Natural Computation

1. **Definition**: By ‘natural’ computation I shall mean simply computation that is performed by non-artifactual creatures, objects, or devices not specifically designed by us humans for computational purposes. Call such devices ‘natural computers’.

The brain, both of human and non-human animals, is widely held to be a natural computer, with various cognitive processes thereof being natural computations. Examples of such computations include the computation by brain’s the early vision system of the Laplacian of the Gaussian (Marr, 1980) and structure-from-motion (Ullman, 1979), motor control of grasping (Shedmehr & Wise, 2005), and the computation by the Tunisian desert ant’s brain of the displacement vector to its nest from any point along a foraging trajectory (Gallistel, 1990). There may be natural computers other than brains – perhaps certain single-celled organisms, maybe even Treky gas clouds for all I know.

Maybe one might prefer to speak of the entire organism, rather than the brain or some cognitive system, as the natural computer. Nothing of significance, I think, hangs on this.

But crucially, natural computers don’t include PCs, Apples, Blackberrys, Iphones, the onboard computers found in modern cars, fire control analog computers in battleships, etc.

I am not committed to there being a sharp distinction between natural and artifactual computers, but I do want to insist that there is such a distinction that we can all recognize, even if it turns out to be a graded one.

2. **The Question**: How are we to understand the claim that a certain natural (non-artifactual) object or device is capable of, or in fact performs, computations of some sort (or equivalently for our present purposes, that a certain non-artifactual object or device is a natural computer)?

In asking this question, I want to focus attention on the question of in what sense, if at all, so-called ‘computational’ theories of cognition are computational.

3. **Why the Issue Is Important**: In the absence of an understanding of just what it is for a natural object or device to be capable of computation, of what it is to be a natural computer, one can easily misunderstand theoretical claims to the effect that certain cognitive processes are computational, mistakenly assuming such processes are computational, if they are, in the same sense that certain causal processes in artifactual computers are computational.
One consequence of this misunderstanding is to fail to appreciate what I will term the fundamentally mathematical function-theoretic character of computational explanations of cognition – ‘function-theoretic’ in the sense of explaining the exercise of the cognitive competence in question in terms of the causal realization of a mathematical function, much in the way physics might provide a mathematical description of the behavior of some physical system, such that what one might pre-theoretically think of as the distinctively computational aspects of these explanations recede into the background.

A second consequence is to conclude that claims to the effect that human cognitive processes are computational is either demonstrably false (Searle) or demonstrably vacuous (Putnam) on apriori conceptual grounds.

A third consequence is to suppose that certain natural computational processes, specifically those that occur in brains, are of a special *sui generis* sort (instances of so-called ‘neurocomputation’), by virtue of certain special characteristics of the basic neural mechanisms that implement these computational processes, from which it might be thought to follow that proposed computational models of various cognitive processes (e.g., Marr, Ullman, Shadmehr & Wise, Gallistel, etc.) are false because they are not formulated in specifically neurocomputational terms.

4. **Argumentative Burden of Paper:** That philosophers have misunderstood claims to the effect that cognitive processes are computational, because they have assumed that such claims are to be understood by doing first a little bit of armchair conceptual analysis first on the notion of cognitive processes and then an extended bit of armchair conceptual analysis on the notion of computation. Indeed many papers on the topic do little more than this, paying little or no attention to the actual proposed computational explanations (and models) of cognitive processes. It is my contention that such apriori analyses shed little light on computationalism as an ongoing research program in cognitive science committed to the thesis that cognitive processes are computational.

Consideration of abstract notions of computation may well disabuse one of adopting one or another overly restrictive view of computationalism, e.g., one that sees it as essentially involving representations or information processing, but it doesn’t provide much by way of understanding computationalism as a research program, and even less by way of understanding just how computational cognitive scientists understand the claim that cognitive processes are computational. If we want to understand how computational cognitive scientists understand claims to the effect that cognitive processes are computational, we need to examine how the notion of computation is understood and used by these scientists.

A basic theme of the latter part of the paper is that quite a bit of the misunderstanding of natural computation can be attributed to a failure to appreciate the role of abstraction (and idealization) in scientific explanation, specifically the manner in which scientific explanations abstract away from detail that is irrelevant to explaining the explanandum in question. The effect of this failure is precisely to mask the way in which the notion of computation is understood and used in computational cognitive science, something that cannot be gleaned from an apriori analysis of the notion of computation as it is
used in computer science or computability theory which is completely divorced from explanatory practice in computational cognitive science.

5. **Overview of the Paper:** I begin by offering what I take to be some relatively uncontroversial reminders about how the basic notion of computation is used and understood by computational cognitive scientists, first as applied to artifactual computers and then as applied to natural computers. I argue that there is subtle shift in emphasis from the former to the latter the effect of which is to entail that computational cognitive scientists mean something different by computation in the two cases, such that we cannot understand computational explanations of natural processes on the model of an explanation of the computational processes of some artifactual computer. To attempt to understand the one in terms of the other will lead us to misunderstand where, and how, the explanatory work of these theories gets done. My aim here is to become clear on just what computational cognitive scientists mean when they call a natural object such as the brain (or some region thereof) a computer, and more specifically what they mean when they speak of such an object as capable of, or as performing, a computation of a particular sort.

6. **Computation:** Computation, as the term would imply, is in the first instance something that computers, either human or machine, do (if I were a philosopher of the Frankfurt school, which I’m not, I’d no doubt call computation a ‘praxis’).

The term ‘computation’ can refer either to the activity that takes place over an interval of time, viz., computing, or to the results of that activity, including perhaps not only the final result of some computing activity but also certain intermediate results.

As most computer scientists and computational cognitive scientists would have it, what gets computed in the course of a computation is the value of some mathematical function (e.g., the addition function) for certain arguments for which the function is defined. (For present purposes we can take functions to be mappings from sets into sets, members of the former set being the arguments of the function, members of the latter its values.)

The computation (of the value) of some function (for certain arguments) is effected by means of an **algorithm**, which for present purposes we can understand simply as ‘a procedure for computing the value of some function in a finite number of steps, each of which involves carrying out some operation’.

We could try to be more precise here about what we mean by ‘algorithm’ by spelling out just what we mean by a ‘procedure’ (‘operation’, ‘step’, and so on), and theories of algorithms try to do just this (e.g., by spelling out the notion of procedure, operation, and step in terms of Turing machines), but efforts to precisify the notion of algorithm typically (invariably?) fail to capture important aspects of our intuitive notion of algorithm, with the consequence that procedures the execution of which we would take to be computations turn out on these proposed precisifications not to be such. (It is some measure of the
difficulty of getting clear on the notion of algorithm that the question of what sort of thing algorithms are remains a live question the mathematical field of algorithmic analysis: there are all sorts of proofs regarding algorithms (existence proofs, equivalence proofs, speedup theorems, etc.), and yet there seem to be insurmountable difficulties with all proposed construals of algorithms as mathematical objects, which is what these proofs seemingly demand. Whatever algorithms are, and contrary to what we are often told, they aren’t much like your grandmother’s recipes, unless of course we are really deluded about your grandmother’s recipes.)

The notions of function and algorithm are nevertheless intimately related; indeed, some functions have no independent characterization beyond a specification of an algorithm that computes it. But by the same token, some functions (e.g., negation) have trivial one-step algorithms.

Formal theory of computation (aka computability theory) undertakes to (i) formalize these informal notions of function, algorithm, and so on, so as to be able to (ii) prove certain theorems regarding the computability of various classes of functions, by various kinds of abstract machines, under various constraints on memory, available procedures, and complexity measures of algorithmic time. These formal results, it should be emphasized, hold only for abstract machines. Actual physical devices satisfy the descriptions of these machines only under very significant idealization. It’s a real stretch to think of computation theory as concerned with real physical devices. This is especially true of complexity results which cannot be easily extrapolated to the behavior of real machines, both because the complexity characterizations are defined over infinite sets of computations and because the algorithmic time in terms of which these results are stated has no direct relation to clock time (except as defined relative to the computational resources, basic operations, and so on of specific physical devices).

7. Two Salient Levels of Description: The twin notions of function and algorithm define two particularly salient ‘levels’ of description at which the computations of a computer, at least an artifactual computer, can be described: (i) at the level of the function (what Marr called the ‘theory of the computation’), which specifies the function being computed, and (ii) at the level of the algorithm (what Marr called the ‘theory of the algorithm’), which specifies the algorithm by which a specified function is computed.

To speak of ‘levels’ of description is potentially misleading, for what we are really speaking about here are degrees (and indeed dimensions) of abstraction in computational description. Descriptions at every level abstract away from detail that would be included in descriptions couched at ‘lower’, less abstract levels of description. Thus, at the level of function, one abstracts away from algorithmic details about how the function is computed, while at the level of algorithm one abstracts away from details about how the algorithm is implemented by more basic computational operations and mechanisms. The choice of one level of description over another expresses a judgment about the appropriate level of abstraction to adopt, given one’s specific practical and theoretical/explanatory interests.

That the levels of function and algorithm should be so salient in computational description says something about their practical and theoretical utility, at least in the domain of artifactual computation: these are levels of abstraction at which it has turned out that one can capture important
generalizations about computations, at least of the sort that interest computer scientists, so much so that there are standard courses in the computer science curriculum (e.g., the ubiquitous two-semester Theory of Algorithms course) devoted to the study certain well characterized functions and the associated algorithms for computing the values of these functions.

The practical and much underappreciated theoretical upshot of the salience of these two levels of abstraction is that computational cognitive scientists come to their theoretical tasks equipped with a toolkit of familiar functions and associated algorithms for computing these functions, which is why much computational cognitive scientific modeling involves fitting known algorithms to new tasks, or modifying known algorithms in ways that enable them to compute new functions. Thus, e.g., Marcus’s (1980) famous parser for a version of Chomsky’s 1970s-era EST is by his own description an augmented version of a deterministic LR(k) parser for context-free languages familiar to any computer science student who has taken a course in compiler theory, Marr’s (1980) intensity gradient smoothing device that computes the Laplacian of the Gaussian is an application of a well-known noise filtering algorithm used in signal detection theory, and Rips (1993) computational model of human deductive reasoning is a modification of well-known Planner-based outside-to-inside theorem-provers.

It is hard to over-emphasize this ‘bricoleur’ approach (as French structuralists would call it) to computational cognitive scientific modeling, which so far as I know [Am I wrong about this?] has gone largely unnoticed by philosophers. Implicit in this approach is the non-trivial empirical assumption that natural computers are built out of a collection of shared (or at least sharable) algorithmic components, drawn from the same algorithmic toolbox as artifactual computers. At least there is such an assumption if the chosen algorithms are offered as empirical hypotheses as to how the functions in question are computed. One might ask whether there is any reason to suppose that natural computers are built from the same algorithmic toolbox as artifactual computers, indeed, as I shall suggest below, whether there is any reason to suppose that natural computers compute the functions that they do by means of algorithms at all. Maybe their implementation of these function-computing processes cannot plausibly be said to involve algorithms in any usual sense of that word.

There may be a case to be made that certain kinds of natural computers, e.g., brains, are constructed out of a limited number of neural hardware components [much in the way that logic circuits are constructed out of a finite stock of Boolean operations], but the path of reasoning that would take one from here to any conclusion about the contents, even the existence, of an algorithmic toolbox from which natural computations are constructed is pretty hard to discern. At very least, there may little reason to be confident that natural computers compute the functions that they do using familiar algorithms, in which case there would be little reason to take as serious empirical hypotheses proposed algorithms for the natural computation of the proposed functions.

8. Theoretical Preoccupation with the Function Computed:

It is hardly surprising that the computational descriptions that computationalists offer in the course of their theorizing should abstract away from irrelevant detail that might be included at a less abstract
level of computational description. The availability and choice of level of description serves precisely to enable theorists to tailor their descriptions to the explanatory tasks at hand. One finds a similar explanatory strategy being employed in all explanatory domains of science: one includes in the description only what is relevant to answer the explanandum question.

What is perhaps surprising is this: the level of computational description that appears to do most of the explanatory work in computational cognitive scientific theorizing is that of the specification of the function computed, so much so it would not be much of an overstatement to describe most such theorizing as fundamentally ‘function-theoretic’ (in the mathematical sense of the term). If we take these theories as they are presented in published work, the focus is invariably on the function computed by the relevant cognitive processes. Much less attention is paid to the algorithms that compute these functions than one might be led to expect, especially if, like any number of philosophers working on issues in computationalism, one comes to these issues thinking of computation in terms of algorithms.

There are, to be sure, a limited number of situations in which algorithms do figure in the explanations: (i) where there is no independent, non-algorithmic characterization of the function computed (thus, e.g., in the case of Marcus’s augmentation of an LR(k) parser, one needs to specify the algorithm in order to characterize the function computed by this augmentation); (ii) where for one reason or another one is concerned with the representational structures involved in the computation, in which case one must appeal to the algorithm in order to specify representational structures that the computational device maintains and over which it operations are defined, though clearly one’s hypotheses about representational structures will be no better than one’s hypotheses about the algorithms that utilize them; and (iii) where one is concerned with content properties of the computation that are implemented procedurally rather than represented explicitly (e.g., the implementation of various locality constraints in sentence processing models), in which case one must again appeal to details of the algorithm that implements the computation. But these sorts of cases aside, it is fair to say, I think, that the algorithms that compute the functions of theoretical interest are generally treated as being of little theoretical interest in themselves. And it is for this reason that computationalists can afford to be rather blasé about their use of admittedly ‘unnatural’, artifactual algorithms (e.g., that attribute to a cognitive system a memory in the form of a pushdown stack).

But why, we should ask, should there be this relative emphasis on the function computed rather than on the implementing algorithm? I suspect that it has mostly to do with computational theorists thinking of their theories as fundamentally mathematical, specifically function-theoretic in character, and only incidentally as computational in the sense of the function’s being implemented by one or another algorithm. Indeed, they may not even think of the theory as being committed to the idea that the hypothesized function is computed by means of some algorithm; maybe they think of the function as simply realized or implemented by certain neural mechanisms (surely that isn’t enough to make for an algorithm).

This emphasis on the function computed may also reflect the fact that in many cases what is being explained (the explanandum) is a particular cognitive competence, and as such a specification of the function that the natural device computes in the course of exercising this competence often suffices for the explanation.

Finally, this emphasis on the function computed may also reflect a wariness about taking too seriously any hypotheses regarding the algorithms that might compute the function, precisely because of uncertainty about whether the natural device (say, the brain) has the computational resources to
support any particular algorithm (say a pushdown stack): with greater abstraction comes greater confidence in one’s explanations, since one is abstracting away from just those details about which one is less confident. The price, of course, is that the explananda that one is able to explain are correspondingly more abstract. But this may well be a virtue if one is worried that the algorithms in one’s toolbox might not be the ones used by natural, non-artifactual computers.

9. Artifactual Computers:

Artifactual computers are designed with the express intent of being capable of performing specific computations (or classes thereof) under certain real-world constraints, where by this I mean being able to compute the values of specific functions. Of course, as we know too well, these machines do all sorts of other things (throw off heat, make various extraneous noises, malfunction, etc.) such that when we speak of them as computers, as engaged in certain computations, we single out a particular class of these machines’ behavior, which under an interpretation that we users impose on the machine we can construe this behavior as the computation of some function of interest to us.

This construal is not gratuitous. Pace Putnam and Searle, a device cannot be plausibly construed as computing any function whatever; otherwise there would be profit to be made selling rocks as hand calculators. But neither is the construal determined as a brute fact by the causal operations of the machine, precisely because there are all sorts of causal goings on in the machine, only some of which are relevant to the computational construal of the device, and even once we isolate the relevant causal goings on, there will still be any number of possible computational construals of those goings-on. But given certain plausible constraints on construal (constraints that reflect our cognitive limitations, interests, and so on), the plausible computational construals will be limited. The aim of the designer of artifactual computers is to constrain these construals even further, making the intended construal as transparent and natural as possible, so natural in fact that it never occurs to us users of the computer that there is any construal other than the one intended by the designer. (It is a testament to its Microsoft designers that we find it very difficult to think of our modern Word program as merely a function that maps sequences of key strokes into uninterpreted pixelations on the monitor screen, though this thought was pretty natural with early non-WYSIWYG word processing programs such as FinalWord that displayed their full command structure in the text.)

A rock, as Searle and Putnam pointed out, can compute the addition function. But only if we are prepared to accept a Goodmanesque interpretation of the rock’s taciturn behavior according to which this rock is at different times in different states representing first the addends and then the sum of a particular addition problem. The price we pay if we want to treat this rock as computing the addition function is that we have to adopt an interpretation of the rock’s behavior that presumes for its successful application that we are able to compute without the aid of the rock the very addition function that the rock is supposedly computing. Put another way, the price we pay for the rock’s computing the addition function is one in which the rock has no computation utility for us, since our interpretation requires that we be able to perform the very computation that we wanted the rock to do, namely, compute the addition function. No wonder there is little profit to be made from selling rocks as calculators (even though P.T. Barnum is surely right that there is a sucker born every minute)!
The point here is that in the case of artifactual computers, *computational utility for us users of computers* turns out to be a (the?) crucial criterion for determining the functions that these devices can be construed as computing. I don’t want to claim that utility alone provides a sufficient answer for the Searle/Putnam ‘my wall computes any arbitrary function’ claim. There are, I suspect, implicit empirical conditions on acceptable interpretation that add further constraint. (And I am not worried here about Kripkensteinian plus/quus issues, because I assume that in the design of artifactual computers, designers design the devices to model our computational dispositions, such as we understand them, over the domains in which they will be exercised.)

But there is an additional point to be emphasized here: to be a computer is to be amenable to being construed as a device that computes a particular function (or that can be so programmed). Computers have material and causal properties that enable them to be construed as such, but their actually being so construed is not among these material and causal properties. Being construed as something that computes a particular function is a kind of *perspectival* property that computers have in virtue of the way we users construe them, though our being able to construe them in these ways is constrained by their material and causal properties.

10. **Natural Computers**

Artifactual computers are designed and prized for their utility (for us users): in the best of cases they lend themselves well (and easily) to being construed as devices that compute specific functions (or are programmable for such). There is, therefore, a reasonably clear standard for whether they are what they are claimed to be and what they are claimed to do (as regards the functions they compute), just as there is a reasonably clear standard for being a teapot, toaster, or television.

Such is generally not the case for natural computers, which are not designed for our utility, if they are designed at all.

This raises an obvious question: in the absence of a standard such as utility-for-us, what is the appropriate standard of construal against which the claim to be a natural computer is to be evaluated, and what is the appropriate standard of construal against which the claim to be a computer that computes a certain function is to be evaluated?

This question is worth worrying about because it is not at all obvious that the notion of natural computation is coherent in the absence of the utility-for-us standard that disciplines our notion of computation in the case of artifactual computers.

So what is the appropriate standard against which the claim to be a natural computer, specifically the claim to be a natural computer that computes a certain function is to be evaluated?

We can get some traction on this question by considering explanatory hypotheses to the effect that a natural creature (or component thereof) computes a specific function (say, the Laplacian of the Gaussian) in the course of the exercise of some cognitive competence (in this case early vision). The
claim that the creature computes this particular function gains plausibility if there is some reason to suppose that determining the value of this particular function for particular arguments of the function is necessary, or at least important, to the exercise of the cognitive competence in question. In the case at hand, computing the Laplacian of the Gaussian of the retinal array produces a smoothed array that facilitates the detection of sharp discontinuities in light intensity gradients across the input retinal array (and hence the retina), the detection of these discontinuities being crucial to the visual system’s ability to detect edges and thus develop 3-D object representations of its environment. The claim that the creature computes this particular function would be much less plausible if we had no reason to suppose that the computation of the values of this function were crucial to the exercise of the competence in question. For suppose that we were told that in the course of visual processing the visual system computed some function the values of which played no role whatever in visual processing or any other cognitive process. We might be able to be convinced that there was an isolable causal process going on in the visual system that could be so construed, but I suspect that we would remain unconvinced that the visual system was really computing this function, not because we think that natural computers like the visual system compute only what is necessary for the success of their cognitive tasks, but because we appreciate that in the absence of principled constraints on what is to count as computed, computational claims are cheap. This suggests that the relevant constraint here is utility for a user, either us or a system: an isolable causal process counts as computing some function under a suitable interpretation of its inputs and outputs only relative to the judgment that computing the value of such function has some utility for a user.

So utility does have a role to play here, too, though it is now utility-for-the-system of which the natural computer is a component. The idea is that utility-for-the-system serves as a constraint on plausible construals of what function a natural computer is computing, just as did utility-for-us-users in the case of artifactual computers. And it is in virtue of its being plausibly construed as computing the value of some function, by virtue of the computation’s utility-for-the-system, that a component of the system can be construed as a natural computer that computes that function.

The rationale for this being the relevant consideration that constrains our construal of a natural device as computing some function (and other than seeming to accord with the practice of computational cognitive scientists) is something like this: in computational theories of cognition we are interested in explaining how certain classes of natural devices (e.g., species, cognitive systems) have the cognitive competences that they do. Of course, in the course of exercising these competences, these devices are doing all sorts of things, only some of which contribute to the exercise of the competence. What the computational model does is pick out those doings that are actually contributory to the competence in question, and furthermore are contributory in that they deliver the value of some function as output, which is then used in some way in the exercise of the competence in question. Call this the utility-of-the-computed-value-for-the-system.

Now this utility-of-the-computed-value-for-the-system may fail to determine uniquely the function being computed. Perhaps there may be two candidate functions, which are extensionally equivalent over the restricted domain in which the competence is exercised. These functions may be indistinguishable on behavioral/observational grounds (something like this is apparently true of the
arithmetic functions computed by hand calculators: they compute functions that coincide with the standard arithmetic functions [addition, subtraction, etc.] over the domain defined by the size of the display in which arguments are entered and values displayed. And yet there surely is a fact of the matter about what function this natural computer is computing (relative to a given standard construal of arguments entered and values displayed). Presumably the only way to decide which function is being computed is to look at the implementation of the computations, ascertaining whether the functions in question can be implemented on the available natural hardware. It might turn out that though we thought that the device was computing the addition function, since the pairing of arguments with values over the domain over which the competence was exercised was co-extensive with the addition function over that domain, the device might be computing a funky function ('funky' is here semi-technical notion) that is coextensive with the addition function over that domain.

But these sorts of cases aside, where we might want to look at implementation details in order to determine which of two functions a device was computing, the fact is that it is still utility-of-the-computed-value-for-the-system that is relevant for deciding how to characterize the behavior of the device over the domain that is relevant to the exercise of the competence, viz., as computing the value of a particular function. Here looking at implementation details does little work; at most it can rule out certain proposals as implausible, again relative to certain constraints on interpretation of causal processes. Looking at implementation details doesn't tell us how to characterize the behavior of the device in computational terms, because such details don't address questions of utility for the system.

Finally, as with artifactual computers, so too with natural computers: computing a particular function is a perspectival property, but in the latter case it is one that these devices have in virtue of the explanatory utility that such descriptions have for us who are concerned to understand the behavior of these devices.

11. A Worry about Utility:

There is a worry about appeals to utility as a way of determining just which function a computer is computing, and it's this: there are all sorts of goings on that have utility for us or for the system of which they are a part, and yet there is no inclination to think of these useful goings-on as computational. Digestive processes, for example, are clearly useful to us, but surely they aren't computational, and surely the gut, for all the useful things it does for us, isn't a natural computer.

So how do we distinguish those useful processes that are from those that aren't computational? It's tempting at this point to distinguish computational processes as algorithmic, informational, regulatory, or some such, but I'm inclined to suspect that all such proposals have problems of their own and ultimately won't do much work, but more importantly they don't, I think, get at what makes some processes, and not others, computational, namely, being processes that compute the value of some function. But the problem here is how to distinguish such processes from those that are 'merely' function describable, such as the dynamic flow of a liquid through a nozzle.
What seems crucial to the notion of a *computational* process is that it be one that is undertaken for the express purpose of determining the *value* of a particular function – determining this value is its *raison d’être*. It is here that we start to distinguish computational processes from other processes useful to a system, e.g., digestive processes: computational processes are useful to the system by virtue of the usefulness to the system of the value of the function computed, which suggests that it is not simply that the output of the process is useful to the system, but also that this useful output can be plausibly construed as the value of some function, say a number, a vector, a numerical array, etc. – something that is not plausibly true of the products of digestive processes.

But how, then, are we to distinguish outputs of processes that are and outputs of processes that are not plausibly construed as the computed value of some function? I don’t see any way of doing this that doesn’t ultimately end up counting as the computed values of functions those outputs that are useful to the system in which they figure in very much the way that the computed values of functions with which we are familiar are useful to us as persons. In other words, we base the distinction between processes that are and processes that are not computational on an intuitive grasp of the notion of the computed value of a function, a grasp that rests (as a philosopher of the Frankfurt school would put it) on our familiarity with computational praxis.

As a way of distinguishing computational for non-computational process, this is admittedly vague (and, I suppose, praxis-centric), but far from being a defect of my proposed construal, I take it to be a strength, as it seems to me that the distinction is not that clearcut in the first place. Thus, for example, it is not at all clear to me that Gallistel’s function-theoretic description of the desert ant’s capacity for path integration is computational in nature, precisely because it is not at all clear that the output of this process is sufficiently like the values of computed functions with which we are familiar to count as computational. What I mean by this is that the ant is clearly doing something that is function-describable as path integration, the result of which it can use to find its way back to its nest, but it is unclear whether the value of this function is being computed in any intuitive sense that we recognize as computational. There is, for example, no reason to suppose that the ant determines the value of the function in question by means of some algorithm that the ant executes. If we think of the ant as computing an algorithm, it is only because we have concluded that the ant is computing a function and furthermore think that computing necessarily involves executing an algorithm. The ant simply behaves in a manner that suggests that at every point along its foraging trajectory it ‘knows’ the value of a displacement vector from the nest to that point, such that it is illuminating to describe the ant as doing path integration even in the absence of any real idea how the ant does it. (I am told that we do know that the ant is probably making use of cumulative side-specific leg movements, perhaps by keeping track of side-specific energy expenditures, inasmuch as its path integration skills decrease significantly on finely powdered surfaces where it lacks traction.) Similarly, even for Marr’s theory of early vision, where it is often a stretch to think of supposed computations as delivering the value of some function. But, and this is the crucial point here, I am not sure that it really matters. For the explanatory force of such explanations seems to be carried by the function-theoretic descriptions of the various processes and not at all, so far as I can see, by the fact that these processes are also computational in a way that would
distinguish them from function-theoretically described causal processes more generally.  *The notion of computation seems to be doing much less work in these explanations than philosophers may imagine.*

But irrespective of whether one thinks of these processes as computational or as merely function-theoretically described, utility-of-the-computed value-of-the-function-for-the-system serves as a criterion for choosing among alternative function-theoretic construals of the causal processes in question.

12. A Limited Teleology without Representation

The appeal to utility in answering the question of what function is being computed in the course of a particular natural computation is teleological in spirit, but the teleology here is enforced by our explanatory goals, not as Millikan and others would have it by broadly teleological-cum-evolutionary considerations. The teleology here is *epistemological*, not metaphysical or ontic, and it is a teleology that emphasizes the role that the computation of the function plays in the overall economy of the device computing that function.

Perhaps as importantly, it is the utility of the device or component, taken as a whole; it is not, as Millikan and others would have it, a teleological constraint levied on the contents of such representations as the device may maintain. This is crucial because as the focus is on the function(s) computed and not on algorithms, there has really been no discussion or commitment whatever to representations of any sort, though of course if the cognitive process in question involves a number of interacting computational operations, each of which computes some function, then one might want to think about the outputs of one computation that are then the inputs to another (as, e.g., in Marr, 1982) as representations. But we should be careful about taking representation talk here to seriously, because there is a case to be made that in most cases the commitment to representations is simply a by-product of our having chopped up the cognitive system into sub-components that compute functions for which we have a well-understood algorithm in our algorithm toolbox. In any event, the crucial point here is that such teleology as there is in my account of what determines the function is being computed accrues not to these representations but to the computation of functions that have these representations as inputs and outputs.

Intentionalist construals of computational models of cognition (e.g., Burge’s) depend crucially on the representational structures of these models to explain the utility of these models for their possessors. Such construals explain the utility of the computations in terms of the reference-like relation that the representations bear to the distal environment, the idea being that the computations are useful to the creature precisely because they construct and maintain such representations. Thus, for example, certain computations in Marr’s account of early vision detect abrupt changes in light intensities across the retinal array and then use these detected intensity changes to construct a representation that depicts these changes as lines on the representation that Marr calls the raw primal sketch. These computations, according to Burge, are useful to the early visual system precisely because it turns out that these lines in the usual case represent the edges of objects in the distal environment. The present account gets the connection to utility without appeal to a reference relation and without commitment
to representations. On the present account the connection to utility is motivated purely by the utility of the computations for the system. This utility determines the appropriate computational description inasmuch as it is that description which explains the cognitive competence that is the explanandum. It does this simply by focusing our explanatory attention on certain causal processes and not others, namely ones that in fact, in our environment but not in others, serve to identify edges of objects in the distal environment.

13. **Computational Explanation Is a Fairly Common Form of Mathematical Scientific Explanation:**

An immediate consequence of my proposed answer to the question of how we determine what function a natural computer is computing is that computational descriptions are, like other mathematical descriptions in science, just a way of isolating the internal causal goings-on of the system that are relevant to the explanandum in question. In the usual case where we are concerned to explain some cognitive competence, our aim is to explain how it is that the system is able to do what it does in the exercise of the competence in question, and we do this by isolating just those causal goings-on that are constitutive of the exercise of the competence, ignoring as irrelevant any goings-on that don’t contribute directly to the competence. We want to explain how it is that the system can do what it does, has the competence that it does. Computational descriptions do this for us, by ignoring all sorts of other goings-on, many of which are no doubt important for the well-being and proper functioning of the system (maintaining energy levels, pumping blood, etc.).

Computational descriptions not only ignore as irrelevant causal goings-on that don’t contribute directly to the competence being explained, but they also ignore as irrelevant many details about those causal goings-on that do contribute directly to the competence being explained. Thus, for example, in providing computational explanations of various cognitive competences, theorists focus almost exclusively on the functions computed, eschewing details not only about hardware implementation but in many cases details about such algorithms as may compute these functions.

The particular picture of computational explanation that seems to emerge from an examination of the actual explanations of computational theorists is one in which computational explanations, like mathematical scientific explanations more generally, are severely abstractive in character, abstracting away from what are taken to be irrelevant detail having to do both with various causal goings-on in the device and with the constitution of the device in virtue of which it can exhibit those causal goings-on that are assumed to be relevant to the explanandum. On this picture, computational explanations are narrowly focused answers to quite specific explananda questions, where, for what are presumably pragmatic reasons that an account of scientific explanation would explicate, the explanation provides only such account of the causal goings on as is minimally necessary to answer the explanandum question.

As an example of the sort of scientific explanation that I take computational explanations to instance, consider continuum mechanical explanations of the molar behavior of fluids, gases, liquids, plasmas, and so on. As Wikipedia explains, ‘modeling objects in this way ignores the fact that matter is made of
atoms, and so is not continuous; however, on length scales much greater than that of inter-atomic distances, such models are highly accurate. Fundamental physical laws such as the conservation of mass, the conservation of momentum, and the conservation of energy may be applied to such models to derive differential equations describing the behavior of such objects’. The crucial point for present purposes is that while the materials that continuum mechanics treats as continuous are not in fact continuous, at the level of abstraction at which they are described and their behavior explained, there is no difference between the behavior of continuous and non-continuous materials. Treating these materials as non-continuous assemblies of constituent objects would not lead to different theoretical descriptions of their behavior at the level of description at which they are described and their behavior explained. Scientists could of course opt to include information about the non-continuous constitution of these materials that are treated mathematically as continua, but it is not as if this information would add anything to the mathematical description of the molar behavior of these materials. The microstructure of these materials ‘washes out’, as it were, from their mathematical description. [Biology, Arnon assures me, is replete with similar examples, e.g., Moran process.]

Much the same sort of thing (or maybe precisely the same sort of thing), I want to argue, goes on in computational models of cognition: the level of abstraction at which both the explanandum and explanation are couched is such that details about the hardware implementation of the basic operations that effect the computations simply ‘wash out’ as irrelevant to explanation. In many cases the algorithms themselves turn out to be irrelevant to the explanation, if indeed there are such: to understand how the device is able to do what it does, exhibit the competence that it does, it may be enough simply to know that it computes a certain mathematical function (or functions). It is in such cases (where it is enough) a further question how the device is able to compute this function, perhaps by executing a certain algorithm, or perhaps by executing a certain algorithm by means of certain basic operations, using available hardware (though I have suggested above that it is doubtful that computational cognitive explanations are committed to the idea that the attributed functions are computed by means of algorithms).

Now, there is obviously a philosophical issue here about how to think about scientific explanation, an issue over which there can be (and has been) a lot of debate and spilling of ink, but however one comes down on this issue, it is pretty clear that much of what computational cognitive scientists present as explanations of various cognitive competences take just such a rather sparse, abstract form in which what is offered by way of explanation is the minimal necessary to explain, at a fairly abstract level of description, how the device manages to do what it does, and this explanation, I claim, very often takes the form of simply a specification of the function(s) computed by the device in the course of the exercise of the competence, along with what Frankie calls an ‘explanatory gloss’ that makes clear why this specification explains what it claims to explain. None of the cognitive scientists who offer these explanations would deny that there are further questions to be asked and answered about the causal processes that are involved in the exercise of the cognitive competence in question, but few, if any, of these scientists would hesitate to describe their abstract explanations as explanations. For them, scientific explanations are narrow answers to specific explanandum questions, even if in the usual course of scientific inquiry, answering one scientific question invariably raises others.
14. **Neurocomputation and Computational Cognitive Theory:**

The issue about what is to count as a computational explanation might seem to be little more than a verbal dispute over how to use a term with honorific overtones. But there *may be* a substantive issue lurking in the background here. Recall Gualtiero’s paper on neurocomputation in which he argues on the basis of the nature of spike trains that basic neurocomputational operations can be neither digital nor analogue, and hence that neurocomputation is *sui generis*. From this he concludes that ‘computational theories of cognition that rely on non-neural notions of computation ought to be replaced or reinterpreted in terms of neural computation’. Leaving aside any questions one might have about the argument for this conclusion, the implication would seem to be that computational theories of cognition of the sort mentioned above (Marr, Ullman, Marcus, Berwick, Gallistel, Shadmehr & Wise, etc.) are defective precisely because they rely on non-neural, perhaps digital, notions of computation.

If I am right about the level of abstraction at which these theories are couched, viz., a level at which they are concerned almost exclusively with the function computed in the course of the existence of the competence in question, then it is simply not the case that they rely on a notion of computation that is intrinsically neural or non-neural, digital or non-digital, or whatever. The function-theoretic explanations that these theories present abstract away from such details, and there is little or no attention paid to questions about how these functions get computed. It is enough for the purposes of these explanations that the natural computer in question can in fact compute the imputed function. How it manages to do this is another question altogether. Of course there may be other computational tasks performed by the brain, tasks that one might insist (misleadingly, in my view) on calling ‘cognitive’ simply in virtue of the fact that these tasks are performed in the course of performing higher level cognitive tasks, where implementation details of the sort Gualtiero discusses will be relevant to an explanation of the performance of these tasks. But the fact that there are computational tasks where neural implementation details are relevant hardly shows that such details are also relevant in the explanation of higher level cognitive tasks. And mere stipulation about what is to count as a scientific explanation won’t change that fact.

It is important not to confuse the view I am endorsing here with the so-called ‘autonomy’ view of cognitive explanation, defended by Fodor, Pylyshyn, Gallistel, and others, according to which neural mechanisms are relevant only to questions regarding the implementation of computational cognitive processes. The present is a view about scientific explanation, to the effect that scientific explanations should include (and in point of fact typically do include) only such detail as is minimally necessary to answer the explanandum question. So conceived it is an empirical matter, and *not* a matter for apriori stipulation, just which detail is, and which is not, relevant to the explanation of some phenomenon.

Matters of neural implementation *might* turn out to be relevant to theorizing about cognitive competence at the level of abstraction illustrated by the work of Marr, Ullman, Marcus, Berwick, Shadmehr & Wise, etc. It might, for example, turn out that neural mechanisms are simply unable to compute the functions that these theories attribute, or might be unable to compute them in a way that matches their observed complexity profiles, etc. But these will be empirical discoveries, not the result of apriori stipulations about what is to count as a cognitive scientific explanation or a scientific explanation simpliciter. But even if such matters of neural implementation do turn out to be relevant, notice that
that their effect will be to affect the sort of function being computed. The explanation will still be
function-theoretic and will not advert to the details of neural implementation that dictate the identity of
the function computed. The explanation still abstracts away from such details.

12. Conclusion:

Let me restate here some of my main points and/or conclusions:

(1) Levels of computational description are in fact levels of abstraction in description, where detail that
is absent at one level of description is included at certain lower levels of description. Choice of level of
description reflects a decision about relevant detail, and where relevance is always relative to pragmatic
and explanatory interests. That the two levels of the (i) function computed and the (ii) algorithm are of
particular interest in computer science tells us something about our pragmatic and explanatory
interests. The practical and much underappreciated theoretical upshot of the salience of these two
levels of abstraction is that computational cognitive scientists come to their theoretical task with a
toolkit of familiar functions and associated algorithms for computing these functions, which is why a lot
of computational cognitive scientific theorizing involves fitting known algorithms to new tasks, or
modifying known algorithms to enable them to compute new functions. Implicit in what I am calling the
‘bricoleur’ approach to computational cognitive scientific theorizing is the non-trivial empirical
assumption that natural computers, like artifactual computers, are built out of a collection of shared (or
at least sharable) algorithmic components. (pp. 4-5)

(2) The level of computational description that does most of the explanatory work in computational
cognitive scientific theorizing is that of the specification of the function computed, so much so it would
not be much of an overstatement to describe most such theorizing as fundamentally ‘function-theoretic’
in the mathematical sense of the term. It is a mistake to assume that these functions must be
computed by an algorithm. (p. 5)

(3) In the case of artifactual computers, utility for us users of computers turns out to be the crucial
criterion for determining the functions that these devices can be construed as computing. (p. 7) In the
case of natural computers, utility for the system turns out to be the crucial criterion for determining the
functions that these devices can be construed as computing. (p. 9)

(4) The appeal to utility in answering the question what function is being computed in the course of a
particular natural computation is teleological in spirit, but the teleology here is enforced by our
explanatory goals; the teleology is not metaphysical or ontic. And perhaps as importantly, it is the utility
of the device or component, taken as a whole; it is not a teleological constraint levied on the contents of
such representations as the device may maintain. (p. 10)

(5) Computational explanations of cognition, like other mathematical scientific explanations more
generally, isolate the causal goings-on in the system that are relevant to the explanandum question and
describe only those goings-on. Irrelevant detail is ignored. At the level of abstraction at which both the
explanandum and explanation of cognitive phenomena are couched, details about the hardware implementation of the basic operations that effect the computation, and often even the algorithms themselves, typically ‘wash out’ of the explanation as irrelevant. (pp. 10-11)

(6) Given the level of abstraction at which computational explanations of cognitive competences are typically couched, a level at which they are concerned almost exclusively with the function computed in the course of the exercise of the competence in question, it is simply not the case that these explanations rely on a notion of computation that is intrinsically neural or non-neural, digital or non-digital, or whatever. The function-theoretic terms in which these explanations are typically couched abstract away from such details. It is enough for the purposes of these explanations that the natural computer in question can compute the imputed function. (p. 12)