International Journal of General Systems

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ggen20

The promise of analog computation

B.J. MacLennan[a]

[a] Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, USA.

Published online: 29 May 2014.

To cite this article: B.J. MacLennan (2014): The promise of analog computation, International Journal of General Systems, DOI: 10.1080/03081079.2014.920997

To link to this article: http://dx.doi.org/10.1080/03081079.2014.920997

International Journal of General Systems, 2014
http://dx.doi.org/10.1080/03081079.2014.920997

Taylor & Francis
Taylor & Francis Group

The promise of analog computation

B.J. MacLennan*

Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, USA

(Received 12 November 2013; accepted 11 December 2013)

Future computing paradigms and technologies will have to be more like the physical processes by which they are realized, and because these processes are primarily continuous, post-Moore's law computing will involve an increased use of analog computation. Traditionally analog computers have computed ordinary differential equations of time, but analog field computation permits massively parallel temporal integration of partial differential equations. In principle many different physical media – not just electronics – can be exploited to implement the basic operations of analog computing, a small number of which are sufficient to approximate a wide variety of analog computations, thus providing a basis for universal analog computation and general-purpose analog computers. The contentious issue of the computational power of analog computers is addressed best on its own terms, rather by asking it within the context of Church-Turing computation, which distorts the relevant questions and their answers.

Keywords: analog computation; continuous computation; field computation; Moore's law; post-Moore's law computation

1. Post-Moore's law computing

Computing technology has benefitted from Moore's law, which says that VLSI density doubles approximately every two years (Moore 1965). However, Moore's law is not a law of nature, but a consequence of decreasing the linear dimension of VLSI circuit elements and their operating voltages (Frank 2005). The energy to change a bit is given by $E = CV^2$, where V is voltage and C is capacitance, which is proportional to area, i.e. linear dimension squared. However, there are practical limits to the continuation of Moore's law. Charge cannot be less than one electron; device dimensions cannot be less than one atomic diameter. Also, operation becomes unreliable if E is much less than $10\,kT$, where k is Boltzmann's constant and T is the temperature of the computing environment (Frank 2005). This minimum energy is about 2.6 electron-volts at room temperature.

During the reign of Moore's law we have had the luxury of multiple hierarchical levels to implement conveniently programmable von Neumann architectures. For example, real numbers are represented by floating-point numbers, comprising dozens of bits, each implemented by multiple semiconductor devices, each controlling large numbers of electrons. Likewise, elementary operations, such as division, are implemented by sequential algorithms, in turn implemented by sequential digital logic, in which each binary operation involves the saturation of a physical quantity (such as a charge or voltage). As we approach the limits of Moore's law, we cannot afford these deep hierarchical representations. Rather, in post-Moore's law computing, our computational processes and the physical processes by which they are implemented must become more alike. Since the laws of physics are what they are, we conclude that in the post-Moore's law regime,

*Email: maclennan@utk.edu

computational processes will be more like the physical processes by which they are realized. Since most physical laws are continuous (i.e. differential equations), this implies an increasing role for analog computing in the future.

2. Meaning of analog computing

The term "analog compution" is usually explained in terms of an analogy existing between physical quantities in the computer and quantities in the *primary system* (the system being modelled, analysed or controlled by the computer). For example voltage in the computer might be used to represent water pressure in the primary system. In fact, in most analog computers the computational and primary quantities have been related by a simple proportion, so that computational operations correspond directly to physical processes. More generally, quantities can be represented less directly (e.g. logarithmically), as long as there is a systematic (quantitative) analogy between the computational and primary systems. According to this definition, however, ordinary digital computers are also analog computers, because there is a systematic relationship between internal representations and processes in the computer and those in the primary system. It just so happens that the relationship is much more complicated than a simple proportion.

Therefore, the essential characteristic of analog computing, as generally understood, is not the existence of an analogy between the computational and primary processes, since digital computation also has this characteristic. Rather, the defining characteristic of analog computing is its use of continuous representations, as opposed to digital computation, which uses discrete representations. This has become the common understanding of these terms, as for example when we contrast digital and analog clocks or analog and digital music recording.

I have argued that post-Moore's law computing will require computing processes to be more like the physical processes that realize them. In particular since most of the laws of physics are continuous, we will have to make greater use of analog computation. Therefore we will have to leave our technological comfort zone (binary electronics) and explore a wider range of physical processes that can be exploited for computation.

Our goal in computation is to use abstract or mathematical relationships to model, analyse or control some primary system. Since computation is a physical process, this is accomplished by using a physical system (the computer) to implement or physically instantiate these abstract relationships. More precisely, we can say that there must be an *approximate homomorphism* from the physical system to the abstract system (MacLennan 1994, 2003, 2004). It is a *homomorphism* because the physical system has at least the structure of the abstract system, but also has additional, irrelevant structure. The homomorphism is *approximate* since a physical system cannot usually exactly realize the intended abstract system. For example, the mathematical system might be linear, but its realization might be linear in only a restricted range or linear only within a specified tolerance.

3. Topology of computation

I have argued, in agreement with common usage, that the essential difference between digital and analog computation is between discrete and continuous representations. Therefore the distinction is effectively topological, and so it will be useful to look at computation in topological terms.

3.1. State space

First we can classify computation in terms of its *state space*, that is, the space in which information is represented.

Digital computation makes use of a *discrete space*, in topological terms, a space in which there are only two fundamental distances: 0 and 1. The distance of a discrete point from itself is 0 and its distance from any other discrete point is 1. That is, the metric tells you if the points are the same or different. This is the foundation of binary representation; the essential property of 0 and 1 is that they can be mechanically and reliably distinguished. The larger alphabets used in other models of digital computation, such as Turing machines and the lambda calculus, also belong to a discrete topology: their essential property is the identity or difference of the symbol types. Of course, for computational purposes we can define additional metrics over discrete spaces, as when we arrange bits to implement integers or floating point numbers, but this additional structure is not inherent in these spaces of discrete symbols.

Traditionally, analog computation is computation over real variables. More generally, the state space of an analog computation is a topological *continuum*, which is non-empty connected compact metric space. For example, closed intervals of the reals, with the usual topologies, are continua. Analog computation is sometimes defined over complex variables or complex-valued *fields* (continuous spatial distributions of continuous quantity). Non-trivial analog computations operate on multiple continua (e.g. multiple real variables), but finite (and countably infinite) arrays of continua form continua under the product topology. In summary, the state spaces of analog computations are topological continua.

Since hybrid analog – digital computation is useful for some purposes, an interesting question is whether there is a more general model of computation that subsumes analog and digital computation as special cases. Elsewhere I have argued that second-countable Hausdorff spaces are a reasonable topology for models of hybrid computation over both discrete and continuous state spaces (MacLennan 2010).

3.2. Processes

Computations take place in time, but traditional analog computations evolve continuously in time whereas discrete computations proceed by discrete steps. Therefore, we need to consider the temporal topologies of computation.

Ordinary digital computation takes place in *sequential time* (van Gelder 1997); that is, there is a sequence of computational steps of unspecified, but finite duration. In practical terms, of course, we expect the operations to be rapid, but in the theory of digital computation, we require only that the individual steps be finite.

It is also possible to do analog computation in sequential time: at each step in the computation some analog operation is applied to continuous quantities to yield a continuous quantity. For example, two real numbers might be fetched, multiplied by analog circuitry and the product stored in a real variable.

Computation in discrete steps – whether analog or digital – need not be purely sequential. More generally we can have a partial order over a discrete set of steps, thus allowing parallel computation.

Sequential-time computation should not be confused with *discrete-time* computation, which is a special case in which the computational steps occur at some definite time interval. Discrete-time computation is in effect computation over the natural numbers, whereas sequential computation is over a discrete total order (or, in the parallel case, a discrete partial order). Discrete-time computations can be analog or digital, and are often used to approximate continuous-time analog computation.

Traditionally analog computers operate in continuous time, that is, in accord with differential equations. Typically, time in the computation is proportional to time in the primary system (either slower, faster or identical, as required by the application), but in principle other continuous,

monotonic relationships could be used. In this article I will focus on continuous-time analog computation, since it is the more common and more general case.

4. Precision and accuracy

In discussing analog computation it is important to distinguish *accuracy* and *precision*. *Accuracy* refers to the mathematical correctness of the operations, for example, the closeness between a mathematical model and the primary system it models or between the intended and actual behaviour of a controlled system. Accuracy depends on many factors, including the completeness of the model and the approximations used in the algorithms. One of these factors is *precision*, which refers to the quality of representations and of computations on them; it depends primarily on two factors: *resolution* (fineness of representation) and *stability* (absence of drift or decay). The precision of a device is usually expressed as a fraction or percentage of *full-scale variation*, that is, of the maximum representable range of values (e.g. a precision of 0.01% or 10^{-4}).

One of the advantages of digital computation is the incremental cost of improving precision. For example, doubling the precision of a number requires adding only one additional bit to a register. On the other hand, doubling the precision of an analog device may require much more expensive materials and manufacturing techniques. Analog cost tends to be exponential in precision, whereas digital is linear or at worst quadratic. This is because digital representations use multiple low-precision (i.e. binary) devices to represent a single quantity, whereas analog computation makes use of a single high-precision device.

The relatively greater cost of analog precision was one factor in the spread of digital technology in the late twentieth century, but recently analog technology has been regaining its advantage, at least for some applications. First, as discussed in Section 1, post-Moore's law computing does not permit us the luxury of using many low-precision devices to represent a single quantity; that wastes space and energy. Second, high precision is not required in many important applications of analog computation, such as signal processing and control. Furthermore, some approaches to analog computation, such as artificial neural networks, do not require high precision representation or computation in order to produce accurate results. Low-precision devices can represent quantities with high precision through *coarse coding* (Rumelhart, McClelland, and the PDP Research Group 1986, 91–96; Sanger 1996). That is, high precision is achieved through a population of low-precision devices. (Indeed, individual neurons seem to operate with less than 10% precision.) Finally, some contemporary analog technologies can achieve quite high precision. For example, there are analog/digital converters with 24 bits of precision, and analog computation with 1% to 0.1% precision is easy to achieve with modest power requirements.

5. Analog computing operations

In this Section 1 will discuss some common primitive operations for analog computing. It will be apparent in most cases that they have relatively straight-forward physical realizations.

The simplest analog computation operations combine real or complex values algebraically: $u(t) = v(t) \pm w(t)$, $u(t) = v(t)w(t)$, $u(t) = v(t)/w(t)$. (In the latter case, division, the range of the output must be limited.) Other common algebraic operations include constant multiplication, $u(t) = cv(t)$, inversion, $u(t) = -v(t)$, and magnitude, $u(t) = |v(t)|$. Analog devices can often implement special functions efficiently, for example, $u(t) = \ln v(t)$, $u(t) = \exp v(t)$, $u(t) = \cos v(t)$. Integration is one of the most useful analog computations: $u(t) = u_0 + \int_0^t v(t)\mathrm{d}t$; it can be realized straight-forwardly by many physical processes. Differentiation, $u(t) = \dot{v}(t)$, is also easy to implement, but it must be used with caution, since noise tends to be high frequency and therefore differentiation amplifies noise. In practice a low-pass filter can be applied before

differentiation. Some analog computers provide tunable bandpass filters, which can be used to measure a signal's power in a set of bands and therefore to compute a discrete Fourier transform of the signal. Others provide programmable analog matrix-vector multipliers, which can be used to implement linear operators, such as filters.

Sometimes it is necessary to compute a function $u(t) = F[v(t)]$, where the function F might have no known closed form. For example, F might be an *empirical function* resulting from the measurement of some phenomena (e.g. physiological or psychological responses). Or it might be that F is simply too complex to compute in terms of basic operations. In this case analog computers can use a continuous version of table lookup. In effect the function's graph can be "drawn" (plotted) in a special medium that computes F directly (Truitt and Rogers 1960, 1-72–81, 2-154–156). In some cases arbitrary functions can be computed by analog interpolation based on finite samples $(v_k, F(v_k))$. There have been analog devices capable of computing arbitrary functions of two variables, $u(t) = F[v(t), w(t)]$ defined by their graphs or by samples, $u_k = F(v_k, w_k)$.

The foregoing is a sample of basic analog computing operations, but any physical process that is mathematically describable may be used to model processes with that same description. For example, an operation that adds noise to a signal, $u(t) = v(t) + \nu(t)$, can be used to model stochastic processes in the primary system (Howe 1961, ch. 7). Noise (randomness) is also useful in some algorithms, such as stochastic resonance and simulated annealing (Benzi et al. 1982; Kirkpatrick, Gellat, and Vecchi 1983). Another useful operation is a delay, $u(t) = v(t - T)$, which can be used to model delays in the primary system or to approximate time derivatives: $\dot{v}(t) \approx [v(t) - v(t - \Delta t)]/\Delta t$. Future progress will include the exploitation of additional physical processes that can be applied to analog computation.

In any given analog computing technology, some operations will be easy and efficient to implement, and others less so, which raises the question of whether there is a minimal set of operations that are universal for analog computing. In fact, for many purposes, addition, scalar multiplication, integration and functon generation are sufficient, as explained in more detail below (Section 7.1).

6. Field computation

6.1. Definition

Historically, most analog computers have been used to integrate ordinary differential equations (ODEs) in which time is the independent variable. However, there have also been analog computers that can integrate, with respect to time, partial differential equations (PDEs) defined over one or two spatial dimensions. This has been called the *field analogy method* (Kirchhoff 1845). Computational devices made use of physical continua or spatially discrete approximations to them. There is a long history of optical field computation (Ambs 2010), and more recently Jonathan Mills has investigated the use of physical continua for analog computation (Mills 1996; Mills et al. 2006).

We define a *phenomenological field* to be a spatially continuous distribution of continuous quantity (typically real or complex) or a discrete distribution that can be usefully treated as though continuous. A phenomenological field is described mathematically by a bounded continuous function $\phi : \Omega \to K$. The domain Ω is a closed and bounded continuum (the space over which the field is defined); physical realizability normally limits it to a region in 1-, 2- or 3-dimensional Euclidean space, although there are techniques for computing over higher dimensional fields (MacLennan 2009b). The codomain K is typically a space of real or complex scalars, vectors or tensors. We write ϕ_p for $\phi(p)$, the value of ϕ at a point $p \in \Omega$, and time-varying fields are notated $\phi(t)$, $\phi_p(t)$, etc. when we want to make the time dependence explicit. $\Phi_K(\Omega)$ is the space of all K-valued fields over domain Ω and we omit K when it is the reals.

Mathematically, fields are functions, and so $\Phi_K(\Omega)$ is a function space, and it is convenient to treat it as a Hilbert space. This requires that fields be "finite energy" (square-integrable), which is reasonable for physically realizable fields. This also requires that the space be complete, which implies that it contains physically unrealizable but mathematically useful limit fields, such as *step functions* and *unit impulses* (Dirac delta functions).

6.2. Pointwise operations

The simplest field computation operations are simply pointwise field extensions of scalar operations. For example, $\psi = \phi \pm \chi$, which means the *pointwise* computation $\psi_p(t) = \phi_p(t) \pm \chi_p(t)$. Likewise we have $\psi = \phi \times \chi$, $\psi = \phi/\chi$ (suitably limited), $\psi = u\phi$ (scalar multiplication), $\psi = \ln \phi$, $\psi = \exp \phi$, etc. We also have pointwise integration with respect to time, $\psi_p(t) = \upsilon_p + \int_0^t \phi_p(\tau)\mathrm{d}\tau$, and differentiation with respect to time, $\psi_p(t) = \dot{\phi}_p(t)$. For vector and tensor fields, we have pointwise inner, outer and cross-products, which may be implemented in terms of scalar field operations. Other operations are specific to fields. For example, a field can be converted to a scalar by definite integration over space, defint $\phi(t) = \int_\Omega \phi_p(t)\mathrm{d}p$, which can be used for computing the average value of a field.

6.3. Spatial operations

One especially useful operation, which has efficient analog realizations, is *spatial convolution*, $\phi * \psi$, where $(\phi * \psi)_p = \int_\Omega \phi_{p-q}\psi_q \mathrm{d}q$. Closely related is the *spatial cross-correlation* operation, $\phi \star \psi$, which is defined $(\phi \star \psi)_p = \int_\Omega \phi_q^* \psi_{p+q} \mathrm{d}q$, where we include the complex conjugate operation "*" in case the fields are complex.

The gradient operation, $\psi = \nabla\phi$, returns a vector field, which can be represented directly by a physical vector field or by a finite array of physical scalar fields. It can be used in many useful computations, such as optimization or learning by gradient ascent or descent.

Another useful field operation is the Laplacian, $\psi = \nabla^2\phi$, which can be approximated by convolving with an appropriate kernel (e.g. a derivative-of-Gaussian field) or more directly in some realizations by diffusion in the computational medium. It is especially useful in *reaction–diffusion computing*, which combines diffusion (often implemented directly by physical diffusion) with non-linear pointwise reactions; for example,

$$\dot{\phi} = k_1\phi^2/\psi - k_2\phi + d_1\nabla^2\phi,$$
$$\dot{\psi} = k_3\phi^2 - k_4\psi + d_2\nabla^2\psi$$

leads to the self-organization of *Turing patterns* (Turing 1952). The operations are pointwise (local), except for the Laplacian. Reaction–diffusion computation has been applied to a variety of problems (Adamatzky, De Lacy Costello, and Asai 2005).

Just as vector and matrix products are useful for computing in finite-dimensional spaces, there are analogous products on fields. Consider two fields, $\Psi \in \Phi(\Omega' \times \Omega)$ and $\phi \in \Phi(\Omega)$, analogous to a matrix and a vector. We define the *field product* $\Psi\phi \in \Phi(\Omega')$ by the *Hilbert-Schmidt integral operator*:

$$(\Psi\phi)_x = \int_\Omega \Psi_{xy}\phi_y \mathrm{d}y.$$

The field Ψ is called the *kernel* of the linear operator $L(\phi) = \Psi\phi$. This operation is the analogue of a matrix-vector product; the analogues of vector-matrix and matrix-matrix products are defined in the obvious way. We also define products among more than two fields, so long as they have compatible domains. For example, $\Psi\phi\psi = (\Psi\phi)\psi$ for $\Psi \in \Phi(\Omega'' \times \Omega' \times \Omega)$, $\phi \in \Phi(\Omega)$, and $\psi \in \Phi(\Omega')$.

Analogous to the outer product of finite-dimensional vectors, we have a *field outer product*. For $\phi \in \Phi(\Omega)$ and $\psi \in \Phi(\Omega')$, the outer product $\phi \wedge \psi \in \Phi(\Omega \times \Omega')$ is defined $(\phi \wedge \psi)_{(x,y)} = \phi_x \psi_y$ for $x \in \Omega$ and $y \in \Omega'$. The outer product is useful for neural network-style learning algorithms and for computing Taylor series approximations to field transformations. (It is closely related both to the Dirac outer product or dyad, $|\phi\rangle\langle\psi|$ and to the tensor product $|\phi\rangle \otimes |\psi\rangle$ familiar from quantum mechanics.)

6.4. *Orthonormal basis fields*

An important property of Hilbert spaces is that they have countable bases, which are spanning sets of orthonormal fields, $\beta_1, \beta_2, \ldots$. Indeed, physically realizable fields have finite bases; they are finite-dimensional. Any field in the space can be represented uniquely as a linear combination of basis fields, $\phi = \sum_k c_k \beta_k$. For working with bases it is convenient to have an inner product operation on fields, for which we use Dirac's bracket notation:

$$\langle \phi \mid \psi \rangle = \int_\Omega \phi_p^* \psi_p \mathrm{d}p,$$

where we include the complex conjugate "*" in case the fields are complex. As usual, ϕ and ψ are *orthogonal* fields if $\langle \phi \mid \psi \rangle = 0$, and the *norm* of a field is defined by $\|\phi\| = \sqrt{\langle \phi \mid \phi \rangle}$. An orthonormal set of fields $\{\beta_k\}$ satisfies $\|\beta_k\| = 1$ and $\langle \beta_j \mid \beta_k \rangle = 0$ for $j \neq k$. Any field can be expanded as a *generalized Fourier series* in terms of a basis, $\phi = \sum_k c_k \beta_k$, where the generalized Fourier coefficients are $c_k = \langle \beta_k \mid \phi \rangle$. The Fourier series for realizable fields have a finite number of terms, so in many analog computations field operations can be replaced by operations on a finite set of scalar variables.

Linear field operators can be computed in terms of their Fourier coefficients. Suppose that $L : \Phi(\Omega) \to \Phi(\Omega')$ and that $\{\beta_k\}$ is a basis for $\Phi(\Omega)$ and $\{\alpha_j\}$ is a basis for $\Phi(\Omega')$. Let $c_k = \langle \beta_k \mid \phi \rangle$ be the Fourier coefficients of the input $\phi \in \Phi(\Omega)$. Then the Fourier coefficients $d_j = \langle \alpha_j \mid L\phi \rangle$ of the output $L\phi \in \Phi(\Omega')$ can be computed by an analog matrix-vector multiplication $\mathbf{d} = \mathrm{M}\mathbf{c}$, where the matrix elements are given by $M_{jk} = \langle \alpha_j \mid L\beta_k \rangle$.

Fields defined over more than two or three spatial dimensions may be physically unrealizable, but they can be represented as one-dimensional fields by means of their generalized Fourier coefficients. Let $\{\beta_k\}$ be a basis for $\Phi(\Omega)$ and $\{\xi_k\}$ a basis for $\Phi([0, 1])$. Then $\mathrm{H} = \sum_k \xi_k \wedge \beta_k^*$ will encode a field over Ω as a field over $[0, 1]$. Likewise, if $\{\alpha_j\}$ is basis for $\Phi(\Omega')$, then $\Theta = \sum_j \alpha_j \wedge \xi_j^*$ will decode a representation over $[0, 1]$ into a field over Ω'. If $K \in \Phi(\Omega' \times \Omega)$ is the kernel of a linear operator, then it can be replaced by a two-dimensional field $L \in \Phi([0, 1]^2)$ by $K = \Theta L \mathrm{H}$.

6.5. *Non-linear computation via topographic maps*

In many regions the brain represents information *topographically*; that is, there is systematic relationship between the value to be encoded and the location in the cortex where it is represented (Knudsen, du Lac, and Esterly 1987). These topographic representations provide an efficient way to compute non-linear operations while encoding ancillary pragmatic factors, which is easily accomplished through analog field computation (MacLennan 1997, 2009b). Suppose S is a space to be encoded (e.g. pitch of sound, orientation and location of visual edges) in a field defined over Ω. We represent the encoding by a map $\mu : S \to \Omega$. Then a particular value $s \in S$ will be represented by a field $\alpha[\mu(s)] \in \Phi(\Omega)$ with its amplitude concentrated around $\mu(s)$. (In the ideal case, it is a Dirac delta function, $\alpha_p[\mu(s)] = \delta[p - \mu(s)]$.)

Likewise, the result of a function $f : S \to T$ can be represented by a map $\nu : T \to \Omega'$. Our goal is that a function value $f(s) \in T$ will be represented by a field $\beta(\nu[f(s)]) \in \Phi(\Omega')$ with

its amplitude concentrated around $\nu[f(s)]$. The function can be computed topographically by a linear operator whose kernel is:

$$K = \int_S \beta(\nu[f(s)]) \wedge \alpha[\mu(s)]\mathrm{d}s \in \Phi(\Omega' \times \Omega).$$

This is effectively a blurred graph of f. The effect of applying the linear operator K to the representation $\alpha[\mu(s')]$ of a value $s' \in S$ is:

$$K\alpha[\mu(s')] = \int_S \beta(\nu[f(s)])\ \langle \alpha[\mu(s)] \mid \alpha[\mu(s')] \rangle\ \mathrm{d}s.$$

That is, the overlaps between the $\alpha[\mu(s)]$ and the input representation $\alpha[\mu(s')]$ weight the corresponding output representations $\beta(\nu[f(s)])$.

Since the *location* of activity $\mu(s)$ represents the input value $s \in S$, the *amplitude* of the field can represent ancillary, pragmatic information p, such as the importance or probability of the input, which weights the corresponding outputs: $K(p\alpha[\mu(s)]) = p(K\alpha[\mu(s)])$. Likewise, because the computation is linear, topographic computation can transform a superposition of inputs into a superposition of outputs, $K(p\alpha[\mu(s)] + q\alpha[\mu(s')]) = p(K\alpha[\mu(s)]) + q(K\alpha[\mu(s')])$.

7. General-purpose analog computers

The notion of a *general-purpose computer* or *universal machine* has both practical and theoretical importance. In the realm of digital computing we have the theoretical Turing machine and its equivalents as well as the practical programmable computers we all use. Likewise, in analog computing there have been general-purpose analog computers (GPACs) both as practical instruments and as objects of theoretical investigation. Practical GPACs of the past include Vanevar Bush's mechanical differential analyser (1930) and the Rockefeller Differential Analyser (1947) and a number of commercial electronic GPACs, which emerged in the 1950s (Small 2001, 42–5, 72–3). GPACs are mostly of theoretical interest now – for determining the limits and fundamental requirements for analog computation – but practical GPACs may become important in the future in situations where special purpose analog hardware or digital simulation are not feasible.

How should universal analog computing be defined? In the theory of digital computation, the Church-Turing Hypothesis asserts that our intuitive notion of effective calculability corresponds to Turing computability. The credibility of this hypothesis rests upon the fact that multiple independent attempts to formalize effective calculability all arrived at equivalents to Turing computability. At this time, however, we have no corresponding agreed upon notion of analog computability. Nevertheless, there are theoretical results useful for the design of GPACs.

7.1. Shannon's analysis of the differential analyser

Claude Shannon studied the theoretical power of the differential analyser, but his results are also applicable to other GPACs (Shannon 1941, 1993). (I will informally summarize his conclusions; see the original paper for details.) He assumed a GPAC capable of addition, constant multiplication, integration and function generation (for functions with a finite number of finite discontinuities), with one source of drive (which limits interconnection), and proved that it would be capable of computing a function if and only if it was not hypertranscendental (therefore, including all algebraic transcendental functions). This is a very large class of functions, but does not include the Riemann zeta function or Euler's gamma function, for example. The non-hypertrancendental functions provide a basis for computing a large number of additional functions, which can be derived from the former by composition, inversion, differentiation, integration, etc. Functions of

multiple variables can be computed if they can be defined in terms of computable functions by means of partial differentiation or inversion with respect to any one variable. These results were extended to show that a finite number of adders and integrators could approximate the computation of any multivariable function that is continuous over a closed region of space. Shannon also showed that his GPAC was capable of solving any system of ordinary differential equations definable in terms of non-hypertranscendental functions.

Shannon's proofs were incomplete, but they were later refined and corrected by (Pour-El 1974) and (Lipshitz and Rubel 1987). For example, the *Shannon-Pour-El Thesis* states that the GPAC can solve initial-value problems for *algebraic differential equations*: equations of the form $P[u, v(u), v'(u), v''(u), \ldots, v^{(n)}(u)] = 0$, in which P is a polynomial that is not identically vanishing in any of its variables (Rubel 1985). However, Rubel (1988) showed that it could not solve the Dirichlet problem for Laplace's equation on the disk.

7.2. *Rubel's extended analog computer*

Motivated in part by his view of the brain as an analog computer (Rubel 1985), Rubel has developed an extension of the GPAC, a conceptual machine that he calls the *Extended Analog Computer* or EAC (Rubel 1993). For example, whereas the GPAC integrates only over time, the EAC is able to integrate over any finite number of real variables. It has a series of levels, each of which uses operations such as adders, multipliers, differentiators, inverters, function composers and analytical continuation devices to solve equations defined over the functions computed by the layer below. The lowest level provides for real polynomials and implements the differentially algebraic functions (like the differential analyser). The EAC is not restricted to initial-value problems, but has an analog device capable of solving ODE and PDE boundary-value problems. Powerful analog computation devices of this sort might seem infeasible, so it is important to recall, as Rubel notes, that they can be implemented by physical processes that obey the same ODEs or PDEs, as was done in the old field analogy method. Jonathan Mills has demonstrated analog field computation devices (Mills 1995, 1996; Mills et al. 2006).

7.3. *Universal approximation theorems*

Another, quite practical approach to general-purpose analog computation makes use of various universal approximation theorems (Haykin 1999, Sections 4.13, 5.3, 5.5, 5.6 and 5.10). For example, any continuous function $F(v_1, \ldots, v_n)$ on bounded variables $v_1, \ldots, v_n$ can be approximated arbitrarily closely by an expression of the form:

$$F(v_1, \ldots, v_n) = \sum_{i=1}^{m} a_i \sigma \left(\sum_{j=1}^{n} w_{ij} v_j + b_i \right),$$

where a_i, b_i and w_{ij} are fixed coefficients (dependant on F) and σ is any non-constant, monotone increasing, bounded, continuous function. Typically it is a *sigmoid function*, such as the logistic sigmoid, $\sigma(x) = [1 + \exp(-x)]^{-1}$. In analog computing, such functions often come "for free" (e.g. as a side-effect of saturation in the underlying physical processes). Of course, such an approximation is equivalent to a two-layer artificial neural network with interconnection weights a_i and w_{ij} and biases (inverse thresholds) b_i. The coefficients can be determined by a simple least-squares minimization (Haykin 1999, Sections 5.3, 5.5–5.6 and 5.10). The same approach can be used for approximating arbitrary field transformations:

$$F(\phi_1, \ldots, \phi_n) = \sum_{i=1}^{m} \alpha_i \times \overline{\sigma}\left(\sum_{j=1}^{n} \omega_{ij} \times \phi_j + \beta_i\right),$$

where the α_i, ω_{ij} and β_i are fixed fields, "$\times$" represents pointwise multiplication of two fields and $\overline{\sigma}$ is pointwise application of the sigmoid function σ. Useful special cases arising from interpolation theory include approximations of the form: $F(\phi) = \sum_{i=1}^{m} \alpha_i \sigma(\langle \omega_i \mid \phi \rangle + \beta_i)$, where the α_i, ω_i and β_i are fixed fields. This is the field equivalent of an artificial neural network. Another useful approximation for field transformations has the form: $F(\phi) = \sum_{i=1}^{m} \alpha_i r(\|\phi - \eta_i\|)$, where the α_i and η_i are fixed fields and $r : \mathbb{R} \to \mathbb{R}$ is a monotonically decreasing function. Thus the $r(\|\phi - \eta_i\|)$ are radial basis functions (Haykin 1999, 264–265). The fixed fields for these approximations can be determined by least-squares minimization (MacLennan 2009b).

Another approach to universal field computation makes use of Taylor's theorem on Hilbert spaces, which allows a field transformation to be expanded as a sum of field products (MacLennan 1987, 2009b). This allows a field transformation $F : \Phi(\Omega) \to \Phi(\Omega')$ to be expanded around a fixed field $\phi \in \Phi(\Omega)$ by Horner's rule: $F(\phi + \alpha) \approx T(\phi) + Q_1(\phi, \alpha)\alpha$, where

$$Q_k(\phi, \alpha) = D_k(\phi) + \frac{1}{k+1} Q_{k+1}(\phi, \alpha)\alpha \quad (k \geq 1),$$

where $D_k(\phi)$ is the kernel of the kth functional derivative of F at ϕ.[1] $D_k \in \Phi(\Omega' \times \Omega^k)$, but these higher dimensional fields can be avoided by representing them by generalized Fourier series (Section 6.4). This shows that a large class of field transformations can be approximated by means of the field product (Hilbert–Schmidt integral), pointwise addition and scalar multiplication. More generally, it illustrates a kind of polynomial approximation of field transformations.

8. Theoretical power of analog computing

A perennial question is the theoretical power of analog computation compared to the Turing machine; are analog computers capable of computing "beyond the Turing limit"? On the one hand, it is easy to show that analog computers' ability to use real numbers (with infinite precision) confers super-Turing power. On the other, analog computers are routinely simulated on ordinary digital computers, which suggest that analog computation in no more powerful than Turing computation. More careful analyses only deepen the paradox; representative examples include Blum et al. (1998), Bournez and Cosnard (1996), Bournez et al. (2006), Branicky (1994), Davis (2006), Franklin and Garzon (1990), Garzon and Franklin (1990), Maass and Sontag (1999), Orponen (1997), Orponen and Matamala (1996), Moore (1996), Omohundro (1984), Pour-El and Richards (1979, 1982), Siegelmann (1999), Siegelmann and Sontag (1994), Stannett (1990), Wolpert and MacLennan (1993).

The root of the paradox is that the Turing machine is a *model* of computation and like all models it makes simplifying or idealizing assumptions. Any model has an associated *frame of relevance*, which is the domain of questions that it is suited to answer and which depends on its simplifying assumptions (MacLennan 2004). We must remember that the Church-Turing model of computation was invented as a way of studying effective calculability and formal derivation in mathematics, which dictated many of its simplifying assumptions, in particular, the use of finite discrete alphabets and discrete sequential steps. In addition, it is generally assumed that symbols and states are perfectly distinguishable and that operation of the machine is flawless. Performance is analysed in terms of the *number* of sequential steps or atomic symbols used by a computation.

In general, these simplifying assumptions are inappropriate for analog computing and many of the interesting questions about analog computing lie outside of the frame of relevance of Church-Turing computation. Variables are continuous and (often) vary continuously in time; noise is

always present and computational precision is limited. Often, analog computation is applied to control problems, for which realtime response is important and asymptotic complexity is largely irrelevant. Therefore, the Church-Turing model of computation is not, in general, a useful model for studying analog computation and its capabilities.

Moreover, when a model is applied to questions outside of its frame of relelvance, or even near the boundaries of its frame, it is likely to produce misleading answers, because in these cases its simplifying assumptions are no longer good assumptions. Often the conclusions reached are more a reflection of the simplifying assumptions of the model than of the system being modelled. This is the reason that analyses of the power of analog computation in the context of Church-Turing computation often yield contradictory conclusions. They may, for example, depend on assumptions irrelevant to analog computing, such as whether all the standard real numbers exist or only the Turing-computable reals.

The question of whether analog computation is super-Turing computation or not is therefore not a fruitful question, because it is asked in the context of the Church-Turing model, which is not an accurate model of analog computation and does not include the relevant questions within its frame of relevance. It is more productive to view analog computation as a form of *non-Turing* computation, about which we make different assumptions and ask different sorts of questions (MacLennan 2009c). These include questions about general-purpose analog computing (Section 7), real-time response relative to the time constants of the underlying physical processes, stability, robustness (in the face of to noise, errors and defects) and so forth.

9. Analog computing hardware

A comprehensive review of present and future analog computing hardware is beyond the scope of this article; a brief discussion of electronic and non-electronic realizations must suffice. For historical information, see (Small 2001), summarized in (MacLennan 2009a).

Carver Mead's *Analog VLSI and Neural Systems* (Mead 1989) signalled a rebirth of interest in electronic analog computing. It illustrated a number of VLSI devices inspired by neural systems, which, he observed, are typically non-linear and analog. Example devices included a "silicon retina" and an "electronic cochlea" (Mead 1989, Chapters 15–16). Analog VLSI is especially suited for post-Moore's law computing because fewer devices are typically required for analog than for digital computation. For example, a four-quadrant adder can be implemented with just four transistors and a four-quadrant multiplier with 9 to 17 transistors, depending on the required operation range (Mead 1989, 87–96). Apparently more complex operations can be implemented with fewer devices: two transistors for ln and exp, three for square root and five for tanh, which is frequently used as a sigmoid function in neural computation (Mead 1989, 70–71, 97–99).

For general-purpose analog computing there are now *field-programmable analog arrays* (FPAAs), which are the analog equivalents of the field-programmable gate arrays (FPGAs) used for rapid implementation of digital systems (Basu et al. 2010). A typical FPAA comprises a number of *computational analog blocks* (CABs), each providing a number of analog computational elements, such as operational transconductance amplifiers, whose gain is controlled by a bias current. These operational amplifiers can be used to implement various functions including integration, differentiation and amplification. CABs might also include tunable bandpass filters, which can be used for Fourier transforms, and small analog matrix-vector multipliers for implementing linear operators. The precision of FPAA computation is about 10^{-3} (0.1%) of full-scale variation.

Typical FPAAs use *floating-gate transistors*, in which the gate has no direct-current connection to other circuit elements and therefore can hold a charge for an indefinite time, which is how analog values are stored. The charge can be increased by electron tunnelling and decreased by hot electron injection. Floating-gate transistors are used as switches in switching matrices that control

the connections between the analog computing elements within and between CABs, which allows analog circuits to be programmed.

Although electronics is the most popular computing technology at this time, the demands of post-Moore's law computing will require us to explore other possibilities. One advantage of analog computing is that the majority of physical processes are continuous – that is, defined by differential equations – and so they are immediately candidates for analog computing technologies. Some are especially suitable for massively parallel analog field computation. I will mention a few possibilities. First is optics, in particular non-linear optics, which can realize many basic analog computation operations (Ambs 2010). Another potential technology is chemical computing, in which analog quantities are represented by chemical concentrations (e.g. Dittrich and Fenizio 2007). For example, reaction–diffusion equations implement a kind of chemical field computation. Finally there is quantum computing. Although the well-known approach to quantum computing is thought of as digital, because it operates on discrete qubits, it is really a form of analog computation because it manipulates complex amplitudes, which are continuous. Likewise, quantum annealing and adiabatic quantum computing exploit continuous evolution of the quantum state to solve optimization problems (Das and Chakrabarti 2008; Santoro and Tosatti 2006). Furthermore, there is the relatively unexplored field of *continuous-value quantum computation*, which is a direct approach to quantum analog computing (Lloyd and Braunstein 1999).

10. Conclusions

I have argued that future computing paradigms and technologies will have to be more like the physical processes by which they are realized, but because these processes are primarily continuous, post-Moore's law computing will require an increased use of analog computation. Traditionally analog computers have computed ordinary differential equations of time, but analog field computation permits massively parallel temporal integration of partial differential equations. In principle many different physical media – not just electronics – can be exploited to implement the basic operations of analog computing, a small number of which are sufficient to approximate a wide variety of analog computations, thus providing a basis for universal analog computation and general-purpose analog computers. The contentious issue of the computational power of analog computers is addressed best on its own terms, rather by asking it within the context of Church-Turing computation, which distorts the relevant questions and their answers.

Note

1. Fréchet and Gâteaux derivatives are the same for field transformations; see (MacLennan 2009b).

Notes on contributor

B.J. MacLennan has a BS in mathematics (with honors, 1972) from Florida State University and an MS (1974) and PhD (1975) in computer science from Purdue University. He joined Intel Corporation in 1975 as a Senior Software Engineer, but in 1979 he returned to academia, joining the Computer Science faculty of the Naval Postgraduate School (Monterey, CA), where he investigated massively parallel computing and artificial intelligence. Since 1987 he has been a member of the Computer Science faculty of the University of Tennessee, Knoxville. For the last three decades MacLennan's research has focused on novel models of computation intended to better exploit physical processes for computation, and to provide new concepts of information representation and processing in natural and artificial systems. He has more than 70 refereed journal articles and book chapters and has had two sole-authored books published (one in its third edition) and has edited one book. He has made more than 70 invited or refereed presentations. MacLennan is also the founding Editor-in-Chief of the *International Journal of Nanotechnology and Molecular Computation*.

References

Adamatzky, A., B. De Lacy Costello, and T. Asai. 2005. *Reaction-Diffusion Computers*. Amsterdam: Elsevier.

Ambs, P. 2010. "Optical Computing: A 60-year Adventure." *Advances in Optical Technologies*. Article ID 372652. doi:10.1155/2010/372652.

Basu, A., S. Brink, C. Schlottmann, S. Ramakrishnan, C. Petre, S. Koziol, F. Baskaya, C. M. Twigg, and P. Hasler. 2010. "A Floating Gate Based Field Programmable Analog Array." *IEEE Journal of Solid State Circuits* 45: 1781–1794.

Benzi, R., G. Parisi, A. Sutera, and A. Vulpiani. 1982. "Stochastic Resonance in Climatic Change." *Tellus* 34: 10–16.

Blum, L., F. Cucker, M. Shub, and S. Smale. 1998. *Complexity and Real Computation*. Berlin: Springer-Verlag.

Bournez, O., M. Campagnolo, D. Graça, and E. Hainry. 2006. The General Purpose Analog Computer and Computable Analysis are Two Equivalent Paradigms of Analog Computation. In *Theory and Applications of Models of Computation (TAMC 2006)*. Vol. 3959, Lectures Notes in Computer Science. 631–643. Berlin: Springer-Verlag.

Bournez, O., and M. Cosnard. 1996. "On the Computational Power of Dynamical Systems and Hybrid Systems." *Theoretical Computer Science* 168 (2): 417–59.

Branicky, M. 1994. "Analog Computation with Continuous ODEs." In *Proceedings IEEE Workshop on Physics and Computation*, 265–74. Dallas, TX.

Das, A., and B. K. Chakrabarti. 2008. "Colloquium: Quantum Annealing and Analog Quantum Computation." *Reviews of Modern Physics* 80: 1061–1081.

Davis, M. 2006. "Why There is No Such Discipline as Hypercomputation." *Applied Mathematics and Computation* 178: 4–7.

Dittrich, P., and P. S. d. Fenizio. 2007. "Chemical Organization Theory." *Bulletin of Mathematical Biology* 69 (4): 1199–1231.

Frank, M. P. 2005. "Introduction to Reversible Computing: Motivation, Progress, and Challenges." In *Proceedings of the 2nd Conference on Computing Frontiers*, 385–390. New York: ACM Press.

Franklin, S., and M. Garzon. 1990. "Neural Computability." In *Progress in Neural Networks*. Vol. 1, edited by O. M. Omidvar, 127–145. Norwood, NJ: Ablex.

Garzon, M., and S. Franklin. 1990. "Computation on Graphs: From Neural Networks to Cellular Automata." In *Progress in Neural Networks*, Vol. 2, edited by O. M. Omidvar, 229–251. Norwood, NJ: Ablex.

Haykin, S. 1999. *Neural Networks: A Comprehenive Foundation*. 2nd ed. Upper Saddle River, NJ: Prentice Hall.

Howe, R. M. 1961. *Design Fundamentals of Analog Computer Components*. Princeton, NJ: Van Nostrand.

Kirchhoff, G. 1845. "Ueber den durchgang eines elektrischen stromes durch eine ebene, insbesondere durch eine kreisförmige [On the Passage of an Electrical Current Through a Plane, in Particular, Through a Circular Plane]." *Annalen der Physik und Chemie* 140/64 (4):497–514.

Kirkpatrick, S., C. D. Gellat, and M. P. Vecchi. 1983. "Optimization by Simulated Annealing." *Science* 220: 671–680.

Knudsen, E., S. du Lac, and S. Esterly. 1987. "Computational Maps in the Brain." *Annual Review of Neuroscience* 10: 41–65.

Lipshitz, L., and L. A. Rubel. 1987. "A Differentially Algebraic Replacment Theorem." *Proceedings of the American Mathematical Society* 99 (2): 367–72.

Lloyd, S., and S. L. Braunstein. 1999. "Quantum Computation Over Continuous Variables." *Physical Review Letters* 82: 1784–1787.

Maass, W., and E. Sontag. 1999. "Analog Neural Nets with Gaussian or Other Common Noise Distributions Cannot Recognize Arbitrary Regular Languages." *Neural Computation* 11: 771–782.

MacLennan, B. J. 1987. "Technology-independent Design of Neurocomputers: The Universal Field Computer." In *Proceedings of the IEEE First International Conference on Neural Networks*, Vol. 3, edited by M.Caudill and C. Butler, 39–49. New York: IEEE Press.

MacLennan, B. J. 1994. "Continuous Computation and the Emergence of the Discrete." In *Origins: Brain & Self-Organization*, edited by K.H. Pribram, 121–151. Hillsdale, NJ: Lawrence Erlbaum. web.eecs.utk.edu/mclennan,cogprints.soton.ac.uk/abs/comp/199906001.

MacLennan, B. J. 1997. "Field Computation in Motor Control." In *Self-Organization, Computational Maps and Motor Control*, edited by P.G. Morasso and V. Sanguineti, 37–73. Amsterdam: Elsevier. web.eecs.utk.edu/mclennan.

MacLennan, B. J. 2003. "Transcending Turing Computability." *Minds and Machines* 13: 3–22.

MacLennan, B. J. 2004. "Natural Computation and Non-turing Models of Computation." *Theoretical Computer Science* 317: 115–145.

MacLennan, B. J. 2009a. "Analog Computation." Chap. 1 In *Encyclopedia of Complexity and System Science, Entry 19*, edited by R. Meyers et al., 161–194, Springer. Reprinted in 2012. Complexity: Theory, Techniques, and Applications, edited by R. A. Meyers, 161–184. Springer.

MacLennan, B. J. 2009b. "Field Computation in Natural and Artificial Intelligence." Chap. 6 In *Encyclopedia of Complexity and System Science, Entry 199*, edited by R. Meyers, 3334–3360. Springer.

MacLennan, B. J. 2009c. "Super-Turing or non-Turing? Extending the concept of computation." *International Journal of Unconventional Computing* 5 (3–4): 369–387.

MacLennan, B. J. 2010. "The U-machine: A Model of Generalized Computation." *International Journal of Unconventional Computing* 6 (3–4): 265–283.

Mead, C. 1989. *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley.

Mills, J. W. 1995. *Kirkhoff Machines*. Technical Report TR439. Dept. of Computer Science, Indiana University.

Mills, J. W. 1996. "The Continuous Retina: Image Processing with a Single-sensor Artificial Neural Field Network." *In Proceedings IEEE Conference on Neural Networks*. New York: IEEE Press.

Mills, J. W. B., B. Himebaugh, B. Kopecky, M. Parker, C. Shue, and C. Weilemann. 2006. "'Empty space' Computes: The Evolution of an Unconventional Supercomputer." In *Proceedings of the 3rd Conference on Computing Frontiers*, 115–126, New York: ACM Press.

Moore, C. 1996. "Recursion Theory on the Reals and Continuous-time Computation." *Theoretical Computer Science* 162: 23–44.

Moore, G. E. 1965. "Cramming More Components onto Integrated Circuits." *Electronics* 38 (8): 114–117.

Omohundro, S. 1984. "Modeling Cellular Automata with Partial Differential Equations." *Physica D* 10: 128–34.

Orponen, P. 1997. "A Survey of Continous-time Computation Theory." In *Advances in Algorithms, Languages, and Complexity*, edited by D.-Z. Du, and K.-I. Ko,, 209–224. Dordrecht: Kluwer.

Orponen, P., and M. Matamala. 1996. "Universal Computation by Finite Two-dimensional Coupled Map Lattices." In *Proceedings, Physics and Computation*, 243–247. Cambridge, MA: New England Complex Systems Institute.

Pour-El, M. 1974. "Abstract Computability and its Relation to the General Purpose Analog Computer (Some Connections Between Logic, Differential Equations and Analog Computers)." *Transactions of the American Mathematical Society* 199: 1–29.

Pour-El, M. B., and I. Richards. 1979. "A Computable Ordinary Differential Equation which Possesses No Computable Solution." *Annals of Mathematical Logic* 17: 61–90.

Pour-El, M. B., and I. Richards. 1982. "Noncomputability in Models of Physical Phenomena." *International Journal of Theoretical Physics* 21: 553–555.

Rubel, L. A. 1985. "The Brain as an Analog Computer." *Journal of Theoretical Neurobiology* 4: 73–81.

Rubel, L. A. 1988. "Some Mathematical Limitations of the General-purpose Analog Computer." *Advances in Applied Mathematics* 9: 22–34.

Rubel, L. A. 1993. "The Extended Analog Computer." *Advances in Applied Mathematics* 14: 39–50.

Rumelhart, D. E., McClelland, J. L., and the PDP Research Group. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press.

Sanger, T. 1996. "Probability Density Estimation for the Interpretation of Neural Population Codes." *Journal of Neurophysiology* 76: 2790–3.

Santoro, G. E., and E. Tosatti. 2006. "Optimization Using Quantum Mechanics: Quantum Annealing Through Adiabatic Evolution." *Journal of Physics A: Mathematical and General* 39 (36): R393.

Shannon, C. E. 1941. "Mathematical Theory of the Differential Analyzer." *Journal of Mathematics and Physics of the Massachusetts Institute Technology* 20: 337–354.

Shannon, C. E. 1993. "Mathematical Theory of the Differential Analyzer." In *Claude Elwood Shannon: Collected Papers*, edited by N. J. A. Sloane and A. D. Wyner, 496–513. New York: IEEE Press.

Siegelmann, H. T. 1999. *Neural Networks and Analog Computation: Beyond the Turing Limit*. Boston: Birkhäuser.

Siegelmann, H. T., and E. D. Sontag. 1994. "Analog Computation Via Neural Networks." *Theoretical Computer Science* 131: 331–360.

Small, J. S. 2001. *The Analogue Alternative*. London: Routledge.

Stannett, M. 1990. "X-machines and the Halting Problem: Building a Super-turing Machine." *Formal Aspects of Computing* 2: 331–341.

Truitt, T. D., and A. E. Rogers. 1960. *Basics of Analog Computers*. New York: John F. Rider.

Turing, A. 1952. "The Chemical Basis of Morphogenesis." *Philosophical Transactions of the Royal Society B* 237: 37–72.

van Gelder, T. 1997. "Dynamics and Cognition." Chap. 16 In *Mind Design II: Philosophy, Psychology and Artificial Intelligence* (revised & enlarged ed.), edited by J. Haugeland, 421–450. Cambridge MA: MIT Press.

Wolpert, D. H., and B. J. MacLennan. 1993. *A Computationally Universal Field Computer That is Purely Linear.* Technical Report CS-93-206:Knoxville: Dept. of Computer Science, University of Tennessee. web.eecs.utk.edu/mclennan.