Unconventional Computer Programming

Susan Stepney[1]

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 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 substrates can be "tortured" to perform 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.

1 Introduction

Let's look at the genesis of conventional computing. Turing formalised the behaviour of real world "computers" (human clerks carrying out calculations [11]) following a finite sequence of discrete, well-defined rules. This formalisation led to an abstract model: the Turing Machine (TM) [46].

Turning to the real world, there are many processes we might want to describe, understand, or exploit computationally: termites building complex nests following (relatively) simple rules; slime moulds growing in the topology of road networks; chemical oscillations set up to perform boolean operations. What are the relevant abstractions? Are they just the discrete TM again?

At this stage in the development of unconventional (or non-Turing) computation, I think that this is the wrong question. First, we should investigate these processes to discover what computations they perform "naturally". I would not trust a slime mould computer to spell-check my writing, or calculate my tax return. But certain forms of computers, for example analogue devices, can perform their computations much more "naturally" (for example, much more power-efficiently [16, p83]) than a digital version. Let's start from this point, discover what kinds of computation are natural to a range of systems, and then abstract from there.

We should not worry that our unconventional computers are ridiculously primitive compared to our smartphones: classical computation has seventy years of an exponentially growing lead on us.

[1] York Centre for Complex Systems Analysis, University of York, UK, email: susan.stepney@york.ac.uk

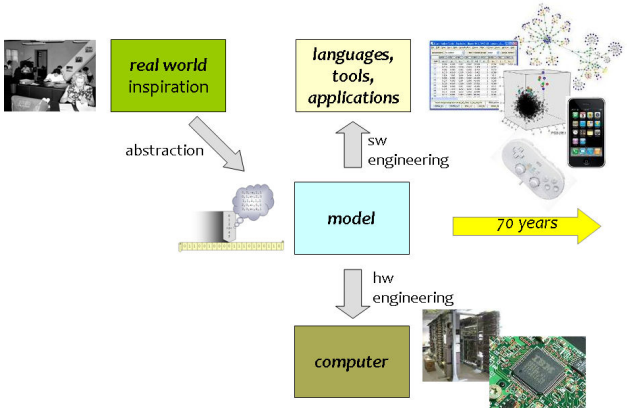

Figure 1. Classical computation: the real world inspiration of human computers led to an abstract model. This was realised in hardware and exploited in software, and developed for 70 years, into a form unrecognisable to its early developers.

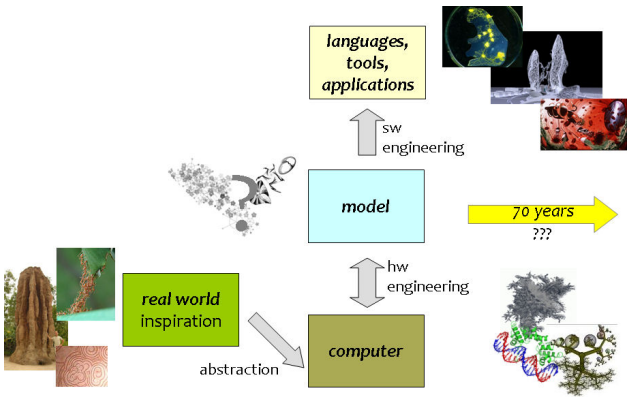

Figure 2. Unconventional computation: the real world inspiration of biological and other systems is leading to novel hardware. This must be abstracted into a computation model, and augmented with appropriate programming languages an tools. 70 years from now, the technology will be unrecognisable from today's ideas.

2 Classical history and unconventional futures

In a sense, classical computation got things backwards: theory before hardware and applications (figure 1). 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 a model. Our job as computer scientists is to work out good underlying computational models and appropriate languages

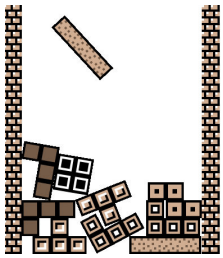

Figure 3. The wrong model (screenshot partway through a game of Not Tetris, http://stabyourself.net/nottetris2)

that naturally fit with the hardware, and also to engineer more efficient and flexible hardware (figure 2).

Getting a good 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.

3 Computational models as abstractions of physics

We know that the classical model of computation captures too little of reality: its underlying workings formalise an essentially Newtonian view of physics. Quantum physics allows multiple symbols to be *superposed* in a single tape position [15], and *entangled* between positions. General relativity 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 [24]. And these two branches of physics are themselves a century old. 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 a countably infinite amount of information. But this does *not* mean that the physical quantity that it models can 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 [35]. The most accurately measured physical quantity, the rubidium hyperfine frequency, is known to an accuracy of 2.5×10^{-15} [36]. The value of the mathematical constant π to 39 digits can give the volume of the observable universe to the nearest atom [4, p17]. 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 [47]: 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, that not only do not impose unphysical requirements, but 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 one that better matches some unnatural deterministic model.

4 Inspired by biological modelling

Let's 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 [10, p49, ch.IV]

> It must be a biology that asserts the primacy of processes over events, of relationships over entities, and of development over structure. — Ingold [25]

A process-centric description is arguably also needed in the context of emergence [43]. To summarise these ideas: "Life is a verb, not a noun." [19, p203, ch.X].

So, the emphasis from these authors is of process, dynamics, development (which, despite themselves being nouns, describe verb-like properties), rather than of entities, states, events. Let's 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 [5, 9, 22, 23, 30–33] 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.

Process algebras, with their non-terminating processes, can have their semantics modelled in non-well-founded set theory [2, 40]. 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 $X \in X$.

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 well-foundedness guarantees exists), 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 non-well-founded sets that just cannot be built this way: there are sets with no "bottom" or "beginning": it can be "turtles all the way down" [21, p1]. NWF set theory allows sets that are intrinsically circular, or self-referential, too. It might be true that "the axiom of foundation has played almost no role in mathematics outside of set theory itself" [7, p58], but set theory has had an enormous impact on the way many scientists model the world. Might it be that the whole concept 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 is taught to school children; NWF set theory, coalgebra, and coinduction, are currently found only in quite densely mathematical books 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" [26].

More importantly for programming unconventional computers, most process algebras cannot exploit endogenous novelty. Processes and communication channels are predefined; no new kinds of processes or channels can emerge and then be exploited by the formal system. This may require a reflective [27] process algebra. Reflection may be a pre-requisite for describing any system displaying open-ended novelty [42]. PiLar [12,13] is a reflective process-algebraic architecture description language, developed to define software architectures in terms of patterns of change; reflection allows it to change itself: to change the patterns of change. The mathematical underpinnings need to incorporate such features; 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 [3, 8, 44]. 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 [41], where the relevant rule is defined by the computer program. Hence it seems that dynamical systems approach can provide a route to an unconventional computational view of physical embodied systems exploiting the natural dynamics of their material substrates.

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 [3,8,44]. Trajectories may correspond to computations and attractors may correspond to computational results [41]; new attractors arising from bifurcations may correspond to emergent properties [20,43].

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*" [1]. 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 deals with closed systems (no inputs or outputs, no coupling to the environment) 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 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 metadynamics, the dynamics of the state space change. Currently metadynamics is handled in an *ad hoc* fashion, by separating it out into a slower timescale change [6,34].

4.3 Development

The requirement for "development", allowing (the state space of) systems to "grow", 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. However, this everyday usage is rarely cast in explicit developmental terms.

Explicit development is captured by generative grammars such as L-systems [38], and by rewriting systems such as P-systems [37] and other membrane computing systems. These discrete systems can be cast as special cases of "dynamical systems with dynamical structure" within the MGS language [17, 18, 29], based on local transformations of topological collections.

These formalisms capture mainly the growth of discrete spaces. There is still the interesting question of growing continuous spaces: how does a new continuous dimension appear in continuous time? How does a hybrid system containing both discrete and continuous 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 space (new dimensions, new kinds of dimensions) should open up new possibilities of behaviour. One way to do this is to embed the rules in the space itself, so that as the space grows, the rules governing how the space grows themselves grow. This approach can be used to program self-replicating spaces [45]. Now the computation is not a trajectory though a static state space, it is the 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, it has come up in the discussions around each individual feature, and is a crucial feature 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 [39]. 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, autopoietic [28]. 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 the program (a virtual machine) and the implementation (the physical machine); life however has only the one, physical, semantics.

However, it is not as straightforward as that. Organic life also has an arbitrary semantics. As Danchin points out [14, p110], 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. 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.

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, development, and self-reference.

Some existing formalisms may suggest appropriate starting points, but 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.

REFERENCES

[1] Ralph H. Abraham, 'Dynamics and self-organization', in *Self-organizing Systems: The Emergence of Order*, ed., F. Eugene Yates, 599–613, Plenum, (1987).

[2] Peter Aczel, *Non-well-founded sets*, CSLI, 1988.

[3] Kathleen T. Alligood, Tim D. Sauer, and James A. Yorke, *Chaos : an introduction to dynamical systems*, Springer, 1996.

[4] Jörg Arndt and Christoph Haenel, π *Unleashed*, Springer, 2001.

[5] J. C. M. Baeten and W. P. Weijland, *Process Algebra*, Cambridge University Press, 1990.

[6] M. Baguelin, J. LeFevre, and J. P. Richard, 'A formalism for models with a metadynamically varying structure', in *Proc. European Control Conference, Cambridge, UK*, (2003).

[7] Jon Barwise and John Etchemendy, *The Liar: an essay on truth and circularity*, Oxford University Press, 1987.

[8] Randall D. Beer, 'A dynamical systems perspective on agent-environment interaction', *Artificial Intelligence*, **72**(1-2), 173–215, (1995).

[9] Luca Cardelli and Andrew D. Gordon, 'Mobile ambients', *Theor. Comput. Sci.*, **240**(1), 177–213, (2000).

[10] Ernst Cassirer, *An Essay on Man*, Yale University Press, 1944.

[11] B. Jack Copeland, 'The modern history of computing', in *The Stanford Encyclopedia of Philosophy*, ed., Edward N. Zalta, fall 2008 edn., (2008).

[12] Carlos Cuesta, Pablo de la Fuente, Manuel Barrio-Solórzano, and Encarnacin Beato, 'Coordination in a reflective architecture description language', in *Coordination Models and Languages*, eds., Farhad Arbab and Carolyn Talcott, volume 2315 of *LNCS*, 479–486, Springer, (2002).

[13] Carlos Cuesta, M. Romay, Pablo de la Fuente, and Manuel Barrio-Solórzano, 'Reflection-based, aspect-oriented software architecture', in *Software Architecture*, eds., Flavio Oquendo, Brian Warboys, and Ron Morrison, volume 3047 of *LNCS*, 43–56, Springer, (2004).

[14] Antoine Danchin, *The Delphic Boat: what genomes tell us*, Harvard University Press, 2002.

[15] David Deutsch, 'Quantum theory, the Church-Turing principle and the universal quantum computer', *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, **400**(1818), 97–117, (1985).

[16] C. C. Enz and E. A. Vittoz, 'CMOS low-power analog circuit design', in *Emerging Technologies, Tutorial for 1996 International Symposium on Circuits and Systems*, 79–133, IEEE Service Center, (1996).

[17] Jean-Louis Giavitto and Olivier Michel, 'Data structure as topological spaces', in *Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02*, volume 2509 of *LNCS*, pp. 137–150. Springer, (2002).

[18] Jean-Louis Giavitto and Antoine Spicher, 'Topological rewriting and the geometrization of programming', *Physica D*, **237**(9), 1302–1314, (2008).

[19] Charlotte Perkins Gilman, *Human Work*, McClure, Philips and Co, 1904.

[20] Jeffrey Goldstein, 'Emergence as a construct: History and issues', *Emergence*, **1**(1), 49–72, (1999).

[21] Stephen W. Hawking, *A Brief History of Time*, Bantam Dell, 1988.

[22] Jane Hillston, *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.

[23] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[24] Mark Hogarth, 'Does general relativity allow an observer to view an eternity in a finite time?', *Foundations of Physics Letters*, **5**(2), 173–181, (1992).

[25] Tim Ingold, 'An anthropologist looks at biology', *Man*, **25**, 208–229, (1990).

[26] Bart Jacobs, *Introduction to Coalgebra. Towards Mathematics of States and Observations*, 2005. draft available from http://www.cs.ru.nl/B.Jacobs/PAPERS/.

[27] Pattie Maes, 'Concepts and experiments in computational reflection', in *OOPSLA'87*, pp. 147–155. ACM Press, (1987).

[28] Humberto R. Maturana and Francisco J. Varela, *Autopoeisis and Cognition: the realization of the living*, D. Reidel, 1980.

[29] Olivier Michel, Jean-Pierre Banâtre, Pascal Fradet, and Jean-Louis Giavitto, 'Challenging questions for the rationale of non-classical programming languages', *International Journal of Unconventional Computing*, **2**(4), 337–347, (2006).

[30] Robin Milner, *A Calculus of Communicating Systems*, Springer, 1980.

[31] Robin Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[32] Robin Milner, *Communicating and Mobile Systems: the* π*-Calculus*, Cambridge University Press, 1999.

[33] Robin Milner, *The Space and Motion of Communicating Agents*, Cambridge University Press, 2009.

[34] E. Moulay and M. Baguelin, 'Meta-dynamical adaptive systems and their application to a fractal algorithm and a biological model', *Physica D*, **207**, 7990, (2005).

[35] NIST. The NIST reference on constants, units, and uncertainty, 2011. http://physics.nist.gov/cuu/Constants/.

[36] NPL. What is the most accurate measurement known?, 2010. http://www.npl.co.uk/reference/faqs/what-is-the-most-accurate-measurement-known-(faq-quantum).

[37] Gheorghe Păun, 'Computing with membranes', *Journal of Computer and System Sciences*, **61**(1), 108–143, (2000).

[38] Przemyslaw Prusinkiewicz and Aristid Lindenmayer, *The Algorithmic Beauty of Plants*, Springer, 1990.

[39] Robert Rosen, *Life Itself: a comprehensive enquiry into the nature, origin, and fabrication of life*, Columbia University Press, 1991.

[40] Davide Sangiorgi, 'On the origins of bisimulation and coinduction', *ACM Transactions on Programming Languages and Systems*, **31**(4), 15:1–15:41, (2009).

[41] Susan Stepney, 'Nonclassical computation: a dynamical systems perspective', in *Handbook of Natural Computing, volume II*, eds., Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, chapter 52, Springer, (2011).

[42] Susan Stepney and Tim Hoverd, 'Reflecting on open-ended evolution', in *ECAL 2011, Paris, France, August 2011*, pp. 781–788. MIT Press, (2011).

[43] Susan Stepney, Fiona Polack, and Heather Turner, 'Engineering emergence', in *ICECCS 2006: 11th IEEE International Conference on Engineering of Complex Computer Systems, Stanford, CA, USA, August 2006*, pp. 89–97. IEEE, (2006).

[44] Steven H. Strogatz, *Nonlinear Dynamics and Chaos*, Westview Press, 1994.

[45] Kohji Tomita, Satoshi Murata, and Haruhisa Kurokawa, 'Self-description for construction and computation on graph-rewriting automata', *Artificial Life*, **13**(4), 383–396, (2007).

[46] Alan M. Turing, 'On computable numbers, with an application to the entscheidungsproblem', *Proceedings of the London Mathematical Society*, **s2-42**(1), 230–265, (1937).

[47] Alexander J. Yee and Shigeru Kondo. Round 2 ... 10 trillion digits of pi, 2011. http://www.numberworld.org/misc_runs/pi-10t/details.html.