The International Journal of Parallel, Emergent and Distributed Systems,
Vol. 20, No. 1, March 2005, 5–19

Taylor & Francis
Taylor & Francis Group

Journeys in non-classical computation I: A grand challenge for computing research

SUSAN STEPNEY†*, SAMUEL L. BRAUNSTEIN†, JOHN A. CLARK†, ANDY TYRRELL†, ANDREW ADAMATZKY‡, ROBERT E. SMITH‡, TOM ADDIS¶, COLIN JOHNSON§, JONATHAN TIMMIS§, PETER WELCH§, ROBIN MILNER‖ and DEREK PARTRIDGE#

†University of York, UK
‡University of the West of England, UK
¶University of Portsmouth, UK
§University of Kent, UK
‖University of Cambridge, UK
#University of Exeter, UK

1. The challenge

A *gateway event* [35] is a change to a system that leads to the possibility of huge increases in kinds and levels of complexity. It opens up a whole new kind of phase space to the system's dynamics. Gateway events during evolution of life on earth include the appearance of eukaryotes (organisms with a cell nucleus), an oxygen atmosphere, multi-cellular organisms and grass. Gateway events during the development of mathematics include each invention of a new class of numbers (negative, irrational, imaginary, . . .), and dropping Euclid's parallel postulate.

A gateway event produces a profound and fundamental change to the system: Once through the gateway, life is never the same again. We are currently poised on the threshold of a significant gateway event in computation: That of breaking free from many of our current "classical computational" assumptions. Our Grand Challenge for computer science is

> to journey through the gateway event obtained by breaking our current classical computational assumptions, and thereby develop a mature science of Non-Classical Computation

2. Journeys versus goals

To travel hopefully is a better thing than to arrive.
– Robert Louis Stevenson, "El Dorado", 1878.

*Corresponding author. Email: susan@cs.york.ac.uk

The International Journal of Parallel, Emergent and Distributed Systems
ISSN 1744-5760 print/ISSN 1744-5779 online © 2005 Taylor & Francis Ltd
http://www.tandf.co.uk/journals
DOI: 10.1080/17445760500033291

Many Grand Challenges are cast in terms of *goals*, of end points: "Achieving the goal, before this decade is out, of landing a man on the moon and returning him safely to earth" [50], mapping the human genome, proving whether $P = NP$ or not. We believe that a goal is not the best metaphor to use for our particular Grand Challenge, however, and prefer that of a *journey*.

The metaphor of a journey emphasises the importance of the entire process, rather than emphasising the end point. In the 17th and 18th centuries it was traditional for certain sections of "Polite Society" to go on "a Grand Tour of Europe", spending several years broadening their horizons: The experience of the entire journey was important. And in the Journey of Life, death is certainly not the goal! Indeed, an open journey, passing through gateway events, exploring new lands with ever expanding horizons, need not have an end point.

A journey of a thousand miles begins with a single step.

– Lao Tzu, *Tao Te Ching*, Chapter 64, ~600 B.C.

Journeys and goals have rather different properties. A goal is a fixed target, and influences the route taken to it. With an open journey of exploration, however, it is not possible to predict what will happen: The purpose of the journey is discovery, and the discoveries along the journey suggest new directions to take. One can suggest starting steps, and some intermediate *way points*, but not the detailed progress, and certainly not the end result.

Thinking of the Non-Classical Computation Challenge in terms of a journey, or rather several journeys, of exploration, we suggest some early way points that appear sensible to aim for. But we emphasise that these *are* early points, that we spy today as we peer through the gateway. As the community's journey progresses, new way points will heave into view, and we can alter our course to encounter these as appropriate.

The Road goes ever on and on.

– J. R. R. Tolkien, *The Lord of the Rings*, 1954.

3. Six classical paradigms to disbelieve before breakfast

Classical computing is an extraordinary success story. However, there is a growing appreciation that it encompasses an extremely small subset of all computational possibilities.

In many avenues of life, we create unnecessary limitations. Perhaps the most invidious of these are the implicit assumptions we make. We need to distinguish *this has to be the case* from the merely *this has always been the case*. Discoveries may emerge when what was considered an instance of the former is found to be an instance of the latter. For example, dropping Euclid's parallel postulate gave rise to the whole field of non-Euclidean geometry, arguably paving the way for General Relativity. We wish to encourage similar revolts against the assumptions of classical computing. So below we identify several paradigms that seem to define classical computing, but that may not necessarily be true in all computing paradigms, and we encourage the community to drop, invert, or otherwise perturb these paradigms in whatever ways seem interesting. Our brochure of reality-based journeys is a start.

Many computational approaches seek inspiration in reality (mainly biology and physics), or seek to exploit features of reality. These *reality-based computing* approaches hold great promise. Often, nature does it better, or at the very least differently and interestingly. Examining how the real world solves its computational problems provides inspirations for

novel algorithms (such as genetic algorithms or artificial immune systems), for novel views of what constitutes a computation (such as complex adaptive systems, and self-organising networks) and for novel computational paradigms (such as quantum computing).

There is a gulf between the maturity of classical computing and that of the emerging non-classical paradigms. For classical computing, intellectual investment over many years is turning craft into science. To fully exploit emerging non-classical computational approaches we must seek for them such rigour and engineering discipline as is possible. What that science will look like is currently unclear, and the Grand Challenge encourages exploration.

Here we outline some assumptions of classical computation, and ways researchers in different fields are challenging them. In later sections we discuss alternatives in more detail. (Some of the categories arguably overlap.)

It ain't necessarily so.

– George Gershwin, *Porgy and Bess*, 1934

3.1 The Turing paradigm

Classical physics: Information can be can be freely copied, information is local, states have particular values. *Rather*, at the quantum level information cannot be cloned, entanglement implies non-locality, and states may exist in superpositions.

Atomicity: Computation is discrete in time and space; there is a before state, an after state and an operation that transforms the former into the latter. *Rather*, the underlying implementation substrate realises intermediate physical states.

Infinite resources: Turing machines have infinite tape state, and zero power consumption. *Rather*, resources are always constrained.

Substrate as implementation detail: The machine is logical, not physical. *Rather*, a physical implementation of one form or another is always required, and the particular choice has consequences.

Universality is a good thing: One size of digital computer, one size of algorithm, fits all problems. *Rather*, a choice of implementation to match the problem, or hybrid solutions, can give more effective results.

Closed and ergodic systems: The state space can be pre-determined. *Rather*, the progress of the computation opens up new regions of state space in a contingent manner.

3.2 The von Neumann paradigm

Sequential program execution. *Rather*, parallel implementations already exist.

Fetch-execute-store model of program execution. *Rather*, other architectures already exist, for example, neural nets, FPGAs.

The static program: the program stays put and the data comes to it. *Rather*, the data could stay put and the processing rove over it.

3.3 The output paradigm

A program is a black box: It is an oracle abstracted away from any internal structure. *Rather*, the trajectory taken by a computation can be as interesting, or more interesting, than the final result.

A program has a single well-defined output channel. *Rather*, we can choose to observe other aspects of the physical system as it executes.

A program is a mathematical function: Logically equivalent systems are indistinguishable. *Rather*, correlations of multiple outputs from different executions, or different systems, may be of interest.

3.4 The algorithmic paradigm

A program maps the initial input to the final output, ignoring the external world while it executes. *Rather*, many systems are ongoing adaptive processes, with inputs provided over time, whose values depend on interaction with the open unpredictable environment; identical inputs may provide different outputs, as the system learns and adapts to its history of interactions; there is no prespecified endpoint.

Randomness is noise is bad: Most computer science is deterministic. *Rather*, nature-inspired processes, in which randomness or chaos is essential, are known to work well.

The computer can be switched on and off: Computations are bounded in time, outside which the computer does not need to be active. *Rather*, the computer may engage in a continuous interactive dialogue, with users and other computers.

3.5 The refinement paradigm

Incremental transformational steps move a specification to an implementation that realises that specification. *Rather*, there may be a discontinuity between specification and implementation, for example, bio-inspired recognisers.

Binary is good: Answers are crisp yes/no, true/false, and provably correct. *Rather*, probabilistic, approximate and fuzzy solutions can be just as useful, and more efficient.

A specification exists, either before the development and forms its basis, or at least after the development. *Rather*, the specification may be an emergent and changing property of the system, as the history of interaction with the environment grows.

Emergence is undesired, because the specification captures everything required, and the refinement process is top-down. *Rather*, as systems grow more complex, this refinement paradigm is infeasible, and emergent properties become an important means of engineering desired behaviour.

3.6 The "computer as artefact" paradigm

Computation is performed by artefacts: Computation is not part of the real world. *Rather*, in some cases, nature "just does it", for example, optical Fourier transforms.

The hardware exists unchanged throughout the computation. *Rather*, new hardware can appear as the computation proceeds, for example, by the addition of new resources. Also, hardware can be "consumed", for example, a chemical computer consuming its initial reagents. In the extreme, nanites will construct the computer as part of the computation, and disassemble it at the end.

The computer must be on to work. *Rather*, recent quantum computation results [47] suggest that you do not even need to "run" the computer to get a result!

Doubtless there are other classical paradigms that we accept almost without question. They too can be fruitfully disbelieved.

4. The Real World: Breaking the Turing paradigm

4.1 Real World as its own computer

The universe does not need to calculate, it just does it. We can take the *computational stance*, and view many physical, chemical and biological processes *as if* they were computations: The Principle of Least Action "computes" the shortest path for light and bodies in free fall; water "computes" its own level; evolution "computes" fitter organisms; DNA and morphogenesis "computes" phenotypes; the immune system "computes" antigen recognition.

This natural computation can be more effective than a digital simulation. Gravitational stellar clusters do not "slow down" if more stars are added, despite the problem appearing to us to be $O(n^2)$. And as Feynman noted [31], the real world performs quantum mechanical computations exponentially faster than can classical simulations.

4.2 Real World as our computer

Taking the computational stance, we may exploit the way the world works to perform "computations" for us. We set up the situation so that the natural behaviour of the real world gives the desired result.

There are various forms of real world sorting and searching, for example. Centrifuges exploit differences in density to separate mixtures of substances, a form of *gravitational sorting*. Vapours of a boiling mixture are richer in the components that have lower boiling points (and the residual mixture is richer in those that have higher boiling points); distillation exploits this to give a form of *thermal sorting*. Chromatography provides chemical means of separation. Ferro-magnetic objects can be separated out from other junk by using industrial-strength magnets. Optics can be exploited to determine Fourier transforms.

Maggots perform the "computation" of eating dead flesh: Historically, maggots were used to clean wounds, that is, to perform their computation in a context to benefit us. More recently, bacterial metabolisms have been altered to perform the "computation" of cleaning up pollution.

Access control computations abound. Suitably constructed shape is used to calculate whether the key inserted in a tumbler lock is the correct one. Physical interlocks are exploited for safety and practical reasons across many industries: For example, it is impossible to insert a nozzle from a leaded petrol pump into the fuel tank of a unleaded petrol car.

4.3 Real World as analogue computer

We may exploit the real world in more indirect ways. The "computations" of the "real world as our computer" are very direct. Often we are concerned with more abstract questions. Sometimes the physical world can be harnessed to provide results that we need: We may be

able to set up the situation so that there is an *analogy* between the computation performed by the real world, and the result we want.

There is an age-old mechanism for finding the longest stick of spaghetti in an unruly pile, exploiting the physics of gravity and rigidity: We can use this to sort by setting up an analogy between spaghetti strand length and the quantity of interest. Mercury and alcohol thermometers use a physical means of computing temperature by fluid expansion: The analogy is between the length of the fluid column and the temperature. Millikan's calculation of the charge on an electron exploits relationships between velocity of falling oil drops, viscosity of air, the charge on those drops and the strength of surrounding electric fields.

Classical computing already exploits physics at the level of electron movements. But there are other ways of exploiting nature.

Analogue computing itself exploits the properties of electrical circuits as analogues of differential equations.

DNA computing [4] encodes problems and solution as sequences of bases (strands) and seeks to exploit mechanisms such as strand splitting, recombination and reproduction to perform calculations of interest. This can result in vast parallelism, of the order of 10^{20} strands.

Quantum computing [71] presents one of the most exciting developments for computer science in recent times, breaking out of the classical Turing paradigm. As its name suggests, it is based on quantum physics, and can perform computations that cannot be *effectively* implemented on a classical Turing machine.† It exploits interference, many worlds, entanglement and non-locality. Newer work still is further breaking out of the binary mind-set, with multiple-valued "qudits", and continuous variables. Research in quantum computing is mushrooming, and it is apparent that we are not yet in position to fully exploit the possibilities it offers. If only small quantum computers were to prove practical then uses could still be found for simulating various quantum phenomena. However, if larger computers prove possible we will find ourselves unprepared.

- Why are there so few distinct quantum algorithms? How can new ones be found?
- How do we discover new a quantum algorithms to solve a given problem? How do we use existing algorithms to solve new problems? How can we find the best algorithms to use given limited computational resources? More generally. . ..
  - What would a discipline of quantum software engineering look like? (See later for more detail.)
- How can quantum computers be harnessed most effectively as part of a hybrid computational approach?

4.4 Real World as inspiration

Many important techniques in computer science have resulted from observing the real world. *Meta-heuristic search* techniques have drawn inspiration from physics (simulated annealing), evolution (genetic algorithms [36,68], genetic programming [7,53]), neurology

†Analogue (as in continuous) computing also breaks the Turing paradigm. But the real world is neither analogue nor classically discrete; it is quantum. So analogue computing might be dismissed as of theoretical interest only. However, the same dismissal might then be made of classically discrete (classical) computation! (The real world is also relativistic, but that paradigm has not been embraced by computation theory, yet.)

(artificial neural networks [11,52,67,83]), immunology (artificial immune systems [25]), plant growth (L-systems [81]), social networks (ant colony optimisation [12]) and other domains.

These have all proved remarkably successful, or look highly promising, yet the science underpinning their use comes nowhere near matching the science of classical computing. Given a raft of nature-inspired techniques we would like to get from problem to solution efficiently and effectively, and we would like to reason about the performance of the resulting systems. But this falls outside the classical refinement paradigm.

- What would a science of non-classical refinement look like? A science would allow us, for example, to reason confidently about the behaviour of neural networks in critical applications, to derive highly effective systems targeted at highly limited resources.

In the virtual worlds inside the computer, we are no longer constrained by the laws of nature. Our simulations can go beyond the precise way the real world works. For example, we can introduce novel evolutionary operators to our genetic algorithms, novel kinds of neurons to our neural nets, and even, as we come to understand the embracing concepts, novel kinds of complex adaptive systems themselves. The real world is our inspiration, not a restriction.

- How can we use nature inspired computation to build "better than reality" systems? What are the computational limits to what we can simulate?
- What is the best you can do given many components, each with highly restricted memory and processing ability?

5. Massive parallelism: Breaking the von Neumann paradigm

Parallel processing (Cellular Automata [94], etc) and other non-classical architectures break out of the sequential, von Neumann, paradigm. (The fact that the sequential paradigm is named after von Neumann should not be taken to imply that von Neumann himself was an advocate of purely sequential computation; indeed, he was also one of the early pioneers of CAs [70].)

Under the classical paradigm assumptions, any parallel computation can be serialised, yet parallelism has its advantages.

Real-time response to the environment. The environment evolves at its own speed, and a single processor might not be able to keep pace. (Possibly the ultimate example of this will be the use of vast numbers of nanotechnological assemblers (*nanites*) to build macroscopic artefacts. A single nanite would take too long, by very many orders of magnitude.)

Better mapping of the computation to the problem structure. The real world is intrinsically parallel, and serialisation of its interactions to map the computational structure can be hard. Parallelism also permits colocation of each processor and the part of the environment with which it interacts most. It then permits colocation of the software: Software agents can roam around the distributed system looking for the data of interest, and meeting other agents in a context-dependent manner.

And once the classical paradigm assumptions are challenged, we can see that serialisation is not necessarily equivalent.

Fault tolerance. Computation requires physical implementation, and that implementation might fail. A parallel implementation can be engineered to continue working even though some of its subset processors have failed. A sequential implementation has only the one processor.

Interference/interaction between devices. Computation requires physical implementation, and those implementations have extra-logical properties, such as power consumption, or electromagnetic emissions, which may be interpreted as computations in their own right (see later). These properties may interfere when the devices are running in parallel, leading to effects not present in a serialised implementation. (Possibly the ultimate example of this is the exponentially large state space provided by the superposed parallel qubits in a quantum computer.)

The use of massive parallelism introduces new problems. The main one is the requirement for *decentralised control*. It is just not possible to have a single centralised source exercising precise control over vast numbers of heterogeneous devices (this is merely a covert attempt to serialise the system). Part of this problem is tackled by the sister *Grand Challenges in Ubiquitous Systems*, and part is addressed in the later section on *open processes*.

6. In the eye of the beholder: Breaking the output paradigm

The classical paradigm of program execution is that an abstract computation processes an input to produce an output. This input–output mapping is a logical property of the computation, and is all that is important: No intermediate states are of interest, the computation is independent of physical realisation, and different instances of the computation yield precisely the same results.

Computation, however, is in the eye of the beholder. Algorithms are implemented by physical devices; intermediate states exist, physical changes happen in the world, different devices are distinguishable. Any information that can be observed in this physical world may be used to enrich the perceived computation [19].

6.1 Logical trajectory observations

An executing algorithm follows a *trajectory* through the logical state space. (Caveat: This is a classical argument: Intermediate *quantum* computational states may be in principle unobservable.) Typically, this trajectory is not observed (except possibly during debugging). This is shockingly wasteful: Such logical information can be a computational resource in its own right. For example, during certain types of heuristic search the trajectory followed can give more information about a sought solution than the final "result" of the search itself.

- How can logical observations made during execution be used to give useful information?

6.2 Physical trajectory observations

An executing algorithm is accompanied by physical changes to the world: For example, it consumes trajectory-dependent power as it progresses, and can take trajectory-dependent time to complete. Such physical resource consumption can be observed and exploited as a computational resource, for example, to deduce features of the logical trajectory. (For example, some recent attacks on smart cards have observed the power consumption

profile and data-dependent timing of internal operations to deduce secret key information [17].) Such physical observations provide a very powerful source of information, currently exploited mainly by attackers, but available for more general computational use.

- What physical observations are feasible, and correlated with logical trajectories?
- What new uses can be found for such physical observations?

6.3 Differential observations

An executing algorithm is realised in a physical device. Physical devices have physical characteristics that can change depending on environmental conditions such as temperature, and that differ subtly across *logically* identical devices. (Indeed, much of the rationale for digitisation is the removal of these differences.) So one can make observations not merely of the output of a single execution, but of a set of outputs from a family of executions, of multiple systems, of different but related systems. For example, if repeated executions of a search each get 90% of elements of a sought solution correct then repeated executions might be combined to give an overall solution.

- How can diversity of multiple computations be exploited?
- How should diversity be engineered? By repeated mutation of a source program? By embracing technologically diverse solution paradigms?

6.4 Higher-order observations

These are observations not of the program execution itself, but of the execution of the program used to design (the program used to design...) the program.

7. Open processes: Breaking the algorithmic paradigm

In the classical paradigm, the ultimate goal of a computation is reaching a fixed point: The final output, the "result" of the computation, after which we may switch off the computer. The majority of classical science is also based around the notion of *fixed-point equilibrium* and *ergodicity* (ergodicity is the property that the system has well defined spatial and temporal averages, because any state of the system will recur with non-zero probability).

Modern theories of physics consider systems that lack repetition and stability: They are *far from equilibrium* and *non-ergodic*. Perhaps the most obvious non-ergodic, far from equilibrium system is that of life itself, characterised by perpetual evolution (change). Most human problems are also best described in such terms; since computation is ultimately in service of such problems, the implications of non-ergodic, far from equilibrium physics must be considered in relationship to computing's future.

Consider the most basic of chaotic systems: The logistic process, parameterised by R.

$$x_{t+1} = Rx_t(1 - x_t)$$

The behaviours of various logistic processes as a function of R are shown in figure 1, where each point on the plot is a point on the attractor.

For values of $1 < R < 3$, these logistic processes have a fixed point attractor. For $R = 3$ they have an attractor of period two. As we raise R, the attractor becomes period four, period eight, etc. This *period doubling* continues as we raise R, and the values of R where each doubling occurs get closer together. For $R > 3.569945671\ldots$ the logistic process's attractor goes through an infinite number of values (except for a few "islands" or order, of attractors with multiples of odd periods). There is a *phase transition* from order (the region of period doubling) to chaos ("random" behaviour). The phase transition point at $R = 3.569945671\ldots$ is the so-called *edge of chaos* [61].

Consider a discretised process whose underlying (continuous) dynamics are those of the logistic equation. Imagine taking measurements from this process. Take very coarse measurements: Say the process outputs 1 if $x > 0.5$, and 0 otherwise; and take samples of length L bits. For a given L, construct an automaton that represents the process. So now the logistic processes generated by various values of R are being interpreted as a variety of automata: *Logistic machines*. It turns out that there is a clear phase transition (a peak in the machine size versus the entropy of the bit sequence) as we move from the period doubling region to the chaotic region [23].

At the phase transition, the machine size versus the length of the sequence L, *expands without bound*. That is, at the edge of chaos, the logistic machine requires an infinite memory machine for accurate representation. There is a leap in the level of intrinsic computation going on in the logistic machine at the edge of chaos. (In terms of the Chomsky hierarchy, the machine has gone from the level of regular grammars to the level of context-free grammars.)

At the edge of chaos, we can add new resources (computational or physical) to get results that are neither redundant (as they are in the structured period doubling regime) nor random (as in the chaotic regime). Within the classical paradigm, such conditions would be anathema, indicating unceasing variety that never yields "the solution". But in life-like

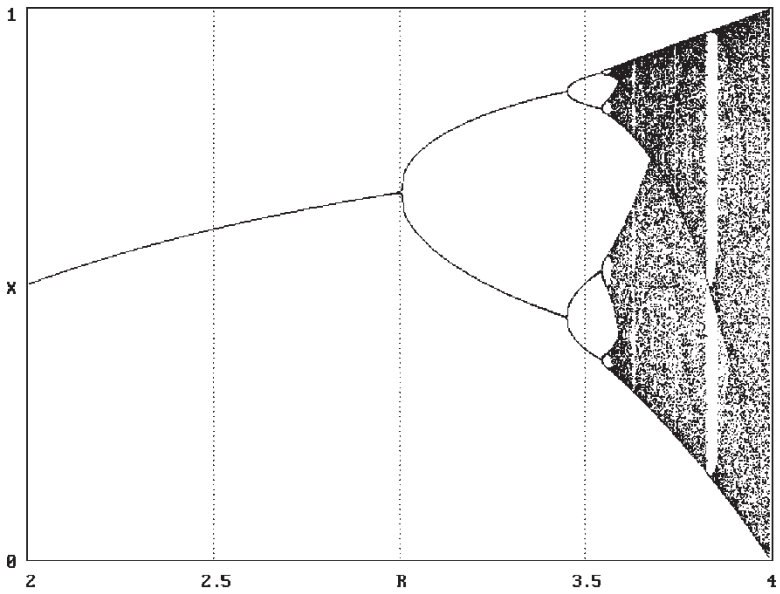

Figure 1. Points on the attractors of various logistic processes, versus the parameter R.

systems, there is simultaneously sustained order, and useful innovation. In this setting, emergence of the unforeseen is a desirable property, rather than disruptive noise.

Some computational approaches attempt to exploit the biological paradigm: Cellular automata, evolutionary computation, recurrent networks (autocatalytic, neural, genomic, immune system, ecological webs, ...), social insect and agent-based systems, DNA-computing, and nanite-systems that build themselves. However, in most of these cases, the implementations of such systems have been locked into themselves, *closed*, unable to take on new matter or information, thus unable to truly exploit emergence.

We should consider *open systems*, systems where new resources, and new kinds of resources, can be added at any time, either by external agency, or by the actions of the system itself. These new resources can provide gateway events, that fundamentally alter the character of the system dynamics, by opening up new kinds of regions of phase space, and so allowing new possibilities. Computational systems are beginning to open themselves, to unceasing flows of information (if not so much to new matter). The openness arises, for example, through human interactivity as a continuing dialogue between user and machine [90], through unbounded networks, through robotic systems with energy autonomy. As computers become ubiquitous, the importance of *open systems physics* to understanding computation becomes critical. The solutions we expect from people are ongoing processes, and this should be our expectation from computers too.

8. A coherent revolutionary challenge, that also respects the past

Classical physics did not disappear when modern physics came along: Rather its restrictions and domains of applicability were made explicit.

Similarly, the various forms of non-classical computation will not supersede classical computation: They will augment and enrich it. And when a wide range of tools is available, we can pick the best one, or the best combination, for each job. For example, it might be that using a quantum algorithm to reduce a search space, and then a meta-heuristic search to explore that, is more effective than using either algorithm alone.

We would like

> to create a general flexible conceptual framework that allows effective and efficient exploitation of hybrid approaches, including classical and non-classical components

The journey is the important thing. At various points in journey-space researches will alight to mark their way, leaving behind diary entries to which they may return at a later date. In common parlance these intermediate recordings may be regarded as "achievements". Opportunities are manifold. We expect journeys relevant to the sub-disciplines to be articulated separately; several have already been prepared. These are given in the second part of this paper: *Journeys in Non-Classical Computation II: Initial journeys and waypoints*. Also relevant are the sister Ubiquitous Systems grand challenges.

It is important these separate journeys are not seen as independent explorations. Rather, their results and insights should provide valuable groundwork for the overarching challenge

> to produce a fully mature science of all forms of computation, that unifies the classical and non-classical paradigms

9. The Grand Challenge Criteria

The Grand Challenge Journey in Non-Classical Computation has been drawn up in response to the UK Computing Research Council's call. UKCRC posed several criteria that a Grand Challenge should meet. Here we show how our Challenge meets these criteria.

It arises from scientific curiosity about the foundation, the nature or the limits of a scientific discipline. It arises from questioning the assumptions of the classical paradigms, and aims at the creation of a new science.

It gives scope for engineering ambition to build something that has never been seen before. It aims to build a new science; the engineering opportunities will follow.

It will be obvious how far and when the challenge has been met (or not). It will never be met fully: It is an open journey, not a closed goal. The science will continue to mature, until itself overtaken by the next paradigm shift.

It has enthusiastic support from (almost) the entire research community, even those who do not participate and do not benefit from it. No. However, in the best tradition of paradigm shifts, the change will occur.

> ***An important scientific innovation rarely makes its way by gradually winning over and converting its opponents: It rarely happens that Saul becomes Paul. What does happen is that the opponents gradually die out, and that the growing generation is familiarised with the ideas from the beginning.***
>
> – Max Planck, *Scientific Autobiography*, 1949

It has international scope: Participation would increase the research profile of a nation. This is a new fundamental area of computer science.

It is generally comprehensible, and captures the imagination of the general public, as well as the esteem of scientists in other disciplines. Much popular literature already exists in several of these areas, written by scientists in other disciplines (quantum computing, complexity, nanotech, ...), and so they and the general public are arguably already ahead of the CS community!

It was formulated long ago, and still stands. Its seeds have been around for a long time, but it has only recently become of obvious importance.

It promises to go beyond what is initially possible, and requires development of understanding, techniques and tools unknown at the start of the project. The structure of the Challenge mirrors the journey suggested by this criterion.

It calls for planned co-operation among identified research teams and communities. It is a multi-disciplinary Challenge, with contributions needed from a range of research specialities.

It encourages and benefits from competition among individuals and teams, with clear criteria on who is winning, or who has won. There need not be a single "winner". Diversity of solutions should be encouraged to be applicable to a range of application domains. Winners may emerge in particular application domains, as the strengths of the various techniques become clear.

It decomposes into identified intermediate research goals, whose achievement brings scientific or economic benefit, even if the project as a whole fails. There are several components to the Challenge that can be explored in parallel.

It will lead to radical paradigm shift, breaking free from the dead hand of legacy. Non-classical computing *is* a radical paradigm shift!

It is not likely to be met simply from commercially motivated evolutionary advance. Applications might be supported by industry, but it is unlikely that the development of the underlying science would be.

10. Initial journeys and waypoints

We have collected several suggested journeys that could be brought under the umbrella of Non-Classical Computation. It is assumed that these journeys would be conducted not in isolation, but in the context of the overall challenge, informing it, and being informed by it. The currently identified journeys are:

- Non-Classical Philosophy—Socially Sensitive Computing
- Non-Classical Physics—Quantum Software Engineering
- Non-Classical Refinement—Approximate Computation
- Computing in non-linear media—reaction-diffusion and excitable processors
- Artificial Immune Systems
- Non-Classical Interactivity—Open Dynamical Networks
- Non-Classical Architectures—Evolving Hardware
- Non-Classical Architectures—Molecular Nano-technology
- Non-von Architectures—Through the Concur-rency Gateway

These initial journeys are expanded on in the second part of this paper: *Journeys in Non-Classical Computation II: Initial journeys and waypoints*.

References and further reading

[1] Editorial article, 2002, *Nature Immunology*, **3**(10), 883, October.
[2] Andrew Adamatzky, 2001, *Computing in Nonlinear Media and Automata Collectives* (IoP).
[3] Andrew Adamatzky (Ed.), 2002, *Collision-Based Computing* (Springer).
[4] Adleman, Leonard M., 1994, Molecular computation of solutions to combinatorial problems, *Science*, **266**, 1021–1024, November.
[5] Thomas Back, Fogel, David B., Zbigniew Michalewicz (Eds.), 2000, *Evolutionary Computation 1: Basic Algorithms and Operators* (IoP).
[6] Per Bak, 1997, *How Nature Works: The Science of Self-Organized Criticality* (OUP).
[7] Wolfgang Banzhaf, Peter Nordin, Keller, Robert E. and Francone, Frank D., 1998, *Genetic Programming, An Introduction: On the Automatic Evolution of Computer Programs and its Applications* (Morgan Kaufmann).
[8] Albert-Laszlo Barabasi, 2002, *Linked: The New Science of Networks* (Perseus).
[9] Berry, G. and Boudol, G., 1992, The chemical abstract machine, *Theoretical Computer Science*, **96**, 217–248.
[10] Hugues Bersini and Varela, Francisco J., 1991, Hints for adaptive problem solving gleaned from immune networks. In: H.P. Schwefel and H. Mühlenbein (Eds.) *Parallel Problem Solving from Nature* (Springer).
[11] Bishop, Christopher M., 1995, *Neural Networks for Pattern Recognition* (OUP).
[12] Bonabeau, Eric W., Marco Dorigo and Guy Theraulaz, 1999, *Swarm Intelligence: From Natural to Artificial Systems* (OUP).
[13] Bradley, Daryl W. and Tyrrell, Andy M., 2000, Hardware fault tolerance: An immunological approach. *Proc IEEE Conf on System, Man, and Cybernetics*.
[14] Brookes, S.D., Hoare, C.A.R. and Roscoe, A.W., 1984, A theory of communicating sequential processes, *Journal of the ACM*, **31**, 560–699.
[15] Calude, Cristian S. and Gheorghe Paun, 2001, *Computing with Cells and Atoms* (Taylor & Francis).
[16] Cardelli, L. and Gordon, A., 2000, Mobile ambients, *Theoretical Computer Science*, **240**, 177–213.
[17] Suresh Chari, Jutla, Charanjit S., Rao, Josyula R. and Pankaj Rohatgi, 2003, Power analysis: Attacks and countermeasures. In: Annabelle McIver and Carroll Morgan (Eds.) *Programming Methodology* (Springer).
[18] Bastine Chopard and Michel Droz, 1998, *Cellular Automata Modeling of Physical Systems* (CUP).

[19] Clark, John A., Susan Stepney and Howard Chivers, 2004, *Breaking the model: Finalisation and A Taxonomy of Security Attacks*, Technical Report YCS-2004-371 (University of York).
[20] Clarke, E.M., Emerson, E.A. and Sistla, A.P., 1986, Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM ToPLaS*, **8**(2), 244–263.
[21] Cleaveland, R., Parrow, J. and Steffen, B., 1993, The concurrency workbench: A semantics based tool for the verification of concurrent systems, *ACM ToPLaS*, **15**, 36–72.
[22] Corne, David W., Marco Dorigo, Fred Glover (Eds.), 1999, *New Ideas in Optimization* (McGraw Hill).
[23] Crutchfield, J.P., 1994, The calculi of emergence: Computation, dynamics, and induction, *Physica D*, **75**, 11–54.
[24] Dipankar Dasgupta (Ed.), 1999, *Artificial Immune Systems and their Applications* (Springer).
[25] de Castro, Leandro N. and Jonathan Timmis, 2002, *Artificial Immune Systems: A New Computational Intelligence Approach* (Springer).
[26] de Castro, Leandro N. and von Zuben, Fernando J., 2000, An evolutionary immune network for data clustering. *SBRN'00, Brazil* (IEEE), pp. 84–89.
[27] Marianne Delorme, Jacques Mazoyer (Eds.), 1999, *Cellular Automata: A Parallel Model* (Kluwer).
[28] Drexler, K. Eric., 1986, *Engines of Creation: The Coming Era of Nanotechnology* (Doubleday).
[29] Drexler, K. Eric, 1992, *Nanosystems: Molecular Machinery, Manufacturing and Computation* (Wiley).
[30] Farmer, J. Doyne, Packard, Norman H. and Perelson, Alan S., 1986, The immune system, adaptation, and machine learning, *Physica D*, **22**, 187–204.
[31] Feynman, Richard P., 1982, Simulating physics with computers, *International Journal of Theoretical Physics*, **21**(6/7).
[32] Floyd, R.W., 1967, Assigning meanings to programs, *Mathematical Aspects of Computer Science, Proceedings Sympoium in Applied Mathematics 19*, 19–32, AMS.
[33] Stephanie Forrest (Ed.), 1991, *Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and Computing Networks* (MIT Press).
[34] Stephanie Forrest, Perelson, Alan S., Lawrence Allen and Rajesh Cherukuri, 1994, Self-nonself discrimination in a computer. *Symposium on Research in Security and Privacy* (IEEE), pp. 202–212.
[35] Murray Gell-Mann, 1994, *The Quark and the Jaguar* (Abacus).
[36] Goldberg, David E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley).
[37] Gordon, M.J.C., 1987, HOL: A proof generating system for higher-order logic. *VLSI Specification, Verification and Synthesis* (Kluwer).
[38] Prabhat Hajela, Jun Sun Yoo. Immune Network Modelling in Design Optimization. In [22]
[39] Emma Hart, Peter Ross. The Evolution and Analysis of a Potential Antibody Library for Use in Job Shop Scheduling. In [22]
[40] Hoare, C.A.R., 1971, An axiomatic basis for computer programming, *CACM*, **4**(1), 39–45.
[41] Hoare, C.A.R., 1985, *Communicating Sequential Processes* (Prentice Hall).
[42] Holland, John H., 1995, *Hidden Order: How Adaptation Builds Complexity* (Addison-Wesley).
[43] Holland, John H., 1998, *Emergence: From Chaos to Order* (OUP).
[44] Yoshitero Ishida, 1996, Distributed and autonomous sensing based on immune network. *Proc Artificial Life and Robotics, Beppu* (AAAI Press).
[45] Henrik Jeldtoft Jensen, 1998, *Self-Organized Criticality: Emergent Complex Behaviour in Physical and Biological Systems* (CUP).
[46] Jerne, Niels K., 1974, Towards a network theory of the immune system, *Annals of Immunology*, **125C**, 373–389.
[47] Richard Jozsa, 1991, Characterising classes of functions computable by quantum parallelism, *Proceedings of the Royal Society of London. A*, 435.
[48] Kauffman, Stuart A., 1993, *The Origins of Order: Self-Organization and Selection in Evolution* (OUP).
[49] Scott Kelso, J.A., 1995, *Dynamic Patterns: The Self-Organization of Brain and Behavior* (MIT Press).
[50] Kennedy, John F., 1961, Announcement to the US Congress, 25 May.
[51] Kephart, Jeffrey O., 1994, A biologically inspired immune system for computers. In: Rodney A. Brooks and Pattie Maes (Eds.) *Artificial Life IV* (MIT Press).
[52] Teuvo Kohonen, 1988, *Self-Organization and Associative Memory* (Springer).
[53] Koza, John R., 1992, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press).
[54] Koza, John R., 1994, *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT Press).
[55] Koza, John R., Bennett III, Forrest H., David Andre and Keane, Martin A., 1999, *Genetic Programming III: Darwinian Invention and Problem Solving* (Morgan Kaufmann).
[56] Kozen, D., 1983, Results on the propositional mu-calculus, *Theoretical Computer Science*, **27**, 333–354.
[57] George Lakoff, 1986, *Women, Fire, and Dangerous Things* (University of Chicago Press).
[58] George Lakoff and Mark Johnson, 1980, *Metaphors We Live By* (University of Chicago Press).
[59] Leslie Lamport, 1994, The temporal logic of actions, *ACM ToPLaS*, **16**(3), 872–923.
[60] Landweber, L.F. and Winfree, E. (Eds.), 2002, *Evolution as Computation* (Springer).
[61] Christopher G. Langton. Computation at the Edge of Chaos: Phase transitions and emergent computation. In [33]
[62] Langton, Christopher G. (Ed.), 1995, *Artificial Life: An Overview* (MIT Press).
[63] Mandelbrot, Benoit B., 1977, *The Fractal Geometry of Nature* (Freeman).

[64] Robin Milner, 1980, *A Calculus of Communicating Systems*, LNCS 92 (Springer).
[65] Robin Milner, 1999, *Communicating and Mobile Systems: The π -Calculus* (CUP).
[66] Robin Milner, Parrow, J. and Walker, D., 1992, A calculus of mobile processes, *Information and Computation*, **100**(1), 1–77.
[67] Minsky, Marvin L. and Papert, Seymour A., 1988, *Perceptrons* (MIT Press).
[68] Melanie Mitchell, 1996, *An Introduction to Genetic Algorithms* (MIT Press).
[69] Mark Neal and Jonathan Timmis, 2003, Timidity: A useful emotional mechanism for robot control? *Informatica: Special Issue on Perception and Emotion Based Reasoning*.
[70] John von Neumann, 1966, In: A.W. Burks (Ed.) *Theory of Self-Reproducing Automata* (University of Illinois Press).
[71] Nielsen, Michael A. and Chuang, Isaac L., 2000, *Quantum Computation and Quantum Information* (CUP).
[72] Mihaela Oprea and Stephanie Forrest, Simulated evolution of antibody gene libraries under pathogen selection. *Systems, Man and Cybernetics* (IEEE).
[73] Derek Partridge, 1995, On the difficulty of really considering a radical novelty, *Minds and Machines*, **5**(3), 391–410.
[74] Derek Partridge, 2000, Non-programmed computation, *CACM*, **43**, 293–301.
[75] Derek Partridge, Bailey, T.C., Everson, R.M., Hernandes, A., Krzanowski, W.J., Fieldsend, J.E. and Schetinin, V., 2004, A Bayesian computer, http://www.cs.york.ac.uk/nature/gc7/partridge.pdf.
[76] Gheorghe Paun, 2002, *Membrane Computing: An Introduction* (Springer).
[77] Heinz-Otto Peitgen and Richter, Peter H., 1986, *The Beauty of Fractals: Images of Complex Dynamical Systems* (Springer).
[78] Petri, C.A., *Kommunikation Mit Automaten*, PhD Thesis, Technical report, Institut fur Instrumentelle Mathemematik, Bonn.
[79] Pnueli, A., 1977, The temporal logic of programs. *Proceedings of FOCS* (IEEE), pp. 46–77.
[80] Pratt, V.R., 1976, Semantical considerations on floyd-hoare logic. *Proc. 17th Symp. Foundations of Computer Science* (IEEE), pp. 109–121.
[81] Przemyslaw Prusinkiewicz and Aristid Lindenmayer, 1990, *The Algorithmic Beauty of Plants* (Springer).
[82] Reynolds, J.C., 1974, Towards a theory of type structure. *Proc. Paris Symposium on Programming*, LNCS 16, (Springer), pp. 408–425.
[83] Rumelhart, David E. and McClelland, James L., 1986, *Parallel Distributed Processing* (MIT Press).
[84] Scott, D.S. and Strachey, C., 1971, Towards a mathematical semantics for computer languages. *Proc. Symposia on Computers and Automata, Microwave Research Institute Symposia 21*, pp. 19–46.
[85] Tanya Sienko, Andrew Adamatzky, Rambidi, Nicholas G., Michael Conrad (Eds.), 2003, *Molecular Computing* (MIT Press).
[86] Derek J. Smith, Stephanie Forrest, David H. Ackley, Alan S. Perelson. Modeling the effects of prior infection on vaccine efficacy. In [24]
[87] Susan Stepney, 2003, Critical critical systems. *Formal Aspects of Security, FASeC'02*, LNCS 2629 (Springer).
[88] Tommaso Toffoli and Margolus, Norman H., 1985, *Cellular Automata Machines* (MIT Press).
[89] Watts, Duncan J., 1999, *Small Worlds: The Dynamics of Networks Between Order and Randomness* (Princeton University Press).
[90] Peter Wegner, 1997, Why interaction is more powerful than algorithms, *CACM*, **40**(5).
[91] Welch, P.H., *et al.*, 2004, Concurrency Research Group, www.cs.kent.ac.uk/research/groups/crg/.
[92] Wittgenstein, L., 1921, *Tractatus Logico-Philosophicus*.
[93] Wittgenstein, L., 1953, *Philosophical Investigations* (Blackwells).
[94] Stephen Wolfram, 1994, *Cellular Automata and Complexity: Collected Papers* (Addison-Wesley).
[95] Andrew Wuensche and Mike Lesser, 1992, *The Global Dynamics of Cellular Automata* (Addison-Wesley).