Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 317 (2004) 115–145
Theoretical Computer Science
www.elsevier.com/locate/tcs

Natural computation and non-Turing models of computation

Bruce J. MacLennan*
Department of Computer Science, University of Tennessee, Knoxville, TN 37996-3450, USA

Received 15 August 2003; received in revised form 17 October 2003

Abstract

We propose certain non-Turing models of computation, but our intent is not to advocate models that surpass the power of Turing machines (TMs), but to defend the need for models with orthogonal notions of power. We review the nature of models and argue that they are relative to a domain of application and are ill-suited to use outside that domain. Hence we review the presuppositions and context of the TM model and show that it is unsuited to *natural computation* (computation occurring in or inspired by nature). Therefore we must consider an expanded definition of computation that includes alternative (especially analog) models as well as the TM. Finally we present an alternative model, of continuous computation, more suited to natural computation. We conclude with remarks on the expressivity of formal mathematics.

Keywords: Analog computation; Computation on reals; Continuous computation; Natural computation; Turing machine

1. Introduction

The principal purpose of this article is to argue for and to propose certain non-Turing Machine models of computation. The intent is not so much to advocate models that surpass the power of Turing Machines (TMs) as to defend the need for models with orthogonal notions of power. Traditionally, the power of a computational model has been defined in terms of the class of functions it can compute. However there are alternative criteria of merit (notions of power) more appropriate to *natural computation* (computation occurring in or inspired by nature).

* Corresponding author.
E-mail address: maclennan@cs.utk.edu (B.J. MacLennan).

0304-3975/$ - see front matter
doi:10.1016/j.tcs.2003.12.008

I begin by reviewing the nature of mathematical models of any sort, arguing that they are relative to a domain of application or concern and are generally ill-suited to use outside that domain. This observation motivates a discussion of the presuppositions and context of the TM model, arguing that, valuable though it is, it is ill-suited to certain important applications. It has been argued that Turing computation is what we *mean* by computation, but I propose a broader definition of computation that includes Turing computation as well as alternative (especially analog) models. This is followed by a discussion of *natural computation*, which asks questions that the TM model is unsuited to answer. Finally I will present an alternative model, of continuous computation, more suited to addressing the important issues in natural computation. I conclude with some remarks on the expressivity of formal mathematics.

We will be discussing models of computation, so it will be worthwhile to begin by reviewing some characteristics of models in general. (A more systematic discussion of these issues has been published elsewhere [30].) Models are tools intended to address a class of questions about some domain of phenomena. They accomplish this purpose by making simplifications (*idealizing assumptions*) relevant to the domain and to the intended class of questions. Such simplification is both their strength and weakness.

The idealizing assumptions have an (often indefinite) range of applicability over which they give reasonably good answers, but outside of this range they give progressively poorer answers, which reflect the structure of the model more than the underlying phenomena. (That is, these answers tell us more about the map than about the territory it purports to describe.) Sometimes, of course, the range of applicability of a model is larger than intended, but this is a lucky accident and should not be assumed. Therefore we should remind ourselves of the intended domain of applicability of the TM model lest we inadvertently extend it outside of its range of usability.

2. The Turing-machine model

2.1. Historical context

We may begin to understand the range of applicability of the TM model by recalling the historical context that gave it birth: it was developed to answer questions in the formalist program in mathematics, which attempted to reduce mathematics to a *calculus* (discrete formal system). Therefore the appropriate information representations were idealizations of mathematical or logical formulas, and the processes operating on these representations were idealizations of formal mathematical proof and calculation. The central issue was the provability of formulas, since this is relevant to questions of consistency and completeness in formal mathematics. Therefore, a key concept was the class of theorems, or formulas derivable from a given set of axioms by given rules of inference.

A closely related issue, when effective calculability is more the concern than formal axiomatics, is the class of functions computable by a specified calculus or class of calculi. Typically the problem of computability is recast as a problem of computing a function of the integers: given an input integer, whether it will eventually compute the

corresponding output integer. In accord with the assumptions of formal logic (a proof can be any finite length), a function was considered computable if for any input the corresponding output would be produced after finitely many steps.

Similarly, as appropriate for questions of consistency and completeness, no bounds were placed on the length of the individual steps, so long as they were finite. Likewise, no bounds were placed on the size of formulas, so long as they were finite.

As this theory of computation evolved, questions of algorithm complexity were eventually addressed, but from a correspondingly abstract perspective. For example, asymptotic complexity was formulated consistently with these assumptions: once one has chosen to ignore the speed of the individual steps, all that remains is how the number of steps grows with the size of the input. Similarly, one may analyze the size of the formulas produced during the computation. This perspective has led to the curious view that any polynomial-time algorithm is "fast" and that problems that are polynomial-time reducible are virtually identical. So, for example, an algorithm that takes N^{100} years is fast, but one that takes 2^N ns is "intractable."

2.2. Assumptions

Having reviewed the TM model in its historical context, we can summarize its principal assumptions. (Note that some of these assumptions are abandoned in some extensions of the TM model, but they form a basis from which the extensions are made.) The TM belongs to the larger class of calculi, therefore we review the characteristics of a *calculus* or *discrete formal system*; see MacLennan [23,30] for a more systematic analysis.

2.2.1. Information representation

In a calculus, *information representation* is formal, finite, and definite. *Formality* means that information processing is both abstract and syntactic, that is, the operation of a calculus depends only on the form of the representations, in contrast to their substance [1] (*abstract formality*) and their meaning (*syntactic formality*). Abstract formality, when combined with an assumption of an unlimited supply of the representing substance, implies infinite producibility (and reproducibility) of formulas having a given form, which is fundamental to the expression of infinity in formal mathematics [31, Chapters 1–2]. Syntactic formality permits mechanical processing, since we know how to make mechanisms respond to the form of representations (but perhaps not to their meanings). This was a primary goal in the formalist program in mathematics, but also important if we want to eliminate the "ghost in the machine" in artificial intelligence or in scientific theories of natural intelligence.

In calculi, information representations are assumed to be *finite* in size and number of constituents. This is certainly reasonable in the context of formalist mathematics, since mathematical formulas are finite arrays of discrete symbols. It was also essential

[1] I use "substance" for the individuating substrate, which makes one instance of a formula different from another. It might be chalk on a blackboard, light on a display screen, electrons on a capacitor, electromagnetic energy of a given frequency at a given time in a given location, etc.

to a principal historical goal of axiomatic mathematics, which was to account for the infinite in terms of the finite (e.g., Peano axioms, limits).

Finally, in a calculus, information representation is assumed to be *definite*, that is, all determinations are simple and positive (hence reliable). This is because formalist mathematicians were concerned with what could be calculated or proved if these formal processes were carried out with the care and reliability that formalization permits.

There are several additional implications of these assumptions with regard to information representation and processing. Although these assumptions are reasonable in the original context of the study of effective calculability (and also, to a large extent, for many applications of digital computers), it should be noted that they are idealizations, and are problematic under less ideal circumstances, such as will be described later.

First we will consider *texts*, the concrete physical instantiations of information representations or formulas. The smallest (and therefore indivisible) constituents of a text are called *tokens*. Their presence or absence is definite (i.e., they are definitely separable from the background) as is their extent (i.e., where one ends and another begins). Tokens belong to a finite number of *types*, the determination of which is definite (i.e., types are reliably and mechanically discriminable). The tokens belonging to a type are mutually interchangeable, without any effect on the operation of the calculus. (That is, for the purposes of the calculus, the set of allowable tokens may be partitioned into a finite number of equivalence classes.) For convenience we may refer to the *matter* and *form* of a token, provided these terms are not interpreted too concretely (e.g., "matter" as physical matter, "form" as shape). Finally, we assume that it is always possible to make another token of a specified type (i.e., we never run out of the "stuff" constituting tokens).

Notice that the notion of mechanically determinable types is not entirely precise. For example, humans can reliably and mechanically discriminate red and green tokens, but probably not beautiful and ugly tokens. Humans could use, with considerable reliability, pictures of the faces of Gödel and Turing as tokens of two distinct types, but it would be more difficult for a mechanical device to do so with contemporary face-recognition technology.

The notion of mechanical determination is partly relative to what we are trying to explain. For example, if we are studying formal mathematics, it is reasonable to assume that mathematicians can reliably distinguish mathematical symbols, and this process is taken to be unproblematic and transparent. However, if we are attempting to give a computational account of human visual pattern recognition, then it is precisely this sort of process than cannot be taken for granted, and must be reduced to more elementary discriminations.

A *text* is a finite and definite ensemble of interrelated tokens; that is, we can determine reliably which tokens belong to the text, and there are a finite number of them. Thus, we may say that a text is *finite in breadth and depth*.

For the purposes of the calculus, the possible physical relations among the tokens of a text are divided into a finite number of equivalence classes (which may be called *elementary schemata*), and it is always definitely, reliable, and mechanically determinable whether or not a particular elementary schema holds among particular tokens. Therefore, as we did for tokens, we may analyze texts in terms of matter and form.

Some combinations of elementary schemata may not be physically realizable. A physically possible combination of elementary schemata is called a *schema*, which is the form of a text for the purposes of the calculus. Further, the schema to which a text belongs depends only on the types of the constituent tokens, not on the specific tokens (a consequence of abstract formality). (That is, the types and elementary schemata induce an equivalence relation on texts, the equivalence classes of which are called schemata.)

There are assumed to be no resource limitations on the construction of texts; that is, one can always obtain tokens of the required types and arrange them into a text belonging to a specified schema. Generally, no a priori limit is set on the size of a text (although in practice they are always limited by physical resources).

2.2.2. Information processing

Like information representation, information processing in a calculus is formal, finite, and definite. In particular, computation comprises a finite number of discrete, atomic steps of definite type, each step requiring finite time and energy. At each step there is a corresponding text, the *state* of the computation, and at each step an *operation* occurs, which replaces the old state with a new state (which may be obtained by rearranging the old state, or by constructing a new state from scratch, or by any combination of the two).[2] The operation applied in a step depends definitely on the old state's schema (that is, on its syntax, not its substance); likewise, it constrains only the schema of the new state, not its substance.

The changes effected by an operation are mechanical, that is, they require no judgment. However, as already remarked, the standard of what is "mechanical" is somewhat relative to what is being explained. For example, the sensorimotor processes involved in the manual application of an inference rule in formal logic and the substitution for a bound variable in the lambda-calculus are commonly taken as mechanical processes, but they are complex to implement on a computer. Another example: deciding the termination of an arbitrary TM description-input pair cannot be accomplished by a TM, but for some purposes we may take this as a primitive operation (i.e., we may assume we have an *oracle* for the halting problem).

The computational process may be deterministic (if in a given state at most one operation is possible) or non-deterministic (if more than one is possible); that is, there are syntactic (formal) constraints on the applicable operations, and these constraints may limit the operations to one, or more than one, at each step. (The absence of any applicable operation is often taken to signal the termination of the computation.) Thus, in the general, possibly non-deterministic framework, we can see that the steps of a computation are defined by a relation between the schemata of the new and old states (e.g., in the context of formal logic, the relation is "immediate derivability").

Finally, the termination of a computation is definite; that is, we can determine reliably whether or not it is finished computing. Two common ways of signaling termination

[2] For simplicity I will restrict my attention to processes without external input. External input is not relevant to the present discussion, and in any case it can be incorporated into the present framework, for example, by making the input text part of the initial state.

of a computation are by generating a text conforming to a class of *terminal schemata*, or by the absence of any applicable operations (as previously remarked).

2.2.3. Algorithms and programs

Because of the foregoing assumptions, a computational process in a calculus can always be represented in a finite number of discrete, finite rules, that is, in a *program*. This is because a finite-energy operation can inspect only a finite number of tokens and elementary schemata; similarly, it can directly affect only a finite number of them. Therefore the preconditions and effects of an operation can be expressed in a finite rule, that is, by a schema constructed of types and elementary schemata akin to those in the original calculus. Thus, computational processes (algorithms) can be represented in physical texts or formulas. This finite, physical representation of computational process allows the construction of universal machines, such as the Universal TM and general-purpose digital computers.

3. Defining computation

3.1. Digital and analog computation

I will be arguing that natural computation, such as occurs in neural networks, requires non-Turing models of computation, but it may be objected that these processes are not, strictly speaking, computation, which is defined in terms of the Turing machine. That is, it might be argued that "computation," a previously vague concept, was first defined precisely by Turing, and that this is the essential import of the Church–Turing Thesis (as discussed by Copeland and Sylvan [5,6]). This would imply that "non-Turing model of computation" is a contradiction in terms. Therefore, it is necessary to start with a discussion of the definition of "computation." The best definition is far from obvious. Indeed an entire issue of the journal *Minds and Machines* (**4**, 4; 1994) was devoted to the question, "What is Computation?"

Certainly "computation" may be defined narrowly or broadly, and the theoretical success of the Turing-machine model and the practical success of digital computers have encouraged a narrower definition. Indeed, many authors have advocated, or assumed, that computation is equivalent to Turing-computation (see quotations in Copeland and Sylvan [5,6]). Although this definition has the obvious advantage of precision, it also has less-obvious disadvantages.

First, "computation" has traditionally included both digital (or discrete) and analog (or continuous) computation. In the era of manual computation we find both digital devices (e.g., abaci, desk calculators) and analog devices (e.g., nomographs, pantographs); the slide rule is hybrid: analog computation with digital readout.

The constructions of traditional (i.e., compass and straight-edge) Euclidean geometry are early examples of continuous computation. Consider, for example, such "problems" as: "to divide a given straight line into two parts, so that the rectangle contained by the whole and one of the parts shall be equal to the square of the other part" (Euclid II.51), "to find a mean proportional between two given straight lines" (VI.13), "to cut a

given straight line in extreme and mean ratio" (VI.30). These constructions are actually hybrid algorithms comprising discrete steps making use of continuous operations.

Similarly, throughout most of the era of automatic computation, there have been analog computers, from Bush's differential analyzer, through electronic analog computers, to modern analog VLSI devices [32].

3.2. A functional definition

In the mid-20th century many scientists embraced computation and information processing as ways of understanding phenomena in their own disciplines. A notable example is the use of computational models in cognitive science, but linguistics, sociology, genetics, and evolutionary biology may also be mentioned. Especially when we are dealing with natural systems such as these, computation is better viewed as a matter of *what* is being accomplished rather than *how* it is accomplished.

For example, it now appears that primary visual cortex (area V1) does a Gabor-wavelet transform [7–10]. That is, it implements a particular mathematical operation, and that seems to be its purpose in the visual system. It is natural and informative to say that it *computes* a Gabor-wavelet transform. However, to apply the narrower definition of computation, we would have to understand the actual mechanism in the brain before we could say this. If we found a discrete process fitting the assumptions of the Church–Turing thesis, we could call it a computation, otherwise we would have to call it something else (a "pseudo-computation"?). But this seems to be perverse. Surely it is more informative and accurate to say that V1 is computing a Gabor-wavelet transform, regardless of whether the underlying technology is "digital" or "analog."

These considerations have motivated the following functional (i.e., purpose based) definition of computation [22,24,28]:

Definition 1. Computation is a physical process the purpose of which is the abstract manipulation of abstract objects.

By an "abstract object" I mean an object defined by its formal properties. This includes the various kinds of numbers, of course, but also such objects as sequences, sets, relations, functions, curves, Booleans (truth values), trees, strings, and so forth. Similarly, "abstract manipulation" includes the application of functions to their arguments (e.g., square root, Fourier transform), as well processes evolving in abstract time (e.g., Newton's algorithm, formal deduction, gradient descent).

However, because abstract objects do not exist physically, the manipulations must be accomplished indirectly, by physical manipulation of physical surrogates. Thus integers may be represented by the beads of an abacus or the bits of a digital computer, and real numbers may be represented by the position of the slide and stock of a slide rule or by electrical current in an analog computer. That is, abstract formal processes are represented by concrete physical processes. (Even mental arithmetic takes place in the physical brain.)

There is an issue that we must address before we consider the relation of the abstract process to its physical realization in computation, and that is the problematic use of

"purpose" in the preceding definition. Scientists are justifiably wary of teleological definitions, which appeal to purpose or function. Nevertheless, I think it is unproblematic in this case, for in two domains—biology and technology—attributions of purpose can be made in an objective way.

It is an objective issue (decidable, for example, by appeal to the designers) whether or not a device has been designed for purposes of computation (i.e., abstract manipulation of abstract objects), and so we can objectively affirm that your laptop is computing, as also did slide rules. Establishing purpose or function in a biological context is more difficult, but scientists routinely draw such conclusions; so, they make objective determinations of the function of the stomach, heart, immune system, and so forth. (The major danger here is oversimplification, since the biological systems often serve multiple functions: evolution is opportunistic. Thus, although there are certainly purely computational systems in the brain, we should not be surprised to find systems that serve other purposes while they compute.)

What about other physical systems? Can we say (to take a classic example) that the solar system is computing Kepler's laws? The answer is No. Where there is no objective basis for attributing purpose (e.g., the motion of the planets), there is little to be gained by viewing a process as computation. No doubt there are borderline cases (e.g. in very simple organisms, populations, simple artificial devices), but they do not destroy the general utility of the definition.

Since the purpose of a computational system is abstract manipulation of abstract objects, we have an operational test of whether or not a particular natural system is computational [24]. If the purpose or function of that system would be fulfilled as well by another system using different physical surrogates with the same formal properties, then the system is computational. That is, since its function is accomplished independently of its concrete physical instantiation, its purpose is abstract, formal, computational. This is *multiple instantiability*, which is a familiar property of digital computation, but holds as well for analog. Thus, for example, if the same abstract (dimensionless) quantities can be represented by electrical voltages or fluidic pressures, or by any other concrete physical quantities with the same formal properties, without loss of function, then the system is computational. In contrast, the digestive system is not computational, since it will not fulfill its function if, for example, enzyme concentrations are replaced by other quantities (e.g., electrical charge density), having the same formal properties.

3.3. Autonomy

It will be worthwhile to make a few remarks about differing degrees of autonomy in computational processes, since the TM model takes a somewhat limited view. *Autonomy* refers to the degree to which a process proceeds independently of any input.[3] (Here I am referring to computational input to an information-processing module; see "Transduction" below on input/output interactions with the physical environment.)

[3] This notion of autonomy, which comes from control theory, is unrelated to that used in artificial intelligence and robotics, which refers to the system behaving intelligently without external guidance.

We may think of the input and state of a process as the independent and dependent variables of the computation. That is, the external inputs are independent of the computation in question (although perhaps they are outputs of other computational processes). The internal state, however, is completely (but perhaps nondeterministically) dependent on the external inputs and previous internal state. (Output is most easily treated as a projection of the state, as will be explained in more detail later.)

The most autonomous computation has a specified initial state and proceeds independently of external input. Simple examples include the computation of a specific number such as $\sqrt{2}$ or e. When the computation is complete, the output is projected from the state in any of several possible ways.

A more common situation, which is only slightly less autonomous, accepts input once, at the beginning of the computation, and then proceeds autonomously until completion, when the result may be projected from the state. (Alternately, the input may become the initial state of the computation. Also, a sequence of outputs might be projected from intermediate internal states.) A simple example of such a computation would be the computation of $\ln x$, for a given number x, by integration of $\int_1^x (1/u)\,\mathrm{d}u$. Another would be the computation of the Fourier transform of a given input image. Clearly, a computation of this sort is computing some function $f(x)$ for an arbitrary unspecified input x. It is worth observing that this is the paradigm case in the traditional theory of computation, which defines computational power in terms of classes of functions.

In the next lower degree of autonomy, the process depends on its input throughout the computation; that is, it depends on a (discretely or continuously) varying input signal. There are two broad subclasses: the output may be projected from the state once at the end of the computation, or it may be projected from intermediate states at various times during the computation. An example in which a time-varying input signal leads to a single output would be a classifier for an auditory signal, such as a spoken word recognition system. For examples in which a (discretely or continuously) varying output signal is generated, we may take any control systems (e.g., feedback control systems for sensorimotor coordination or industrial process control).

Perhaps the least autonomous computation is one in which the internal state (and hence also the output) depends only on the current external input; such a memoryless process is completely reactive.

These ideas are easy to express mathematically; for simplicity I will restrict my attention to deterministic processes. Let the internal state space Ψ and the external input space Φ be any given spaces, and let T be an appropriate (discrete or continuous) time domain. Then the state transition function $P : \Psi \times \Phi \to \Psi$ defines the state at the next instant of time: $\psi(t') = P[\psi(t), \phi(t)]$, for time-varying internal state $\psi : T \to \Psi$ and time-varying input $\phi : T \to \Phi$. For discrete-time processes $t' = t + 1$ (representing the next step); for continuous-time we may take (informally, for the time being) $t' = t + \mathrm{d}t$.

Then, a completely autonomous process, which depends on no external input, has the form $P(\psi, \phi) = P_0(\psi)$ where $\psi(0) = \psi_0$, the fixed initial state (taking, without loss of generality, $t = 0$ as the initial time). For a process that depends only on the initial

input, $P(\psi, \phi) = P_1[\psi, \phi(0)]$. For a completely non-autonomous, reactive process, we have $P(\psi, \phi) = P_2(\phi)$. This should be sufficient to illustrate the approach.

3.4. Transduction

Before leaving the definition of computation, it is necessary to say a few words about transduction. *Transduction* is the process that converts a signal in a specific physical form into the implementation-independent representation suitable for computation, or vice versa. For example we may have a transduction that converts an external optical signal into the internal computational media (e.g., electrical signals in a computer or brain), or which converts from the computational media to mechanical forces applied by output effectors (e.g., muscles or mechanical actuators). Although computation is by definition physically instantiated, it is *generically instantiated*, and so formal. However, in transduction, if the physical realization of the peripheral variable is changed (say, from voltage to fluid pressure), the transduction cannot be expected to serve its purpose in the system; it is *specifically* instantiated. Transduction brings formal computation into interaction with the physical world, and therefore it is a critical issue in situated intelligence and the *symbol grounding problem* [12,13,21].

We may divide the relations that govern the behavior of a situated computational system into *formal relations* and *material relations*, which govern computation and transduction, respectively. In the formal relations, all that is relevant to the computation is the form of the representations. Although these forms must be instantiated in some physical quantities, the specific physical instantiation is irrelevant to the computational purpose. Thus, a computer may make use of electrical representations in some places and optical representations in others. So also in the brain, both electrical and chemical representations are used. Since the physical quantities are irrelevant, the formal relations are effectively dimensionless. The material relations, on the other hand, deal with physical quantities in an essential way, otherwise they will not serve their purpose. For example, material relations that relate light intensity to computational inputs will not serve their purpose (in an organism or a robot) if they are altered to relate some other physical quantity (say, pressure). Therefore, at least some of the variables in the material relations carry physical dimensions. Specifically, an input transduction has dimensionless outputs, but physically dimensioned input; the opposite holds for an output transduction.

From another perspective we can think of information representations in terms of their *form* and their *matter* (by which I mean the "stuff"—be it physical matter or energy—constituting the representations). *Pure computation* deals with the form of representations; their matter is irrelevant to the computation. On the other hand, a *pure transduction* changes the matter of a representation without altering its form. Thus an optical signal might be converted to an electrical representation. Pure transductions typically remove the physical dimensions of input quantities or add them to output quantities, in effect scaling the variables. Thus the material relations of a pure transduction may have a very simple form (e.g., "$e = v/10$ mv" relates an input voltage v to a dimensionless real number e). Pure transduction is actually quite rare; it is more common to change the form along with the matter of the representation. For example,

a continuous physical quantity might be categorized and discretized by an analog-to-digital converter. Conversely, an output device might interpolate continuously between its discrete inputs, as in a digital-to-analog converter. Even transductions that do not convert between discrete and continuous quantities typically alter the form of the information. For example, a photodiode implements an "impure" transduction since it filters the analog signal as well as converting it from an optical to an electrical representation. As previously remarked, such combination of function is especially common in natural systems.

3.5. Classification of computational processes

Hitherto, so far as possible, I have dealt with computation in "topology-neutral terms," so that the discussion applies equally well to discrete- and continuous-time processes. Now, however, we must distinguish three different classes of processes characterized by the topologies of their state and (computational) input spaces and by the topology of time [28]. There are three important classes:[4]

C: A continuous-time process over continuous state-space,
CD: A discrete-time process over continuous state-space,
D: A discrete-time process over a discrete state-space.

For examples of class D we may take conventional digital computer programs. Examples of class CD include Newton's algorithm and some formulations of computation over the reals [2]; they are closely related to topological algorithms and other systems studied by Burgin [3,4]. Systems of differential equations are obvious examples of class C.

In the most common cases time is linear. Thus discrete time is a well-ordered discrete set, typically homeomorphic to the integers (or a subset thereof). Continuous time is homeomorphic to the reals $[0, \infty)$ or a closed interval thereof. More generally, a computational time domain may be defined as a partial order on an appropriate discrete set or continuum of "instants."

Although the continuous/discrete distinctions are mathematical and precise, hybrid computations are possible. For purposes of this classification, if any component of the state or any of the input spaces are discrete, then the process must be considered class D or (if some are continuous) CD.

3.6. Realization as homomorphism

The physical realization of an abstract computational process can be expressed more precisely by putting it into mathematical form. The abstract process that a computational system implements has some mathematical structure; for example, it might be a function on the integers or a system of differential equations. The purpose of the computational system is to physically instantiate or realize this abstract structure; thus the system has the generic structure common to all realizations of that abstract computation. However, the realizing physical system always has additional specific properties irrelevant to the

[4] Class DC, continuous-time processes over discrete state-spaces, is excluded by the laws of physics.

computation. (For example, an electronic AND gate radiates heat and electromagnetic noise, but these properties are not relevant to the computation of logical conjunction.) Therefore, in an ideal case, there is a *homomorphism* from the realizing system to the abstract system, a mapping that loses irrelevant physical structure while preserving the relevant (computational) structure.

More formally, consider a computational process $\Pi: \Psi \times \Phi \to \Psi$ on abstract state space Ψ and abstract input space Φ. There will be a corresponding (discrete or continuous) time domain T and a monotonically increasing function $v: \mathrm{T} \to \mathrm{T}$ that gives the next "instant" of abstract time (as before, this is informal for continuous processes). The state and input are functions of time, $\psi: \mathrm{T} \to \Psi$ and $\phi: \mathrm{T} \to \Phi$, satisfying the abstract process equation:

$$\psi[v(\tau)] = \Pi[\psi(\tau), \phi(\tau)] \quad \text{for } \tau \in \mathrm{T}.$$

Similarly, an intended physical realization is a function $P: S \times X \to X$ on physical state space S and physical input space X. This process will take place in real-time $\mathbf{R}^{\geqslant 0} = [0, \infty)$ at (discrete or continuous) instants of time given by monotonically increasing $n: [0, \infty) \to [0, \infty)$. The states $s: [0, \infty) \to S$ and inputs $x: [0, \infty) \to X$ are functions of time satisfying the concrete process equation:

$$s[n(t)] = P[s(t), x(t)] \quad \text{for } t \in [0, \infty).$$

Further, suppose we have functions mapping the physical representations to their abstract correspondents: $\mathscr{H}_s: S \to \Psi$, $\mathscr{H}_i: X \to \Phi$, and $\mathscr{H}_t: [0, \infty) \to \mathrm{T}$. The latter function defines a correspondence between time in the abstract and physical systems:

$$v(\mathscr{H}_t\{t\}) = \mathscr{H}_t\{n(t)\}.$$

(If $\mathscr{H}_t$ is an identity function, we have a real-time system; that is, abstract time is physical time.) The physical process realizes the abstract process provided that these functions commute under the state and input mappings:

$$\mathscr{H}_s\{\mathsf{s[n(t)]}\} = \Pi[\mathscr{H}_s\{\mathsf{s(t)}\}, \mathscr{H}_i\{\mathsf{x(t)}\}].$$

3.7. Approximate realization

Most realizations are imperfect. That is, some of the abstract structure may not be supported by the physical implementation. For example, integer arithmetic on a digital computer may overflow and depth of recursion may be limited; analog computation is subject to noise. To put it differently, an explanation of a physical system as implementing a particular abstract computational process is a model of that physical system, and typically an idealization of the real system.

It is also important to observe that an approximate realization can be of a different type to the realized system. To take a familiar example, a system of differential equations may be realized approximately by a digital computer program (manipulating discrete floating point numbers in discrete steps). Conversely, although perhaps less obviously, a continuous physical system can approximately realize a discrete computational or formal system. For example, in every digital computer binary logic devices

are implemented approximately by electronic circuits obeying continuous differential equations. So also, the mathematician scribbling on a blackboard is an approximate continuous realization of formal logic and mathematics.

I must digress to forestall a possible misunderstanding. When we classify an abstract process as C, CD, or D, the classification is precise, because we are dealing with a mathematical classification of mathematical objects (i.e., their topology). However, when we classify physical realizations as continuous or discrete, we must treat them at the relevant level of abstraction. For example, for most circuits it is reasonable to treat electrical charge as a continuous quantity. However, for very small devices we must take account of the fact that charge is quantized in terms of electron charges. At even smaller scales we must treat electrons as continuously distributed probability amplitudes. What they are *ultimately* (supposing we can ever know) is irrelevant; what we want is a model appropriate to the scale at which we are working. Thus, when we classify a physical process as C or D, for example, we mean that is a good model for the purposes at hand.

4. Natural computation

4.1. Natural computation defined

The reader may agree that the TM grew out of a particular set of concerns somewhat removed from modern computation, but see no reason to doubt its efficacy as a general model of computation. Therefore, in this section I will discuss "natural computation" [e.g., 1] as an important area of computer application to which the TM model is especially unsuited (see MacLennan [30] for a fuller discussion).

Definition 2. Natural computation is computation occurring in nature or inspired by that in nature.

Examples of computation occurring in nature include information processing in the brain, in the immune system, and through evolution by natural selection; indeed, the entire discipline of cognitive science is oriented around computational models. In all of these cases (and more) scientists have found it fruitful to understand natural processes in terms of computation. Therefore, natural computation is an important discipline for its contribution of theories, models, and metaphors to the other sciences.

Examples of computation inspired by nature include artificial neural nets, genetic algorithms, simulated immune systems, ant colony optimization, particle swarm optimization, and simulated annealing. Their considerable actual and potential importance in many applications has illustrated the technological value of understanding computation in nature. These non-traditional computational paradigms are most relevant in those applications that are most similar to natural systems, for example, autonomous robotics, real-time control systems, and distributed intelligent systems. Therefore, in order to understand the models of computation most relevant to natural computation, we will need to keep in mind these kinds of applications as well as natural systems in which computation plays a role.

4.2. Relevant issues

I will begin by reviewing some of the issues that are most relevant in natural computation, and therefore which should be addressed in suitable models of computation. One of the most obvious requirements of natural computational systems is *real-time response*. For example, generally an animal must respond to a sensory stimulus within a fraction of a second; similarly, sensorimotor coordination takes place in real-time. Thus the speed of the basic operations is critical; also, the absolute number of steps from input to output in a discrete-time process and the rate of a continuous-time process must be such as to deliver usable results in prescribed real-time bounds. Further, algorithms that yield progressively closer approximations to an answer will be more useful than those in which intermediate results are useless, since the former will permit the use of premature results, if so required by real-time considerations. Such algorithms also facilitate anticipatory processes, which prepare for a response before its initiation.

Analysis of algorithms based on the traditional (TM) model of computation is oriented toward asymptotic complexity, that is, how utilization of some resource (typically time or space) grows with the size of the input. Such analysis is less relevant in the context of natural computation, since the size of the input is generally fixed (e.g., by the structure or anatomy of a sensory system). For example, our optic nerves have approximately one million nerve fibers delivering impulses at several hundred hertz. That is the magnitude of input with which our visual systems must deal. If it can deliver its results in the required real-time constraints, it does not matter how its algorithms would perform with 10 times the number of inputs or 10 times the impulse rate.

Certainly algorithms in natural computation can be compared, but the criteria of merit are different. One of these criteria is *speed of response*. Although the sizes of the inputs and outputs are fixed, one algorithm may be better than another if it can deliver a result in less real-time. Another criterion of merit is *generality of response*. That is, while the input and output dimensions and the real-time response limits are fixed, a natural computation may be improved by increasing the range of inputs to which it responds well.

A related criterion is *flexibility in response to novelty*. That is, still within the bounds of its input/output spaces, one computational system may be better able to respond appropriately to novel inputs than can another. A novel input is one that an artificial algorithm was not designed to handle, or that was outside of the environment of evolutionary adaptedness that led to the evolution of a naturally occurring algorithm.

Related to this is the issue of *adaptability*. Since the natural environment is unpredictable and ever-changing, an important issue in natural computation is whether a system can adapt to a changing environment, and how quickly it can do so, while retaining existing competence. Thus we are concerned with computational processes that can change their dynamics on various timescales. Therefore, natural computation systems can be compared with respect to the quality and speed of their adaptation as well as the stability of their learning. This does not imply that all natural computation algorithms are adaptive, but that models of natural computation should easily accommodate adaptation.

Further, since we assume the presence of noise and other sources of uncertainty, *gradual adaptability* is generally preferable to precipitous change. Thus the gradual adaptation exhibited by, for example, neural networks and genetic algorithms, is more appropriate than the discrete addition and deletion of rules typical of learning algorithms based on calculi (traditional discrete formal systems).

Tolerance to noise, *error*, *faults*, *and damage* is also important in natural computation. The natural world is messy, so animals and autonomous robots, for example, must be able to make use of inputs that are very noisy. Furthermore, natural computation itself is noisy and imprecise: biological devices such as neurons have many sources of noise and their computation is inaccurate. For example, neural signaling has perhaps one digit of precision. Analog computational devices have similar properties.

Finally, since the natural world is dangerous and often hostile, natural computation systems may be damaged, and so their behavior should be robust in the presence of faults or other sources of error. Therefore natural computation systems must operate in such a way as to be immune to noise, errors, faults, and damage, or even to exploit them (as, for example, noise is exploited in stochastic resonance).

4.3. Relevant assumptions

Based on these considerations, it is possible to outline some of the assumptions that are appropriate for a model of natural computation. First, a natural computation system must be physically realizable, and so its use of matter and energy must be *finite*; all physically instantiated quantities must be finite. Furthermore, noise and other characteristics inherent in physical instantiation may dictate other sorts of finiteness (e.g., bandwidth, rates of variation). For example, noise is often high-frequency, which limits bandwidth on the high-frequency end. Also, spatially distributed information may have an underlying "graininess," which is equivalent to high-frequency noise. Real-time response requirements and physical size may limit frequency (temporal or spatial) on the low end.

It is reasonable to suppose that natural computation exhibits a kind of *syntactic formality*, by which I mean that the computation is governed by the physical aspects of representations, not by any meanings that they may be supposed to have. (Here I am not using *syntax* in any precise linguistic sense, but by analogy with formal languages and to indicate the physical form of a signal as opposed to its *semantics* or meaning.) In this sense we can speak of a *continuous formal system* [25]. There are at least two reasons for this assumption. First, a principal reason for using computational models in the natural sciences is to banish the "ghost in the machine" from our scientific explanations. If we can account for some behavior or cognitive capacity, for example, by a purely mechanical process, then we are confident that we are not falling into a circular argument and assuming what we are trying to explain. Second, in order to design autonomous, intelligent machines, we have to be able to reduce natural computation to purely mechanical processes, that is, to systems that we can design and build.

Natural computation systems, insofar as they are purely computational, also exhibit *abstract formality*, that is, a dependence on the abstract forms of representations and their formal relationships, rather than on their substance. Of course, as previously

explained, we cannot assume that naturally occurring information processing systems will be purely computational, since nature often combines functions. Indeed, as we apply natural computation in artificial systems, we too may find it advantageous to combine function. Nevertheless, natural computation qua computation is characterized by abstract formality.

As previously discussed, *real-time response* is generally important in natural computation. Therefore the notion of an abstract sequence of (albeit finite) computational steps is of limited use in natural computation. Instead, regardless of whether we are dealing with discrete- or continuous-time processes, we will generally want, at least in principle, to be able to relate these to real-time. That is, we will be concerned with the rates of continuous-time processes and with absolute bounds on the duration of the steps of discrete-time processes.

Since the laws of physics are continuous (especially at the typically relevant scales), often input, output, and state spaces should be assumed to be continua, and information processing should be assumed to be continuous in real-time. Therefore, continuous models are often better matches to the relevant phenomena than discrete models.

Natural computation assumes that noise, uncertainty, error, and indeterminacy are always present (in both information representation and processing). For each "correct" representation there will be others that are arbitrarily close, so representational spaces are best treated as continua. Hence *robustness* is important: small errors should lead to small effects. Therefore it is generally appropriate to assume that functions and processes are continuous. Discontinuous processes can lead to brittle response, which is typical of conventional computation, but undesirable in natural computation.

On other hand, input, output and other representations are assumed to be of *fixed* "*size*" (e.g., dimension, physical extent, bandwidth), as opposed to the "finite but unbounded" representations typical of TM computation.

The ability to adapt gradually to novelty implies that physical representations of natural computational processes are at least partially *continuous* (as opposed to digital computer programs, which are finite, discrete structures, which, if they adapt at all, must do so in discrete steps).

In the most characteristic cases, natural computation is *non-terminating*. That is, a natural computation system is in continuous interaction with its environment, and that interaction terminates only when the system (e.g., organism, population) ceases to exist. Thus it is not usually useful to think of natural computation as computing a function (essentially a model more appropriate to old-fashioned "batch" computing). Rather, most natural computation systems are better viewed as real-time control systems. Therefore we assume that, in the general case, useful natural computation may be non-terminating.

5. Motivation for continuous computation

5.1. Principle of continuity

The preceding discussion of natural computation makes no commitment as to whether discrete or continuous models are preferable. This is an empirical issue, and no doubt

different instances of natural computation will require different sorts of models. Nevertheless, for a number of reasons in the remainder of this paper I will focus on continuous models, and in particular on *field computation*. First, discrete models are already familiar, and so there is little need to discuss them further here. Second, continuous computation, and in particular field computation, will serve as an example of an alternative model to the TM, which is more relevant to natural computation in the brain. It is also relevant to large artificial neural networks and to massively parallel analog computers (optical computers, Kirkhoff machines, etc.). Furthermore, continuity avoids brittleness and enhances robustness and flexibility. Small changes have small effects. Hence continuous information representation and processing is especially suited to natural computation. Therefore, in order to keep our attention focused on this alternative model, in the remainder of this paper I shall adopt a *Principle of Continuity*, which constrains our models to be continuous; in particular, information representation and processing are assumed to be continuous.

5.2. Continuous information representation

We focus on continuous information representation for a variety of reasons. First, there is considerable evidence for the use of continuous representations in the brain. One should not be misled by the "all or nothing" generation of a neural impulse, for (1) the impulse is a continuous waveform defined by differential equations (the Hodgkin–Huxley equations), (2) information is encoded in the continuously variable rate and phase of impulses, and (3) impulses in dendrites are graded and interact spatiotemporally according to differential equations (the cable equations). Further, (4) the synaptic efficacies, which—so far as we know—encode memory, are complex functions of the spatial distribution of (albeit discrete) receptors. Similarly, most artificial neural net models are, at least partially, continuous. Although they are often implemented on digital computers, they are most naturally described by continuous mathematics (real numbers, linear algebra, derivatives, differential equations, etc.).

Therefore, in accord with our Principle of Continuity, we assume that all information representations are continuous (i.e., they are drawn from continuous spaces). Naturally, continuous quantities can be approximated by discrete quantities, but we must beware of modeling artifacts resulting from the process of approximation, especially when we are investigating fundamental properties of computation. The more direct—and safer!—approach is to use continuous models from the beginning. A discrete approximation is adequate only if it does not alter the phenomena of interest.

How, then, is information represented continuously? Certainly finite-dimensional vector spaces are appropriate for many purposes, and they are a common medium of representation in both natural and artificial neural systems. However, for many purposes *infinite*-dimensional vector spaces (i.e., Hilbert spaces) are more useful. In particular, in modeling the activity of the nervous system it is often useful to treat an information representation as a *field*, that is, a *spatially extended continuum of continuous quantity*.

Thus, sensory images are naturally described as fields; consider a static visual scene: intensity (of various wavelengths) varies continuously over the optical field. Similarly

in an auditory image the sound pressure varies continuously with time. Indeed, visual images are also time-varying, so they are continuous functions of space and time. We should not be misled by the fact that, for example, the retina comprises a finite number of discrete receptors, for the number is so large (10^8) that it is mathematically more transparent to treat it as a continuum. Motor images are also naturally modeled as fields, since they represent the continuous motion of the body in space [26].

Cortical maps, in which significant information in represented by spatially distributed activity in a patch of cortex, have sufficiently many elements to be treated as fields. There are at least 150 thousand neurons in each square millimeter of cortex, and so even the smallest cortical maps have hundreds of thousands of neurons, enough to be treated mathematically as a continuum. As a consequence, field-oriented models have been useful in explaining the operation of cortical maps (citations given elsewhere [26,27]).

It is worth observing that fields contradict many of the assumptions underlying calculi. First, whereas formulas are built up from tokens, fields do not have atomic components in any useful sense. Certainly we can think of the (uncountable) infinity of infinitesimal points, such as the light intensity of a particular wavelength at an infinitesimal point in space and time, but this is far indeed from the concrete tokens of a calculus.

In fact, whereas the fundamental operations in a calculus operate on tokens, and more complex operations result from combinations of these, in field computation the field is treated as a whole. We may analyze what happens to individual points (e.g., ray tracing in optics), but that is for *our* cognitive convenience. In nature, the field is processed as a whole in parallel. Think of the processing of a visual image by the retina and through the visual system, or of the sensation of touch distributed over the skin, or of the motor output to the muscles of a gymnast or dancer.

Certainly, the sensory system analyzes images to extract information from them, and the motor system synthesizes a total motor image from subimages, but in neither case is the decomposition given and canonical, as in the formulas of a calculus. Indeed, learning an appropriate decomposition is often a critical problem for a sensory system.

Although there is evidence that the nervous system makes use of mathematical decompositions of fields, such as Fourier transforms and wavelet decompositions [e.g., 7–10,18], these operations are continuous and holistic. Even when a finite discrete set of (continuous!) coefficients is extracted, as in a generalized Fourier series,

$$\phi = \sum_{k=0}^{N} c_k \beta_k,$$

the coefficients are computed by inner products over the entire image (or over extended parts of it, as in some windowed Fourier transforms). That is, to compute from image ϕ the coefficient c_k corresponding to basis function β_k, we integrate over whole images:

$$c_k = \langle \beta_k, \phi \rangle = \int_{\Omega} \beta_k(x)\phi(x)\,\mathrm{d}x.$$

Thus, in continuous representations, the orientation is on analysis rather than on synthesis (or construction), as it is in discrete representations.

5.3. Continuous information processing

Given that information representation in the brain is continuous, information processing might be either continuous-time or discrete-time (i.e., class C or CD). In addition to the Continuity Principle, there are several reasons for focusing on continuous information processing.

First, the underlying physical processes in the brain are continuous at the relevant level of abstraction; for example, electrical propagation and chemical diffusion processes are defined by differential equations. Certainly, abrupt events may occur, such as the firing of a neuron, but these are best treated as continuous processes that are only approximately discrete. (And indeed the firing of a neuron is described by a differential equation, the Hodgkin–Huxley equation.) Similarly, information processing in analog computers is defined by differential equations. Even when artificial neural networks are simulated on digital computers, the program is often performing a discrete-time approximation to a continuous mathematical process (as when a learning algorithm, such as back-propagation, approximates gradient descent by taking discrete, finite steps).

Second, the bulk of the information processing in animals is continuous. For example, sensorimotor coordination is a continuous, real-time process. Even many higher cognitive processes are accomplished by continuous manipulation of mental images. For example, Shepard [47,48] has shown that mental rotation of three-dimensional objects is a continuous process (see additional citations elsewhere [16]).

5.4. Apparently rule-like behavior

A cognitive domain in which discrete representations and processes might seem to be required includes language, verbal reasoning, propositional knowledge representation, and other apparently rule-based behavior. And this may be so. However, we think that even here continuous models have much to contribute, especially in explaining the flexibility and adaptability of human rule-like behavior, including, in particular, formal methods as applied by mathematicians [11,16]. Understanding these mechanisms could give artificial systems some of these same advantages.

For example, we can sketch the following model of rule-like behavior in a continuous system [25,29]. First observe that a rule-based system categorizes a situation into one of a finite number of classes, each of which is handled by an applicable rule. Then an applicable rule extracts from the situation certain low-dimensional index information (represented by the variables in the rule), which particularizes the situation. The actions performed by the rule depend only on this index information and the content of the rule. So if it creates a complex representation, all of the information must come either from the rule itself or from the particulars selected by the low-dimensional index information.

From the perspective of continuous computation, a rule projects a complex image through a low-dimensional subspace. Further, any function that can be decomposed

into a finite set of such projections will act as though it is obeying a finite set of rules even if the actual intermediate space is not physically represented. That is, if a function $F : \Phi \to \Psi$ between high-dimensional spaces Φ and Ψ, can be decomposed, $F = \bigcup_{k=1}^{N} Q_k \circ P_k$, where $P_k : \Phi \to I_k$ and $Q_k : I_k \to \Psi$, for low-dimensional intermediate spaces I_k, then the system will appear to be following rules, even if the physical computation is not structured in this way. That is, a system may appear to be following rules even though it is not; in effect the rules are illusory.

This continuous model of rule-like behavior has several advantages. First, if a function is approximately decomposable in this way, then its behavior will be correspondingly approximately rule-like. Thus we have an approach to dealing with exceptions in rule-like behavior. In effect, although intermediate information may be generally confined to the low-dimensional spaces I_k, it may occasionally (exceptionally) be outside this space, although still in a larger, higher-dimensional intermediate space. That is, we are assuming that a more accurate decomposition is represented by $P_k : \Phi \to \mathrm{X}_k$ and $Q_k : \mathrm{X}_k \to \Psi$, where $I_k \subset \mathrm{X}_k$ and X_k is high dimensional. Therefore the difference between exactly and only approximately rule-like behavior is the difference between the range of P_k being entirely or only mostly confined to a low-dimensional I_k.

More importantly, this model shows how rule-like behavior may be gradually adaptive. In a discrete computational system that is actually following rules, fundamental adaptation requires the deletion or addition of rules, which will result in abrupt changes of behavior. In a continuous system, however, through gradual adaptation, the ranges of some or all of the projections may first expand from the small subspaces I_k to larger subsets of the X_k and then contract to different low-dimensional spaces I'_k. From the perspective of the observer, the system will have evolved from rule-like behavior, through an intermediate non-rule-like phase, into a new phase that appears to be following different rules. Apparently, the rules gradually dissolve and then resolidify as different rules.

6. Foundations of continuous computation

6.1. Information representation

With this introduction to some of the advantages we hope to obtain from an understanding of continuous information representation and processing, we can turn to a more precise account of its properties. Certainly, there have been a number of approaches to continuous and topological computation [2,4,39,42,44–46]. However, in the following I will postulate certain properties that we expect to hold for any continuous computational system. This in effect defines a kind of *continuous formal system*, which we term a *simulacrum*, and, as a possible foundation for continuous information processing, is the continuous analog of a calculus, or discrete formal system, in its role as the foundation of TM computation [19,23,25]. Since simulacra are intended to be physically realizable—as they must be as a model of natural computation—these postulates will be constrained by what seems to be physically possible in the most general terms.

6.1.1. Topology of images

Through a phenomenological analysis of the invariants encountered in continuous information representation and processing systems, we have proposed a set of common characteristics of simulacra [23], which are summarized here.

We begin by characterizing *image spaces*, that is, the spaces from which *images*, or continuous representations, are drawn. (Images correspond to the formulas of a calculus.) Our analysis suggests that similarity of images can be quantified and that such quantification has the mathematical properties of a *metric*. Further, what defines an image space as a single space is that any image can be continuously transformed into any other in the space. Thus:

Postulate 1. *Image spaces are path-connected metric spaces.*

Therefore, image spaces have at least the cardinality of the real numbers [14, p. 175].

For various technical reasons it is reasonable to assume two additional properties of images spaces:

Postulate 2. *Image spaces are separable and complete.*

A space is *complete* if all its Cauchy sequences have limits in the space, and it is *separable* if it has a countable dense subset. The practical implications of this postulate are that there is a countable set of images in the space that can be used to approximate any image by a sequence of increasingly similar images, and conversely that all such sequences have limits in the space. Completeness and separability are in effect the conditions that permit the description of continuous spaces in our discrete mathematical language; they bridge the continuous and discrete.

In support of this postulate we may observe that a *continuum* is often defined as a non-trivial connected compact metric space [40, p. 158], and that a compact metric space is both separable and complete. Furthermore, a theorem of Urysohn shows that a metric space with a countable base, such as a continuum, is homeomorphic (topologically equivalent) to a subset of the Hilbert space $\mathscr{L}_2(\mathbf{R})$ [41, p. 324]. As we have seen, Hilbert spaces are natural mathematical models of many image spaces and are widely used in natural computation. It is also the basis of *field computation*, which is computation over Hilbert spaces [15,17,20,26,27].

6.1.2. Images and their forms

Physical realizability places additional constraints on images. First observe that images are extended over some physical continuum (which, in the simplest case, may be a single point). That is, the domain Ω of an image (considered as a function $\phi : \Omega \to K$) is a topological continuum (connected compact metric space). Certain physical quantities, the values of the image $\phi(\omega)$, $\omega \in \Omega$, vary continuously over its extent. That is, the image defines a *continuous* function $\phi : \Omega \to K$ of the domain Ω. Further, since Ω is compact we know that ϕ must be *uniformly continuous* [33, p. 178].

A field's domain is bounded (finite in extent), which means that it occupies a finite amount of "space" (whatever concept of space is appropriate to the physical

representation, as represented in the metric). Note that for the typical case in which $\Omega \subset \mathbf{R}^n$ the Heine–Borel theorem guarantees that the compact domain Ω is closed and bounded. This finiteness requirement must be modified slightly for temporal images, that is, images varying in time. In these cases the domain is given by $\Omega = \Upsilon \times \mathbf{R}^{\geqslant 0}$, where $\mathbf{R}^{\geqslant 0}$ represents time (from process initiation) and Υ is a bounded continuum (the signal's non-temporal extent). Thus $\phi(\upsilon, t)$ is the value of the field at location $\upsilon \in \Upsilon$ and time t.

The codomain K of an image is also a continuum, and since the range of a field's variation is finite, the codomain K is bounded continuum. Typically, $K \subset R^n$, a closed and bounded subset of a finite-dimensional vector space. In summary:

Postulate 3. *An image is a uniformly continuous function over a bounded continuum* (*which may*, *however*, *be unbounded in the positive time dimension*).

As previously remarked, it is often appropriate to assume in addition that the variation is band-limited; that is, if the image is expanded in an ordinary Fourier series, then all the coefficients are zero beyond some point.

Subject to the preceding restrictions, we assume that it is possible to construct an image with any pattern of variation over its extent (that is, all bounded continuous functions are possible images). Also, in accord with the principle of abstract formality, only the pattern of variation (the form) of the image is relevant to computation, not its substance. Finally, in accord with the Principle of Continuity, we stipulate:

Postulate 4. *Maps between images spaces are continuous.*

This postulate has many important consequences for classification and categorization in image spaces, which are discussed elsewhere [19, 23]. Another consequence of the Continuity Principle is:

Postulate 5. *Interpretations of simulacra are continuous.*

Since a continuous image of a compact path-connected metric space is a compact path-connected metric space, the interpretations of images constitute an image space. That is, for image space Φ and continuous f, the range $f[\Phi]$ is an image space.

6.2. Information processing

6.2.1. States and processes

A process has a complete state, comprising a finite number of internal state images, and a finite number (possibly zero) of (external) input images. All these images vary continuously in time (if they vary at all). The instantaneous *configuration* of a process comprises the forms of its internal state and external input images. This configuration governs (not necessarily deterministically) the continuous change of (internal) state through time, and such government depends continuously on the configuration.

For non-deterministic processes, there is a continuous probability density function defined over possible computational trajectories. Therefore, there is a continuum between

possible trajectories and impossible trajectories, and thus there are *soft constraints* on the admissibility of trajectories.

Instead of asking whether a continuous process terminates, it is generally more meaningful to determine whether it is asymptotically stable. Such processes converge continuously toward their results, so if an agent must act before an optimal answer has been obtained, it will still have a relatively good result. (Of course, such equilibria are only temporary, since natural computation never stops; typically, the equilibrium will be destabilized by a change of input, which then enables a new equilibrium to be achieved.)

6.2.2. Topological definition of process

In topological terms, a deterministic autonomous process is a continuous function $P: \Psi \times \mathbf{R} \to \Psi$ (where Ψ is the complete state space) that defines the state at a future time in terms of the current state: $\psi_{t+\Delta t} = P(\psi_t, \Delta t)$. Clearly, such processes satisfy the group properties:

$$P(\psi_t, 0) = \psi_t$$

$$P[P(\psi_t, t_1), t_2] = P(\psi_t, t_1 + t_2).$$

More generally, for non-autonomous as well as autonomous processes, the Principle of Continuity requires:

Postulate 6. *Formal processes in simulacra are continuous functions of time, input images, and process-state images.*

We have seen that image spaces are homeomorphic to subsets of Hilbert spaces, and since Hilbert spaces are Banach spaces, we can define the derivative of a process. Note that

$$\dot{\psi}_t = \lim_{\tau \to 0} \frac{\psi_{t+\tau} - \psi_t}{\tau} = \frac{\mathrm{d}}{\mathrm{d}\tau} P(\psi_t, \tau)\bigg|_{\tau=0}.$$

Therefore, if P is differentiable, we can write a deterministic autonomous process as a differential equation $\dot{\psi} = Q(\psi)$ where $Q(\psi) = (\mathrm{d}/\mathrm{d}\tau)P(\psi_t, \tau)|_{\tau=0}$. A similar approach can be used for non-autonomous processes.

6.2.3. Potential descent

Hill-descending processes illustrate many of these ideas. Suppose $V(\psi)$ is a bounded scalar-valued potential function defined over states $\psi \in \Psi$. Then gradient descent is a simple deterministic continuous autonomous process: $\dot{\psi} = -r\nabla V(\psi)$, where r is the rate of descent.

For non-deterministic descent, we can define, for example, the probability density $P(\psi_t, \dot{\psi})$ of change $\dot{\psi}(t)$ to state $\psi(t)$ by a soft constraint such as this:

$$P(\psi_t, \dot{\psi}) = \frac{[-\nabla V(\psi) \cdot \dot{\psi}]^+}{\|\nabla V(\psi)\| \cdot \|\dot{\psi}\|},$$

where x^+ represents the non-negative part of x (of course, a smooth, sigmoidal function could be used instead). The effect of this is to make the probability density of a state change equal to the non-negative part of the cosine of the angle between the negative gradient and the direction of state change. Therefore, descent along the negative gradient will be most probable, but other potential-decreasing directions will also be allowed, with their probability decreasing to zero as they approach orthogonality to the gradient. Then $P\{\psi, a, b\}$, the probability of following trajectory ψ from time a to time b, can be expressed:

$$P\{\psi, a, b\} = \exp \int_a^b \log P[\psi(t), \dot{\psi}(t)]\, \mathrm{d}t.$$

6.3. Process representation

Finally, it will be worthwhile to consider the form that programs may take in the context of continuous computation. This is important theoretically, for its relevance to universal computation, and practically, as a foundation of general-purpose continuous computers.

6.3.1. Discrete formulas

Of course, many continuous processes can be defined by differential equations or other mathematical formulas. In these cases we are using static discrete structures (the formulas) to define continuous-time processes. Similarly, researchers from Shannon onward have designed general-purpose analog computers on the base of interconnecting discrete computational elements from a finite set [15,17,27,39,42,44–46].

Even in this familiar case, however, there are some subtleties that we should notice. Consider the simple differential equation, $y' = ry$. If r is a rational number, then this equation can be written down, that is to say, it can be represented in a finite, discrete structure. If r is not rational, we cannot write it down (represent it discretely and finitely), but if r is a *computable* real number, then we can at least provide a finite procedure for generating progressively better rational approximations. That is, we have a finite, discrete structure (a digital computer program) p such that $\lim_{k\to\infty} p(k) = r$. In effect, our finite, discrete representation is $y' = [\lim_{k\to\infty} p(k)]y$. Notice, however, that the set of computable real numbers is denumerable, so most real numbers are non-computable. Therefore, most continuous processes obeying an equation of the form $y' = ry$ will not be finitely describable in discrete symbols, either directly (by giving a finite formula for rational r) or indirectly (by giving a finite algorithm for approximating computable r). In general, we can see that most continuous processes cannot be expressed or even approximated arbitrarily closely by a finite, discrete structure.

In contrast, an analog computer has no such limitation. If we have an analog computer programmed to integrate $y' = ry$ for a given rate r, then this input can be provided directly as a continuous quantity (e.g., a voltage or light intensity); there is no need to express it discretely (e.g., as a string of digits or an approximating digital computer program). The equation $y' = ry$ can be finite in size, provided we are allowed to

represent r directly by a continuous magnitude rather than a finite, discrete formula. That is, r must be represented by an image rather than a formula.

It is important to avoid several traps into which we may be drawn by our discrete thinking habits. For example, it may be argued that the continuous output of an analog computation has to be measured, which converts it to a rational number. Conversely, to input a quantity, it is argued, it must be typed as a number or selected from a finite set, which means that the set of possible inputs is denumerable. However, both of these objections arise from the incorrect assumption that a continuous computation is interfacing with a discrete environment (such as a human user typing in numbers or viewing a digital readout).

First, even if a human is using an analog computer, inputs and outputs may be continuous: input can be gestural or through a joystick or slider; output can be a dialless pointer or a visual image. Further, in the context of natural computation (which is our focus here), there is generally no "user" providing inputs or consuming outputs. Typically, an organism is responding to continuous inputs from its environment by making continuous actions in its environment. Input, processing, output: they are all continuous, and there need not be discrete computing anywhere. The simplest and most appropriate model is to assume that all the quantities and processes are continuous, as they are normally assumed to be in physics.

6.3.2. *Guiding images*

We have seen that there are limits to expressing continuous computational processes in finite, discrete formulas ("programs"), therefore we might ask if there is some alternative more appropriate to continuous computation. We have already seen one possible extension: the inclusion of continuous quantities in an otherwise discrete representation. This suggests that, just as discrete computational processes are most naturally represented by finite, discrete formulas (programs), so continuous computational processes might be represented by finite, continuous images. We call these continuous analogues of programs *guiding images*.

For a concrete example, consider a potential surface $V(\psi)$ defined over states $\psi \in \Psi$. This can serve as a simple guiding image for a continuous computation: for a deterministic computation, start in an initial state ψ_0 and follow the gradient downward, $\dot{\psi} = -r\nabla V(\psi)$, until an equilibrium (a minimum or saddle-point) is reached. (An asymptotic equilibrium must exist due to the boundedness of images.) The same guiding image can also govern a non-deterministic descent, as was explained in Section 6.2.3.

But where do we get the guiding image for a continuous computation without describing it discretely? Just as a human can write a digital computer program, so a human can "sculpt" (or "paint" or "dance") the guiding image of a continuous process. More likely, perhaps, a human may be in an interactive continuous feedback loop with a continuous computation system that is creating a guiding image. Finally, just as a rule-based system can be constructed by a learning algorithm, so also a guiding image may be sculpted by a continuous adaptive algorithm. (This is, in effect, what many neural net learning algorithms do.) In nervous systems, the guiding images are

created by continuous developmental processes and experiential learning. This is the origin of many of the guiding images encoded in cortical maps.

7. Ubiquity of calcular assumptions

I have argued that the TM model acquires many of its characteristics from the context in which it developed: problems in formal logic and mathematics. I have also argued that a different, equally important set of concerns, those involved in natural computation, suggests a different set of assumptions and consequently different models of computation, including continuous computation. Nevertheless, I would like now to come full circle and consider some issues in the epistemology of mathematics raised by our broadened idea of computation.

Although I have pointed out that many specific continuous computations are inexpressible in finite, discrete formulas, it will be apparent that I have made full use of the tools of mathematics—including topology and functional analysis—to discuss continuous computation. This may seem odd, given that I have stated my intention to adopt the Continuity Principle and eschew discrete representations and processes. The reason, of course, is that I want to attain some precision in my statements and arguments. Nevertheless, the reader may be left with the impression that discrete representations and processes are somehow more fundamental than continuous. To explore this issue, it will be necessary to consider some developments in the history of mathematics.

Recall that in Euclid's *Elements* continuous magnitudes and discrete numbers are separately axiomatized; in effect they are taken to be equally fundamental. Nevertheless, mathematicians were more comfortable with the integers, perhaps because of troublesome issues of irrationality and infinity associated with the continuum. In any case, the "arithmetization of geometry" became a project in the development of mathematics, which was eventually declared solved as a consequence of the late-19th century constructions of Dedekind, Weierstrass, Cantor, and others. Therefore, we now routinely accept that the (continuous) real numbers are constructed in some way from the (discrete) rationals. The integers, from which the rationals are constructed, are considered most fundamental. Hence the historical importance of the recursive construction of the integers (e.g., by the Peano axioms) and of computation defined in terms of functions on the integers.

The reasons for this preference lie very deep, historically and psychologically, and are outside of the scope of this article. Nevertheless, it is relevant to indicate some of the issues involved. As is well known, the ancient Pythagoreans made use of figured numbers, that is, arrangements of identical tokens, to discover and prove theorems in number theory. Thus, square numbers were literally square figures, and so forth for triangular numbers, pentagonal numbers, etc. Typically the tokens were pebbles (Greek *psêphoi*, Latin *calculi*), and from the manipulation of these we get our words *calculus*, *calculate*, etc. These are the historical roots of the theory of discrete formal systems and of the TM model of computation.

A preference for the integers is just one aspect of a tendency to analyze complex phenomena into parts or units that are simple, elementary, atomic (literally,

"indivisible"), and nearly featureless (having only the simplest features, preferably quantized), but that have a definite identity (each unit is absolutely identical to itself and absolutely different from each other unit). Here also we may see the roots of modern particle physics (which traces its ancestry to the century after Pythagoras) and of genetics. There is much to recommend this view of the world (witness the success of modern science and technology), but it has less obvious limitations when applied to the complexity of the natural, especially the biological, world [e.g., 43].

It may be argued that mathematics has advanced far beyond the figured numbers of the Pythagoreans, or the crude axioms of Euclid, and that topology, for example, is able to describe spaces with varied and rich structures. But observe: mathematical topology is *point-set* topology. Topology is built on the concept of a space as set of points: self-identical and featureless, but each absolutely distinct from all other points (although, of course, they may be nearer or farther in some metric sense). Conceptually, continua are sets of points, functions and relations are sets of point pairs, and so forth. Mathematical points, indivisible tokens, conceptual atoms: they are all psychologically the same.

It may seem that there is no alternative, but that is because the point set approach to mathematics has been so successful. A function does not *have* to be viewed as a set of point pairs; at one time it was more common to understand it as a continuous curve or graph, and category theory treats functions and sets more holistically. Certainly the point-set approach is a triumph of generality, but it comes with a price, a fundamental atomic bias.

The point-set approach has not been accepted without criticism [34]; for example Karl Menger [35] provides a useful survey of various approaches to "topology without points" [36–38, 49–51]. His own approach begins with *lumps*, which are "closer to the physicist's concept of space" than are idealized points. Nevertheless, he concludes that

> even the introduction of points as nested sequences of lumps somehow transcends what can be observed in nature. For, by a lump, we mean something with a well defined boundary. But well-defined boundaries are themselves results of limiting processes rather than objects of direct observation. Thus, instead of lumps, we might use at the start something still more vague—something perhaps which has various degrees of density or at least admits a gradual transition to its complement. [35]

But let us dig deeper. Set theory is defined by some axiom system such as the Zermelo–Fraenkel axioms, which are typically expressed in a formal language such as first-order predicate logic with equality (FOPLE), in which equality is axiomatized with its familiar properties (reflexivity, symmetry, transitivity). As a consequence, the objects described by the Zermelo–Fraenkel axioms (be they interpreted as sets, functions, relations, numbers, or anything else) have the character of self-identical, mutually distinct atomic units.

It may be supposed that the equality axioms are the source of this character, but they only manifest it most clearly. In any consistent formal logical system, we will have some well-formed formulas that are provable and others that are not, and therefore induced relations of identity and non-identity in any valid domain of interpretation (model). (Indeed, the domain of interpretation is itself taken to be a mathematically

well-defined domain, which means that the identity of its objects will be definite.) The distinctness and definiteness of the tokens, types, and syntactic relations in our formal languages are inherently connected with the distinctness and definiteness of the mathematical objects (points, etc.) about which they can speak (express true propositions).

This is, I think, the implication of the Löwenheim–Skolem Theorem, which says that any consistent formal axiom system must have a countable model (valid domain of interpretation). (Such a model is constructed from the formal language and axiom system itself.) Thus a discrete formal system cannot escape definitively and absolutely from the discrete realm. In particular, the real continuum cannot be uniquely characterized in a discrete formal language. (In this sense, the historical project of arithmetizing geometry has failed.)

Now, my purpose is not to criticize mathematics, which is as important a tool in natural computation as in other scientific and engineering disciplines. Rather, I am trying to call attention to the fact that when we put on the spectacles of modern mathematics we are apt to see discreteness—"points"—even in continua, and we are apt to suppose that such continua, and the continuous processes operating on them, are completely and adequately describable by discrete formal systems.

Of course, mathematics is intended to be a language of precision, but the Löwenheim–Skolem Theorem and similar results hint that the very discreteness of formal syntax and inference may limit what it can express. However, one of the lessons of natural computation is that in many natural systems precision may be unnecessary and even detrimental. More generally, there are many kinds of information that are useful to organisms, and many ways of processing it; mathematics is just one kind, of limited applicability, primarily useful to a relatively small subset (scientists, etc.) of one species (*Homo sapiens*). That is, while the discrete formal language of mathematics may be useful for talking *about* natural computation, there is good reason for doubting that it is anything like the medium *of* natural computation.

8. Conclusions

We have seen that models are relative to a context of concerns; although they may be applicable outside of their historical context of origin, they cannot be assumed to be so. Further, using a model outside of its appropriate (but often indeterminate) domain runs the risk of deceiving us with incorrect results. Therefore we must expose the idealizing assumptions of a model and determine the extent to which they are applicable in any intended domain of application.

In particular, I have argued that the TM model owes its idealizing assumptions to issues in the formalist program in mathematics. Nevertheless, in part because the earliest digital computers were designed by scientists educated in this same background, the TM has proved reasonably successful as a model of traditional (especially batch-processing) digital computing.

However, as we have tried to apply computational models to nature, and as we have sought to design algorithms, computers, and robots inspired by biological systems, natural computation has emerged as an important area of concern, which asks

different questions and addresses different issues from the traditional theory of computation. In particular, the real-time response, flexibility, robustness, and adaptability of natural computation make continuous models of computation attractive. Therefore I have argued for a broadened definition of computation, which includes continuous representations and processes, on the basis that computation is a matter of what is being accomplished (manipulation of abstract form independently of material substrate), rather than of how it is accomplished (digital or analog technology). Continuous computation, in fact, contradicts many of the assumptions of the TM model; moreover it is better suited to addressing the issues of natural computation.

Finally, I indicated briefly that the contrast between discrete and continuous formal systems is related to deeper issues in epistemology and the foundations of mathematics.

Acknowledgements

I am grateful to Mark Burgin for directing me to Menger's very informative "Topology without points" [35].

References

[1] D.H. Ballard, An Introduction to Natural Computation, MIT Press, Cambridge, MA, 1997.
[2] L. Blum, M. Shub, S. Smale, On a theory of computation and complexity over the real numbers: NP completeness, recursive functions and universal machines, Bull. Amer. Math. Soc. 21 (1988) 1–46.
[3] M. Burgin, Universal limit Turing machines, Notices Russian Acad. Sci. 325 (1992) 654–658 (translated from Russian).
[4] M. Burgin, Topological algorithms, in: Proc. ISCA 16th Internat Conf.: Computers and their Applications, ISCA, Seattle, 2001, pp. 61–64.
[5] B.J. Copeland, The Church-Turing thesis, in: J. Perry, E. Zalta (Eds.), The Stanford Encyclopedia of Philosophy, http://plato.stanford.edu, 1996.
[6] B.J. Copeland, R. Sylvan, Beyond the universal Turing machine, Austral. J. Philos. 77 (1999) 46–67.
[7] J.G. Daugman, Spatial visual channels in the Fourier plane, Vision Res. 24 (1984) 891–910.
[8] J.G. Daugman, An information-theoretic view of analog representation in striate cortex, in: E.L. Schwartz (Ed.), Computational Neuroscience, MIT Press, Cambridge, MA, 1985, pp. 403–423.
[9] J.G. Daugman, Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters, J. Opt. Soc. Amer. A 2 (1985) 1160–1169.
[10] J.G. Daugman, Complete 2-D Gabor transforms by neural networks for image analysis and compression, IEEE Trans. Acoust. Speech Signal Process. 16 (1988) 1169–1179.
[11] H. Dreyfus, S. Dreyfus, Mind Over Machine, Macmillan, New York, 1986.
[12] S. Harnad, The symbol grounding problem, Physica D 42 (1990) 335–346.
[13] S. Harnad, Grounding symbols in the analog world, Think 2 (1993) 12–78.
[14] F. Hausdorff, Set Theory, Chelsea, New York, 1957 (transl. J.R. Aumann).
[15] B.J. MacLennan, Technology-independent design of neurocomputers: the universal field computer, in: M. Caudill, C. Butler (Eds.), Proc. IEEE 1 Internat. Conf, Neural Networks, Vol. 3, IEEE Press, New York, 1987, pp. 39–49.
[16] B.J. MacLennan, Logic for the new AI, in: J.H. Fetzer (Ed.), Aspects of Artificial Intelligence, Kluwer, Dordrecht, 1988, pp. 163–192.
[17] B.J. MacLennan, Field computation: a theoretical framework for massively parallel analog computation, Parts I–IV, Tech. Report CS-90-100, Department Computer Science University of Tennessee, Knoxville, 1990. http://www.cs.utk.edu/∼mclennan.

[18] B.J. MacLennan, Gabor representations of spatiotemporal visual images, Tech. Report CS-91-144, Dept. Comp. Sci., Univ. Tennessee, Knoxville, 1991. http://www.cs.utk.edu/∼mclennan.
[19] B.J. MacLennan, Characteristics of connectionist knowledge representation, Inform. Sci. 70 (1993) 119–143, http://www.cs.utk.edu/∼mclennan.
[20] B.J. MacLennan, Field computation in the brain, in: K.H. Pribram (Ed.), Rethinking Neural Networks: Quantum Fields and Biological Data, Lawrence Erlbaum, Hillsdale, 1993, pp. 199–232, http://www.cs.utk.edu/∼mclennan.
[21] B.J. MacLennan, Grounding analog computers, Think 2 (1993) 48–51, http://www.cs.utk.edu/∼mclennan or cogprints.soton.ac.uk/abs/comp/199906003.
[22] B.J. MacLennan, Continuous computation and the emergence of the discrete, in: K.H. Pribram (Ed.), Origins: Brain & Self-Organization, Lawrence Erlbaum, Hillsdale, 1994, pp. 121–151, http://www.cs.utk.edu/∼mclennan or cogprints.soton.ac.uk/abs/comp/199906001.
[23] B.J. MacLennan, Continuous symbol systems: the logic of connectionism, in: D.S. Levine, M. Aparicio IV (Eds.), Neural Networks for Knowledge Representation and Inference, Lawrence Erlbaum, Hillsdale, 1994, pp. 83–120, www.cs.utk.edu/∼mclennan.
[24] B.J. MacLennan, Words lie in our way, Minds Mach. 4 (1994) 421–437, http://www.cs.utk.edu/∼mclennan or cogprints.soton.ac.uk/abs/phil/199906001.
[25] B.J. MacLennan, Continuous formal systems: a unifying model in language and cognition, in: Proc. IEEE Workshop on Architectures for Semiotic Modeling and Situation Analysis in Large Complex Systems, Monterey, CA, 1995, pp. 161–172. http://www.cs.utk.edu/∼mclennan or cogprints.soton.ac.uk/abs/comp/199906002.
[26] B.J. MacLennan, Field computation in motor control, in: P.G. Morasso, V. Sanguineti (Eds.), Self-Organization, Computational Maps and Motor Control, Elsevier, Amsterdam, 1997, pp. 37–73, http://www.cs.utk.edu/∼mclennan.
[27] B.J. MacLennan, Field computation in natural and artificial intelligence, Inform. Sci. 119 (1999) 73–89, http://www.cs.utk.edu/∼mclennan.
[28] B.J. MacLennan, Can differential equations compute?, Tech. Report UT-CS-01-459, Department of Computer Science, University of Tennessee, Knoxville, 2001. http://www.cs.utk.edu/∼mclennan.
[29] B.J. MacLennan, Continuous information representation and processing in natural and artificial neural networks, Tech. Report UT-CS-03-508, Department of Computer Science, University of Tennessee, Knoxville, 2003. http://www.cs.utk.edu/∼mclennan.
[30] B.J. MacLennan, Transcending Turing computability, Minds Mach. 13 (2003) 3–22, http://www.cs.utk.edu/∼mclennan.
[31] A.A. Markov, Theory of Algorithms, transl. by J.J. Schorr-Kon, PST Staff, Israel Prog. Sci. Transl. US Department Comm. Ofc. Tech. Service OTS 60-51085, Jerusalem, 1961. (Transl. of Teoriya Algorifmov, Acad. Sci. USSR, Moscow, 1954).
[32] C. Mead, Analog VLSI and Neural Systems, Addison-Wesley, Reading, MA, 1989.
[33] B. Mendelson, Introduction to Topology, 2nd ed., Dover, New York, 1990.
[34] K. Menger, Dimensionstheorie, Teubner, Leipzig, 1928.
[35] K. Menger, Topology without points, Rice Inst. Pamphlet 27 (1940) 80–107.
[36] A.N. Milgram, Partially ordered sets, separating systems and inductiveness, Rep. Math. Colloq. Notre Dame Ind. 2 Ser. 1 (1939) 18–30.
[37] A.N. Milgram, Partially ordered sets and topology, Rep. Math. Colloq. Notre Dame, Ind. 2nd Ser. 1 (1940) 3–9.
[38] A.N. Milgram, Partially ordered sets and topology, Proc. Natl. Acad. Sci. USA 26 (1940) 291–293.
[39] C. Moore, Recursion theory on the reals and continuous-time computation, Theoret. Comput. Sci. 162 (1996) 23–44.
[40] T.O. Moore, Elementary General Topology, Prentice-Hall, Englewood Cliffs, NJ, 1964.
[41] V.V. Nemytskii, V.V. Stepanov, Qualitative Theory of Differential Equations, Dover, New York, 1989.
[42] M.B. Pour-El, Abstract computability and its relation to the general purpose analog computer (Some connections between logic, differential equations and analog computers), Trans. Amer. Math. Soc 199 (1974) 1–29.

[43] T. Roszak, The Gendered Atom: Reflections on the Sexual Psychology of Science, Conari Press, Berkeley, CA, 1999.
[44] L.A. Rubel, A universal differential equation, Bull. (NS) Amer. Math. Soc. 4 (1981) 345–349.
[45] L.A. Rubel, The extended analog computer, Adv. Appl. Math. 14 (1993) 39–50.
[46] C.E. Shannon, Mathematical theory of the differential analyzer, J. Math. Phys. MIT 20 (1941) 337–354.
[47] R.N. Shepard, Form, formation, and transformation of internal representations, in: R.L. Solso (Ed.), Information Processing in Cognition: The Loyola Symposium, Lawrence Erlbaum, Hillsdale, 1975.
[48] R.N. Shepard, J. Metzler, Mental rotation of three-dimensional objects, Science 171 (1971) 701–703.
[49] M.H. Stone, The theory of representation for Boolean algebras, Trans. Amer. Math. Soc. 40 (1936) 37–111.
[50] M.H. Stone, Applications of the theory of Boolean rings to general topology, Trans. Amer. Math. Soc. 41 (1937) 375–481.
[51] H. Wallman, Lattices and topological spaces, Ann. Math. (Ser. 2) 39 (1938) 112–126.