Everything Computes

Peter J Bentley

Abstract

The emergence of complexity in our universe is caused by generative processes that confound us. Collective behaviours going under names such as evolution, swarming, embryogenesis, thought, and emergence produce results that seem counterintuitive (or downright impossible) when the individual elements in the collective are examined.

In this seminar I explore the causes of complex behaviour in general and suggest a new viewpoint based on a combination of systemics and computation. I suggest that all behaviour in the universe can be viewed as a form of computation – a systemic computation – which is in many respects faster and more efficient than the classical views of computation we use in traditional computer designs. By viewing behaviour, and specifically biological systems, as types of highly parallel, nested computations, we can begin to understand how complex forms and functions emerge. With this knowledge we might one day be able to exploit the computation of natural systems and use it for our own purposes.

Introduction

The most bizarre thing about the universe around us is that it exists. The second most bizarre thing is why it behaves in the way it does. We're not much closer to providing a satisfactory answer to the first question, despite some imaginative theories. But we're finding much better answers to the second question.

The behaviour of entities in our universe seems relatively straightforward at a low level. Quarks, atoms, and molecules all interact according to (more or less) known properties, resulting in the familiar forms of matter and energy we observe every day. But when we look at more complex groups of molecules, it all starts to feel confusing. A virus is an astonishing molecule. A cell is a monstrously complicated group of molecules. An organism is a terrifyingly complex group of cells. A species is a remarkable group of organisms. An ecology is an surprising group of species. And so it goes on. The superlatives are easily used because our brains are not clever enough to grasp how a generative process can create such complexities on their own. How can a design exist without a designer? How can something physical exist without a builder? How can damage be repaired without a repairer? How can adaptation to change occur without an adaptor?

Computation is changing our conceptions of what is and is not possible. Today we can demonstrate that all of these seemingly miraculous effects can occur inside our computers. We can show how collections of smaller components interact and generate extraordinary emergent patterns and behaviour. There is no need for a designer or ruler of these systems – they will generate their own behaviour by themselves. The main thing we have to do is make our computer act like a womb in which something akin to biology can grow.

At this point we run into problems. A conventional computer may theoretically be able to compute everything that occurs in a biological system, but even our fastest computers struggle to calculate the interactions of two complex molecules. Somehow biology is much better at performing computations than our computers are. In fact, everything in the universe is much better at computing. Everything computes. But it doesn't compute in the same way that our computers do.

If we can understand what kind of computer we need in order to produce similar natural complexity to biological systems, we might just understand a little more about the kind of computation going on in the universe around us. Maybe we might learn something about why things behave in the way they do.

Computers are not good at computing

You might say that humankind has become a little obsessed with computers recently. The internet and imminent rise of ubiquitous computing will soon ensure a computer is incorporated into most devices around us (Gershenfeld, 2000). Today we have computers in unlikely places such as lightswitches, toothbrushes, radios, car engines, pens and pets. In the following decades these will only become more numerous and more integrated as wireless communication enables every device to communicate with its neighbours in dynamic local networks (Weiser,1993).

But the obsession is an unhealthy one. Our computers are failing us. As computation becomes more dynamic and unpredictable, with viruses, incompatibilities, aging software and side-effects from legacy code, so reliability is being reduced. (During the writing of this document the word processor crashed seven times, and that is normal by today's standards.) But worse than this, our computers are not capable of scaling to achieve the level of computation that we desire. As computational analysis, modelling and data mining for the biosciences becomes increasingly important (Fogel and Corne 2003), we are finding that even our supercomputers struggle to model or analyse behaviour – Moore's Law cannot keep up with the data produced by advances in DNA synthesis and sequencing productivity (Carlson, 2003). As our ambitions to exploit biological processes in "intelligent" algorithms increase, we discover that the processing power required to evolve and develop solutions, or create immunity in a network, is prohibitive using

current technology. (The problem is so well-known and widely accepted that a recent European Union Future and Emerging Technologies programme in October 2005 generated 38 €1.5-3M proposals, of which over half proposed methods for improving our ability to model biology[1].)

This failure of modern computers to model or mimic biological processes is hardly surprising when one examines the origins of the computer. Conventional computation was built using principles of engineering. The early designs of computer by pioneers such as von Neuman comprised four logical elements: the Central Arithmetical unit (CA), the Central Control unit (CU), the Memory (M), and Input/Output devices (IO) (von Neuman, 1945). Although intended as a high-level abstraction, its separation of functionality was based on the technology of the day, and constructed such that each element could be performed by a human (in much the same way that a factory production-line is decomposed into separate jobs, each performed either by machine or by hand) (Aspray, 1990). The design speaks volumes about the traditional values underpinning the society that led to its creation: the *central* control unit, the *central* arithmetic unit – these centralised methods of control have little to do with effective computation and may instead point to the traditional ideas pervasive in government, aristocracy and family values (Jost et al 2003). You needed a General to lead an army, a ruler to run a country and a central control unit to run a computer. Yet despite the 1940s design, the separation of functionality persists to the present day – an arbitrary design constraint inherited by every computer in the world. It has affected all design decisions since – "parallel software" is nothing more than a simulation of parallelism, operating systems must waste processor time with system calls and resource allocation, processor speedups must rely on instruction predictions or fast memory caches.

Conventional	**Natural**
Deterministic	Stochastic
Synchronous	Asynchronous
Serial	Parallel
Heterostatic	Homoestatic
Batch	Continuous
Brittle	Robust
Fault intolerent	Fault tolerant
Human-reliant	Autonomous
Limited	Open-ended
Centralised	Distributed
Precise	Approximate
Isolated	Embodied
Linear causality	Circular causality

Table 1 Features of conventional vs natural computation

Without biology, we might be content to stay with these old designs, but the study of living systems has taught us that biology seems to perform far superior computation (or at least something that is equivalent to computation) and that it does it using radically different methods to ours (Stepney et al, 2005a,b).

It is remarkable that almost every parameter that can be used to measure the operation of a computational system shows natural systems at one end of the spectrum and conventional computing at the other, see table 1. The lack of similarity between natural and conventional computing systems seems so extreme that it is astonishing that the former can be modelled by the latter at all. Of course, if a system of computation is Turing Complete then it can mimic any other computer, given sufficient time and resources (Brainherd et al, 1974). But just as it is ludicrous (but possible) to model every operation of a modern supercomputer with a real ant colony, it is ludicrous (and maybe impossible, because of the embodied nature of the colony) to for a modern supercomputer to model every operation of an ant colony. The two systems of computation may be mathematically equivalent at a certain level of abstraction, but they are *practically* so dissimilar that they become virtually incompatible.

Not all computers are made from silicon

The differences between natural and traditional computation are becoming recognised and better understood. Von Neuman did not design his EVDAC computer with today's technology in mind. His design was for a machine the size of a room, simpler than most digital watches today (Aspray 1990). Today's computer-saturated world has begun a new era. Ubiquitous computing looms – computers may soon even be sprayed onto surfaces like paint (Arvind, 2004). Wireless communication between microscopic computers will be dynamic, unpredictable, parallel, stochastic and asynchronous. The future of computation seems inescapable – it will increasingly resemble the "wet computation" within our skulls.

In response, researchers are now beginning to follow the path of von Neuman and consider alternative views of computation. Cellular automata have proven themselves to be a valuable approach to emergent, distributed computation (Wolfram, 2002; Polack et al, 2005). Generalisations such as constrained generating procedures and collision-based computing provide new ways to design and analyse emergent phenomena (Holland, 1998; Adamatzky 2002). Indeed it is clear that von Neuman himself believed that alternative forms of computation were likely to be valuable as can be seen by his final works on self-organising and reliable automata and "organisms" (von Neuman, 1956, 1966; Asprey 1990). Bio-inspired grammars (Lindenmayer,1968) and algorithms introduced notions of homeostasis (Varela et al, 1974), for example in artificial immune systems (Stepney et al, 2005b), fault-tolerance as seen in embryonic hardware (Tyrrell et al, 2003) and parallel stochastic learning, for example in swarm intelligence and genetic algorithms (Fogel & Corne 2003). Alternative architectures such as analogue or pulse computers show potential for so-called "Super-Turing

[1] One of the top ranked proposals to be funded was PERPLEXUS (the successor to POETIC), which will create new modular, wireless hardware to speed up modelling of biology; information used with permission from EU.

Computation" (Siegelmann, 1996) (e.g., they could solve problems in a lower time complexity than Turing machines, or work with irrational numbers with the same efficiency that a finite Turing machine works with rational numbers).

These new methods of computation are showing extraordinary generative powers, and they are providing more clues about the nature of computation in the universe. For example, traditional (silicon) computation is separated from our physical world. Why should a computer work so differently from the brain of a living creature? Why should computation be performed in terms of symbolic manipulation when an ant colony knows no symbols? Why should calculations be performed serially and in time to the tick of a clock signal when there is almost no example in nature or physics of anything working according to similar principles? (Adamatzky 2001)

We are finally beginning to realise that traditional computation is simply a tiny subset of the possible ways in which computation can be performed. It is a heavily constrained, slow and inefficient way of computing (Stepney et al 2005a,b). In reality, computation is performed all around us, all the time. Our brains compute, our DNA computes, cell growth computes, protein interactions compute, evolution of organisms compute. Everything computes.

To understand how, we just need to look at the universe in a slightly different way.

Systemic Computation

"Systemics" is a world-view where traditional reductionist approaches are supplemented by holistic, system-level analysis. Instead of relying on a notion of compartmentalising a natural system into components, analysing each in isolation, and then attempting to fit the pieces into a larger jigsaw, a systemic approach would recognise that each component may be intricately entwined with the other components and would attempt to analyse the interplay between components and their environment at many different levels of abstractions (Eriksson, 1997). For example, a reductionist analysis of the immune system might reach very specific conclusions about the effect of certain stimuli on certain cells (e.g., freezing of cells in culture, which causes necrosis). A systemic approach might recognise that the real immune system would never encounter such stimuli in isolation (cold temperatures cause many important physiological effects in the body which would change the environment of the cells), and thus the results may be inaccurate.

"Systemic computation" is a world-view proposed by this author that interprets behaviour in terms of interacting systems, focussing specifically on the attributes of natural computation listed in table 1. Systemic computation suggests that everything is composed of *systems*, which transform each other through their interactions (figure 1). Two systems interact in the context of a third system, which defines the result of their interaction. This is intended to mirror all conceivable natural processes, for example:

- molecular interactions (two molecules interact according to their shape, within a specific molecular and physical environment)
- cellular interactions (intercellular communication and physical forces imposed between two cells occurs in the context of a specific cellular environment)
- individual interactions (evolution relies on two individuals interacting at the right time and context, both to create offspring and to cause selection pressure through death)

In this view, each system can potentially affect and be affected by another system. When two systems interact, they will be transformed in some manner, see figure 2. This incorporates the idea of circular causality (e.g., A may affect B and simultaneously B may affect A) instead of the linear causality inherent in traditional views (Wheeler and Clark 1999). Such ideas are vital in biological systems – for example, consider two plants growing next to each other, the leaves of each affecting the growth of the leaves of the other at the same time.

Exactly how the two interacting systems are transformed will be determined by the nature of the two systems and the nature of the contextual system in which the interaction takes place. For example, when water interacts with the interior of a glass in the context of gravity, gravity dictates that the water will form into a glass-shaped pool. But when water interacts with the interior of a glass in the context of zero gravity, the water may form into a collection of spherical globules. So we can consider the third contextual system (gravity) as defining the transformations of two interacting systems (water and glass).

But in this view, everything is made from systems, so every system can potentially define the transformations of two other systems in its context, while also being transformed or transforming other systems it interacts with. For example, water can act as a context for two other systems, defining how they interact. (An example closer to biology might be a hormone, which can act as a context and define how two cells interact with each other, while also being transformed by other proteins into a different form.)

A refinement to the idea of systemic computation is the addition of scope. Not every system can interact with every other. There will always be some notion of a limit, whether caused by distance, strength, form or size, which prevents some systems from affecting others (for example, binding forces of atoms, chemical gradients of proteins, physical distance between physically interacting individuals, temporal distance between events). So it is natural to consider systems existing within the same scope, and systems that exist in other scopes. Because everything is made from systems, the scope is defined by another system. Systems may be pushed into or out of a scope as a result of some transformation caused by an interaction, see figures 3 and 4. For example, the cell has a very clearly defined scope (the system which creates the scope is the cell wall), with proteins outside not able to interact with proteins inside. However, a protein interacting with another within the cell wall may result in a transformation which enables it to enter the cell. Because scopes are defined by systems, this allows us to have no limit to the number of systems contained within

systems. So the levels of structure we observe in nature (DNA, organelles, cell, organ, organism, population, species, ecology) can all be considered as nested computational systems, one inside the other.

Given the right set of systems, the resulting transformations will result in complex, emergent behaviour, see figures 5 and 6. But whether the behaviour is complex or simple, if systems transform each other, then computation is occurring. We know this is true, because we can construct a systemic computer and make it perform computations that we desire.

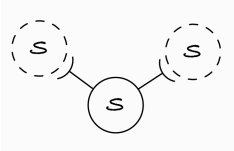

Figure 1 Everything (and every group of things) in the universe is a system. Two systems are modified through interactions with each other, in the context of a third system.

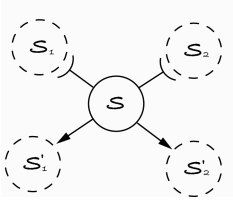

Figure 2 Each system provides a context in which other systems interact. When they interact, they are transformed according to their nature and the nature of their context.

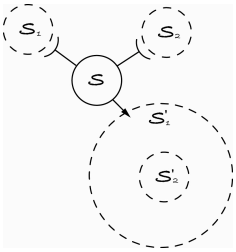

Figure 3 Systems can only interact with each other if they are in the same scope, defined by another system. A system may be pushed inside the scope of a second system through interaction with it.

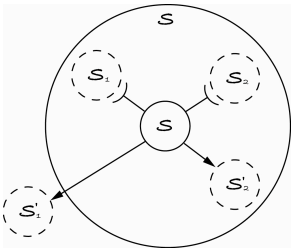

Figure 4 A system within the scope of a larger system may be pushed outside that scope through interaction with another system.

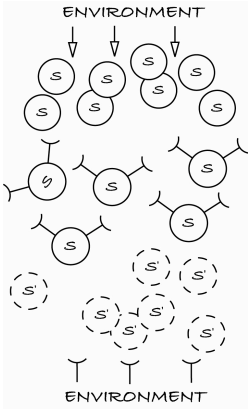

Figure 5 Everything is made from systems, so a group of systems will affect and be affected by its environment by transforming and being transformed by some of the systems that constitute the environment.

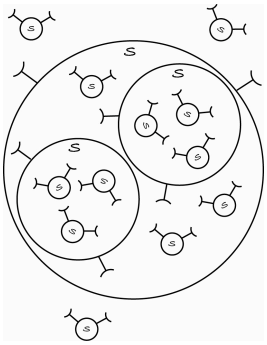

Figure 6 Most systems seem to naturally form themselves into hierarchical, nested structures that provide modularity, selectivity and protection against the environment.

The Systemic Computer

Building a systemic computer is surprisingly simple. Instead of the traditional centralised view of computation, here all computation would be distributed. There would be no separation of data and code, or functionality into memory, ALU, and I/O. While it is possible for more than two systems to interact at once, our computer can focus on two at any given point in time. So if A, B and C all interact in the context of D (written as D[A,B,C]) then this can be broken down into binary interactions (e.g. D[A,B] -> E, D[E,C]).

In the systemic computer, each system comprises schemata and functionality. The two schemata of a system define which other systems may be affected by this system. The functionality of a system defines how two systems are changed when they interact with each other in the context of this system. It is likely that such behaviour would enable more realistic modelling of natural processes, where all behaviour emerges through the interaction and transformation of components in a given context.

The idea of scope is a familiar one to computation. Scope in our systemic computer can be made infinitely recursive so systems may contain systems containing systems and so on. Scope also makes this form of computation tractable by reducing the number of interactions possible between systems to those in the same scope.

Figure 5 can be thought of as an idealised view of a computation using this approach. The environment (or problem, or user) would affect the overall system by being converted into a series of compatible systems. These are transformed by the program (a previously defined pool of systems equivalent to software or hardware) which then affect the environment through transducers. The separation into layers is shown purely for clarity – in reality most systemic computation will be a complex interwoven structure made from systems that affect and are affected by their environment. This might resemble a molecule, a cell, an organism or a population, depending on the level of abstraction used in the model. Figure 6 shows a more realistic organisation of a systemic computation, showing the use of structure enabled by scopes. The structure would enable the equivalent of modules, subroutines, recursion and looping. Most or all systems might be active and capable of enabling interactions. Note the similarity between a typical systemic computation with the structure of biological systems such as the cell.

S1	T	S2
##111101######01	1111101011101101	11#######110110####
Schema for System1	*Transformation function set*	*Schema for System2*

Figure 7 Each system can be represented by a (binary) string comprising three elements: S1 and S2 (the schemata that define which other Systems may match) and T (the transformations that the two matching Systems undergo).

We can design a systemic computer in even more detail. For example, systems can be implemented using representations similar to those used in genetic algorithms and cellular automata. In the example shown in figure 7, a system comprises three binary strings: two schemata that define sub-patterns of the two matching systems and one coded list of transformation functions. Two systems that match the schemata have their own binary strings transformed according to the transformation functions.

A simple example of a (partially interpreted) system string might be:

"0####### $S_1 1 = SUM(S_1 1, S_2 1)$; $S_1 2 = SUM(S_1 2, S_2 2)$; $S_1 2 = 0$; $S_2 2 = 0$; 0#######"

meaning: for every two systems with most significant bit of "0" that interact in the context of this system, add their two S1 values, storing the result in S1 of the first system and add the two S2 values, storing the result in S2 of the first system, then set S1 and S2 of the second system to zero. Given a pool of inert data systems, for example (where NOP means "no operation"):

"00010111 NOP 01101011" and "00001111 NOP 00010111"

after a sufficient period of interaction, the result will be a single system with its S1 and S2 values equal to the sum of all S1 and S2 values of all data systems, with all other data systems having S1 and S2 values of zero.

Systems within the same scope could be presented to each other randomly just as most interactions in the natural world have a stochastic element. The computation would proceed asynchronously and in parallel, distributed amongst all the separate systems, structurally coupled to its environment, with parallelism and embodiment providing the same kind of speedup seen in biological systems. Computation would be continuous (and open-ended) with homeostasis of different systems maintaining the "program". It seems likely that robustness, fault-tolerance and autonomy would emerge naturally in systemic computers. In short, this computer would encapsulate all of the features of natural computation listed in table 1, while maintaining a low-level simplicity that will enable analysis of computational behaviour.

Those familiar with cellular automata (and the views of Varela (McMullin & Varela, 1997)) may find the ideas described here familiar. Not only would a systemic computer incorporate similar notions of distributed emergent computation, its application would be similar. Just as cellular automata may be considered an algorithm and implemented on a traditional computer, or viewed as a computer architecture and run as dedicated parallel hardware, so a systemic computer could be simulated in an algorithm or hardware. Researchers using cellular automata have demonstrated notable but limited success in modelling natural processes and implementation of bio-inspired techniques (limited perhaps because of its perceived requirements for regular spatially arranged components and constrained interactions). A systemic computer would have equal – and probably greater – practical capabilities and hence a broader appeal. It can be shown quite trivially that systemic computation is Turing Complete and that a small number of interacting systems behave in an equivalent manner to a Turing machine. Whether a systemic computer would enable so-called Super-Turing computation is still unknown.

Discussion

Computer scientists have their own view of the world and it is perhaps not surprising that the theory of emergent behaviour presented here is phrased in terms of computation. Like every other theory in science, systemic computation is nothing more than an approximation of the truth. However, here I suggest that viewing behaviour as interactions between systems, in the context and scope of other systems, is considerably more useful than attempting to fit traditional ideas of computation or traditional reductionist approaches that insist on linear causality.

In the classic question, “which came first, the chicken or the egg?” linear causality stumbles over itself and causes confusion. Systemic computation would answer, “they computed each other – so neither came first”.

Everything computes. Everything is made from systems that interact with each other and transform each other. Should a systemic computer ever be built[2], then we may begin to understand the nature of this computation caused by the transformations. Perhaps we will finally arrive at a consensus for the meaning of a *complex* system, or a *generative* system (for example, a generative system might be defined as a system which is capable of non-randomly transforming an infinite number of other systems).

Perhaps we will understand how to control the unceasing computation that surrounds us and make it work for us. If a glass of water has more molecules in it than all the grains of sand on all the beaches in the world, just how powerful a computer must it be?

References

Andy Adamatzky (2001) *Computing in Nonlinear Media and Automata Collectives*. IoP Publishing, Bristol, 410 pp. ISBN 075030751X.

Andy Adamatzky (Ed) (2002) *Collision-Based Computing*. Springer-Verlag, XXVII, 556 pp. Softcover; ISBN 1-85233-540-8.

William Aspray (1990) *John Von Neumann and the Origins of Modern Computing*. Cambridge, Massachusetts: The MIT Press.

D K Arvind, K J Wong (2004) Speckled Computing: Disruptive Technology for Networked Information Appliances. In Proceedings of the IEEE International Symposium on Consumer Electronics (ISCE'04) (UK), pp 219-223, September 2004.

Brainerd, W.S., Landweber, L.H. (1974), *Theory of Computation*, Wiley.

Rob Carlson (2003) The Pace and Proliferation of Biological Technologies. Biosecurity and Bioterrorism: Biodefense Strategy, Practice, and Science Volume 1 Number 3, August 2003.

Darek Eriksson (1997) A Principal Exposition of Jean-Louis Le Moigne’s Systemic Theory. Review Cybernetics and Human Knowing. V4:2-3.

Gary B. Fogel, David W. Corne (Eds) (2003) Evolutionary Computation in Bioinformatics. Morgan Kaufmann Pub. ISBN: 1558607978

Neil Gershenfeld (2000) *When Things Start to Think*. Henry Holt and Company, New York.

John H. Holland (1998) *Emergence. From Chaos to Order*. Oxford University Press, UK.

Jost, J.T., Glaser, Kruglanski & Sulloway, F.J. (2003). Political conservatism as motivated social cognition. Psychological Bulletin, 129(3), 339-375.

A. Lindenmayer. (1968) Mathematical models for cellular interaction in development, parts I and II. Journal of Theoretical Biology, 18:280-315.

Barry McMullin and Francisco J. Varela (1997) Rediscovering Computational Autopoiesis. In Proc of European Conference on Artificial Life (ECAL 1997), Brighton, UK July 1997.

Fiona Polack, Susan Stepney, Heather Turner, Peter Welch, Fred Barnes (2005). An architecture for modelling emergence in CA-like systems. ECAL'05, Kent, UK, September 2005. LNCS. Springer.

Hava T. Siegelmann (1996) The simple dynamics of super Turing theories. Theoretical Computer Science. Vol168:2, 20 Nov. 1996, 461-472.

S. Stepney, S. Braunstein, J. Clark, A. Tyrrell, A. Adamatzky, R. Smith, T. Addis, C. Johnson, J. Timmis, P. Welch, R. Milner and D. Partridge (2005a) Journeys in Non-Classical Computation I: A Grand Challange. Int Jnl of Parallel, Emergent and Distributed Systems 20(1):5-19.

S. Stepney, S. Braunstein, J. Clark, A. Tyrrell, A. Adamatzky, R. Smith, T. Addis, C. Johnson, J. Timmis, P. Welch, R. Milner and D. Partridge (2005b) Journeys in Non-Classical Computation II: Initial Journeys and Waypoints Int. Journal of Parallel, Emergent and Distributed Systems.

Andy M. Tyrrell, Eduardo Sanchez, Dario Floreano, Gianluca Tempesti, Daniel Mange, Juan-Manuel Moreno, Jay Rosenberg, and Alessandro E.P. Villa (2003) POEtic Tissue: An Integrated Architecture for Bio-inspired Hardware. A.M. Tyrrell, P.C. Haddow, and J. Torresen (Eds.): Proc of the International Conference on Evolvable Systems (ICES 2003), LNCS 2606, pp. 129–140, 2003. Springer-Verlag Berlin Heidelberg.

Francisco J. Varela, Humberto R. Maturana, and R. Uribe (1974) Autopoiesis: The organization of living systems, its characterization and a model. BioSystems, 5:187-196.

J. von Neumann (1945) "First Draft of a Report on the EDVAC". Moore School of Engineering (University of Pennsylvania).

J. von Neumann (1956) Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. Automata Studies, pp.43-98.

J. von Neumann (1966) The Theory of Self-Reproducing Automata. A. W. Burks, ed. University of Illinois Press, Urbana, IL.

Mark Weiser (1993) Some Computer Science Problems in Ubiquitous Computing. Communications of the ACM, July 1993. (reprinted as "Ubiquitous Computing". Nikkei Electronics; December 6, 1993; pp. 137-143.)

Wheeler, M. and Clark, A. (1999) Genic representation: reconciling content and causal complexity. British Jnl for Philosophy of Science 50, 103135.

Wolfram, Stephen (2002) *A New Kind of Science*. Wolfram Media, Inc., May 14, 2002. ISBN 1579550088.

[2] The DNA computer created by Ehud Shapiro looks very similar to a systemic computer, so perhaps a simple biological systemic computer already exists.