I suspect that Prof. Taylor would agree.

Separating these two notions, however, has its cost: I lay myself open to the criticism that I -- and computational theorists in general -- will thereby fail <;even more thoroughly than Taylor imagined> to be able to deal with self-interpreting, "full blooded" rationality. But my response to that is that that remains to be seen -- the deep causal connection between that interpretive process and the computational process which it effects may enable that challenge to be met.

A third way to summarize our discussion is as an account of various kinds of theory of human mental life we can develop, and as an exploration of the kinds of relationship, or reduction, we may expect to find among them. In order to review how our arguments have gone, let me adopt Prof. Taylor's terminology, and distinguish between simple case II

reductions, "multiplex" case II reductions, and full case III reductions. Suppose in addition that I identify the following three kinds of theory of the human cognitive apparatus:

T1. A <;physiological> theory (biological, electronic, whatever);

T2. A <;computational> theory (say, an idealized extrapolation of the kind pursued in AI).

T3. A <;psychological> theory (by which I mean one explanatory of our lay conception of our human rationality).

Fodor's position, according to Taylor (and I would agree), is that T2 is the best T3 we are going to get. Additionally, he (Fodor) claims that the reduction from T2 to T1 (and hence from T3 to T1) is one of token physicalism, Taylor's multiplex case-II. Taylor challenges Fodor on the relationship between T3 and T2, arguing that, far from being one of plausible identity, it is likely itself be case III. It is a much further cry, according to Taylor, from psychology to computation than it is from computation to physics.

I have argued as follows: First, regarding T2-T1: I think it is at least possible that it is more serious than either Taylor or Fodor realizes; while I am not convinced that it is case III, I am not convinced that it is <;not> case III, either. Fundamentally, the problems is that I think that notions of reference may lie behind more computational jargon than people realize. And if reference is required in order to identify interpretive processes, and if interpretive processes are required in order to identify symbols, there may be no case II reduction to physics even in the computational case, without a prior reduction of reference, and the reduction of semantics to physics, I take it, is at least a good candidate for case III.

But no matter. This can perhaps be explored. The important point is that Taylor's claim, that T3-T2 must be case III, is a claim for which I as yet see no argument (although I must reiterate that I think it is possible). My basic claim is that in spite of my experience with computation, and in spite of what I have read, I don't think I know what computation is well enough to answer this question. I think computation may be closer to psychology, and further from physics, than anyone realizes, and certainly than Taylor wants to admit. In fact, computers, I suspect, may be a projection into formalism of external human cognitive abilities -- i.e., they may already be metaphorical in essence. Who knows -- perhaps T2 will turn out to reduce to T3. <;Then> what will we say?