THE CONCEPT OF COMPUTATION IN COGNITIVE SCIENCE

1.) INTRODUCTION

Two theses, which I call the "Computational Theory of Cognition" (CTC) and the "Representational Theory of Intentionality" (RTI), form the foundation of cognitive science.[1] In its broadest statement, RTI asserts that mental states are about the world (have content) in virtue of a representation relation holding between the world and those states. The CTC holds that cognition is the computation of complex functions on such representational states, where computation consists in performing operations defined over the syntactic structures of representational states. CTC/RTI portrays cognizers as receiving input (usually via sensory organs and/or memory) and generating outputs (memories, inputs to other processes, and/or motor response commands).

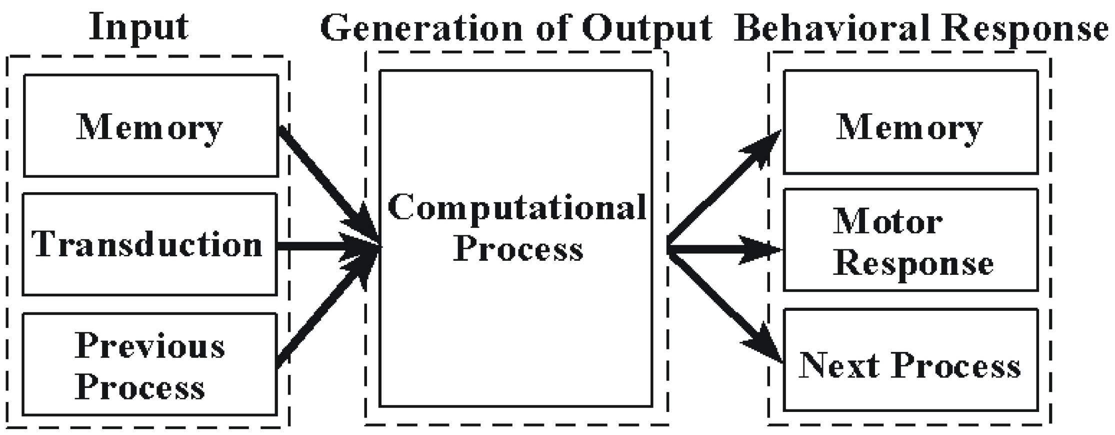


Figure 1

The notion of computation generally assumed relevant to the CTC is Turing computation, and its development in contemporary computer science and mathematics. I argue that the basic Turing concept of computation is an inadequate and/or inapplicable model of the actual functioning of many of the systems studied by cognitive science. Extensions of Turing computation to the real numbers and functions on the real numbers, e.g. by Grzegorczyk (1955, 1957), fail to be sufficiently general and physically realistic. I conclude that adopting a Montague (1962) notion of computation better serves cognitive science since it extrapolates from Turing computation in a way that encompasses all the systems considered by cognitive scientists. I then argue that objections in the literature fail to undermine a Montague approach.

2.) TURING COMPUTABILITY AND ITS PROBLEMS

To illustrate the standard explication of computational explanation in cognitive science and the Turing notion of computability, consider a purely computational explanation of a system's, ALAN's, capacity to compute the successor function on a finite set of the natural numbers. One encodes a number as a series of strokes on a sheet of blank squares (one stroke per square, starting at the second square), and sets it before ALAN. It proceeds as follows:

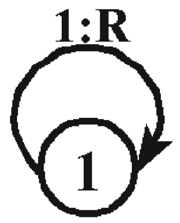


Figure 2
Starting in state S_1 at the second square, ALAN moves the sheet one square to the right each time it sees a stroke in the square, and looks at the new square. (Circles indicate ALAN's state. Arrows encode state-dependent operations: In S_1 ALAN moves the paper to the R(ight) when it sees a '1'.)

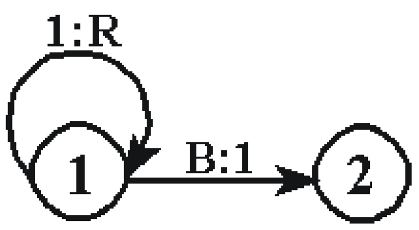


Figure 3
Upon seeing a blank square, ALAN writes a stroke (B:1), enters S_2, and looks at the modified square.

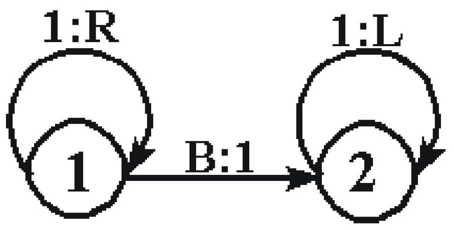


Figure 4
ALAN henceforth moves the sheet one square to the left each time it sees a stroke (1:L) and looks at the new square.

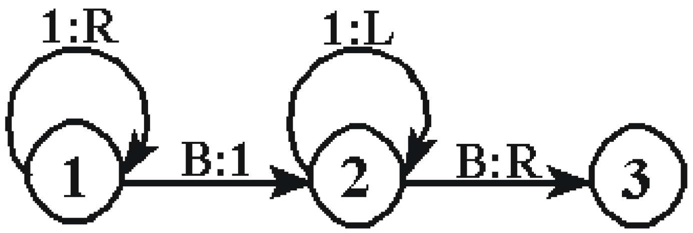


Figure 5
Upon reaching a blank square, Alan moves the sheet one square to the right (B:R), enters S_3, and stops .

ALAN, as a finite instantiation of the above-diagramed Turing machine, straightforwardly computes the function (x + 1). Applying the computational explanation schema given earlier, the first stage involves stroke inputs, the second stage involves dispositions to operate upon the inputs, and the third stage is ALAN stopping at the beginning of the new stroke string.

A computational explanation of ALAN's cognitive capacity to compute the successor function is generally thought to involve three steps; (1) type-individuate ALAN's states, (2) map its input and output states to the relevant objects/properties in the task domain by the representation relation (here the relevant "objects" are numbers), and (3) utilizing the representation relation, show how ALAN's state transitions generate the correct output for each input within the task domain, often including significant intermediary steps. Sometimes step (3) takes the form of a proof given (1) and (2), often it takes the form of a generalization based upon experimental results. No one, for example, attempts to prove that chess computers are master level players. Rather, these programs are evaluated by their ability to beat chess masters in actual games.

Beginning in the initial state, ALAN looks at the first of a string of n strokes, moves (n + 1) squares, writes a stroke, and returns to the beginning of the new string of (n + 1) strokes. If ALAN begins looking at the first stroke of a six stroke string, it moves seven squares, writes a stroke, and ends looking at the first stroke of seven strokes. Thus, one can type ALAN's inputs and outputs according to the number of strokes on the sheet. One can then assign representational content via the standard interpretation as follows:

1.) A blank sheet of paper maps onto zero.

2.) A sheet with n strokes maps onto the number n.

Explanations of an actual machine's computational ability differ somewhat from computability proofs. In logic and mathematics, the generally operative notion of computability for the positive integers is tied only to the notion of a Turing machine given the Church-Turing thesis (i.e., a function of the positive integers is effectively calculable only if recursive). In other words, the notion of computability has only two components: (1)

The assumption that the set of effectively computable functions on the positive integers is the set of Turing computable functions. (2) The notion that a Turing machine computes a function iff for every argument/value double in the set that characterizes the function, the machine would generate that value (under the interpretation, and with certain other restrictions) given the argument or set of arguments (under the interpretation). The mathematical notion of computation makes no reference to one's ability to exploit a Turing machine's computational ability nor to tractability, i.e., to one's actually being able to generate and use an interpretation, and to the constraints on computability given real world resource/time constraints respectively. While not necessarily relevant to theoretical explications of computation, such concerns are paramount in the explanation of the computational abilities of real world systems.

Shifting to ALAN's real resource constraints necessarily modifies the explanation. For example, the size of the sheets of paper places a limit on the size of the positive integers that ALAN can represent. Such factors serve to truncate the actual task one attributes to a cognitive system. ALAN can compute the $(x + 1)$ function for positive integers within a certain range. Relative to this task specification, the above assignment of numbers to strokes on sheets of paper works fine.

One knows how ALAN behaves given an input, so one can predict what physical state (now representational state) results from ALAN's state changes given some input (now representational state). One uses these predictive results to make a further claim about ALAN: Within its finite domain, ALAN's state transitions generate an output representing the successor of the input (under the representational schema).

John Haugeland (1981) explicates the structure of explanation within the computational theory of cognition (CTC) in terms of a semantic engine: Cognizers are physical engines--dynamic systems operating upon physical principles. By understanding physical engines in terms of type-individuated, syntactically structured states and principled interactions between such states, one can understand how physical engines instantiate syntactic engines--dynamic systems operating in accordance with syntactic principles and doing syntactic work. Finally, via an interpretation function providing a semantics for the syntactic engine, one can understand how syntactic engines instantiate semantic engines--dynamic systems operating according to semantic principles and doing semantic work. Moreover, this strategy potentially applies to multiple levels of description of a given machine.[2]

One can illustrate the general explanatory mechanisms as follows (figure 6):

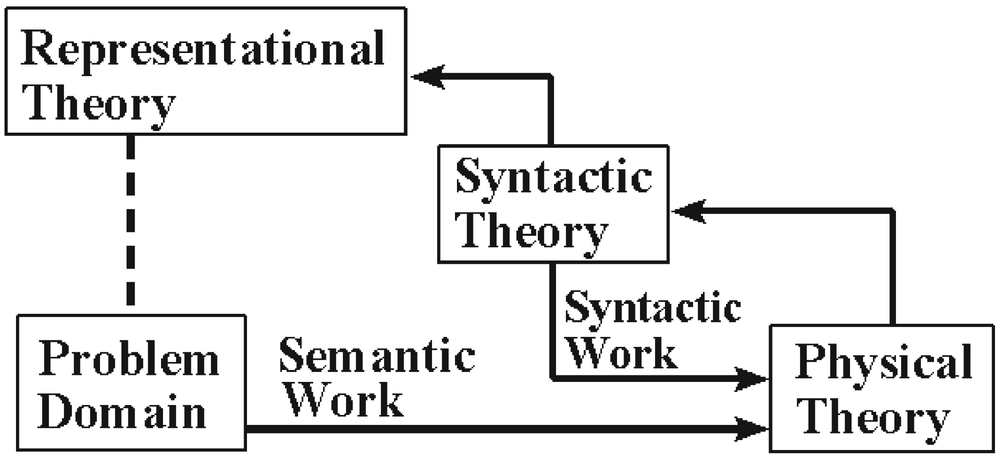


Figure 6

One has a general knowledge about ALAN's operation at the physical level. In virtue of the syntactic type-individuation one translates one's physical explanations into syntactic explanations that relate classes of inputs and outputs. This allows syntactic explanations to ride piggyback on physical theory. Hence, one can understand how physical principles can do syntactic work. In virtue of the assignment of representational content via the interpretation function, one translates syntactic explanations into explanations of how ALAN's syntactic states relate to positive integers within a certain range. The representation relation rides piggyback on syntactic theory, and serves to tie ALAN's states to objects in the problem domain so that one can explain ALAN's ability to compute the (x + 1) function. One then understands how physical principles do semantic work. That is, the representational theory relates the system's states to the objects of the problem domain so as to explain how the system's operations can potentially model the objects/properties and the relations relevant to the performance of a particular task.

The above explanatory schema is powerful, having fruitful applications throughout cognitive science. I have explicated the explanatory schema in terms of basic (naive) Turing computability, in accordance with common practice. However, the limitations placed upon the schema by literally conceiving of computability as Turing computability are too severe for it to serve as a the basis for explanations of cognitive capacities in cognitive science. The above explanatory schema works by asserting that cognizers, literally, are computers. Were one to adopt the pure Turing framework delineated above, this assertion amounts to an assertion that cognizers are Turing equivalent computers. However, Turing computability explicates computability only for the positive integers. Many functions considered within cognitive science are not mappable to the positive integers. Functions used to explain cognitive capacities such as vision in natural systems, as well as those used to explain connectionist systems (i.e., functions the systems hypothetically compute) are defined over the reals. Hence, many of the functions considered by cognitive scientists fall outside the Turing concept of "computable". In other words, one cannot use the unmodified notion of a Turing computer, nor of any of its equivalent versions (e.g., post systems or lambda systems) to explain (model) how these systems actually compute functions on the reals because the Church-Turing notion of computability, the model of how the system computes, does not apply to functions on the reals.[3]

Furthermore, Turing computation explicates computation in terms of five basic mechanical processes, scanning a square, writing a stroke, erasing a stroke, moving a square to the right, and moving a square to the left. One can rarely construe the systems studied by cognitive scientists as operating in this fashion, or in any Turing equivalent manner. That is, there is little or no neurological evidence to support the hypothesis that brains operate by performing read/write operations on syntactically structured strings. No one, for instance, supposes that neurons and neuronal activity implement recursive read/write operations over binary code. Rather, natural cognitive systems are best construed as more generally generating output from input in a manner best described by complex equations on the reals.[4]

Note just a few of the computations attributed to systems that are not explicable by the basic Turing notion of computation because they are defined over the reals: Brodie et al (1989) model the input-output of the Limulus retina in response to moving stimuli (through time in a single dimension) using Fourier transforms as follows (ξ is the spatial frequency and ω is the temporal frequency): (pp. 332-3)

$$\mathfrak{L}(x - vt) \rightarrow \mathbb{R}(x,t) = 1/2\pi \int e^{i(\xi x - \xi vt)} \mathscr{F}(\xi, -\xi v) \tilde{\mathfrak{L}}(\xi) d\xi$$

Similarly, edge detection is thought to occur in the early visual system by means of convolution of image intensity arrays (in the retinal ganglion and LGN), and is widely thought to be computed in the brain by taking the Laplacian (below) of Gausian filters of the retinal array: (Horn 1989, p.120)

$$\partial^2/\partial x^2 + \partial^2/\partial y^2$$

Marr and Poggio solve the binocular vision problem using the following difference equations: (1989, p.263)

$$C_{xyd}^{(n+1)} = \sigma \left\{ \sum_{x'y'd' \in S(xyd)} C_{x'y'd'}^{(n)} + C_{xyd}^{(0)} \right\}$$

One possible response to the reality of computational explanation in cognitive science falls back on the Church-Turing thesis and rejects the CTC, asserting that cognition is not the computation of functions over representational states, since computability is Turing computability. This response, however, is unacceptable. Though often portrayed as the definition of computability simpliciter, the Church-Turing thesis states only that the set of computable functions for the positive integers is identical to the set of Turing computable functions. It has long been known that systems intuitively accepted as computers compute functions other than the Turing computable functions.[5] Moreover, natural cognitive systems seem to be examples of such intuitively acceptable computers in that they are best understood as processing information. Rejecting the strictly Turing CTC requires both that one abandon a fruitful approach to cognition without a plausible replacement in hand and that one abandon one's intuitive notion of computation in favor of an unintuitive, but rigorous definition. Fortunately, other alternatives to rejecting the CTC exist.

2.) TURING-GRZEGORCZYK COMPUTABILITY

The first, seemingly least painful alternative to rejecting the CTC saves the CTC by extending the basic Turing notion of computability to functions on the reals and adopting a modified version of what is popularly portrayed as Church's Thesis. The original Turing definition of "effectively computable" has been extended, for example, by Grzegorczyk (1955, 1957), to functions on the reals as follows: One starts by extending the notion of computability to the notion of a computable sequence of rational numbers thus (Pour-El and Richards 1989, p.14):

> A sequence $\{r_k\}$ of rational numbers is computable if there exist three recursive [Turing computable] functions a,b,s from $\mathbb{N}$ to $\mathbb{N}$ such that $b(k) \neq 0$ for all k and $r_k = (-1)^{s(k)}a(k)/b(k)$ for all k.

A sequence of rational numbers is computable if one can code the numbers (e.g. by enumeration) and there are three Turing computable functions that one can use to compute each rational number from the coded representation for that number. For example, -.1 is simply -1 times 1 divided by ten, where $s(k) = 1$, $a(k) = 1$, and $b(k) = 10$. Using the notion of a computable sequence of rationals, one can define effective convergence of a sequence of rationals to a real number, and define a computable real in terms of the two concepts (Ibid. pp. 14 and 16):

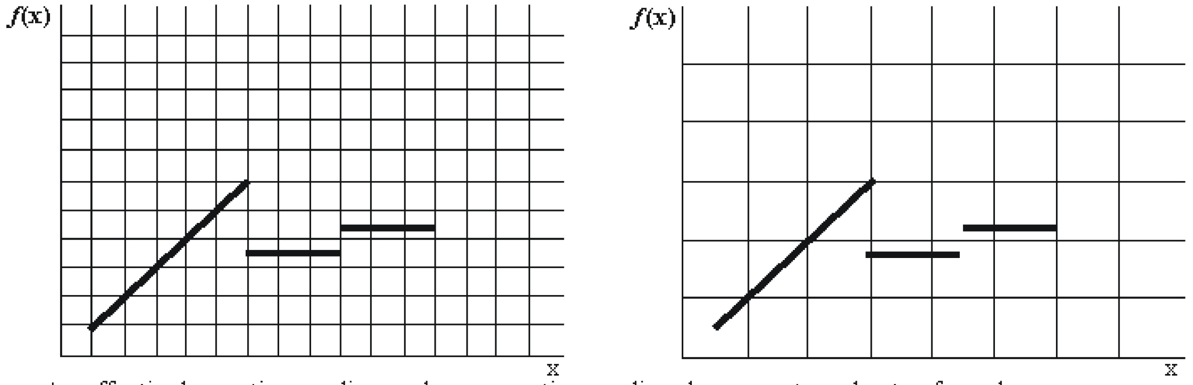
> A sequence $\{r_k\}$ of rational numbers converges effectively to a real number x [$r_k \rightarrow x$] if there exists a recursive function e: $\mathbb{N} \rightarrow \mathbb{N}$ such that for all N:
>
> $$k \geq e(N) \text{ implies } |r_k - x| \leq 2^{-N}.$$
>
> A real number x is computable if there exists a computable sequence $\{r_k\}$ of rationals which converges effectively to x.

On this notion, a real number is computable if there exists some sequence of rationals $\{r_k\}$ such that for any degree of approximation $d = \pm 2^{-N}$ to the real, there is a computable rational in the sequence $\{r_k\} = e(N)$ beyond which the rationals in $\{r_k\}$ are within d of the real. So, a real number is computable if a Cauchy sequence of rationals for that real is sequentially computable. For example, in base ten, the real number, $.a_1a_2a_3...$ is **T-G** computable if there exists a Turing machine such that the Turing machine starts with a blank tape and prints a tape of the form $...000Xa_1Xa_2Xa_3X...000...$ for any degree of approximation. (Minsky 1967, p.158) One can then generalize from the above definitions to definitions for a computable sequence of reals and finally a computable function on the reals (Ibid. pp. 18,25):

> A sequence of real numbers $\{x_n\}$ is computable (as a sequence) if there is a computable double sequence of rationals $\{r_{nk}\}$ such that $r_{nk} \rightarrow x_n$ as $k \rightarrow \infty$, effectively in k and n.
>
> A function $f: I^q \rightarrow \mathbb{R}$ is computable if:
> (i) f is sequentially computable, i.e. f maps every computable sequence of points $x_k \in I^q$ into a computable sequence $\{f(x_k)\}$ of real numbers;
> (ii) f is effectively uniformly continuous, i.e. there is a recursive function d: $\mathbb{N} \rightarrow \mathbb{N}$ such that for all x, y $\in I^q$ and all N:
>
> $$|x - y| \leq 1/d(N) \text{ implies } |f(x) - f(y)| \leq 2^{-N},$$
>
> where $|\ |$ denotes the euclidean norm, $I^q \subseteq \mathbb{R}^q$ such that $I^q = \{a_i \leq x_i \leq b_i, 1 \leq i \leq q\}$, and where

the end points a_i,b_i are computable reals.

A sequence of reals is computable if for each member of the sequence of reals the double sequence of rationals is such that it converges to a sequence of Cauchy sequences for the reals. The above definition of computability for the real functions states, in effect, that a function on the reals is computable over some interval from a sequence (I^q) of integer approximations if (1) for every computable sequence of reals in that interval that are arguments for the function, the sequence of corresponding values is computable for any degree of approximation, $a = \pm 2^{-N}$, (2) if there is a difference between elements of the domain less than the approximation, $\pm 1/d(\mathbf{N})$, then the difference between the values is less than the approximation $\pm 2^{-N}$ for any $\mathbf{N}$. Or more standardly, (2) there is a recursive function d on $\mathbb{N}$ such that for any $x \in I^q$ and any degree of approximation, $\delta = \pm 1/d(N)$, there exists an $\epsilon = \pm 2^{-N}$ such that the computed approximation of $f(x) = f(x) \pm 2^{-N}$. Intuitively, if one thinks of approximations as squares on a piece of graph paper (see figure 7), there has to be a relationship between the units of the x and the y axis such that no matter how large or how small the units for x become, the units for y are such that each segment of the line would fall in square adjacent to the previous segment and only one square is filled for each unit value for x.



An effectively continuous line and a noncontinuous line drawn on two sheets of graph paper representing different degrees of approximations. In the second drawing the noncontinuous line fails to travel through adjacent squares such that the line fills one and only one square for each unit value for x .

Figure 7

The first thing to note with regard to the extended notion of computability is that even the above, expanded definition of computability does not extend the Turing definition of computability to all reals or to all functions on the reals. The definition requires that the function be effectively uniformly continuous. Not all functions on the reals are effectively uniformly continuous, and, in fact, as Brodie et al. note in relation to the mathematical modeling of natural cognitive systems,

> The assumption of continuity.... ...is a mild assumption for most systems in the middle of their operating range, but it often does not hold for systems operating at the extremes of their range. (1989, p. 331)

An example from computational modeling where continuity assumptions might well be violated are threshold functions. Connectionist modelers often include threshold functions in their models. The threshold acts to prevent a given unit from generating an output until it receives the prerequisite activation from other units. To make the use of thresholds computationally tractable in connectionist systems modelers assume that thresholds have integer values. However, though this feature of models is commonly thought to correspond to one of the most basic mechanisms of neurological functioning (action potentials) the supposition of static integer values for thresholds is unrealistic neurologically. In neurons the polarization of post-synaptic membranes occurs as the result of membrane conduction triggered by activity at the synapse. The dendrites and the cell body of a neuron act to sum localized depolarization and hyperpolarization resulting from input to the neuron from other neurons at synapses. When the net change in polarization at the axion hillock (from molecular activity in the dendrites and cell body) is 10mv an action potential or spike occurs. Such changes in the polarization in the membrane result from the subtle dynamic interaction of (at least) neuronal synaptic input, summation both within dendrites and between dentrities, and cellular function and are neither static nor are they plausibly modeled using integer values. [See for example (Polsky, Mel, and Schiller 2004)]

However, if one extends again the notion of a computable function in a manner consistent with Grzegorczyk computability, step functions with computable jump points and values are computable under the new definition, though the computational process will result in dramatically incorrect values part of the time.[6] First order ordinary differential equations are **T-G** computable if the functions are twice continuously differentiable, but **T-G** noncomputable for twice differentiable functions that are not continuously differentiable.[7] One can also show that higher order ordinary differential equations are not sequentially computable unless they can be reduced to a series of first order equations.[8] So, the **T-G** notion of computability does not apply to a large class of functions. In fact, for reasons of noncomputability and of tractability, many computational models in cognitive science are based on approximations of functions attributed to natural systems in theoretic descriptions of those systems. The Laplacan function given earlier is approximated in computational models of edge detection. As late as 1984, the state of the art computation of the approximation of the Laplacian function for a mere 1,000 pixel picture was one second, a full order of magnitude greater than 100 to 300ms it takes the visual system to recognize objects using retinal arrays having literally hundreds of millions of pixels. Watt and Morgan (1984) have gathered evidence that in the brain the Laplacan function is applied to the second derivative of a Gaussian filter of the retinal image. In describing their work, Marr and Poggio tell readers that

> This algorithm can be realized by various mechanisms, but parallel, recurrent, nonlinear interactions, both excitatory and inhibitory, seem most natural. The difference equations set out above would then represent an approximation [my emphasis] to the differential equations that describe the dynamics of the network. (1989, p. 264-5)

So, some of the functions which have been utilized in computational models in cognitive science are approximations to the noncomputable or intractable functions which seem to best describe the input/output of the real cognitive systems studied. This suggests that **T-G** computability, like Church-Turing computability, though adequate to create models of some natural cognitive processes, is an in principle inapplicable theory of those processes.[9]

Likewise, **T-G** computability relies upon recursive read/write operations defined over syntactically structured strings, making it as implausible a physical theory of natural cognizers as the Church-Turing notion. To date, no model of neuronal networks, even recurrent networks said to be able to "implement" Turing machines, actually operate by finite read/write rules over syntactically structured strings. Moreover, while the original

Turing definition stated, in effect, that there existed a finite set of instructions such that given any input for the function of finite length, the system could generate the finite output in finite time. The new definition says in effect that there exists a finite set of instructions such that for any given input of finite length, the system can generate a finite approximation of the output in finite time. Thus, computability for functions goes from generating the output to generating any arbitrarily close approximation of the output. For example, when one presses the "π" function key of one's calculator, it will not display a decimal representation of π. Rather, one's calculator computes a decimal representation of π to some approximation (in my case to the billionth). This last fact, remains true no matter how large and expensive one's calculator. I do not here underplay the monumental intellectual accomplishments of Turing and his successors. Rather, I wish only to highlight the distinction between computation by successive approximation on the rationals and computation by instantiation. The former is well-understood in relation to the reals, the latter is too often ignored. To compute by instantiation is to compute by having states that are type-individuable so that the I/O relation maps exactly the target function. So, ALAN computes the successor function for its subset of the integers by instantiation because, in contrast to one's calculator and π, ALAN starts with an exact representation of the predecessor and computes an exact representation of the successor. Two strokes for two becomes three strokes for three, and so on. Natural cognitive systems are best construed as computing functions on the reals by instantiation, rather than by successive approximation. As a result, one sees another reason to suppose that the **T-G** model of computation is an implausible model of computation in natural cognizers.

Even though the natural systems studied by cognitive science compute functions on the reals by instantiation, it often seems that too few philosophers and cognitive scientists take this fact to heart. A clear example of this trend is the debate over connectionism. One finds, on the one hand, Smolensky (1988) correctly noting that connectionist models operate by instantiating differentiable equations in distributing activation between units (and hence, are simulated on digital computers by successive approximations on the rationals and discretization). One then finds Fodor and Pylyshyn (1988) and Fodor and McLaughlin (1991) claiming that "...if its [connectionism's] account of systematicity requires mental processes that are sensitive to the constituent structure of mental representations, then the theory of cognition it [connectionism] offers will be, at best, an implementation architecture for a "classical" (language of thought) model." (1991, p.331) The classical model presumably includes both the notion of a "language of thought" and the Turing or the Turing-Grzegorczyk notion of effective computability. Indeed, Fodor and Pylyshyn are explicit about this, saying

> It would not be unreasonable to describe Classical Cognitive Science as an extended attempt to apply the methods of proof theory to the modeling of thought.... (Fodor and Pylyshyn 1988, pp.29.30)

The latter, Turing-Grzegorczyk component of the classical view, of course, will not necessarily be satisfied by the hypothesized Smolensky machine, making Fodor et al's claim false. That is, the system may be sensitive to constituent structure of mental representations in the sense that it responds to inputs with different constituent structure differently without employing the mechanism of recursive read/write operations on those strings. Fodor et al., ironically, seem to note this mistake when they offer the further "criticism" that the hypothesized Smolensky system architecture would still compute by association. (1991, p.349 and 1988, p.67) Cummins' writings on connectionism, though admirably informed about computability issues, are potentially misleading in that they do not sufficiently differentiate Turing computability from another notion of computability: In "Connectionism, Computation, and Cognition" Cummins and Schwarz note (1991, p.68) that “any system that is cognitive in virtue of its representational state changes, but effects those state changes non-computationally is a counter-example to computationalism.” They then note that:

> This, of course, is more than a mere possibility. A [connectionist] network whose representational states are real-valued activation vectors and weight matrices, and whose dynamics is given by a set of differential equations, is in general, going to be just such a system. (Ibid.)

In his 1991 "Representation in Connectionism," Cummins claims that connectionist systems are correctly described as executing "...computation of representations via spreading activation as determined by connection weights." (p.97) So, Cummins appears to be asserting that connectionist systems are computers and are not computers. Cummins, of course, is not contradicting himself. He has two different notions of computing in mind.

A proper understanding of the Church-Turing thesis and its implications for computation and computability would clear away much of the seeming confusion in the above-mentioned discussions. Fortunately, John Earman (1986) has done much of this work [see also Copeland (1996)]. Earman distinguishes three senses of the Church-Turing Thesis: (1986, p.125)

(CP1) The class of programmable or algorithmically computable functions of the integers is to be identified with the Turing computable functions.

(CP2) The class of programmable or algorithmically computable functions of the reals is to be identified with the Grzegorczyk computable functions.

(CP3) The class of effectively, mechanically computable functions is to be identified with the class of programmable or algorithmically computable functions and, thus, with the Turing-Grzegorczyk computable functions.

As Earman indicates, (CP1) is the most consistent with the work of Church and Turing and the problem to which it was addressed. (CP2) represents the state of the art in digital computation, but it is actually rather implausible since Grzegorczyk computability is merely one of many nonequivalent extensions of Turing computability to functions on the reals. (Earman 1986, pp.126-7) (CP2) is generally held to be more plausible than it is because people often conflate (CP1) with (CP3) [even, perhaps, Turing at times] from which (CP2) follows. Such a conflation allows one to falsely suppose that no other physical systems besides systems literally behaving like Turing Machines (i.e., computing by following a finite set of finite rules for manipulating syntactically structured representations) satisfy our intuitive notion of computation. Hence, Fodor/McLaughlin falsely suppose that a connectionist system computing a function that is (they hypothesize) a Turing-Grzegorczyk computable function must prove a mere implementation of a Turing-Grzegorczyk architecture. However, as eluded to above, systems that seem to compute, but which do not instantiate Turing-Grzegorczyk architectures abound, e.g. diferential analyzers and, I would argue, natural cognitive systems. If one conflates computability (what functions can be computed by some architecture) with computation (what processes can be harnessed to compute) then one will likely think of (CP3) as a well-supported thesis, and infer that all computers are Turing-Grzegorczyk computers.

4.) COMPUTABILITY VS COMPUTATION AND MONTAGUE

One might wonder what I mean when I claim that people conflate computability with computation. One defines computability by reference to an explicit statement about what counts as a computation. Indeed, one of Turing's great contributions to the field is to make the notion of an effective procedure explicit and to use that notion to define computation. Once one has such a definition of computation, then one can define computability relative to one's notion of computation by asserting that to be computable is to be a function that can be mapped by a computation. However, computability is a relative term. It is a false conflation to suppose that just because a function is computable relative to some definition that all systems that compute that function must be computers under that definition. Indeed, many of the same functions are computable under the several nonequivalent definitions of what counts as a computation that appear in this paper.

Recall that Turing computability specifies computability in terms of what functions can be isomorphically mapped to I/O patterns (via an interpretation function) mediated by a finite series of five simple processes; moving left, moving right, scanning a square, and writing a stroke in a square, and erasing a stroke from a square. One way to generalize Turing computability is take advantage of Turing computable functions to compute cleverly coded functions in other domains–hence, the application of Turing computability to include sequences of rationals. Another strategy loosens the definition of a mapping of the function to the I/O pattern–hence, the extension of Turing computability to the reals via successive approximation and approximation to the norm (see note 3). Yet another proposed extension would allow supertasking. Boolos and Jeffery (1989), Grunbaum (1967) Earman and Norton (1993), and Hogarth (1994) elaborate such machines. There remains, however, a little explored possibility for extending the notion of computability;

extend the set of possible physical processes capable of underwriting computation. The idea here is that one should extend the notion of an operation, not merely beyond the basic Turing five, but to operations not necessarily characterizable by a finite set of rules for manipulating syntactically structured representations. If one rejects (CP3), and allows the mediation of I/O patterns by other sets of operations (e.g. physical laws), one can extend the notion of computation and computability in a direction that is both intuitively satisfying and capable of underwriting the CTC. The source of this inspiration is Montague (1962). In effect, Montague sought to define a generic notion of computation. He required only that the system have some aspect which signaled starting and stopping, that some aspect of the system that represented the input and output, that the laws of system were such that the system treated type-identical inputs identically (that it operates on tokens according to their type), and that all I/O's mapped isomorphically to the function.

To get a clearer idea of Montague computation, consider a toy Montague computer (figure 8):

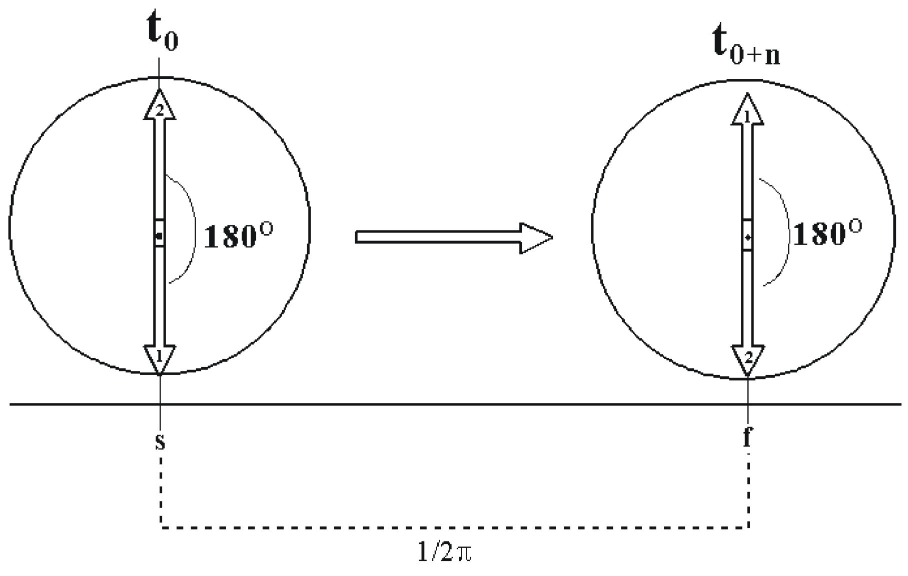


Figure 8

On the assumption that the wheel in figure 8 is perfectly round and the radius is exactly one-half of a unit, the above system might be used to compute πx, where $x \in \mathbb{R}$ and $0 \leq x \leq 1$. The start state for the system consists of the wheel's start mark placed upon the start mark on the floor, the number one arrow aligned with the start mark, and the number two arrow at the angle corresponding to the real number ($\triangle/360$) value for x. One then rolls the wheel until its the start mark reaches arrow two, at which time, the distance rolled equals πx. It is obvious that the above system has a starting and stopping point, and that the I/O relation maps onto the target function.

Once one understands the general notion of Montague computation, one can make the notion rigorous using Montague's definitions. Montague computation defines a computer as a set <i,j,k,l,a,b>, relative to a set of possible histories, **H**. **H** satisfies the follow three conditions (Montague 1962, pp.121-122):

(1) **H** is the set of possible world histories.
(2) All elements of **H** have a common domain, which Montague called the index set of **H** ($\mathbf{W}_H$). That is, the aspects of the world that remain constant from one history to the next, but are consistent with possible variations in their behavior from one history to the next.
(3) **H** is closed under temporal translation.

Montague specifies **H** using a set of laws constituting a theoretical model of the system's operation.[10] Hence, I call the set <**H**,i,j,k,l,a,b> an architecture specification. i,j,k,l are aspects of the world, $\mathbf{W}_H$, with i and j input, and k and l outputs. i and k serve respectively as I/O signals indicating the commencement and completion of computation, where the user designates the value a for i as indicating commencement and b for k indicating completion. j and l are respectively the assigned to the argument and value. One can then define a computer, a computed value for an argument, and a computable function as follows:

(Montague 1962, pp.122-4)

If **A** is any set of real numbers, then t is a minimal element ($\in_m$) of **A** if $t \in$ **A** and for any real number, t', $(t > t') \rightarrow (t' \notin$ **A**).

A computer ($\mathbf{C}_H$) is an architecture specification <**H**,i,j,k,l,a,b> such that $i,j,k,l \in \mathbf{W}_H$, and, for all **D**,**E**,t_0,t_1 if
(1) **D**,**E** $\in$ **H**,
(2) $t_0 \in \mathbb{R}$,
(3) $\mathbf{D}_i(t_0) = \mathbf{E}_i(t_0) = a$,
(4) $\mathbf{D}_j(t_0) = \mathbf{E}_j(t_0)$
(5) $t_1 \in_m$ of the set of reals $t \geq t_0$ for which $\mathbf{D}_k(t) = b$,
then
(6) there exists a real number t_2 such that $t_2 \in_m$ of the set of reals $t \geq t_0$ for which $\mathbf{E}_k(t_2) = b$, and
(7) $\mathbf{E}_l(t_2) = \mathbf{D}_l(t_1)$.

The above definition of a computer guarantees that, given the set of possible histories, **H**, defined by the laws that specify the theoretical model of the system, the system maps type-identical inputs to type-identical outputs. In other words, the system operates on tokens according to their type. One can then define the value computed by a system for a given input and whether the system computes a function as follows:

Let $\mathbf{C}_H$ be a computer with an architecture specification <**H**,i,j,k,l,a,b>. y is a value computed for the argument x by $\mathbf{C}_H$ [symbolized $\mathbf{Val}_H$<y,x,$\mathbf{C}_H$>]if
(1) **D** $\in$ **H**,
(2) $t_0, t_1 \in \mathbb{R}$,
(3) $\mathbf{D}_i(t_0) = a$,
(4) $\mathbf{D}_j(t_0) = x$,
(5) $t_1 \in_m$ of the set of reals $t \geq t_0$ for which $\mathbf{D}_k = b$, and
(7) $\mathbf{D}_l(t_1) = y$.

Call the function computed by $\mathbf{C}_H$, $f_{C,H}$. $f_{C,H}$ is the set of ordered pairs <y,x,> such that $\mathbf{Val}_H$<y,x,$\mathbf{C}_H$>. A function f is computable$_H$ iff there exists a computer $\mathbf{C}_H$ such that $f = f_{C,H}$.

Hence, a function is computable relative to an architecture if the representation function isomorphically maps the I/O states of the system to the domain and range of the function ($f = f_{C,H}$).

As the above definitions indicate, adopting a Montague approach to computation requires one change somewhat the questions one asks with regard to computability. First, computability is defined relative to an architecture specification. One does not simply ask if some function *f* is computable. Rather, one asks if the function is $\mathbf{C}_H$-computable for some architecture specification <**H**,i,j,k,l,a,b>. Second, computability is no longer necessarily tied to syntactically structured strings of symbols, but can also be defined as a relationship between real valued quantities related by laws, as is the case with the above πx computer.

One can illustrate the versatility of the above definition by considering how one would show ALAN and a standard three layer connectionist system to be Montague computers. One can specify ALAN's architecture as follows: **H** is the set of possible tape configurations ALAN can realize in state one and the corresponding resultant state and configuration combinations determined by the "laws" of the state transition diagram. ALAN's i is ALAN in $\mathbf{S}_1$ at the beginning of a string. ALAN's j is the number of strokes in the string. ALAN's k is ALAN's coming to rest at the beginning of a string of strokes in $\mathbf{S}_3$. Finally, ALAN's l is the number of strokes in the $\mathbf{S}_3$ string. It is straightforward to show from this how ALAN satisfies the above definition of a computer and that the successor function is computable$_{ALAN}$; one merely calls upon the standard proof that ALAN computes (x + 1).

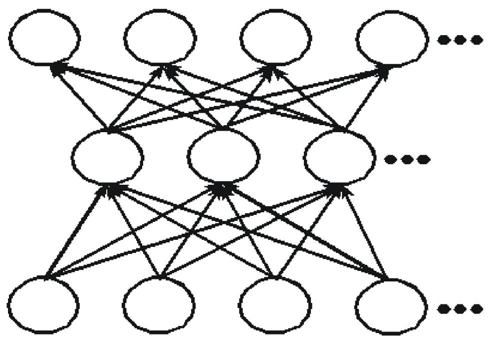


Figure 9

For the above schematized connectionist system, i equals the initial activation of the input layer. The activation values for the individual input nodes correspond to j. k would then be the system once it has settled into stable activation levels. Finally, l corresponds to the activation values for the output array. **H** is defined in terms of the individual system's weights, thresholds, and the standard mathematical apparatus for the spread of activation to units. The proof that an individual system is a computer would involve showing that the laws of the system's operation are such that it would settle identically for identical activation patterns within the target domain. As I indicate later, connectionists already have a theoretical proof that there exists a three level connectionist system that satisfies the above definition of a computer for every continuous function $f\colon [0,1]^n \rightarrow \mathbb{R}^m$.

5.) OBJECTIONS

Montague computation is an internally consistent generalization of Turing framework for understanding computability. It is capable of underwriting current otherwise noncomputational "computational" explanations of cognitive capacities. But, there are, at least, five objections to adopting Montague computation as the definitive notion of computation for Cognitive Science.

First, one might note that according to the Montague definitions the partial recursive functions on the reals are not computable by Turing computers. Surely, one might argue, a definition with such a consequence proves as dismal a failure as the T-G notions I criticize at capturing important intuitions about what functions are computable. Since the Montague definition of computing a function as presented here requires an exact mapping of the system's I/Os to the elements of the domain and range of a target function (i.e., $f = f_{C,H}$), it is strictly true that the partial recursive functions on the reals are not computable under the

Montague definitions. However, I would argue that this point is not problematic. Indeed, one might adopt either of two positions in response to such claims.

Recall, that T-G definitions of computability expand the notion of computable so that systems that operate so as to map discrete approximations of the elements of a given real function's domain to corresponding discrete approximations of the elements of that real function's range compute real functions under the definition. A system computes a function on the reals according to this approach in so far as, for any degree of approximation, the system's operations enforce the appropriate mapping.

One line of response to the first criticism is to note that while it follows from the above Montague definitions that the partial recursive functions are not computable on the reals by Turing machines, it does not follow that there are facts about the abilities of Turing machines that cannot be established in the Montague framework. In fact, it is straightforward to establish that for any function shown to be computable by Turing machines under the T-G definitions, Turing machines can compute the discrete approximation of that function for any degree of approximation. The difference, then, between frameworks is how they classify the same Turing machine abilities. The T-G definitions classify the ability to map discrete approximations of the domain of some real function to discrete approximations of the range of that function (for any degree of approximation) as computing the function on the reals. Within the Montague framework, the same mapping abilities are classified as computing discrete approximations of those functions on the reals (for any degree of approximation).

A second line of response would simply note that one can modify the Montague definition of computing a function in the same manner one modified the original Turing notion, i.e, to include the successive approximation and approximation to the norm. Such a modification would make the desired computability results possible without compromising the advantages of the Montague approach. Either of these lines of response strike me as completely adequate to defuse the first objection.

Second, critics might claim that no decision procedures exist for deciding whether a function is Montague computable relative to some architecture specification or class of such systems. This objection is made by Fodor and Pylyshyn with regard to connectionist systems as follows:

> One of the attractions of Connectionism for many people is that it does employ some heavy mathematical machinery,.... But in contrast to many other mathematically sophisticated areas of cognitive science,..., the mathematics has not been used to map out the limits of what the proposed class of mechanisms can do. (Fodor and Pylyshyn 1988, p.6)

The mere lack of current results, were it true, is not to my mind an objection. To start, it is unfair to require results in an area of science or mathematics prior to work in the area. No one to my knowledge has offered a proof that such results are impossible. In fact, some results are already available. Pour-El (1974) has presented a notion of analogue computability. The most famous, though not the most directly useful, result from connectionism is probably **Kolmogorov's Mapping Neural Network Existence Theorem** adapted from Kolmogorov's work in 1957:

> Given any continuous function f: $[0,1]^n \rightarrow \mathbb{R}^m$, $f(x) = y$, f can be implemented exactly by a three-layer feedforward neural network having n fanout processing elements in the first (x - input) layer, and m processing elements in the top (y - output) layer. (Hecht-Nielsen 1991, p.122)

Minksy and Papert (1969) have provided a highly influential proof of the limitations of Perceptron architechtures. Moreover, cognitive science, in large part, studies existing cognitive systems. For this reason, much of a Montague-CTC cognitive science would remain unscathed if it turned out that there were no general decision procedures for important classes of Montague computers, since by assumption of the CTC a system exists such that it $\mathbf{C}_H$-computes the relevant function.

A second objection to the adoption of Montague computation is that everything turns out to be a computer. This might tempt one to believe one of the following two propositions: (1) Adopting Montague computation would result in everything's computing every function. Hence, CTC dies the death of the plentiful but trivial result. (2) Everything is computer, therefore, by the CTC, everything is a cognizer. There are viable responses to both versions of the objection.

Since a system is a Montague computer relative to its laws of I/O transformation, it is not the case that every system computes every function. Substantive work in the theory of computation would show what functions are computable by what systems. Such results can prove illuminating and useful to cognitive science. If, for example, it turns out that a cognitive function is not computable over some subset of the domain (e.g., the system computes a function that is only partially isomorphic to the target function), this will provide an explanation of cognitive fallibility. Such a case of partial isomorphism occurs in vision: To overcome noise inherent in the system, the visual system pools the input from several rods or cones, in effect, taking the average. This process is only partially isomorphic, and causes relative brightness illusions, called assimilation effects, in cases where dark and light fields are closely interspersed. (Goldstein, 1989 p.153)

Searle (1980) famously offers the second version of the objection to the Turing version of the CTC. Critics who reject Searle in the case of Turing computability can continue to take solace in the, I think, viable positions that emerge in the discussion resulting from Searle's work. Each position notes that the CTC is not the sole element of computational explanations in cognitive science. First, researchers standardly employ theories of mental representation offered as part of the RTI (representational theory of intentionality), not computational ability, to delimit the mental and nonmental. A system must not only compute, it must compute over mental representations. A second, in my opinion better, route is to note that, as discussed above, computational ability and the exploitation of computational ability to perform cognitive tasks are conceptually distinct, and more often than not, physically distinct. Those pursuing this line hold that cognitive systems must be able to exploit their computational abilities in a tractable manner in the real world domain. Hence, cognition, unlike computation, has distinct computational, representational, and epistemic components. While all physical systems compute, not all systems can exploit their computational abilities to perform cognitive tasks. (See Wallis, 1994)

One might claim that my position with regard to Montague computation tries to make a virtue out of a vice. That is, even if one can define computation in such a way that systems compute by instantiation on the reals, it does not follow that one can recognize or use such computers, unless these systems have discrete, finite representations as inputs and outputs. A system that computes a function on the reals will only be as useful as our interpretation. With the πx computer, for instance, one can only use the system to compute the function to the degree of accuracy with which one can manipulate the wheel, measure the angle of the arrows, and measure the length of the roll. In other words, one can only exploit such systems by approximate, digital interpretations of their I/O, so what is the net gain?

There are two aspects to the above objection which one must separate. The first aspect of the above objection has to do with the value of a more general notion of computability given that many of the functions under consideration are computable on either definition. The relevant point here is that cognitive science does not merely wish to understand how digital computers could compute cognitive functions. Cognitive science seeks to explain how natural cognizers compute cognitive functions. It is not enough to understand how a digital computer would compute the function, if the natural cognitive system does not compute the function in this manner. The **T-G** notion of computability does not help one understand how natural cognizers could be computers. Hence, one needs the more general notion of computability.

The second aspect is the issue of the ability to exploit an interpretation to do computational work. One should start by noting that the problem of formulating and exploiting an interpretation is a separate issue from computability. Computation has and can be defined abstractly, without reference to issues of epistemic access. So, a notion of computability need not suffer from worries about one's ability to recognize and exploit the computational abilities of such systems. One should also note that the epistemic problem is not unique to Montague computation. One can, for example, understand the extension of computability to sequences of rational numbers as a discovery about how to exploit an interpretation to do computational work.

While epistemic issues raise problems from a design and discovery standpoint, a system will exploit computational abilities, not in virtue of recognizing interpretations, but by being hooked up to the world in the relevant way, and having operations that are sensitive to representational differences. The important issue is not merely what the researcher knows and can do, but what the system itself does. In short, a system that computes motor response commands from image array intensity information need only perform the computations and have motor responses that are causally sensitive to the real differences in representational output. Researchers can, of course, have evidence (through approximate measurements) that this last property is true of the system, even when they cannot (even need not) specify an interpretation exactly.

Finally, one need note that only part of my arguments presuppose adopting a theoretical framework for computation defined over the reals. Even were one to claim that biological systems are finite instantiations of computational systems defined over the reals, it does not follow that one need not adopt a Montague approach in cognitive science. It is false to suppose, as classicalists almost universally do, that if one can truncate the range and domain of a function that it automatically becomes Turing computable and that the Turing implementation of such a computer is biologically feasible. For instance, Turing machines with finite tapes (i.e., finite representational capacities) cannot even compute all of the computable functions on the integers. Likewise, theoretical Turing computability does nothing to establish biological plausibility.[11]

The final objection I wish to consider asserts that the adoption of Montague computation would undermine the important and standardly accepted reaction time paradigm for determining cognitive architecture. Since, the objection runs, Montague computation allows computation that is not step-wise, comparisons of relative time to complete a task will not differentiate competing computational explanations.

One can undermine the above claim by the following considerations. First, regarding the choice of theories having distinct architectures: Given two different architecture specifications, $\mathbf{S}_1$ and $\mathbf{S}_2$, it is quite possible that the t_1 predicted by each architecture will, in fact, differ. Such is the case with human (massively parallel) object recognition and digital computer simulations, both of which are Montague computers. In fact, such across-architecture comparisons of reaction time have played an important role in the connectionist-classicist debate (Feldman, 1985). Furthermore, since most theories of natural cognitive systems differ in the number of functional components proposed (i.e., the number of boxes), and not in the implementing mechanisms (neurons), the likelihood of differing reaction times in a Montague-CTC remains high. One should also note that Turing-Grzegorczyk computers having different architectures/algorithms may also have identical reaction times, making Turing-Grzegorczyk computing subject to the same objection. These last considerations underline the fact that reaction time is only evidence for, not proof of, a hypothesized architecture/algorithm.

Second, it is presumed that there will be provable results, like Kolmogorov's theorem, as to what functions are computable by classes of architectures which will rule out large numbers of functions for various architecture specifications. In other words, competing explanations must posit architecture specifications which can compute the hypothesized function and which are plausibly instantiated by the system before reaction times become relevant to distinguishing between them.

Finally many reaction-time experiments test the role of a posited subsystem in the performance of a cognitive task by placing increased computational load upon that subsystem during the performance of the task. Other reaction time experiments suppose that increased computational complexity due to, for instance, degraded information results in increased reaction times. Nothing about Montague computation alters the basic assumption that increased computational complexity or diminished computational resources often results in increased computation time. Hence, none of these common uses of reaction time experiments are affected by the adoption of a Montague-CTC. For example, by showing that reaction time increased linearly with angle of rotation (i.e., reaction times varied with task complexity) Shepard and Metzler (1971) used reaction time experiments to provide evidence that comparisons of mental images was done by rotation.

5.) CONCLUSION

The many successful computational explanations in cognitive science attribute computational abilities to systems that make the Turing-Grzegorczyk definition of computation implausible. These explanations have greater plausibility as accounts of the actual functioning of natural cognitive systems. One is, therefore, faced with the dilemma of holding steadfastly to a rigid interpretation of the Church-Turing thesis and rejecting CTC or expanding the notion of computation. The adoption of Montague computation by cognitive science painlessly preserves the insights of computational studies of cognition. Hence, cognitive science should adopt Montague computation as the primary model of computation for the CTC.

NOTES

1.) This terminology originates, I believe, with Cummins (1989). My usage is consistent with his.

2.) I have argued elsewhere Wallis 1994) that such an explanation schema while substantially correct, mistakenly ignores the epistemic aspects of cognitive explanations. Put simply, my claim is that explanations of cognitive abilities are explanations of a systems ability to compute **and** to exploit that computing ability to acta as epistemic engines.

3.) I discuss the proper construal of Church's thesis latter in this paper.

4.) Fodor and Pylyshyn (1988) might take exception to this claim. However, even they can argue only that the hypothesis that brains are digital computers in the classic sense has not been proven impossible. They cite no actual theory of how the brain could be a digital computer, but merely note that one might compensate for the brain's relatively slow speed by parallel processing in the classic sense. For a summary of the most often noted reasons to suppose brains are not Church-Turing type machines see Churchland and Sejnowski 1990.

5.) See Copeland and Sylvan (1997) for a survey of notational machines that calculate functions that are not Turing-Machine-Computable. See Copeland (1996) for a nice discussion of current misconceptions of the Church-Turing thesis. Machines that compute Turing non-computable functions using non-Turing mechanisms go aback at least to Lord Kelvins use of mechanical integrators to create a tide prediction machine in 1876 a before.

6.) Pour-El and Richards (1989) start with the definition "$L^p[a,b]$: for $1 \leq p < \infty$, the space of all measurable functions f on [a,b] for which the L^p norm $\|f\|_p = [\int_b^a |f(x)|^p \, dx]^{1/p}$ is finite." (p.8) They then define intrinsic L^p-computability as follows: "A function $f \in L^p[a,b]$ is L^p-computable if there exists a sequence $\{g_k\}$ of continuous functions which is computable [in the Grzegorczyk sense] and such that the L^p-norms $\|g_k - f\|_p$ converge to zero effectively as $k \rightarrow \infty$." (p.84) This extension again loosens the criteria for and interpretation. The definition says, in effect, that a function is intrinsic L^p-computable so long as one can compute approximations of the jump points and then compute approximations of the values for the function at the jump points. The result is that, for any degree of approximation, some values for the function will be dramatically wrong.

7.) This has the odd consequence that "...the classical theorem that a continuous function on a compact set is uniformly continuous does not effectivize." (Pour-El and Richards 1989, p.50)

8.) See Earman 1986, p.118 and Pour-El and Richards 1989, pp.55-57.

9.) Many functions in cognitive science are **T-G** intractable in real time. One response gaining momentum is the development of models using analog processing arrays. See

Sivilotti et al (1987) for a well-developed example.

10.) To specify the class of physically possible histories, H, Montague uses the notion of models of a scientific theory. Working within first-order predicate calculus with identity, definite descriptions, n-place predicates, and operations, Montague understands a scientific theory as the pair (T, U), where T is a set of formulas which can be regarded as the axioms of the theory and U a set of observationally meaningful terms of the theory containing exactly one free variable (for simplicity) referring to time. A model of the theory is a model in which all members of T are true. A physically possible history relative to (T,U) is a function, D, with domain U such that there is a standard model M of (T,U) such that D restricts all the functions in M defined by the meaningful expressions in U to the set of real numbers.

11.) Since the acceptance of this paper by Philosophy of Science, I have since elaborated on this point in "Computation and Cognition".

BIBLIOGRAPHY

Boolos, G.S. and Jeffrey, R.C. (1989). Computability and Logic. Combridge: Cambridge University Press.

Brodie, S., Knight, B., and Ratliff, F. (1988). "The Response of the Limulus Retina to Moving Stimuli: A Prediction by Fourier Synthesis," in J. Anderson and E. Rosenfeld (eds.) Neurocomputing: Foundations and Research. Cambridge: MIT Press.

Churchland, P. and Sejnowski, T. (1990). "Neural Representation and Neural Computation,"
in J. Tomberlin (ed.) Philosophical Perspectives. 4:343-82.

Copeland, B.J. (1996). “The Church-Turing Thesis,” in The Stanford Encyclopedia of Philosophy http://www/plato.stanford.edu/entries/church-turing/

Copeland, B.J., Sylvan, R. (1997). “Computability: A Heretical Approach,” Forthcoming.

Cummins, R. (1989). Meaning and Mental Representation. Cambridge: MIT Press.

Cummins, R. and Georg Schwarz (1991). "Connectionism, Computation, and Cognition," in T. Horgan and J. Tienson (eds.) Connectionism and the Philosophy of Mind. Dordrecht: Kluwer Academic Publishers.

Cummins, R. (1991a). "The Role of Representation in Connectionist Explanations of Cognitive Capacities," in W. Ramsey et. al. (eds.) Philosophy and Connectionist Theory. Hillsdale: Lawrence Earlbaum Associates.

Earman, J. (1986). A Primer on Determinism. Dordrecht: D. Reidel Publishing Company.

Earman, J. and Norton, J. (1993). "Forever is a Day: Supertasks in Pitowsky and Malamet-Hogarth Spacetimes," in Philosophy of Science 5:22-42.

Feldman, J. (1985). “Connectionist Models and Their Applications: Introduction,” in Cognitive Science 9: 1-2.

Fodor, J. and Pylyshyn, Z. (1988). "Connectionism and Cognitive Architecture: A Critical Analysis," in Cognition. 28:3-71.

Fodor, J. and McLaughlin, B. (1991). "Connectionism and the Problem of Systematicity: Why Smolensky's Solution Doesn't Work," in T. Horgan and J. Tienson (eds.) Connectionism and the Philosophy of Mind. Dordrecht: Kluwer Academic Publishers.

Grunbaum, A. (1967). Modern Science and Zeno's Paradox. London: George Allen and Unwin.

Grzegorczyk, A. (1955). "Computable Functionals," in Fundamenta Mathematica. 42:168-202.

Grzegorczyk, A. (1957). On the Definitions of Computable Real Continuous Functions," in Fundamenta Mathematica. 44:61-71.

Haugeland, J. (1981). "Semantic Engines," in Mind Design. Cambridge: MIT Press.

Hecht-Nielsen, R. (1991). Neurocomputing. Reading: Addison-Wesley Publishing Company.

Hogarth, M. (1994). "Non-Turing Computers and Non-Turing Computability," in PSA 1994,
Volume I, pp. 126-138.

Horn, B. (1986). Robot Vision. New York: McGraw-Hill Book Company.

Marr, D. and Poggio, T. (1989). "Cooperative Computation of Stereo Disparity," in J. Anderson and E. Rosenfeld (eds.) Neurocomputing: Foundations and Research. Cambridge: MIT Press.

Marr, D. (1981). Vision. New York: W. H. Freeman and Company.

Minsky, M. (1967). Computation: Finite and Infinite Machines. Englewood Cliffs: Prentice Hall.

Minsky, M. and Papert, S. (1969). Perceptrons. Cambridge: MIT Press.

Montague, R. (1962). "Towards a General Theory of Computability," in B. Kazemier and D. Vuysje (eds.) Logic and Language. Dordrecht: D. Reidel Publishing Company.

Polsky, A, Mel, B., and Schiller, J. (2004). "Computational Subunits in Thin Dendrites of Pyramidal Cells," in *Nature Neuroscience*. 7:621-627.

Pour-El, M. (1974). "Abstract Computability and Its Relation to the General Purpose Analogue Computer," in Transactions of the American Mathematical Society. 199:1-28.

Pour-El, M. and Richards, J. (1989). Computability in Analysis and Physics. New York: Springer-Verlag.

Searle, J. (1980). "Minds, Brians, and Programs," in Brain and Behavioral Sciences. 3:417-457.

Smolensky, P. (1988). "On the Proper Treatment of Connectionism," in Brain and Behavioral Sciences. 11:1-74.

Shepard, R. and Metzler, J. (1971). “Mental Rotation of Three-dimensional objects,” in Science. 171:701-703.

Sivilotti, M., Mahowald, M., and Mead, C. (1989). "Real-Time Visual Computations Using
Analog CMOS Processing Arrays," in J. Anderson and E. Rosenfeld (eds.) Neurocomputing: Foundations and Research. Cambridge: MIT Press.

Turing, A. (1937). "On Computable Numbers with an Application to the Entschiedungsproblem," in Proceedings of the London Mathematical Society 42:230-265.

Wallis, C. (1994). “Using Representation to Explain,” E. Dietrich (ed.) Thinking Computers and Virtual Persons, pp. 307-330. Dortrecht: Kluwer (1994).

Watt, R and Morgan, M (1983). "The Recognition and Representation of Edge Blur: Evidence for Spatial Primitives in Human Vision," in Vision Research. 12:1465-
1477.