Are Computational Transitions Sensitive to Semantics?

Michael Rescorla

Abstract: The *formal conception of computation* (FCC) holds that computational processes are not sensitive to semantic properties. FCC is popular, but it faces well-known difficulties. Accordingly, authors such as Block and Peacocke pursue a 'semantically-laden' alternative, according to which computation can be sensitive to semantics. I argue that computation is insensitive to semantics within a wide range of computational systems, including any system with 'derived' rather than 'original' intentionality. FCC yields the correct verdict for these systems. I conclude that there is only one promising strategy for semantically-laden theorists: identify special computational systems that help generate their own semantic properties, and then show that computation within those systems is semantically-laden.

Keywords: computational theory of mind; syntax; semantics

1. The Formal Conception of Computation

A popular view holds that computational processes are formal, rule-governed manipulations of symbols. By 'formal,' I mean roughly that the processes are not sensitive to semantics. As Fodor puts it, computation has 'no access to the *semantic* properties of… representations, including the property of being true, of having referents, or, indeed, the property of being representations *of the environment*' [1981: 231]. Similarly, Egan writes that computational operations 'are sensitive only to *formal* (i.e. *non-semantic*) properties of the representations over which they are defined, not to their content' [2010: 254]. Other

advocates include Gallistel and King [2009: 72] and Pylyshyn [1984: 50]. Advocates can acknowledge that symbols manipulated during computation *have* semantic properties. But they deny that such properties inform the transitions between computational states.

In the 1980s, opposition mounted against the 'classical' picture of computation as rule-governed symbol manipulation. Many researchers now insist that *connectionist networks* can compute without manipulating symbols or syntactically structured entities. However, most connectionists also hold that a connectionist network's semantic properties, if it has any, do not inform how it evolves. Thus, connectionists usually endorse the doctrine that computation is not sensitive to meaning or semantic properties. I call that doctrine *the formal conception of computation* (FCC).

Researchers often combine FCC with *the computational theory of mind* (CTM), which holds that mental processes are computational. CTM+FCC entails that mental processes are not sensitive to semantics. Intuitively, then, it assigns intentionality a less significant role within cognitive activity than one might have expected. Opponents develop this worry in various ways. Peacocke [1994] complains that CTM+FCC renders semantic properties *explanatorily irrelevant* to scientific psychology. Block [1990] worries that it depicts semantic properties as *causally irrelevant* to mental activity, thereby engendering epiphenomenalism. Proponents of CTM+FCC pursue two main strategies for addressing such worries:

(a) *Show that, despite initial appearances, CTM+FCC can assign intentionality a non-trivial role in our theory of mind*. Fodor [1994: 2-54] adopts this strategy. He develops a detailed version of CTM+FCC that supposedly honours the causal and explanatory relevance of mental content.

(b) *Jettison content from scientific psychology*. Stich [1983: 149-183] adopts this strategy. He argues that cognitive science should explain mental activity through purely syntactic theories that assign no significant role to semantics.

Both (a) and (b) are controversial. In particular, many critics insist that Fodor's pursuit of (a) ends in failure [e.g. Aydede and Robbins 2001], while most philosophers reject Stich's purely syntactic approach as unacceptable.

In response to these difficulties, some authors propose that we replace FCC with a 'semantically-laden' conception, according to which semantic properties can inform how computation proceeds. For instance, Block claims that a computer's 'internal processors can be sensitive to *both* syntax and semantics' [1990: 151]. Other advocates of semantically-laden computation include Burge [2010: 95-98], Figdor [2009], O'Brien and Opie [2006], and Peacocke [1994]. If we reject FCC, we can say that mental processes are computational *and* sensitive to semantics. Advocates contend that CTM+¬FCC is better positioned than CTM+FCC to vindicate the causal and explanatory relevance of mental content.

But can we develop a convincing semantically-laden account? Semantically-laden theorists must do more than claim that certain computations are 'sensitive to semantics.' They must elaborate that claim into a compelling theory securely grounded in our best mathematical models of computation. Unfortunately, current attempts tend to be sketchy, unconvincing, or poorly integrated with existing mathematical models. I will support this assessment below.

Despite my criticisms, I find the semantically-laden account attractive. I think that CTM is a promising theory of how the mind works, but I find (a) and (b) unpromising. My goal is to promote better semantically-laden theories. This project is important whether or not you share my wariness towards (a) and (b). The issues here are so fundamental that every

option requires thorough investigation. My paper contributes to that end by isolating a necessary condition upon any satisfactory semantically-laden theory.

2. Computational Transitions versus Computational Individuation

The dispute over FCC is somewhat nebulous, because the term 'sensitive' is not very precise. One might elucidate 'sensitivity' by citing causal or explanatory relevance, appropriate participation in scientific laws, and so on. One finds various such glosses in the literature. I take no stand on the correct gloss. I want to explore whether there is *any* interesting sense in which computational transitions can be 'sensitive' to semantics. As Piccinini [2008] notes, we must distinguish this question from questions such as: What determines whether a physical process is computational? Which properties of a physical process are relevant to its computational nature? Which properties of a physical process help determine which computations it implements? These questions concern computational *individuation*, whereas my question concerns computational *dynamics*. I now illustrate the distinction through two examples.[1]

2.1 Is Semantics Necessary for Physical Computation?

Mathematical models of computation, such as the Turing machine, are abstract entities. They do not exist in space or time, and they do not participate in causal relations. Under suitable circumstances, a physical system *implements* or *physically realizes* an abstract computational model. Some philosophers hold that a physical system implements a computational model only if the system has semantic or representational properties [Dietrich

1989], [Ladyman 2009]. Call this *the semantic view of computational implementation*. In contrast, Chalmers [1995], Piccinini [2008], and others deny any essential tie between semantics and physical computation.

I agree with Chalmers and Piccinini. Consider the Turing machine. A 'machine table' summarizes rules governing how a 'scanner' manipulates syntactic items (paradigmatically, strings of strokes on a machine 'tape'). The machine table dictates subsequent machine behaviour based solely upon the tape's current configuration and the scanner's current internal state. It does not mention the semantic properties, if any, of syntactic entities inscribed on the tape. As Chalmers emphasizes [1995: 399], there is no principled bar to realizing an uninterpreted Turing machine in an uninterpreted physical system. (To avoid worries about physical implementation of an infinite memory tape, suppose that the tape has finite length.) A physical system that lacks semantic properties can reliably execute instructions for manipulating uninterpreted syntactic items. Thus, a physical system can implement a computational model even if the system lacks semantic properties.

Let us set this point aside. Suppose we grant, for the sake of argument, that all physical computations are representational. Can we infer that computation is 'sensitive' to semantics? Dietrich [1989: 133] claims that we can. I disagree. We must distinguish two questions:

Do semantic properties help determine whether a physical system is computational?

Do semantic properties inform how a computational system transits between states?

The first question concerns *individuation*, while the second concerns *dynamics*. One can consistently answer the first question affirmatively while answering the second question negatively. Fodor [1981: 226-227] adopts this combination of views in a famous passage:

---

[1] Some authors use slogans like 'computation is formal' or 'computation is semantically-laden' to summarize claims about individuation, rather than dynamics [Piccinini 2008]. I use the slogans to describe dynamics.

> I take it that computational processes are both *symbolic* and *formal*. They are symbolic because they are defined over representations, and they are formal because they apply to representations in virtue of (roughly) the *syntax* of the representations… What makes syntactic operations a species of formal operations is that being syntactic is a way of *not* being semantic.

Fodor endorses both the semantic view of computational implementation (all computational processes are representational) and FCC (computational operations apply to symbols in virtue of syntactic rather than semantic/representational properties).

2.2 Do Semantic Properties Help Individuate Computational States?

Another controversy concerns whether semantic properties inform the individuation of computational states. Egan [1999] and Piccinini [2008] argue that they do not. As Egan puts it, '[c]omputational states are individuated by a computational theory without essential reference to the contents assigned to them by the interpretation function' [1999: 180]. On her view, a computational system may have semantic properties, but those properties are irrelevant to its computational nature. They play no role in a bona fide computational description of the system. Peacocke [1999] disagrees. He argues that, in many cases, a state's semantic properties bear directly upon its computational nature.

To illustrate, consider a Turing machine that doubles the number of contiguous stroke marks. We cannot say that the machine computes a *number-theoretic* function until we interpret the stroke language. Depending on how we interpret strings of strokes, our Turing machine computes different number-theoretic functions. Notate a string of m strokes as '$\underline{m}$'. If $\underline{m}$ denotes the number m, the machine computes the function $f(n) = 2n$. If $\underline{m}$ denotes the number m-1, the machine computes $g(n) = 2n + 1$. Thus, a Turing machine computes a

number-theoretic function only given a semantic interpretation for the syntactic items it manipulates. Semantics bears upon which non-syntactic function, if any, a computational system computes. Peacocke concludes that purely syntactic description of a computational system does not exhaust the system's computational nature. There is a level of computational description that cites semantic properties. (Peacocke speaks of 'content-involving properties,' rather than 'semantic properties.' For our purposes, these locutions are basically equivalent.)

I find Peacocke's analysis convincing. However, the analysis casts no doubt upon FCC. One can consistently hold the following combination of views: a physical system's semantic properties inform its computational nature, so they help type-identify its computational states; yet the transitions between computational states are not sensitive to semantic properties. Not all features that help individuate a system's computational states need be relevant to the system's computational dynamics. For instance, consider a pocket calculator. The calculator's semantic properties inform which number-theoretic function it computes. In that sense, its semantic properties are bona fide computational properties. Yet it also seems plausible that transitions between the calculator's computational states are determined by an underlying formal, 'syntax-driven' process. As this example illustrates, one can use content to taxonomize a system's computational states while denying that content influences the system's computational evolution.

3. Computation, Syntax, and Semantics

Let us now examine whether the semantically-laden conception is plausible. I focus initially on 'classical' systems, which satisfy the following condition:

> **SYNTACTIC RULES:** Computation is manipulation of syntactic entities according to mechanical rules. We can specify those rules in syntactic terms, without mentioning semantic properties such as meaning, truth, or reference.

The Turing machine is paradigmatic. It manipulates syntactic entities according to mechanical rules that do not mention semantic properties. A typical rule runs as follows:

> If the scanner is in state q_1, and if it is currently scanning a memory location that contains symbol r_1, then replace r_1 with r_2 and shift to scanner state q_2.

This rule does not mention any semantic properties that r_1 or r_2 may have. In sections 5-6, I will generalize my discussion to consider systems that violate SYNTACTIC RULES.

A system can acquire semantic properties in two radically different ways, often described as *derived* versus *original* intentionality. These labels have connotations I want to avoid, so I will instead speak of *inherited* versus *indigenous* meanings. *Inherited meanings* arise when the system's semantic properties are assigned to it by other systems, through either explicit stipulation or tacit convention. Nothing about the system itself helps generate its own semantics. For instance, words in a book have inherited meanings. *Indigenous meanings* arise when a system helps generate its own semantics. Indigenous meanings do not arise merely from external assignment. They arise partly from the system's own activity, perhaps with ample help from other factors, such as causal, evolutionary, or design history. Virtually all commentators agree that the mind has indigenous meanings.

Consider again a simple Turing machine that doubles the number of strokes. In this case, it seems plausible that semantic interpretation results from external assignment, not from the machine's syntactic manipulations. Any meanings are inherited rather than indigenous. Quite plausibly, the same conclusion encompasses many computational devices employed in daily life, such as a pocket calculator or even a desktop computer. Do *any* classical computational systems have indigenous meanings? Searle [1984: 39] defends a

negative answer, encapsulated by the slogan ‘syntax is not sufficient for semantics.’ Many philosophers disagree, including Block [1990]. These philosophers hold that syntactic manipulation can help generate intentionality, at least if it interfaces appropriately with sensory and behavioural transducers. This response to Searle is sometimes called the ‘Robot Reply,’ because it maintains that a sophisticated robot can generate its own indigenous meanings. The robot’s syntactic manipulations, interfacing with perceptual inputs and motor outputs, help confer content upon internal robot states.

I focus initially on inherited rather than indigenous meanings. I consider a system whose syntactic manipulations, including any interactions with transducers, do not help generate content. Any semantic properties result solely from external assignment. Since the system’s syntactic machinations play no role in determining its semantics, we can hold the machinations fixed while varying semantics *arbitrarily*. Thus, the system satisfies the following condition:

> **FREEDOM:** We can offer a complete syntactic description of the system’s states and the mechanical rules governing transitions between states (including any interaction with sensory and motor transducers), while leaving semantic interpretation unconstrained. More precisely, it is metaphysically possible for the system to satisfy our syntactic description while instantiating arbitrarily different semantic properties.

One might summarize FREEDOM by the slogan ‘syntax does not constrain semantics.’ For example, consider again a Turing machine that doubles the number of strokes. Ultimately, the machine is just a device for mechanically manipulating formal syntactic items, subject to whatever interpretation we please. Depending on our linguistic conventions, a string could denote a number, or a person, or anything else, or nothing at all.

Admittedly, some semantic interpretations seem more ‘natural’ than others. Base-10 interpretation of a standard calculator seems more natural than a base-13 interpretation, since

the former but not the latter allows us to interpret the calculator as performing useful arithmetical calculations. However, if our interests were sufficiently perverse, then we could choose the 'unnatural' interpretation. In themselves, the calculator's syntactic machinations do not favour one interpretation over another.

Assume that computational system *P* conforms to SYNTACTIC RULES and FREEDOM. We can formulate a convincing argument that *P*'s computations are not sensitive to *P*'s semantic properties:

> ***The Reinterpretation Argument:*** By SYNTACTIC RULES, we can describe how *P* evolves solely by citing syntactic properties of its states, without mentioning semantic properties. By FREEDOM, our syntactic description is compatible with radically divergent semantic interpretations. We can arbitrarily vary *P*'s semantic properties without altering the underlying progression of states, as described syntactically. Thus, the transitions between computational states are determined solely by syntax, not by semantics. So *P*'s computations are sensitive to syntax but not semantics.

The Reinterpretation Argument is somewhat impressionistic, if only because it crucially relies on several unclarified notions: *syntax, semantics, sensitivity*, and so on. Nevertheless, it mounts an intuitively compelling case that computation is formal, at least within systems conforming to SYNTACTIC RULES and FREEDOM.

In the next section, I present an expanded version of the Reinterpretation Argument. In subsequent sections, I explore whether we can rebut FCC by examining computational systems that violate either SYNTACTIC RULES (sections 5-6) or FREEDOM (section 7).

4. The Reinterpretation Argument Expanded

Under what conditions does physical system *P* implement abstract computational model *C*? Most commentators agree that *P* must reliably conform to the mechanical rules encoded by *C*. Specifically, *P* must instantiate counterfactual patterns that mirror *C*'s mechanical rules [Chalmers 1995; Copeland 1996]. If *C* is a classical system, then the rules mention syntactic but not semantic properties, so the relevant counterfactuals look something like this:

(1) If *P* were to instantiate a state with syntactic properties α, then *P* would transit to a state with syntactic properties β.

To illustrate, consider a modified Turing machine *M* whose tape has finite length *L*. Suppose that *M* doubles the number of strokes, until it exhausts the tape. For all $n<L/2$, *M* falls under the following syntactic rule:

(2) *M* receives input $\underline{n} \rightarrow$ *M* outputs $\underline{2n}$

(2) supports counterfactuals about any physical system *P* that realizes *M*: for all $n<L/2$,

(3) If *P* were to begin computation by receiving $\underline{n}$ as input, then *P* would eventually halt with $\underline{2n}$ as output.

In general, any physical system that realizes a classical computational model satisfies counterfactuals roughly like (1).

I continue to assume that meanings are inherited rather than indigenous. I consider a classical system *P* whose syntactic manipulations do not help generate semantic properties. *P* may *have* semantic properties, but they result from external assignment, not from anything about *P*'s syntactic activity. My question is whether *P*'s computations are sensitive (in any interesting sense of 'sensitive') to these inherited semantic properties.

P has many properties: *being a certain distance from the Eiffel Tower, being my grandmother's favourite physical system*, etc. In most cases, we can vary those properties as we please without altering *P*'s computations. The computations are not in any natural sense

sensitive to the properties. Why should inherited semantic properties be any different? Why should *P*'s computations exhibit more sensitivity to inherited meanings than it exhibits to distance from the Eiffel Tower or to my grandmother's preferences? *C*'s mechanical rules describe how to manipulate syntactic entities, without any explicit mention of semantics. Why should physical computations that reliably conform to those rules be 'sensitive' to semantic properties arbitrarily imposed by external assignment?

By analogy, suppose that α is currently my grandmother's favourite syntactic property. Does this preference inform *P*'s transition from a state with α to a state with β? Of course not. But why are we so sure? Quite plausibly, because we can arbitrarily change my grandmother's syntactic preferences (say, through brainwashing) without changing *P*'s syntactic machinations:

(4) If *P* were to instantiate a state with syntactic properties α, but if α were no longer my grandmother's favourite syntactic properties, then *P* would still transit to a state with syntactic properties β.

As philosophers often put it, syntax 'screens off' grandmother preferences. Similarly, since *P*'s semantic properties result entirely from external assignment, we can alter them arbitrarily without altering *P*'s syntactic machinations (FREEDOM). If *d'* is an arbitrary interpretation distinct from *P*'s actual interpretation *d*, then *P* satisfies a counterfactual of the form

(5) If *P* were to instantiate a state with syntactic properties α but semantic properties *d'* rather than *d*, then *P* would still transit to a state with syntactic properties β.

Syntax 'screens off' semantics, just as it screens off grandmother preferences.[2]

Philosophers sometimes try to avoid such worries by shifting attention from counterfactuals such as (5), which mention both syntax and semantics, to purely semantic counterfactuals. To illustrate, suppose we stipulate that, for all *m*,

(6) $\underline{m}$ denotes m.

Then the purely syntactic rule (2) yields a purely semantic rule: for all $n<L/2$,

(7) M receives an input denoting $n \rightarrow M$ yields an output denoting $2n$.

Peacocke calls (7) a *content-involving description*, because it type-identifies states through their semantic (i.e. content-involving) properties. Assume that semantic interpretation (6) is 'counterfactually robust,' in the sense that it attaches to P in all reasonably nearby possible worlds. Then, quite plausibly, (7) supports counterfactuals: for all $n<L/2$,

(8) If P were to begin computation by receiving n as input, then P would eventually halt with $2n$ as output.

Thus, the content-involving generalization (7) supports counterfactuals. In many cases, we can offer content-involving descriptions not just of input-output behaviour but also of internal operations [Peacocke 1999: 197]. For instance, we can describe a computer as repeatedly *dividing by 2* before producing a final output. This description is content-involving, because it individuates computational states through representational relations to numbers. It supports content-involving counterfactuals.

Citing content-involving counterfactuals, Peacocke urges that Turing computation is sensitive to semantics. He concedes that 'the syntactic behaviour of a Turing machine is fully described by its machine table' [1999: 197]. But he insists that a Turing machine *also* falls under content-involving mechanical rules that support counterfactuals. Those rules treat computation as responding to semantic properties, not syntactic properties. As he puts it, '[t]he total story of a sequence of events at the syntactic level will indeed say nothing about content. This is entirely consistent with the causal story at the content-involving level making use of a notion of computation' [1994: 326].

---

[2] In a more thorough treatment, I would argue that we can rephrase this paragraph using the 'interventionist' counterfactuals emphasized by Woodward (2003).

I agree with Peacocke that we can often describe Turing machine activity at a purely semantic level. But that is not enough to rebut FCC, as manifested by the fact that Fodor [1994: 1-26] unabashedly emphasizes content-involving counterfactuals while also endorsing FCC. Physical system *P* conforms to content-involving counterfactuals by virtue of two facts: it conforms to syntactic counterfactuals, and it satisfies some semantic interpretation (in nearby worlds). Assuming FREEDOM, we can vary semantic interpretation arbitrarily while holding fixed *P*'s syntactic machinations. Apparently, then, semantics is just 'along for the ride,' floating atop an underlying syntactic evolution. Content-involving descriptions do not secure content any role in determining transitions between computational states.

By analogy, consider again grandmother-preference properties. Suppose that α is my grandmother's favourite property and that β is her least favourite. Assuming that her preferences are counterfactually robust, (1) yields

(9) If *P* were to instantiate a state that has my grandmother's favourite property, then it would transit to a state that has her least favourite property.

just as surely as it yields (8). Yet no one would suggest that (9) secures any interesting sense in which *P*'s computations are sensitive to my grandmother's preferences. No one would argue on the basis of (9) that computation is 'grandmother-preference-laden.'

How exactly we describe the situation here will depend on our background theories of explanatory and causal relevance. Perhaps we should concede that *in some sense P* is sensitive to semantics. If so, we should concede that *in the same sense P* is sensitive to my grandmother's preferences. In other words, there is no *interesting* sense in which computation is sensitive to semantic properties. Semantic properties do not inform transitions between *P*'s computational states any more than grandmother-preferences do.

Some readers may question the analogy between semantic properties and grandmother-preferences. Isn't there some relevant difference, such that the former but not the latter can influence the evolution of physical computation?

One difference is that semantics plays a central role in computer science, while grandmother-preferences do not. Since the inception of computer science, programmers have described physical computation in content-involving terms (e.g. *divide by 2*). For instance, the inventors of the world's first stored-program electronic digital computer, the Manchester 'Baby,' described it as performing arithmetical operations, including division and factorization [Williams and Kilburn 1948]. As Peacocke [1999] emphasizes, such descriptions offer several advantages over purely syntactic description. Perhaps most notably, they offer a distinctive order of generality. Physical systems that employ disparate numerical notations (e.g. Arabic notation versus binary notation) may fall under the same content-involving description. Such systems are heterogeneous under syntactic description but homogeneous under content-involving description.

However, these observations establish nothing about computational transitions. At best, they ensure semantics an *individuative* role, rather than a *dynamical* role. They suggest that computer scientists fruitfully use semantic properties to type-identify computational states and processes. They do not show that computational processes are any more 'sensitive' to semantic properties than to grandmother-preferences.

I conclude that the Reinterpretation Argument is compelling. If a system conforms to both SYNTACTIC RULES and FREEDOM, then its computations are formal. Hence, we cannot develop a viable semantically-laden theory that applies to *all* computational systems. The only hope for semantically-laden theorists is to focus more narrowly upon computational systems that violate either SYNTACTIC RULES or FREEDOM.

5. Non-Syntactic Computation?

Can we defuse the Reinterpretation Argument by studying systems that violate SYNTACTIC RULES? Since I have left 'syntax' unelucidated, it is not entirely clear what this would involve. However, commentators usually regard connectionist networks as non-syntactic. So one might propose that we shift attention from classical to connectionist models.

This proposal faces two problems. First, by surrendering classical computation to FCC, it concedes too much. Many cognitive science theories emphasize classical computation [Gallistel and King 2009]. If the semantically-laden perspective applies only to non-classical computation, then its scope is more restricted than one might hope. Second, connectionist systems themselves succumb to a variant of the Reinterpretation Argument, based on the following doctrines:

> **EVOLUTION:** We can specify the rules governing how a connectionist system evolves (e.g. back-propagation) by citing the weights between nodes and the activations of nodes, without mentioning semantic properties such as meaning, truth, or reference.
>
> **C-FREEDOM:** A complete description of the rules governing how weights and activations evolve does not constrain the connectionist network's semantic properties.

These two claims generate a

> ***Connectionist Variant of The Reinterpretation Argument:*** By EVOLUTION, we can describe how a connectionist system evolves solely by citing weights and activations, without mentioning semantic properties. By C-FREEDOM, our description is compatible with radically divergent semantic interpretations. We can arbitrarily vary the system's semantic properties without altering the underlying progression of states, as described in terms of weights and activations. Thus, the transitions between

> computational states are determined solely by weights and activations, not by semantics. So the computation is sensitive to weights and activations but not semantics.

EVOLUTION is highly plausible. C-FREEDOM applies to any connectionist system with inherited rather than indigenous meanings, since inherited meanings vary arbitrarily with respect to the underlying evolution of weights and activations. Finally, the Connectionist Variant seems about as credible as the Reinterpretation Argument itself. Thus, shifting attention to connectionist computation does not markedly alter the dialectical terrain.

6. Peacocke on Content-Involving Algorithms

Peacocke explores a different strategy for circumventing SYNTACTIC RULES. He holds that mechanical rules can operate directly over semantic properties, without intercession by syntax, connectionist weights and activations, or other non-semantic properties. He introduces the notion of a *content-involving computational description* of a pair of events [1994: 309]:

> (i) a specification of a content-involving property of the first event (or state);
>
> (ii) a specification of a content-involving property of the second event (or state); and
>
> (iii) a specification of a content-involving rule stating how the content-involving property given in (ii) is computed from the content-involving property given in (i).
>
> This third specification is a description of a content-involving algorithm.

According to Peacocke, computational psychology routinely subsumes mental events under content-involving computational descriptions. For instance, vision science might describe how the perceptual system computes an object's depth from a representation of retinal disparity. To accommodate computation's interface with sensory inputs and motor outputs,

Peacocke also allows 'mixed' algorithms, which describe inputs and outputs in non-content-involving terms. A mixed input-level algorithm might describe how retinal stimulations induce representations of retinal disparity, or how an intention to move one's arm induces a certain motor gesture.

As Peacocke emphasizes, the notion of a content-involving algorithm does not involve syntax in any essential way. To illustrate, consider the Euclidean algorithm for computing the greatest common divisor of two numbers. To perform the algorithm, one iterates the division operation, eventually arriving at the desired answer. The algorithm makes no overt reference to numerals or syntax, although for practical reasons we would typically execute it on paper. When we state it, we allude only to the number themselves, i.e. the denotations of our thoughts or our visual aids. Thus, the algorithm conforms to Peacocke's template (i)-(iii). It mentions numbers and arithmetical operations, but it does not mention syntactic mechanisms. Still less does it mention connectionist weights and activations. It operates at the level of representational properties, not at the level of non-semantic computational vehicles.

Does Peacocke's account vindicate the semantically-laden perspective? I will now argue that it does not. It is too schematic to constitute a satisfying alternative to FCC.

A pure *content-involving algorithm* is a rule specifying how to compute one content-involving property from another. Peacocke does not offer a general characterization of this notion beyond what I have already quoted. What counts as a 'rule' for 'computing' one content-involving property from another? Peacocke does not say. He places no restrictions on putative content-involving algorithms. Consider the following rule:

(a) Given the Gödel number of some sentence from the language of Peano Arithmetic, output 'yes' if the sentence is true, 'no' if false.

The function *f* 'computed' by this 'rule' is not recursive. By Church's thesis, *f* is not intuitively computable. Yet Peacocke's account does not disbar (a) as a content-involving algorithm. Nothing in Peacocke's account precludes even the notorious 'universal question-answerer' [Fodor 1981: 15]:

(b) Given any question as input, output the answer to that question.

Peacocke's account places no constraints on which functions are computable, or on which procedures defined over contents count as algorithms. As Egan notes [1999: 187], (i)-(iii) do not include anything specifically computational, besides the unexplained word 'compute.' The theory counts as computational literally any transition from one contentful state type to another.

How does Turing avoid lapsing into similar vacuity? As Fodor emphasizes [1981: 13-15], Turing isolates elementary operations upon syntactic entities, such as erasing or writing a stroke. He treats computation as iterated application of these operations, governed by rules that allude solely to the scanner's current state and to the syntactic item currently being scanned. Rules such as (a) and (b) are not definable within this restrictive setting, so they are disbarred. Similarly, connectionist models offer precise rules governing the evolution of weights and activations in the network, rules that leave no room for anything like (a) or (b). As these two examples illustrate, standard models of computation place heavy emphasis on instructions defined in non-semantic terms. Suppose we want to construct a machine that executes some content-involving algorithm (e.g. an algorithm for converting a representation of retinal disparity into a representation of depth). The typical strategy is to delineate appropriate *non-semantic* instructions. For Turing machines, the instructions dictate how to manipulate elements of the machine alphabet. For connectionist networks, the non-semantic instructions govern the evolution of weights and activations. Conformity to the non-semantic

instructions, coupled with satisfaction of an appropriate semantic interpretation, ensures conformity to desired content-involving instructions.

Can we construct a computational model whose mechanical rules cite semantic properties? At present, the suggestion is speculative. It does not reflect how actual computational models work. For instance, the rules governing Turing computation do not explicitly mention the denotations (if any) of symbols.

An exception is the *register machine*. The original [Shepherdson and Sturgis 1961] paper on register machines allows them to be defined directly over numbers, without any intercession by syntactic entities or other non-semantic states. The first register machine introduced in that classic paper contains infinitely many registers, 'each of which can store any natural numbers 0, 1, 2, …' [1961: 219]. The machine's elementary operations include 'add 1 to the number in register *n*' and 'subtract 1 from the number in register *n*' [1961: 219]. This register machine is defined over natural numbers, and its elementary operations are specified in content-involving terms. Plausibly, then, it conforms to Peacocke's approach.

Nevertheless, I doubt that this lone example provides much support for Peacocke's position. Natural numbers are a special case. The set of natural numbers is isomorphic to the set of numerals, so it is not surprising that one can define a mathematical model of computation directly over numbers. The question is whether that paradigm generalizes, yielding computational models directly defined over other content-involving states, such as *representing that an object has a certain depth*. Peacocke provides no hint how such a generalization might proceed.[3]

A satisfying philosophical theory of mental computation must ground itself in mathematical models of computation. Setting aside register machines, which do not seem to

[3] As Copeland and Sylvan [1999] note, Abramson's extended Turing machines can store arbitrary real numbers in memory cells. Similarly, Blum-Shub-Smale machines are basically register machines that can store arbitrary real numbers. Setting aside whether such models are genuinely computational, they do not operate directly over the representational contents posited by scientific psychology. Hence, they provide little help for Peacocke.

generalize appropriately, existing computational models do not operate directly over the semantic properties of computational states. Hence, given present mathematical knowledge, Peacocke's notion of 'content-involving algorithm' is too programmatic to constitute a compelling alternative to FCC.

Burge's recent discussion [2010: 95-98] faces a similar worry. According to Burge, the visual system instantiates computations governed by laws that '*cannot be described in purely syntactical or purely formal terms*. The principles that describe the transformations among states in the visual system concern specific kinds of perceptual --- representational --- states' [2010: 97]. Thus, Burge rejects SYNTACTIC RULES in favour of content-involving 'algorithmic laws' [2010: 97]. He holds that visual computation falls under content-involving algorithmic descriptions but not formal syntactic algorithmic descriptions.

Burge does not ground his conception in mathematical models of computation. He says that transformations in the visual system are 'effective procedures, procedures that follow an algorithm' [2010: 95], but he does not indicate what kind of computational model the transformations implement. To motivate the label 'computation,' Burge notes that the '(approximately) algorithmic laws of transformations among states in the perceptual system… can be modeled on a computer' [2010: 97]. This restriction blocks the counter-examples (a) and (b) facing Peacocke. But it does not distinguish vision from numerous non-computational processes, such as planetary motion, that can also be modelled on a computer. To treat a process as computational, we must show more than that one can model it on a computer. We must show that *the process itself* implements an abstract mathematical model of computation. Burge does not even mention abstract computational models. Ultimately, then, he does not identify anything specifically *computational* about transformations executed by the visual system.

7. The Indigenous Meaning Strategy

Setting aside the issues raised in section 6, a more fundamental worry remains. If a computational system's meanings are inherited rather than indigenous, then surely we can formulate a variant of the Reinterpretation Argument. Since the system's meanings result entirely from external assignment, the system presumably falls under a computational description that satisfies FREEDOM, C-FREEDOM, or some comparable constraint. Thus, we can change inherited meanings arbitrarily while holding fixed some underlying non-semantic computational description, whether syntactic, connectionist, or otherwise. For instance, suppose we describe a physical system through a register machine that operates over natural numbers. Suppose that the system represents natural numbers only due to external assignment. Then we can *also* describe the system through a register machine that operates over syntactic items --- numerals --- rather than numbers. Moreover, we can hold fixed the system's syntactic operations over numerals while arbitrarily reinterpreting the numerals.

I submit that a convincing semantically-laden theory should isolate a computational system that has indigenous meanings. We must identify a computational system that acquires semantics partially through its own activity, not merely through external assignment. We must then argue that the system's computations are sensitive to semantics. I call this *the indigenous meaning strategy*.

A few semantically-laden theorists pursue the indigenous meaning strategy. However, the results tend to be unsatisfactory. Consider Figdor's [2009] discussion. She rejects FREEDOM, as applied to human mental computation. Indeed, she proposes that semantic properties of mental states supervene upon syntactic properties. On that basis, she argues that 'we have no reason to think that semantic properties are not among those to which internal processing mechanisms are sensitive' [2009: 11]. Unfortunately, she provides no hint how

her view applies to existing computational models. For instance, she provides no hint how 'syntax' and 'semantics,' as she construes them, relate to a Turing machine table description. She does not discuss the mathematical theory of computation in any detail. Thus, her account is far too programmatic to mount a serious challenge to FCC.

Some indigenous meaning theorists make more effort to integrate their approach with current mathematical models of computation. As I will now illustrate, the resulting theories still tend to be unsatisfactory.

7.1 Simulation and Isomorphism

Cummins [1989: 87-113] introduces a notion that he dubs *s-representation*. A system s-represents some domain when the system's operations 'simulate' salient operations in the domain. For instance, if a calculator's syntactic manipulations are 'isomorphic' to arithmetical operations on numbers, then the calculator represents numbers [1989: 89-91]. Cummins concludes that even a pocket calculator, generates its own s-representationality. A calculator represents numbers simply by virtue of executing appropriate syntactic machinations. Thus, contrary to FREEDOM, content-involving counterfactual-supporting generalizations are already implicit in our syntactic description of a typical computational system. These content-involving generalizations secure a robust causal role for content. As Cummins puts it, 's-representational descriptions are going to be exactly as causal as descriptions in terms of [syntactic] functions satisfied' [1989: 135].

I am not sure whether Cummins regards computation as 'formal' or 'semantically-laden.' The answer probably depends upon how we gloss 'sensitivity to semantics.' Cummins does not address the issue. Either way, I believe that Cummins's account is problematic.

As Cummins himself emphasizes, s-representational contents differ significantly from familiar intentional contents, such as denotations and truth-conditions [1989: 136-141]. One sign of this difference is that Cummins attributes indigenous meanings even to a pocket calculator, whereas few philosophers would agree that a calculator helps confer traditional intentional content upon its states. Another sign is that, according to Cummins, 'representational contents are radically nonunique' [1989: 136]. A computational system that simulates one domain will simulate many others. So Cummins's account entails that a computational system s-represents many domains if it s-represents any. In contrast, most philosophers hold that representational mental states have determinate intentional contents. Thus, s-representation differs profoundly from 'semantics' as construed by Burge, Fodor, Peacocke, and most other philosophers.

Cummins retorts that scientific psychology should jettison traditional intentional content, replacing it with his simulationist proxy. As he puts it, CTM 'shouldn't --- indeed *mustn't* --- concern itself with intentionally specified explananda' [1989: 140]. In a crucial respect, then, Cummins agrees with Stich. Both philosophers eschew standard intentional contents, such as denotations and truth-conditions. Unlike Stich, Cummins proposes *substitute contents* that he dubs 'representational.' The substitution will not placate philosophers who seek a non-trivial role for traditional intentional contents rather than for Cummins's substitutes.

Of course, one might follow Stich and Cummins in eschewing traditional intentional content. I believe that Burge and Fodor have convincingly attacked any such manoeuvre. In particular, Burge [2010] argues convincingly that traditional intentional content is explanatorily central to contemporary perceptual psychology.

Cummins [1996] subsequently offers a new theory. Whereas his old theory holds that representation stems from an 'isomorphism' between computational operations and

operations in the represented domain, the new theory holds that representation stems from an 'isomorphism' between *the representation itself* and the represented domain. His paradigm is cartographic representation, which he claims involves an isomorphism between maps and physical space. O'Brien and Opie [2006] defend a related theory, according to which a computational state represents a domain if the state and the domain share relevant structural organization. For instance, O'Brien and Opie consider a connectionist network trained in a colour categorization task. Activation patterns in the network structurally resemble patterns in the reflectance spectra received as input. O'Brien and Opie conclude that the network represents colours by virtue of its own activity. Citing such examples, they defend a semantically-laden analysis: '*[c]omputations are causal processes that implicate one or more representing vehicles, such that their trajectory is shaped by the representational contents of those vehicles*' [2006: 32].[4]

These newer theories engender representational indeterminacy comparable to Cummins's original theory. Isomorphisms come cheap. Any system isomorphic to one domain will be isomorphic to many other domains. Thus, an isomorphism-based theory of 'representation' treats representational contents as radically nonunique. It thereby diverges markedly from traditional notions of intentional content.[5]

7.2 Block on Causal Relevance

Block [1990] elucidates meaning as functional role. By specifying a classical computational model of mental activity, including perceptual and behavioural transducers, we

[4] O'Brien and Opie restrict their semantically-laden perspective to connectionist and analog computation. They hold that digital computation is insensitive to semantics [2009: 56-57]. The key difference is that they accept FREEDOM while rejecting C-FREEDOM (along with a parallel doctrine regarding analog computation).
[5] Ramsey [2007: 93-96] emends Cummins's s-representational approach, hoping to avoid the indeterminacy embraced by Cummins. Ramsey also endorses FCC [2007: 85]. Thus, proponents of semantically-laden computation can hardly enlist Ramsey as an ally.

fix the narrow functional roles of mental representations. Block concludes that a sufficiently sophisticated computational system confers meanings (= functional roles) upon its own internal states. On this basis, he argues that semantics influences the evolution of mental computation.

The intuitive idea behind Block's account is that we cannot arbitrarily vary indigenous meanings while holding underlying syntactic activity fixed. In other words, Block rejects FREEDOM. For instance, we cannot change a representation's indigenous meaning from *water* to *danger* without serious changes in syntactic processing. Block concedes that we can change semantic properties in *certain* ways while holding syntax fixed [1990: 152]. In particular, we can change a representation's reference from *water* to Twin Earth *twater* without any underlying syntactic change. Syntactic description *constrains* semantics without *determining* semantics.

This intuitive idea seems reasonable. But the way that Block develops it is problematic. He endorses the following counterfactual:

(1) If mental representation *r* had had a different meaning, then it would have had different characteristic syntactic/motor effects.

As he puts it, 'it is false that a representation would have had just the effects that it did have if its meaning had been different. Different meaning requires different functional role, and different functional role requires different causes and/or effects' [1990: 151]. Block concedes that certain artificial changes to *r*'s meaning, such as a change from *water* to *twater*, entail no change syntactic/motor developments. But he contends that the 'closest' counterfactual scenarios in which *r* has a different meaning are also scenarios in which *r* has different effects within computation [1990: 152]. From (1), Block concludes that a mental state's semantic properties are causally relevant to syntactic/motor developments within mental computation.

Block's conclusion only follows if we presuppose a counterfactual theory of 'causal relevance' along these lines:

(2) (*C* is causally relevant to *E*) if (*E* would not have occurred had *C* not occurred).

Unfortunately, *backtracking* generates serious problems for (2). Following Woodward [2003: 139-140], suppose I take a pill that causes blood pressure decrease E_1 along with numerous side-effects $E_2, \ldots, E_n$ such as headache, chills, etc. It seems natural to say that:

(3) If E_2 & …& E_n had not occurred, then E_1 would not have occurred.

Together, (2) and (3) entail that side-effects $E_2, \ldots, E_n$ are causally relevant to the blood pressure decrease E_1. Yet the side-effects are *not* causally relevant to the blood pressure decrease. As Woodward argues, (3) follows even from Lewis's [1986: 47-56] theory of counterfactuals, which is designed to avoid backtracking. Apparently, then, Block presupposes a defective theory of causal relevance. We cannot accept Block's account as it stands.

8. A Mixed Verdict

There is only one promising strategy for developing CTM+¬FCC: the indigenous meaning strategy. Yet those few theories that pursue this strategy are problematic. Luckily, we did not discover any unifying flaw in those theories. We simply found that each theory exhibits its own particular flaws. Even though no one seems to have executed the indigenous meaning strategy in satisfactory detail, we found no reason to suspect that *the strategy itself* is hopeless. Thus, the strategy deserves further investigation. Many reasonable theories that espouse CTM+¬FCC may still await discovery.

I have not proposed my own semantically-laden theory. My goal has simply been to direct attention towards the most promising area of logical space. Rather than asking whether

computation *in general* is semantically-laden, we should ask whether computation *in systems with indigenous meanings* is semantically-laden. By focusing the debate accordingly, we promote more fruitful study of CTM.[6]

University of California, Santa Barbara

References

Aydede, M. and Robbins, P. 2001. Are Frege Cases Exceptions to Intentional Generalizations?, *Canadian Journal of Philosophy* 31/1: 1-22.

Block, N. 1990. Can the Mind Change the World?, in *Meaning and Method: Essays in Honour of Hilary Putnam*, ed. G. Boolos, Cambridge: Cambridge University Press: 137-170.

Burge, T. 2010. *Origins of Objectivity*, Oxford: Oxford University Press.

Chalmers, D. 1995. On Implementing a Computation, *Mind and Machines* 4/4: 391-402.

Copeland, J. 1996. What Is Computation?, *Synthese* 108/3: 335-359.

Copeland, J. and Sylvan, R. 1999. Beyond the Universal Turing Machine, *The Australasian Journal of Philosophy* 77/1: 46-66.

Cummins, R. 1989. *Meaning and Mental Representation*, Cambridge, MA: MIT Press.

Cummins, R. 1996. *Representations, Targets, and Attitudes*, Cambridge, MA: MIT Press.

Dietrich, E. 1989. Semantics and the Computational Paradigm in Cognitive Psychology, *Synthese* 79/1: 119-141.

Egan, F. 1999. In Defence of Narrow Mindedness, *Mind and Language* 14/2: 177-194.

Egan, F. 2010. Computation Models: a Modest Role for Content, *Studies in History and*

[6] I am grateful to C. Anthony Anderson, José Luis Bermúdez, Tyler Burge, Kevin Falvey, Ian Nance, Gualtiero

Philosophy of Science 41/3: 253-259.

Figdor, C. 2009. Semantic Externalism and the Mechanics of Thought, *Minds and Machines* 19/1: 1-24.

Fodor, J. 1981. *Representations*, Cambridge, MA: MIT Press.

Fodor, J. 1994. *The Elm and the Expert*, Cambridge, MA: MIT Press.

Gallistel, C. R. and King, A. 2009. *Memory and the Computational Brain*, Malden: Wiley-Blackwell.

Ladyman, J. 2009. What Does it Mean to Say that a Physical System Implements a Computation?, *Theoretical Computer Science* 410/4-5: 376-383.

Lewis, D. 1986. *Philosophical Papers*, vol. 2, Oxford: Oxford University Press.

O'Brien, G. and Opie, J. 2006. How Do Connectionist Networks Compute?, *Cognitive Processing* 7/1: 30-41.

O'Brien, G. and Opie. J. 2009. The Role of Representation in Computation, *Cognitive Processing* 10/1: 53-62.

Peacocke, C. 1994. Content, Computation, and Externalism, *Mind and Language* 9/3: 303-335.

Peacocke, C. 1999. Computation as Involving Content: a Response to Egan, *Mind and Language* 14/2: 195-202.

Piccinini, G. 2008. Computation Without Representation, *Philosophical Studies* 137/2: 205-241.

Pylyshyn, Z. 1984. *Computation and Cognition*, Cambridge: MIT Press.

Ramsey, W. 2007. *Representation Reconsidered*, Cambridge: Cambridge University Press.

Searle, J. 1984. *Minds, Brains and Science*, Cambridge, MA: Harvard University Press.

Shepherdson, J. and Sturgis, H. E. 1961. Computability of recursive functions, *Journal of the*

Piccinini, Nathan Salmon, Kranti Saran, and two anonymous referees for this journal for helpful feedback.

Association of Computing Machinery 10/2: 217-255.

Stich, S. 1983. *From Folk Psychology to Cognitive Science*, Cambridge, MA: MIT Press.

Williams, F. and Kilburn, T. 1948. Electronic Digital Computers, *Nature* 162: 487.

Woodward, J., 2003. *Making Things Happen*, Oxford: Oxford University Press.