The U-Machine: A Model of Generalized Computation

Technical Report UT-CS-06-587

Bruce J. MacLennan[*]

Department of Computer Science
University of Tennessee, Knoxville
`www.cs.utk.edu/~mclennan`

December 14, 2006

Abstract

We argue that post-Moore's Law computing technology will require the exploitation of new physical processes for computational purposes, which will be facilitated by new models of computation. After a brief discussion of computation in the broad sense, we present a model of generalized computation, and a corresponding machine model, which can be applied to massively-parallel nanocomputation in bulk materials. The machine is able to implement quite general transformations on a broad class of topological spaces by means of Hilbert-space representations. Neural morphogenesis provides a model for the physical structure of the machine and means by which it may be configured, a process that involves the definition of signal pathways between two-dimensional data areas and the setting of interconnection strengths within them. This approach also provides a very flexible means of reconfiguring of the internal structure of the machine.

Keywords: analog computer, field computation, nanocomputation, natural computation, neural network, radial-basis function network, reconfigurable, Moore's law, U-machine.

AMS (MOS) subject classification: 68Q05.

[*]This report may be used for any non-profit purpose provided that the source is credited.

1 Introduction

This report describes a computational model that embraces computation in the broadest sense, including analog and digital, and provides a framework for computation using novel materials and physical processes. We explain the need for a generalized model of computation, define its mathematical basis, and give examples of possible implementations suitable to a variety of media.

As is well known, the end of Moore's Law is in sight, and we are approaching the limits of digital electronics and the von Neumann architecture, so we need to develop new computing technologies and methods of using them. This process will be impeded by assuming that binary digital logic is the only basis for practical computation. Although this particular approach to computation has been very fruitful, the development of new computing technologies will be aided by stepping back and asking what computation is in more general terms. If we do this, then we will see that there are many physical processes available that might be used in future computing technologies. In particular, physical processes that are not well suited to implementing binary digital logic may be better suited to alternative models of computation. Thus one goal for the future of computing technology is to find useful correspondences between physical processes and models of computation. But what might an alternative to binary digital computation look like?

Neural networks provide one alternative model of computation. They have a wide range of applicability, and are effectively implementable in various technologies, including conventional general-purpose digital computers, digital matrix-vector multipliers, analog VLSI, and optics. Artificial neural networks, as well as neural information processing in the brain, show the effectiveness of massively-parallel, low-precision analog computation for many applications; therefore neural networks exemplify the sorts of alternative models we have in mind.

More generally, nature provides many examples of effective, robust computation, and *natural computation* has been defined as computation occurring in nature or inspired by it. In addition to neural networks, natural computation includes genetic algorithms, artificial immune systems, ant colony optimization, swarm intelligence, and many similar models. Computation in nature does not use binary digital logic, so natural computation suggests many alternative computing technologies, which might exploit new materials and processes for computation.

Nanotechnology, including nanobiotechnology, promises to provide new materials and processes that could, in principle, be used for computation, but they also provide new challenges, including three-dimensional nanoscale fabrication, high defect and failure rates, and inherent, asynchronous parallelism. In many cases we can, of course, engineer these materials and processes to implement the familiar binary digital logic, but that strategy may require excessive overhead for redundancy, error correction, synchronization, etc., and thus waste much of the potential advantage of these new technologies.

We believe that a better approach is to match computational processes to the physical processes most suited to implement them. On the one hand, we should have an array of physical processes available to implement any particular alternative useful model of computation. On the other, we should have libraries of computational models that can implemented by means of particular physical processes. Some guidelines for the application of a wider range of physical processes to computation can be found elsewhere [10], but here we will consider a model of generalized computation that can be implemented in a variety of potential technologies.

2 Generalized Computation

We need a notion of computation that is sufficiently broad to allow us to discover and to exploit the computational potentials of a wide range of processes and materials, including those emerging from nanotechnology and those inspired by natural systems. Therefore we have argued for a generalized definition of computational systems that encompasses analog as well as digital computation and that can be applied to computational systems in nature as well as to computational devices (i.e., *computers* in the strict sense) [5, 6, 9]. According to this approach, the essential difference between computational processes and other physical processes is that the purpose of computational processes is the mathematical "manipulation" of mathematical objects. Here "mathematical" is used in a broad sense to refer to any abstract or formal objects and processes. This is familiar from digital computers, which are not limited to manipulating numbers and to numerical calculation, but can also manipulate other abstract objects such as character strings, lists, and trees. Similarly, in addition to numbers and differential equations, analog computers can process images, signals, and other continuous information representations. Here, then, is our definition of generalized computation [5, 6, 9]:

Definition 1 (Computation) *Computation is a physical process, the purpose of which is abstract operation on abstract objects.*

What distinguishes computational processes from other physical systems is that while the abstract operations must be instantiated in *some* physical medium, they can be implemented by *any* physical process that has the same abstract structure. That is, a computational process can be *realized* by any physical system that can be described by the same mathematics (abstract description). More formally:

Definition 2 (Realization) *A physical system* realizes *a computation if, at the level of abstraction appropriate to its purpose, the system of abstract operations on the abstract objects is a sufficiently accurate model of the physical process. Such a physical system is called a* realization *of the computation.*

Any perfect realization of a computation must have at least the abstract structure of that computation (typically it will have additional structure); therefore, we may say there is a homomorphism from the structure of a physical realization to the abstract computation; in practice, however, many realizations are only approximate (e.g., digital or analog representations of real numbers) [5, 9].

With this background, we can state that the goal of this research is to develop a model of generalized computation that can be implemented in a wide variety of physical media. Therefore we have to consider the representation of a very broad class of abstract objects and the computational processes on them.

3 Representation of Data

3.1 Urysohn Embedding

First we consider the question of how we can represent quite general mathematical objects on a physical system. The universe of conceivable mathematical objects is, of course, impractically huge and lacking in structure (beyond the bare requirements of abstractness and consistency). Therefore we must identify a class of mathematical objects that is sufficiently broad to cover analog and digital computation, but has manageable structure. In this paper we will limit the *abstract spaces* over which we compute to *second-countable metric spaces*, that is, metric spaces with countable bases.[1] Such metric spaces are *separable*, that is, they have countable dense subsets, which in turn means there is a countable subset that can be used to approximate the remaining elements arbitrarily closely. Further, since a topological *continuum* is often defined as a (non-trivial) connected compact metric space [11, p. 158], and since a compact metric space is both separable and complete, the second-countable spaces include the topological continua suitable for analog computation. On the other hand, countable *discrete* topological spaces are second-countable, and so digital data representation is also included in the class of spaces we address.

In this paper we describe the *U-machine*, which is named after Pavel Urysohn (1898–1924), who showed that any second-countable metric space is homeomorphic to a subset of the Hilbert space E^∞. Therefore, computations on second-countable metric spaces can be represented by computations on Hilbert spaces, which brings them into the realm of physical realizability. To state the theorem more precisely, we will need the following definition [12, pp. 324–5]:

Definition 3 (Fundamental Parallelopiped) *The* fundamental parallelopiped $Q^\infty \subset E^\infty$ *is defined:*

$$Q^\infty = \{(u_1, u_2, \dots u_k, \dots) \mid u_k \in \mathbb{R} \wedge |u_k| \leq 1/k\}.$$

[1]Indeed, we can broaden abstract spaces to *second-countable regular Hausdorff spaces*, since according to the *Urysohn metrization theorem*, these spaces are metrizable [17, pp. 137–9].

Urysohn showed that any second-countable metric space (X, δ_X) is homeomorphic to a subset of Q^∞ and thus to a subset of the Hilbert space E^∞. Without loss of generality we assume $\delta_X(x, x') \leq 1$ for all $x, x' \in X$, for if this is not the case we can define a new metric

$$\delta'_X(x, x') = \frac{\delta_X(x, x')}{1 + \delta_X(x, x')}.$$

The embedding is defined as follows:

Definition 4 (Urysohn Embedding) *Suppose that (X, δ_X) is a second-countable metric space; we will define a mapping $U : X \to Q^\infty$. Since X is separable, it has a countable dense subset $\{b_1, b_2, \ldots, b_k, \ldots\} \subseteq X$. For $x \in X$ define*

$$U(x) = \mathbf{u} = (u_1, u_2, \ldots, u_k, \ldots), \text{ where } u_k = \delta_X(x, b_k)/k.$$

Note that $|u_k| \leq 1/k$ so $\mathbf{u} \in Q^\infty$.

Informally, each point x in the metric space is mapped into an infinite vector of its (bounded) distances from the dense elements; the vector elements are generalized Fourier coefficients in E^∞. It is easy to show that U is a homeomorphism and that indeed it is uniformly continuous (although U^{-1} is merely continuous) [12, p. 326]. There are of course other embeddings of second-countable sets into Hilbert spaces, but this example will serve to illustrate the U-machine.

Physical U-machines will not always represent the elements of an abstract space with perfect accuracy, which can be understood in terms of ϵ-nets [12, p. 315].

Definition 5 (ϵ-net) *An ϵ-net is a finite set of points $\mathcal{E} = \{b_1, \ldots, b_n\} \subseteq X$ of a metric space (X, δ_X) such that for each $x \in X$ there is some $b_k \in \mathcal{E}$ with $\delta_X(x, b_k) < \epsilon$.*

Any compact metric space has an ϵ-net for each $\epsilon > 0$ [12, p. 315]. In particular, in such a space there is a sequence of ϵ-nets $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_k, \ldots$ such that $\mathcal{E}_k$ is a $\frac{1}{k}$-net; they provide progressively finer nets for determining the position of points in X. The union of them all is a countable dense set in X [12, p. 315]. Letting $\mathcal{E}_k = \{b_{k,1}, \ldots, b_{k,n_k}\}$, we can form a sequence of all the $b_{k,l}$ in order from coarser to the finer nets ($k = 1, 2, \ldots$), which we can reindex as b_j. Most of the second-countable spaces over which we would want to compute are compact, and so they have such a sequence of ϵ-nets. Therefore we can order the coefficients of the Hilbert-space representation to go from coarser to finer, $U(x) = \mathbf{u} = (\ldots, u_j, \ldots)$ where $u_j = \delta_X(x, b_j)/j$. I will call this a *ranked* arrangement of the coefficients.

Physical realizations of computations are typically approximate; for example, on ordinary digital computers real numbers are represented with finite precision, integers are limited to a finite range, and memory (for stacks etc.) is strictly bounded. So also, physical realizations of a U-machine will not be able to represent all the elements of an infinite-dimensional Hilbert space with perfect accuracy. One way to get a finite precision approximation for an abstract space element is to use a ranked organization for its coefficients, and to truncate it at some specific n; thus, the representation is a finite-dimensional vector $\mathbf{u} = (u_1, u_2, \ldots, u_n)^{\mathrm{T}}$. This amounts to limiting the fineness of the ϵ-nets. (Obviously, if we are using finite-dimensional vectors, we can use the simpler definition $u_k = \delta_X(x, b_k)$, which omits the k^{-1} scaling.)

The representation of a mathematical object in terms of a finite discrete set $\mathbf{u}$ of continuous coefficients is reminiscent of a neural-net representation, in which the coefficients u_k ($1 \leq k \leq n$) are the activities of the neurons in one layer. Furthermore, consider the "complementary" representation $\mathbf{s} = \mathbf{1} - \mathbf{u}$, that is,

$$s_k = 1 - u_k = 1 - \delta_X(x, b_k).$$

We can see that the components of $\mathbf{s}$ measure the closeness or similarity of x to the ϵ-net elements b_k. This is a typical neural-net representation, such as we find, for example, in radial-basis function (RBF) networks. Therefore, some U-machines will compute in an abstract space by means of operating, neural-network style, on finite-dimensional vectors of generalized Fourier coefficients. (Examples are discussed below.)

Other physical realizations of the U-machine will operate on *fields*, that is, on continuous distributions of continuous data, which are, in mathematical terms, the elements of an appropriate Hilbert function space. This kind of computation is called *field computation* [2, 3, 4, 8]. More specifically, let $\mathcal{L}_2(\Omega)$ be a Hilbert space of functions $\phi : \Omega \to \mathbb{R}$ defined over a domain Ω, a connected compact metric space. Let $\xi_1, \xi_2, \ldots, \xi_k, \ldots \in \mathcal{L}_2(\Omega)$ be an orthonormal basis for the Hilbert space. Then an element $x \in X$ of the abstract space is represented by the field $\phi = \sum_{k=1}^{\infty} u_k \xi_k$, where as usual the generalized Fourier coefficients are $u_k = \delta_X(x, b_k)/k$.

Physical field computers operate on band-limited fields, just as physical neural networks have a finite number of neurons. If, as in the ordinary Fourier series, we put the basis functions in order of non-decreasing frequency, then a band-limited field will be a finite sum,

$$\hat{\phi} = \sum_{k=1}^{n} u_k \xi_k.$$

The Euclidean space E^n is isomorphic (and indeed isometric) to the subspace of $\mathcal{L}_2(\Omega)$ spanned by $\xi_1, \ldots, \xi_n$.

It will be worthwhile to say a few words about how digital data fit into the representational frame we have outlined, for abstract spaces may be discrete as well as continuous. First, it is worth recalling that the laws of nature are continuous, and our digital devices are implemented by embedding a discrete space into a continuum (or, more accurately, by distinguishing disjoint subspaces of a continuum). For example, we may embed the discrete space $\{0, 1\}$ in the continuum $[0, 1]$ with the ordinary topology. Similarly, n-bit strings in $\{0, 1\}^n$ can be embedded in $[0, 1]^n$ with any appropriate topology (i.e., a second-countable metric space). However, as previously remarked, countable discrete topological spaces (i.e., those in which $\delta_X(x, x') = 1$ for

$x \neq x'$) *are* second countable, and therefore abstract spaces in our sense; the space is its own countable dense subset. Therefore the Urysohn Hilbert-space representation of any $b_k \in X$ is a (finite- or infinite-dimensional) vector $\mathbf{u}$ in which the $u_k = 0$ and $u_j = 1$ for $j \neq k$. Also, the complementary similarity vector $\mathbf{s} = \mathbf{1} - \mathbf{u}$ is a unit vector along the k-th axis, which is the familiar 1-out-of-n or unary representation often used in neural networks. On a field computer, the corresponding representation is the orthonormal basis function ξ_k (i.e., discrete values are represented by "pure states").

3.2 An Abstract Cortex

Hitherto, we have considered the Hilbert-space representation of a single abstract space, but non-trivial computations will involve processes in multiple abstract spaces. Therefore, just as the memory of an ordinary general-purpose digital computer may be divided up into a number of variables of differing data types, we need to be able to divide the state space of a U-machine into disjoint variables, each representing its own abstract space. We have seen that an abstract space can be homeomorphically embedded in Hilbert space, either a discrete set $[0,1]^\infty$ of generalized Fourier coefficients or a space $\mathcal{L}_2(\Omega)$ of continuous fields over some domain Ω. Both can easily accommodate the representation of multiple abstract spaces.

This is easiest to see in the case of a discrete set of Fourier coefficients. For suppose we have a set of abstract spaces $X_1, \ldots, X_N$, represented by finite numbers $n_1, \ldots, n_N$ of coefficients, respectively. These can be represented in a memory that provides at least $n = \sum_{k=1}^{N} n_k$ coefficients in $[0,1]$. In effect we have at least n neurons divided into groups (e.g., layers) of size $n_1, \ldots, n_N$. This is exactly analogous to the partitioning of an ordinary digital computer's memory (which, in fact, could be used to implement the coefficients by means of an ordinary digital approximation of reals in $[0,1]$).

In a similar way multiple field-valued variables can be embedded into a single field-valued memory space. In particular, suppose a field computer's memory represents real-valued time-varying fields over the unit square; that is, a field is a function $\phi(t) \in \mathcal{L}_2([0,1]^2)$. Further suppose we have N field variables over the same domain, $\psi_1(t), \ldots, \psi_N(t) \in \mathcal{L}_2([0,1]^2)$. They can be stored together in the same memory field $\phi(t)$ by shrinking the sizes of their domains and translating them so that they will occupy separated regions of the memory's domain. Thus $\psi_j(t)$ is replaced by $\hat{\psi}_j(t)$, which is defined over $[u, u+w] \times [v, v+w]$ $(0 < u, v, w < 1)$ by the definition

$$\hat{\psi}_j(t; x, y) = \psi_j\left(t; \frac{x}{w} - u, \frac{y}{w} - v\right).$$

That is, $\hat{\psi}_j$ is ψ_j shrunk to a w^2 domain with its $(0,0)$ corner relocated to (u,v) in the memory domain. With an appropriate choice of locations and domain scale factors so that the variables are separated, $\phi(t) = \sum_{j=1}^{N} \hat{\psi}_j(t)$. Of course, shrinking the field variables pushes information into higher (spatial) frequency bands, and so the capacity of the field memory (the number of field variables it can hold) will be determined by its bandwidth and the bandwidths needed for the variables.

As will be discussed in Sec. 4.6, there are good reasons for using fields defined over a two-dimensional manifold, as there are for organizing discrete neuron-like units in two-dimensional arrays. This is, of course, to a first approximation, the way neurons are organized in the neural cortex. Furthermore, much of the cortex is organized in distinct cortical *maps* in which field computation takes place [7, 8]. (Indeed, cortical maps are one of the inspirations of field computing.) Therefore the memory of a U-machine can be conceptualized as a kind of abstract cortex, with individual regions representing distinct abstract spaces. This analogy suggests ways of organizing and programming U-machines (discussed below) and also provides a new perspective on neural computation in brains.

4 Representation of Process

4.1 Introduction

The goal of our embedding of second-countable metric spaces in Hilbert spaces is to facilitate computation in arbitrary abstract spaces. As a beginning, therefore, consider a map $\Phi : X \rightarrow Y$ between two abstract spaces, (X, δ_X) and (Y, δ_Y). The abstract spaces can be homeomorphically embedded into subsets of the Hilbert space Q^∞ by $U : X \rightarrow Q^\infty$ and $V : Y \rightarrow Q^\infty$. These embeddings have continuous inverses defined on their ranges; that is, $U^{-1} : U[X] \rightarrow X$ and $V^{-1} : V[Y] \rightarrow Y$ are continuous. Therefore we can define a function $\mathbf{F} : U[X] \rightarrow V[Y]$ as the topological conjugate of Φ; that is, $\mathbf{F} = V \circ \Phi \circ U^{-1}$. Thus, for any map Φ between abstract spaces, we have a corresponding computation $\mathbf{F}$ on their Hilbert-space representations. (In the preceding discussion we have used representations based on discrete sets of coefficients in Q^∞, but a field representation in a Hilbert function space would work the same. Suppose $U : X \rightarrow \mathcal{L}_2(\Omega_1)$ and $V : Y \rightarrow \mathcal{L}_2(\Omega_2)$. Then $\mathcal{F} = V \circ \Phi \circ U^{-1}$ is the field computation of Φ.)

In all but the simplest cases, U-machine computations will be composed from simpler U-machine computations, analogously to the way that ordinary programs are composed from simpler ones. At the lowest level of digital computation, operations are built into the hardware or implemented by table lookup. Also, general-purposes analog computers are equipped with primitive operations such as addition, scaling, differentiation, and integration [14, 15, 16]. Similarly, U-machines will have to provide some means for implementing the most basic computations, which are not composed from more elementary ones. Therefore our next topic will be the implementation of primitive U-machine computations, and then we will consider ways for composing them into more complex computations.

4.2 Universal Approximation

In order to implement primitive U-machine computations in terms or our Hilbert-space representations we can make use of several convenient "universal approximation theorems," which are generalizations of discrete Fourier series approximation [1, pp. 208–9, 249–50, 264–5, 274–8, 290–4]. These can be implemented by neural network-style computation, which works well with the discrete Hilbert space representation.

We will consider the case in which we want to compute a known function $\Phi : X \to Y$, for which we can generate enough input-output pairs (x_k, y_k), $k = 1, \ldots, P$, with $y_k = \Phi(x_k)$, to get a sufficintly accurate approximation. Our goal then is to approximate $\mathbf{F} : [0,1]^M \to [0,1]^N$, since computation takes place on the finite-dimensional vector-space representations. Our general approach is to approximate $\mathbf{v} = \mathbf{F}(\mathbf{u})$ by a linear combination of H vectors $(\mathbf{a}^j)$ weighted by simple scalar nonlinear functions $r_j : [0,1]^M \to \mathbb{R}$, that is,

$$\mathbf{v} \approx \sum_{j=1}^{H} \mathbf{a}^j r_j(\mathbf{u}).$$

Corresponding to (x_k, y_k) define the Hilbert representations $\mathbf{u}^k = U(x_k)$ and $\mathbf{v}^k = V(y_k)$. For exact interpolation we require

$$\mathbf{v}^k = \sum_{j=1}^{H} \mathbf{a}^j r_j(\mathbf{u}^k), \tag{1}$$

for $k = 1, \ldots, P$. Let $V_{ki} = v_i^k$, $R_{kj} = r_j(\mathbf{u}^k)$, and $A_{ji} = a_i^j$. Then Eq. 1 can be written

$$V_{ki} = \sum_{j=1}^{H} R_{kj} A_{ji}.$$

That is, $\mathbf{V} = \mathbf{RA}$. The best solution of this equation, in a least-squares sense, is $\mathbf{A} \approx \mathbf{R}^+\mathbf{V}$, where the pseudo-inverse is defined $\mathbf{R}^+ = (\mathbf{R}^{\mathrm{T}}\mathbf{R})^{-1}\mathbf{R}^{\mathrm{T}}$. (Of course a regularization term can be added if desired [1, ch. 5].)

The same approach can be used with field computation. In this case we want to approximate $\psi = \mathcal{F}(\phi)$ for $\phi \in \mathcal{L}_2(\Omega_1)$ and $\psi \in \mathcal{L}_2(\Omega_2)$. Our interpolation will be based on pairs of input-output fields, (ϕ_k, ψ_k), $k = 1, \ldots, P$. Our approximation will be a linear weighting of H fields $\alpha_j \in \mathcal{L}_2(\Omega_2)$, specifically, $\psi \approx \sum_{j=1}^{H} \alpha_j r_j(\phi)$. Therefore the interpolation conditions are

$$\psi_k = \sum_{j=1}^{H} \alpha_j r_j(\phi_k), \qquad k = 1, \ldots, P.$$

Let $R_{kj} = r_j(\phi^k)$ and notice that this is an ordinary matrix of real numbers. For greater clarity, write the fields as functions of an arbitrary element of their domain, and the interpolation conditions are then $\psi_k(s) = \sum_{j=1}^{H} R_{kj}\alpha_j(s)$ for $s \in \Omega_2$. Let $\boldsymbol{\psi}(s) = [\ldots, \psi_k(s), \ldots]^{\mathrm{T}}$ and $\boldsymbol{\alpha}(s) = [\ldots, \alpha_j(s), \ldots]^{\mathrm{T}}$, and the interpolation condition is $\boldsymbol{\psi}(s) = \mathbf{R}\boldsymbol{\alpha}(s)$. The best solution is $\boldsymbol{\alpha}(s) \approx \mathbf{R}^{+}\boldsymbol{\psi}(s)$, which gives us an equation for computing the "weight fields":

$$\alpha_j \approx \sum_{k=1}^{P} (\mathbf{R}^{+})_{jk}\psi_k.$$

There are many choices for basis functions r_j that are sufficient for universal approximation. For example, for perceptron-style computation we can use $r_j(\mathbf{u}) = r(\mathbf{w}_j \cdot \mathbf{u} + b_j)$, where r is any nonconstant, bounded, monotone-increasing continuous function [1, p. 208]. The inner product is very simple to implement in many physical realizations of the U-machine, as will be discussed later. Alternately, we may use radial-basis functions, $r_j(\mathbf{u}) = r(\|\mathbf{u} - \mathbf{c}_j\|)$ with centers $\mathbf{c}_j$ either fixed or dependent on $\mathbf{F}$. If $H = P$ and $\mathbf{c}_k = \mathbf{u}^k$ $(k = 1, \ldots, P)$, then $\mathbf{R}$ is invertible for a variety of simple nonlinear functions r [1, p. 264–5]. Further, with an appropriate choice of stabilizer and regularization parameter, the Green's function G for the stabilizer can be used, $r_k(\mathbf{u}) = G(\mathbf{u}, \mathbf{u}^k)$, and $\mathbf{R}$ will be invertible. In these cases there are suitable nonlinear functions r with simple physical realizations.

In many cases the computation is a linear combination of a nonlinearity applied to a linear combination, that is, $\mathbf{v} = \mathbf{A}[\mathbf{r}(\mathbf{B}\mathbf{u})]$, where $[\mathbf{r}(\mathbf{B}\mathbf{u})]_j = r([\mathbf{B}\mathbf{u}]_j)$. For example, as is well known, the familiar affine transformation used in neural networks, $\mathbf{w}_j \cdot \mathbf{u} + b_j$, can be put in linear form by adding a component to $\mathbf{u}$ that is "clamped" to 1.

For an unknown function Φ, that is, for one defined only in terms of a set of desired input-output pairs (x_k, y_k), we can use a neural network learning algorithm, or interpolation and non-linear regression procedures, on the corresponding vectors $(\mathbf{u}^k, \mathbf{v}^k)$, to determine an acceptable approximation to $\mathbf{F}$.

For known functions $\Phi : X \to Y$ and their topological conjugates $\mathbf{F} : U[X] \to V[Y]$ we can generate input-output pairs $(\mathbf{u}^k, \mathbf{v}^k)$ from which to compute the parameters for a sufficiently accurate approximation of $\mathbf{F}$. This could involve computing the inverse or pseudo-inverse of a large matrix, or using a neural network learning algorithm such as back-propagation. Alternately, if Φ is sufficiently simple, then the optimal approximation parameters may be determined analytically.

Optimal coefficients (such as the $\mathbf{A}$ matrix) for commonly used and standard functions can be precomputed and stored in libraries for rapid loading into a U-machine. (Possible means for loading the coefficients are discussed below, Sec. 4.6.)

Depending on the application, standard operations might include those found in digital computers or those found in general-purpose analog computers, such as scalar addition and multiplication, and differentiation and integration [14, 15, 16]. For image processing, the primitive operations could include Fourier transforms, wavelet transforms, and convolution.

4.3 Decomposing Processes

The simplest way of dividing computations into simpler computations is by functional composition. For example, for second-countable metric spaces (X, δ_X), (Y, δ_Y), and (Z, δ_Z) suppose we have the embeddings

$$U : X \to Q^\infty,\ V : Y \to Q^\infty,\ \text{and}\ W : Z \to Q^\infty.$$

Further suppose we have functions $\Phi : X \to Y$ and $\Psi : Y \to Z$, which may be composed to yield $\Psi \circ \Phi : X \to Z$. To compute $\Psi \circ \Phi$ we use its Hilbert-space conjugate, which is

$$W \circ (\Psi \circ \Phi) \circ U^{-1} = (W \circ \Psi \circ V^{-1}) \circ (V \circ \Phi \circ U^{-1}) = \mathbf{G} \circ \mathbf{F},$$

where $\mathbf{G}$ and $\mathbf{F}$ are the conjugates of Ψ and Φ, respectively. This means that once the input has been translated to the Hilbert-space representation, computation through successive stages can take place in Hilbert space, until the final output is produced.

If, as is commonly the case (see Sec. 4.2), computation in the Hilbert space is a linear combination of nonlinearities applied to a linear combination, then the linear combination of one computation can often be composed with that of the next. Specifically, suppose we have $\mathbf{v} = \mathbf{A}[\mathbf{r}(\mathbf{Bu})]$ and $\mathbf{w} = \mathbf{A}'[\mathbf{r}'(\mathbf{B}'\mathbf{v})]$, then obviously, $\mathbf{w} = \mathbf{A}'(\mathbf{r}'\{\mathbf{C}[\mathbf{r}(\mathbf{Bu})]\})$, where $\mathbf{C} = \mathbf{B}'\mathbf{A}$. The implication is that a U-machine needs to performs two fundamental operations, matrix-vector multiplication and component-wise application of a suitable nonlinear function, which is ordinary neural network-style computation.

Finally we consider direct products of abstract spaces. Often we will need to compute functions on more than one argument, which we take, as usual, to be functions on the direct product of the spaces from which their arguments are drawn, $\Phi : X' \times X'' \to Y$. However, since $(X', \delta_{X'})$ and $(X'', \delta_{X''})$ are second-countable metric spaces, we must consider the topology on $X = X' \times X''$. The simplest choices are the p-product topologies ($p = 1, 2, \ldots, \infty$), defined by the p-product metric:

$$[\delta_X(x_1, x_2)]^p = [\delta_{X'}(x_1', x_2')]^p + [\delta_{X''}(x_1'', x_2'')]^p,$$

where $x_1 = (x_1', x_1'')$ and $x_2 = (x_2', x_2'')$. It is then easy to compute the generalized Fourier coefficients u_k of (x', x'') from the coefficients u_k' of x' and u_k'' of x'':

$$u_k = \frac{[(u_k')^p + (u_k'')^p]^{1/p}}{2}.$$

For $p = 1$ we have $u_k = \frac{u_k' + u_k''}{2}$, and for $p = \infty$ we can use $u_k = \max(u_k', u_k'')$.

4.4 Transduction

We have seen that once the input to a U-machine computation has been mapped into its Hilbert-space representation, all computation can proceed on Hilbert representations, until outputs are produced. Therefore, the issue of translation between the abstract spaces and the internal representation is limited to input-output transduction. In these cases we are translating between some physical space (an input or output space), with a relevant metric, and its internal computational representation. Many physical spaces are described as finite-dimensional vector spaces of physical quantities (e.g., forces in a robot's joints). Others (e.g., images) are conveniently described as functions in an appropriate Hilbert space. In all these cases transduction between the physical space and the computational representation is straight-forward, as will be outlined in this section.

For input transduction suppose we have a physical input space (X, δ_X) with an ϵ-net $(b_1, \ldots, b_n)$. Then the components of the Hilbert-space representation $\mathbf{u} \in [0,1]^n$ are given by $u_k = \delta_X(x, b_k)$, $k = 1, \ldots, n$. (We omit the $1/k$ scaling since the representational space is finite dimensional.) In effect the input transduction is accomplished by an array of feature detectors $r_1, \ldots, r_n : X \to [0,1]$ defined by $r_k(x) = \delta_X(x, b_k)$; typically these will be physical input devices. The input basis functions r_k are analogous to the receptive fields of sensory neurons, the activities of which are represented by the corresponding coefficients u_k. For field computation, the representing field ϕ is generated by using the u_k to weight the orthonormal basis fields of the function space: $\phi = \sum_{k=1}^{n} u_k \xi_k$ (recall Sec. 3).

Physical output spaces are virtually always vector spaces (finite dimensional or infinite dimensional, i.e., image spaces), and so outputs usually can be generated by the same kinds of approximation algorithms used for computation. In particular, if (Y, δ_Y) is an output space and $V : Y \to [0,1]^n$ is its embedding, then our goal is to find a suitable approximation to $V^{-1} : V[Y] \to Y$. This can be done by any of the approaches discussed in Sec. 4.2. For example, suppose we want to approximate the output with a computation of the form

$$V^{-1}(\mathbf{v}) = y \approx \sum_{j=1}^{H} \mathbf{a}^j r_j(\mathbf{v})$$

for suitable basis functions r_j. (Note that the $\mathbf{a}^j$ are physical vectors generated by the output transducers and that the summation is a physical process.) Let $y^1, \ldots, y^P$ be the elements of some ϵ-net for (Y, δ_Y) and let $\mathbf{v}^k = V(y^k)$. Let $Y_{ki} = y_i^k$ (since we are assuming the output space is a finite-dimensional vector space), $R_{kj} = r_j(\mathbf{v}^k)$, and $A_{ji} = a_i^j$. Then the interpolation conditions are $\mathbf{Y} = \mathbf{RA}$, and the best least-squares approximation to $\mathbf{A}$ is $\mathbf{R}^+\mathbf{Y}$.

4.5 Feedback and Iterative Computation

Our discussion so far has been limited to feed-forward or non-iterative computation, in which the output appears on the output transducers with some delay after the inputs are applied to the input transducers. However in many applications a U-machine will interact with its environment in realtime in a more significant way. In general terms, this presents few problems, since to the time-varying values $x(t)$ in the abstract spaces there will correspond time-varying elements $u(t) = U[x(t)]$ in the Hilbert-space representation. In practical terms these will be approximated by time-varying finite-dimensional vectors $\mathbf{u}(t)$ or time-varying band-limited fields $\phi(t)$. (For simplicity we will limit discussion here to finite-dimensional vectors.) Therefore a dynamical process over abstract spaces $X_1, \ldots, X_N$ can be represented by a set of differential equations over their vector representations $\mathbf{u}_1(t), \ldots, \mathbf{u}_N(t)$. Integration of these equations is accomplished by accumulation of the differential values into the areas representing the variables (enabled by activating the feedback connections in the variable areas; see Sec. 4.6).

4.6 Configuration Methods

The model U-machine is divided into a variable (data) space and a function (program) space. Typically the data space will comprise a large number M of scalar variables $u_k \in [0, 1]$. This set of scalar variables may be configured into disjoint subsets, each holding the coefficients representing one abstract space. More precisely, suppose that a U-machine computation involves abstract spaces $X_1, \ldots, X_N$ with their corresponding metrics. Further suppose that we have decided to represent X_i with n_i coefficients in $[0, 1]^{n_i}$. Obviously $M > \sum_{i=1}^{N} n_i$ or the capacity of the U-machine will be exceeded. Therefore we configure N disjoint subsets (individually contiguous) of sizes $n_1, \ldots, n_N$ to represent the N spaces. (For field computation, the data space is a field defined over a domain Ω, in which can be defined subfields over disjoint connected subdomains $\Omega_1, \ldots, \Omega_N \subset \Omega$.)

The program space implements the primitive functions on the Hilbert representations of the variables. As we have seen, an attractive way of computing these functions is by linear combinations of simple scalar basis functions of the corresponding input vector spaces (e.g., radial-basis functions). To implement a function $\mathbf{F} : [0, 1]^m \to [0, 1]^n$ by means of H basis functions requires on the order of mH connection coefficients for the basis computation and Hn for the linear combination. (More may be required if the same basis functions are not used for all components of the output vector.) In practical computations the number of connections in a U-machine configuration will be much less than the M^2 possible, and so a U-machine does not have to provide for full interconnection.

To permit flexible interconnection between the variables of a computation without interference, it is reasonable to design the U-machine so that interconnection takes place in three-dimensional space. On the other hand, we want the scalar coefficients to be stored as compactly as possible subject to accessibility for interconnection, so it makes sense to arrange them two-dimensionally. (For the same reasons, a field computer would use fields over a two-dimensional manifold.)

The foregoing considerations suggest the general physical structure of a U-machine (see Fig. 1). The data space will be a two-dimensional arrangement of scalar vari-

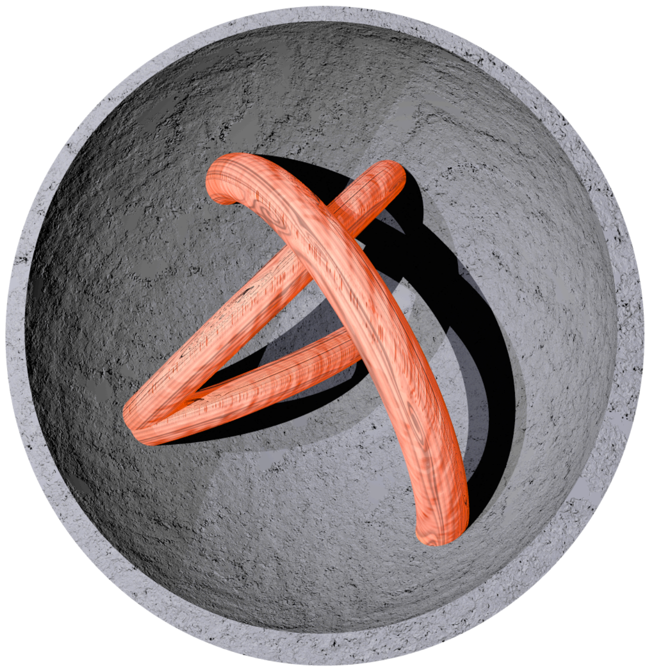

Figure 1: Conceptual depiction of interior of U-machine showing hemispherical data space and several interconnection paths in program space. Unused program space is omitted so that the paths are visible.

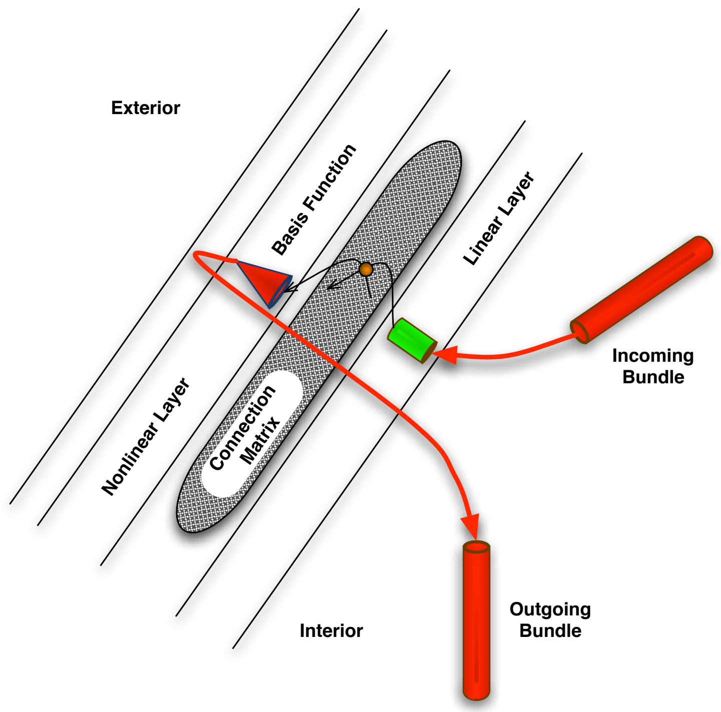

Figure 2: Layers in the data space. Incoming bundles of fibers (analogous to axons) from other parts of the data space connect to the Linear Layer, which can either register or integrate the incoming signals. Many branches lead from the Linear Layer into the Connection Matrix, which computes a linear combination of the input signals, weighted by connection strengths (one connection is indicated by the small circle). The linear combination is input to the Basis Functions, fibers from which form bundles conveying signals to other parts of the data space. The Connection Matrix also implements feedback connections to the Linear Layer for integration.

ables (or a field over a two-dimensional manifold), assumed here to be a spherical or hemispherical shell in order to shorten connections. In one layer will be the hardware to compute the basis functions; in many cases these will not be fabricated devices in the usual sense, but a consequence of bulk material properties in this layer. In a parallel layer between the basis functions and the variable layer will be the material for interconnecting them and forming linear combinations of the outputs of other basis functions (see Fig. 2). These areas of interconnection, which we call *synaptic fields*, effectively define the variable areas. In the interior of the sphere is an interconnection medium or matrix which can be configured to make connections to other variable areas. Input and output to the U-machine is by means of access to certain variable areas from outside of the sphere (see Fig. 3).

Configuring a U-machine involves defining the variable areas and setting the inter-

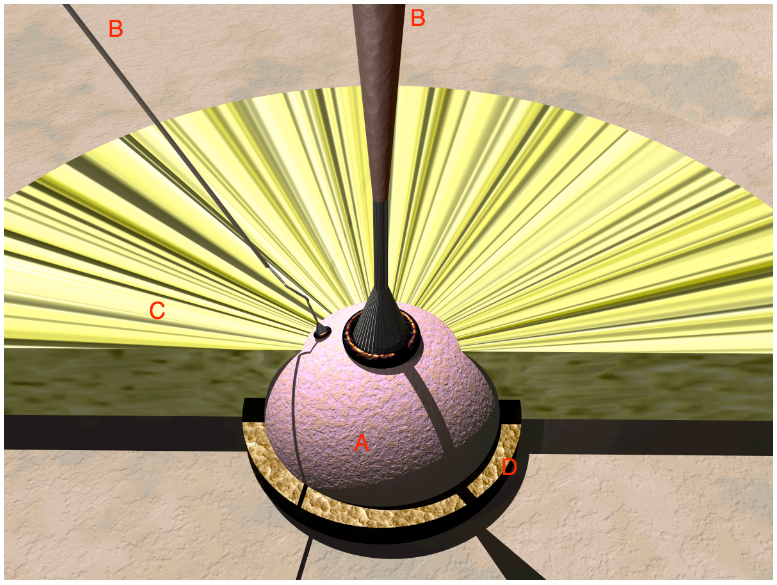

Figure 3: Conceptual depiction of exterior of U-machine with components labeled. (A) Exterior of hemispherical data-space. (B) Two transducer connections (one high dimensional, one low) to transduction region of data space. (C) Disk of configuration-control connections to computation region of data space, used to configure interconnection paths in program space (front half of disk removed to expose data space); signals applied through this disk define connection paths in program space (see Sec. 4.7). (D) Auxiliary support (power etc.) for data and program spaces.

connection weights. This can be accomplished by selecting, from outside the sphere, the input and output areas for a particular primitive function, and then applying signals to set the interconnection coefficients correctly (e.g., as given by $\mathbf{A} = \mathbf{R}^+\mathbf{V}$). We have in mind a process analogous to the programming of a field-programmable gate array. The exact process will depend, of course, on the nature of the computational medium.

4.7 Analogies to Neural Morphogenesis

In may be apparent that the U-machine, as we have described it, has a structure reminiscent of the brain. This is in part a coincidence, but also in part intentional, for the U-machine and the brain have similar architectural problems to solve. Therefore we will consider the similarities in more detail, to see what we can learn from the brain about configuring a U-machine.

In a broad sense, the variable space corresponds to neural grey matter (cortex) and the function space to (axonal) white matter. The grey matter is composed of the neuron cell bodies and the dense networks of synaptic connections; the white matter is composed of the myelinated axons conveying signals from one part of the brain to another. The division of the two-dimensional space of Hilbert coefficients into disjoint regions (implicit in the synaptic fields) is analogous to the division of the cortex into functional areas, the boundaries between which are moveable and determined by synaptic connections (see also Sec. 3.2). (The same applies in a field computer, in which the subdomains correspond to cortical maps.) Bundles of axons connecting different cortical areas correspond to computational pathways in program space implementing functions between the variable regions of the U-machine.

In order to implement primitive functions, bundles of connections need to be created from one variable region to another. Although to some extent variable regions can be arranged so that information tends to flow between adjacent regions, longer distance connections will also be required. Since the U-machine is intended as a machine model that can be applied to computation is bulk materials, we cannot, in general, assume that the interior of the U-machine is accessible enough to permit the routing of connections. Therefore we are investigating an approach based on neural morphogenesis, in which chemical signals and other guidance mechanisms control the growth and migration of nerve fibers towards their targets. By applying appropriate signals to the input and output areas of a function and by using mechanisms that prevent new connection bundles from interfering with existing ones, we can configure the connection paths required for a given computational task. The growth of the connections is implemented either by actual migration of molecules in the bulk matrix, or by changes of state in relatively stationary molecules (analogous to a three-dimensional cellular automaton).

Computation is accomplished by a linear combination of basis functions applied to an input vector. Mathematically, the linear combination is represented by a matrix

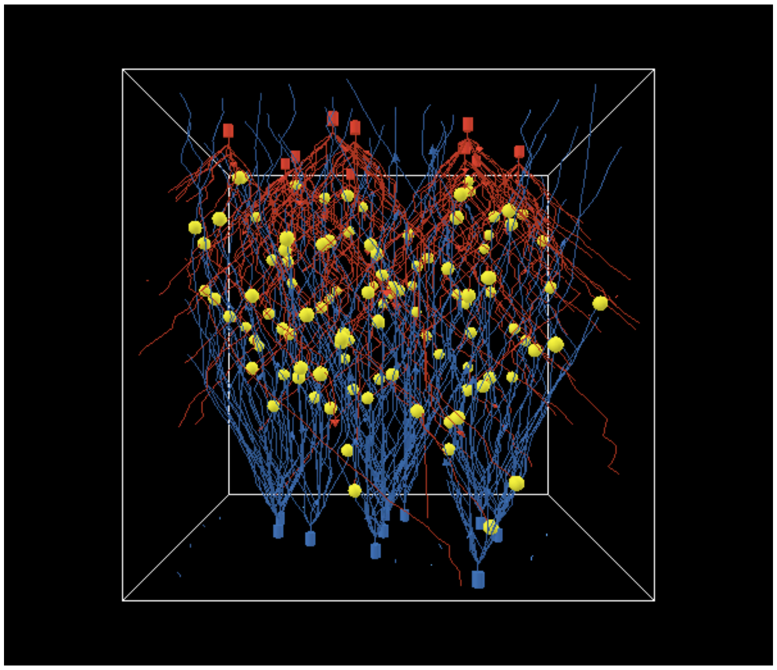

Figure 4: Simulation of growth of synaptic field, with ten "axons" (red, from top), ten "dendrites" (blue, from bottom), and approximately 100 "synapses" (yellow spheres).

multiplication, and the basis functions typically take the form of a fixed nonlinearity applied to a linear combination of its inputs (recall Sec. 4.2). In both cases we have to accomplish a matrix-vector multiplication, which is most easily implemented by a dense network of interconnections between the input and output vectors. The interconnection matrices may be programmed in two steps.[2]

In the first phase a *synaptic field* of interconnections is created by two branching growth processes, from the input and output components toward the space between them, which establish connections when the fibers meet. Figure 4 shows the result of a simulation in which "axonal" and "dendritic" fibers approach from opposite directions in order to form connections; fibers of each kind repel their own kind so that they spread out over the opposite area. (A fiber stops growing after it makes contact with a fiber of the opposite kind.) The existence of such a synaptic field effectively defines a variable area within the data space.

The pattern of connections established by growth processes such as these will not, of course, be perfect. Thanks to the statistical character of neural network-style computation, such imperfections are not a problem. Furthermore, the setting of the matrix elements (described below) will compensate for many errors. This tolerance to imprecision and error is important in nanocomputation, which is much more subject to manufacturing defects than is VLSI.

[2]Both of these processes are reversible in order to allow reconfiguration of a U-machine.

Once the interconnection matrix has been established, the strength of the connections (that is, the value of the matrix elements) can be set by applying signals to the variable areas, which are accessible on the outside of the spherical ("cortical") shell (Fig. 3). For example, an $m \times n$ interconnection matrix $\mathbf{A}$ can be programmed by

$$\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} \mathbf{v}_i \mathbf{u}_j^{\mathrm{T}},$$

where $\mathbf{v}_i$ is an m-dimensional unit vector along the i-th axis, and $\mathbf{u}_j$ is an n-dimensional unit vector along the j-th axis. That is, for each matrix element A_{ij} we activate the i-th output component and the j-th input component and set their connection strength to A_{ij}. In some cases the matrix can be configured in parallel.

As mentioned before, we have in mind physical processes analogous to the programming of field-programmable gate arrays, but the specifics will depend on the material used to implement the variable areas and their interconnecting synaptic fields. Setting the weights completes the second phase of configuring the U-machine for computation.

5 Conclusions

In summary, the U-machine provides a model of generalized computation, which can be applied to massively-parallel nanocomputation in a variety of bulk materials. The machine is able to implement quite general transformations on abstract spaces by means of Hilbert-space representations. Neural morphogenesis provides a model for the physical structure of the machine and means by which it may be configured, a process that involves the definition of signal pathways between two-dimensional data areas and the setting of interconnection strengths within them.

Future work on the U-machine will explore alternative embeddings and approximation methods, as well as widely applicable techniques for configuring interconnections. Currently we are designing a simulator that will be used for empirical investigation of the more complex and practical aspects of this generalized computation system.

6 References

[1] S. Haykin (1999), *Neural Networks: A Comprehensive Foundation,* 2nd ed., Upper Saddle River: Prentice-Hall.

[2] B.J. MacLennan (1987), Technology-independent design of neurocomputers: the universal field computer, in M. Caudill & C. Butler (ed.), *Proc. IEEE 1st Internat. Conf. on Neural Networks,* Vol. 3, New York: IEEE Press, pp. 39–49.

[3] B.J. MacLennan (1990), *Field computation: a theoretical framework for massively parallel analog computation, Parts I–IV*, Technical Report CS-90-100, Dept. of Computer Science, Univ. of Tennessee, Knoxville.

[4] B.J. MacLennan (1993), Field computation in the brain, in K.H. Pribram (ed.), *Rethinking Neural Networks: Quantum Fields and Biological Data*, Hillsdale: Lawrence Erlbaum, pp. 199–232.

[5] B.J. MacLennan (1994), "Words lie in our way", *Minds and Machines* **4**: 421–437.

[6] B.J. MacLennan (1994), Continuous computation and the emergence of the discrete, in K.H. Pribram (ed.), *Origins: Brain & Self-Organization*, Hillsdale: Lawrence Erlbaum, pp. 121–151.

[7] B.J. MacLennan (1997), Field computation in motor control, in P.G. Morasso & V. Sanguineti (ed.), *Self-Organization, Computational Maps and Motor Control*, Amsterdam: Elsevier, pp. 37–73.

[8] B.J. MacLennan (1999), Field computation in natural and artificial intelligence, *Information Sciences* **119**: 73–89.

[9] B.J. MacLennan (2004), Natural computation and non-Turing models of computation, *Theoretical Computer Science* **317**: 115–145.

[10] B.J. MacLennan (2005), *The nature of computing — computing in nature*, Technical Report UT-CS-05-565, Dept. of Computer Science, Univ. of Tennessee, Knoxville. Submitted for publication.

[11] T.O. Moore (1964), *Elementary General Topology*, Englewood Cliffs: Prentice-Hall.

[12] V.V. Nemytskii & V.V. Stepanov (1989), *Qualitative Theory of Differential Equations*, New York: Dover.

[13] M.B. Pour-El (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.

[14] L.A. Rubel (1981), A universal differential equation, *Bulletin (New Series) of the American Mathematical Society* **4**: 345–349.

[15] L.A. Rubel (1993), The extended analog computer, *Advances in Applied Mathematics* **14**: 39–50.

[16] C.E. Shannon (1941), Mathematical theory of the differential analyzer, *Journal of Mathematics and Physics of the Massachusetts Institute Technology* **20**: 337–354.

[17] G.F. Simmons (1963), *Topology and Modern Analysis*, New York: McGraw-Hill.