Philosophical Studies (2006)
DOI 10.1007/s11098-005-5385-4

GUALTIERO PICCININI

COMPUTATION WITHOUT REPRESENTATION[1]

ABSTRACT. The received view is that computational states are individuated at least in part by their semantic properties. I offer an alternative, according to which computational states are individuated by their functional properties. Functional properties are specified by a mechanistic explanation without appealing to any semantic properties. The primary purpose of this paper is to formulate the alternative view of computational individuation, point out that it supports a robust notion of computational explanation, and defend it on the grounds of how computational states are individuated within computability theory and computer science. A secondary purpose is to show that existing arguments for the semantic view are defective.

1. THE PROBLEM OF COMPUTATIONAL INDIVIDUATION

In some fields, such as computer science and cognitive science, there are scientific theories that explain the capacities of a mechanism – respectively, the computer and the brain – by appealing to the computations it performs. This paper pertains to the individuation of computational states, inputs, and outputs within such scientific theories. For short, I will mainly talk about computational states; the same conclusions apply to computational inputs and outputs.

The received view is that "[t]here is no computation without representation" (Fodor 1981, p. 180). The reason usually given is that computational states are individuated, or taxonomized, by their semantic properties. The same point is sometimes made by saying that computational states have their content essentially.[2] I call this the *semantic view of computational individuation*. The semantic view may be formulated in stronger or weaker forms. In its strongest version, all and only semantic properties of a state are relevant to its

computational individuation. Probably, no one subscribes to this version. In weaker versions, either *all* semantic properties or *only* semantic properties of a state are relevant to its individuation. The weakest, and most plausible, versions maintain that a computational state is *partially* individuated by *some* of its semantic properties (and partially by non-semantic properties).[3] Although supporters of the semantic view have rarely distinguished between weak and strong versions of their view and have not specified in great detail which semantic properties and which non-semantic properties are relevant to the individuation of computational states, this will not matter much here. In this paper, I argue against *any* version of the semantic view.[4]

In this paper, I propose an alternative to the semantic view, which I call *functional view of computational individuation*. According to the functional view, computational states are individuated by their functional properties, and their functional properties are specified by a mechanistic explanation in a way that need not refer to any semantic properties. A mechanistic explanation is a description according to which a mechanism (e.g., the human body) has certain components (e.g., the heart), the components have certain functions (e.g., pumping blood) and are organized together (e.g., the heart is connected to the arteries in such and such a way), and the mechanism exhibits its capacities (e.g., blood circulation) because it is constituted by the relevant components, their functions, and their organization.[5] Before proceeding, a few caveats are in order.

First, the issue of computational individuation should not be confused with the issue of which properties of computational states are causally efficacious within a computation. Here, the received view is that computational states are causally efficacious by virtue of properties that are *not* semantic. According to this view, which may be called the *non-semantic view of computational causation*, computational processes are "insensitive" or "indifferent" to the content of computational states; rather, they are sensitive only to some non-semantic

properties of computational states. The properties to which computational processes are sensitive are often labeled as "formal" or "syntactic."[6] It remains to be seen how computational states are to be individuated.

Second, the issue of computational individuation should not be confused with the issue of whether computation is sufficient for content or intentionality. Some critics of computational theories of mind maintain that being a computational state is insufficient for having original, or non-derived, intentionality (e.g., Searle, 1980; Horst, 1996). They assume that computational states are not semantically individuated and argue that (non-semantically individuated) computational states cannot have original intentionality. These authors, however, do not offer a fully worked out alternative to the semantic view of computational individuation. One of them has even argued that there is no observer-independent way to individuate computational states (Searle, 1992). In response, many supporters of the computational theory of mind have retained the view that computational states are individuated, at least in part, by their semantic properties.

Although I reject the semantic view of computational individuation, I will remain neutral on whether being computational is sufficient for having original intentionality. That depends on what original intentionality amounts to – something on which there is little consensus. The functional view of computational individuation does not entail that computational states have no content – they may or may not have content. Nor does it entail that being computational is insufficient for having original content or intentionality – perhaps some aspects of original content supervene on computational properties. What it does entail is that, if a computational state has content, then, to the extent that such content does not supervene on the computational properties essential to the state, the computational state does not have its content essentially.

Third, I reject the most popular methodology adopted by participants in the current debate. At least since Burge's

"Individualism and the Mental" (Burge, 1986), the principal battlefield in this area has been the correct interpretation of Marr's theory of vision (Marr, 1982). Unfortunately, as some participants have noted, there may be no fact of the matter, for Marr was not explicitly concerned with philosophical issues about the individuation of computation. More importantly, as distinguished as his work is, Marr was only one person. What if his theory was wrong or confused in this respect?

Instead of providing another Marr commentary, I will focus on the way computational states are individuated within the scientific practices that are the source of the modern, rigorous notion of computation and computational explanation. The modern notion of computation, on which computational explanation in psychology and neuroscience is based, originates in computability theory and computer science. In getting clear on the individuation of computational states, those are the scientific practices that we should examine.

In moving forward, we must avoid a common but serious mistake. Many readers, especially those familiar with computer science and computability theory, will readily agree that in those disciplines, computational states are individuated by their formal or syntactic properties. Since anyone who has taken an introductory logic course knows that syntactic properties are distinct from semantic properties, these readers will be tempted to see the functional view of computational individuation as old news. Surely, such readers will conclude, every philosopher worth her salt must acknowledge that computational states are individuated syntactically rather than semantically.

But this is far from the case. The semantic view is, indeed, widely shared – explicitly or implicitly – among those who have taken a stance on the individuation of computational states in print.[7] The explanation is not that philosophers of mind are generally unfamiliar with computer science and computability theory. Rather, supporters of the semantic view, as we shall see later in more detail, have reasons for

their view. One such reason is that it has seemed difficult to give an account of syntactic properties without appealing to semantic properties (cf. Crane, 1990; Jacquette, 1991; Bontly, 1998). This is where mechanistic explanation comes in.

I don't know how to tell whether a property is syntactic. In addition, although the notion of syntactic property plausibly applies to at least some computational states that are made out of strings of symbols, I doubt that it applies in any interesting sense to monadic computational states, such as the internal states of Turing machines (see below). Finally, even when the notion of syntactic property applies, I doubt that it can be used to individuate computational states as finely as we need. Therefore, I am not arguing that computational states are individuated by their syntactic properties. Rather, the functional view accounts for computational properties directly in terms of functional properties.

The functional properties that are relevant to computational individuation are the presence in a mechanism of certain components (such as memories, processors, etc.), relevant relations between components (such as the transmission of signals), states of those components (such as letters from a discrete alphabet and monadic digital states), and functions of the components (such as performing operations on letters). My contention is that the appropriate kind of mechanistic explanation is sufficient to individuate computational states without appealing to either semantic or syntactic properties.

Perhaps syntactic properties are a specific subset of computational properties, and perhaps some semantic properties can be partially explained in terms of some computational properties. If so, then the functional view of computational individuation offers resources for an account of syntactic properties and at least some aspects of some semantic properties. I will not explore these matters here, and I will remain neutral about them.

The only alternative to the semantic view that is clearly stated in the philosophical literature is that computational states are individuated by their causal properties (Chalmers,

1996; Copeland, 1996; Scheutz, 1999). But causal individuation, without constraints on which causal powers are relevant and which irrelevant to computation, is too weak. It does not support a robust notion of computational explanation – the kind of explanation that is needed to explain the capacities of computers, brains, and other putative computing mechanisms in terms of their putative computations.

Supporters of the causal individuation of computational states readily admit that under their view, every state is a computational state and every causal process is a computation. But this is tantamount to collapsing the notion of computation into the notion of causal process. In this paper, we are not concerned with causal processes in general but with the specific processes that are invoked by computer scientists and cognitive scientists to explain the capacities of computers and brains. Hence, we need a more restrictive notion of computation. Here is, again, where mechanistic explanation helps. For mechanistic explanation gives us resources to distinguish between explanatorily relevant and irrelevant components, functions, and causal relations. When supplemented by an account of functional properties that are relevant to the individuation of computational states, mechanistic explanation provides the basis for a notion of computation that does not collapse into the notion of causal process.[8]

The primary purpose of this paper is to formulate the functional view of computational individuation, point out that it supports a robust notion of computational explanation, and defend it on the grounds of how computational states are individuated within computability theory and computer science. This will be done in the next section. A secondary purpose is to show that existing arguments for the semantic view are defective. This will be done in the following section. Along the way, I will hint that the notion of functional property employed by the functional view can be legitimately fleshed out by supplementing mechanistic explanation with an account of the functional properties that are relevant to computation. In this paper, I do not have room for a detailed

defense of the last claim; that will have to wait for another occasion.

2. THE FUNCTIONAL VIEW OF COMPUTATIONAL INDIVIDUATION

In the mathematical theory of computation, abstract computing mechanisms are individuated by means of formal descriptions, some of which are called programs. Programs and other formal descriptions of computing mechanisms specify which inputs may enter the mechanism, how the inputs affect the internal states of the mechanism, and which outputs come out of the mechanism under which conditions. Inputs and outputs are strings of letters from a finite alphabet, often called symbols.

In computability theory, symbols are typically marks on paper individuated by their geometrical shape (as opposed to their semantic properties). Symbols and strings of symbols may or may not be assigned an interpretation; if they are interpreted, the same string may be interpreted differently, e.g., as representing a number, or a program, etc., depending on what the theorist is trying to prove at any given time. In these computational descriptions, the identity of the computing mechanism does not hinge on how the strings are interpreted. Hence, within computability theory, symbols do not have their content essentially.

For example, the best-known computational formalism is that of turing machines (TMs). Standard TMs have two principal components: a potentially infinite tape, whose function is to hold symbols, and an active device, whose function is to move along the tape and write and erase symbols on it. Particular TMs are individuated by a finite list of instructions of the following form: if on the tape there is a certain letter and the active device is in a certain internal state, then the active device prints a certain letter, moves one step to the left (or right), and goes into a certain state. Each particular TM is uniquely individuated by its specific list of instructions, which implicitly

defines the relevant alphabet. Nothing in these descriptions involves any semantic properties; they simply describe how the active component of the TM reacts to the presence of certain letters on the tape while in certain internal states.

In computability theory, internal states of TMs are monadic and are never assigned interpretations. Inputs and outputs of TMs, which are strings of symbols, are typically interpreted, but they need not be. Sometimes it is useful to describe TMs without assigning any interpretation to their inputs and outputs. A good example is a TM discovered by J. Buntrock and H. Marxen in 1989 (cited by Wells, 1998). It uses two symbols and has only five possible internal states. This machine is offered as a demonstration of how difficult it is to predict the behavior of a TM from its abstract description, and how a very simple TM can have very complex behavior. When started on a blank tape, this simple TM halts after executing 23,554,764 steps. As Wells describes it, nothing in Buntrock and Marxen's TM has any content under anybody's notion of content, yet a computability theorist has no difficulty in recognizing it as a specific TM, which is uniquely individuated by its instructions.

The identity of specific TMs is determined by their instructions, not by the interpretations that may or may not be assigned to their inputs and outputs. More generally, the whole mathematical theory of computation can be formulated without assigning any interpretation to the strings of symbols being computed (e.g., Machtey and Young 1978). Even more generally, much research in fields that rely on computational formalisms, such as algorithmic information theory and the study of formal languages, proceeds without assigning any interpretations to the computational inputs, outputs, and internal states that are being studied.

The above considerations apply straightforwardly to ordinary, non-universal TMs, and any other computing mechanism whose behavior is not *controlled* by a program. This class includes standard connectionist computing mechanisms. Given their architecture, these are mechanisms that always perform

the same computation on their inputs. Which computation they perform is determined by their architecture and their characteristic list of instructions (or weight distribution, in the case of most connectionist computing mechanisms), but those instructions (or weight distributions) are not executed by the machine – they are hardwired, as it were.

What about universal TMs and their concrete counterparts, program-controlled computers? They operate by executing instructions, and which computation they perform depends on which instructions they execute. This special capacity may appear to require individuation by the semantic properties of the instructions. Thus, a general treatment of computational individuation cannot ignore computing mechanisms that execute instructions. In dealing with them, I will shift from the abstract to the concrete realm, thereby also showing that both realms can be handled within the same functional account.

In the practice of computer programming, programs are created by combining instructions that are *prima facie* contentful. For example, a high-level programming language may include a control structure of the form UNTIL P TRUE DO ___ ENDUNTIL.[9] In executing this control structure, the computer does ___ until the variable P has value TRUE and then moves on to the next instruction. The programmer is free to insert any legal sequence of instructions in the ___, knowing that the computer will execute those instructions until the value of P is TRUE. This awesome ability of computers to execute instructions is one of the motivations behind the semantic view of computational individuation.[10] For when people execute instructions, i.e., they do what the instructions *say* to do, they do so because they *understand* what the instructions say. By analogy, it is tempting to conclude that in some sense, computers respond to the semantic properties of the instructions they execute, or at least instructions and the corresponding computational states of the mechanism are individuated by their content. This temptation is innocuous to the extent that one understands how computers execute instructions and specifies the relevant notion of

computational content accordingly; otherwise, to speak of computers responding to semantic properties or of instructions being individuated by their content is misleading.[11] So, I will briefly explain how computers execute instructions.

In ordinary stored-program computers, instructions are encoded as binary strings (strings of bits). Each bit is physically realized by a voltage level in a memory cell or some other state capable of physically affecting the computer in the relevant way. Before the processor of a computer can execute a binary string written in a high-level programming language, the computer must transform the string into a machine language instruction, which the machine can execute. A machine language instruction is a binary string that, when placed into the appropriate register of a computer processor, *causes* the computer's control unit to generate a series of events in the computer's datapath. For example, the sequence of events may include the transfer of binary strings from one register to another, the generation of new strings from old ones, and the placement of the new strings in certain registers.

The computer is designed so that the operations performed by the computer's processor (i.e., the control unit plus the datapath) in response to a machine language instruction correspond to what the instruction means *in assembly language*. For instance, if the intended interpretation of an assembly language instruction is to copy the content of register x into register y, then the computer is designed so that when receiving a machine language encoding of that assembly language instruction, it will transfer the content of register x into register y. This feature of computers may be used to assign their instructions (and some of their parts) an interpretation, to the effect that an instruction asserts what its execution accomplishes *within* the computer. This may be called the *internal* semantics of the computer.

Internal semantics is not quite semantics in the sense usually employed by philosophers. When philosophers say 'semantics,' they mean *external* semantics, that is, semantics that relates a state to things *other* than its computational

effects within a computer, including objects and properties in the external world. Notice, however, that contents assigned to a state by an external semantics need not be concrete objects and properties in the environment; they may be numbers, counterfactual events, phonemes, non-existent entities like Aphrodite, etc.[12]

Internal semantics is no help to the supporters of the semantic view of computational individuation, for they are concerned with individuation by external semantic properties. This is because the semantic view is largely motivated by computational explanations of mental states and processes, which are widely assumed to be individuated by their (external) contents.

Internal semantics is fully determined by the functional properties of program-controlled computers, independently of any external semantics. This can be seen clearly by reflecting on the semantics of high-level programming language instructions. For instance, the semantics assigned above to UNTIL P TRUE DO ___ ENDUNTIL was ambiguous between an internal and an external reading. As I said, the instruction means to do ___ until the variable P has value TRUE. Doing ___ is a computational operation, so this component of the interpretation is internal. P is a variable of the programming language, which ranges over strings of symbols to be found inside the computer – again, this is an internal content. Finally, 'TRUE' may be taken to mean either *true* (the truth-value), or the word 'TRUE' itself as written in the relevant programming language (a case of self-reference).

When writing programs, it is convenient to think of 'TRUE' as referring to a truth-value. But for the purpose of individuating the computation, the correct interpretation is the self-referential one. For what a computer actually does when executing the instruction is to compare the (implementations of) the two strings, the one that is the value of P and the one that reads 'TRUE'. All that matters for the individuation of the computation is which letters compose the two strings and how they are concatenated together. If they are

the same letters in the same order, the processor proceeds to the next instruction; otherwise, it goes back to doing ___. Whether either string (externally) means a truth-value, or something else, or nothing, is irrelevant to determining which state the computer is in and which operation it's performing for the purpose of explaining its behavior. In other words, having an internal semantics does not entail having an external semantics.

This is not to say that instructions and data, either at a high level or at the machine language level, *lack* an external semantics. Each element of the machine implementation of a high-level instruction has a job to do, and that job is determined at least in part by the high-level instruction that it implements. Besides its internal semantics, that high-level instruction may well have a semantics that is, at least in part, external. By the same token, each element of a machine language datum is a component of the machine implementation of a high-level datum, and that high-level datum typically has an external semantics. It may be difficult or impossible to univocally break down the external contents of high-level instructions and data into external contents of machine language instructions and data, but this is only an epistemic limitation. As a matter of fact, machine language instructions and data may well have external semantic properties. This is perfectly compatible with the point at issue. The point is that the states of computing mechanisms, including program-controlled computers, do not have their external contents essentially – they are fully individuated without appealing to their external semantic properties.

Assigning instructions and data a semantics, either external or internal, is indispensable to designing, programming, using, and repairing computers, because that is the only way for designers, programmers, users, and technicians to understand what computers are doing or failing to do. But in an explanation of computer instruction execution, a complex instruction like UNTIL P TRUE DO ___ ENDUNTIL is a string of letters, which will be encoded in the computer as a

binary string, which will affect the computer's processor in a certain way. A computer is a powerful, flexible, and fascinating mechanism, and we may feel compelled to say that it responds to the semantic properties of the instructions it executes. But as I briefly argued, this kind of 'computer understanding' is exhaustively and mechanistically explained without ascribing any external semantics to the inputs, internal states, or outputs of the computer. The case is analogous to non-universal TMs, whose computational behavior is entirely determined and uniquely individuated by the instructions that are 'hardwired' in their active component.

In summary, the functional view of computational *individuation* holds that a program-controlled computer is a physical system with special functional properties that are specified by a certain kind of mechanistic explanation. Although for practical purposes the internal states of computers are usually ascribed content by an external semantics, this need not be the case and is unnecessary to individuate their computational states and explain their behavior.

The functional view is consistent with the non-semantic view of computational *causation* but goes beyond it. It holds that the identity conditions of computing mechanisms, their states, and the functions they compute are completely determined by their (non-semantic) functional properties. Even in the special case of program-controlled computers, where the functional individuation of computational states gives rise to an internal semantics, external semantics is not part of the individuation of computational states. From the functional view of computational individuation, it follows that computational descriptions are not *ipso facto* (external) semantic descriptions. So, if the functional view is correct, the semantic view of computational individuation is incorrect. From now on, unless otherwise noted, by 'semantics' I will mean external semantics, and by 'content' I will mean content ascribed by an external semantics.

The functional view of computational individuation bears some similarity to a view proposed by Egan (1992, 1995,

1999, 2003). Egan appears to reject the semantic view of computational individuation, because she rejects the view, championed by many philosophers, that the computational states postulated by psychological theories are individuated by the cognitive contents of those states (e.g., visual contents for the states of visual mechanisms, auditory contents for the states of auditory mechanisms, etc.). Instead, Egan argues that computational states are individuated individualistically, i.e., by properties that are shared by all physical duplicates of a mechanism. But when Egan specifies how computational states are individuated, she points to their 'mathematical contents', namely the 'mathematical' functions whose domain and range elements are denoted by the inputs and outputs of the computations (Egan, 1995, p. 187; 2003, p. 96). Although I agree with much of what Egan says, Egan's view does not capture the way computational states are individuated within computability theory and computer science; hence, it should be replaced by the functional view of computational individuation. Egan's view also faces an internal difficulty, which may be resolved by resorting to a version of the functional view.

Egan's mathematical contents behave differently from cognitive contents in some types of counterfactual reasoning. A salient difference is that mathematical contents – unlike cognitive ones – are not dependent on the relations between a mechanism and its environment. Under most views of mental content, whether an organism is thinking about water depends, inter alia, on whether there is H_2O in her environment (Putnam, 1975). But whether the same organism is thinking about the number seven does not seem to depend on anything in her environment. In this sense, mathematical contents are shared by physical duplicates in a way that cognitive contents (under most views of cognitive content) are not.

But there is a sense in which mathematical contents are no more intrinsic to computing mechanisms than cognitive contents. Mathematical contents are still contents – they are still relational properties of states, which depend on the relations between a mechanism and something else (numbers, sets, or

whathaveyou). From a formal semantics perspective, there is no principled difference between mathematical and cognitive contents. Both can be assigned as interpretations to the states of a mechanism, and both can be assigned to the mechanism's physical duplicates. It is certainly possible to assign the same mathematical interpretation to all physical duplicates of a computing mechanism, but in the same way, it is equally possible to assign the same cognitive interpretation to all physical duplicates of a computing mechanism.[13] Moreover, just as internal states of the same mechanism may be given different cognitive interpretations, it is well known that the same set of symbolic strings may be given different mathematical interpretations. In this sense, mathematical contents are shared by physical duplicates neither more nor less than cognitive contents. If the latter are not individualistic enough for Egan's purposes, the former shouldn't be either.

If someone wants to individuate computational states in a rigorously individualistic way, she should drop the individuation of computational states by their semantic properties – cognitive or mathematical – altogether. She might opt for an individualistic version of the functional view of computational individuation: under a narrow construal of mechanistic explanation, the functional properties of computing mechanisms are individualistic in precisely the sense desired by Egan.[14]

I will not defend an individualistic version of the functional view of computational individuation, however, because I am skeptical of the narrow construal of mechanistic explanation. I find a wide (non-individualistic) construal of mechanistic explanation more plausible. For present purposes, it is important to distinguish between wide individuation and individuation based on wide content. Individuation based on wide content is one type of wide individuation, but wide individuation is a broader notion. Wide individuation appeals to the relations between a mechanism and its context, relations which may or may not be semantic. For my purposes, of course, what is needed is wide individuation that does not appeal to semantic relations.

Mechanisms have many intrinsic properties, only some of which are functionally relevant. In order to know which intrinsic properties of mechanisms are functionally relevant, it may be necessary to consider the interaction between mechanisms and their contexts.[15] For instance, plants absorb and emit many types of electromagnetic radiations, most of which have little or no functional significance. But when radiation within certain frequencies hits certain specialized molecules, it helps produce photosynthesis – an event of great functional significance. Without knowing which external events cause certain internal events and which external effects those internal events have, it may be difficult or impossible to distinguish the functionally relevant properties of a mechanism from the irrelevant ones. As a consequence, scientific theories typically individuate the functional properties of mechanisms widely.[16]

The same point applies to the functional properties of computing mechanisms. As Harman (1988) points out, many philosophers have assumed that computing mechanisms are individuated purely individualistically (Putnam, 1967; Fodor, 1980; Stich, 1983). But this assumption is false. Concrete computing mechanisms, like all other mechanisms, have many intrinsic properties, only some of which are relevant to the results of their computations. For instance, most ordinary computers would not work for very long without a fan, but the fan is not a computing component of the computer, and blowing air is not part of the computer's computations.

As with any other mechanism, we need to distinguish the properties of a computing mechanism that are functionally relevant from the ones that are irrelevant. We also need to distinguish the functional properties that are relevant to computation from the irrelevant ones. In order to draw these distinctions, we need to know which of a computing mechanism's properties are relevant to its computational inputs and outputs and how they are relevant. In order to know that, we need to know what the computational inputs and outputs of the mechanism are. That, in turn, requires knowing how the mechanism's inputs and outputs interact with their context. In

the next section, I will adapt an argument by Shagrir to support this conclusion. For now, I conclude that the functional view of computing mechanisms may be better grounded on a wide construal of functional properties.[17]

At this juncture, someone might worry that at least in the case of computing mechanisms, wide functional individuation and individuation by wide content are equivalent. For instance, a wide function of an internal state might be to co-vary with an external variable. Under some theories of content, this is the same as representing that variable. If so, it may seem that wide functional individuation is the same as individuation by wide content, and that the functional account of computational individuation collapses into the semantic account. In response to this worry, I have two points to make.

First, the functional properties that are relevant to computational individuation, even when they are wide, are not *very* wide. They have to do with the normal interaction between a computing mechanism and its immediate mechanistic context via its input and output transducers. In the case of artificial computing mechanisms, the relevant context is, on the one hand, the relation between the forces exerted on input devices (such as keyboards) and the signals relayed by input devices to the computing components, and on the other hand, the relation between the computing components' outputs and the signals released by the output devices. Those relations, together with the internal relations between components and their activities, determine whether a computation is performed by a mechanism and which computation it is.

By the same token, in the case of organisms, the wideness of putative computational properties of nervous systems does not even reach into the organisms' environment; it only reaches sensory receptors and muscle fibers, for that is enough to determine whether a nervous system performs computations and which computations it performs. As a matter of fact, the main piece of empirical evidence that was originally employed by McCulloch and Pitts (1943) to justify the first computational theory of mind was the all-or-none properties

of neural signals, and those properties were originally discovered and identified to be functionally significant by studying the interaction between neural signals and muscle fibers.[18]

Second, the extent to which wide functional properties are the same as wide contents depends on which theory of content one adopts. In most of the literature on wide contents, wide contents are largely ascribed by intuition, and theories of content are tested by determining whether they agree with the relevant intuitions. By contrast, under the functional view of computational individuation, the functional properties that are relevant to the computational individuation of a mechanism are to be found by elaborating mechanistic explanations under the empirical constraints that are in place within the natural sciences. This establishes the computational identity of a mechanism without appealing to any semantic intuitions. Perhaps, under some theories of content, some wide semantic properties will turn out to supervene on some computational (or more generally, functional) properties. But this is not a weakness of the functional view – it's a strength. For under the functional view, computational properties can be discovered and individuated without appealing to semantic properties, thereby providing kosher naturalistic resources that may be used in a theory of content.

The same point may be put in the following way. One problem with naturalistic theories of content that appeal to computational properties of mechanisms is that, when conjoined with the semantic view of computational individuation, they become circular. For such theories explain content (at least in part) in terms of computation, and according to the semantic view, computational states are individuated (at least in part) by contents. The functional view breaks this circle: computations are individuated by (somewhat wide) functions; contents may then be explained (at least in part) in terms of computations, without generating any circularity.[19]

The present argument in favor of the functional view of computational individuation may be unpersuasive to someone firmly committed to the semantic view of computational

individuation. She might prefer to use the semantic view of computational individuation as a premise and conclude that the functional view of computational individuation must be incorrect. This would fly in the face of how computability theorists and computer scientists individuate computing mechanisms and their states. But the fact that philosophers have maintained the semantic view of computational individuation for decades in spite of computability theory and computer design shows that she wouldn't be deterred. To address this possible reply, I will discuss arguments for the semantic view of computational individuation.

3. AGAINST THE SEMANTIC VIEW OF COMPUTATIONAL INDIVIDUATION

There are three main arguments on offer for the semantic view of computational individuation. The first pertains directly to computing mechanisms and their states, and it goes as follows:

3.1. *Argument from the Identity of Computed Functions*

(1) Computing mechanisms and their states are individuated by the functions they compute.
(2) Functions are individuated semantically, by the ordered couples ⟨domain element, range element⟩ *denoted* by the inputs and outputs of the computation.
(3) Therefore, computing mechanisms and their states are individuated semantically.

Variants of the argument from the identity of computed functions may be found in the writing of several authors (Dietrich, 1989; Smith, 1996; Shagrir, 1997, 1999; Peacocke, 1999).[20]

The argument from the identity of functions ignores that when talking about computation, functions may be individuated in two ways. One appeals to the set of the ordered couples ⟨domain element, range element⟩ denoted by the inputs

and outputs of the computation (e.g., {⟨1, 10⟩, ⟨10, 11⟩, …}, where '1', '10', '11', … denote the numbers 1, 2, 3, …). The other individuates functions as the set of ordered couples ⟨input type, output type⟩, where input and output types are realized by the strings of digits that enter and exit the computing mechanism (e.g., {⟨1, 10⟩, ⟨10, 11⟩, …}, where '1', '10', '11', … denote inscriptions of types '1', '10', '11', …). In other words, functions can be defined either over entities such as numbers, which may be the content of computational inputs and outputs, or over entities such as strings of (suitably typed) letters from an alphabet, which are the inputs and outputs themselves. Both ways of individuating functions are important and useful for many purposes. Both can be used to describe what is computed by a computing mechanism. The relevant question is which of these ways of individuating functions is relevant to individuating computational states within a scientific theory of mechanism.

In light of the previous section, the function individuation that is relevant to computational explanation is the one based on strings. The other, semantic way of individuating functions may be useful for many other purposes, including explaining why people build computers the way they do and why they use them, but it is irrelevant to explaining the capacities of the mechanisms.

Given a functional description of a computing mechanism that individuates the function being computed in terms of input and output strings, one may ask how it is that that mechanism also computes the function ⟨domain element, range element⟩, defined over numbers or other entities. In order to explain this, what is needed is a further fact: that the inputs and outputs of the computation *denote* the elements of the domain and range of the function. This is a semantic fact, which relates functionally individuated inputs and outputs to their content. Stating this semantic fact requires that we individuate the inputs and outputs of the computation independently of their denotations. So, a non-semantic individuation

of computational states is a prerequisite for talking about their content.

Another problem with the argument from the identity of computed functions is that using the semantic values of the inputs and outputs does not individuate computing mechanisms and their states as finely as we need when talking about computing mechanisms, and it is hard to see what other semantic properties should be added to the semantic values in order to reach an adequate fineness of grain. Any domain of objects (e.g., numbers) may be represented in indefinitely many ways (i.e., notations). Any computable function may be computed by indefinitely many algorithms. Any algorithm may be implemented by indefinitely many programs written in indefinitely many programming languages. Finally, any program may be executed by indefinitely many computer architectures. Even within the same programming language or computer architecture, typically there are different ways of implementing the same algorithm. So the semantically individuated function itself, or even the function in combination with the algorithm,[21] does not individuate the computing mechanism and its states as finely as we need. This way of individuating computational states has the paradoxical consequence that mechanisms that have different architectures, use different programming languages, and execute different programs that implement different algorithms (perhaps of different computational complexity) and manipulate different notations, are ascribed the same computational states only because they compute the same semantically individuated function. To avoid this, individuating functions in terms of input and output strings is not enough. We should also allow other functional (non-semantic) aspects of the computation, such as the program and the architecture, to be part of the mechanistic explanation that individuates the computing mechanism and its computational states.

The second argument for the semantic view of computational individuation appeals to computational explanations of mental processes.

3.2. *Argument from the Identity of Mental States*

(1) Computational states and processes are posited in explanations of mental states and processes (e.g., inference).
(2) Mental states and processes are individuated by their semantic properties.
(3) Therefore, computational states and processes are individuated by the semantic properties of the mental states and processes they explain.

Variants of the argument from the identity of mental states may be found in many places (the most explicit include Fodor, 1975; Pylyshyn, 1984; Burge, 1986; Peacocke, 1994a, 1999; Wilson, 2004).[22]

Premise 1 is uncontroversial; it simply takes notice that some scientific theories explain mental states and processes computationally. Premise 2 has been challenged (e.g., by Stich, 1983), but for the sake of the argument I will ignore any concerns about whether content may be legitimately used to individuate mental states for scientific purposes.

As appealing as the argument from the identity of mental states may sound, it is a *non sequitur*. As Egan (1995) notes, the only way the conclusion can be derived from the premises is by assuming that *explanantia* must be individuated by the same properties that individuate their *explananda*. This assumption is at odds with our explanatory practices. The relevant type of explanation is constitutive explanation, whereby a property or capacity of a mechanism is explained in terms of the functions and organization of its constituents. For example, consider the explanation of digestion. The *explanandum*, a certain type of state change of some organic substances, is individuated by the chemical properties of substances before, during, and after they enter the stomach.

Its *explanans*, which involves secretions from certain glands in combination with the stomach's movements, is individuated by the activities of the stomach, its glands, and their secretions. This example shows that the individuation of *explanantia* independently of their *explananda* is an aspect of our explanatory practices. There is no reason to believe that this should fail to obtain in the case of explanations of mental states and processes. And without the assumption that *explanantia* must be individuated by the same properties that individuate their *explananda*, the argument from the identity of mental states doesn't go through.[23]

In recent years, a subtle new argument for a weakened version of the semantic view of computational individuation has been formulated by Shagrir (2001). Here is a formulation, using my terminology, of the thrust of Shagrir's argument:

3.3. *Argument from the Multiplicity of Computations:*

(1) The same computing mechanism M implements multiple (non-semantically individuated) computations $C_1, \ldots, C_n$ at the same time.
(2) For any task that M may perform, there is a unique $C_i \in \{C_1, \ldots, C_n\}$, such that C_i alone explains M's performance of the task, and C_i is determined by the task performed by M in any given context.
(3) Tasks are individuated semantically.
(4) Therefore, in any given context, C_i is individuated semantically (in part).
(5) Therefore, in so far as computations explain the performance of a task by a mechanism in any given context, they are individuated semantically (in part).

Premise (1) appeals to the fact that the inputs, outputs, and internal states of a mechanism can be grouped together in different ways, so that different computational descriptions apply to them. For instance, imagine a simple device that takes two input digits and yields one output digit and whose

inputs and outputs may take three possible values (which may be called 0, ½, and 1). And suppose that the outputs are related to the inputs as follows:

Inputs	→ Output
0, 0	→ 0
0, ½	→ ½
½, 0	→ ½
0, 1	→ ½
1, 0	→ ½
½, ½	→ ½
½, 1	→ ½
1, ½	→ ½
1, 1	→ 1

The above is a bona fide computational description of our device. Under this description, the device performs an averaging task of sorts. Since this averaging task exploits all of the functionally significant inputs and outputs of the device, I will refer to it as the *maximal* task of the device, and to the corresponding computation as the maximal computation.

If we group together and re-label our inputs and outputs, we may find other computational descriptions. For instance, we may group '0' and '½' together and call both of them 0, or we may group '½' and '1' together and call both of them 1. In the first case, our device turns into what is ordinarily called an AND gate, whereas in the second case, it turns into an OR gate. As a consequence of this grouping and re-labeling, our device implements several computations at once: our original averaging, the AND operation, the OR operation, etc. These operations form our set of computations C_1, …, C_n mentioned in premise (1), all of which are implemented by our device at the same time.[24]

In principle, our device could be used to perform different tasks, each of which corresponds to one of the computations implemented by the device. It could be used to perform its maximal task (averaging) as well as a number of non-maximal

tasks (conjunction, disjunction, etc). But in any given context, our device might be used to perform only one specific task. For example, our device might be part of a larger device, which uses it to perform conjunctions. Premise (2) points out that in order to explain how our device performs a given task, say conjunction, we must appeal to the relevant computational description, namely AND. So, the task performed by a computing mechanism in a given context determines which computational description is explanatory in that context.

Although premises (1) and (2) are true and suggestive, they probably make little difference in most scientific contexts. For in practice, computing mechanisms like our simple device above are usually employed to perform their maximal task. In engineering applications, it would be unnecessarily costly and cumbersome to build a device with inputs and outputs of three kinds but use it to perform tasks that require inputs and outputs of only two kinds. In nature, it is unlikely that natural selection would generate a process that can differentiate between more possible inputs and outputs than it needs to in order to carry out its task. Although it is common for the same naturally occurring mechanism to perform different tasks, usually each task is subserved by a different process within the mechanism. And although some natural computational processes may have evolved from ancestors that required to differentiate more inputs and outputs than the current process, this seems unlikely to be the most common occurrence. So the possibilities mentioned in premises (1) and (2) may not have great practical significance. Nevertheless, it is philosophically useful to know what they entail about the individuation of computation, so let us examine the rest of the argument.

Premise (3) says that tasks are semantically individuated. For instance, one of our device's tasks, averaging, is defined over quantities, which are the implicit referents of the inputs and outputs of the device. Since, by (2), tasks determine which computational description is explanatory in a given context, (4) concludes that the computational identity of a

device in a given context is partially determined by semantic properties. In other words, the computation that is explanatory in any given context is partially individuated semantically. Given that the argument does not depend on the specific device or computational description, (5) is a universal generalization of (4).

Before discussing the merits of the argument from the identity of computational tasks, notice that its conclusion is weaker than the traditional semantic view of computational individuation. For the argument begins by conceding that the (multiple) computations implemented by a device are individuated non-semantically. Semantic constraints only play a role in determining which of those computations is explanatory in a given context. As I pointed out above, it is likely that in most contexts of scientific interest, computing mechanisms perform their maximal task, so that semantic constraints are unnecessary to determine which computation is explanatory. If this is correct, and if the argument from the multiplicity of computations is sound, then semantic constraints will play a role in few, if any, practically significant contexts. It remains to be seen whether the argument is sound.

The problem is with premise (3), and it is analogous to the problem with premise (2) in the argument from the identity of functions. The task of a computing mechanism is to compute a certain function. As I pointed out above, functions may be individuated semantically, and hence so may tasks. But as I also pointed out above, functions may be individuated non-semantically, and hence so may tasks. For the same reasons given in the case of functions, the task description that is relevant to individuating computing mechanisms and their processes is non-semantic.

Shagrir's reason for (3) seems to be that he works under a narrow construal of functional properties. If functional properties are construed narrowly, then they are insufficient to determine which task a mechanism is performing within a context, and hence which computation is explanatory in that context. It goes to Shagrir's credit that he showed this to us.

But the solution need not be an individuation of computations based on content, for there is also the possibility – which I advocated in section 2 – of a wide construal of functional properties. Shagrir gives no reason to prefer a semantic individuation of computations to a wide functional individuation. Provided that the interaction between a mechanism and its context plays a role in individuating its functional (including computational) properties, a (non-semantic) functional individuation of computational states is sufficient to determine which task is being performed by a mechanism, and hence which computation is explanatory in a context.

In order to know which of the computations that are implemented by a computing mechanism is explanatory in a context, we need to know the relevant relations between computations and contexts. Therefore, we cannot determine which computation is explanatory within a context without looking outside the mechanism. I agree with Shagrir about this, and also about the fact that interpreting computations – describing computations semantically – is one way to relate computations to context. But it's not the only way: computations have effects on, and are affected by, their context. By looking at which effects of which computations are functionally significant within a context, we can identify the computation that is explanatory within that context. Going back to our example, suppose our device is a component of a larger mechanism. By looking at whether the containing mechanism responds differentially to a '0', '½', and '1' or responds identically to two of them, we can determine which computational description is explanatory without needing to invoke any semantic properties of the computations.

4. CONCLUSION

Existing arguments for the semantic view of computational individuation fail. There is no reason to believe that computational states are individuated by their semantic properties. Instead, computational states are individuated by their

functional properties, as specified by the mechanistic explanation of the mechanism that bears those states.

The point is not that content has no role to play in formulating and evaluating computational theories. It has many important roles to play, at least under the most common methodologies and assumptions. The point, rather, is that computing mechanisms and their states have functional identity conditions, and that the functional properties of computing mechanisms are all that is needed to individuate computing mechanisms and their states. Once computational states are functionally individuated, interpretations may (or may not) be assigned to them.

In both computer science and cognitive science, the most perspicuous way of individuating tasks is often semantic. We speak of computers doing arithmetic and of visual systems inferring properties of the world from retinal images – these are semantic characterizations of their tasks. But to those semantic characterizations, there correspond an indefinite number of possible non-semantic characterizations, which individuate different computational architectures, running different programs, written in different programming languages, executing different algorithms. Before a semantic characterization of a task can be mapped onto a particular mechanism, the semantic characterization needs to be replaced by a functional, non-semantic task description. Only the latter is what determines the identity conditions of the mechanism *qua* computational.

A first corollary is that being a computational state does not entail having semantic properties. This applies to artifacts and natural systems alike. A computer can be truly described computationally without ascribing content to it, and so can a mind. This corollary is important in light of the tendency among many theorists to construe the computational states postulated by computational theories of mind (CTMs) as representational. This is a mistake, which begs the question of whether the computational states postulated by a theory of mind have content.[25] Perhaps they do, but perhaps – as Stich (1983) pointed out some time ago – they don't. Whether

mental states have content should not be determined by the metaphysics of computational states; it should be an independent substantive question. A good account of computation should not entail – as the semantic view of computational individuation does – that one cannot be a computationalist about mental states while also being an eliminativist about their content.

If mental states have content, there is a separate question of whether the contents of states posited by computational theories match the contents ascribed by folk psychology. Perhaps some or all internal computational states have contents that match the folk psychological contents, as many computationalists believe (e.g., Fodor, 1987; Pylyshyn, 1984). Or perhaps they don't, as other computationalists maintain (e.g., Dennett, 1987, esp. chap. 5). These are substantive questions that depend on the relationship between computational explanations of mental states and capacities and theories of mental content, and are at least in part empirical; they should not be settled by philosophizing on the metaphysics of computation. In light of these considerations, the functional view of computational individuation has the appealing feature that it leaves the questions of whether mental states have content and what content they have independent of the question of whether mental states are computational.

A second corollary relies on the premise that the possession of semantic properties does not entail the possession of computational properties. Since I'm not aware of any claim to the contrary, I will not argue for this premise. The second corollary is that being computational is logically independent of having content, in the sense that it is possible to be a computational state without having content and *vice versa*. CTM and the representational theory of mind (RTM) address independent (orthogonal) problems. CTM should be formulated and discussed without any theory of content, indeed without even presupposing that minds have content, so as to avoid getting entangled with the difficult issue of mental content.

And RTM should be formulated without presupposing that mental states are computational.

My conclusions have no consequences on whether minds or computers have content, whether mental and computational content are the same, and whether mental content is reducible to computational content. All I'm saying is that those questions must be answered by a theory of content, not by a theory of computation or a CTM.

NOTES

[1] A version of this paper was presented at the 2004 Eastern APA in Boston, MA. Thanks to my commentator, Larry Shapiro, and the audience, especially Robert Cummins, for their helpful feedback. A previous ancestor was presented at the 2002 Northwest Philosophy Conference, in Portland, OR. Thanks to that audience and commentator, Anastasia Panagopoulos. I also thank those who commented on previous versions of this paper, especially Robert Cummins, Frances Egan, Manuel Gatto, Peter Machamer, Susan Schneider, Michael Rescorla, Oron Shagrir, and the anonymous referees. Finally, thanks to Jerry Fodor for correspondence on this topic.

I developed the view defended here mostly by thinking about explanatory practices in computer science in light of Carl Craver's work on mechanistic explanation. I was also influenced by previous philosophical work on computation, especially Jerry Fodor's. Although the paper focuses most explicitly on the differences with other views, there are also varying degrees of continuity between the present view and earlier works on computation, including Chalmers (1996), Cummins (1983), Egan (1995), Fodor (1980), Glymour (1991), Horst (1999), Newell (1980), Pylyshyn (1984), Shagrir (2001), and Stich (1983). Of course, I am solely responsible for any errors contained in this paper.

[2] Shagrir has pointed out to me that someone might maintain that computational states are necessarily representations while denying that computational states are individuated by their semantic properties, perhaps on the grounds of an analogy with anomalous monism. According to anomalous monism (Davidson, 1970), mental states are necessarily physical even though they are not individuated by their physical properties; by the same token, computational states might be necessarily representational even though they are not individuated by their semantic properties. Since I'm not aware of anyone who has taken this stance explicitly or any reason for taking it, I will not discuss it further.

[3] Here is the most explicit formulation that I know of:

Suppose we start with the notion of a syntactic description of representations. I don't think that this begs any questions because I don't think syntactic individuation requires semantic individuation. Roughly (at least for the case of natural languages) it requires (i) an inventory of basic objects (morphemes, as it might be) and (ii) a recursive definition of WFF (I think all the recursions are on constituents; but I doubt that matters in the present context). Finally, I assume that every computation is a causal sequence of tokenings of such states.

Given that, there are two questions: 1. What distinguishes those of such causal sequences that *constitute computations* from those that don't? Answer, the former preserve *semantic properties* of the strings (paradigmatically, they take one from true inputs to true outputs). This requires that the tokened states have semantic interpretations (since, of course, only what is semantically interpreted can be evaluated for truth). So, in that sense, *the representations in question are individuated by their semantic properties inter alia*. 2. What are the constraints on the causal processes defined on such states? Answer, the effects of being in state S must be fully determined by the syntactic properties of S (together, of course, with the rules of state transition). That's the sense in which computation is a syntactic process.

So computations are syntactic processes defined over semantically interpreted arrays of representations. (Fodor, personal correspondence, emphasis added.)

[4] Supporters of the semantic view include Fodor (1998, p. 11), Pylyshyn (1984, p. 30), and many others. They often disagree on whether the semantic properties that individuate computational states are wide or narrow. For a sample of this debate, cf. Bontly (1998), Burge (1986), Butler (1996), Egan (1999), Segal (1989, 1991), Shagrir (2001), Shapiro (1997). Egan's view is significantly different from the others; I will discuss it below.

[5] The best account of mechanistic explanation that I know of is due to Craver (forthcoming). See also Bechtel and Richardson (1993), Machamer et al. (2000), and Glennan (2002).

[6] The *locus classicus* is Fodor (1980): "computational processes are both *symbolic* and *formal*... What makes syntactic operations a species of formal operations is that being syntactic is a way of *not* being semantic" (Fodor, 1980, p. 64). See also Newell 1980. The non-semantic view of computational causation has been challenged, typically on the grounds of the semantic view of computational individuation (Dietrich, 1989; Peacocke, 1994a, 1999; Shagrir, 1999). Since I am arguing against the semantic view of computational individuation, I need not address this challenge.

[7] The reader who remains skeptical should consult the literature referred to in the preceding footnotes.

[8] For an extended argument to this effect, see Piccinini (forthcoming).

[9] I took this example from Loop Programs, a simple but powerful programming language invented by Robert Daley at the University of Pittsburgh.

[10] For evidence of this, see Fodor (1968) and Piccinini (2004a).

[11] For example, Dietrich (1989) argues that since computing mechanisms respond to semantic properties of computational states, the non-semantic view of computational causation should be rejected. The following considerations explain why Dietrich's conclusion is unwarranted.

[12] Dennett (1987) uses the expressions 'internal semantics' and 'external semantics' in a similar sense, and Fodor (1978) discusses some related issues. Curiously, I devised and named this distinction before reading Dennett's work. The distinction between internal and external semantics should not be confused with that between semantic internalism and semantic externalism, which pertain to the identity conditions of contents (specified by an external semantics).

[13] Of course, some of those interpretations may turn out to be intuitively anomalous within a cognitive theory of an organism, in the sense that they may fail to capture the way the organism relates to her actual environment (as opposed to a possible environment). In computer science, however, all that matters for interpreting computational states is the formal adequacy of a candidate interpretation, that is, whether the states can be systematically interpreted in one way or another. There is nothing intuitively anomalous about interpreting a computer on Twin Earth as computing something about H_2O, even if there is no H_2O on Twin Earth. In this respect, the semantics of artificial computing mechanisms is different from that of organisms. Perhaps this is because the semantics of computing mechanisms is derived, whereas that of organisms is original.

[14] This is individualism about *computing* mechanisms, not about *psychological* mechanisms. A narrow reading of the functional view of computational individuation is compatible with there being psychological computing mechanisms that include features of both individuals and their environment, as argued by Wilson (1994, 2004).

[15] The context of a mechanism need not coincide with the environment of an organism. If a mechanism is an internal component of a larger system, its context is constituted by other relevant components of the system and their activities.

[16] A similar view is defended by Kitcher (1985), Harman (1988), and Shapiro (1994). These authors do not refer to mechanistic explanation but to functional explanation. I hope to address the relationship between functional and mechanistic explanation elsewhere. Here, suffice it to say that in my view, functional explanation is a kind of mechanistic explanation, and the same considerations that favor wide functional explanation over narrow functional explanation apply, more generally, to wide mechanistic explanation over narrow mechanistic explanation.

[17] Again, this is compatible with Wilson's (1994, 2004) wide computationalism, according to which a psychological computing mechanism may spatially extend beyond the boundaries of an organism, but it is also compatible with the negation of Wilson's view. I have argued that functional (including computational) properties are partially individuated by their interactions between a mechanism and its context. I am officially neutral on whether the components of psychological computing mechanisms extend beyond the spatial boundaries of organisms.

[18] For more on the discovery of the all-or-none properties of neural signals, see Frank (1994). For a detailed study of the considerations about the physiology of neurons that are at the origin of the computational theory of mind, cf. Piccinini (2004b).

[19] For a detailed version of this argument, to the effect that the semantic view of computational individuation should be abandoned so as to avoid the circularity inherent in theories of content that appeal to computation, see Piccinini (2004a).

[20] Cf. Dietrich and Peacocke:

> a correct account of computation requires us to attribute content to computational processes in order to explain which functions are being computed (Dietrich, 1989, p. 119).

> There is no such thing as a purely formal determination of a mathematical function (Peacocke, 1999, p. 199).

[21] Using algorithms in combination with semantically individuated functions has been proposed in the literature as a way to individuate computational states (e.g., Pylyshyn, 1984). However, there is no accepted way to individuate algorithms themselves other than non-semantically (Markov, 1960), and it is doubtful that any satisfactory account of the identity conditions of algorithms in semantic terms is forthcoming (as argued by Dean, 2002).

[22] Cf. Burge and Peacocke:

> There is no other way to treat the visual system as solving the problem that the [computational] theory sees it as solving than by attributing intentional states (Burge, 1986, pp. 28–29).

> One of the tasks of a subpersonal computational psychology is to explain how individuals come to have beliefs, desires, perceptions and other personal-level content-involving properties. If the content of personal-level states is externally individuated, then the contents mentioned in a subpersonal psychology that is explanatory of those personal states must also be externally individuated. One cannot fully explain the presence of an externally individuated state by citing only states that are internally individuated. On an externalist conception of subpersonal psychology, a content-involving computation commonly consists in the explanation of some externally individuated states by other externally individuated states (Peacocke, 1994b, p. 224).

[23] For a similar reply to the argument from the identity of mental states, see Egan (1995, p. 57ff).

[24] Shagrir suggests another way in which a mechanism might implement multiple computations, namely, by letting different sets of properties (e.g., voltage and temperature) implement different computations (Shagrir, 2001, p. 375). But then either the different sets of properties correlate, in which case the two computations are the same, or they don't, in which case we simply have two processes performing two different computations within the same mechanism. (A mechanism may perform many activities at the same time thanks to different internal processes, which may or may not have some parts in common; in the case of this example, both activities are computations and both processes are computational.)

[25] This is true only under the assumption, almost universally shared among supporters of the semantic view of computational individuation, that computational states are individuated by the same contents that individuate the mental states realized, in whole or in part, by those computational states. If one rejects that assumption, then the semantic view of computational individuation is compatible with intentional eliminativism. But if one rejects that assumption, the semantic view of computational individuation ceases to have any significant positive motivation.

REFERENCES

Bechtel, W. and Richardson, R.C. (1993): *Discovering Complexity: Decomposition and Localization as Scientific Research Strategies*, Princeton, NJ: Princeton University Press.

Bontly, T. (1998): 'Individualism and the Nature of Syntactic States', *British Journal for the Philosophy of Science* 49, 557–574.

Burge, T. (1986): 'Individualism and Psychology', *Philosophical Review* 95, 3–45.

Butler, K. (1996): 'Content, Computation, and Individuation in Vision Theory', *Analysis* 56, 146–154.

Chalmers, D.J. (1996): 'Does a Rock Implement Every Finite-State Automaton?', *Synthese* 108, 310–333.

Copeland, B.J. (1996): 'What is Computation?', *Synthese* 108, 224–359.

Crane, T. (1990): 'The Language of Thought: No Syntax Without Semantics', *Mind and Language* 5(3), 187–212.

Craver, C. (forthcoming): *Explaining the Brain*, Oxford University Press.

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

Davidson, D. (1970): 'Mental Events. Experience and Theory in L. Foster and J.W. Swanson', Amherst, MA, University of Massachusetts Press. Reprinted in Davidson, Essays on Actions and Events. Oxford, Clarendon Press, 1980.

Dean, W. (2002): What Algorithms Could Not Be. In Proceedings of the Computing and Philosophy Conference, Pittsburgh, PA.

Dennett, D.C. (1987): *The Intentional Stance*, Cambridge, MA: MIT Press.

Dietrich, E. (1989): 'Semantics and the Computational Paradigm in Cognitive Psychology', *Synthese* 79, 119–141.

Egan, F. (1992): 'Individualism, Computation, and Perceptual Content', *Mind* 101(403), 443–459.

Egan, F. (1995): 'Computation and Content', *Philosophical Review* 104, 181–203.

Egan, F. (1999): 'In Defence of Narrow Mindedness', *Mind and Language* 14(2), 177–194.

Egan, F. (2003): 'Naturalistic Inquiry: Where Does Mental Representation Fit in?', in L.M. Antony and N. Hornstein (eds.), *Chowsky and His Critics*, Malden: Blackwell, pp. 89–104.

Fodor, J.A. (1968): 'The Appeal to Tacit Knowledge in Psychological Explanation', *Journal of Philosophy* 65, 627–640.

Fodor, J.A. (1975): *The Language of Thought*, Cambridge, MA: Harvard University Press.

Fodor J.A. (1978): 'Tom Swift and His Procedural Grandmother', *Cognition* 6, 229–247.

Fodor J.A. (1980): 'Methodological Solipsism Considered as a Research Strategy in Cognitive Psychology', *Behavioral and Brain Sciences* 3(1), 63–109.

Fodor J.A. (1981): 'The Mind–Body Problem', *Scientific American* 244 (January 1981). Reprinted in Heil, J. (ed.) (2004a). *Philosophy of Mind: A Guide and Anthology*, Oxford: Oxford University Press, pp. 168–182.

Fodor, J.A. (1987): *Psychosemantics*, Cambridge, MA: MIT Press.

Fodor, J.A. (1998): *Concepts*, Oxford: Clarendon Press.

Frank, R. (1994): 'Instruments, Nerve Action, and the All-or-None Principle', *Osiris* 9, 208–235.

Glennan, S. (2002): 'Rethinking Mechanistic Explanation', *Philosophy of Science* 69, 5342–5353.

Glymour, C. (1991): 'Freud's Androids', in J. Neu (ed.), *The Cambridge Companion to Freud*, Cambridge: Cambridge University Press.

Harman G. (1988): 'Wide Functionalism', in S. Schiffer and S. Steele (eds.), *Cognition and Representation*, Boulder: Westview, pp. 11–20.

Horst, S.W. (1996): *Symbols, Computation, and Intentionality: A Critique of the Computational Theory of Mind*, Berkeley, CA: University of California Press.

Horst, S. (1999): 'Symbols and Computation', *Minds and Machines* 9(3), 347–381.

Jacquette D. (1991): 'The Myth of Pure Syntax', in L. Albertazzi and R. Poli (eds.), *Topics in Philosophy and Artificial Intelligence*, Bozen: Istituto Mitteleuropeo di Cultura, pp. 1–14.

Kitcher, P. (1985): 'Narrow Taxonomy and Wide Functionalism', *Philosophy of Science* 52(1), 78–97.

Machamer, P.K., Darden, L. and Craver, C. (2000): 'Thinking About Mechanisms', *Philosophy of Science* 67, 1–25.

Machtey, M. and Young, P. (1978): *An Introduction to the General Theory of Algorithms*, New York: North Holland.

Markov, A.A. (1960 [1951]): 'The Theory of Algorithms', *American Mathematical Society Translations, Series 2* 15, 1–14.

Marr, D. (1982): *Vision*, New York: Freeman.

McCulloch, W.S. and Pitts, W.H. (1943): 'A Logical Calculus of the Ideas Immanent in Nervous Activity', *Bulletin of Mathematical Biophysics* 7, 115–133.

Newell, A. (1980): 'Physical Symbol Systems', *Cognitive Science* 4, 135–183.

Peacocke, C. (1994a): 'Content, Computation, and Externalism', *Mind and Language* 9, 303–335.

Peacocke C. (1994b): 'Content', in S. Guttenplan (ed.), *A Companion to the Philosophy of Mind* (pp. 219–225), Oxford: Blackwell.

Peacocke, C. (1999): 'Computation as Involving Content: A Response to Egan', *Mind and Language* 14(2), 195–202.

Piccinini, G. (2004a): 'Functionalism, Computationalism, and Mental Contents', *Canadian Journal of Philosophy* 34(3), 375–410.

Piccinini, G. (2004b): 'The First Computational Theory of Mind and Brain: A Close Look at McCulloch and Pitts's 'Logical Calculus of Ideas Immanent in Nervous Activity'', *Synthese* 141(2), 175–215.

Piccinini, G. (forthcoming): 'Computational Modeling vs. Computational Explanation: Is Everything a Turing Machine, and Does It Matter to the Philosophy of Mind?' *Australasian Journal of Philosophy*.

Putnam, H. (1967): *Psychological Predicates. Art, Philosophy, and Religion*, Pittsburgh, PA: University of Pittsburgh Press.

Putnam, H. (1975): 'The Meaning of ''Meaning'', in K. Gunderson (ed.), *Language, Mind and Knowledge*, Minneapolis: University of Minnesota Press. Reprinted in Putnam, H. (1975): *Mind, Language and Reality: Philosophical Papers*, Vol. 2. (pp. 215–271) Cambridge, UK: Cambridge University Press.

Pylyshyn, Z.W. (1984): *Computation and Cognition*, Cambridge, MA: MIT Press.

Scheutz, M. (1999): 'When Physical Systems Realize Functions', *Minds and Machines* 9, 161–196.

Searle, J.R. (1980): 'Minds, Brains, and Programs', *The Behavioral and Brain Sciences* 3, 417–457.

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

Segal, G. (1989): 'Seeing What is Not There', *Philosophical Review* 98, 189–214.

Segal, G. (1991): 'Defence of a Reasonable Individualism', *Mind* 100, 485–493.
Shagrir, O. (1997): 'Two Dogmas of Computationalism', *Minds and Machines* 7(3), 321–344.
Shagrir, O. (1999): 'What is Computer Science About?', *The Monist* 82(1), 131–149.
Shagrir, O. (2001): 'Content, Computation and Externalism', *Mind* 110(438), 369–400.
Shapiro, L.A. (1994): 'Behavior, ISO Functionalism, and Psychology', *Studies in the History and Philosophy of Science* 25(2), 191–209.
Shapiro, L. (1997): 'A Clearer Vision', *Philosophy of Science* 64, 131–153.
Smith, B.C. (1996): *On the Origin of Objects*, Cambridge, MA: MIT Press.
Stich, S. (1983): *From Folk Psychology to Cognitive Science*, Cambridge, MA: MIT Press.
Wells, A.J. (1998): 'Turing's Analysis of Computation and Theories of Cognitive Architecture', *Cognitive Science* 22(3), 269–294.
Wilson, R.A. (1994): 'Wide Computationalism', *Mind* 103, 351–372.
Wilson, R.A. (2004): *Boundaries of the Mind: The Individual in the Fragile Sciences*, Cambridge, UK: Cambridge University Press.

Department of Philosophy
University of Missouri – St. Louis
599 Lucas Hall (MC 73), 1 University Blvd.
St. Louis, MO 63121-4400
USA
E-mail: piccininig@umsl.edu