Entropy **2012**, *14*, 1939-1952; doi:10.3390/e14101939

OPEN ACCESS

entropy

ISSN 1099-4300

www.mdpi.com/journal/entropy

Article

Programming Unconventional Computers: Dynamics, Development, Self-Reference

Susan Stepney

York Centre for Complex Systems Analysis, University of York, York, YO10 5DD, UK;
E-Mail: susan.stepney@york.ac.uk

Received: 23 August 2012; in revised form: 8 October 2012 / Accepted: 9 October 2012 / Published: 17 October 2012

Abstract: Classical computing has well-established formalisms for specifying, refining, composing, proving, and otherwise reasoning about computations. These formalisms have matured over the past 70 years or so. Unconventional Computing includes the use of novel kinds of substrates–from black holes and quantum effects, through to chemicals, biomolecules, even slime moulds–to perform computations that do not conform to the classical model. Although many of these unconventional substrates can be coerced into performing classical computation, this is not how they "naturally" compute. Our ability to exploit unconventional computing is partly hampered by a lack of corresponding programming formalisms: we need models for building, composing, and reasoning about programs that execute in these substrates. What might, say, a slime mould programming language look like? Here I outline some of the issues and properties of these unconventional substrates that need to be addressed to find "natural" approaches to programming them. Important concepts include embodied real values, processes and dynamical systems, generative systems and their meta-dynamics, and embodied self-reference.

Keywords: unconventional computing; development and morphogenesis; self-reference and reflection

1. Introduction

Let us look at the genesis of conventional computing. Turing formalised the behaviour of real world "computers" (the name for human clerks carrying out calculations [1]) following an "effective procedure" (a finite sequence of discrete, well-defined rules). This formalisation led to the conventional abstract computational model: the Turing Machine (TM) [2].

Turning to the real world, there are many natural systems we might want to describe, understand, exploit or emulate computationally: termites building complex nests following (relatively) simple rules; slime moulds growing in the topology of road networks [3]; chemical oscillations set up to perform boolean operations [4,5]. Questions that arise from a TM-model mindset are: How can we interpret these behaviours as conventional computations? What are the relevant abstractions to build a computational model? How do they fit within the discrete TM-model?

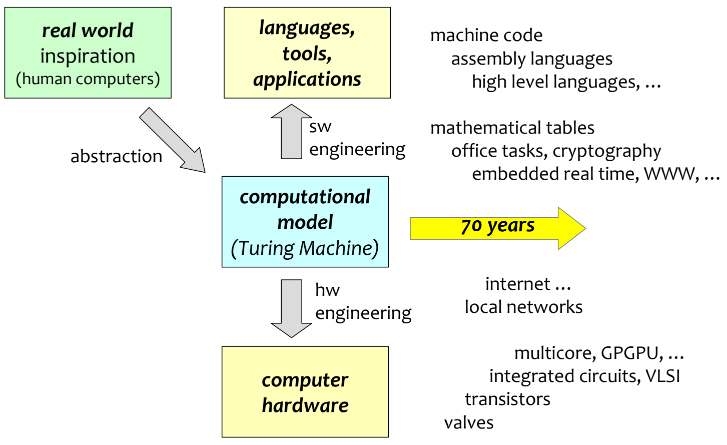
At this stage in the development of unconventional (or non-Turing) computation, I think that these are the wrong questions to ask. Instead, we should first investigate these systems to discover what computationally-interesting processes they perform "naturally": we should be physical process-driven rather than abstract model-driven [6,7]. Trying too hard to build conventional models obscures the potential power of these systems. I would not trust a slime mould computer to spell-check my writing, or calculate my tax return; its strengths lie elsewhere. Certain forms of computers, for example analogue devices, can perform their computations much more "naturally" (for example, much more power-efficiently [8]) than a digital version. So let us start from this point, discover what kinds of computation are natural to a range of systems, and then abstract our new unconventional computational models from there.

Furthermore, we should not worry that our current unconventional computers are ridiculously primitive compared to, say, our smartphones. Granted, some unconventional devices may have scaling issues (potentially due to their unary data encodings). Granted, we might have to sacrifice some benefits of classical computation to gain other benefits of unconventional computation; for example, Conrad [9] argues that "a computing system cannot at the same time have high programmability, high computational efficiency, and high evolutionary adaptability". But it is early days yet. Classical computation has seventy years of an exponentially growing lead on us.

2. Classical History and Unconventional Futures

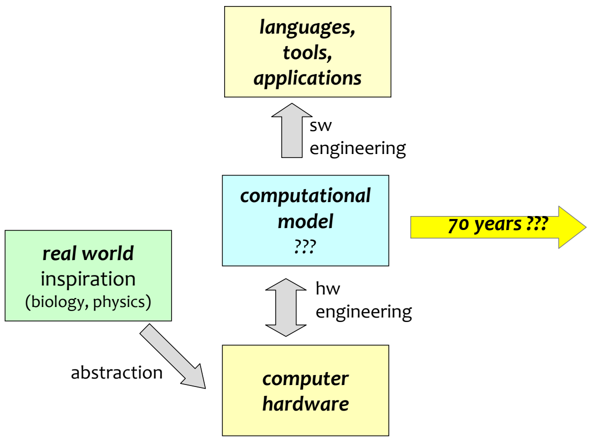
In a sense, classical Turing-model computation got things backwards: the computational theory was developed before most hardware and applications (Figure 1): the hardware had to be coerced into following the dictates of the theory, to compute what the theory demanded, rather than the theory following from what the hardware could compute naturally. The applications were then doubly constrained: by the theoretical model, and by the "unnatural" physical implementation of that model. Transistors in their "natural" state (if anything as highly engineered as a transistor can be considered to have a natural state) are analogue amplifiers; they are coerced to act as digital switches by being put in a saturated state. Sloman [10] argues that the TM model, by focusing on computation as a formal mathematical construct, ignores the important "class of information-processing machines that can interact causally with other physical systems", the class "important for AI (and philosophy of mind)".

Figure 1. Classical computation: the real world inspiration of human computers led to an abstract model, the Turing Machine. This was realised in hardware and exploited in software, and developed for 70 years, into a form unrecognisable to its early developers.


real world inspiration (human computers)
languages, tools, applications
machine code
assembly languages
high level languages, ...
abstraction
sw engineering
mathematical tables
office tasks, cryptography
embedded real time, WWW, ...
computational model (Turing Machine)
70 years
hw engineering
internet ...
local networks
computer hardware
multicore, GPGPU, ...
integrated circuits, VLSI
transistors
valves

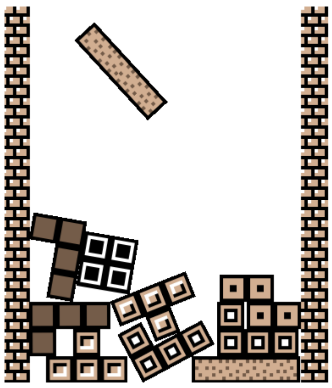
Unconventional computing seems to be taking a different route: the real world inspiration is leading to novel hardware (in some cases, wetware) devices, rather than directly to an abstract computational model. This "hardware first" approach is somewhat analogous to the emergence of life, where the physical material must exist before it could evolve to perform information processing. Our job as computer scientists is to work out good underlying computational models and appropriate languages that naturally fit with the hardware, and also to engineer more efficient and flexible hardware (Figure 2).

Figure 2. Unconventional computation: the real world inspiration of biological and other systems is leading to novel hardware. This must be abstracted into a computation model, and augmented with appropriate programming languages and tools. Seventy years from now, the technology will be unrecognisable from today's ideas.



Getting an appropriate abstract model is important. The wrong model (Figure 3), an unnatural model, will mean that our ability to exploit the unconventional substrates will be compromised.

Figure 3. The wrong model: screenshot partway through a game of Not Tetris (http://stabyourself.net/nottetris2, accessed on 6 August 2012).



3. Computational Models as Abstractions of Physics

Whereas classical computation may be charaterised as "theory (mathematics) first", unconventional computation might be thought of as "hardware (physics) first". Our unconventional computational models must respect, and be constrained by, the laws of physics. These constraints can change as our understanding of physical laws changes: for example, accelerating Turing Machines [11] are legal under Newtonian laws (which allow unbounded speeds), disallowed by special relativity (where the speed of light is an upper bound), and allowed again by general relativity (where the observer and computer can experience different proper times).

We know that the classical model of computation captures too little of reality: the physical model implicitly underlying the TM formalises an essentially Newtonian view of physics. Quantum physics allows more: it allows multiple symbols to be *superposed* in a single tape position [12] and *entangled* between positions. General relativity allows more: it allows the machine's frame and the observer's frame to experience different proper times; in particular a Malament–Hogarth spacetime allows an observer to experience finite proper time whilst the machine that they are observing experiences infinite proper time [13]. These two branches of physics are themselves a century old, and ununified. What of quantum gravity computers, or string-theoretic computers? The Turing model is *unphysical*.

However, some unconventional computational models capture too much: like TMs they are unphysical, but in a different way. Analogue computers usually use *continuous* physical quantities as analogues of the value being computed. These continuous physical quantities are *modelled* as real numbers. A single real number can encode an infinite amount of information in its countably infinite number of digits. But this does *not* mean that the physical quantity that it models can similarly encode an infinite amount of information. This has nothing to do with quantum limits to continuity. Well before such limits, even the most accurately measured fundamental physical constants are not measured to more than 10 or 12 decimal places [14]. The most accurately measured physical quantity, the rubidium hyperfine frequency, is known to an accuracy of 2.5×10^{-15} [15]. The value of the mathematical constant π to 39 digits can give the volume of the observable universe to the nearest atom [16]. To measure π to more precision than this, we would need a measuring device bigger than the size of the universe. Despite this, π has been calculated to 10 trillion decimal places [17]: an impressive computation, but a completely physically unmeasurable value. Computational models need to be based on real-world physics: not only the laws, but also the practical constraints.

What models of computation are suitable for natural physical computers? This includes not only exotic physics, but also biological systems. We need good abstractions, ones that do not impose unphysical requirements, and furthermore that naturally fit the implementations. So, for example, if a system is naturally noisy and non-deterministic, it is better to find a model that can exploit this, rather than engineer the substrate away from its natural state to coerce it into one that better matches some unnatural (for this substrate) deterministic model.

4. Inspired by Biological Modelling

Let us look at how biology is being conceptualised and modelled, in order to get some pointers to requirements for computational models of biological computers. We start with a pair of quotations, about organism-centric biology.

> Organic life exists only so far as it evolves in time. It is not a thing but a process—a never-resting continuous stream of events — Cassirer [18]

> It must be a biology that asserts the primacy of processes over events, of relationships over entities, and of development over structure. — Ingold [19]

A process-centric description is arguably also needed in the context of emergence [20]. To summarise these ideas: "Life is a verb, not a noun." [21].

So, the emphasis from these authors is of process, dynamics, development (which words, despite themselves being nouns, describe verb-like properties), rather than of entities, states, events. Let us look at these three features, how well they are captured by current formalisms, and what more is needed.

4.1. Process

Process might seem like an easy starting point, as we have process algebras and process calculi galore [22–29] in computer science. These describe the interactions between concurrently-executing processes, and (one of) the semantics of a process is its *trace*: a ("never-resting") stream of events. There is a move in this direction as process algebras have been used, for example, to model aspects of systems biology [30,31], and to model slime mould computation [32].

Process algebras, with their non-terminating processes, can have their semantics modelled in non-well-founded (NWF) set theory [33,34]. NWF set theory replaces the usual axiom of foundation with the anti-foundation axiom (AFA); many of the well-known operations of set theory (such as union, intersection, membership) carry across. The crucial difference is that, unlike in the better-known well-founded set theory, in NWF set theory we can have perfectly well-defined systems with infinite chains of membership that do not bottom-out, $\ldots \in X_3 \in X_2 \in X_1 \in X_0$, and cycles of membership, such as $X \in Y \in X$ and even $X \in X$. So, for example, the NWF set with (valid) circular definition $C = \{t, C\}$ "unfolds" to $C = \{t, \{t, \{t, \ldots\}\}\}$.

Using NWF set theory gives a very different view of the world. With well-founded sets, we can start at the bottom (that is what the axiom of foundation requires to exist), with the relevant "atoms", and construct sets from these atoms, then bigger sets from these sets, inductively. This seems like the natural, maybe the only, way to construct things. But non-well-foundedness is not like this. There are perfectly good NWF sets that just cannot be built this way: there are sets with no "bottom" or "beginning": it can be "turtles all the way down" [35]. NWF set theory also allows sets that are intrinsically circular, or self-referential. It might well be true that "the axiom of foundation has played almost no role in mathematics outside of set theory itself" [36], but set theory has had an enormous impact on the way many scientists model the world. Might it be that the whole paradigm of reductionism relies on the *mathematically unnecessary* axiom of foundation? Process algebras, with their NWF basis, might well offer a new view on how things can be constructed.

But it is not all good news. Well-founded set theory and mathematical induction is taught to school children (well, it was in my day, at least); NWF set theory, coalgebra, and coinduction are currently found only in quite densely-mathematical textbooks and papers. We need a *Coalgebra for Dummies*. One of the most accessible introductions currently available is Bart Jacobs' "two-thirds of a book in preparation" [37].

More importantly for programming unconventional computers, most process algebras cannot exploit endogenous novelty. Process and communication channel types are predefined; no new kinds of processes or channels can emerge and then be exploited from within the formal system. This ability may require a *reflective* [38] process algebra (one that can represent, reason about, and act on aspects of itself). Reflective self-modification may be a prerequisite for describing any high-level system displaying open-ended novelty [39,40]. For example, PiLar [41,42] is a reflective process-algebraic architecture description language, developed to define software architectures in terms of patterns of change; reflection allows it to change the patterns of change. The mathematical underpinnings need to incorporate reflective processes; NWF set theory, with its allowance of circular definitions, is suitable for modelling reflective systems that can model themselves.

4.2. Dynamics

For a formalism underpinning *dynamics*, we could consider dynamical systems theory [43–45]. This is a very general formalism: a dynamical system is defined by its state space, and a rule determining its motion through that state space. In a continuous physical dynamical system, that rule is given by the relevant physical laws. Classical computation can be described in discrete dynamical systems terms [46], where the relevant rule is defined by the computer program. Hence it seems possible that a dynamical systems approach could provide a route to an unconventional computational view of physically embodied systems exploiting the natural dynamics of their material substrates (such as exhibited by Pask's ear [47,48], Thompson's FPGA tone discriminator [49], and Harding's *in materio* computation [50]).

Dynamical systems can be understood at a generic level in terms of the structure of their state space: their attractors, trajectories, parameterised bifurcations, and the like [43–45]. Trajectories may correspond to computations and attractors may correspond to computational results [46]; new attractors arising from bifurcations may correspond to emergent properties [20,51].

A dynamical systems view allows us to unify the concepts of process and particle (of verb and noun). Everything is process (motion on a trajectory, from transient behaviour to motion on an attractor), but if viewed on a long enough timescale, its motion on an attractor blurs into a particle. "An attractor functions as a symbol when it is observed ... *by a slow observer*" [52]. On this longer timescale the detailed motion is lost, and a stable pattern emerges as an entity in its own right. This entity can then have a dynamics in a state space of its own, and so on, allowing multiple levels of emergence.

However, the mathematical underpinnings support none of these exciting and intuitive descriptions. Classical dynamical systems theory (which pertains to nonlinear differential and difference equations) deals with systems in a static, pre-defined state space.

The closest the state space itself comes to being dynamic is by being parameterised, where a change in the parameter value can lead to a change in the attractor structure, including bifurcations. Here the parameter links a family of dynamical systems. If the parameter can be linked to a feature of the computational system, then it can be used to control the shape of the dynamics.

Ideally, the control of the parameter should be internal to the system, so that the computation can have some control over its own dynamics. Current dynamical systems theory does not have this reflective component: the parameter is external to the system. A full computational dynamical systems theory would need to include meta-dynamics, the dynamics of the state space change. Currently meta-dynamics is handled in an *ad hoc* fashion, by separating it out into a slower timescale change [53,54].

4.3. Development

The requirement for (the state space of) systems to "grow" is addressed biologically as the study of *development*: the growth from egg or seed to mature organism. Death is also a fundamental part of this process; certain structures die off in order to produce the mature patterns (for example: during neural development [55]; removing webbing between fingers and toes [56]). This process is referred to in biology as "programmed cell death". Programmed death can be thought of as a kind of non-conservative extension.

State space growth happens naturally in most classical programming languages: for example, statements such as malloc(n) or new Obj(p) allocate new memory for the computation to use, thereby increasing the dimensionality of the computational state space. Death happens when the memory is later freed. However, these everyday programming actions are rarely cast in explicit developmental terms. One notable exception is Fontana's λ-calculus-based "algorithmic chemistry" [57,58].

Mathematically, such dynamic changing of the state space itself is less common. For example, a set of coupled differential equations might be used to model multiple reacting chemical species, with the rate of change of each species being modelled by its own equation, and dependent on the concentrations of the other species. The state space is given by the individual species and their concentrations, one dimension per species. However, there is no way within the system for new equations to be added to cater for new species that may develop; there is no way for the system to change its own state space. Any such addition has to be done from outside the system of equations. For example, a metadynamics approach can be used whereby a graph is overlaid on the system of equations, controlling which ones are active at any time [59]; this approach, however, still requires prior knowledge of which equations will be needed.

One example of a formal logical system that can be grown (non-conservatively) is Leśniewski's inscriptional logic [60], although again, this growth must be performed outside the system itself.

Explicit development is captured by generative grammars such as L-systems [61], where symbol growth rules are iteratively applied to all the symbols of a string in parallel, and by rewriting systems such as P-systems [62] and other membrane computing systems. These discrete systems can be cast as special cases of "dynamical systems with dynamical structure" within the MGS language [63–65], based on local transformations of topological collections. There are further models of computational morphogenesis, for example [66–69].

These formalisms capture mainly the growth of discrete state spaces. There is still the interesting question of growing continuous state spaces: how does a new continuous state space dimension appear in continuous time? How does a hybrid system containing both discrete and continuous state space dimensions grow?

If we are thinking of systems that can exhibit perpetual novelty and emergence, then we also need a system where the growth rules can grow. The growing state space (new dimensions, new kinds of dimensions) should open up new possibilities of behaviour. One way to do this is to embed the rules into the space itself, so that as the space grows, the rules governing how the space grows can themselves grow. Such an approach can be used to program self-replicating spaces [70]. Self-Modifying Cartesian Genetic Programming (SMCGP) [71,72] is an evolutionary algorithm that includes a self-modifying developmental process: the digital genome encodes a phenotype that is a graph program; some of the graph program operations can be instructions that change the graph structure, including changing the self-modification itself.

In such cases of growth, the computation is not a trajectory though a static state space; it is the meta-trajectory of the growing space itself.

4.4. Self-reference

Although *self-reference* is not one of the features identified from the biological modelling inspiration above, it has come up in the discussions around each individual feature, and is a crucial aspect of classical computation and biological self-reproduction.

The biologist Robert Rosen claims that there is a sense in which self-definition is an essential feature of life that cannot be replicated in a computer [73]. He defines organisms to be "closed to efficient causation": Aristotle's "efficient cause" is the cause that brings something about; life is self-causing, self-defining, self-building, self-maintaining, autopoietic [74–77]. Rosen claims that "mechanisms", including computer programs (and hence simulations of life), cannot be so closed, because they require something outside the system to define them: they have an arbitrary non-grounded semantics. That is, there is an arbitrary separation of the semantics of a program (a virtual machine) and of its implementation (the physical machine); life however has only the one, physical, semantics.

However, it is not as straightforward as that. Organic life also has a degree of arbitrariness in its semantics. As Danchin points out [76], there is a level of indirection in the way organisms represent their functionality: the mapping from DNA codons to the amino acids they code for is essentially arbitrary. Additionally, there is the observation that emergent properties (such as life itself) can be relatively insensitive to the details of the substrate [78], suggesting the possibility of a certain arbitrariness in the allowed mappings from physical substrate to emergent process. So life too may be embodied in a virtual machine with arbitrary semantics.

What is common to biological and computational self-reference is that the "data" and "program" are the "same kind of stuff", so that programs can modify data that can be interpreted as new programs. In biology this stuff comprises chemicals: a chemical may be passive data (uninterpreted DNA that codes for certain proteins); it may be an executing "program" (some active molecular machinery, possibly manipulating DNA).

So self-referential, self-modifying code is crucial in biology. It is achievable in classical computation through reflective interpreted programs. It is plausible that this capability is also crucial for unconventional computation executing on the natural embodied dynamics of physical substrates. Embodied developmental computation could have software processes that in part control the growth of new hardware, which in turn supports novel software capabilities [79].

5. Conclusions

Unconventional computers, particularly those embodied in biological-like substrates, may require novel programming paradigms. By looking to biology, we see that these paradigms should include as first class properties the concepts of: process; dynamics and meta-dynamics; development; and self-reference.

The challenges to the unconventional computing research community are to develop such paradigms in a manner that supports powerful models of computation, and to engineer varieties of programmable hardware that implement such models "naturally". The noted formalisms merely suggest appropriate starting points; much yet remains to be done. This should not be surprising: classical computation has matured tremendously over the last seventy years, while unconventional computing is still in its infancy. If over the next seventy years unconventional computing can make even a fraction of the advances that classical computing has made in that time, that new world of computation will be unrecognisably different from today.

Acknowledgments

This is a revised and extended version of an invited paper [80] presented to the Symposium on Natural/Unconventional Computing and Its Philosophical Significance, at the AISB/IACAP World Congress 2012. My thanks to Gordana Dodig-Crnkovic, Julian Miller, Simon O'Keefe, and the anonymous referees, for helpful comments, and to Peter Simons for drawing the work of Stanisław Leśniewski to my attention.

References

1. Copeland, B.J. The Modern History of Computing. In *The Stanford Encyclopedia of Philosophy*, 2008 ed.; Zalta, E.N., Ed.; Archived online: http://plato.stanford.edu/archives/fall2008/entries/computing-history/ (accessed on 6 August 2012).
2. Turing, A.M. On computable numbers, with an application to the entscheidungsproblem. *Proc. Lond. Math. Soc.* **1937**, *S2–S42*, 230–265.
3. Adamatzky, A. *Physarum Machines: Computers From Slime Mould*; World Scientific: Singapore, 2010.
4. Tóth, Á.; Showalter, K. Logic gates in excitable media. *J. Chem. Phys.* **1995**, *103*, 2058–2066.
5. Adamatzky, A.; de Lacy Costello, B.; Asai, T. *Reaction-Diffusion Computers*; Elsevier: Amsterdam, The Netherlands, 2005.
6. Miller, J.F.; Downing, K. Evolution *in materio*: Looking Beyond the Silicon Box. Proceedings of NASA/DoD Conference on Evolvable Hardware, Washington D.C., USA, 15–18 July 2002; pp. 167–176.
7. Stepney, S. The neglected pillar of material computation. *Phys. D: Nonlinear Phenom.* **2008**, *237*, 1157–1164.
8. Enz, C.C.; Vittoz, E.A. CMOS Low-power Analog Circuit Design. In *Emerging Technologies, Tutorial for 1996 International Symposium on Circuits and Systems*; IEEE Service Center: Piscataway, NJ, USA, 1996; pp. 79–133.
9. Conrad, M. The Price of Programmability. In *The Universal Turing Machine*; Herken, R., Ed.; Oxford University Press: Oxford, UK, 1988; pp. 285–307.
10. Sloman, A. The Irrelevance of Turing Machines to Artificial Intelligence. In *Computationalism: New Directions*; Scheutz, M., Ed.; MIT Press: Cambridge, MA, USA, 2002; pp. 87–127.
11. Copeland, B.J. Accelerating turing machines. *Minds Mach.* **2002**, *12*, 281–300.
12. Deutsch, D. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A Math. Phys. Sci.* **1985**, *400*, 97–117.
13. Hogarth, M. Does general relativity allow an observer to view an eternity in a finite time? *Found. Phys. Lett.* **1992**, *5*, 173–181.
14. NIST. The NIST Reference on Constants, Units, and Uncertainty, 2011. Available online: http://physics.nist.gov/cuu/Constants/ (accessed on 6 August 2012).
15. NPL. What is the most accurate measurement known? 2010. Available online: http://www.npl.co.uk/reference/faqs/what-is-the-most-accurate-measurement-known-(faq-quantum) (accessed on 6 August 2012).
16. Arndt, J.; Haenel, C. π *Unleashed*; Springer: Berlin/Heidelberg, Germany, 2001.
17. Yee, A.J.; Kondo, S. Round 2 …10 Trillion Digits of Pi, 2011. Available online: http://www.numberworld.org/misc_runs/pi-10t/details.html (accessed on 6 August 2012).
18. Cassirer, E. *An Essay on Man*; Yale University Press: New Haven, CT, USA, 1944.
19. Ingold, T. An anthropologist looks at biology. *Man* **1990**, *25*, 208–229.

20. Stepney, S.; Polack, F.; Turner, H. Engineering Emergence. Proceedings of the ICECCS 2006: 11th IEEE International Conference on Engineering of Complex Computer Systems, Stanford, CA, USA, 15–17 August 2006; pp. 89–97.
21. Gilman, C.P. *Human Work*; McClure, Philips and Co: New York, NY, USA, 1904.
22. Hoare, C.A.R. *Communicating Sequential Processes*; Prentice Hall: Upper Saddle River, NJ, USA, 1985.
23. Milner, R. *A Calculus of Communicating Systems*; Springer: Berlin/Heidelberg, Germany, 1980.
24. Milner, R. *Communication and Concurrency*; Prentice Hall: Upper Saddle River, NJ, USA, 1989.
25. Milner, R. *Communicating and Mobile Systems: The π-Calculus*; Cambridge University Press: Cambridge, UK, 1999.
26. Milner, R. *The Space and Motion of Communicating Agents*; Cambridge University Press: Cambridge, UK, 2009.
27. Cardelli, L.; Gordon, A.D. Mobile ambients. *Theor. Comput. Sci.* **2000**, *240*, 177–213.
28. Hillston, J. *A Compositional Approach to Performance Modelling*; Cambridge University Press: Cambridge, UK, 1996.
29. Baeten, J.C.M.; Weijland, W.P. *Process Algebra*; Cambridge University Press: Cambridge, UK, 1990.
30. Ciocchetta, F.; Hillston, J. Process Algebras in Systems Biology. In *Formal Methods for Computational Systems Biology*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5016, pp. 265–312.
31. Calder, M.; Hillston, J. Process Algebra Modelling Styles for Biomolecular Processes; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5750, pp. 1–25.
32. Schumann, A.; Adamatzky, A. Logical Modelling of Physarum Polycephalum. *CoRR* **2011**, doi:abs/1105.4060.
33. Aczel, P. *Non-well-founded Sets*; CSLI: Stanford, CA, USA, 1988.
34. Sangiorgi, D. On the origins of bisimulation and coinduction. *ACM Trans. Progr. Lang. Syst.* **2009**, *31*, 15:1–15:41.
35. Hawking, S.W. *A Brief History of Time*; Bantam Dell: New York, NY, USA, 1988.
36. Barwise, J.; Etchemendy, J. *The Liar: An Essay on Truth and Circularity*; Oxford University Press: Oxford, UK, 1987.
37. Jacobs, B. *Introduction to Coalgebra. Towards Mathematics of States and Observations*; draft v2.00. Available online: http://www.cs.ru.nl/B.Jacobs/CLG/JacobsCoalgebraIntro.pdf (accessed on 8 October 2012).
38. Maes, P. *Concepts and Experiments in Computational Reflection*; ACM Press: New York, NY, USA, 1987; pp. 147–155.
39. Hickinbotham, S.; Stepney, S.; Nellis, A.; Clarke, T.; Clark, E.; Pay, M.; Young, P. Embodied Genomes and Metaprogramming. In *Advances in Artificial Life, ECAL 2011*, Proceedings of the Eleventh European Conference on the Synthesis and Simulation of Living Systems, Paris, France, August 2011; MIT Press: Cambridge, MA, USA, 2011; pp. 334–341.

40. Stepney, S.; Hoverd, T. Reflecting on Open-Ended Evolution. In *Advances in Artificial Life, ECAL 2011*, Proceedings of the Eleventh European Conference on the Synthesis and Simulation of Living Systems, Paris, France, August 2011; MIT Press: Cambridge, MA, USA, 2011; pp. 781–788.
41. Cuesta, C.; de la Fuente, P.; Barrio-Solórzano, M.; Beato, E. Coordination in a Reflective Architecture Description Language. In *Coordination Models and Languages*; Arbab, F., Talcott, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2315, pp. 479–486.
42. Cuesta, C.; Romay, M.; de la Fuente, P.; Barrio-Solórzano, M. Reflection-Based, Aspect-Oriented Software Architecture. In *Software Architecture*; Oquendo, F., Warboys, B., Morrison, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3047, pp. 43–56.
43. Alligood, K.T.; Sauer, T.D.; Yorke, J.A. *Chaos : An Introduction to Dynamical Systems*; Springer: Berlin/Heidelberg, Germany, 1996.
44. Strogatz, S.H. *Nonlinear Dynamics and Chaos*; Westview Press: Boulder, CO, USA, 1994.
45. Beer, R.D. A dynamical systems perspective on agent-environment interaction. *Artif. Intell.* **1995**, *72*, 173–215.
46. Stepney, S. Nonclassical Computation: A Dynamical Systems Perspective. In *Handbook of Natural Computing*; Rozenberg, G., Bäck, T., Kok, J.N., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 4, Chapter 59, pp. 1979–2025.
47. Pask, G. The Natural History of Networks. In *Self-Organzing Systems*; Yovits, M.C., Cameron, S., Eds.; Pergamon Press: Oxford, UK, 1960.
48. Cariani, P. To evolve an ear: Epistemological implications of Gordon Pask's electrochemical devices. *Syst. Res.* **1993**, *10*, 19–33.
49. Thompson, A.; Layzell, P.J.; Zebulum, R.S. Explorations in design space: Unconventional electronics design through artificial evolution. *IEEE Trans. Evol. Comput.* **1999**, *3*, 167–196.
50. Harding, S.L.; Miller, J.F.; Rietman, E.A. Evolution in Materio: Exploiting the physics of materials for computation. *IJUC* **2008**, *4*, 155–194.
51. Goldstein, J. Emergence as a construct: History and issues. *Emergence* **1999**, *1*, 49–72.
52. Abraham, R.H. Dynamics and Self-Organization. In *Self-organizing Systems: The Emergence of Order*; Yates, F.E., Ed.; Plenum: New York, NY, USA, 1987; pp. 599–613.
53. Baguelin, M.; LeFevre, J.; Richard, J.P. A Formalism for Models with a Metadynamically Varying Structure. Proceedings of the European Control Conference, Cambridge, UK, 1–4 September 2003.
54. Moulay, E.; Baguelin, M. Meta-dynamical adaptive systems and their application to a fractal algorithm and a biological model. *Phys. D* **2005**, *207*, 79–90.
55. Kuan, C.Y.; Roth, K.A.; Flavell, R.A.; Rakic, P. Mechanisms of programmed cell death in the developing brain. *Trends Neurosci.* **2000**, *23*, 291–297.
56. Pajni-Underwood, S.; Wilson, C.P.; Elder, C.; Mishina, Y.; Lewandoski, M. BMP signals control limb bud interdigital programmed cell death by regulating FGF signaling. *Development* **2007**, *134*, 2359–2368.
57. Fontana, W. Algorithmic Chemistry. In *Artificial Life II*; Addison-Wesley, Boston, MA, USA, 1991; pp. 159–209.

58. Fontana, W.; Buss, L.W. The Barrier of Objects: From Dynamical Systems to Bounded Organizations. In *Boundaries and Barriers*; Casti, J.L., Karlqvist, A., Eds.; Addison-Wesley: Boston, MA, USA, 1996; Chapter 4, pp. 56–116.
59. Bagley, R.J.; Farmer, J.D. Spontaneous Emergence of a Metabolism. In *Artificial Life II*; Addison-Wesley: Boston, MA, USA, 1991; pp. 93–140.
60. Simons, P. Reasoning on a tight budget: Lesniewski's nominalistic metalogic. *Erkenntnis* **2002**, *56*, 99–122.
61. Prusinkiewicz, P.; Lindenmayer, A. *The Algorithmic Beauty of Plants*; Springer: Berlin/Heidelberg, Germany, 1990.
62. Păun, G. Computing with membranes. *J. Comput. Syst. Sci.* **2000**, *61*, 108–143.
63. Giavitto, J.L.; Michel, O. Data Structure as Topological Spaces. Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02, Kobe, Japan, 15–19 October 2002; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2509, pp. 137–150.
64. Giavitto, J.L.; Spicher, A. Topological rewriting and the geometrization of programming. *Phys. D* **2008**, *237*, 1302–1314.
65. Michel, O.; Banâtre, J.P.; Fradet, P.; Giavitto, J.L. Challenging questions for the rationale of non-classical programming languages. *Int. J. Unconv. Comput.* **2006**, *2*, 337–347.
66. Doursat, R. Organically Grown Architectures: Creating Decentralized, Autonomous Systems by Embryomorphic Engineering. In *Organic Computing*; Würtz, R.P., Ed.; Springer: Berlin/Heidelberg, Germany, 2008, pp. 167–200.
67. Doursat, R.; Sayama, H.; Michel, O. *Morphogenetic Engineering: Toward Programmable Complex Systems*; Springer: Berlin/Heidelberg, Germany, 2012, in press.
68. MacLennan, B.J. Models and Mechanisms for Artificial Morphogenesis. In *Natural Computing*, Proceedings in Information and Communications Technology, Himeji, Japan, September 2009; Peper, F., Umeo, H., Matsui, N., Isokawa, T., Eds.; Springer: Tokyo, Japan, 2010; Volume 2, pp. 23–33.
69. MacLennan, B.J. Artificial morphogenesis as an example of embodied computation. *Int. J. Unconv. Comput.* **2011**, *7*, 3–23.
70. Tomita, K.; Murata, S.; Kurokawa, H. Self-description for construction and computation on graph-rewriting automata. *Artif. Life* **2007**, *13*, 383–396.
71. Harding, S.; Miller, J.F.; Banzhaf, W. Self-modifying Cartesian Genetic Programming. Proceedings of the GECCO 2007, London, UK, 7–11 July 2007; Lipson, H., Ed.; ACM: New York, NY, USA, 2007; pp. 1021–1028.
72. Harding, S.; Miller, J.F.; Banzhaf, W. Self-modifying Cartesian Genetic Programming. In *Cartesian Genetic Programming*; Miller, J.F., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; chapter 4, pp. 101–124.
73. Rosen, R. *Life Itself: A Comprehensive Enquiry Into the Nature, Origin, and Fabrication of Life*; Columbia University Press: New York, NY, USA, 1991.
74. Maturana, H.R.; Varela, F.J. *Autopoeisis and Cognition: The Realization of the Living*; D. Reidel Publishing Company: Boston, MA, USA, 1980.

75. Mingers, J. *Self-Producing Systems: Implications and Applications of Autopoiesis*; Plenum: New York, NY, USA, 1995.
76. Danchin, A. *The Delphic Boat: What Genomes Tell Us*; Harvard University Press: Cambridge, MA, USA, 2002.
77. McMullin, B. Thirty years of computational autopoiesis: A review. *Artif. Life* **2004**, *10*, 277–295.
78. Laughlin, R.B. *A Different Universe: Reinventing Physics from the Bottom Down*; Basic Books: New York, NY, USA, 2005.
79. Kampis, G. Self-Modifying Systems. In *Biology and Cognitive Science: A New Framework for Dynamics, Information, and Complexity*; Pergamon Press: Oxford, UK, 1991.
80. Stepney, S. Unconventional Computer Programming. Proceedings of the Symposium on Natural/Unconventional Computing and Its Philosophical Significance, Birmingham, UK, 2–3 July 2012; Dodig-Crnkovic, G., Giovagnoli, R., Eds.; pp. 12–15.