The Nature of the Extended Analog Computer

Jonathan W. Mills

Leverhulme Trust Professor
CEMS
The University of the West of England
Bristol, BS16 1QY, UK
Jonathan4.Mills@uwe.ac.uk

Associate Professor
Computer Science Department
Indiana University
Bloomington, Indiana, 47405, USA
jwmills@cs.indiana.edu

Abstract

During the past decade, researchers have asked fundamental questions about the nature of Rubel's extended analog computer, the EAC. The questions have made it clear that the design, implementation, and applications of the EAC are based on a paradigm unfamiliar to most users of conventional digital computers. The basic difference is that the EAC's components visibly implement only a few explicit functions. The rest are implicit, being properties of nature that are described mathematically. A new approach to bridge the "paradigm gap" is needed. This paper introduces the Δ-digraph, a directed graph that can be labeled to show how both unconventional and conventional computers relate nature, mathematics and computer architecture. The Δ-digraph defines a semantic hierarchy that bridges the paradigms of **analogy** and **algorithm**. It shows how many applications for the EAC are developed by choosing the semantics for an analogy, rather than programming an algorithm. Finally, concise case studies show how society is in the early stages of adopting the EAC. These applications suggest the future for unconventional computers.

PACS codes and keywords:

Computer Hardware 07.05.Bx, Very Large Scale Integration (VLSI) 85.40.–e, Computational Techniques–Mathematics 02.07.–c, Partial Differential Equations 02.30.Jr

Other Keywords:

Analog Computing, Analogy–Diagrammatic Semantics, Extended Analog Computer, Lukasiewicz Logic

1. Introduction

What is the EAC? Why does it work, since some of its components do not obviously perform computation? How are applications developed? What can it teach us about the nature of computing?

These questions arise because previous publications about the EAC and its predecessors, the Kirchhoff-Lukasiewicz machine (KLM) [1] and Lukasiewicz logic arrays (LLAs) [2], [3], [4], did not explain *how to think about its paradigm,* ***analogy***. Although the paradigm it used was unfamiliar, it was not explained. As a result, the typical reader found that the EAC was too confusing to understand, even though the idea of an unconventional computer was appealing. A new way of thinking about the EAC is needed. So far, it has only been acquired after hands-on experience by the people who have used it.

This paper takes a new tack by disassembling the EAC prototype to reveal its parts. Next, the Δ-digraph is derived, which is a visual diagram that helps distinguish the explicit and implicit functions of the EAC. The Δ-digraph is then used to explain the unconventional paradigm **analogy**, as contrasted to the conventional computing paradigm of **algorithm**.

Applications of the EAC are shown to be semantic constructs chosen by the user and mapped to simple machine configurations. The configurations are simple, but may be difficult to find, and are often evolved rather than programmed.

2. At first glance

Pictured below is the current implementation of *Rubel's extended analog computer* (EAC) designed by Bryce Himebaugh at Indiana University in 2005 [5]. It is connected to a digital computer host, either a PC or a Macintosh, which runs the *EAC operating system* (jEAC) designed by Ryan Varick at Indiana University in 2006 [6].

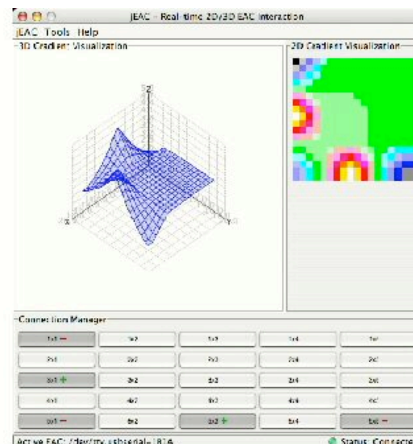


Figure 1. (a) EAC to left of Macintosh (b) jEAC Interface in operation

The EAC is the board to the left of a Macintosh computer (Figure 1a). The blue LEDs indicate that the EAC configuration is actively computing a neural-tissue model of exclusive-OR. The components of the EAC that perform the computation will be revealed in the next section with photographs of the disassembled board.

The jEAC operating system, running on the Macintosh iBook to the right (Figure 1b), provides a visual interface to the EAC. It is based on a coordinate array, similar to a

spreadsheet. The user may assign one function to each of the twenty-five connections to the conductive foam sheet. Users may also access the interface language of the EAC, and configure its operation automatically.

However, many functions of the EAC are invisible to the human eye. They are implicit in the properties of materials, described by laws of nature that have emerged from the observations, experiments and mathematical descriptions of scientists (see Feynman [7] and [8]). The Δ-digraph relates these functions to the visible components of the EAC.

3. The visible components of the EAC

The EAC R002 (research prototype, version 002) is constructed with a two-sided motherboard. Both the top and the bottom sides are covered with clear Plexiglas to protect the components. A square area is covered with a separate translucent Plexiglas sheet to diffuse the light emitted by an output array of blue LEDs (Figure 2). The four holes that contain small pins are visible as dark spots on the middle LEDs along each edge of the translucent Plexiglas. They allow boards to be interconnected at the north-south-east-west edges for use as a cellular automaton, or stacked for 2½D applications.

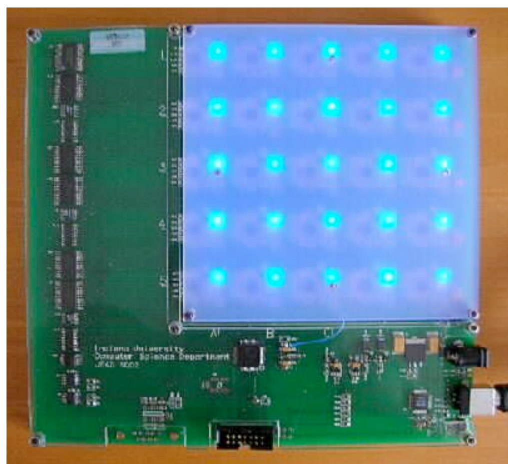
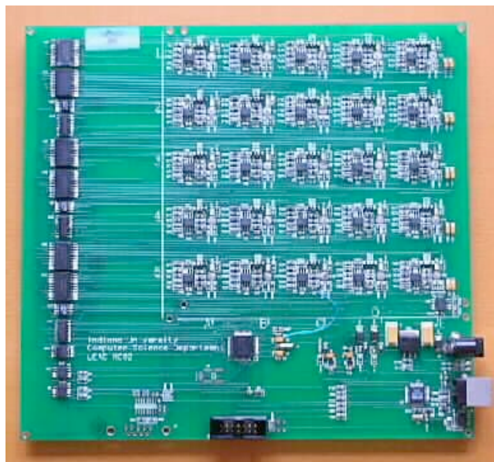


Figure 2. Upper side of EAC R002 during operation

When the protective sheets are removed (Figure 3a), the five-by-five input/output array of LEDs, programmable current sources and sinks and the surface mounted capacitors is exposed. To the left are the analog-to-digital and digital-to-analog converters (ADCs and DACs). In the lower middle of the board, beneath the input/output array, is the microprocessor that controls the array and the USB interface. It also digitally generates Lukasiewicz logic array (LLA) functions using interpolation tables, and stores the configuration parameters for the EAC. At the lower right are the power regulator and the USB interface circuits. A serial port used for testing is at the bottom of the board.



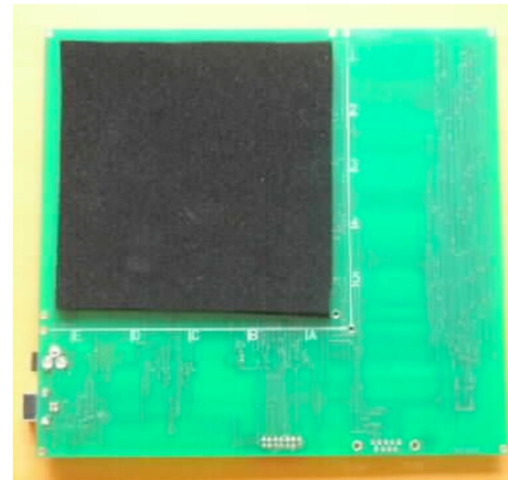


Figure 3. (a) Upper side of EAC R002 (b) Underside of EAC R002

Attached to the opposite side of the board (Figure 3b), underneath the input/output array, is the primary computing element of the EAC, a sheet of conductive plastic foam. Other materials have been used, including silicon chips, gelatin [9] and slime mold [10]. The foam is the same substance often used to package digital integrated circuits. It is approximately 120mm x 120mm square, but unlike conventional digital computers, its dimensions are not critical. If one looks carefully, the irregular edges are apparent. This piece was cut from a larger sheet of foam with a pair of scissors.

The foam does not appear to be a computing element, especially to a naïve user. It is just some black “stuff” that has no apparent function. (Some users have asked if it can be removed after shipping?–No.) The connections to the circuitry on the top of the board, a five-by-five array of short pins, pierce the foam but do not extend through it.

That is all there is to the EAC. As described so far, the machine appears to be little more than a flexible user interface with some vague kind of computation performed by a piece of foam, and Lukasiewicz logic functions emulated with a digital microprocessor (in earlier designs, the Lukasiewicz logic functions were reconfigurable analog circuits [4]).

A user’s block diagram of the components used when the board is operated summarizes the design (Figure 4a and 4b). A detailed diagram may be found in a recent paper [11].

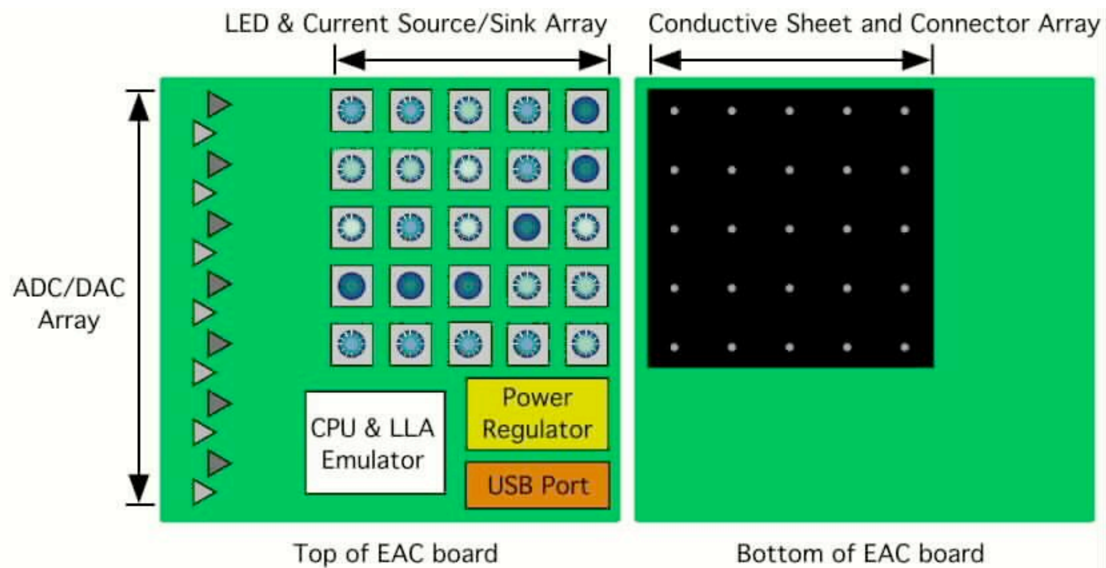


Figure 4. (a) Diagram of upper side of EAC R002 (b) Diagram of underside of EAC R002

The visible components of the EAC are simple. However, no study of analog computers from 1845 [12] through the 1950s [13], up to the author’s own research prior to this paper, ever questioned *why* these analog components worked. An exhaustive list of the

"Great Principles" of computing proposed by Denning primarily addressed the design and engineering of digital computers, but neither the physics nor the semantics of computers was discussed [14], [15]. Many philosophers and computer scientists have argued about what constitutes a computer. One view is mechanistic and functional: a computer is a distinct device, a "large" form of calculator (see Searle [16] and Piccinini [17]). Another view is that computing devices are defined semantically, but no formalism is given that defines how the semantics relate to the mechanism (see Shagrir [18]). In a concept paper Costa and Mycka [19] suggested that natural objects compute by "endowment"—that is, by semantic attribution—but did not pursue the idea further:

> *" When we observe natural phenomena and endow them with computational significance, it is not the algorithm we are observing but the process. [But in the 1940's it was] digital technology and theory that was to become the main paradigm of computation…[processing] information transformed and coded in binary. With analog machines, on the contrary, there would be few or no steps between natural objects and the work and structure of computation."*

A novel argument was proposed by Crowcroft [20], stating that the nature of computing is a unique universe of discourse, detached from physical reality:

> *" Computing has never established a simple connection between the natural and the mathematical. ...computing represents a third place in the world of discourse—distinct from the natural and the artificial of science and engineering. Computing involves (virtual) systems that may never exist, either in nature or through human creation."*

Prior discussions of the meaning of computation are relevant, but not helpful without a formalism to understand the semantics of real computers, both unconventional and conventional. The persistent questions about the EAC indicate that the “Great Principles” of digital computing are insufficient to understand it. A simple connection between nature, mathematics and computer architectures must be found to comprehend this simple machine. We need a way to define “computational significance” in the function and structure of natural objects used as computers. We need a tool to study **analogy**.

4. The “invisible components” of the EAC

The rest of the components of the EAC are inherent *functions* of the materials that it is composed of, which is to say that they are mental constructs that can only be understood as properties of nature whose representations are physical and mathematical principles.

The primary method of identifying these inherent functions is through the computing paradigm of **analogy**. The only visible form of any specific analogy is a configuration of the EAC, that is, a set of inputs, outputs and Lukasiewicz logic functions assigned according to a topology of the conductive sheet. More complex analogies may need multiple EACs. Thus, the major parts of most analogies are invisible to the user. They reside in the properties of nature and the semantics that are assigned to the EAC.

This has proved confusing to most prospective users. Previous papers did not clarify matters because a single application merged multiple semantic levels that were overlaid onto the visible components of the EAC. This paper presents the Δ-digraph to define the implicit functions of the EAC. The Δ-digraph also illustrates the nature of its computing

paradigm **analogy**, which will be contrasted with **algorithm**, and used to "dissect" the semantics of the EAC's components and two EAC applications.

This is a departure from the strategy used in earlier papers, which began by describing Rubel's mathematical model of the EAC [23], naming all visible components (and confusing the components with their inherent functions), then presenting one or two applications without an explanation of their semantics. In this paper, the *paradigm* to understand the EAC will be derived before discussing the *model*. This will allow the reader to distinguish the EAC (and other unconventional computers) from traditional digital computers. It will clarify the relationship between the EAC's components (shown in Figures 3 and 4), Rubel's mathematical model of the EAC (which will be given) and the properties of the materials whose inherent functions perform the computation.

5. Deriving the Δ-digraph

The Δ-digraph originated in a difficult question, "How are partial differential equations compiled to the extended analog computer?" The first application of the EAC, butterfly wing pattern morphogenesis, was designed by translating a visual model of the butterfly wing to the conductive sheet. The model was developed by Nijhout [21], who had also presented a system of PDEs for pattern generation based on work by Turing [22].

Mathematical models are extracted from physical systems by scientists, not automatically recognized by a computer (in fact, Turing had no computer capable of this task). The first path in the diagram (Figure 5), developing partial differential equations (PDEs) from nature, is a task for natural intelligence. In the second path in the diagram, the author mapped Nijhout's model through topology-preserving deformations to a material that had

properties similar to the diffusion of chemicals found in a butterfly's wing. However, for the third path, no method was evident to compile the system of PDEs to the EAC.

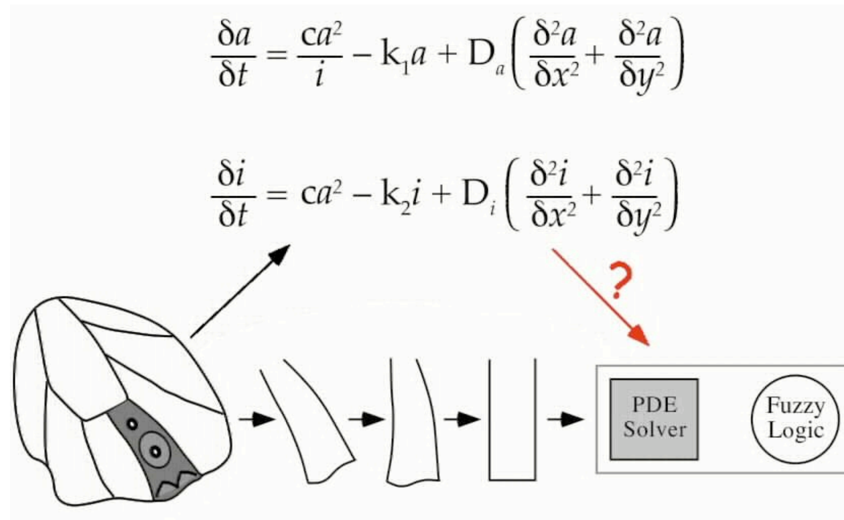


Figure 5. Developing a specific analogy for the EAC

This diagram was used in conference presentations for several years before realizing that it contained the key to the problem. Three things define the large-scale semantics of any computing paradigm, whether it is analog or digital:

1. *Mathematics,* which is a language to express mental ideals that rigorously describe objects, processes and the relationships between them; this includes logic and theoretical models of computing such as Turing's machine, Rubel's EAC, etc.),

2. *Nature,* which is everything found in the physical universe; it includes those objects that are used for computation as well as those that are not, including artificial constructs, and

3. *Computer architectures,* which are either objects that are expressly built for computation (solving problems), or objects, materials and processes that are used for computation by structuring them or ascribing a semantics to their behavior; thus all computer architectures are embedded in nature.

The diagram in Figure 5 and these ideas are formalized as the Δ-*digraph* (Figure 6). It is a set of tuples, each composed of an order-3 directed graph whose edges and vertices are labeled with definitions for the specific semantics of one "level" *i* in a computing system:

$\{((V_0, E_0), L_{V_0}, L_{E_0}), ((V_1, E_1), L_{V_1}, L_{E_1}), ((V_2, E_2), L_{V_2}, L_{E_2}), \ldots, ((V_n, E_n), L_{V_n}, L_{E_n}), (E_a, L_b)\}$ where $|V_i| = 3$, $|E_i| \geq 0$, $|L_{V_i}| \geq 0$, $|L_{E_i}| \geq 0$, $|E_a| \geq 0$ and $|L_b| \geq 0$

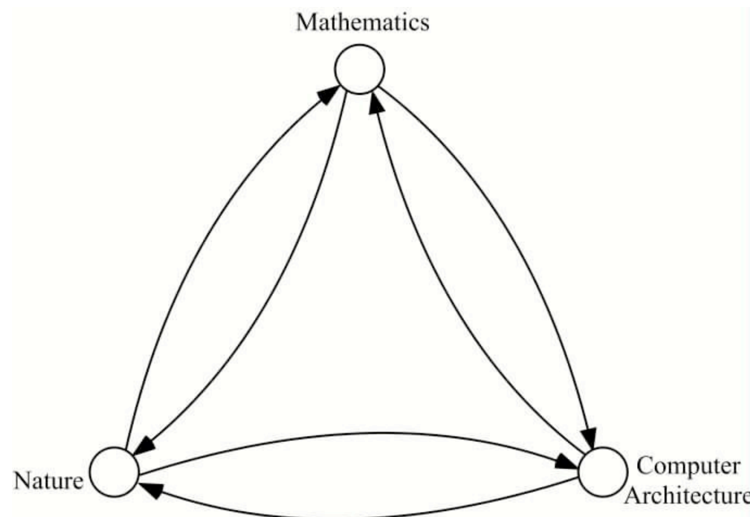


Figure 6. One level of a general Δ-digraph

Experience has shown that the semantics of a computing system may be known only partially. The possibility of a partial, incomplete semantics is permitted by allowing the number of edges and labels to be zero at any level. No upper bound is set on the number of tuples, nor within each tuple on the number of edges and labels, although they must be

finite. Composition of tuples is permitted using an additional tuple in the set whose edges and labels define “overloaded” meanings in both the paradigms **analogy** and **algorithm**.

The leftmost and topmost nodes in the diagram are not restricted to any specific sub-discipline of natural science or mathematics. The rightmost node is restricted to computer architectures, since that is the discipline we are investigating. However, one could change it to "swimming machines" or "buildings" or any artificial device whose semantics one wants to define. The diagram is used by restricting the three vertices and edges with a labeling, and then seeing what semantics the relationships defined by the labeling will yield. Conversely, a semantics such as “digital computer modeling weather prediction” can be mapped to the Δ-digraph, and the unlabeled parts of the diagram explored.

6. Applying the Δ-digraph to understand **analogy**

The computing paradigm **analogy** differs from **algorithm** because it uses *implicit computation,* the idea that the properties of nature inherently perform computation. An analogy is a specific relationship binding a computer architecture directly to nature. As the term is employed here, an analogy is restricted to devices that do not need to be programmed, but may need to be configured. A configuration specifies the structure of functional elements, thus selecting the spatial arrangement of the elements, their inputs, and outputs, and the means of measuring or observing the machine. The computational complexity is hidden in the material out of which the EAC is fabricated. Here is the Δ-digraph for the paradigm **analogy** as illustrated by butterfly wing pattern morphogenesis modeled on the EAC. The explicit, “active” parts of the analogy are shown in red, while the implicit relationships are shown in black (Figure 7).

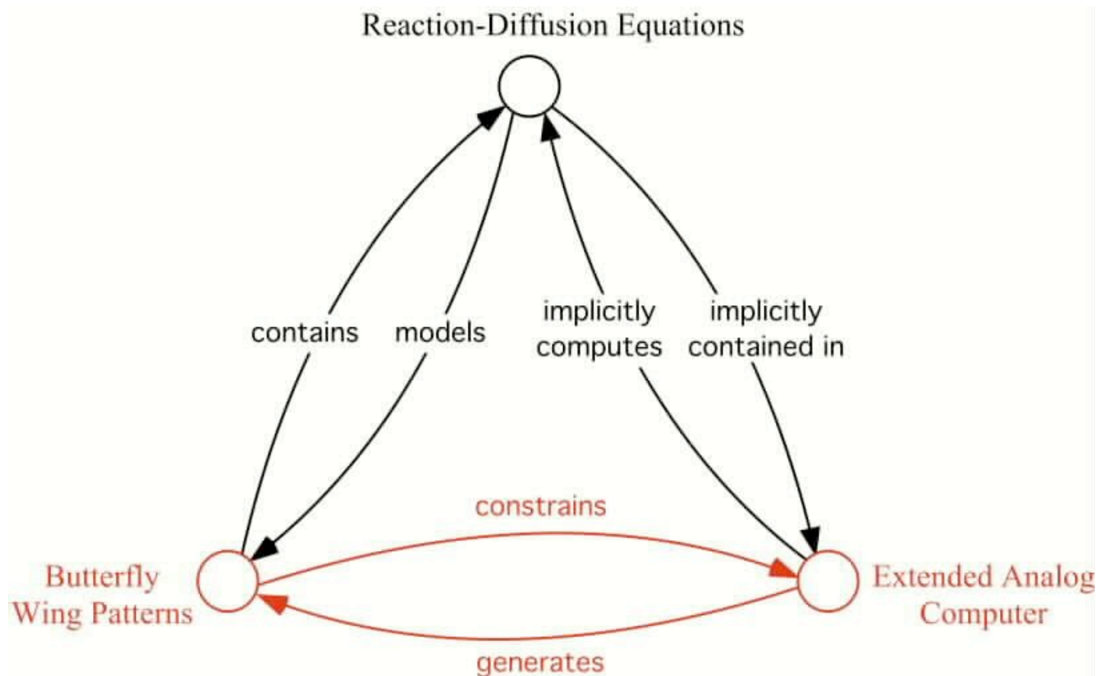


Figure 7. Diagrammatic representation of the computing paradigm **analogy**

An analogy may be simple, yet the system may be complex. Consider an EAC that models embryonic growth at the cellular level. It might need tens of thousands of conductive sheets, each with several Lukasiewicz logic elements, to react to temporally varying conditions, but such a machine is composed of simple components (like a cellular automaton). It need not even use more silicon area than a digital computer; it would simply use it in a different manner. Its behavior is complex, not its computer architecture.

7. Applying the Δ-digraph to understand **algorithm**

The paradigm **algorithm** differs from **analogy** because the computing device must be explicitly "instructed" to perform each operation, and to explicitly generate an output representation (a device using **analogy** often, but not always, displays its output in the material out of which it is composed). The visual display of a modern computer

originates in its numeric output, usually large tables, matrices or lists, which are translated into colors at positions on a graphical display. The paradigm **algorithm** for butterfly wing pattern morphogenesis expressed using the Δ-digraph is shown below (Figure 8). Where the EAC, once configured, has no further need to use the mathematical model, a conventional digital computer will repeatedly perform computations based on some algorithm that is a direct translation of the equations in the model. Initially, a human must write the specification—a computer program—based on the equations. It does not matter what language the person uses, nor does it matter that in most cases today the program is translated automatically—compiled—into a form that the digital computer recognizes. The explicit, "active" relationships in this example are shown in red.

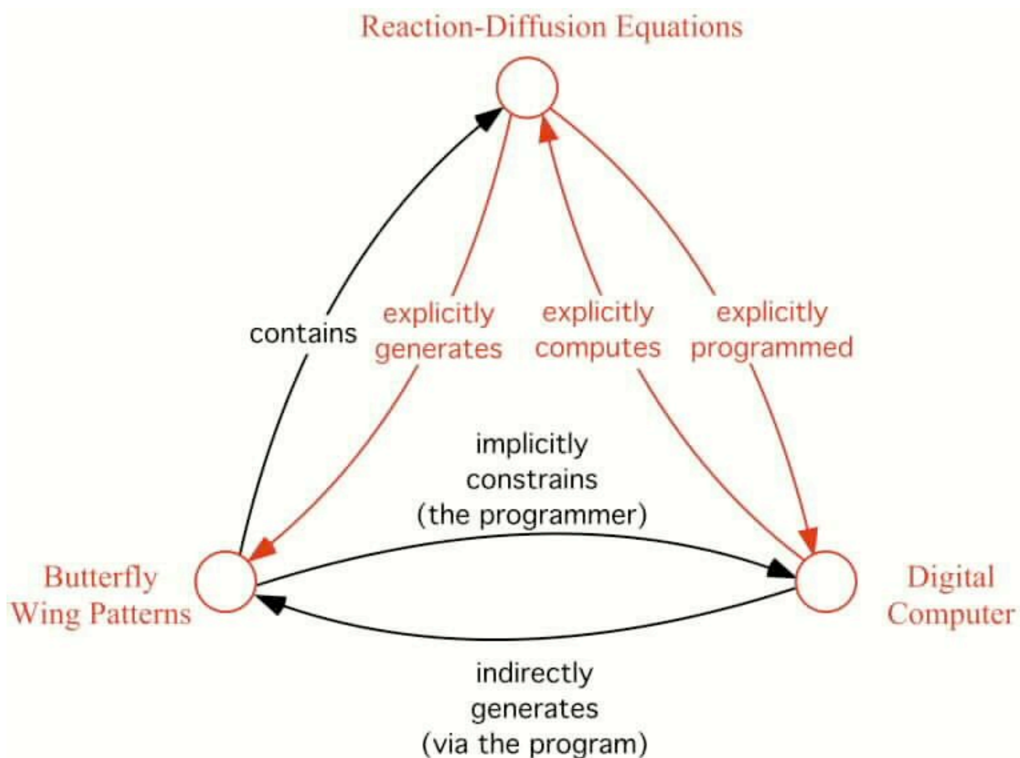


Figure 8. Diagrammatic representation of the computing paradigm **algorithm**

The point is that the paradigm **algorithm** cannot escape from the need to execute the specification. Its original form is a mathematical model composed of smaller objects whose semantics are given by logic and arithmetic. The EAC does not do this; it “runs” its configuration using the properties of matter. Using **analogy**, nature computes.

8. Understanding implicit and explicit functions

To understand why the components of the EAC work, the difference between the *explicit* and *implicit functions* of the components in digital and analog computer architectures must be understood. Intuitive and diagrammatic definitions are given below:

> A function is *explicit* if the structure of the component is one-to-one, reflexive and symmetric with its definition in the model. (The need for a component defined by the model is identified with a separate edge). The significant relations are shown with two edges in the Δ-digraph. One connects the model to the component, defining its structure. The second connects the component to the model, indicating that its definition can be extracted from the component.
>
> A function is *implicit* if the structure of the component that implements it does not directly correspond to the definition given in the mathematical model. There are no defining edges in the Δ-digraph between the model and component that are one-to-one, reflexive and symmetric. The structure of the component is not entailed by the definition given in the model. The operation of the component is denoted by some path in the Δ-digraph that leads to the model, but the definition in the model cannot be extracted from component’s structure.

Two examples will clarify these definitions, and illustrate the difference between a conventional digital computer and the EAC. First, consider the *explicit function* of *addition* in a conventional digital computer (Figure 9). It is specified by a complex expression in Boolean logic that defines the physical structure of the adder in terms of logic gates. In turn, from this structure the logical definition of addition can be extracted.

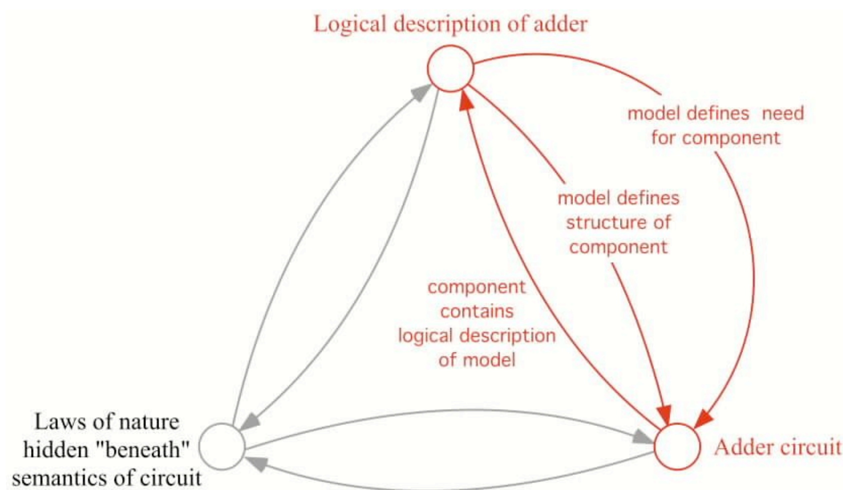


Figure 9. Explicit addition

In contrast, *addition* in an analog computer is an *implicit function* (Figure 10).

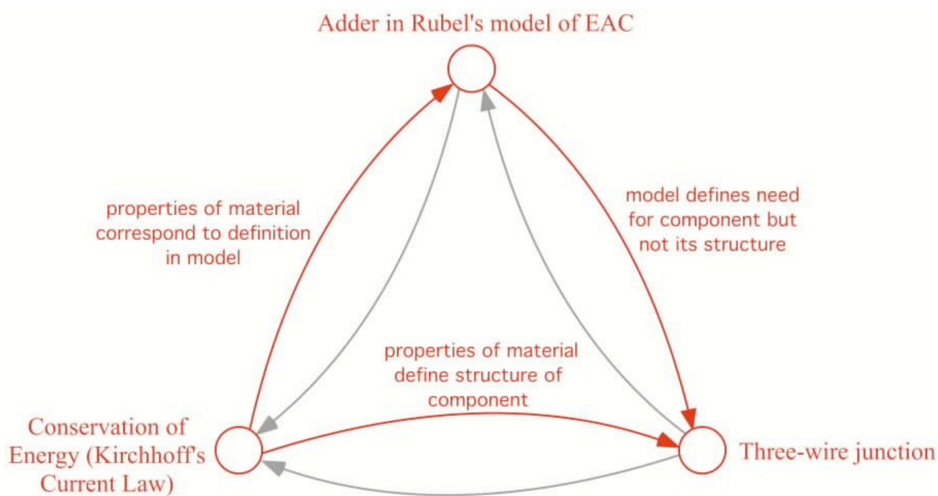


Figure 10. Implicit addition

Addition can be implemented with a three-wire junction, but there is no sentence in Boolean logic that directly (or usefully) corresponds to its structure. This adder-without-binary-bits uses the Law of Conservation of Energy, specifically, Kirchhoff's Current Law. Moreover, while a three-wire junction performs addition, it is not a logical necessity that it do so. For example, it may apply a common voltage to the gates of two field-effect transistors in a current mirror.

A more complex arithmetic function in the EAC is *multiplication* (Figure 11). It can be implemented in several ways. One is by scaling an input with the conductive sheet, an *implicit* function. Another is with the slope of a Lukasiewicz logic function, an *explicit* function. A two-level Δ-digraph with the pertinent vertices and edges highlighted in red shows the semantic hierarchy of Lukasiewicz logic used define multiplication. From the structure of the LLA circuit, the logical definition of multiplication can be extracted.

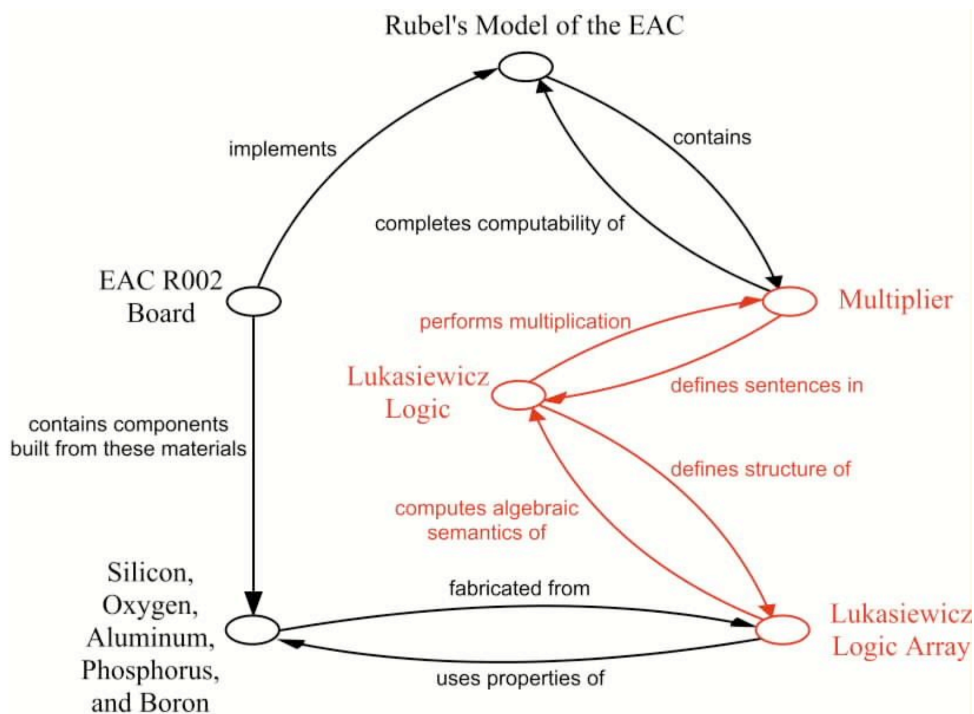


Figure 11. Explicit analog multiplication

9. The explicit functions of the EAC

Rubel's EAC [23] is such a simple device that there is some degree of implicitness in all of its functions. However, paraphrasing Orwell's novel *Animal Farm,* some functions are more explicit than others. The degree to which a component of the EAC uses the properties of matter to "carry" a computing function, and the visibility of the function's operation with respect to the component, together with the relationship criteria of the function typing (Section 8), determines its classification as explicit or implicit.

Initial setting and constants. Initial settings $s_1, s_2, s_3, \ldots$ and constants $c_1, c_2, c_3, \ldots$ are fixed, arbitrary real numbers that are not required to be rational or digitally computable, that is, able to be generated by a Turing machine. In nature, this assumption may not hold true, which is beyond the scope of discussion in this paper (see Siegelmann and Sontag [24] and Landauer [25] for different perspectives). The distinction between the initial settings and constants is that initial settings are only produced at the first level, N=0, in the machine, while constants may be produced at any level. Both are *explicit* inputs to the EAC. In EAC R002, their values are produced by the output of DACs that are precise to ten bits, which control current sources of slightly less precision due to noise. These outputs are connected to the conductive sheet and the Lukasiewicz logic units.

Independent variables. These variables, $x_1, x_2, x_3, \ldots$, are arbitrary real numbers produced at any level N of the EAC. Rubel did not require that they be measurable. Independent variables *explicitly* correspond to measurement points in the EAC. In the current version of the EAC, they are voltages measured directly by ADCs that are precise

to 12 bits, or to currents that can be computed digitally from measured voltages using Ohm's Law and sent back as inputs to a DAC.

Hook-ups and feedback (wires). The initial settings, constants, independent variables, and the inputs $u_1, u_2, u_3, \ldots$ and outputs $v_1, v_2, v_3, \ldots$ of all the varieties of "black boxes" are *explicitly* connected with wires defined by pairs (w_i, w_j) from the Cartesian product $W = (s_1, s_2, s_3, \ldots \times c_1, c_2, c_3, \ldots \times x_1, x_2, x_3, \ldots \times u_1, u_2, u_3, \ldots \times v_1, v_2, v_3, \ldots)$. There are three constraints: (1) the outputs of any level N can only be used as inputs at level N+½, N+1 and higher, (2) no two outputs can be connected to the same input, and (3) each input must be connected to at least one output.

Rubel was vague about the topology of interconnections, but stated that there was "a great deal of feedback." Yet, according to his definition, feedback is only possible locally within a half-level N or N+½, but not more widely. Our implementations have not followed this restriction, although in general the machine operates without feedback as computation progresses "upward" from one level to the next. Finally, in the EAC R002, the topology of the connections is configurable.

Wires are *explicit functions* in the connections between components, but are *implicit functions* within the conductive sheet, where all connection points in the foam sheet are "wired together" by its conductive properties.

Lukasiewicz logic arrays (LLAs). Lukasiewicz logic arrays are *not* part of Rubel's definition of the EAC. The piecewise linear logic functions they compute are *explicit functions* (as was shown earlier for multiplication, Figure 11). LLAs are included because of a theorem by McNaughton [26], which proved that sentences in Lukasiewicz logic

approximate algebraic differential equations (ADEs) arbitrarily closely. Because the EAC computes ADEs as wells as PDEs, this functionality is necessary to complete the implementation of an EAC.

10. The implicit functions of the EAC

"Boundary-value problem" box. Rubel considered this component to be the most significant element of the EAC, saying that *'The quintessential black box is the "boundary-value-problem" box...[that] solves a finite system Σ of PDEs (maybe including some ODEs)...'"* [23]. The EAC R002 implements the *implicit* solution of both PDEs and ODEs using the conductive foam sheet. A specific system Σ of PDEs is solved by defining various configurations of the EAC, and selecting a new analogy (semantics).

The "boundary-value problem" box—a conductive sheet—has been known since the 1950's to solve Laplacian and Poisson PDEs using materials such as carbon paper or resistive films [13]. In the EAC, silicon and conductive plastics apply this historical technique with modern materials, directly accessing the properties of conducting or semiconducting materials (charge recombination-regeneration, drift and diffusion). With oscillating inputs, the foam solves the wave equation. ODEs are solved *implicitly*, too, but can be defined for direct *explicit* solution [27].

Adders. Adders sum the vectors $u_1(x_1, x_2, x_3, \dots x_k)$ and $u_2(x_1, x_2, x_3, \dots x_k)$ to yield $u_1 (x_1, x_2, x_3, \dots x_k) + u_2 (x_1, x_2, x_3, \dots x_k)$ This operation is similar to the concatenated adders in a general-purpose analog computer. Adders are *implicitly* implemented in the conductive sheets and the Lukasiewicz logic units, whose operation is governed by Kirchhoff's Current Law, itself based on the Law of Conservation of Energy.

Multipliers. Multipliers input the vectors u_1 $(x_1, x_2, x_3, \ldots x_k)$ and u_2 $(x_1, x_2, x_3, \ldots x_k)$ to yield $u_1 (x_1, x_2, x_3, \ldots x_k) \cdot u_2 (x_1, x_2, x_3, \ldots x_k)$. This is similar to the concatenated multipliers in a general-purpose analog computer. Multipliers are *implicit* functions of the conductive sheet (scaling by a resistive constant) and *explicit* functions of the Lukasiewicz logic functions (scaling by the slope of a curve).

Differentiators. For $f(x_1, x_2, x_3, \ldots x_k)$ a differentiator outputs a possibly mixed partial derivative $Df(x_1, x_2, x_3, \ldots x_k)$ produced by:

$$Df = \frac{\partial^{a_1+a_2+a_3+\ldots+a_n} f}{\partial x_1^{a_1}\, \partial x_2^{a_2}\, \partial x_3^{a_3} \ldots \partial x_n^{a_n})}$$

Holding a variable x_i fixed is implemented by forcing it to a constant k, or, over a series of points in space, a function of the variable $f(x_i)$ such that it is not influenced by the partial derivatives of the other variables. Differentiators are *implicit* in the conductive sheets [28]. Simpler versions are *implicit* semantic attributions of Lukasiewicz logic elements, which model Laplacian differentiators, well known as edge detectors [4].

Substituters. For a vector of values $v(x_1, x_2, x_3, \ldots x_l)$ and the input vector $u_1(x_1, x_2, x_3, \ldots x_k)$, …, $u_l(x_1, x_2, x_3, \ldots x_k)$ the EAC replaces each x_i in $v(x_1, x_2, x_3, \ldots x_l)$ with the corresponding value $u_i(x_1, x_2, x_3, \ldots x_k)$ to yield $v(u_1(x_1, x_2, x_3, \ldots x_k), \ldots, u_l(x_1, x_2, x_3, \ldots x_k))$. Substituters are *implicit* functions of the *explicit* connections—the wires—from one component, such as a conductive sheet, to another, introducing the value(s) to be substituted from the outputs of other conductive sheets or Lukasiewicz logic functions.

Inverters. For a well-defined C^{ω} function, the inverter "locks down" the outputs and generates the inverse of the function, yielding the inputs $u_1(x_1, x_2, x_3, \ldots x_k)$, …, $u_l(x_1,$

$x_2, x_3, \ldots x_k$) that yield $f(u_1(x_1, x_2, x_3, \ldots x_k), \ldots, u_l(x_1, x_2, x_3, \ldots x_k))$. Inverters are semantic attributions of a level of the EAC, which *implicitly* solves the inverse of a function, for example, by implementing backpropagation in a neural network as used to implement a character recognizer with a single-pixel retina [28].

Set theoretic operators: >0, ≥0, union, intersection, projection. The set theoretic operators provide comparison and combinations of functions of variables $f(x_1, x_2, x_3, \ldots x_k)$. It should be noted that in finite time it is not possible to exactly compare any value to zero, because the sequence of digits in a real number such that its partial representation is 0.00000000… may have some digit beyond those checked that is non-zero. This is not an issue for a theoretical model, but is an example of the limits of measurability for a physical implementation. These operators are *implicit* in the connections through Lukasiewicz logic functions that have a constant value along a piecewise interval.

Restricted limits. The restriction on taking limits is enforced by only permitting boundary values that have been computed at the immediately prior level N–1 in the EAC. Permitting an unbounded series of boundary value computations would permit the EAC to compute all C^{ω}-functions. Then, as Rubel wrote, "we would have no "computer" at all" [23]. In implementation, this is an *implicit* constraint on the physical connections between components—and one that is ignore in the implementation, as there are too few levels, even in the proposed EAC supercomputer [11], to take "unrestricted" limits.

Analytic continuation. The analytic continuation "black box" is a mathematical property of a function such that one point defines other points with a neighborhood. Practically, this is a condition of interpolation, and barring discontinuities in the material from which

the EAC is fabricated, is *implicit* in the regularity of matter at the macroscopic, classical level, and the Laws of Symmetry and Conservation of Energy.

Extremely well-posed determinism. This form of determinism enforces a compact space on the output at any point in the computation. This does not say that there cannot be sharp gradients or rapid changes in a function, but it does demand that any perturbation by some small amount ε produces a change in the resulting output that is a C^{ω}-function $f(\varepsilon)$. Extremely well-posedness is an *implicit* function enforced by the Law of Conservation of Energy, which prevents gross discontinuities in the output of the EAC. For example, taking the derivative of a 0-to-1 step function, such as occurs at the edge of an image, is theoretically infinite but in practice is limited by the available power in the circuit. This was observed in Lukasiewicz logic arrays acting as Laplacian differentiators (Figure 20).

11. Examples of **analogy** in unconventional computers

Here are two problems solved with **analogy,** which are "dissected" using the Δ-digraph.

The first is Hamiltonian Circuit, an NP-complete problem that on the surface does not seem to be isomorphic with Butterfly Wing Morphogenesis. The link is the semantic concept of an abstract alphabet (see Section 13 in this paper). Students unwittingly revealed the first abstract alphabets by using Sandved's *Butterfly Alphabet* [29] (Figure 19a) to solve assignments on the EAC [30]. They needed a technique similar to the "Hello, world!" program used to introduce beginning students to programming languages such as C++ or Fortran. Morphogenesis served this purpose although its semantics are only now explained here for the first time (Figure 12). The author's students intuitively used the alphabet for these and other applications because no formalization existed.

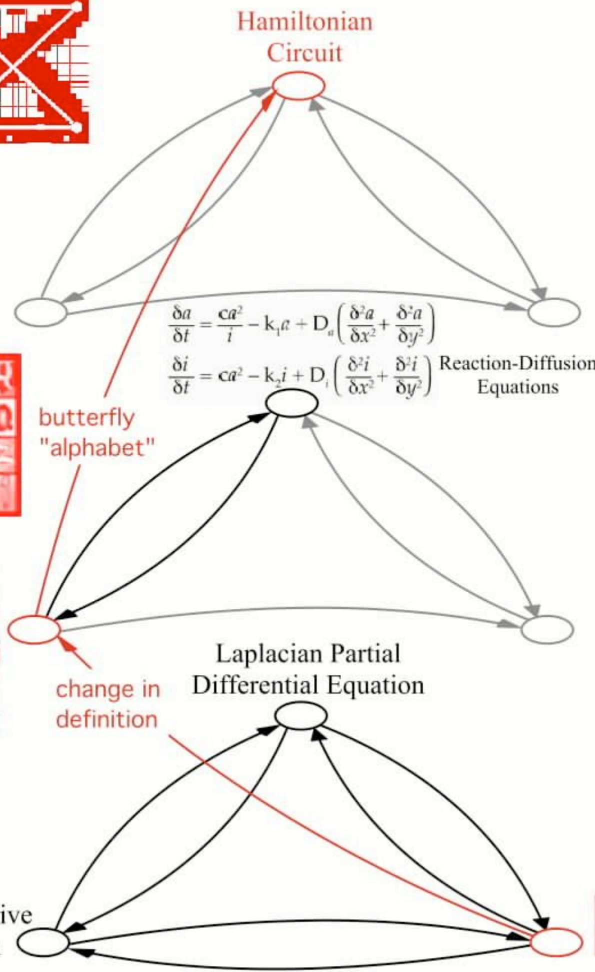


Figure 12.Semantic hierarchy of the EAC analogy for Hamiltonian Circuit

Abstract alphabets and the Δ-digraph together open interesting areas for research. What kinds of alphabets are there—spatial? Temporal? Three-dimensional? What grammars can be found that apply to different problems? Can *semantic complexity classes* be rigorously defined for **analogy**, in a manner similar to the complexity classes known for digital computers? Is P=NP in **analogy**, or do P and NP contain different classes of problems? Even for the same problem, does **analogy** separate its solutions into distinct complexity classes? (Costa and Mycka [19] outline a research program in this area.)

The second example shows that the Δ-digraph is useful to model other systems. Here, a slime mold, *Physarum sp.*, computes Minimum Spanning Tree [10], [31] (Figure 13). Only two levels in the semantic hierarchy are needed. The slime mold becomes a computer by configuring the structure of the organism's environment, then ascribing a meaning—an analogy—to the behavior of this living unconventional computer. The Δ-digraph also shows how an EAC could be mapped to this problem (compare Figure 12).

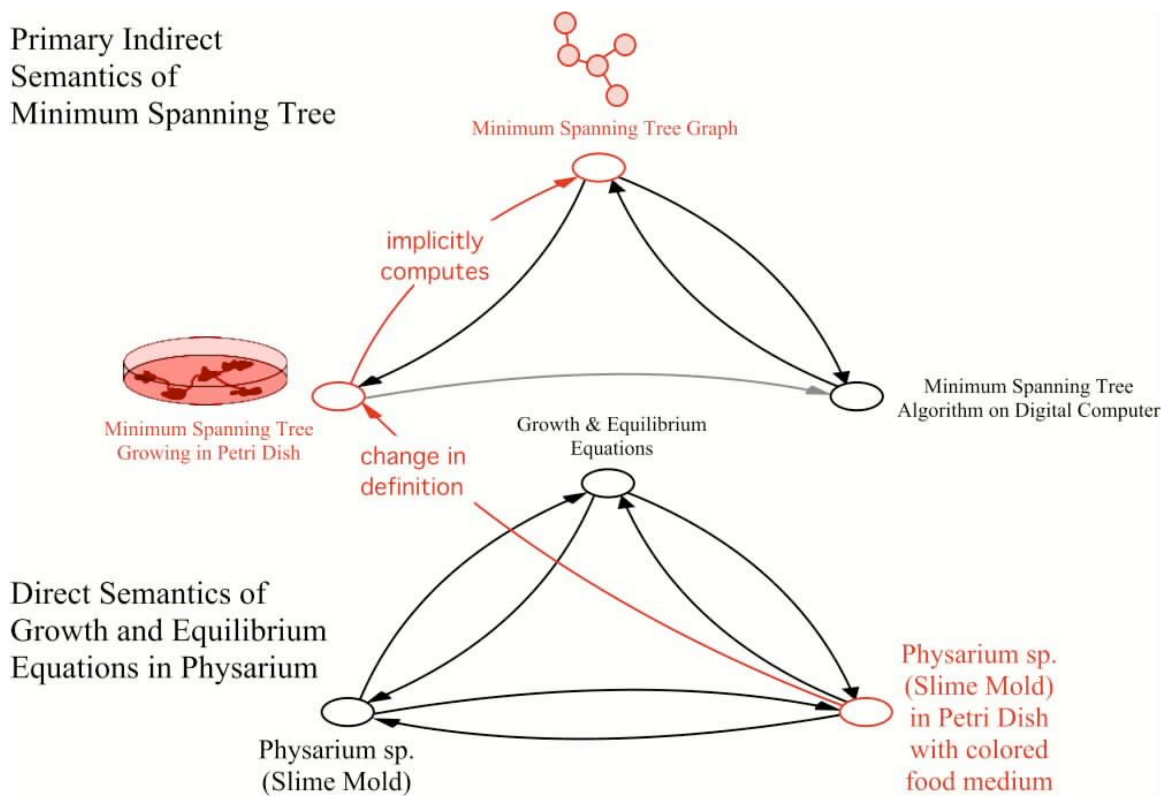


Figure 13. Semantic hierarchy of the *Physarum sp.* analogy for Minimum Spanning Tree

12. Practicality of the EAC

No unconventional computer will be adopted for its novelty. First, it must be understood to be useful. The Δ-digraph was developed because current and potential users of the

EAC have requested a starting point to develop their applications. But why was the EAC attractive to users in industry and the military? *Is the EAC a practical computer?*

Eight reasons have emerged that answer this question, “Yes.” The EAC is *fast, cheap, robust, low power, reconfigurable, scalable, “low tech,”* that is, capable of being fabricated with older VLSI process technology, and, in some cases, it is *disposable*.

Three of these reasons go hand-in-hand. They are *speed, cost and robustness,* which are all due to its simple structure. The EAC is a naturally *fast* architecture because it omits several of the levels of computation that are used in conventional digital computers (Figure 14). This physical result can also be observed semantically in the Δ-digraph, and has implications for the complexity of unconventional computing [19].

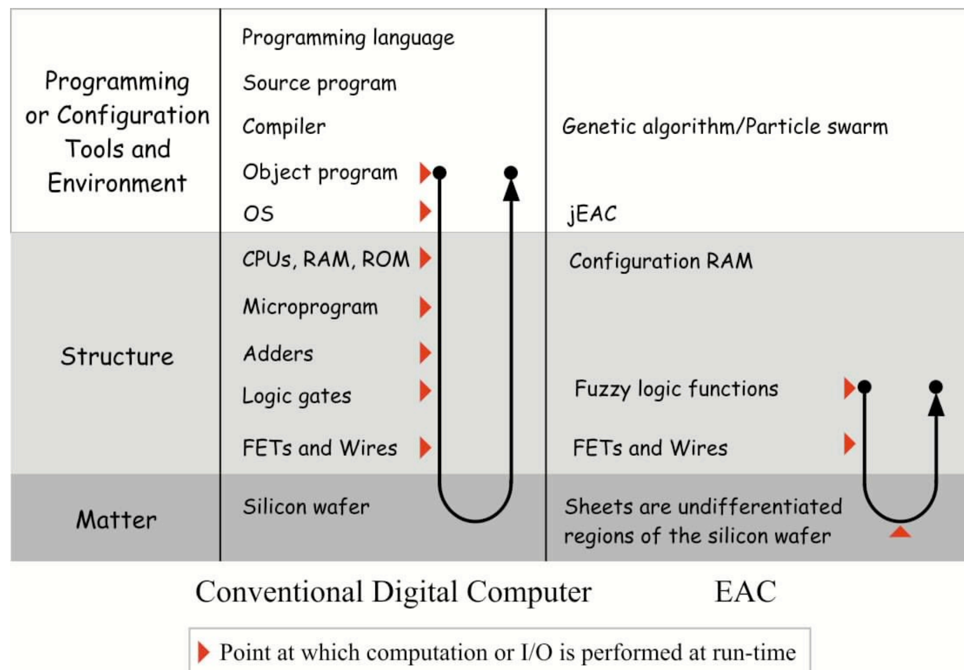


Figure 14. Computational bottlenecks

Not only does the EAC have fewer computational bottlenecks, the one it does have is wider (Figure 15). In applications without feedback, it resembles a fluid pipeline.

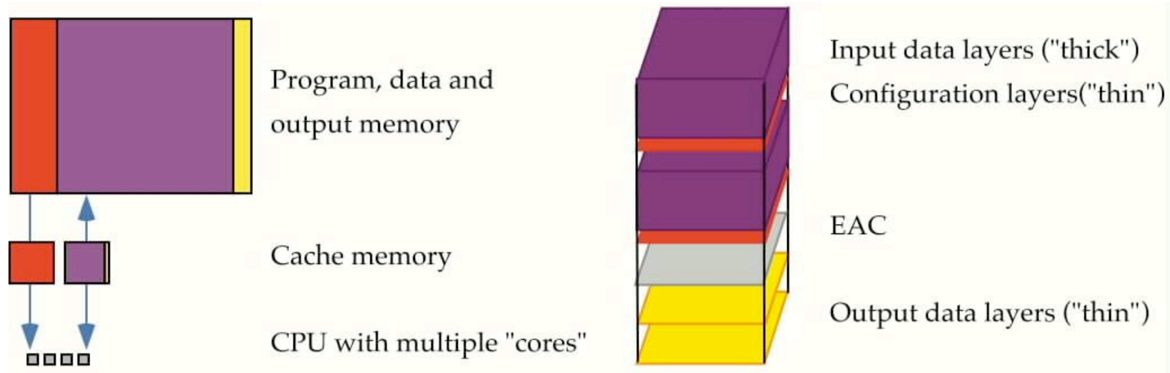


Figure 15. (a) von Neumann bottleneck (b) EAC bottleneck

The two-dimensional bottleneck of the conductive sheets is shown in these VLSI circuits, which were fabricated over a decade ago (Figure 16).

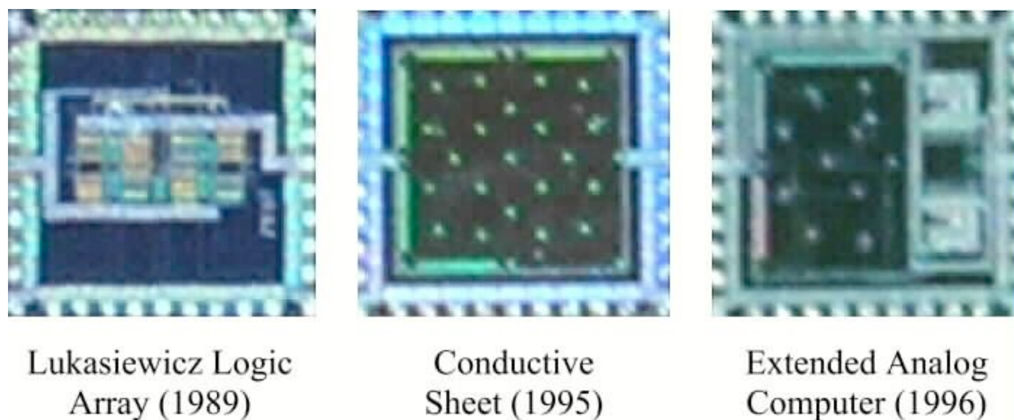


Figure 16. VLSI circuits that preceded EAC R002

Their simplicity makes them *fast* and *cheap*. Their large transistors and large areas of "empty space," that is, areas on the chip that have no transistors, only wires and ohmic connections, means that the die yield of an EAC is over 95%. If an error compensation map is used to correct outputs by interpolation, the yield is 100%. Of the twenty MOSIS prototypes fabricated, the yield was a flawless 100%. The LLA's H-tree structure can also tolerate a failed node, and experiments have demonstrated that the LLA H-tree increases precision and accuracy by "narrowing" the Gaussian error distribution [32].

Silicon and foam conductive sheet are not significantly affected by defects. Experiments showed that a foam sheet functioned without error after 78% of its area was removed, then worked with slightly less accuracy until its connections were cut [33] (Figure 17).

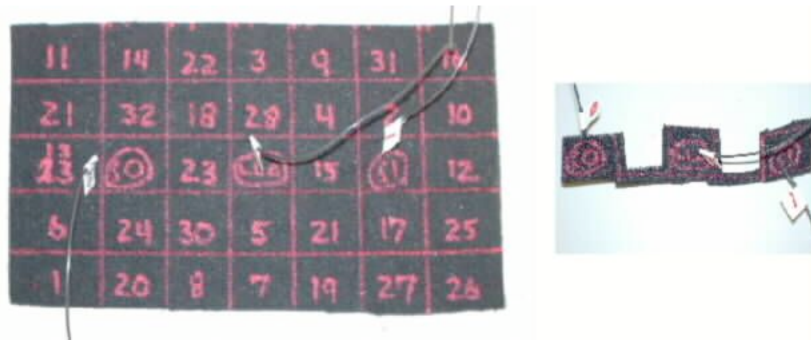


Figure 17. (a) Intact conductive sheet (b) Sheet after random damage

Radiation does not appreciably affect the continued functioning of a VLSI EAC (it will affect its output temporarily if it is not shielded, but a known level of radiation can be compensated for), nor will slight physical damage affect its operability. VLSI circuits fabricated in 1995 continue to work, even after being washed with distilled water to remove dust! Digital circuits are not this robust. While the EAC may not be self-healing, a property that one military agency seeks, it is a computer that is very difficult to "kill."

13. Techniques used to apply the EAC

Over time, general techniques useful in a wide variety of applications have been discovered, similar to classes of algorithms for digital computers. Four are listed in order of the scope of their usefulness: generate-and-recognize, analog alphabets, dynamically-evolving reconfiguration, and the solution of ordinary differential equations for feedback controllers. The first two, which are the most interesting techniques, are discussed here.

Generate-and-recognize originated in the original design of the EAC single-pixel retina shown in Figure 16 [28]. The original design operates by generating a manifold gradient when an image is projected onto it. The gradient is sampled and recognized by LLAs whose functions are selected in a form of training similar to backpropagation.

The retina was intended to be a biologically-inspired chip for robot control [4], [34], but soon found use as a general pattern recognizer. The breakthrough insight for robot control and many other applications was that inputs to the circuit did not have to be photovoltaic, but could come from any source of current. In the design for a proprioceptic (self-sensing) Stiquito robot (Figure 18a), EAC "retinas" were used to create the *umwelt,* the robot's sense of the world and its position in it. The cognitive schematic (Figure 18b) shows visual and "tactile" sensors in black. Internal EAC "retinas" are used for mapping the spatial location of objects and associating them with positive and negative responses to generate behavior (shown in red). Only one EAC acts as a visual recognizer; the rest handle data with other semantics, such as the robot looking at its legs as they move.

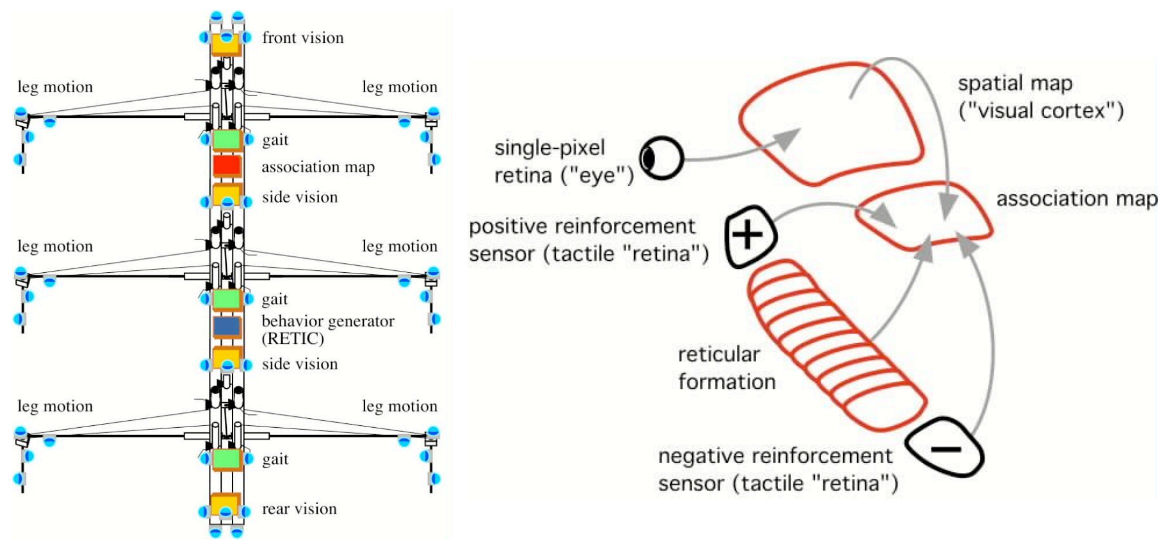


Figure 18. (a) Proprioceptic Stiquito design (b) Cognitive schematic of proprioceptic robot

An EAC retina can also “look at” digital values that are translated by a digital-to-analog converter. For example, one EAC can process thirty-two bytes from a packet of Internet data in parallel to generate its “image.” The speed at which the EAC operates and its two-dimensional input stream has led us to study its application to data mining, where multiple EACs categorize query results by visualizing them. This is an abstract **analogy**.

Abstract alphabets were derived from the generate-and-recognize technique, inspired by the butterfly wing morphogenesis model. Other alphabets, that is, series of distinct manifolds generated by the EAC’s conductive sheets, were used to distinguish valid from invalid local connections in Hamiltonian Cycle [30], protein folding, military-versus-civilian aircraft configurations, etc. In Figure 19 the evolution of an abstract alphabet for detecting distributed denial of service (DDoS) attacks is shown in subfigures (a) to (d).

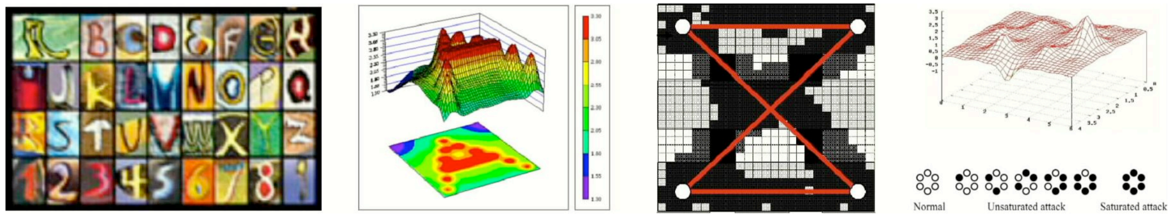


Figure 19. (a) Butterfly alphabet (b) Foam “A” (c) Hamiltonian Cycle (d) DDoS alphabet

Note that although the precision of a single EAC is limited to no more than sixteen bits, and is usually less, a large array of EACs has a “shallow” precision over the aggregate machine that can reach tens of thousands of bits. The number of individual EACs in the machine is directly related to the machine’s increased computational “word” size.

This allows temporal and spatial sequences of “characters” to scale up to larger problem instances. An EAC architecture composed of multiple elements, each an EAC similar to

the research prototype EAC R002, is applied to “inspect” problem instances for legal strings from the relevant analog alphabet. This is also an abstract form of **analogy**.

14. Concise EAC case studies

Five concise case studies illustrate the reasons that the EAC was of interest to an adopter, and the techniques that were used in each application. These are not blue-sky proposals, but actual applications for which the EAC was either considered, adopted, or is currently being evaluated. To address concerns for privacy and intellectual property rights, the specific details of the applications and the identity of the adopters are not given.

The applications for which the EAC was not chosen fell victim to natural “causes,” usually a decision not to risk a new technology, but sometimes due to economic factors outside the adopters’ control (downturns, lack of commercial availability of VLSI EACs, etc.). Still, it is a fact that the potential of the machine was attractive. As interest grows, the first commercial VLSI EAC, even a small embedded controller, is expected to create a sharp upturn in its use, as these case studies show. Its practicality is no longer in doubt.

Spall detector for ceramic impeller (1992). A military agency needed a replacement for a spall detector when they switched from ferrous to ceramic high-speed pump impellers. Old detectors relied on magnets to capture the chips that were broken from the ferrous impeller blades due to cavitation. These detectors no longer worked with non-magnetic ceramic spall. A publication about an LLA retina suggested that it was fast enough to distinguish between spall and bubbles in the lubricant flow, and would be robust enough to work in the hostile mechanical environment [4]. However, the retina used discrete Lukasiewicz implication cells (Figure 20a), and detected only edges (Figure 20b, a–h).

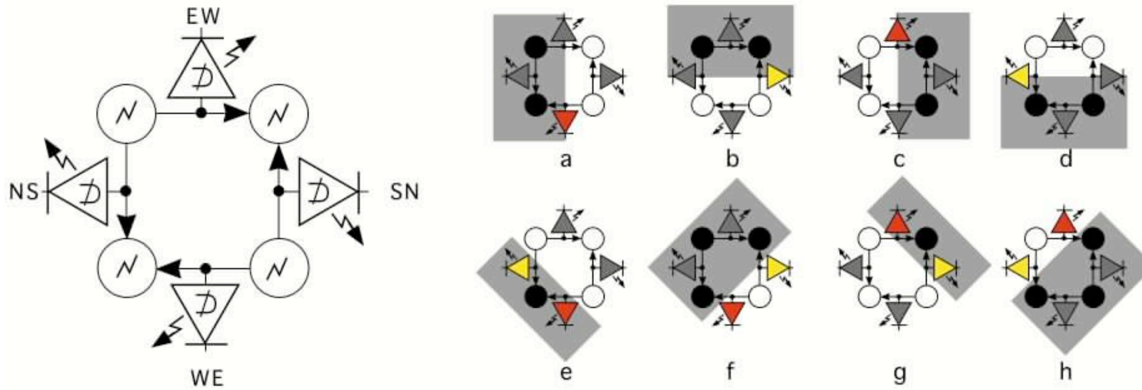


Figure 20. (a) LLA retina cell (b) Edges detected by the LLA retina cell

Using edges to distinguish between smooth bubbles and ragged spall required massive processing outside of the LLA chip. The task of image recognition for the hybrid processor was daunting. The EAC, generate-and-recognize and abstract alphabets that could have defined "characters" for bubbles and spall (or "not bubbles") had not yet been developed. The LLA retina was not suitable for this task. It was not adopted.

Embedded quality-of-service controller for Internet router and fuzzy categorizer (2000, 2006). A telecommunications firm in Canada evaluated the EAC for use as a quality-of-service (QoS) controller in an Internet router, and as an in-phone, low power fuzzy categorizer that would learn when and where users wished to accept calls. The factors of interest in the router were speed, and reconfigurability; the device had to evaluate QoS service constraints and network traffic levels dynamically in the nanosecond time domain. For the mobile call categorizer, the low power of the EAC and its ability to evolve rules as abstract alphabets were the most important factors. The EAC initially was not adopted due to the dot.com "bust," but interest has recently been revived. The company now has an EAC R002 board.

Programmable field logic sensor/controller (2006). The technology research office of a multinational semiconductor firm in the People's Republic of China, which employed at least one of the author's students, became interested in using the EAC as a programmable logic device for feedback control. Factors that led to the evaluation included low cost, robustness, "low tech" fabrication, and reconfigurability. However, the firm needed to perform adaptive control by computing the poles of functions, which meant that explicit solutions to ordinary differential equations (ODEs) were needed. The author and an engineer flown to Indiana University from the People's Republic of China jointly developed a new technique based on McNaughton's theorem that allowed an arbitrary algebraic ODE to be represented in a compact form using the EAC. The firm now has an EAC R002 board, and is evaluating designs based on it.

Evolved embedded systems controller (2006). An electronics design firm in the United States believed that the EAC could be used as a low-cost embedded controller configured using evolutionary algorithms. The firm's chief scientist began an eighteen-month series of visits to Indiana University An individual with long experience in computing, the chief scientist first saw the EAC as a kind of 1950's general purpose analog computer (GPAC), then an odd form of digital computer, and finally, in what he called "an epiphany," a machine with its own paradigm of computing. Shortly afterwards, the firm and Indiana university filed an application patent. The firm has purchased a number of EAC R002 boards, and is seeking a partnership with a major semiconductor manufacturer. Factors in the EAC's adoption include its low cost, dynamically evolvable reconfigurability, and ability to tolerate harsh industrial and military environments. If a low cost VLSI EAC is

fabricated commercially, it will enable construction of an analog supercomputer, just as today's digital video games have spawned a new generation of digital supercomputers.

Protein folding (2007). An Argentinean pharmaceutical consortium, whose chief executive officer explained that it is too expensive to obtain state-of-the-art supercomputers, was interested in the EAC because it is fast, cheap, scalable, and can be fabricated with "low tech" process technology (in fact, the group is now looking for older two or three micron photolithography equipment). An EAC supercomputer composed of an array of individual processors, each similar to, but simpler than, an EAC R002 board, would use both local and global generate-and-recognize for an abstract "protein folding" alphabet. The protein would be represented by a 2½D model, with local bond bending inspected for global interference between molecular groups. Although this problem is computationally complex for a digital computer, the EAC array operates using an "organic" analogy. The string computes in parallel, as does a real molecule. While it now takes the consortium's digital computers hundreds of days to design one protein, they estimate that an EAC array could perform the same computation in hours. The group has chosen to build its own EAC array using information available in articles and patents (the EAC is in the public domain; the LLAs were not patented internationally). The possibility of becoming a vendor of "Third World" supercomputers was a factor in their decision.

15. The future of the EAC

It has been necessary to go backward to go forward. The speed of the EAC R002 is a step backward from previous VLSI circuits that operate in a nanosecond or less due to their simplicity and the materials used to fabricate them [35]. Experience showed that

sacrificing speed for an open user architecture was a great leap forward. Its flexible digital interface enabled the development of software tools and interfaces.

However, EAC R002 is a machine built for flexibility, not speed. Users of this board control all facets of its configuration, including the ability to make arbitrary connections to the conductive sheet and to upload new Lukasiewicz logic functions at run time. However, there is a price for this flexibility, and that price is speed.

The embedded microprocessor that controls the interface and emulates the Lukasiewicz Logic Arrays is slow. Its response time is further reduced because it polls-and-controls each of the twenty-five elements in the conductive sheet's input/output array. The inexpensive ADCs and DACs are slow. Finally, the serial USB interface limits throughput to the host computer. The jEAC user interface operates at about four updates per minute. Faster speeds are possible by writing a program to interact with the EAC, but the input/output and emulation bottlenecks set a limit of about 1000 updates per minute. Still, Himebaugh's design succeeded beyond our expectations, as the case studies show, and as adopters in academia confirm.

Adopters in academia (as of 2007). The EAC is so inexpensive that a number of universities now own EAC R002 boards. At the time of writing, these include The University of the West of England (UK), the University of York (UK), The Polish Academy of Sciences (Warsaw), the École Polytechnique Fédérale de Lausanne (Switzerland) and Indiana University–Purdue University, Indianapolis (USA). Professor Andrew Adamatzky at the University of the West of England has developed an open EAC interface, and Dr. Simon Harding created an "evolutionary harness" that evolves

EAC configurations that are the solutions to symbolic equations (Figure 21) [36]. In this case, a function of the voltage V at the position (x,y) on the foam was evolved:

$$V(x,y) = ((x-3)^2 + 3(x-3)) \div (y+1)$$

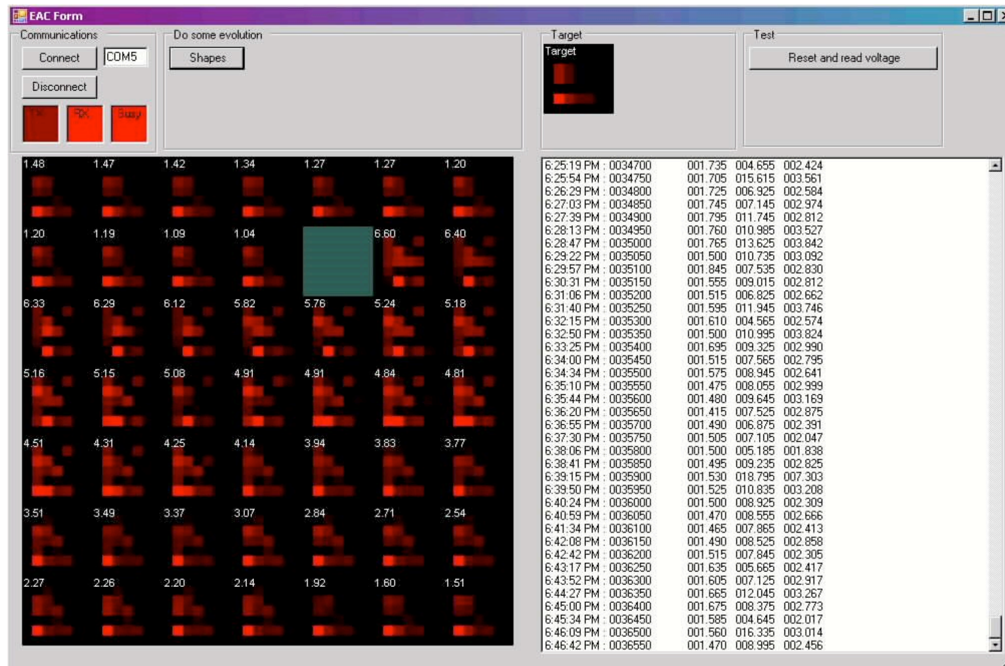


Figure 21. Harding's "evolutionary harness" evolving the solution to a symbolic equation on EAC R002

Dr. Harding is the first researcher to bridge the semantic gap between symbolic mathematical functions and the EAC architecture. That gap was illustrated at the start of this paper with the PDEs that describes butterfly wing morphogenesis (Figure 5).

16. In conclusion

The EAC R002 allows computer scientists in industry and academia to inexpensively explore the paradigm **analogy**, and create new interfaces, tools, and applications. The Δ-digraph simplifies understanding of the EAC's operation, and the semantics of its applications. With its dual symbolic and diagrammatic forms, the Δ-digraph also serves

as the basis for a new approach to the theory of the EAC and, possibly, other unconventional computers (as was shown in Figure 13). New directions include:

- a primitive spatial and temporal version of Lukasiewicz' logic to describe both the EAC's structure and its operation,
- semantic complexity classes defined visually (as an undergraduate, the author rewrote Smale's proof that Satisfiability is NP-complete in the form of semantically-connected clause matrices, rather like LEGO®[39]),
- new theories to rigorously model the visualization of analogies (for recent work related to this direction see Giaquinto [37] and Kulvivki [38]),
- a philosophical approach to analog computing, its logic and its visual meaning based on an extension of the works of Wittgenstein (the literature is extensive; useful introductions to his *Picture Theory* in the early and later writings can be found in Pears [41] and Kenny [42]).
- a version of Ashcroft and Wadge's *Lucid* dataflow programming language [40] for the EAC, and
- "natural pragmas" that add properties of materials to symbolic languages; these languages would resemble Lucid or HTML rather than C++ or Fortran, with the natural pragmas providing a way to translate "dusty deck" code to the EAC,

What is the next step that should be taken to expand society's use of the EAC?

Researchers and engineers in academia and industry may choose to study the lessons learned from EAC R002, and its present tools and interfaces. It is now time for a fast

VLSI EAC and its system interface. Every advance in the past ten years has made it easier to go “back to the future” and build a complete VLSI EAC-R002-on-a-chip. When that happens, the vision of a very unconventional analog VLSI supercomputer that solves trillions of partial differential equations per second [11] will become a reality.

Acknowledgements

This work was supported by a grant from the Leverhulme Trust. The University of the West of England and Indiana University–Bloomington are deeply thanked for all of their support, including the author’s sabbatical awarded by Indiana University.

Many people have contributed to this research since 1989. Special thanks go to Bryce Himebaugh for the EAC R002, Andrew Adamatzky for his friendship and the Leverhulme Trust grant nomination, Christof Teuscher for the invitation to UC’07 in Santa Fe and his continued patience and encouragement, and then, in alphabetical order, Larry Bull, John Greenman, Simon Harding, Ioannis Ieropoulos, Genaro Martinez, Sally McKee, Julian Miller, Amr Sabry and Susan Stepney. All of the mistakes are mine.

Thanks are due to all those in industry who visited Bloomington from other parts of the USA and from overseas to learn about the EAC, the numerous anonymous referees whose comments over the past decade have helped the author understand his own research, and the hundreds of students in the author’s B644 VLSI Design course (which has evolved into a course about the EAC). Finally, the opportunity to interact with my students would not have been possible without the support of Dennis Gannon, who, as chair of the Computer Science Department, put B644 “back on the books” in 2000.

A very special acknowledgement is given to the late Lee A. and Nina Rubel, and Dick and Sasha Rubel. Thank you, Lee, for your tough but unfailingly sincere encouragement, and Nina, Dick and Sasha for your warm friendship. I am grateful for every minute of it.

References

[1] J. Mills, Kirchhoff-Lukasiewicz Machines, Internal Research Note, Indiana University (1995). (This note is available from the corresponding author.)

[2] J. W. Mills, M. G. Beavers, and C. A. Daffinger, Lukasiewicz Logic Arrays, *Proc. 20th Int. Symp. Multiple-Valued Logic* (1990).

[3] J. Mills and C. Daffinger, CMOS VLSI Lukasiewicz Logic Arrays, *Proc. Application Specific Array Processors*, Princeton, New Jersey (1990).

[4] J. W. Mills, Area-Efficient Implication Circuits for Very Dense Lukasiewicz Logic Arrays, *Proc. 22nd Int. Symp. Multiple-Valued Logic*. Sendai, Japan: IEEE Press (1992).

[5] B. Himebaugh, Design of EAC R002, www.cs.indiana.edu/~bhimebau (2005).

[6] R. Varick, Building tools for the analog researcher: an interaction design challenge, Senior Research Project Report, Indiana University (2006). (This paper is available from the corresponding author.)

[7] R. Feynman, *Six Not-So-Easy Pieces*, Perseus Books: Cambridge, MA (1963).

[8] R. Feynman, *The Character of Physical Law*, MIT Press: Cambridge, MA (1967).

[9] J. Mills, Polymer Processors, *2nd Int. Symp. Non-Silicon Computation* (2000).

[10] A. Adamatzky, From reaction-diffusion to Physarum computing, *Physica D* (2007), submitted.

[11] J. Mills, B. Himebaugh, et. al., “Empty Space” Computes: The Evolution of an Unconventional Supercomputer, *Proc. ACM Computing Frontiers Conf.* (2006).

[12] G. Kirchhoff, Flow of Electric Current through a Plane, Particularly through a Circular Disk, *Ann. Phys. Chemie,* 64, p. 499 (1845). This citation is a translation from the original German by student Stephan Oberreichholz (2004).

[13] W. Karplus, *Analog Simulation*, McGraw-Hill: NY, (1958).

[14] P. J. Denning, Great Principles of Computing, *Comm. ACM*, **46** (11), pp. 15-20 (2003).

[15] P. J. Denning with C. Martell, Great Principles of Computing, Preliminary Summary (2007).

[16] J. R. Searle, Is the Brain a Digital Computer?, Proc. and Addr. of the Am. Phil. Assoc. **64** (3), pp. 21-37 (1990).

[17] G. Piccinini, Computers, read at Canadian Soc. for the History and Philosophy of Science, Toronto, Canada (May 2002; revised 2006).

[18] O. Shagrir, Why We View the Brain as a Computer, *Synthese* **153** (3), pp. 393-416 (2006).

[19] J. F. Costa and J. Mycka, What Lies Beyond the Mountains? Computational systems beyond the Turing limit, Technical report, Universidade Tecnica de Lisboa, Lisboa, Portugal (2004). (available from: fgc@math.ist.utl.pt)

[20] J. Crowcroft, On the Nature of Computing, *Comm. ACM* **48** (2), pp. 19-20 (2005).

[21] F. Nijhout, *Development and Evolution of Butterfly Wing Patterns*, Smithsonian Inst. Press: Wash., D.C., (1991).

[22] A. Turing, Morphogenesis, *(in The Collected Works of A. M. Turing, P. T. Saunders, ed.),* North-Holland: Amsterdam (1992).

[23] L. A. Rubel, The Extended Analog Computer, *Advances in Applied Mathematics*, **14** pp. 39-50, (1993).

[24] H. Siegelmann and E. Sontag, On the Computational Power of Neural Nets, *Proc. Fifth ACM Workshop on Computational Learning Theory*, Pittsburgh, (1992).

[25] R. Landauer, Wheeler's Meaning Circuit?, *Foundations of Physics* **16** (6) pp. 551-564 (1986).

[26] R. McNaughton, A theorem about infinite-valued sentential logic, *J. Symbolic Logic* **16** pp. 1-13 (1950).

[27] J. Mills, Solving Ordinary Differential Equations with the EAC, Internal Research Note, Indiana University (2006).

[28] J. Mills, The Continuous Retina: Image Processing with a Single-Sensor Artificial Neural Field Network. *Proc. IEEE Conf. Neural Networks* (1996).

[29] K. Sandved, The Butterfly Alphabet Poster (2000).

[30] L.-S. Wu and G.-T. Kim, Using the butterfly wing pattern to solve Hamiltonian Circuit, B644 VLSI Final Project Report, Indiana University (2004).

[31] A. Adamatzky, Growing spanning trees in Plasmodium Machines, *Kybernetes* (2007), in press.

[32] R. A. Montante and J. W. Mills, Measuring Information Capacity in a VLSI Analog Logic Circuit, Indiana University Computer Science Department, TR 377 (1993).

[33] B. Czoch and A. Ivancic, Error Tolerance of the Extended Analog Computer: Dissecting It on the Fly, Senior Independent Study Report (2005).

[34] J. Mills, T. Walker and B. Himebaugh, Lukasiewicz' Insect: Continuous-Valued Robotic Control after Ten Years, *Int. Jour. Multiple-Valued Logic* (2003).

[35] A. Biswas, Some Investigations with Laser Beams on an LLA Retina, MS Thesis, Indiana University Computer Science Department (1994).

[36] S. Harding, Programming the Extended Analogue Computer using Evolutionary Algorithms (2007), in preparation.

[37] M. Giaquinto, *Visual Thinking in Mathematics: An Epistemological Study,* Oxford University Press: Oxford (2007).

[38] J. V. Kulviki, *On Images: Their structure and content,* Clarendon: Oxford (2006).

[39] J. Mills, A Visual Proof that Satisfiability is NP-complete, (1985), annotations to M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman: San Francisco (1979).

[40] W. Wadge, and E. Ashcroft, *Lucid, The Dataflow Programming Language*, London: Academic Press (1985).

[41] D. Pears, *Paradox and Platitude in Wittgenstein's Philosophy*, Clarendon Press: Oxford (2006)

[42] A. Kenny, *The Wittgenstein Reader* (2nd Ed.), Blackwell: Oxford (2006).

Figure captions

Figure 1. (a) EAC to left of Macintosh (b) jEAC Interface in operation

Figure 2. Upper side of EAC R002 during operation

Figure 3. (a) Upper side of EAC R002 (b) Underside of EAC R002

Figure 4. (a) Diagram of upper side of EAC R002 (b) Diagram of underside of EAC R002

Figure 5. Developing a specific analogy for the EAC

Figure 6. One level of a general Δ-digraph

Figure 7. Diagrammatic representation of the computing paradigm analogy

Figure 8. Diagrammatic representation of the computing paradigm algorithm

Figure 9. Explicit addition

Figure 10. Implicit addition

Figure 11. Explicit analog multiplication

Figure 12.Semantic hierarchy of the EAC analogy for Hamiltonian Circuit

Figure 13. Semantic hierarchy of the Physarum sp. analogy for Minimum Spanning Tree

Figure 14. Computational bottlenecks

Figure 15. (a) von Neumann bottleneck (b) EAC bottleneck

Figure 16. VLSI circuits that preceded EAC R002

Figure 17. (a) Intact conductive sheet (b) Sheet after random damage

Figure 18. (a) Proprioceptic Stiquito design (b) Cognitive schematic of proprioceptic robot

Figure 19. (a) Butterfly alphabet (b) Foam “A” (c) Hamiltonian Cycle (d) DDoS alphabet

Figure 20. (a) LLA retina cell (b) Edges detected by the LLA retina cell

Figure 21. Harding’s “evolutionary harness” evolving the solution to a symbolic equation on EAC R002