Matthias Scheutz

COMPUTATIONALISM—THE NEXT GENERATION

The mind is to the brain as the program is to the hardware.
Johnson-Laird

1. Generations: Computationalism and Star Trek

Everyone familiar with current science fiction will most likely recognize the origin of the attribute "the next generation" in the title of this chapter; it is borrowed from the *Star-Trek* saga. So what, you might ask, do *computationalism* and *Star Trek* have in common? For one, both apparently have a "next generation", and furthermore, I would speculate, both "next generations" share the same fate regarding their initial popularity.

What I mean by this analogy is this: when *Star Trek: The Next Generation* came out, every *Star Trek* fan I know missed the old *Starship Enterprise*, its old crew, and in general the old technology, despite the fact that the new Enterprise was faster, the new crew smarter, and the new technology more advanced. It took some time to get used to the new frontiers and to appreciate the new show's potential. Once it grew to be appreciated, however, fans reflected upon the limitations of the original series with a smile, perhaps even belittling the flashing lamps on the console indicating that the computer was performing some complex computation.

The next generation of computationalism might just be in a similar situation: most computationalists will probably not like it at first glance, for various reasons. Maybe because the new notion of computation it involves will be too broad in their view, or maybe because it will place emphasis on practical feasibility as opposed to theoretical possibility. Whatever one's reason for a cautious confrontation with a successor version of computationalism may be, I would hope that the additional explanatory power slumbering in such an extension would trigger, as in *Star Trek*, the same "next generation" effect.

Nevertheless, there is a major difference between both "next generations" as far as their *status quo* is concerned: *Star Trek's* successor has been produced already, while the new computationalism is still in the making. To get an idea of where this rebound of cognitive science's main view on mind—the "next generation of computationalism"—might be heading, I first trace the roots of computation to the 17th century to expose the strong, original bond between mind and computation. Then, after sketching the main tenets of the "old generation" of computationalism, I offer but a glance at some of the shortcomings for which computationalism has been criticized, interspersed with a list of issues that a successor notion of computation will have to address. Since this chapter is intended to be largely introductory, I have included references to subsequent chapters wherever appropriate—using only the last name of the respective author—for more detailed discussions of the various aspects of a "new computationalism".[1]

[1] Some of the following material is based on "The Cognitive-Computational Story" in Scheutz (2000) with the permission of Academia Verlag, St. Augustin.

2. Mind as Mechanism

The notion of computation, although most prominently visible and increasingly entrenched in human culture these days, is not an invention of our times, despite the plethora of different computing devices that have become part of our daily lives. Rather, it dates back to the 17th century and before, when different kinds of *mechanisms* were constructed to control the behavior of various types of machines (from looms, to organs, to watches and clocks). In particular, the first functioning *mechanical calculators* were built at that time: from Schickard's "calculating clock", to the first adding machine constructed by Pascal, to Leibniz's "Stepped Reckoner", and others, some of which are still in working order today (e.g., see Williams, 1997, or Augarten, 1985). Composed of different mechanical parts like wheels, gears, springs, cogs, levers, etc., they employed the technology developed by watchmakers and were able to perform simple numerical operations such as the addition of two decimal numbers. Note that these machines were far from being *autonomous* (in that they did not have their own energy source) or *automatic* (in that they could not perform operations by themselves). To use Sloman's distinction, both *energy* and *control information* have to be provided by humans for them to function properly.

While the potential of mechanisms for the construction of new kinds of machines was generally recognized, philosophers (like Descartes, Hobbes, Leibniz, La Mettrie, d'Holbach, and others) realized another potential of "mechanism": that of *mechanistic explanation*. As Copeland shows, parts of the human body were described in terms of analogies to mechanical parts (such as springs and wheels) and the behavior of the whole body explained in terms of mechanistic principles (e.g., Descartes). For some (e.g., La Mettrie) this explanation did not have to stop at the body, but could be further extended to the human mind (e.g., because the mind was viewed as organized in correspondence to parts of the body, i.e., the brain, which, in turn, could be explained in mechanistic terms). *In nuce*, the *mind* was viewed as a *machine*, giving rise to what Copeland calls "historical mechanism".

As Sloman points out, mechanical calculators are, in some sense, special kinds of mechanisms, since they are not used to control other machines or mechanical devices, but rather to perform calculations, i.e., operations on numbers. However, since numbers are abstract entities, calculations cannot be performed directly on them, but have to be mediated through something physical that can be manipulated. Typically, these mediators were found in physical objects whose physical properties obey laws that are governed by operations that correspond systematically to the ones performed in the calculation. For example, the property "mass" is "additive": put two objects on a scale and their respective masses add up (this is why we are interested in an operation like "addition" in the first place). Hence, by correlating the magnitudes of a physical dimension (e.g., the length of physical objects) with numbers, calculations can be performed by physically manipulating objects (e.g., arranging them in a line), and then measuring the resulting magnitude (e.g., measuring the total length using a ruler). It was not until Vieta had introduced the concept of "representatives" that these slow and error-prone operations gave way to operations using *representations* (in modern terminology, a transition was made from "analog" to "digital" representations). The advantages of *marks* to *stand in* for numbers (over correlating them with physical magnitudes of objects) led to a rapid expansion of this method for carrying out calculations with representations and eventually became a paradigm for thought in general (Pratt, 1987). In the end, this rise of modern mathematics supported the view (due to Descartes, Hobbes, Leibniz, Locke, and others) that not only calculating, but thinking in general involves *the use and manipulation of representations*. In its most radical form, this idea can be and has been summarized by saying that "everything done by our mind is a computation" (Hobbes, 1994, p. 30).

3. To Reason is to Calculate...

In presenting these two views on mind that figure crucially in the origins of computationalism (the "mind" as some sort of a machine, e.g., like a calculator, and "thinking" as involving "the manipulation of representations"), I did not specify what kind of machine mind is taken to be, nor what kinds of manipulation thinking is supposed to involve. While the first question is left unanswered by historical mechanists, the second may find an answer in Leibniz's conception of a "mechanical reasoner", a mechanical system that performs logical reasoning without any human aid.

Two important ideas underwrite the possibility of a mechanical reasoner: (i) that reasoning involves the manipulation of representations, and (ii) that logic can be viewed as a formal, deductive system in which reasoning takes the form of deductions that proceed according to rules. The first idea is intrinsically connected to Leibniz' view on concepts and language that there are simple representations, from which all complex representations are built, but which themselves cannot be analyzed in terms of other representations—Harnad would call these simple representations "grounded". As an aside, one of the challenges in cognitive science nowadays is to give an account of these simple representations, what they are, where they come from, and how they can be used (see also Harnad's chapter).

The second idea is crucial to "mechanizing reasoning": it is by virtue of viewing logic as a formal, deductive system, that valid principles of reasoning can be formulated as rules of deduction, and once principles of reasoning are cast in the form of such rules, we can apply them directly to representations without having to know what the representation is *a representation of* and without having to justify the validity of the conclusion. Hence, a mechanism constructed to take "representations" (of whatever form) and apply rules to them (similar to the calculating machines that perform operations on representations of numbers) would be able to reason by virtue of mere "calculations". Leibniz was even convinced that his formal method of reasoning would resolve any philosophical disagreement, for once a statement is formalized, its validity can be checked *mechanically*:

> *…there would be no more need for disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, to sit down to their slates, and say to each other (with a friend to witness, if they liked): calculemus-let us calculate. (Leibniz, 1875-90, p. 200)*

Note that Leibniz' mechanical reasoner is an early example of what Haugeland (1985) calls "automatic formal system", and that, furthermore, his view of calculating (e.g., as employed in the mechanical reasoner) already hints at the modern computationalist proposal that as long as the syntax is right, semantics will take care of itself.

The above sketch of views and relationships among mind, mechanism, and reasoning is—besides functioning as a brief historic exposition of the mechanist thinking—intended to introduce a suggestion I would like to make with respect to the origins of computationalism: the core ideas of present day computationalism can be already found in the 17th century. Even though their ideas were not expressed in today's terminology and are not based on our modern understanding of the notion of computation, the early "computationalists" (or historical mechanists) realized that the notion of computation effected a link between the mental and the physical. This is because the notion of computation was intrinsically connected to the operations performed by mechanical calculators, on one hand, and to cognitive processes using representations (such as calculating, and reasoning), on the other. It was this link that eventually gave rise to the hypothesis that *mind* might be *mechanizable*.

4. The Separation of Mind and Mechanism

It is important to be aware of the relationships among mind, computation, and mechanism, which were viewed as intrinsically intertwined by the early "computationalists", to be able to appreciate the developments of computationalism in centuries to follow. For one thing, the materialism implied by the early computationalists was overshadowed by various forms of German idealism, which took the mental and not the physical to be primary and whose adherents were by no means open to mechanistic explanations. Not surprisingly, the mechanistic view of mind was not very popular in the 18th and 19th century except with a few convinced materialists, e.g., the German physiologists Vogt, Büchner, or Moleschott).

The last century, however, witnessed a major advance in both the construction of computing devices and the conception of computation. While many attempts were made at building mechanical calculators up to the end of the 19th century (some of which were quite successful, e.g., see Pratt, 1987, Williams, 1997, or Augarten, 1985), the computing capabilities of these mechanical devices remained extremely modest compared to the electronic computers we know today, a development initiated by the rapid progression in the engineering of electronic components (from vacuum tubes, to transistors, to integrated circuits, and beyond, e.g., see Williams, 1997).[2] Similarly, the notion of computation (e.g., as used by the early computationalists) remained at a mere intuitive level until it became the focus of logical analysis stimulated by investigations of the notions "formal system" and "demonstrability" (i.e., proof by finite means) of formulas in formal systems, which, in turn, led to further studies of notions such as "recursive function", "effectively computable function", "algorithm", "finite state automaton", and others in the first half of the 20th century.

It was at about this time that the notion of computation "split" and took off in two different directions, each of which led to a particular view on and interest in computation. The "logical" or "theoretical" route was concerned with a logical explication of the intuitive notion of computation and theoretical limitation results of what could be computed in principle, whereas the "*techno*-logical" or "practical" route was very much focused on all sorts of problems connected to producing reliable computing devices that could be used in a variety of contexts, from scientific computing, to military and commercial applications, and satisfy the increasing demands for fast and powerful machines.

While both approaches were concerned with important aspects of computation and are by no means incompatible, their use of the term "computation" is subject- and interest-specific. Moreover, since research can be conducted quite independently in both disciplines, it is not surprising that these two routes did not cross very often in the past. Only in more recent times do we witness a mutual interest, as logic became more sensitive to real-world constraints

[2] The timeline available at IEEE's web site demonstrates graphically the enormous gain in momentum of computer development ever since the first transistor was constructed in 1947, which led Gordon Moore in 1965 to make the prediction—now called "Moore's Law"—that the number of transistors per integrated circuit would double every 18 months. Interestingly, Moore's Law is still true today.

(complexity theory, for example). Alternative conceptions of computations such as interactive Turing machines, games, etc. are thought to overcome the separation and dissociation of classical logical models from worldly concerns.

5. Cognitive Science or the Rebirth of Computationalism

This separation of computation into two quite independent notions somewhat parallels the separation of mind and mechanism that had taken place at various times in the history of philosophy before, but most notably with Descartes. While it enabled people to talk about computations without the need to refer to a particular mechanism that could carry them out, it introduced an explanatory gap between computations *qua computations* and what does the computing, i.e., the *mechanism* or *computer*, eventually leading to what Smith calls the *mind-body problem for machines*: how are computations related to computers? (I will come back to this problem in section 9).

The independence of computations from their physical realizers, however, was one major source of attraction for some psychologists in the late 1950s. Another was the potential of computers to process information—an ability thought to be crucial to human cognition. Together they gave rise to a powerful metaphor, often called the "computer metaphor", that "the mind is to the brain as the program is to the hardware" (Searle, 1980, or Johnson-Laird, 1988).[3] It was this computer metaphor that underwrites the rebirth of computationalism in the 20th century, and the birth of what is nowadays known as *cognitive science* (e.g., see Gardner 1985): by viewing cognitive functions as computations, explanations of mental processes in terms of programs become scientifically justifiable without having to take neurological underpinnings into account—the "wetware brain" is simply viewed as a computer on which the software "mind" is running (or if not mind itself, then at least all the cognitive functions that constitute it).

Pinpointing the various positions and claims subsumed under the notion computationalism would be a research project in its own right. Just to give you an idea of what can be found in the literature, slogan-like phrases such as "the brain is a computer", "the mind is the program of the brain", "cognition is computation", or "the mind is a computer", are not uncommon, and these are only a few. Note that in a phrase like "cognition is computation" the interpretation of every single word matters, "is" included—do we interpret "is" as "extensional identity" or "extensional inclusion"? Or do we read it intensionally, etc.? Such statements are necessarily condensed and cannot be taken at face value; for if they were read together, essentially distinct notions (such as program and process, mind and cognition, etc.) would be equivocated.

There are other descriptions of computationalism that emphasize the information processing capabilities of computers. For example, computationalism has been characterized as the conjunction of the theses "thinking is information processing", "information processing is computing (i.e., is symbol manipulation)", and "the semantics of those symbols connect mind and world". Again others emphasize the reliance on logical notions of computations. Dietrich, for example, takes computationalism to be "the hypothesis that cognition is the computation of functions", which

> *makes no claims about which functions are computed, except to say that they are all Turing-computable (computationalists accept the Church-Turing thesis), nor does it make any specific claims as to how they got computed, except to say that the functions are systematic,*

[3] Note that the computer metaphor should read "the mind is to the brain as *(computational) processes* are to the hardware" to avoid blurring the "programs-processes" distinction.

productive, and interpretable in a certain way. (Dietrich, 1990, p. 135)

Since many computationalists seem to be (or have been) content with notions of computation as provided by formal logic, it is helpful to know the interest of logicians in computation and to understand the motivations and results of the "logical route" to be able to put the logical contributions to and possible influence on 20th century computationalism into perspective.

6. Logic and Computation: The Rise of Turing Machines

The logical side of the history of computation started in the Thirties with different attempts to make the intuitive notion of computation (then called "effective calculability") formally precise. It was solely concerned with what could be computed *in principle*, which, in turn, required a thorough analysis of the *intuitive notion of computation*. The most crucial insight of the 1930s with respect to the meaning of this intuitive notion of computation was most likely the fact that three different attempts to characterize it formally could be proven equivalent: the class of recursive functions equals the class of λ-definable functions equals the class of Turing machine computable functions. These equivalence results are possible, because what "computing" means with respect to any of the suggested formalisms is expressed in terms of functions from inputs to outputs; and using functions as mediators, the different computational formalisms can be compared according to the class of functions they compute.

Later, other formalisms such as Markov algorithms, Post production systems, universal grammars, PASCAL programs, as well as various kinds of automata were also shown to "compute" the same class of functions, referred to as *recursive functions* (e.g., see Hopcroft and Ullman, 1979). The extensional identity of all these formalisms supports a definition formulated by Church, which later became known as "Church's Thesis":

We now define the notion [...] of an effectively calculable function of positive integers by identifying it with the notion of a recursive function on positive integers (or of a λ-definable function of positive integers). This definition is thought to be justified by the considerations which follow, so far as positive justification can ever be obtained for the selection of a formal definition to correspond to an intuitive notion. (Church, 1936, p. 356, also in Davis, 1965, p.100)

Using the various equivalence results it follows from "Church's Thesis" that any of the above-mentioned formalisms captures our intuitive notion of computation, that is, *what it means to compute*. Although this thesis cannot be proved in principle as mentioned by Church himself, it became more and more plausible as newly conceived computational formalisms were shown to give rise to the same class of "computable" functions.

What is common to all these computational formalisms, besides their attempts to specify formally our intuitive notion of "computation", is their property of being independent of the physical. In other words, computations in any of these formalisms are defined *without* recourse to the nature of the physical systems that (potentially) realize them. Even Turing's machine model, the so-called "Turing machine", which is considered the prototype of a "mechanical device", does not incorporate physical descriptions of its inner workings, but abstracts over the mechanical details of a physical realization.

Interestingly enough, Turing (1936) invented his machine model of "computation" in order to capture the human activity of "computing", i.e., the processes a person (the "computer") goes through while performing a calculation or computation using paper and pencil. Although he used the term "computer" to refer to whatever is doing the computations, he did not intend it for a digital computer, but for a human person doing computations—at that time digital computers did not yet exist.

In his analysis of the limitations of the human sensory and mental apparatus five major constraints for "blindly" following rules to do computations crystallize (I follow the presentation in Gandy, 1988)[4]:

1. Only a finite number of symbols can be written down and used in any computation
2. There is a fixed bound on the amount of scratch paper (and the symbols on it) that a human can "take in" at a time in order to decide what to do next[5]
3. At any time a symbol can be written down or erased (in a certain area on the scratch paper called "cell")
4. There is an upper limit to the distance between cells that can be considered in two consecutive computational steps
5. There is an upper bound to the number of "states of mind" a human can be in and the current state of mind together with the last symbol written or erased determine what to do next

Although there are certainly some steps in Turing's analysis of an abstract human being performing calculations that seem rather quick and not well supported, one can summarize the above in Gandy's words as follows:

The computation proceeds by discrete steps and produces a record consisting of a finite (but unbounded) number of cells, each of which is blank or contains a symbol from a finite alphabet. At each step the action is local and is locally determined, according to a finite table of instructions. (Gandy, 1988, p. 81)

In other words, by "abstracting away" from persons, scratch paper, etc., Turing claimed that all "computational steps" a human could possibly perform (only by following rules and making notes) could also be performed by his machine. In this way the Turing machine became a model of human computing, an *idealized* model to be precise, since it could process and store arbitrarily long, finite strings of characters. It is worth pointing out that Turing, as opposed to Church, did not only state a "thesis" regarding the intuitive notion of computation, but he actually intended it as a theorem (see also Gandy, 1988, p. 83, who restates Church's Thesis as Turing's Theorem): "Any function that can be computed by a human being following fixed rules, can be computed by a Turing machine".

Turing also believed the converse, that every function computed by a Turing machine could also be computed by a human computer, although this, again, does not take time and space restrictions seriously, but rather assumes an abstract human computer not subject to such worldly limitations. In particular, Turing was convinced that "the discrete-state-machine model is the relevant description of one aspect of the material world—namely the operation of brains". (Hodges, 1988, p. 9, see also Copeland's chapter). The origins of Turing's claim can be found in the intrinsic connection between the notion of "computability" and Gödel's notion of "demonstrability" (of a proof in a formal system): that which can be "demonstrated" using "definite methods" amounts to what can be done by a Turing machine (see Turing, 1936). By

[4] Note that "Turing's account of the limitations of our sensory and mental apparatus is concerned with perceptions and thoughts, not with neural mechanisms. and there is no suggestion that our brains act like Turing machines." (Gandy, 1988, p. 87)

[5] This requirement does not exclude an arbitrary amount of scratch paper. It just delimits the range of perception, i.e., the amount of information the human "computer" can use at any given time to determine the next step in the computation.

relating the limitations of formal systems as pointed out by Gödel to the limitations of his machine model, Turing

> *[…] perceived a link between what to anyone else would have appeared the quite unrelated questions of the foundations of mathematics, and the physical description of mind. The link was a scientific, rather than philosophical view; what he arrived at was a new materialism, a new level of description based on the idea of discrete states, and an argument that this level (rather than that of atoms and electrons, or indeed that of the physiology of brain tissue) was the correct one in which to couch the description of mental phenomena. (Hodges, 1988, p.6)*

7. Strong AI – The Heydays of Computationalism

Ever since its conception, the notion of a Turing machine has been used in the meta-theoretical discourse of AI and cognitive science for two opposing purposes: either to justify the use of computers to model, simulate, or duplicate cognition as underwriting computationalism, or to argue that minds are not Turing machines (e.g., King, 1996). This philosophical tension is reflected in literature throughout AI's (and cognitive science's) fifty-year history. Copeland, for example, shows how various deviants of the Church-Turing thesis as well as certain statements by Turing himself have been mistaken as claims about the nature of mind. In accordance with Agre, one could add this tension—is the human mind a Turing machine?—to the many other foundational discrepancies, which have been tacitly imported into AI from the surrounding intellectual territory without having ever been digested.

Despite and mostly independent of this and other unresolved philosophical debates (see Agre's and Sloman's chapters), AI based on the *computer metaphor* (also called *strong AI* or *GOFAI*, for "good old-fashioned AI"), has been quite a successful project, which already early on produced many impressive programs (from Newell, Shaw, and Simon's *Logic Theorist* and *General Problem Solver*, to Samuel's checkers program, to Bobrow's *Student*, to Weizenbaum *Eliza* and Colby's *Parry*, to Winograd's *Shrldu*, to Lenat's *Automated Mathematician*, to various expert systems like *DENDRAL*, *MYCIN*, and *XCON* and other game playing programs, e.g., see Crevier, 1993).

Two main assumptions are buried in this metaphor: (1) that the mind can somehow be "understood as a computation" or be "described by a program" (this requires the adoption of a notion of computation or program, respectively), and (2) that the same kind of relation that obtains between computational processes (i.e., executed programs) and computer hardware—the implementation relation—obtains between minds and brains, too. While assumption (1) led to fertile research in artificial intelligence and cognitive science, which in turn has been taken as evidence for its truth, assumption (2) by and large remained at the level of an assumption.[6]

Expanding a bit on the first assumption to get a better idea of how to think of the mind as being described by a program, note that computation can be viewed as the *rule-governed manipulation of representations*, very much in line with the views of the early computationalists. After all, this is what computers do: they manipulate symbol tokens (e.g., data structures

[6] This is so, presumably, because neither AI researchers nor psychologists need to pay attention to it. AI researchers, who build computational models, implicitly deal with the implementation relation of software on computer hardware on a daily basis, but are not concerned with the implementation relation of minds on "brain hardware"; nor are psychological studies, which remain at the level of "program description". Neuroscience would probably be the closest discipline concerned with implementation issues of brains. Yet, neuroscientists do not attempt to relate program-like descriptions to brain areas, rather they attempt to study the functional role of these areas (with respect to the rest of the brain) directly by virtue of their physiological functions. However, so-called "computational neuroscientists" already make use of "alternative models of computation", e.g., connectionist networks.

implemented in sequences of bits), which themselves are representations of the subject matter the computation is about (compare this to Newell's notion of "physical symbol system", 1980). These representations, in turn, have both *formal* and *semantic* properties, of which only the former are *causally efficacious*, i.e., can do "real work" in the system. This view of computational processes as manipulating symbols by virtue of their formal and not their semantic properties—call it "formal symbol manipulation"—is predominantly found in the philosophical literature (e.g., Fodor, 1981, or Haugeland, 1985), but summarizes nicely what is appealing (or appalling, depending on one's point of view) about computationalism: that the *semantics* of symbols, or more generally, *intentionality* plays no role in cognitive processing, leading to the hope that computation might be the cornerstone for a theory of consciousness (e.g., see Dennett, 1991) and intentionality (see also Smith's and Haugeland's chapters) and.

Computationalism has many appealing facets, especially when it comes to high-level cognition: many features related to logic and language (such as systematicity, productivity, compositionality and interpretability of syntax or the compositionality of meaning, e.g., see Fodor and Pylyshyn, 1988) are supported by computations "almost for free", and many mental operations on various kinds of representations such as rotating three-dimensional images, searching for a good move in a chess game, reasoning about other people's behavior, planning a route through a city avoiding construction sites, etc. can be described computationally and implemented on computers.

In sum, the crucial factors making the notion of computation an attractive candidate for explaining cognition include the potential of computations to have semantics, while being causally efficacious, algorithmically specifiable (in programming languages), and implementable in digital computers.[7]

8. Computationalism under Attack

Computationalism has always been under attack from various directions even before it was officially recognized as such (if not for anything else then for the mere fact that for some it was inconceivable that mind can be mechanized; especially, for any dualist, who believes that the *mental substance* is different from the physical, the computationalist paradigm is unacceptable). Yet, the possibility of using formal methods to characterize the notion of computation, especially the notion of "effective procedure", enabled critics of computationalist views to utilize results from formal logic in order to make their points more rigorously. At the beginning of the 1960s, for example, Lucas (1961) used Gödel's incompleteness theorems to argue that the mind cannot be a Turing machine. The ensuing intellectual debate about what is right and what is wrong with this claim had various repercussions throughout the subsequent decades, most recently spurred by Penrose (1989,1994) (e.g., see the various comments on his books in *Behavioral and Brain Sciences* 13 and *PSYCHE* 2).

Various other objections, not based on formal logic, such as problems with the notion of representation, wide notions of meaning, content and supervenience, universal realization results, etc. have been advanced by philosophers over the years and debated at great length (e.g., see Putnam 1975 and 1988, Dreyfus 1979 and 1992, and others). One particularly famous and still

[7] Compare this to Bridgeman's statement about the subject matter of artificial intelligence: "Artificial intelligence is about programs rather than machines only because the process of organizing information and inputs and outputs into an information system has been largely solved by digital computers. Therefore, the program is the only step in the process left to worry about." (Bridgeman, 1980).

widely discussed problem is Searle's Chinese room thought experiment (e.g., see Searle, 1980 and its various commentaries, Searle, 1990, and also Harnad, 2001).

More recently, connectionists have tried to broaden (if not replace) the notion of computation with alternatives arguing that the symbolic/computational level of description so crucial to computationalism cannot be taken for granted. Others—biologists and neuroscientists, for example—are trying to "go in under" the computational level to understand the mind directly at the level of the brain. Diversely, some social theorists and roboticists have argued that the essence of intelligence is to be found in situated interaction with the external world, rather than in a purely internal world of symbol manipulation. Harnad, for example, calls such "disconnected", internal symbols "ungrounded" and shows how it is possible in a robotic system to ground some of its internal symbols (in sensorimotor projections of distal objects), by virtue of which other "ungrounded" symbols can obtain their "meaning".

While some connectionists believe that symbolic activity should emerge from a "sub-symbolic level" (e.g., Smolensky, 1988), most of the so-called "dynamicists" (e.g., Port and van Gelder, 1995) find the symbolic level of description superfluous altogether and argue instead for an explanation of cognition in terms of dynamic systems.[8] For example, van Gelder (1998) argues that the notion of an effective procedure is essential to computation, and that essential to the notion of an effective procedure, in turn, is the notion of discrete steps in an algorithm. He claims that this discreteness, in both its temporal and non-temporal aspects, prevents computation from explaining many aspects of cognition, which he considers to be a fundamentally dynamical phenomenon. Hence, instead of using any of the standard computational formalisms, one ought to use *dynamical systems* in describing cognitive functions, the idea being that every real-world system (and thus cognitive systems as well!) involving change can potentially be modeled by a dynamical system—this is what dynamic systems have been designed to do. According to the respective system, this will happen at different levels of description, at the very low level of fields (take Maxwell's equations), or the very high level of human decision making (e.g., take the decision field theory by Busemeyer and Townsend, 1993).[9]

Another attack, also advanced by dynamicists, challenges the role of representation in cognitive science in general, and *a fortiori* can be seen as a challenge to the role of computation in cognitive science. Especially psychologists have argued that certain allegedly "cognitive" tasks have nothing to do with cognition proper, but are really motor control tasks that can be explained and modeled in the language of dynamical system without resorting to manipulations of representations (e.g., see Thelen and Smith, 1994). As a consequence, the following question arises: to what extent do notions of representation have to be involved in explaining cognitive abilities; and furthermore: is it possible to invoke "representation" within dynamical system

[8] Surprisingly, a common (mis)perception among dynamicists seems to be that computationalism and "dynamicism" are mutually exclusive.

[9] To describe a physical system, one needs to introduce a variable for each relevant physical dimension and consider it as a function of time. The simplest way to specify the behavior of the physical system would be to provide graphs of each such variable over time, that is, to have a set of functions $X_1(t)$, $X_2(t)$, ... , $X_n(t)$ where $X_i(t)$ yields the "state" of the relevant physical dimension X_i at time t. This set of functions will determine the behavior of the system for all times. However, it does not reveal the possible dependencies of the X_i on each other. This is where differential equations come in handy. They provide a way of specifying the various interdependencies of different variables in such a way that graphs of each variable can be obtained from them, yet the interdependencies are also brought to the open. The nature of these interdependencies become a crucial factor in an explanation of the behavior of the system, and the mathematical theory of dynamical systems seems well-suited to describe quantitatively systems that exhibit such interdependencies.

theory itself when needed, thus bypassing the "classical representational=computational" level of description (see the various articles in Port and van Gelder, 1995)?

Finally, there are criticisms concerning the very founding notions of computationalism: the notion of computation and its conceptual complement, the notion of implementation, some of which are presented next.

9. The Recalcitrant Notions of Computation and Implementation

As Smith argues, traditional notions of computation underwriting computationalism are conceptually inadequate and at best incomplete, revealing crucial hidden assumptions and conceptual dependencies, on which various different construals of "computation"—from "formal symbol manipulation", to "effective computability", to "execution of an algorithm"—are based (see also Smith, 1996 and forthcoming). One main consequence is that computation cannot provide a theory of intentionality (as hoped by computationalists), but rather seems to depend on it. Haugeland, for example, exposes deep and complex relationships and dependencies between intentionality and responsibility, which have been largely ignored by computationalists, and which need to be accounted for in the computational terms if computationalism is to succeed as an explanatory theory of mind.

To get a feeling for the kinds of problems that arise from classical notions of computation, take the view of computation as "computation of an input-output function" (e.g., as proposed by Dietrich, 1990) and try to answer the following questions: can every computation be explained as the computation of a function? Consider, for example, arcade video games. What input-output functions do they compute? Or take operating systems. They are specified by programs, which are designed to never halt. Contrast this then with the classical approach, where such so-called "divergent" functions (i.e., functions that are not defined for certain arguments) are neglected. Further questions about *how* a function is computed arise: does computing a function imply "following rules" or "executing an algorithm"? Does the computation of a function have to be *productive* or would non-algorithmic methods that arrive at the same results count as computing the function too (a big "look-up table", for instance)? And, in general, how can we say that a particular physical system computes a given function, i.e., that the system implements a particular computation?

Most computationalists tacitly assume that the notion of implementation (as used in computational practice) is unproblematic and it is quite common to think of implementation as some sort of correspondence between computational and physical states. For example, there is a tight correspondence between parts of the architecture of a von Neumann CPU and expressions of the assembly language for that CPU. Another example of how computational descriptions can "mirror" physical descriptions is a logic gate (e.g., an AND gate), whose "computational capacity" is described by a *Boolean function*, the values of which in turn can be related to physical magnitudes in the physically realized circuit.

While we may be able to establish a functional correspondence between physical and computational states for certain artifacts (i.e., devices we have designed to allow for computational descriptions), it is unclear that this can be done for natural kinds (such as brains) too. In particular, one has to be aware that any such correspondence depends crucially on "adequate" physical states:[10] computations "mirror" the causal structure of a system under a given

[10] In the case of electronic devices the appropriate physical states can be determined either by asking the engineers who designed the devices or by looking at the blueprint and comparing it to the computational description of the device. In the case

correspondence function between computational and physical states only relative to the choice of the physical states. Computational explanations of the behavior of a given physical system, therefore, depend essentially on those physical states of the system that can be set in correspondence to some computation. This dependence, *per se*, is not problematic as long as one can assume “appropriate physical states” of a system (e.g., as in the case of electronic devices). If, however, it could be shown that for any computational description and any given physical system, one can find “physical states” of that system that can be set in correspondence with the computational ones and are, furthermore, appropriate (in a certain sense of “appropriate” which depends on the underlying physical theory), then computational explanations would be in danger: every system could then be seen to compute! In other words, computationalism would be vacuous if every physical system could be viewed as implementing every computation.

And, indeed, it has been argued that assuming an intuitive notion of implementation any (open) physical system can be seen to implement any finite state automaton (Putnam, 1988), or that walls implement the WordStar program (Searle, 1992). If that is so, then something must have clearly gone wrong with our intuitions, since such a view of computation and implementation is not tenable, neither from the theoretician’s nor the practitioner’s perspectives.

Many people have attacked these claims by finding flaws in their arguments, some have even attempted positive accounts of what it means for a physical system to implement a computation (e.g., Chalmers, 1996, or Melnyk, 1996). However, even these revised accounts have problems of their own (e.g., Scheutz, 2001), which in the end might require an account of implementation that is not based on the idea of “mirroring” computations in the physical system, but rather on a series of abstractions over the physical peculiarities of the implementing system (Scheutz, 1999).

10. A Rebound of Computationalism?

While all of the above-mentioned critiques of computationalism vary, they share a common theme: computation fails as an *explanatory notion* for mind, because computation necessarily neglects the real-time, embodied, real-world constraints with which cognitive systems intrinsically have to cope. This is a consequence of assuming computation to be defined solely in abstract syntactic terms (abstracting over physical realization, real-world interaction, and semantics).

How can we continue to hold on to computationalism, you may ask, when it seems that digitality is restrictive, formal symbol manipulation is not sufficiently world-involving, and Turing machines are universally realizable to the point of vacuity? It should not come as a surprise that these issues combined with recent progress made by dynamicists (while the classical approach *prima facie* appears stagnant), led many cognitive scientists to abandon computationalism altogether.

Despite the dire prospects for computationalism currently envisioned by some (especially convinced dynamicists), an increasing number of researchers are not willing to give up what has

---

of biological systems, however, such states are not clearly defined. Consider, for example, physical states of a pyramidal cell: which of those states could correspond to computational states such that the respective computation captures essential parts of the causal structure of these cells? It has been suggested that “firing” vs. “not firing” would be “natural” candidates (e.g., by McCulloch and Pitts, 1943), but it turns out that this computational model is too reductive as essential temporal processes (such as temporal integration of signals, maximal firing rates, etc.) are completely neglected. Hence, more factors about pyramidal cells need to be taken into account, yielding more physical states that have to correspond to computational ones, etc. Artificial neural networks seem to be promising candidates for such computational descriptions, but the issue has not to my knowledge been resolved. It might well be that in the end the complex behavior of pyramidal cells defies a computational description, but this is obviously an empirical issue.

been *in nuce* a very promising and up to now partly quite successful approach. Inspired by their recognition of the fact that real-world computers, like minds, also deal with issues of embodiment, situated interaction, physical implementation, and semantics, they are motivated to reconsider the very notion of computation as well as the whole enterprise of computing and computational modeling. This motivation is at least partly based on the possibility of classical computationalism failing *not because computing is irrelevant to mind*, but because the classical, purely "logical" or "abstract" theories of computation, which were taken to be the fundaments of the "old generation computationalism", do not to address real-world aspects that are vital to both (real-world) computers and minds.

Artificial intelligence has never been out of touch with the practical, technological route of computation (as Sloman reminds us). Many of the features of present day computers employed in artificial intelligence research were originally inspired by features of (natural) brains, and conversely inspired theories about brains. Yet, this fact is usually ignored by any theoretical discourse that views computation as Turing-machine computability.

Furthermore, by (mis)interpreting and applying results from the logical route of computation, in particular the Church-Turing thesis, to argue that cognition is or is not Turing-computation, Turing computability is often equated with computability by a mechanism. However, as Copeland points out, there are *conceptual computing machines* that can outperform any Turing machine (e.g., Turing's O-machine—whether such a machine could possibly exist is a separate issue), which are compatible with the mechanistic views of the early computationalists (the operations of such machines can also be described algorithmically, see, e.g., Shagrir, 1997). Such computing devices undermine the common prejudice that computation is limited to "effective computation" (or Turing-computability).

Another part of the problems of computationalism may be the conceptual separation of mind and mechanism, as reflected in the separation of computation and what does the computing underwriting computationalism. It almost seems as if this split attempted to "hold something apart from and above material reality", to use Agre's words, who locates a general pattern of such "dissociations" permeating AI.

All of this seems to point in the same direction: that the current problems of computationalism do not so much lie in computing *per se*, but *in our present understanding of computing*. The notion of Turing computability (and with it other classical notions) may be part of the foundational problems of computationalism, if—as Smith suggests—the theory of Turing machines is not, contrary to current orthodoxy, a theory of computation in the first place, but rather a *theory of marks*, and hence *not* a theory of *intentional phenomena*. In fact, as Sloman's reconstruction of the historic development of computers and mechanical artifacts (especially within artificial intelligence research) shows, it may be irrelevant and not required for computationalism at all.

However, what is required and will have to be addressed by the next generation is the intrinsic relationship between intentionality and responsibility, which Haugeland makes clear in his positive account of intentionality: for a system to be able to have *genuine intentionality* means to have the capacity to accept what he calls "authentic responsibility". A milestone along the way to implementing genuine intentionality may be to start with Harnad's suggestion of "grounding" symbols in sensorimotor capacities. Such grounded symbols can be used to form new complex symbol structures, which inherit their meaning systematically from their constituents, and may lie at the heart of language, and consequently, meaning.

At this point, I have explicated the *status quo* of the “old generation” and identified some of the goals for the “next generation” of computationalism. The contours of this “next generation” are already visible, but need to be fleshed out in detail. Now is the time for the chapters to take over, elaborating many of the above as well as other issues in a first attempt to rehabilitate what may still be our best bet for explaining cognition.