WHAT IS COMPUTATIONALISM?

Georg Schwarz*

* Since the following paper was commissioned on relatively short notice, I have incorporated previously existing material. In particular, much of section III is taken almost verbatim from a joint paper with Robert Cummins, "Connectionism, computation, and cognition", which will appear in Horgan, T., and J. Tienson, eds., *Connectionism and the Philosophy of Mind*, Kluwer, scheduled for publication in early 1991. For a detailed analysis of computalism, see Cummins, R (1989), *Meaning and Mental Representation.* Boston, MA: The MIT Press/A Bradford Book.

Also, since the purpose of this paper is to present the main tenets of computationalism to an interested, but presumably not specialized, audience, I have skipped the usual references to literature in order to facilitate reading.

I: Introduction, Plot.

The working hypothesis of most of current cognitive science is that systems are cognitive in virtue of computing appropriate functions. Understanding this assumption requires understanding the explanatory role computation is to play in the overall enterprise of cognitive science. I therefore start out by briefly distinguishing various notions of computation that are sometimes confused in discussions of these matters.

I consider a simple, but uncontroversial, example to introduce the principled components of a computationalist explanation. Since, according to computationalism, systems are cognitive in virtue of computing certain functions, a successful explanation requires a specification of what these functions are. While this problem is independent of the question what a computational explanation is, I nevertheless briefly rehearse the traditional position on what makes a function a cognitive one.

But what if some or most cognitive phenomena resist the kind of autonomous specification required by such an orthodox computationalism, as seems increasingly likely? I conclude by outlining some of the implications that are entailed by the intractability of the specification problem.

II: Three notions of computation

It is by now well known that computation plays an important role is cognitive science. Just what this role is, however, is often much less appreciated. This becomes especially clear when one looks at the discussion surrounding the thesis that "the mind is a computer". The arguments found there, primarily put forward by opponents of the thesis, involve a rather mixed bag of issues, ranging from certain metamathematical results (e.g. that formal systems have more than one interpretation) all theway to the details of current computer architectures (e.g., how can a serial computer help explain the highly parallel workings of the mind/brain?).

All this is mainly due to mixing up several different notions of computation that better be kept separate. Moreover, only one of these notions satisfies the requirements of an empirical investigation of the mind. It will thus be useful to go over these notions briefly, thereby setting the stage for things to come.

Computability. The theory of computation is concerned with determining what (classes of) functions can or cannot be computed. Underlying this enterprise is the *Church-Turing Thesis* which, in its simplest form, states that any function that is computable at all is computable by some Turing machine. A Turing machine, in turn, is a mathematical construct that consists of a finite set of deterministic rules for manipulating tokens of some specified set, and a (potentially infinitely big) memory (or tape), which is used for reading, processing, and

writing these symbols. A function is computable, then, if there exists a Turing machine which, starting with symbols representing the arguments of the function, produces the representations of the corresponding values in a finite (but potentially gigantic) number of steps. Such a Turing machine realizes an *algorithm* for computing the function.

Function computability is thus defined in terms of the existence (in the mathematical sense) of appropriate Turing machines. Notice that nowhere is there any reference made to the inner workings of actual physical systems. Generally speaking, results from the theory of computation apply to the above mentioned debate only indirectly, in the form of imposing negative constraints. As far as the computationalist is concerned, if a function is not computable, and if cognitive processes turn out to be computational ones nevertheless, then it just isn't part of cognition. Opponents of computationalism, however, sometimes claim that the very existence of non-computable functions shows that "the mind cannot be a computer". But this clearly won't do. Even if it could be shown that humans do instantiate such a function (which would also presuppose a more detailed, but not computable specification of the function itself), and so far it hasn't, what would follow is only that *some* aspect of human cognition is not computed.

All this should come as little surprise. Computationalism assumes that cognitive systems compute functions; the existence of non-computable functions would serve to refute it only if these functions could be shown to be constitutive for cognition. So far, this has not happened.

This is not to say, of course, that computationalism cannot be refuted at all. All 1 want to emphasize here is that results from the theory of computation *alone* have little bearing on the issue. The real debate is to be decided somewhere else, as we shall see.

Before leaving the topic of theoretical computability altogether, however, I should at least mention an area of research that is concerned with *practical* computability. *Complexity theory* examines how "expensive" it is to compute certain (classes of) functions. One way of calculating these "costs" consists in determining how the number of processing steps N(I) necessary to produce the output for a given input 1 rises as a function of the "length" of I. Without going into further detail, it has been shown that certain functions are so prohibitively expensive to compute (the number of processing steps rises exponentially with increasing lengths of I), that these functions are, while still theoretically computable, for all practical purposes *computationally intractable*. In particular, they can be expected not to occur in natural systems. Again, the results achieved here will be useful for imposing negative constraints on computationalist research. After all, the time and energy required for computing these inflationary functions presumably outweighs by far whatever benefits they might have for an instantiating biological organism.

Computer. Another notion of computation that can be found in the literature is closely linked to currently available computer architectures. In particular, it is often pointed out that computers are serial machines, executing only one instruction at a time, while brains are known to process several tasks in a highly parallel fashion. Therefore, so the claim goes, the mind/brain cannot be a computer.

It is no doubt true that most current computers are still Von Neumann machines, i.e. they consist of an Input/Output module, a memory unit, and just one central processor. And they are particularly suited for implementing serial algorithms. But the argument just sketched would be valid only if *all* computational processes were necessarily serial. But the rapid development of parallel algorithms should make it clear that the space of computational processes must not be confused with the design principles of the hardware that is currently available for their study. In any case, once parallel hardware will become widely available, all remaining confusions are bound to disappear.

Computation. The careful reader may have noticed that my purpose so far has been largely negative. 1 addressed two different notions of computation, and dismissed them both as either irrelevant or severely confused. What remains to be done, of course, is to supply the Proper Role of Computation.

III: The explanatory role of computation

The working hypothesis of most of current cognitive science is that systems are cognitive in virtue of computing appropriate functions. just how are we to understand this claim? I begin with an illustration of computationalism since it, like many things, is better understood by reference to a prototypical exemplar than by appeal to an abstract characterization.

Calculation. During an archeological expedition you unearth a strange looking device. Close examination reveals that the device has a lid which hides twelve buttons, each with a strange symbol etched on its face, and a small window. Curiosity leads you to experiment, and you notice that certain sequences of button pushings result in certain symbols being displayed in the window, displays which consist only of symbols that also appear on the buttons. Moreover, these display states seem to be correlated with the respective buttons pushed in some nonarbitrary way.

Intrigued by the device's increasingly interesting behavior, you decide to do some cryptography. The display uses only ten of the twelve strange symbols found on the buttons, so you soon start mapping them onto the ten digits familiar from our decimal numeral symbol. On this interpretation the device performs operations on numbers, so you assume further that one of the remaining two buttons serves to seperate the arguments, while the other one is some kind of ENTER key, like "=" on a standard calculator. The rest is easy, for a few test runs show that the system's output (its display state) can be consistently interpreted as the product of the numbers that interpret the antecedent sequence of button pushings. As it turns out, your ancient device is a multiplier.

What happened?

Satisfying. Multiplication is the familiar relation between pairs of numbers and a third one, their product. Calculators, on the other hand, are mechanical or electronic devices whose behavior is governed by the laws of physics. In virtue of what facts can calculators *satisfy the product function* at all?

What reveals the device as a multiplier is the fact that button pushings and display states can be consistently interpreted as numbers and their products. What makes a device a multiplier, then, is that there exists (in the mathematical sense) an interpretation function that systematically maps physical state transitions onto the arguments and values of the product function. And it is precisely this interpretation function that reveals *physical states* of the system as *representations* of the arguments and values of the function satisfied, i. e. as numerals. *Any* physical system with the right representational states will therefore count as a multiplier.

A multiplier is a device such that causing it to represent a pair of numbers causes it to represent their product. Thus, to explain how a device multiplies is (at least) to explain how representations of multiplier and multiplicand cause representations of (correct) products. One possible solution to this problem is to show that the device computers representations of products from representations of multipliers and multiplicands. The working hypothesis of the computationalist is that systems like our multiplier compute the functions they satisfy, and it is in virtue of these computations that the relevant semantic regularities are preserved (i.e. that representations of two numbers *cause* representations of two products). Computing. One way

to satisfy a function is to compute it. Calculators satisfy the product function by computing it. A falling apple, on the other hand, satisfies the function $D=(at^2)/2$ but does not compute it. Wherein lies the difference? Most of us compute the multiplication function by executing the partial products algorithm. It, like all symbolic algorithms, is defined for a certain notation. For example, it does not work for Roman numerals. In case of the partial products algorithm, the objects of computation — the things that the algorithm manipulates — are the standard decimal numerals.

While executing an algorithm always involves following rules for manipulating things, tokens, we might call them, the things manipulated need not always be symbols. Indeed, the objects need not even be representations of any sort. Algorithms implemented on a computer typically manipulate symbols, but many common and useful executed algorithms do not. The most obvious examples are following recipes or instructions manuals.

According to the computationalist, computing a function *f* is executing an algorithm that gives *o* as its output in input *i* just in case that *f(i)=o*. The problem of explaining function computation reduces to the problem of explaining what it is to execute an algorithm for that function. The obvious strategy is to exploit the idea that algorithm execution involves steps. While some of these steps may in turn require the computation of some other functions like adding up the partial products in the example above, the elementary steps will be treated as functions that the executing system (or one of its components) simply satisfies. To execute an algorithm is to satisfy the basic steps, and to do so, as we say, "in the right order". To put it in a nutshell: to compute a function is to execute an algorithm for it, and algorithm execution is disciplined step satisfaction.

It remains to explain what it is for step satisfaction to be disciplined. This seems straight-forward enough in principle. The fundamental idea is that system *d* satisfies the steps in the right order because it is so structured that the steps interact *causally*: satisfying a step is an event in *d*, and events in *d* have effects in *d*; and among those effects are events that constitute the satisfaction of other steps, and so on. The discipline we are looking for is underwritten by the causal structure of the executing device.

Explaining. A physical system, then, computes a function (rather than simply satisfies it) if it executes an algorithm for that function. The falling apple, on the other hand, did not compute its trajectory towards Newton's hand. It is important to appreciate that computation, thus understood, is an abstract causal process — i. e. a causal process specified abstractly as an algorithm. This is exactly what is required if computation is to play any role at all in the explanation of the actual behavior of physical systems. There are lots of causal processes, and only some of them are instances of function computations. It is precisely the latter that constitute the realm of computational explanation. Moreover, there are many ways of satisfying functions, even computable ones G. e. functions for which there exist algorithms), that do not involve computing an algorithm.

A computationalist explanation of the capacity of a device to multiply will consequently begin by attempting to identify representational states of the device and an algorithm defined over them that specifies the "right" (abstract) causal structure of the device, i. e. the causal structure that disciplines transitions from one representational state to another. Our knowledge of already invented multiplication algorithms (such as partial products or successive addition) might help, as might our familiarity with various formal schemes for representing numbers (binary, decimal, etc.). On the other hand, the system might turn out to be a very fast and powerful look-up table, a gigantic associative network, perhaps. And it turns out that the principles governing the most efficient design of *artificial* systems sometimes are of little help when it gets down to explaining the inner workings of *natural* ones.

So far we have been solely concerned with the nature of computational explanation. What we have not addressed, yet, is what it is about cognitive systems that is to be thus explained.

Cognition. If cognitive systems are to be explained computationally, they have to satisfy certain functions. On one (thoroughly rationalist) conception of cognition, a system is said to cognize a domain rather than merely to respond to an environment when that behavior essentially involves respecting epistemological constraints appropriate to that domain, the domain it is said to cognize. That is, its behavior is cogent, or warranted, or rational, relative to both its input and its internal state. These epistemological constraints can be thought of as determining a cognitive function, which in turn allows us to think of a system that satisfies these constraints as satisfying the corresponding function. Computationalism is the hypothesis that systems are cognitive in virtue of computing the relevant cognitive functions. This they do in the same way that multipliers compute the product function, i.e., by executing an algorithm that operates on representations of the arguments to the function to produce representations of their values.

Representation. The commitment to representation alone, by the way, is *not* a distinguishing feature of computationalism. Almost everybody in the field these days buys into representationalism. And everybody believes that representations are linked causally. What does distinguish computationalism is that it is not just causal, but computational processes that discipline the occurrences of representations. For the computationalist the objects of semantic interpretation — the representations are identical with the objects of computation — the things manipulated by the algorithm. It is this extra constraint that distinguishes computationalism from everybody that does not conceive of representations as having, quite literally, a particular form, a form that allows for their computational manipulation. Another point worth mentioning in this context concerns the recent debate about the differences between so-called "orthodox" models and the new connectionism. It is useful to keep in mind that computationalism as such is not committed or restricted to any one particular representational scheme. On the contrary, most current connectionist models share with even the most oldfashioned logicist models a firm commitment to representations. Wherein they differ, and quite radically so, is *what* particular representational schemes (and consequently, what classes of algorithms) they use, not *whether* they use representations at all.

IV. The specification problem

Crucial to computationalism is the idea that there exist independently specifiable cognitive functions, i.e. that cognitive capacities can be specified in a way that is independent of the way any particular system instantiates or computes those functions. A cognitive function, recall, is a function that expresses a set of epistemic constraints satisfaction of which counts as cognizing some particular problem. Thus, computationalism implies that these constraints are always expressible as a computable function. Yet there is reason to believe that, at least for some domains, the relevant constraints aren't even specifiable, let alone specifiable as a computational function.

An analogy with science makes this seem rather plausible. A special science codifies what there is to know about its proprietary domain. But you can't have a special science for any old domain. For example, a science of temperature per se is only possible if, as we say, thermal phenomena form an autonomous domain of inquiry. What this means is that thermodynamics is possible only to the extent that thermal phenomena are governed by laws that describe these phenomena "in their own terms", i.e. autonomously. By way of contrast an autonomous special science of clothing is hardly possible, because there are no special laws of clothing. Clothing, of course, falls under all kinds of laws, e. g. the laws of physics, chemistry, and economics, so there is no question that scientific study of clothing is possible. There are also, no doubt, "rules" in the clothing domain: how to dress for different occasions, what fabrics

need to be washed at what temperatures, etc. But it is doubtful whether these rules can be compiled into an expert system, hence doubtful whether such a set of rules can be specified that expresses what constraints humans satisfy, however it is that they do it. Perhaps human performance in such domains can't be rule driven because it isn't rule describable, and perhaps it is not rule describable because it is, in the very end, example driven.

What if? The potential intractability of the specification problem would also explain the spectacular lack of success in *synthesizing* cognition. As much of the history of Artificial Intelligence shows, the degree of "well-definedness" of a function frequently stands in an inverse relationship to how well humans are at performing it: the "crisper" a function, the more straightforward its computational implementation, as, for example, in the case of theorem proving. Only too often, however, humans turn out to be particularly bad at performing such a task, as, for example, in the case of theorem proving. On the other hand, the "messy" stuff that humans are so good at, like recognizing someone's handwriting, has been haunting AI from the very beginning (not to mention the notorious and apparently perpetually elusive "common sense knowledge").

Should the specification problem thus prove intractable, as seems increasingly likely, then cognitive science will be left without an autonomous domain of inquiry. But this need not be as disastrous as it may sound. For while the notion of a cognition per se will have proven illusory (and a certain rationalist tradition associated with it), we will continue to be surrounded by biological exemplars. Cognitions will be identified ostensively as what humans do when … they recognize their grandmother, solve certain problems, find their way home, etc. Explanations of these capacities, however, will now be much more concerned with the possibilities allowed for, and the constraints imposed by, the biological and/or computational processes of the human central nervous system. The recently exploded interest in connectionism may already indicate this shift in focus.