Non-Classical Computation: a Dynamical Systems Perspective

Susan Stepney
Department of Computer Science, University of York, UK

1 Introduction

In this chapter we investigate computation from a dynamical systems perspective.

A dynamical system is described in terms of its abstract *state space*, the system's current state within its state space, and a rule that determines its motion through its state space. In a classical computational system, that rule is given explicitly by the computer program; in a physical system, that rule is the underlying physical law governing the behaviour of the system. So a dynamical systems approach to computation allows us to take a unified view of computation in classical discrete systems and in systems performing non-classical computation. In particular, it gives a route to a computational interpretation of physical embodied systems exploiting the natural dynamics of their material substrates.

We start with *autonomous* (closed) dynamical systems: those whose dynamics is not an explicit function of time, in particular, those with no inputs from an external environment. We begin with computationally conventional discrete systems, examining their computational abilities from a dynamical systems perspective. The aim here is both to introduce the necessary dynamical systems concepts, and to demonstrate how classical computation can be viewed from this perspective. We then move on to continuous dynamical systems, such as those inherent in the complex dynamics of matter, and show how these too can be interpreted computationally, and see how the material embodiment can give such computation "for free", without the need to explicitly implement the dynamics.

We next broaden the outlook to *open* (non-autonomous) dynamical systems, where the dynamics is a function of time, in the form of inputs from an external environment, and which may be in a closely coupled feedback loop with that environment.

We finally look at *constructive*, or developmental, dynamical systems, where the structure of the state space is changing during the computation. This includes vari-

ous growth processes, again investigated from a computational dynamical systems perspective.

These later sections are less developed than for the autonomous cases, as the theory is less mature (or even non-existent); however these are the more interesting computational domains, as they move us into the arena of considering biological and other natural systems as computational, open, developmental, dynamical systems.

2 Autonomous dynamical systems

Consider a dynamical system with N degrees of freedom; it has an abstract state space $\mathcal{X}^N$. Its state can be defined by N state variables $x_i \in \mathcal{X}$, or, equivalently, by an ND state vector $\mathbf{x} \in \mathcal{X}^N$ (for example, $\mathbf{x}$ may be a vector of binary bits, or a vector of continuous variables such as position and momentum). The state vector $\mathbf{x}_t$ defines the value of the system state at a given time.

The deterministic dynamics is given by a function $f : \mathcal{X}^N \to \mathcal{X}^N$, which defines how a state vector $\mathbf{x}$ changes with time, that is, it defines the *trajectory* that the system takes through its state space. So the dynamics associates a vector with each point in the state space, defining how that point evolves under the dynamics[1]. If f is not itself an explicit function of time, then the system is *autonomous*.

In general, dynamical systems theory is not concerned with details of individual trajectories, but rather with the qualitative behaviours of sets of trajectories. For example, consider a set of states occupying some initial volume of the state space: as these states evolve under the dynamics, how does the volume change? We are mostly interested here in *dissipative systems*, where the volume contracts to *attractors* (regions of state space that attract trajectories), and we interpret such attractors from a computational perspective. This contracting behaviour is a property of systems that dissipate energy or information. (Closed non-dissipative dynamical systems, on the other hand, have no attractor structure.)

2.1 Discrete space, discrete time dynamical systems

We start by considering finite discrete spaces (finite number of finite dimensions), with discrete time dynamics $t \in \mathcal{N}$ (where $\mathcal{N}$ is the set of natural numbers).

[1] In conventional use, the meaning of this vector is unfortunately different in the discrete and continuous time cases. In the discrete time case (§2.1, §2.2), $\mathbf{x}_{t+1} = f(\mathbf{x}_t)$, and so the vector is the next state; in the continuous time case (§2.3), $\dot{\mathbf{x}} = f(\mathbf{x})$, and so the vector is the derivative, pointing towards the next state an infinitesimal time later: $\mathbf{x}_{t+dt} = \mathbf{x}_t + f(\mathbf{x}_t)\,dt$. It would be possible to have a uniform meaning, by redefining the discrete case vector to be the difference in states, with $\mathbf{x}_{t+1} = \mathbf{x}_t + f(\mathbf{x}_t)$ (and an implicit $\Delta t = 1$). However, here we follow the conventional, and inconsistent, use.

We take $\mathcal{X} = \mathcal{S}$, some set with finite cardinality $|\mathcal{S}| \in \mathcal{N}$ (typically $\mathcal{S}$ will be the boolean set $\mathcal{B}$, but it is not restricted to this). For an N dimensional system, the state is defined by N state variables $s_i \in \mathcal{S}$, and the state space $\mathcal{S}^N$ comprises $|\mathcal{S}|^N$ distinct discrete states. (When $\mathcal{S} = \mathcal{B}$, these states fall on the vertices of an N-dimensional hypercube.) Let the state vector be $\mathbf{s} \in \mathcal{S}^N$.

The dynamics of a particular system are determined by its particular transition function $f : \mathcal{S}^N \to \mathcal{S}^N$, with $\mathbf{s}_{t+1} = f(\mathbf{s}_t)$. There are $(|\mathcal{S}|^N)^{|\mathcal{S}|^N}$ such functions f.

Given a particular state $\mathbf{s}_0 \in \mathcal{S}^N$, its trajectory under f is a sequence of states $\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_t, \ldots$. Eventually, because the state space is finite, a state that was met before will be met again: there exists a k such that $\mathbf{s}_k = \mathbf{s}_{k+p}$, for some p. Since the dynamics is deterministic, the trajectory will then recur: for all $i \geq k . \mathbf{s}_i = \mathbf{s}_{i+p}$. The system has entered an *attractor*, with cycle length or period p. States not on an attractor are called *transient*.

Given a trajectory $\ldots, \mathbf{s}_t, \mathbf{s}_{t+1}, \ldots$, then $\mathbf{s}_t$ is the *pre-image* of $\mathbf{s}_{t+1}$ in this trajectory, and $\mathbf{s}_{t+1}$ is the *successor* of $\mathbf{s}_t$. Every state has precisely one successor (because the dynamics is deterministic). It may have zero, one, or more pre-images (trajectories may merge); if it has zero pre-images, it is a *Garden of Eden* state.

The set of all states $\mathbf{s}_i$ whose trajectories lead to the same attractor forms the *basin of attraction* of that attractor. The total state space is partitioned into these basins: every state is in precisely one basin. Note that there is no necessary correlation between the volume of the basin (the proportion of state space it occupies, and hence the probability that a state chosen at random will be in it) and the length of the attractor that it leads to.

The *microstate* of the system is which particular $\mathbf{s} \in \mathcal{S}$ it is in. The *macrostate* is which particular attractor (or basin of attraction if the microstate is currently a transient state) the system is in.

For a given dynamics f, there is a minimum of one attractor basin (all states are in the same attractor basin, for example, the zero function), and a maximum of $|\mathcal{S}|^N$ (the identity function where every state forms its own single-state attractor basin). There is a minimum transient length of 0 (all states on some attractor cycle, for example, the increment modulo 2^N function, interpreting the binary encoded state $\mathcal{B}^N$ as a number), and a maximum transient length of $|\mathcal{S}|^N - 1$ (for example, the decrement and halt on zero function). There is a minimum number of Garden of Eden states of 0 (all states on some attractor cycle), and a maximum number of Garden of Eden states of $|\mathcal{S}|^N - 1$ (for example, the zero function). Non-trivial dissipative computational systems rarely lie at any of these extremes, however. (Note that reversible, non-dissipative, systems have no Garden of Eden states, and no merging trajectories.)

2.1.1 Visualising the attractor field

Visualising the basins of attraction can help in understanding some aspects of their dynamics. For small systems, the most common approach is to lay out the state transition graph to highlight the separate basins, their attractors, and their symmetries

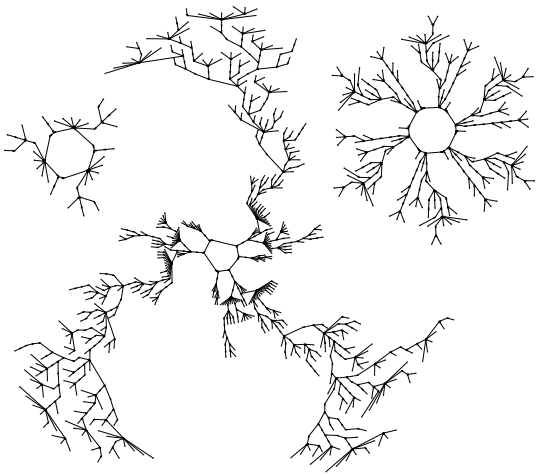

Fig. 1 Visualisation of part of the state transition graph for ECA rule 110 (see §2.1.5 on Elementary Cellular Automata), on the periodic lattice $N = 12$, showing three of the basins of attraction. Each node corresponds to a state $\mathbf{s}_i$; each edge corrsponds to a transition $\mathbf{s}_i \rightarrow \mathbf{s}_{i+1}$. The leaves of the graphs are Garden of Eden states; the attractor cycles can be seen at the centres of the basins.

(see figure 1). Wolfram [108, fig 9.1] used this approach in early work on cellular automata; Wuensche [110, 111] has developed special purpose layout software, and uses this approach consistently, to highlight aspects of the dynamics.

2.1.2 Computation

Given a finite system in initial microstate $\mathbf{s}_0$ (which may be considered to contain an encoding of any input data), the system follows its dynamics f until it reaches the relevant attractor. If this attractor has a cycle length of one, the system then stays in the single attractor state. For longer cycle lengths, the system perpetually repeats the cycle of states.

In terms of attractors. The *computation* performed by the system as it follows its dynamics f can be interpreted as the determination of which attractor basin it is in, by progressing to the attractor from its initial state $\mathbf{s}_0$.

The output of the computation may be the microstates, or some suitable projection thereof, of the discovered attractor cycle. (See, for example, §2.1.5, Example 1: the density classification task).

In terms of trajectories. Alternatively, the *computation* performed by the system as it follows its dynamics f can be considered to be (some projection of) the microstates it passes through along its trajectory, including both transient and attractor cycle states. (See, for example, §2.1.5, example 2: the Rule 30 PRNG).

Programming task. The programming task involves determining a dynamics f that leads to the required trajectories or attractor structure. (See, for example, section 2.1.5, Example 1: the density classification task.)

For feasible programs, the discovery of an attractor should be performed in polynomial time (implying polynomial length transients and attractor cycles). At the other complexity extreme, the dynamics should not be defined merely by a $|\mathcal{S}|^N$-entry lookup table (which would allow all computations to find the attractor in a single step); it should admit a "compressed" description. (See, for example, §2.1.5 on cellular automata, and §2.1.6 on random Boolean networks, which define the global dynamics f in terms of the composition of local dynamics ϕ_i.)

Implementation. Natural physical systems do not tend to directly implement a discrete dynamics, particularly one that has been designed to perform a specific task. However, any such dynamics can be implemented (or simulated) on a classical digital computer. Hence there are no implementation constraints imposed on the design of the dynamics f.

Inputs and outputs. The input is encoded into the initial state; the output is decoded from (a projection of) the resulting attractor state(s). It is important when analysing the complexity of the computation to take into account any "hidden" computation needed to encode the input, or to decode the output. This is particularly important if the computational interpretation is far removed from the dynamics, for example, if there is some kind of virtual machine present.

2.1.3 Virtual machine dynamics

In some cases a dynamics need to be accompanied by a very carefully chosen initial condition in order to implement the required computation. For example, when cellular automata are used to implement Turing Machines (TMs; see §2.1.5 on Universality), they are given a carefully chosen initial configuration that corresponds to the "program" of the TM, plus the "true" input corresponding to its initial tape. This requirement for an exquisitely tuned initial condition constrains the system to traverse only a very small part of its state space (certain basins of attraction are never explored; some transient trajectories are never taken). What is happening in these cases is that the underlying broader dynamics is being used to implement a "virtual machine" with its own dynamics confined to a small sub-space of the underlying system; this sub-space and its trajectories correspond to the computation of the virtual machine, and may possibly be implemented more directly. In the continuous case, this more direct implementation is what we want: the natural dynamics of the system, with no need for such highly tuned initial conditions, performs the desired computation.

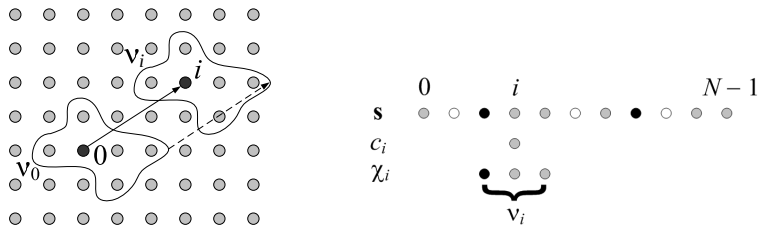

Fig. 2 CA neighbourhood. (a) A CA's regular neighbourhood, ν, illustrated in a 2D lattice. The neighbourhood of cell i is the image of the neighbourhood of cell 0, translated by i. (b) The state of the neighbourhood, illustrated in a 1D lattice. χ_i, the state of the neighbourhood of cell i, is the projection of the full state $\mathbf{s}$ onto the neighbourhood ν_i.

2.1.4 Infinite dimensional state spaces

When N is (countably) infinite, the dynamics of the system can change qualitatively.

In a finite dissipative system (which contains both transients and cycles), there must be Garden of Eden states, but this is no longer true in an infinite system: transient behaviour on the way to the attractor cycle need not have any starting Garden of Eden states (for example, the decrement and halt on zero function).

An infinite system need not have an attractor cycle: there is no guarantee that the system will reach a previously seen state (for example, the increment function). Even if there are attractor cycles, there may be states not in their basins (for example, the function **if** even **then** add 2 **else** halt).

A Turing Machine (TM) operates in a state space with a countably infinite (or, more precisely, finite but unbounded) number of dimensions. In dynamical systems terms, halting is reaching one of a number of particular attractor cycles of length 1, which describes the halting states. The output of the TM (the contents of its tape) is simply the microstate of the subspace representing the tape in this halting state. Hence there are potentially many attractors, one for each halting state with different tape contents, corresponding to different initial tape contents (different initial states $\mathbf{s}_0$).

The Halting Problem means that it is in general undecidable whether a given initial state $\mathbf{s}_0$ is in a halting basin, in some other ("looping") basin, or not in a basin at all.

2.1.5 Cellular automata

A finite cellular automaton (CA) comprises N *cells* laid out in a regular grid or lattice, usually arranged as an n-dimensional torus (n in common examples is typically 1 or 2). Each cell i at time t has a state value $c_{i,t} \in \mathbb{S}$.

Each cell has a *neighbourhood* of k cells, comprising itself and certain nearby cells in the grid. This neighbourhood is the same for all cells, in that ν_i, the neighbourhood of c_i, is ν_0, the neighbourhood of the origin c_0, translated by i (figure 2a).

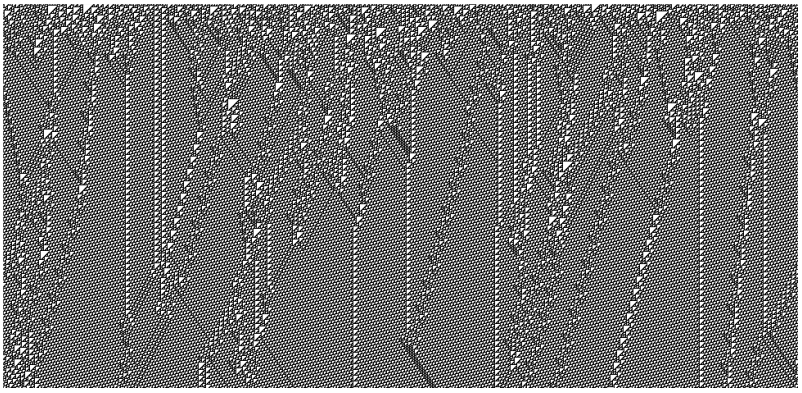

Fig. 3 Visualisation of the time evolution of ECA rule 110 (see §2.1.5 on Elementary Cellular Automata), with $N = 971$, with random initial state $\mathbf{s}_0$, over 468 timesteps. Each horizontal line shows the N-bit representation of the state $\mathbf{s}_t$; subsequent lines correspond to subsequent timesteps $\mathbf{s}_t, \mathbf{s}_{t+1}, \ldots$.

The state of cell c_i's neighbourhood ν_i at time t is $\chi_{i,t} \in \mathcal{S}^k$, a k-tuple of cell states that is the projection of the full state onto the neighbourhood ν_i (figure 2b).

The local state transition rule, or update rule, is $\phi : \mathcal{S}^k \to \mathcal{S}$. (There are $|\mathcal{S}|^{|\mathcal{S}|^k}$ such rules, some of which are related by symmetries. So, since typically $k \ll N$, CA rules capture only a small fraction of all the possible dynamics over an ND space.) These cells form an array of state transition machines. At each timestep, the state of each cell is updated in parallel, $c_{i,t+1} = \phi(\chi_{i,t})$.

The global dynamics f is determined by the local rule ϕ and the shape of the neighbourhood. This global behaviour from a given initial state is conventionally visualised in the 1D case by drawing the global state at time t as a line of cells (with colours corresponding to the local state), then drawing the state at $t+1$ directly below, and so on (see figure 3). Higher dimensional CAs are conventionally visualised as animations.

Figure 1 shows three basins of attraction of a small CA. There are many "repeated" basins (basins with identical topologies, over different specific states) due to the symmetries in a CA rule [73, 74]. Every rule over a finite space with periodic boundary conditions has "shift" symmetries from shifting the arbitrary origin; additionally, some rules also have reflectional and other symmetries.

Elementary Cellular Automata (ECAs). ECAs are 2-state ($\mathcal{S} = \mathcal{B}$) CAs, with the cells arranged in a 1D lattice, and with a neighbourhood size of 3 (comprising the cell and its immediate left and right neighbours). There are $2^{2^3} = 256$ distinct ECA rules, conventionally referred to by a base 10 number $0 \ldots 255$, representing the base 10 interpretation of the rule table bitstring. After reflection and inversion symmetries have been taken into account, there are 88 essentially distinct rules.

Wolfram's classification. Wolfram [105, 104] provides a qualitative characterisation of CAs, classifying their long term evolution into four classes:

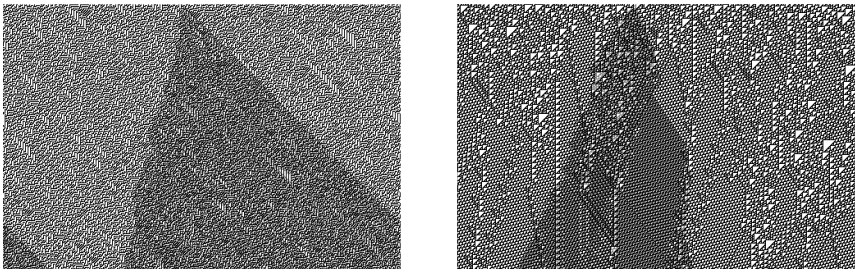

Fig. 4 Sensitive dependence on initial conditions. Each plot overlays the evolutions of two initial states differing in only one bit: the growing central dark region is different; the outer regions are the same. $N = 971$, random initial state $\mathbf{s}_0$, over 646 timesteps (a) ECA rule 45; (b) ECA rule 110

1. a unique homogeneous state, independent of the initial state (a single state attractor cycle); patterns disappear with time
2. a simple periodic pattern of states (short attractor cycles, length $\ll |\mathcal{S}|^N$); patterns become fixed
3. chaotic aperiodic pattern of states (long attractor cycles, length $O(|\mathcal{S}|^N)$, or no cycles in the infinite case); patterns grow indefinitely
4. complex localised long-lived structures; patterns grow and contract

Wolfram's classification scheme has been criticised for a variety of reasons, including the fact that even determination of quiescence (long term fixed patterns) is undecidable [18]. This is a recurring problem for any classification scheme: "CA behavior is so complex that almost any question about their long-term behavior is undecidable" [24]. See also [94]. Wolfram classes are nevertheless widely used in a qualitative manner to distinguish kinds of behaviours.

Class 3 and class 4 CAs demonstrate *sensitive dependence on initial conditions*: the effect of a minimal (one cell state) change to the initial condition propagates across the system, eventually resulting in a completely different dynamics (see figure 4). The 88 essentially distinct ECAs cover all 4 Wolfram classes of behaviour.

Universality. Even though CA rules capture only a small fraction of all the possible dynamics over an ND space, there are Computationally Universal CAs, that can emulate a TM. ECA rule 110 (figure 3) is universal [14], as is Conway's Game of Life 2D CA [10, 77]. The proof of universality in these cases involves constructing a virtual machine in the CAs, on which is implemented a TM (or equivalent). In other words, the computation is being performed by a carefully engineered initial condition, and a carefully engineered interpretation of the dynamical behaviour. So these systems explore only a small fraction of their full state space: that fraction that corresponds to an interpretation of the state of a TM.

Wolfram [105, §8] speculates that "class 4 cellular automata are characterized by the capability for universal computation" (in the case of infinite-dimensional CAs).

Example CA 1: the density classification task. The requirement is to design a local two-state 1D CA rule ϕ, independent of lattice size N (assume N odd for

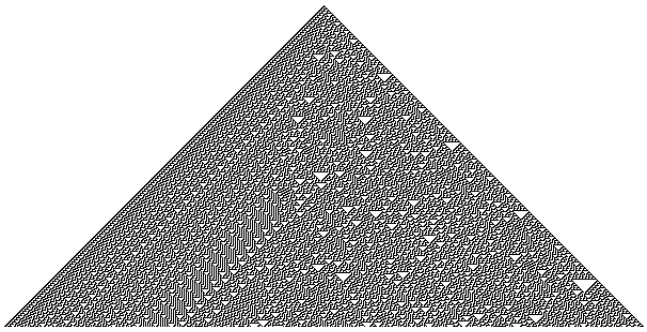

Fig. 5 ECA rule 30 (equation 1), initial state of a single non-zero cell

simplicity), such that the global dynamics f has the following properties: (i) the system has two attractors, each of cycle length 1, one being the all zeros state, one being the all ones state; (ii) any initial state $\mathbf{s}_0$ with more zeros than ones is in the basin of the all zeros state, and vice versa; (iii) the maximum transient length is at worst polynomial in n (ie the attractor is discovered in polynomial time). The computation determines which attractor basin the initial state is in, and hence, by inference, whether the initial state has more zeros than ones.

No two-state CA with a bounded neighbourhood size can solve this problem exactly on arbitrary lattice size N [49]. The best rules designed or evolved (for example, [25, 61, 109]) fail to correctly classify about 20% of states: that is, they define a dynamics where these states are in the "wrong" basin of attraction. Bossomaier et al [11] investigate the problem specifically in terms of the attractor basins.

Example CA 2: the rule 30 pseudo-random number generator. Wolfram [108] discusses ECA rule 30:

$$\phi_{30}(c_{i-1}, c_i, c_{i+1}) = c_{i-1} \text{ XOR } (c_i \text{ OR } c_{i+1}) \tag{1}$$

Starting from an initial state $\mathbf{s}_0$ with a single cell j "on", $c_{i,0} = (\textbf{if } i = j \textbf{ then } 1 \textbf{ else } 0)$, (figure 5), then the sequence of bits under this single on bit, $\tau = c_{j,0}, c_{j,1}, \ldots, c_{j,t}, \ldots$ forms a (pseudo-)random sequence. Wolfram [108] presents evidence that the cycle length of the attractor starting from a state with a single non-zero cell grows exponentially with CA lattice size N.

This sequence τ is a projection (onto the single boolean state variable c_j) of the trajectory from state $\mathbf{s}_0$ under the dynamics defined by rule 30.

2.1.6 Random Boolean Networks

A Random Boolean Network (RBN) comprises N *nodes*. Each node i at time t has a binary valued state, $c_{i,t} \in \mathcal{B}$. Each node has k inputs assigned randomly from k of the N nodes (an input may be from the node itself); the wiring pattern is fixed

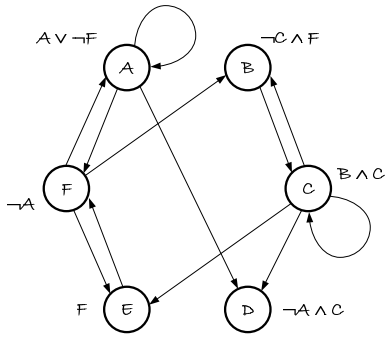

Fig. 6 An example RBN with $N = 6, k = 2$. Each node has $k = 2$ inputs; it can have any number of outputs. So the neighbourhood function is $\nu_A = (A,F), \nu_B = (C,F)$, etc. Each node combines its inputs by a random boolean function ϕ_i: that function might ignore one (or both) of the inputs.

throughout the lifetime of the network. This wiring defines the cell's neighbourhood, ν_i. See figure 6.

The state of cell i's neighbourhood at time t is $\chi_{i,t} \in \mathcal{B}^k$, a k-tuple of cell states that is the projection of the full state onto the neighbourhood ν_i.

Each node has its own randomly chosen local state transition rule, or update rule, $\phi_i : \mathcal{B}^k \to \mathcal{B}$. These cells form a network of state transition machines. At each timestep, the state of each cell is updated in parallel, $c_{i,t+1} = \phi_i(\chi_{i,t})$.

The global dynamics f is determined by the local rules ϕ_i and the connectivity pattern of the nodes ν_i. In contrast to the regularity of a CA, in an RBN each node has its own random neighbourhood connection pattern and its own random update rule. Like CAs, RBNs capture only a small fraction of all the possible dynamics over an ND space.

Kauffman [42, 43] investigates the properties of RBNs[2] as a function of connectivity k. Unlike CAs, RBNs tend not to have repeated basins, because of the random nature of the connections, and hence the relative lack of symmetries. Despite all the randomness, however, "such networks can exhibit powerfully ordered dynamics" [42], particularly when $k = 2$ (figure 7; table 1). Drossel [23] notes that subsequent computer simulation of much larger networks shows that "for larger N the apparent square-root law does not hold any more, but that the increase with system size is faster".

Kauffman identifies $k = 1$ as the "ordered" regime, with a very large number of short period attractors. Large k is the chaotic regime, with very long period attractors (compare Wolfram's class 3 behaviour). $k = 2$ occurs at a "phase transition"

[2] The wiring conditions given above are not stated explicitly in these references. However, in the $k = N$ case, Kauffman [43, p.192] states that "Since each element receives an input from all other elements, there is only one possible wiring diagram". This implies that multiple connections from a single node are not allowed (otherwise more wiring diagrams would be possible) whereas self connections are allowed (otherwise k would be restricted to a maximum value of $N-1$). Subsequent definitions (for example [23]) explicitly use the same conditions as given here.

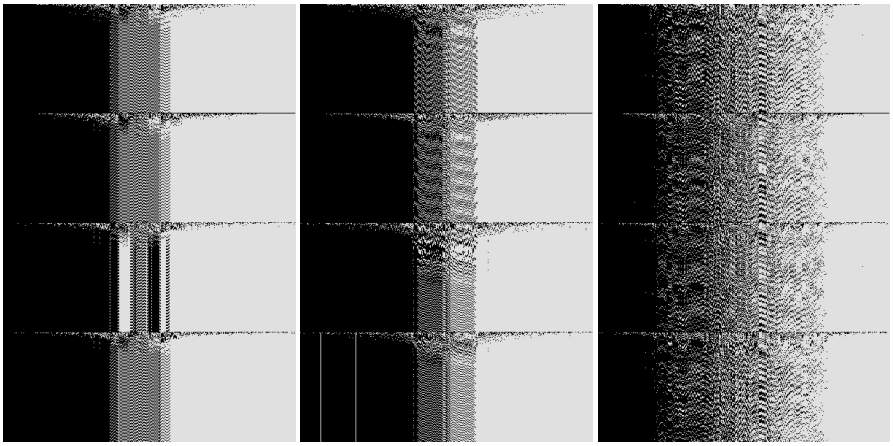

Fig. 7 Visualisation of the time evolution of three typical $k = 2$ RBNs, with $N = 400$, and initial condition all nodes "off"; after 150 timesteps all nodes are set to "on", then all nodes are randomised (50% "on", 50% "off") every further 150 timesteps, to explore other attractors. They exhibit ordered behaviour: short transients, and low period attractors. The visualisation scheme used here [91] orders the nodes to expose the *frozen core* [43, p.203] of nodes that do not change state on the attractor; this frozen core is well-preserved on different attractors.

k	attractor cycle length	# attractors
1	$O(\sqrt{N})$	$O(2^N)$
2	$O(\sqrt{N})$	$O(\sqrt{N})$
> 5	$O(2^N)$	$O(N)$

Table 1 Dynamics of RBNs for different k (adapted from [43, table 5.1])

[42], separating the ordered and chaotic regimes; it exhibits a moderate number of moderate period attractors.

Kauffman investigates RBNs as simplified models of gene regulatory networks (GRNs). He notes that "cell types are constrained and apparently stable recurrent patterns of gene expression", and interprets his RBN results as demonstrating that a "cell type corresponds to a state cycle attractor" [43, p.467] (in a $k = 2$ network).

Emergent macrostates of the dynamics, in addition to the attractor cycle length, are the number of nodes whose states change during a cycle, compared to the number that form the static *frozen core*. In the GRN interpretation, the frozen core would correspond to genes whose regulatory state was constant in a particular cell type, and the changing nodes to those genes whose regulatory state was cycling.

2.2 Continuous space, discrete time

We next consider continuous spaces, with $\mathcal{X} = \mathcal{R}$ (where $\mathcal{R}$ is the set of real numbers), with discrete time dynamics $t \in \mathcal{N}$. Let the state vector be $\mathbf{r} \in \mathcal{R}^N$. The dynamics of a particular system is determined by its particular transition function $f : \mathcal{R}^N \to \mathcal{R}^N$, with $\mathbf{r}_{t+1} = f(\mathbf{r}_t)$.

These systems are called *difference equations* or *iterated maps*. A discrete-time trajectory $\mathbf{r}_t, \mathbf{r}_{t+1}, \mathbf{r}_{t+2}, \ldots$ is also called an *orbit*.

The trajectories can display a range of behaviours, corresponding to a range of types of attractor, depending on the system. Trajectories may diverge ($|\mathbf{r}_t| \to \infty$); they may converge to a fixed point attractor ($\mathbf{r}_t \to \mathbf{r}^*$, where $f(\mathbf{r}^*) = \mathbf{r}^*$); they may converge to a periodic attractor; they may be chaotic, never repeating but still confined to a particular sub-region of the state space. So continuous space systems have chaotic behaviour that differs qualitatively from the "chaotic" behaviour of discrete space systems, since the finite discrete systems must eventually repeat and hence be periodic (although with exponentially long periods).

2.2.1 Parameterised families of systems

It is often convenient to consider a family of dynamics related by some parameter $p \in \mathcal{P}$, that is, $f(\mathbf{r}, p)$, and investigate how the dynamics of a system vary as a function of this parameter. The trajectories of such parameterised systems can display the whole range of behaviours corresponding to the range of types of attractor, depending on the value of the parameter. A small change to the parameter can make a fixed point move, or become unstable, or periodic, or disappear. It can move a system from period P to period $2P$ (*period doubling*), or from periodic to chaotic behaviour. The parameter values where these qualitative changes in the dynamics occur are called *bifurcation points*. As the parameter crosses the bifurcation point, the change in the dynamics can be continuous (smooth), or discontinuous (*catastrophic*).

Many systems exhibit a sequence of period doublings as the parameter changes. Subsequent doublings happen ever more rapidly (requiring ever smaller changes to the parameter), then the system moves into a chaotic regime. This is known as the *period doubling route to chaos*. Appearance of a period doubling cascade in a parameterised system is indication that chaos will ensue if the parameter changes further.

Another typical behaviour is the *intermittency route to chaos*. Here the parameter starts in a region with periodic dynamics; as it is changed, the periodic behaviour is broken by intermittent bursts of irregular behaviour. As the parameter changes further, there are more and more of these bursts, until the behaviour becomes completely chaotic. Hence a fully deterministic system may be apparently periodic, interrupted by what look like bursts of noise, where these bursts are simply part of the same overall dynamics of the system, and in need of no external explanation.

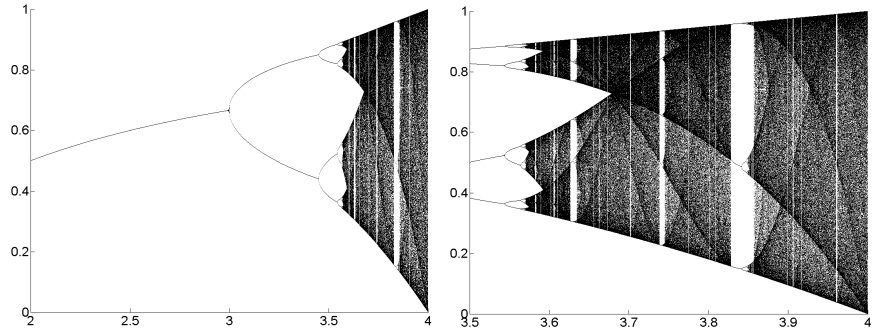

Fig. 8 Orbit diagram of the logistic map, (a) $2 \leq \lambda \leq 4$, showing the period doubling route to chaos; (b) $3.5 \leq \lambda \leq 4$, zooming in on the window of period 3.

2.2.2 Logistic map

The logistic map, a parameterised family of 1D iterated maps:

$$r_{t+1} = \lambda\, r_t (1 - r_t) \tag{2}$$

is a well-studied example of such a system (see figures 8 and 9). It is usually studied for $\lambda \in [0,4]$, since in this region its dynamics is confined to the unit interval: $r \in [0,1]$. It does have complex dynamics for other values of λ, but these are not constrained to the unit interval.

Its unintuitively rich, complex properties have been researched in detail, and in 1976 Robert May advised [53]:

> Not only in research, but also in the everyday world of politics and economics, we would all be better off if more people realised that simple nonlinear systems do not necessarily possess simple dynamical properties.

The attractor structure, visible in figure 8, is summarised in table 2 as a function of parameter λ. It has period doubling cascades to chaos as λ is increased. The first cascade leads to the onset of chaos at $\lambda = 3.56994\ 56718\ldots$. Within this chaotic region, there are windows of periodicity (such as the window of period 3), which then also period-double back to chaos. Given the existence of a period 3 cycle in a map, then every possible period can also be found in that map [50]. In the logistic map, there are windows of every period for some $3 < \lambda < 4$. Each of these periodic windows then period-doubles back to chaos as λ increases. The order in which these cascades occur itself has a complex structure, which can be calculated iteratively, in terms of symbolic sequences [57]; the lowest order sequences are shown in table 2.

Cycles of the same period can nevertheless have different kinds of behaviours: see, for example, figure 10.

The logistic map also exhibits the intermittency route to chaos. If the parameter λ falls in a window of periodic behaviour, and is then slowly *reduced*, intermittent behaviour is seen (for example, figure 9e), until fully chaotic behaviour is reached.

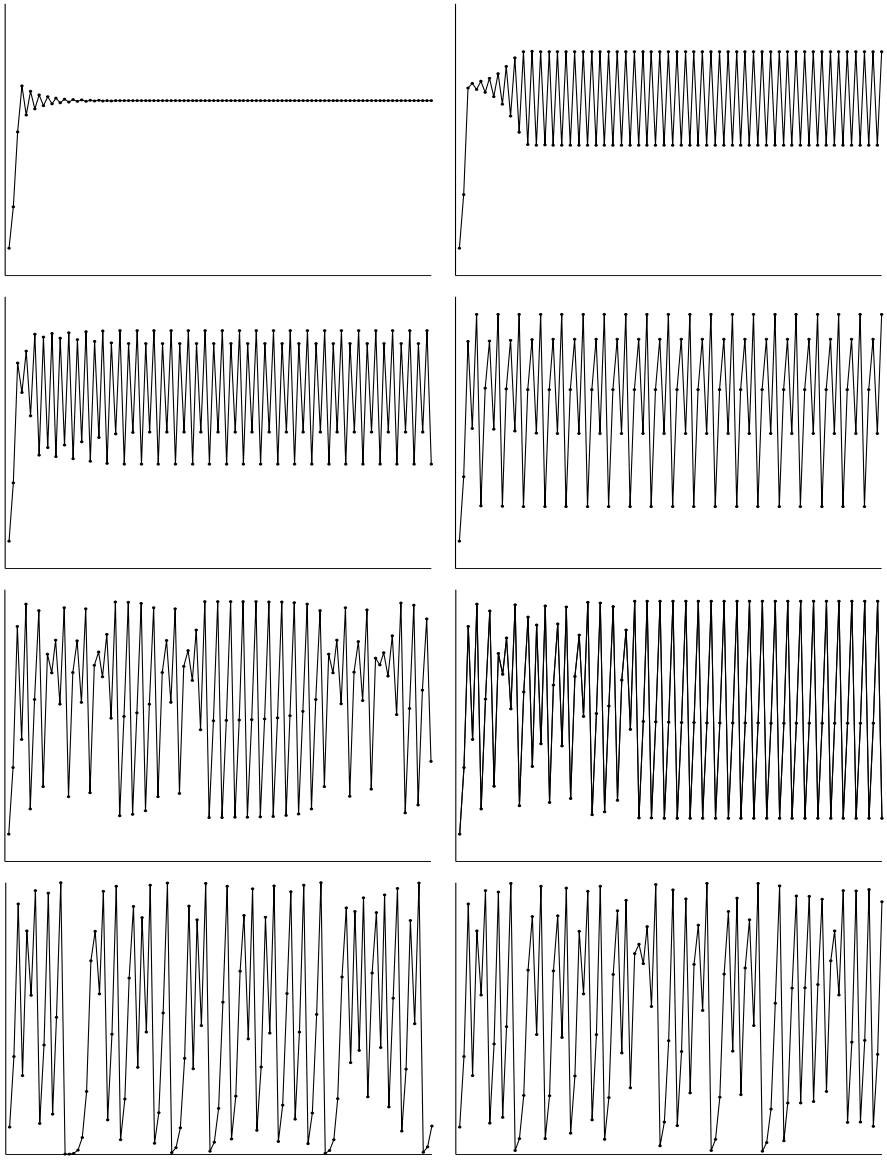

Fig. 9 Timeseries of the logistic map. 100 iterations, with $r_0 = 0.1$, for various λ.
Top row: (a) $\lambda = 2.8$, period 1; (b) $\lambda = 3.3$, period 2.
Second row: (c) $\lambda = 3.5$, period 4; (d) $\lambda = 3.74$, period 5.
Third row: (e) $\lambda = 3.828$, chaos before period 3: period 3 behaviour is interleaved with intermittent bursts of chaotic behaviour; (f) $\lambda = 3.829$, period 3.
Bottom row: (g) $\lambda = 4$, chaos; (h) $\lambda = 4$, with $r_0 = 0.10001$, demonstrating sensitive dependence on initial conditions.

lambda	dynamics	Sloane number [86]
$(0,1)$	fixed point $r^* = 0$	
$(1,3)$	fixed point, function of λ (eg, $\lambda = 2, r^* = 0.5$	
3	first period doubling, to period 2	
$1+\sqrt{6} = 3.449\ldots$	second period doubling, to period 4	
3.54409 03595…	third period doubling, to period 8	A086181
3.56440 72660…	fourth period doubling, to period 16	A091517
3.56994 56718…	end of first cascade; first onset of chaos	A098587
3.62655 31616…	appearance of period 6; period 6 cascade	A118453
3.70164 07641…	1st period 7 cascade	A118746
3.73817 23752…	appearance of period 5; 1st period 5 cascade	A118452
3.7741…	2nd period 7 cascade	
$1+2\sqrt{2} = 3.828427\ldots$	appearance of period 3; sole period 3 cascade	
3.841499…	period doubling, to period 6	
3.8860…	3rd period 7 cascade	
3.9055…	2nd period 5 cascade	
	4th period 7 cascade	
3.9375…	period 6 cascade	
	5th period 7 cascade	
3.9601…	period 4 cascade	
	6th period 7 cascade	
3.9777…	period 6 cascade	
	7th period 7 cascade	
3.9902…	3rd and last period 5 cascade	
	8th period 7 cascade	
	period 6 cascade	
	9th and last period 7 cascade	
4	fully chaotic	

Table 2 Dynamical structure of the logistic map as a function of parameter λ (some λ values taken from [86]); including order of occurrence of period doubling cascades, for all initial periods up to 7 (adapted from [29, table 2.2]). Higher initial periods cascades are interleaved with these.

Binary shift map. Consider the fully chaotic case, $\lambda = 4$. Changing variables, to $x = \frac{1}{\pi}\cos^{-1}(1-2r)$, yields the equation for the binary shift (Bernoulli) map:

$$x_{t+1} = 2x_t \bmod 1 \tag{3}$$

If x is expressed in base 2, each iteration of the map results in a left shift of the number (the multiplication by 2), and dropping any resulting leading 1 in front of the binary point (the mod 1). For example, if $x_t = 0.1110001\ldots$, then $x_{t+1} = 0.110001\ldots$, $x_{t+2} = 0.10001\ldots$, etc. If x_0 is rational, its binary expansion is periodic, and hence the iteration will be periodic; if x_0 is irrational, its binary expansion is non-periodic, and hence the iteration will be non-periodic. Separate values of x_0 that are the same up to their nth bit will initially have similar iterations, but will eventually diverge, until they are completely different at the nth iteration: this is a manifestation of sensitive dependence on initial conditions.

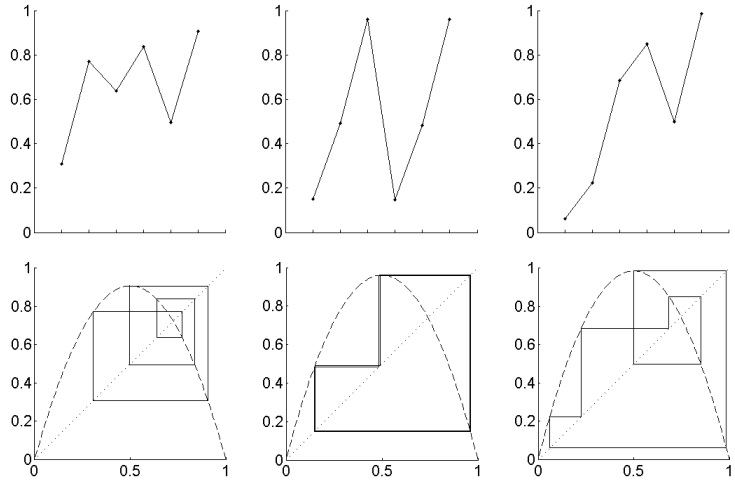

Fig. 10 Different classes of behaviour of three different period 6 cycles. Top shows the time series of one period; bottom shows the geometry of the behaviour in a cobweb diagram. (a) $\lambda = 3.6266$; (b) $\lambda = 3.8418$; (c) $\lambda = 3.93755$

Computational properties. Crutchfield [16] investigates the *statistical complexity* of the logistic map as a function of λ. The statistical complexity is essentially a measure of the size of a stochastic finite state machine that can predict the statistical properties of a system. He finds that the map has *low* statistical complexity both when the map is periodic and when it is chaotic (essentially random), and has *highest* statistical complexity at the onset of chaos (via the period doubling route, or the intermittent route). At this point there is a *phase transition* in the complexity of the machine needed to predict the behaviour, and these parameter values indicate the highest computational capacity.

Despite these observations, the majority of computational applications of the logistic map exploit its completely chaotic behaviour, with λ at or near 4, and use this behaviour to implement random number generators [41, 72, 99], encryption [46, 69], etc.

2.2.3 Coupled Map Lattices

Kaneko [37, 38, 39, 40] introduces one-dimensional Coupled Map Lattices (CMLs), where an array of N iterated maps are coupled together locally, with the following local dynamics ϕ:

$$r_{i,t+1} = \phi_0(r_{i,t}) + \frac{\varepsilon}{2}\left(\phi_c(r_{i-1,t}) - 2\phi_c(r_{i,t}) + \phi_c(r_{i+1,t})\right) \tag{4}$$

where ε is the coupling strength. Each element in the lattice evolves under the dynamics of the local process ϕ_0, with an additional interaction contribution $\varepsilon\phi_c$ from

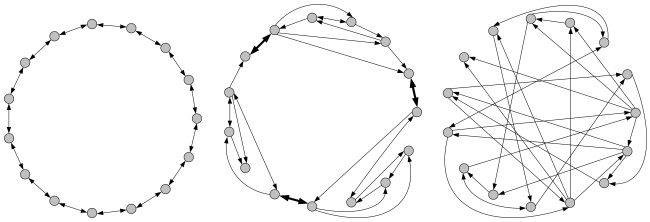

Fig. 11 Dimensionality versus topology. Examples for three $N = 5, k = 2$ networks with different topologies. (a) regular $N = 15$ CA-like structure arranged in a 1D spatial lattice (periodic boundary conditions); (b) three $N = 5, k = 2$ RBNs linked in a 1D spatial lattice (that is, random connections with the groups of 5 nodes, lattice links between the groups); (c) general $N = 15, k = 2$ RBN

its neighbours, whilst passing on a similar amount of its own to its neighbours (under periodic boundary conditions). The local state $r_i \in \mathcal{R}$ and local dynamics $\phi : \mathcal{R}^3 \to \mathcal{R}$ defines the global state $\mathbf{r} \in \mathcal{R}^N$ and global dynamics $f : \mathcal{R}^N \to \mathcal{R}^N$. The system is parameterised by the coupling strength ε, as well as by any parameters in ϕ.

An initial value of $r_{i,0} = \kappa$, where all the maps have the same initial value, is trivial, since all maps evolve in lock-step. Kaneko [39] investigates several cases, one of which is where ϕ_0 is the logistic map with λ in the period-3 window with cycle (r_1^*, r_2^*, r_3^*), with $r_{i \leq N/2,0} = r_1^*, r_{N/2<i,0} = r_2^*$, or with random $r_{i,0}$, and with $\phi_c = \phi_0$. The dynamics exhibits the period doublings, intermittencies, and chaos of the logistic map, and in addition exhibits spatial patterns and structures similar to 1D CAs. Crutchfield and Kaneko [17] examine the properties of this class of system in some detail.

Subsequent work generalises the approach to topologies other than 1D nearest-neighbour (2D, tree-structured, irregular, larger neighbourhoods), and allows the coupling to be asymmetric. In particular, Holden *et al* [34] provide a generic formalism, and investigate coupled map lattices in terms of their computational properties. A CML approach to the density classification problem has been evolved [7]. Open CMLs are also being exploited computationally (see §3.3.1).

2.2.4 Note on dimensionality and topology

Papers on CMLs tend to describe them as having "discrete time, discrete space, and continuous state" [40]. Here we describe them as "continuous space", because here we are talking purely about the *state space*.

A CML of N maps laid out in 1D line in (physical) space has an (abstract) state space of $\mathcal{R}^N$. The "dimensionality" of the layout in physical space (a 1D line of maps) is unrelated to the dimensionality of the state space (of ND, because there are N maps); if the same N maps were instead laid out in a 2D grid, or even a 27D grid, say, it would not affect the dimensionality of the state space.

Instead, this "dimensionality" is related the *topology* of the connections between the maps, and hence the potential information flow between the maps (figure 11).

Compare the information flow visible in a 1D (physical space) CA (figure 4) and that not visible in an RBN with a similar number of cells (figure 7). For this reason, it makes sense to talk of the (physical spatial) dimension of a CA (it has a regular local topology), but not of an RBN (it has an irregular graph topology).

When the physical spatial layout moves from discrete to continuous, the state space moves from being finite (discrete physical space, finite number of cells) or countably infinite (discrete physical space, countably infinite number of cells) to uncountably infinite (continuous physical space). See §2.3.4

2.2.5 Numerical errors and the Shadowing Lemma

Chaotic systems, such as the logistic map with $\lambda = 4$, display sensitive dependence on initial conditions: trajectories with nearby initial conditions diverge exponentially (compare figures 9g,h).

Such systems are usually studied by numerical simulations, which generate *pseudo-orbits*, because of numerical noise. Additionally, the real physical systems that these maps (and below, differential equations) model are themselves subject to noise during their execution, and during measurement, and so potentially execute pseudo-orbits (or pseudo-trajectories) with respect to the model systems. The question naturally arises: what is the relation of these pseudo-orbits to the model orbits? Are they representative of the modelled dynamics?

Fortunately the answer is (a qualified) "yes". The *Shadowing Lemma* states that, for certain classes of system, the pseudo-orbit *shadows* (stays close to) some true orbit of the (modelled) system for all time; this true orbit has a slightly different initial condition from the pseudo-orbit. This result has been extended to a wider class of systems, including those studied here, that the pseudo-orbit shadows some true orbit of the system for "a long time" [28].

However, are the true orbits of the model that are shadowed by these pseudo-orbits themselves representative of the underlying dynamics; that is, are they typical true orbits? This is harder to answer, and is clearly false for some particular cases. For example, consider the binary shift map (equation 3). For any limited precision binary arithmetic implementation, *all* pseudo-orbits converge to 0 when the number of iterations (shifts) exceeds the binary numerical precision. But this case appears to be an exception, because most numerical trajectories do not behave like this, and [33]:

> If otherwise reliable-looking pseudo-trajectories *are* atypical, they must be atypical in an extremely subtle way, because researchers have been making apparently reliable, self-consistent, peer-reviewed conclusions based on numerical simulations for decades.

So we continue here in assuming that the simulated pseudo-orbits, and the pseudo-trajectories of actual physical systems, are in general representative of the true dynamics defined by the equations.

2.3 Continuous space, continuous time

We now consider continuous spaces, with $\mathcal{X} = \mathcal{R}$, with continuous time dynamics $t \in \mathcal{R}$. Let the state vector be $\mathbf{r} \in \mathcal{R}^N$. The dynamics of a particular system is determined by its particular transition function $f : \mathcal{R}^N \to \mathcal{R}^N$, with $\dot{\mathbf{r}} = f(\mathbf{r})$. Hence the system is defined by a set of N coupled first order ordinary differential equations (ODEs).

We can recast higher order equations into this normal form by adding new variables. For example, consider the 1D equation for damped Simple Harmonic Motion:

$$\ddot{r} + \kappa \dot{r} + \omega^2 r = 0 \tag{5}$$

Let $r_1 = r, r_2 = \dot{r}$. Then, rearranging, we get the 2D normal form version:

$$\dot{r}_1 = r_2 \quad ; \quad \dot{r}_2 = \omega^2 r_1 - \kappa r_2 \tag{6}$$

Note how this normalisation takes the single state variable r, and results in two state variables r_1 (r, position) and r_2 ($\dot{r}$, velocity). In a continuous system, and particularly when the state variables are position $\mathbf{r}$ and momentum $m\dot{\mathbf{r}}$, the state space is also called the *phase space*.

Physical systems embody their own specific dynamics. If that dynamics can be controlled and exploited in a computational manner, it can be used to reduce the load on, or even replace, conventional classical digital control in embedded systems [89]. Understanding the dynamical behaviour of complex material systems from a computational perspective is also a necessary step along the way to understanding biological systems as information processing systems [90].

For a good overview of continuous dynamical systems from an embodied computational perspective, see Beer [9]. Abraham and Shaw [2] provide an excellent visual description of various concepts such as attractors and bifurcations. For more background, see a textbook such as that by Strogatz [93].

2.3.1 Kinds of attractor

In these continuous time systems, trajectories are continuous paths through a continuous state space. There are four distinct kinds of attractor that can occur.

Point attractors. A point attractor is a single point in state space that attracts the trajectories in its basin. An example is the equilibrium position and zero velocity that is the unique end state of damped simple harmonic motion (figure 12a).

Limit cycle attractors. A limit cycle is a closed loop trajectory that attracts nearby trajectories to it. See, for example, figure 12b.

Toroidal attractors. A toroidal attractor is a 2D surface in state space with periodic boundary conditions: shaped like a torus. Trajectories are confined to the surface of the torus. If the *winding number* (the number of times the trajectory loops around in one dimension whilst it performs one loop in the other dimension) is rational, the

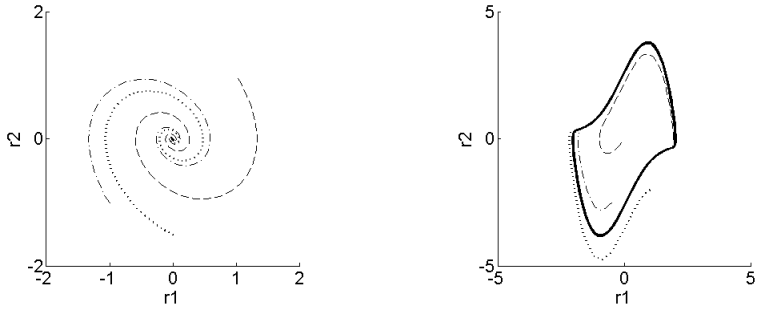

Fig. 12 Attractors and their transient trajectories: (a) point attractor: damped SHM, $\ddot{r}+\lambda\dot{r}+r=0$, with $\lambda=0.5$; (b) limit cycle attractor: van der Pol oscillator, $\ddot{r}+\lambda(r^2-1)\dot{r}+r=0$, with $\lambda=2$.

trajectory is periodic, otherwise it is *quasiperiodic*, and eventually covers essentially the entire surface of the torus.

Strange attractors. A strange attractor attracts trajectories to its region of state space, but within this region, nearby trajectories diverge exponentially: it exhibits sensitive dependence on initial conditions, and thus chaotic behaviour. This combination of attraction and divergence requires at least three dimensions in which to occur. The detailed structure of a strange attractor is usually fractal.

Example : Rössler strange attractor. The Rössler system is defined by:

$$\begin{aligned}\dot{r}_1 &= -r_2 - r_3 \\ \dot{r}_2 &= r_1 + a r_2 \\ \dot{r}_3 &= b + r_3(r_1 - c)\end{aligned} \tag{7}$$

It is a family of dynamical systems that displays a range of kinds of dynamics, some with strange attractor behaviour, some without. It exhibits the period doubling cascade route to chaos (figure 13a).

The Rössler strange attractor occurs when $a=0.2, b=0.2, c=5$ (figure 13b). It is the simplest strange attractor, with only one non-linear term.

Example : Lorenz strange attractor. The Lorenz strange attractor is defined by:

$$\begin{aligned}\dot{r}_1 &= 10(r_2 - r_1) \\ \dot{r}_2 &= 28 r_1 - r_2 - r_1 r_3 \\ \dot{r}_3 &= r_1 r_2 - 8 r_3/3\end{aligned} \tag{8}$$

(See figure 14.) It is a member of a family of dynamical systems that displays a range of kinds of dynamics, some with strange attractor behaviour, some without.

Sensitive dependence on initial conditions is popularly known as *The Butterfly Effect*. Lorenz suggests [51, p.14] that the name may have arisen from the title of a talk he gave in 1972, "Does the Flap of a Butterfly's Wings in Brazil Set Off a

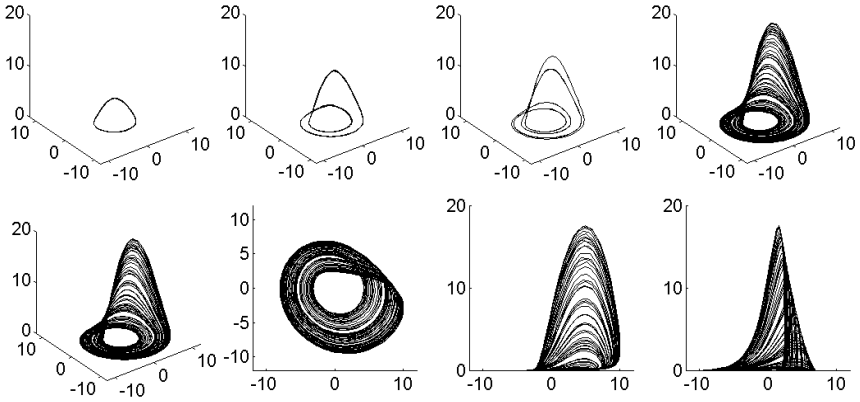

Fig. 13 Rössler system: (a) period doubling cascade to chaos, with $a = b = 0.2$, $c = 2.5, 3.5, 4.0, 5.0$; (b) Rössler strange attractor

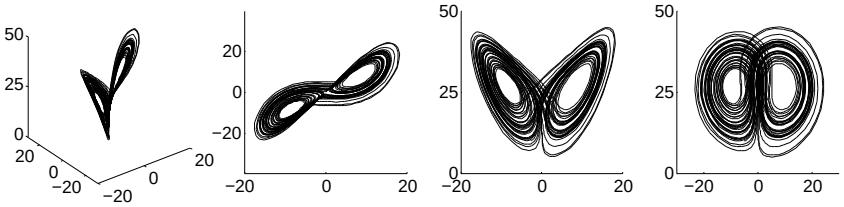

Fig. 14 Lorenz strange attractor

Tornado in Texas?", coupled with the butterfly-like shape of Lorenz attractor seen from some directions (figure 14).

Reconstructing the attractor. Given a physical continuous dynamical system with a high-dimensional state space (state vector $\mathbf{r}_t$), one can determine properties of its dynamics, given only scalar discrete time series observations (time series data $r_\tau, r_{2\tau}, \ldots, r_{n\tau}, \ldots$, where r_t is some scalar projection of the state vector $\mathbf{r}_t$). In particular, one can distinguish chaos (motion on a strange attractor) from noise.

The process of reconstructing the attractor from this data involves constructing a d-dimensional state vector $\hat{\mathbf{r}}_t$ from a sequence of time-lagged observations:

$$\hat{\mathbf{r}}_{n\tau} = (r_{n\tau}, r_{(n+k)\tau}, r_{(n+2k)\tau}, \ldots, r_{(n+(d-1)k)\tau}) \tag{9}$$

d should be $> 2d_a$, where d_a is the dimension of the system's attractor; d should also be as small as possible, to avoid fitting noise; k should be large enough that the attractor is sufficiently sampled, but not so large that the correlations are lost. Taken's *embedding theorem* then relates the invariants of motion, including attractor structure, of $\hat{\mathbf{r}}$ to those of $\mathbf{r}$. For more on this process, and other techniques for analysing chaotic systems, see [68].

Relationship to discrete state space attractors. Wolfram [105] draws a rough analogy between his class 1, 2, and 3 CAs (see §2.1.5) and point, limit cycle, and strange attractors respectively. He mentions that the class 4 CAs (the ones conjectured universal) "behave in a more complicated manner". This might be thought to imply that there is no continuous analogue of the discrete class 4 systems, and hence no universal computational properties in continuous matter. This is not so, as we see below (§2.3.4, §2.3.5).

Kauffman [42] calls RBNs whose attractor cycle length increases exponentially with n, "chaotic". He emphasises that this does not mean that flow on the attractor is divergent (it cannot be, in a discrete deterministic system); the state cycle is the analogue of a 1D limit cycle. However, there is an analogy: exponentially long cycles cover a lot of the state space before repeating (chaotic strange attractors never repeat), and "nearby" states (1 bit different) potentially *do* diverge (even possibly onto another attractor). However, in the discrete system, there is no direct analogue of "nearby states diverging exponentially, but staying on the same attractor", since there is usually no concept of distance between states in discrete dynamical systems, and if there were, successive hops through state space can be of any size: there is no simple "continuity" from which to diverge.

2.3.2 Computation in terms of attractors

We can interpret computation as finding which attractor basin the system is in, by following its trajectory to the relevant attractor. The output could be (some projection of) the computed attractor, including a subspace of the state space. Most instances of analogue computing fall in this domain.

The programming problem is in finding the relevant dynamics, now restricted to natural (albeit engineered) material system properties, which are not arbitrary. The aim is to minimise the engineering required to implement the desired dynamics, by exploiting the natural dynamics.

Continuous systems computing in this way can exhibit *robustness*. A small perturbation to the system might shift it a small distance from its attractor, but its subsequent trajectory will converge back to the attractor. There can be a degree of continuity in the attractor basins, such that a small perturbation tends to remain in the same basin, unlike the discrete case.

It is not necessary for a dynamical system to have a complex (chaotic, strange) dynamics in order to be interesting or useful. Kelso [44, p.53] makes this point eloquently:

> Some people say that point attractors are boring and nonbiological; others say that the only biological systems that contain point attractors are dead ones. That is sheer nonsense from a theoretic modeling point of view, as it ignores the crucial issue of what fixed points refer to. When I talk about fixed points here it will be in the context of collective variable dynamics of some biological system, not some analogy to mechanical springs or pendula.

That is, the dynamics, including the underlying attractor structure, is part of the specific model, in particular, what state variables are used to capture the real world

system. State variables can capture more sophisticated concepts than simple particle positions and momenta.

2.3.3 Continuous time logistic equation

The logistic growth equation is one of the simplest biologically-based non-linear ODEs. It is a simple model of population growth where there is exponential growth for small populations, but an upper limit, or *carrying capacity*, that prevents unbounded growth. The equation was suggested by Verhulst in 1836 (see, for example, [64, p.2]):

$$\dot{r} = \rho r(1 - r/\kappa) \tag{10}$$

It is rare among non-linear ODEs in that it has an analytic solution:

$$r(t) = \frac{\kappa r_0 e^{\rho t}}{\kappa + r_0(e^{\rho t} - 1)} \tag{11}$$

It has a single point attractor at $r_\infty = \kappa$: whatever the initial population r_0, it always converges to the carrying capacity κ.

Contrast this smooth behaviour with the very different complex periodic and chaotic behaviour of its discrete time analogue: the logistic map, §2.2.2. This is a general feature: the discrete time analogue of simple ODEs can exhibit similarly complex behaviour as the logistic map. This does *not* mean, however, that ODEs themselves are unable to display computationally-interesting dynamics.

2.3.4 Infinite dimensions: PDEs

Consider the reaction-diffusion (RD) equation, which models chemical species reacting (non-linearly) locally with each other, and diffusing (linearly) through space. The relevant state variables are the concentrations of the reacting diffusing chemicals, and are functions of time, and also of space: $r_i(t, \mathbf{x})$. For a two chemical species, the RD equation is:

$$\begin{aligned} \frac{\partial r_1}{\partial t} &= f_1(r_1, r_2) + k_1 \nabla^2 r_1 \\ \frac{\partial r_2}{\partial t} &= f_2(r_1, r_2) + k_2 \nabla^2 r_2 \end{aligned} \tag{12}$$

Each r_i, since it is a function of continuous space $\mathbf{x}$, can be thought of as an (uncountably) infinite dimensional state variable. Rather than having a state vector with a finite number of indices r_i, we can consider an infinite-dimensional state vector indexed by position, $r(\mathbf{x})$. The state variable at each position can itself have multiple components (such as the two chemical concentrations, above), leading to $\mathbf{r}(\mathbf{x})$. The space derivative is used to define the dynamics in terms of a local (infinitesimal) neighbourhood.

There is a natural link between partial differential equation (PDE) systems and Cellular Automata. CAs are one natural way to simulate PDEs in a discrete domain [5] [106, prob.9]. Care is needed in this process to ensure that the CA models the correct PDE dynamics, and does not introduce artefacts due to its own discrete dynamics [101, 107]. Despite this caveat, there are some exact correspondences: *ultradiscretisation* [96] can be used to derive CA-like rules that preserve the properties of a given continuous system, for a class of integrable PDEs; *inverse ultradiscretisation* [48] transforms a CA into a PDE, preserving its properties. A different approach represents CA configurations using continuous *bump functions* [66], and derives a PDE that evolves the bumps to follow the given CA rule.

The dynamical theory of these infinite dimensional spaces is not as well developed as in the finite case. Much of the work concentrates on PDEs whose dynamics can be rigorously reduced to a finite sub-space, so that the existing dynamical systems theory is applicable. See, for example, [78, 95].

2.3.5 Reaction-diffusion computers

Reaction-diffusion computers [5] use chemical dynamics. The relevant state variables are the concentrations of the reacting diffusing chemicals, which are functions of time and space: $r_i(t,\mathbf{x})$. For a two chemical species, the relevant RD equation is given by equation 12.

Reaction-diffusion systems have a rich set of behaviours, exhibiting spatial-temporal patterns including oscillations and propagating waves. The computation is performed by the interacting wave fronts; the output can be measured from the concentrations of the reagents. RD systems have been used to tackle a wide variety of computation problems (for example, image processing [47], robot navigation [4]); here we look at two that demonstrate computation exploiting the natural dynamics, and one that demonstrates the potential for universal computation.

Voronoi diagrams. An RD computer can solve a 2D Voronoi problem: given a set S of points in the plane, divide the plane into $|S|$ regions R such that every point in a given region $R_i(s_i)$ is closer to s_i than to any other $s_j \in S$.

This problem can be solved directly by a 2D RD computer [97] (figure 15). One reagent forms a substrate; the second reagent marks the position of the data set of points S. The data-reagent diffuses and reacts with the substrate-reagent, forming waves propagating from each data point, and leaving a coloured precipitate. Waves meet at the borders of the Voronoi regions, since their constant speed of propagation implies that they have travelled equal distances from their starting points. When waves meet, they interact and form no precipitate. So the lack of precipitate indicates the computed boundaries of the Voronoi regions.

Shortest path searching. A propagating wave technique can be used to find the shortest path through a maze or around obstacles [88, 6, 36]. The maze can be encoded in the chemical substrate, or by using a light mask. A wave is initiated at the start of the maze, and a series of time lapse pictures are taken as the wave

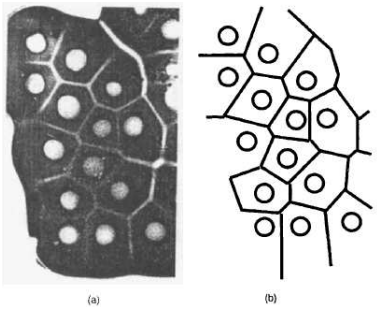

Fig. 15 An RD computer solving a Voronoi problem (figure from [97, fig.3])

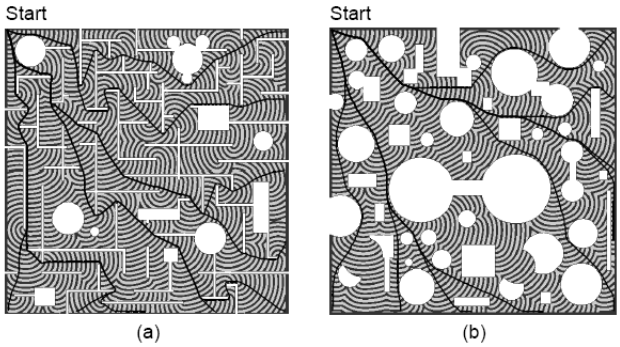

Fig. 16 Computer simulation of a shortest path computation in mazes with obstacles (from [36])

propagates at a uniformly speed, which provide a series of equidistant locations from the starting point; these are used in a post-processing phase to construct the shortest path (figure 16).

Logic gates. Propagating waves can be confined to channels ("wires") and interact at junctions ("gates") so arranged such that the interactions perform logical operations. See, for example, [62, 82, 98]. Hence the continuous RD system dynamics can be arranged by careful choice of initial conditions to simulate a digital circuit.

2.3.6 Generic analogue computers

In general, analogue computers gain their efficiency by directly exploiting the physical dynamics of the implementation medium. There is a wide range of problem-specific analogue computers, such as the reaction-diffusion computers described above, but there are also general purpose analogue computers.

For example, Mills has built implementations of Rubel's general purpose extended analog computer [81]. The computational substrate is simply a conductive sheet [58, 60], which directly solves differential equations; the system is "pro-

	discrete: $t \in \mathcal{N}$	continuous: $t \in \mathcal{R}$
$\mathbf{s} \in \mathcal{S}^N$	finite CAs ; RBNs	
$\mathbf{s} \in \mathcal{S}^\infty$	infinite CAs ; TMs	
$r \in \mathcal{R}$	iterated maps	
$\mathbf{r} \in \mathcal{R}^N$	CMLs	ODEs
$\mathbf{r} \in \mathcal{R}^\infty$		PDEs
$\mathbf{x} \in \mathcal{S}^M \times \mathcal{R}^N$	hybrid	

Table 3 Classification of the different kinds of autonomous dynamical systems, in terms of their state space and time evolution.

grammed" by applying specific potentials and logic functions at particular points in the sheet. Mills has developed a computational metaphor to aid programming by analogy [59], and is currently developing a compiler to enable straightforward solution of given differential equations [private communication, Dec 2008].

2.4 Hybrid dynamical systems

So far, all the systems considered have homogeneous dimensions, for example, all $\mathcal{S}$ or all $\mathcal{R}$. More complicated dynamical systems have heterogeneous dimensions. For example, coupling a classical finite state machine with a continuous system would yield a hybrid system with a dimensionality like $\mathcal{S}^M \times \mathcal{R}^N$.

It is likely that the topology of such a hybrid system would consist of relatively weakly coupled sub-components (figure 11b), which should help in their analysis from a dynamical systems point of view.

2.5 Summary

The classes of autonomous dynamical systems that have been discussed are summarised in table 3.

From a computational perspective, one important classification dimension is the implementation: whether the computational dynamics is implemented "naturally", that is, directly by the physical dynamics of the underlying medium, or whether it is implemented in terms of a virtual machine (VM) itself implemented on that underlying dynamics.

Discrete systems tend to be implemented in terms of VMs. This has the advantage that the computational dynamics is essentially independent of the underlying medium (witness the diversity of systems that implement boolean logic, for exam-

ple), and so can be analysed in isolation. It has the disadvantage of the computational overhead imposed by the VM layer.

One goal of continuous dynamical systems is to provide a computational dynamics closely matched to the physical dynamics, with corresponding gains in efficiency. The downside is that such systems are more likely to be constrained by their physical dynamics, and so are less likely to be Turing-universal computational systems.

3 Open dynamical systems

3.1 Openness as environmental inputs

The systems described so far are *autonomous*, or *closed*. They have an *initial condition* (identifying one state $\mathbf{x}_0$ from the $\mathcal{X}^N$ possible), and then the fixed (non-time-dependent) dynamics proceeds with no input or interference from the outside world. They move to an attractor, the result of the computation, or they may not discover an attractor, in which case the computation has no result. This is the classical, "ballistic" style of computation exemplified by the Turing Machine, or a closed dissipative system relaxing to equilibrium.

Open, or non-autonomous, systems, on the other hand, have dynamics that are governed by parameters that change over time. These parameters are inputs from the environment.

Consider an open dynamical system with N degrees of freedom: its state can be defined by an ND state vector $\mathbf{x}(t) \in \mathcal{X}^N$. The state space is $\mathcal{X}^N$. Now there is also an input space $\mathcal{P}$, and an output space $\mathcal{Q}$. The dynamics f maps the current state and input to the next state and output; $f : \mathcal{X}^N \times \mathcal{P} \to \mathcal{X}^N \times \mathcal{Q}$.

There is a similarity here to a parameterised *family* of dynamics (§2.2.1). But here the parameter p is a function of t, and is considered an *input* to the dynamics, a way of modulating or controlling the dynamics, for example, moving it between periodic and chaotic attractor behaviours.

3.1.1 Timescales

Understanding open systems is significantly more challenging than understanding closed systems, and depends in part on the relationship between the timescale on which the input is changing and the timescale on which they dynamics is acting. Dynamical systems have a "natural" timescale: the time needed to discover the attractor. As Beer [9] says:

> Because ... the flow is a function of the parameters, in a nonautonomous dynamical system the system state is governed by a flow which is changing in time (perhaps drastically if the parameter values cross bifurcation points in parameter space). Nonautonomous systems are much more difficult to characterize than autonomous ones unless the input has a particularly

simple (e.g., periodic) structure. In the nonautonomous case, most of the concepts that we have described above (e.g., attractors, basins of attraction, etc.) apply only on timescales small relative to the timescale of the parameter variations. However, one can sometimes piece together a qualitative understanding of the behaviour of a nonautonomous system from an understanding of its autonomous dynamics at constant inputs and the way in which its input varies in time.

That is, an input changes the dynamics of the system, by changing to a different member of the parameterised family. This new member might have moved attractors, or be on the other side of a bifurcation point with different kinds of attractors. A system immediately after an input will be in the same position in its state space, but the underlying attractor structure of that space may have changed.

So, if the input is changing slowly with respect to the dynamics, the system is able to complete any transient behaviour and reach the changed attractor before the input changes the dynamics yet again, even if it has passed through a discontinuous, catastrophic bifurcation point. On these timescales, the system is able to "track" the changing dynamics, and so its behaviour can be analysed piecewise, as a sequence of essentially unchanging systems. Even so, such systems can exhibit *hysteresis*: restoring a parameter to a previous value may not necessarily restore the system to its corresponding previous state, if this path through parameter space crosses catastrophic bifurcation points.

If the input is changing quickly with respect to the dynamics, then the system is unable to respond to changed dynamics before it has changed again. It will mostly be exhibiting transient behaviour.

Most interesting and complex is the case where the input is changing on a timescale similar to that of the dynamics: then the system is influenced by its dynamics, but it may never quite, or only just, reach any attractor before the next change occurs.

The situation can get even more complicated, when the input parameter p is a function of space as well as time, $p(\mathbf{x},t)$. For example, it might be a temperature gradient, or magnetic field gradient, which can also drive the system.

3.1.2 Computation in terms of trajectories

Since such open systems need not reach a "halting state" of being on an attractor, the computational perspective is necessarily broader. The computation being performed can be viewed as the *trajectory* the system takes through the changing attractor space: which attractor basins are visited, in which order.

The discussion of timescales implies that, for useful computation, the dynamical timescale of the system should not be significantly slower than the input timescales.

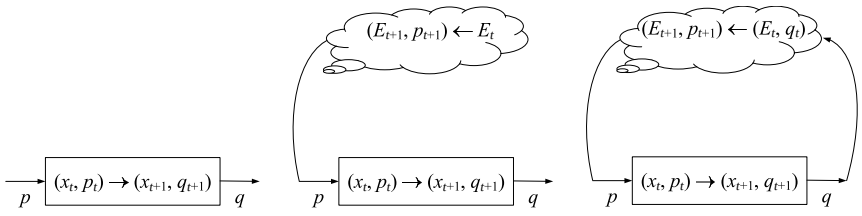

Fig. 17 Inputs. (a) arbitrary input stream; (b) constrained inputs from a structured environment; (c) constrained inputs from an interacting environment in a feedback loop.

3.1.3 Environmental constraints

Type A: arbitrary input stream. In the simplest open case, it is assumed that the system can be provided with any input, regardless of what it is doing, or has done (figure 17a). The input may as well be considered random. This is not particularly interesting, except in the cases where the system can somehow exploit noise, for example, using some kind of informational analogue of a ratchet mechanism (for example [21]).

This case is formally equivalent to a closed system, as the sequence of arbitrary inputs could conceptually be provided at the start, embedded in an (expanded) state $\mathcal{X}^N \times seq\ \mathcal{P}$ as part of the initial condition, along with some pointer to the "current time" value, and the dynamics updated to allow access only to the current value (for example, see [15], where such an approach is taken to embed inputs into the initial state of the model of a formal language). So there is no new computational capability, except as provided by the (potentially, much) larger state space.

Type B: environmentally constrained input stream. More interesting is the case where the inputs come from an environment that has some rich dynamical structure that the system can couple to and exploit (figure 17b). Here the environment is an autonomous dynamical system, unaffected by any inputs of its own.

Since the environment is autonomous, its sequence of inputs could again conceptually be provided at the start. However, this case is qualitatively different from the previous one. We are now assuming that there is some *structure* in the environment, and hence in the sequence of inputs. This implies that there are regions of the system state space that are never explored, parts of its underlying dynamics that are never exercised. As before when talking of virtual machines (§2.1.3), the computation is restricted to a sub-space: the sub-space and its trajectories here correspond to the computation in the context of the structured environment: the inputs provide *information* that the system need not itself compute. And since the environment may be unboundedly large, the sequence of inputs may represent an unboundedly large amount of computation provided to the system.

Type C: feedback constrained input stream. Most interesting is the case where the environment and system are both open dynamical systems in a rich feedback loop. Then outputs from the system will alter the environment, and affect its subse-

quent inputs (figure 17c). So the actual sequence of inputs cannot even conceptually be provided at the start.

Again, the environment's inputs will be constrained to a region of state space, but here this region is (partly) determined by the system: the environment and systems are coupled, the dynamics of each perturbing the trajectory of the other, in a feedback loop.

Such an environment may well contain other open systems similar to the system being considered. And again, since the environment may be unboundedly large, it may represent an unboundedly large amount of computation provided to the system, but here, the specific computation provided is affected by what the system does, because of the feedback coupling. This opens up the possibility for a system to offload some of its computational burden onto the environment (see §3.4.4).

Beer [9] points out that such a coupled environment and system together form a higher-dimensional autonomous dynamical system, with its own attractors, and, because of its larger state space, that this combined system can "generate a richer range of dynamical behavior than either system could do individually".

3.2 Open discrete space, discrete time dynamical systems

The autonomous discrete systems have an *initial condition* (identifying one state $\mathbf{s}_0$ from the $|\mathbb{S}|^N$ possible). In the finite case, they always "halt" on an attractor cycle, this halting state (cycle) being the result of the computation; in the infinite case, they either halt on such a cycle, or they may not discover an attractor, in which case the computation has no result.

In the open discrete space, discrete time case, where $\mathbb{X} = \mathbb{S}$, the time evolution of the state, and the output function, are given by $(\mathbf{s}_{t+1}, q_{t+1}) = f(\mathbf{s}_t, p_t)$.

3.2.1 Modulating the dynamics

The input can provide a "kick" or perturbation, changing (a few bits of) the state at a particular timestep. This may move the system into a different attractor basin.

The input might also "clamp" the system into a particular substate (by fixing the value of some bits for many timesteps). This not only perturbs the system at the point where the bits are clamped, but can also change the global dynamics (if the natural dynamics would change the value of the clamped bits, for example), producing new attractors, and changing or removing existing attractors. For example, clamping some bits in a CA can result in "walls" across which information cannot flow, isolating regions, and hence changing the dynamical structure of the system. (See figure 18.) Such a simple partitioning is harder to achieve in a more irregularly connected structure such as an RBN.

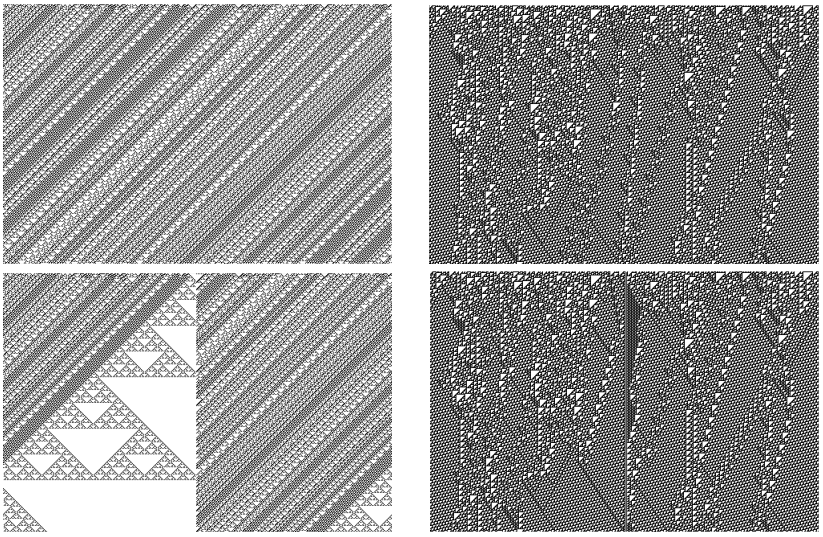

Fig. 18 Dependence of CA dynamics on a "clamped" bit. The upper plot shows ordinary periodic boundary conditions; the lower plot shows the same initial conditions, but with the central bit "clamped" to 0. (a) ECA rule 26; (b) ECA rule 110

3.2.2 Perturbing Random Boolean Network state

Kauffman [42] defines a *minimal perturbation* to the state of an RBN to be flipping the state of a single node at one timestep. Flipping the state of node i at time t is equivalent to changing its update rule at time $t-1$ to be $c_{i,t} = \neg\phi_i(\chi_{i,t-1})$. Such a perturbation leaves the underlying dynamics, and hence the attractor basin structure, the same; it merely moves the current state to a different position in the state space, from where it continues to evolve under the original dynamics: it is a transient perturbation to the state.

Kauffman [42] describes the *stability* of RBN attractors to minimal perturbations: if the system is on an attractor and suffers a minimal perturbation, does it return to the same attractor, or move to a different one? Is the system *homeostatic*? (Homeostasis is the tendency to maintain a constant state, and to restore its state if perturbed.)

Kauffman [43] describes the *reachability* of other attractors after a minimal perturbation: if the system moves to a different attractor, is it likely to move to any other attractor, or just a subset of them? If the current attractor is considered the analogue of "cell type", how many other types can it *differentiate* into under minimal perturbation?

Kauffman's results are summarised in table 4, which picks out the $k = 2$ networks as having "interesting" (non-chaotic) dynamics (a small number of attractors, with small cycle lengths) and interesting behaviour under minimal perturbation (high stability so a perturbation usually has no effect; low reachability so when a perturbation

k	stability	reachability
1	low	high
2	high	low
> 5	low	high

Table 4 Dynamics of RBNs for different k (adapted from [43, table 5.1])

moves the system to another attractor, it moves it to one of only a small subset of possible attractors).

3.2.3 Perturbing Random Boolean Network connectivity

Kauffman [42] also defines a *structural perturbation* to an RBN as being a permanent mutation in the connectivity or in the boolean function. So a structural perturbation at time t_0 could change the update rule of cell i at all time $t > t_0$ to be $\phi_i'(\chi_{i,t})$ or change the neighbourhood of cell i at all time $t > t_0$ to be $\chi_{i,t}'$. Since the dynamics is defined by all the ϕ_i and χ_i, such a perturbation changes the underlying dynamics, and hence the attractor basin structure: it is a permanent perturbation to the dynamics, yielding a new RBN.

Such a perturbation could have several consequences: a state previously on an attractor cycle might become a transient state; a state previously on a cycle might move to a cycle of different length, comprising different states; a state might move from an attractor with a small basin of attraction to one with a large basin; a state might move from a stable (homeostatic) attractor to an unstable attractor; and so on.

Kauffman [43] relates structural perturbation to the *mutation* of a cell; if there is only a small change to the dynamics, this represents mutation to a "similar" kind of cell.

3.3 Open continuous space, discrete time dynamical systems

In the open continuous space, discrete time case, where $\mathcal{X} = \mathcal{R}$, the time evolution of the state, and the output function, are given by $(\mathbf{r}_{t+1}, q_{t+1}) = f(\mathbf{r}_t, p_t)$.

3.3.1 Open Coupled Map Lattices, and Chaos Computing

Sinha and Ditto [83, 84] investigate the computational properties of coupled chaotic logistic maps ($\phi_0 =$ the logistic map with $\lambda = 4$, see §2.2.3) with open boundaries. The coupling function ϕ_c is a threshold function: $\phi_c =$ **if** $r < \theta$ **then** 0 **else** $r - \theta$, and is unidirectional. This unidirectional relaxation propagates along the lattice until all elements have a value below threshold (all the $r_i \leq \theta$), at which point the next

timestep iteration of the logistic dynamics occurs (so the timescale of the relaxation is much shorter than the timescale of the logistic dynamics). The boundary of the lattice is open, so the final element can relax below threshold by removing its excess value from the system. Depending on the particular threshold value, and the current state of the system, this process can result in non-linear avalanches of relaxation along the lattice. Transients are short (typically one dynamical timestep), and the system displays a variety of attractor dynamics, stable to small amounts of noise, and determined by the input threshold parameter.

Sinha and Ditto [83, 84] discuss how to use this system to perform computation. A single lattice element with two inputs (either external, or from the coupled output of other lattice elements) and appropriate input value of its threshold can implement a universal NOR logic gate in a single iteration of the logistic dynamics, where the amount of output encodes the result. Hence networks of such elements can be coupled together to implement more complicated logic circuits. Sinha and Ditto [84] note that it is the chaotic properties in general, not the logistic map in particular, that give these coupled systems their computational abilities, and suggest a possible implementation based on non-linear lasers forming a coupled chaotic Lorenz system. They dub their approach *chaos computing*.

Additionally, Sinha and Ditto [83, 84] discuss how to implement other functions, such as addition, multiplication, and least common multiple, by suitable choice of numerical encoding, coupling and input thresholds. These choices program the underlying chaotic dynamics to perform computation directly, rather than emulating a virtual machine of compositions of logic gates [84]:

> ... dynamics can perform computation not just by emulating logic gates or simple arithmetic operations, but by performing more sophisticated operations through self-organization rather than composites of simpler operations.

Despite this observation, their subsequent work [22, 85] concentrates on using more complicated (but realisable) chaotic dynamics to implement robust logic circuits, that can be readily reconfigured merely by altering thresholds. They emphasise the openness of their approach [85]:

> it can yield a gate architecture that can dynamically switch between different gates, without rewiring the circuit. Such configuration changes can be implemented either by a predetermined schedule or by the outcome of computation. Therefore, the flexibility of obtaining different logic operations using varying thresholds on the same physical element may lead to new dynamic architecture concepts

3.4 Open continuous space, continuous time dynamical systems

In the open continuous space, continuous time case, $\mathcal{X} = \mathcal{R}$, the time evolution of the state, and the sequence of outputs, are given by $\left(\dot{\mathbf{r}}, q(t)\right) = f(\mathbf{r}, p(t))$.

Note again the similarity to a parameterised *family* of dynamics (§2.2.1), with the (input) parameter p being a function of t. Examples of such input parameters

might include the temperature T of the system, or the value of an externally-imposed magnetic field $\mathbf{B}$ permeating the system.

3.4.1 Ott, Grebogi, Yorke (OGY) control laws

There are unstable periodic orbits in strange attractors. Small perturbations in the control parameter can be used to keep the system in one of these; the required perturbations are calculated using the OGY control laws [67]. (It is not necessary to know the underlying dynamics to do this; application of the control laws involves calculating the required parameter from observations of the system.) There can be long transient behaviour before the system gets "close" to the desired periodic orbit, and noise can result in bursts of chaotic behaviour. The approach can also be used to switch between different periodic orbits with different characteristics (with some transient chaotic behaviour).

Ott et al [67] note that a chaotic system is potentially more flexible, because this approach can be used to hold it in a variety of different periodic orbits, whereas this range of behaviours would require a range of separate systems with non-chaotic dynamics. Sinha and Ditto (§3.3.1) make similar observations about their coupled nonlinear maps, although there they claim a lower computational burden (the thresholds can be simply calculated, and stored in a lookup table) and shorter (essentially zero) transient behaviours.

3.4.2 Liquid Crystal systems

Liquid crystals are a form of matter that lies on the boundary between solids and liquids (sometimes called "the fourth phase of matter"). A liquid crystal has both complex dynamics (the molecules can flow and rotate) and complex structure (the molecules are ordered on length scales much bigger than their individual sizes).

Such materials can perform computation. Harding and Miller [30, 31, 32] have demonstrated that a liquid crystal chip can be programmed to act as a tone discriminator (a simple arbitrary input system, where the inputs are tones of two different frequencies that are to be discriminated) and as a robot controller (a constrained feedback system, where the inputs from the environment depend on the robot's position, and the robot's outputs change its position).

It is currently unclear how the liquid crystal performs its computations: in the referenced cases the material was programmed using an evolutionary algorithm. It would be interesting to analyse these results from a dynamical systems perspective.

3.4.3 NMR logic gates

Nuclear magnetic resonance (NMR) uses radio frequency pulses to manipulate nuclear spins in a magnetic field. Depending on the particular values chosen from a

rich potential set of parameters (frequency, phase, duration, delay, and more), pulses can be combined in various ways to produce different outputs. These pulses and the outputs can be interpreted as encoding binary values. Under these interpretations, the system can be used to implement a single universal logic gate, a circuit of these gates combined in parallel and in sequence that implement other logic gates, and a half-adder circuit [79].

Under these interpretations, this is a Type A open system: any arbitrary set of boolean inputs is permitted, and the system implements the corresponding logic gate computation on these inputs. Under the wider view of the full parameter space, this is a Type B open system, with the inputs being restricted by the environment (the experimenter) to values that can be interpreted as binary bits. Hence we see how this set-up corresponds to a logic gate virtual machine (§2.1.3) implemented on the underlying dynamics of the nuclear spin system.

3.4.4 Embodiment

Beer [9] takes a dynamical systems approach to adaptive agents in a changing environment. For example, a robot agent adapts its walking behaviour depending on the kind of terrain it is moving across. So an agent receives sensory input from a structured environment (it sees its current surroundings), which affects its internal dynamics (its state changes), which affects its outputs (its leg movements), which in turn affects the environment (at a very minimum, the agent moves to a new location in the environment, and hence sees new surroundings). So "a significant fraction of behavior must be seen as emerging from the ongoing interaction between an agent and its environment". Beer is talking here of Type C open dynamical systems (§3.1.3), coupled to their environment in a feedback loop.

Beer's aim is to use the language and concepts of dynamical systems theory to develop a theoretical framework for designing adaptive agents. Certain computational tasks, such as planning, can be greatly simplified by exploiting input from the structured environment (for example, the agent seeing where it is). This relates to Brooks' design principle: "use the world as its own model" [12].

Beer emphasises that, because of the coupling with the environment, "an agent's behavior properly resides only in the dynamics of the coupled system", and hence cannot be understood or analysed in isolation. Indeed, an agent is adapted to some environments and not others. Beer analyses this idea of adaptive fit, and determines that it requires that an agent maintains its trajectory within a certain volume of its state space (which volume may change with time) under perturbations from the environment. This is related to an *autopoietic* [52] (self-creating and self-maintaining) view of adaptive fitness, and helps define what is needed for *homeostasis*.

This area also links with a whole burgeoning sub-discipline of using dynamical systems theory to model the brain and its cognitive processes. However, that is deemed to be outside the scope of this particular discussion. The interested reader is referred to [44].

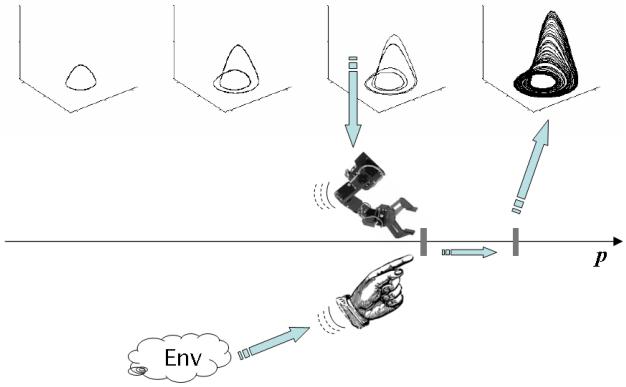

Fig. 19 Two sources of parameter variation: external control versus internal dynamics.

3.5 External control versus internal dynamics

Here we have considered the parameter $p(t)$ to be an externally provided input, moving a dynamical system between members of its parameterised family. Alternatively, we could have a case where the parameter p was another degree of freedom, or dimension, of the dynamical system, where the dynamics of the system itself affect the value of p (figure 19).

This potentially gives a model of a *self-organising* system. For example, Melby et al [54] describe a logistic map (§2.2.2) where the value of the dynamical variable r_t is fed through a low-pass filter to (slowly) affect the parameter λ: the system self-adjusts to values of λ at the edge of chaos, and does so even when subject to an external force attempting to drive it back into the chaotic region [55]. The situation is a little more complicated in the presence of noise: the system still self-adjusts to suppress chaos, but a power-law distribution of chaotic outbreaks occurs [56].

An alternative view is of a hierarchy of coupled dynamical systems, where the outputs at one level couple to the control parameter(s) at another level. Abraham and Shaw [1, 2] explore this idea in more detail.

4 Constructive dynamical systems

So far, we have considered predetermined fixed state spaces of given dimensionality. However, this is not the case even for classical computational systems. Their data structures define their abstract state space: every time new memory is allocated in the course of the computation, the state space grows in dimension; memory deallocation shrinks the state space. A dynamical systems perspective on computation needs to consider cases where the state space itself dynamically changes dimensionality, $\mathcal{X}^{N(t)}$.

In such cases, the dynamics *constructs* the state space, and the state space constrains the dynamics. Closed autonomous systems can exhibit such growth, classical computation being one such example. In open systems, this process can be thought of as the flow of information, matter, energy being recruited to construct new dimensions of the system: the system inputs "food" (constructs dimensions), and excretes "waste" (collapses dimensions). This process of modifying its own dimensionality affects the possible dynamics (recall, for example, that strange attractors require at least a 3D continuous space §2.3.1).

This is an aspect of dynamical systems theory that has received very little attention: the current theory assumes a pre-defined, fixed state space. Because of the paucity of theory in this area, this section is intended to illustrate the kinds of processes that need to be incorporated in such a theory.

Metadynamics. One approach to incorporating growth might be to model the system with an infinite number of dimensions, confining the dynamics to the finite-dimensional subspace corresponding to the current state space. As new dimensions are needed, the dynamics expands into the pre-existing dimensions. However, this approach appears to be a mathematical "trick" that hides the essential properties of the underlying system[3]: precisely what confines the dynamics to a subspace, what causes it to grow into new dimensions, and what determines the topology (the way the information flows between dimensions)?

A different approach might be to have these a new dimensions start in some small "curled up" form; a dimension could "uncurl" or "unfold" sufficiently to support the current state value along that dimension, and curl back up when no longer needed. Alternatively, the dimensions might be fractal in nature, with the fractal dimension of the state space increasing, gradually "fattening up" the new dimension.

This suggests that the state space itself might have a *metadynamics*. The metadynamics (the high-level dynamics of the state space) can be studied in isolation from the low-level dynamics (trajectories of the system through the state space).

Another form of metadynamics is to allow the topology (see §2.2.4) to change dynamically, even if the dimensionality is constant. Such "network dynamics" allows the neighbourhood function (§2.1.5) to be a function of time, $\nu(t)$.

Baguelin et al [8, 63] consider a metadynamical transition rule, which changes the state space (and the corresponding dynamical state transition rule) on a slower timescale that the system's dynamics: the system spends most of its time evolving under its dynamics, punctuated by relatively few metadynamical state space changes. The overall system dynamics is defined by "concatenating" the various lower level dynamical behaviours at the metadynamical change points. Baguelin et al [8, 63] apply this approach to interacting populations of bacteria and phages (the dynamics) that are also evolving (a slower timescale metadynamics).

[3] This approach also has a small technical issue: how to distinguish a system that is not using a particular dimension, from one that is using it, but is currently confined to the zero value of that dimension: how to distinguish "absence" from the "presence of nothing". This can in turn be solved by a further mathematical trick, of introducing some special value, $\perp$, to make this distinction. But the tricks are piling up.

Whatever approach is used, it needs to be able to cope with different types of dimensions, as the dynamics results in new types of state variables being constructed (for example, in evolution, new species can be considered to occupy new types of dimensions). If such types cannot be statically predetermined, but computed only by the unfolding dynamics, it is harder to see how to incorporate this into an analysis based on a known, pre-existing set of state spaces.

Timescales. Important timescales in a constructive dynamical system are the speed at which the state space grows compared to dynamics on that space. Where growth timescales are much less than the dynamical timescales (a change in the number of dimensions tends to happen after the system has relaxed to an attractor), piecewise approximations may be made in a way similar to slow open systems (§3.1.1), as in the metadynamics approach outline above. But in faster-changing systems (for example, L-Systems §4.1.2, where the dimensionality can change every iteration timestep) this is not possible, and new analysis techniques must be sought.

Computation in terms of construction. In an autonomous (closed) constructive system, the result of the computation (if the computation halts) could still be considered as the attractor (final macrostate), and additionally include the structure (dimensionality and topology) of the final grown state space.

In an open, non-halting constructive system, the result of the computation could still be considered as the trajectory through a growing state space, and additionally include the structure (dimensionality and topology) of the state space along that trajectory.

4.1 Constructive discrete dynamical systems

4.1.1 Classical computation

Turing Machine. A Turing Machine can be thought of as a growing system: the tape is of finite but unbounded length, and can be considered to grow (lazily) whenever a new tape position is required. It can also be considered to shrink if the last symbol on the tape is erased.

So the question of growth can be linked to undecidability: whether the dimensionality of the state space stays bounded (if the computation halts or loops), or grows without limit, is formally undecidable.

Object orientation. OO systems are particularly constructive in this sense: new objects are created, increasing the dimensionality of the state space; when they go out of scope, the state space shrinks again.

Such systems are usually designed on the software engineering principle of ensuring "weak coupling" by small interfaces, which keeps components separate and modular by restricting the information flow. That is, the topology of the dynamics is deliberately restricted.

Contrast the connectivity of RBNs, which have a "small-world" topology, and require a large number of connections to be cut to partition the graph (figure 11). This enables rapid information flow, potentially enabling different classes of dynamics. The consequences of these design decisions are rarely addressed from a dynamical systems perspective.

Closed or open. These classical computational systems can be closed (for example, the TM model), or open (interactive systems). The open systems are usually analysed as Type A (§3.1.3), where the external environment can potentially provide any input.

4.1.2 Rewriting systems

Imagine trying to "grow" an RBN where, if it got to a certain state, a new node would appear, or an existing node would disappear (with consequent rewiring). The appearance or disappearance of a node would change the dimensionality of the state space (to $N \pm 1$), and also the underlying attractor structure, and hence change the computation being performed.

Similarly imagine trying to "grow" a 1D CA. One could add a new cell, growing the dimensionality of the state space from $\mathcal{S}^N$ to $\mathcal{S}^{N+1}$. Where should the cell be added in the line? The CA dynamics is symmetric under renumbering shifts, and so it should be possible to add the cell anywhere with equal ease. But unless it is added at special position $i = N$, subsequent cells have to be renumbered. This illustrates the tyranny of a global coordinate system: "The introduction of numbers as coordinates ... is an act of violence" [102]. Instead we would like a coordinate-free, or purely topological, approach to growth.

L-Systems. Lindenmeyer's L-Systems [76] are such a coordinate-free approach to growth: they are generative grammars that define how symbols in a string are rewritten ("grow") depending on their local context (their topology, their neighbourhood symbols), not on any global coordinate system.

The simplest D0L-Systems (deterministic, context free) can be described in the notation of this chapter as follows. The state $\mathbf{s}$ is a string of elements (symbols) drawn from the finite alphabet $\mathcal{S}$, so $\mathbf{s} \in \mathcal{S}^*$. The local dynamics ϕ (rewriting rules, or productions) is given by $\phi : \mathcal{S} \to \mathcal{S}^*$. At each timestep, each element in the state string[4] is updated (rewritten) in parallel: $s_{i,t+1} = \phi(s_{i,t})$, and the resulting substrings concatenated to form the new state string.

In conventional use, one starts from some initial state (axiom) $\mathbf{s}_0 \in \mathcal{S}^+$, and follows the dynamics for several iterations to reveal the "grown" structure. However, the dynamics does not not necessarily result in a different number of dimensions (a changed length string). For example [19, 20] use L-Systems to model protein fold-

[4] Note here that indices are used merely to identify the string elements in order to help define the local dynamics. They are not a fundamental part of the string data type, which supports operations such as insertion and deletion, unlike the fundamentally indexed array data type.

ing: there the rewriting models the change in conformation of the (rendered) string, and does not result in any change in the length of the string itself.

In Parametric L-Systems [76], each symbol has associated parameters, and so the state space is $(\mathcal{S} \times \mathcal{P})^*$.

In the context-free case there is no information flow between the dimensions: each individual dimension grows into new dimensions in a manner based on its own substate alone. Context-sensitive L-Systems [76] couple the dimensions together, and define a tree-structured topology on the growing space. In standard L-Systems, the topology is explicitly encoded in the state string, by use of reserved symbols to define branching.

L-Systems also include a *rendering* step, where the symbols in the string are interpreted as commands in some language, typically a turtle graphics language, to construct a representation of the string [76]. In standard L-Systems this rendering produces a geometrical structure that follows the tree-structured topology of the state space, typically to produce rendered images of plants. However, other renderings are possible, and can produce non-geometric outputs and outputs with different topologies. For example, [35] interpret the symbols as instructions for generating topological descriptions of neural networks.

These simplest L-Systems are autonomous dynamical systems, and any computation performed in the rendering step is separate from the dynamics of the growth process.

Parametric L-Systems, where the values of (some) parameters are inputs, are Type A or Type B open dynamical systems (§3.1.3). In typical L-System applications, they are used as Type B, because the input is assumed to be coming from some constrained environmental source (for example day length, or other weather conditions).

Environmentally-sensitive L-Systems [75] are also Type B open dynamical systems (§3.1.3), of a different form. The dynamics again takes into account external inputs (for example, the amount and direction of sunlight, or collision with static obstacles, when modelling plant growth), but here the particular rendering process is coupled into the dynamics, as the input is a function of the specific geometry of the rendered result (rendered leaves shading other leaves, etc), not just the topology of the abstract state.

Open L-Systems [65] are Type C open dynamical systems (§3.1.3). They include a feedback process, so the environment is affected by the L-System (for example, the environment might be the water supply, and the L-System the growing roots that are affecting the amount of water). Again, this approach couples the particular rendering process into the dynamics.

P-Systems. Păun's P-systems [71], and other membrane computing formalisms, are another form of rewriting systems with an underlying tree-structured topology. Here the tree-structure models not the branching of a plant, but the nesting of membranes, or containers. Membranes contain symbols, and rules that act on the symbols, and on membranes (creating, connecting, and dissolving them). The computational model successively applies the rules until no more are applicable; the

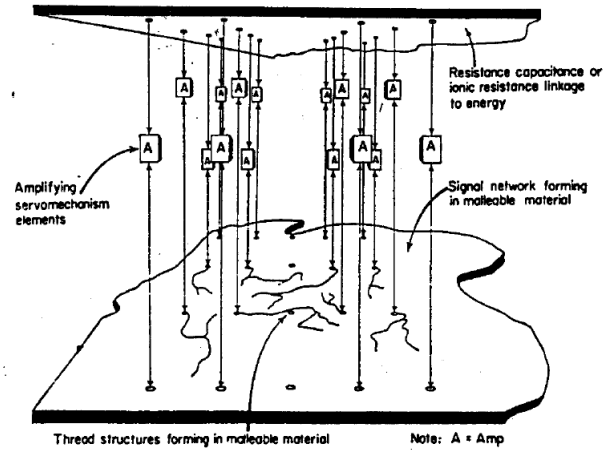

Fig. 20 Pask's schematic indicating the relationship between the electrode array and the ferrous sulphate medium ([70], as cited by [13, fig.1])

computation has halted, and the symbols in a given membrane are the result of the computation. Much theoretical effort is expended on determining the computational power of different classes of membrane systems: Turing-complete systems exist. So this prototypical system has a rather classical view of computation, but there is a plethora of variants with different computational models.

Topological approach. Michel and co-workers [26] contrast the "absolute space" coordinate-based philosophy of Newton, with the relative space, coordinate-free philosophy of Leibniz, from a computational perspective. Their approach to (discrete) "dynamical systems with a dynamical structure $(DS)^2$" is topological, focussing on the neighbourhood relation and induced subcollections (essentially a set of related elements in a neighbourhood). The neighbourhood relation can specify structures ranging from the regular spatial neighbourhood of a CA to the logical neighbourhood of a data structure such as a list, graph, or array [27]. The approach then specifies global dynamics in terms of parallel application of local transformations that rewrite subcollections [87, 27]. This approach subsumes dynamic rewriting systems such as L-Systems and P-systems, as well as more static-structured CA-like rewritings [27].

4.2 Constructive continuous dynamical systems

Gordon Pask was a member of the Cybernetics movement. In the 1950s he built a system that could "grow" into an "ear" (a sensor capable of detecting sound waves, or magnetic fields) by adaptively laying down suitable conducting filaments (figure 20) in response to environmental inputs and a structured "reward" input.

Cariani [13] provides an excellent review of this work, and some implication in terms of "organizationally-closed" (that is, able to construct their own input filters), "informationally-open" systems. In an open, adaptive, self-constructing system

> the dimensionality of the signal space can increase over time as new informational channels evolve. Hill-climbing is thus accomplished not only by following gradients upwards, but also by changing the dimensionality of the problem landscape when one can go no further using those dimensions already available.

4.3 Constructive hybrid dynamical systems

Winfree and co-workers [103, 80] are implementing "algorithmic self assembly". They use carefully-designed DNA fragments to implement nano-scale "sticky tiles", which self-assemble into crystalline structures. The self-assembly can be interpreted as a computation: which computation is performed depends on the tiles' design, which determines which tiles they will stick to. Their running example is the implementation of a rule 90 ECA (which from a single "on" state grows a Sierpinski gasket). Other tile designs can implement other CAs, or even a representation of the tape of a TM.

The model of this system is a hybrid system. There is an abstract Tile Assembly Model that captures the discrete dynamics of tiles (the particular CA or TM behaviour). This is augmented with a kinetic Tile Assembly Model, that captures continuous properties that account for errors in the assembly process.

They speculate that a similar process could be used to construct molecular electronic circuits and other nano-scale devices. That is, design a constructive dynamics to grow a structure that then has an autonomous computational dynamics.

5 Discussion and Conclusions

Classical computational systems can be analysed in terms of autonomous discrete dynamical systems. The dynamical systems approach can be extended to continuous systems, to give a computational perspective on embodied computational devices. It can be extended again, to include open dynamical systems coupled to a structured dynamical environment, and again to include constructive, or growing systems. The dynamical systems theory is, however, less well developed in these latter cases.

In addition to the straightforward computational perspective, this dynamical systems viewpoint gives insight into important properties of complex computational systems: robustness, and emergence.

5.1 Robustness

Homeostasis. Robustness is the ability to maintain function in the presence of perturbations, stresses, or errors. The system exhibits *homeostasis*. Classical computation, on the other hand, is notoriously *fragile*: a single bit change can completely alter the behaviour of the system.

From a dynamical systems computational point of view, robustness is the ability to find the attractor even when perturbed from the current trajectory. A system will find the attractor from anywhere within its basin of attraction: if a perturbation nevertheless maintains the basin, the system will be robust to that perturbation.

Here we see the source of classical computation's fragility: extremely small basins of attraction, such that any perturbation is likely to move the system to a different basin, hence to a different attractor, resulting in a different computation.

Compare this situation to the dynamical structure of $k = 2$ RBNs (§2.1.6). There a perturbation is likely to leave the system in the same basin. "Small attractors located inside a volume of states constituting their basins of attraction are the natural image of stable systems exhibiting homeostasis." [43, p.467] So robustness requires large basins of attraction, which implies relatively few macrostates compared to the number of microstates.

Continuous systems have greater potential for robustness. They have a notion of locality: a small perturbation moves the system a small distance in its state space, and is therefore likely to be in the same basin. (Although it is true that as well as fractal strange attractors, there can be fractal-structured basins; in such cases a small perturbation might end up in one of many other basins.) Alternatively, the small perturbation could be a small change to a parameter. Provided the parameter does not cross a bifurcation point, this will result in only a small change to the dynamics.

Kitano [45] explicitly casts a discussion of various kinds of biological robustness in terms of dynamical systems attractors.

Precision. Robustness comes at the price of precision, however. Many microstates per macrostate means a loss of precision: we no longer distinguish these microstates. Also, real-valued ouput from a continuous system cannot be measured with arbitrary precision. Similar to the case when considering the timescales, the precision of the computation and output should be matched to the precision of the environment and inputs.

Homeorhesis. Waddington [100] notes that homeostasis is too restrictive a term when considering growing systems, since they do not have a "steady" state to which to return: they are embarked on some trajectory through their ever-changing state space. He introduces the term *homeorhesis* [100, p.32], meaning "constant flow", in that systems actually maintain a constant trajectory through developmental space. He deprecates the alternative term, *canalisation*, because it may [100, p.43-44]

> suggest too concrete an image to be suitable as a name for the abstract quality to which it refers; but this seems a less important failing than those involved in the alternative term homeostasis.

Note that this implies that the developmental trajectory is in some sense an attractor of the growth process: a perturbation away from the growth trajectory is attracted back to that *trajectory*, not merely to some final attractor state. In biological terms, this is robust morphodynamics.

5.2 Emergence

The attractors and their basins, and the bifurcation points of parameterised systems, are *emergent properties* of their dynamics [92].

Although the microstates are changing on an attractor cycle, this can lead to a stable macrostate if observed on a timescale longer than the attractor period. In the dynamical systems view, everything is process (motion on an attractor), but when it is viewed on a suitable timescale, it can behave like a thing.

> An attractor functions as a symbol when it is viewed ... by a *slow observer*. If the dynamic along the attractor is too fast to be recorded by the slow-reading observer, he then may recognize the attractor only by its averaged attributes ..., but fail to recognize the trajectory along the attractor as a deterministic system. [1]

The slow observer does not see the intricate dynamics on the attractor, just some averaged behaviour, and this dynamics becomes an atomic component in its own right. What is lost is the microstructure; what remains is a stable pattern that becomes an entity in its own right. This connects directly with the concepts of relative timescales, where there is a sufficient difference between a fast timescale (of the underlying dynamics of the system), and a slower timescale (of the inputs perturbing the system §3.1.1; of the metadynamic timescales of state space change §4; of the slow observer of these processes).

These new high-level emergent entities can have their own state space and (meta)dynamics, particularly when coupled to other dynamical systems such as the environment, or in some hierarchical structure. So their (higher level) dynamics will exhibit attractors; motion on these attractors will appear as emergent entities to (even slower) observers. And so on.

5.3 The Future

These discussions imply that we need a dynamical systems theory of open, constructive computational systems, where new dimensions of state space, and new types of dimensions, are constructed by the computation as it progresses. This will provide one of the tools we need to enable us to study, understand, design and reason about robust emergent computational systems, covering the range from classically discrete, to non-classical embodied systems.

Acknowledgements

My thanks to Ed Powley for providing figures 1, 3, 4, and 18, and for drawing my attention to the work on exact correspondences between CAs and PDEs in section 2.3.4. My thanks to Andy Adamatzky, Leo Caves, Ed Clark, Simon Hickinbotham, Mic Lones, Adam Nellis, Ed Powley, and Jon Timmis for comments on a previous draft.

References

1. Abraham, R.H.: Dynamics and self-organization. In: Yates [112], pp. 599–613
2. Abraham, R.H., Shaw, C.D.: Dynamics: a visual introduction. In: Yates [112], pp. 543–597
3. Adamatzky, A. (ed.): Collision-Based Computing. Springer (2002)
4. Adamatzky, A., Arena, P., Basile, A., Carmona-Galan, R., Costello, B., Fortuna, L., Frasca, M., Rodriguez-Vazquez, A.: Reaction-diffusion navigation robot control: from chemical to VLSI analogic processors. IEEE Trans. Circuits and Systems **51**(5), 926–938 (2004)
5. Adamatzky, A., Costello, B.D.L., Asai, T.: Reaction-Diffusion Computers. Elsevier (2005)
6. Agladze, K., Magome, N., Aliev, R., Yamaguchi, T., Yoshikawa, K.: Finding the optimal path with the aid of chemical wave. Physica D **106**, 247–254 (1997)
7. Andersson, C., Nordahl, M.G.: Evolving coupled map lattices for computation. In: EuroGP'98, *LNCS*, vol. 1391, pp. 151–162. Springer (1998)
8. Baguelin, M., LeFèvre, J., Richard, J.P.: A formalism for models with a metadynamically varying structure. In: Proc. European Control Conference, Cambridge, UK (2003)
9. Beer, R.D.: A dynamical systems perspective on agent-environment interaction. Artificial Intelligence **72**, 173–215 (1995)
10. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning Ways for Your Mathematical Plays, vol. 2. Academic Press (1982)
11. Bossomaier, T., Sibley-Punnett, L., Cranny, T.: Basins of attraction and the density classification problem for cellular automata. In: Virtual Worlds 2000, *LNAI*, vol. 1834, pp. 245–255. Springer (2000)
12. Brooks, R.A.: Intelligence without representation. Artificial Intelligence **47**, 139–159 (1991)
13. Cariani, P.: To evolve an ear: epistemological implications of Gordon Pask's electrochemical devices. Systems Research **10**(3), 19–33 (1993)
14. Cook, M.: Universality in elementary cellular automata. Complex Systems **15**(1), 1–40 (2004)
15. Cooper, D., Stepney, S., Woodcock, J.: Derivation of Z refinement proof rules: forwards and backwards rules incorporating input/output refinement. Tech. Rep. YCS-2002-347, Department of Computer Science, University of York (2002)
16. Crutchfield, J.P.: The calculi of emergence. Physica D **75**, 1154 (1994)
17. Crutchfield, J.P., Kaneko, K.: Penomenology of spatio-temporal chaos. In: H. Bin-Lin (ed.) Direction in Chaos, pp. 272–353. World Scientific (1987)
18. Culik II, K., Yu, S.: Undecidability of CA classification schemes. Complex Systems **2**(2), 177–190 (1988)
19. Danks, G., Stepney, S., Caves, L.: Folding protein-like structures with open L-systems. In: ECAL 2007, *LNAI*, vol. 4648, pp. 1100–1109. Springer (2007)
20. Danks, G., Stepney, S., Caves, L.: Protein folding with stochastic L-systems. In: ALife XI, pp. 150–157. MIT Press (2008)
21. Dasmahapatra, S., Werner, J., Zauner, K.P.: Noise as a computational resource. Int. J. Unconventional Computing **2**(4), 305–319 (2006)
22. Ditto, W.L., Murali, K., Sinha, S.: Chaos computing: ideas and implementations. Phil. Trans. R. Soc. A **366**, 653–664 (2008)

23. Drossel, B.: Random Boolean Networks. In: H.G. Schuster (ed.) Reviews of Nonlinear Dynamics and Complexity, vol. 1. Wiley (2008). arXiv:0706.3351v2 [cond-mat.stat-mech]
24. Durand, B., Formenti, E., Varouchas, G.: On undecidability of equicontinuity classification for cellular automata. In: M. Morvan, E. Rémila (eds.) Discrete Models for Complex Systems, DMCS'03, vol. AB, pp. 117–128. DMTCS (2003)
25. Gács, P., Kurdyumov, G.L., Levin, L.A.: One dimensional uniform arrays that wash out finite islands. Problemy Peredachi Informatsii **12**, 92–98 (1978)
26. Giavitto, J.L., Michel, O.: Data structure as topological spaces. In: Unconventional Models of Computation, *LNCS*, vol. 2509, pp. 137–150. Springer (2002)
27. Giavitto, J.L., Michel, O., Cohen, J., Spicher, A.: Computation in space and space in computations. In: UPP 2004, *LNCS*, vol. 3566, pp. 137–152. Springer (2005)
28. Hammel, S.M., Yorke, J.A., Grebogi, C.: Numerical orbits of chaotic processes represent true orbits. Bull. Amer. Math. Soc. **19**(2), 465–469 (1988)
29. Hao, B.L., Zheng, W.M.: Applied Symbolic Dynamics and Chaos. World Scientific (1998)
30. Harding, S.L., Miller, J.F.: A tone discriminator in liquid crystal. In: CEC 2004, pp. 1800–1807. IEEE Press (2004)
31. Harding, S.L., Miller, J.F.: Evolution *in materio*: A real-time robot controller in liquid crystal. In: Proc. NASA/DoD Conference on Evolvable Hardware, pp. 229–238. IEEE Press (2005)
32. Harding, S.L., Miller, J.F., Rietman, E.A.: Evolution in materio: Exploiting the physics of materials for computation. arXiv:cond-mat/0611462 (2006)
33. Hayes, W., Jackson, K.R.: A survey of shadowing methods for numerical solutions of ordinary differential equations. Applied Numerical Mathematics **53**, 299–321 (2005)
34. Holden, A.V., Tucker, J.V., Zhang, H., Poole, M.J.: Coupled map lattices as computational systems. Chaos **2**(3), 367–376 (1992)
35. Hornby, G.S., Pollack, J.B.: Body-brain coevolution using L-systems as a generative encoding. In: GECCO 2001, pp. 868–875. Morgan Kaufmann (2001)
36. Ito, K., Aoki, T., Higuchi, T.: Design of an excitable digital reaction-diffusion system for shortest path search. In: ITC-CSCC2004, Japan (2004)
37. Kaneko, K.: Transition from torus to chaos accompanied by frequency lockings with symmetry breaking. Progr. Theoret. Phys. **69**(5), 1427–1442 (1983)
38. Kaneko, K.: Period-doubling of kink-antikink patterns, quasiperiodicity in antiferro-like structures and spatial intermittency in coupled logistic lattice. Progr. Theoret. Phys. **72**(3), 480–486 (1984)
39. Kaneko, K.: Spatiotemporal intermittency in coupled map lattices. Progr. Theoret. Phys. **74**(5), 1033–1044 (1985)
40. Kaneko, K.: Lyapunov analysis and information flow in coupled map lattices. Physica D **23**, 436–447 (1986)
41. Kanso, A., Smaoui, N.: Logistic chaotic maps for binary numbers generations. Chaos, Solitons & Fractals (2007). In press, doi:10.1016/j.chaos.2007.10.049
42. Kauffman, S.A.: Requirements for evolvability in complex systems. Physica D **42**, 135–152 (1990)
43. Kauffman, S.A.: The Origins of Order. Oxford University Press (1993)
44. Kelso, J.A.S.: Dynamic Patterns. MIT Press (1995)
45. Kitano, H.: Biological robustness. Nature Reviews Genetics **5**, 826–837 (2004)
46. Kocarev, L., Jakimoski, G.: Logistic map as a block encryption algorithm. Phys. Lett. A **289**(4-5), 199–206 (2001)
47. Kuhnert, L., Agladze, K., Krinsky, V.: Image processing using light-sensitive chemical waves. Nature **337**, 244–247 (1989)
48. Kunishima, W., Nishiyama, A., Tanaka, H., Tokihiro, T.: Differential equations for creating complex cellular automaton patterns. J. Phys. Soc. Japan **73**, 2033–2036 (2004)
49. Land, M., Belew, R.K.: No perfect two-state cellular automata for density classification exists. Phys. Rev. Lett. **74**(25), 5148–5150 (1995)
50. Li, T.Y., Yorke, J.A.: Period three implies chaos. Am. Math. Monthly **82**(10), 985–992 (1975)
51. Lorenz, E.N.: The Essence of Chaos. UCL Press (1993)

52. Maturana, H.R., Varela, F.J.: Autopoeisis and Cognition. D. Reidel (1980)
53. May, R.M.: Simple mathematical models with very complicated dynamics. Nature **261**, 459–467 (1976)
54. Melby, P., Kaidel, J., Weber, N., Hübler, A.: Adaptation to the edge of chaos in a self-adjusting logistic map. Phys. Rev. Lett. **84**(26), 5991–5993 (2000)
55. Melby, P., Weber, N., Hübler, A.: Robustness of adaptation in controlled self-adjusting chaotic systems. Fluctuation and Noise Letters **2**(4), L285–L292 (2002)
56. Melby, P., Weber, N., Hübler, A.: Dynamics of self-adjusting systems with noise. Chaos **15**, 033902 (2005)
57. Metropolis, N., Stein, M., Stein, P.: On finite limit sets for transformations on the unit interval. J. Comb. Theory **15**(1), 25–43 (1973)
58. Mills, J.W.: The architecture of an extended analog computer core. In: UCAS-4, Austin, TX, USA (2008)
59. Mills, J.W.: The nature of the extended analog computer. Physica D **237(9)**, 1235–1256 (2008)
60. Mills, J.W., Parker, M., Himebaugh, B., Shue, C., Kopecky, B., Weilemann, C.: "Empty Space" computes: The evolution of an unconventional supercomputer. In: Proc. 3rd ACM Computing Frontiers Conf, pp. 115–126 (2006)
61. Mitchell, M., Crutchfield, J.P., Das, R.: Evolving cellular automata with genetic algorithms: A review of recent work. In: E.D. Goodman, V.L. Uskov, W.F. Punch (eds.) Evolutionary Computation and Its Applications: EvCA'96 (1996)
62. Motoike, I.N., Adamatzky, A.: Three-valued logic gates in reaction-diffusion excitable media. Chaos, Solitons & Fractals **24**, 107–114 (2004)
63. Moulay, E., Baguelin, M.: Meta-dynamical adaptive systems and their application to a fractal algorithm and a biological model. Physica D **207**, 79–90 (2005)
64. Murray, J.D.: Mathematical Biology, 2nd edn. Springer (1993)
65. Měch, R., Prusinkiewicz, P.: Visual models of plants interacting with their environment. In: SIGGRAPH '96, pp. 397–410. ACM (1996)
66. Omohundro, S.: Modelling cellular automata with partial differential equations. Physica D **10**, 128–134 (1984)
67. Ott, E., Grebogi, C., Yorke, J.A.: Controlling chaos. Phys. Rev. Lett. **64**(11), 1196–1199 (1990)
68. Ott, E., Sauer, T., Yorke, J.A. (eds.): Coping with Chaos. Wiley (1994)
69. Pareeka, N.K., Patidara, V., Sud, K.K.: Image encryption using chaotic logistic map. Image and Vision Computing **24**(9), 926–934 (2006)
70. Pask, G.: The natural history of networks. In: M.C. Yovits, S. Cameron (eds.) Self-Organzing Systems. Pergamon Press (1960)
71. Păun, G.: Computing with membranes. J. Comput. System Sci. **61**(1), 108–143 (2000)
72. Phatak, S.C., Rao, S.S.: Logistic map: A possible random-number generator. Phys. Rev. E **51**(4), 3670–3678 (1995)
73. Powley, E.J., Stepney, S.: Automorphisms of transition graphs for elementary cellular automata. Journal of Cellular Automata **4**(2), 125–136 (2009)
74. Powley, E.J., Stepney, S.: Automorphisms of transition graphs for linear cellular automata. Journal of Cellular Automata (2009). In press
75. Prusinkiewicz, P., James, M., Měch, R.: Synthetic topiary. In: SIGGRAPH '94, pp. 351–358. ACM (1994)
76. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer (1990)
77. Rendell, P.: Turing universality of the Game of Life. In: Adamatzky [3], chap. 18
78. Robinson, J.C.: Infinite-dimensional Dynamical Systems: An Introduction to Dissipative Parabolic PDEs and the Theory of Global Attractors. Cambridge University Press (2001)
79. Roselló-Merino, M., Bechmann, M., Sebald, A., Stepney, S.: Classical computing in nuclear magnetic resonance. Int. J. Unconventional Computing (2009). In press
80. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol **2**(12), e424 (2004)

81. Rubel, L.A.: The Extended Analog Computer. Adv. in Appl. Math. **14**, 39–50 (1993)
82. Sielewiesiuk, J., Gorecki, J.: Logical functions of a cross-junction of excitable chemical media. J. Phys. Chem. A **105(35)**, 8189–8195 (2001)
83. Sinha, S., Ditto, W.L.: Dynamics based computation. Phys. Rev. Lett. **81**(10), 2156–2159 (1998)
84. Sinha, S., Ditto, W.L.: Computing with distributed chaos. Phys. Rev. E **60**(1), 363–377 (1999)
85. Sinha, S., Ditto, W.L.: Exploiting the controlled responses of chaotic elements to design configurable hardware. Phil. Trans. R. Soc. A **364**, 2483–2494 (2006)
86. Sloane, N.J.A.: The On-Line Encyclopedia of Integer Sequences (2008). http://www.research.att.com/~njas/sequences/ (accessed 10 November 2008)
87. Spicher, A., Michel, O., Giavitto, J.L.: A topological framework for the specification and the simulation of discrete dynamical systems. In: ACRI 2004, *LNCS*, vol. 3305, pp. 238–247. Springer (2004)
88. Steinbock, O., Tóth, A., Showalter, K.: Navigating complex labyrinths: optimal paths from chemical waves. Science **267**, 868–871 (1995)
89. Stepney, S.: Embodiment. In: D. Flower, J. Timmis (eds.) In Silico Immunology, chap. 12, pp. 265–288. Springer (2007)
90. Stepney, S.: The neglected pillar of material computation. Physica D **237**(9), 1157–1164 (2008)
91. Stepney, S.: Visualising random boolean network dynamics. In: GECCO 2009. ACM (2009)
92. Stepney, S., Polack, F., Turner, H.: Engineering emergence. In: ICECCS 2006, pp. 89–97. IEEE (2006)
93. Strogatz, S.H.: Nonlinear Dynamics and Chaos. Westview Press (1994)
94. Sutner, K.: Universality and cellular automata. In: Machines, Computations, and Universality 2004, *LNCS*, vol. 3354, pp. 50–59. Springer (2005)
95. Teman, R.: Infinite-Dimensional Dynamical Systems in Mechanics and Physics, 2nd edn. Springer (1997)
96. Tokihiro, T., Takahashi, D., Matsukidaira, J., Satsuma, J.: From soliton equations to integrable cellular automata through a limiting procedure. Phys. Rev. Lett. **76**(18), 3247–3250 (1996)
97. Tolmachiev, D., Adamatzky, A.: Chemical processor for computation of Voronoi diagram. Advanced Materials for Optics and Electronics **6**(4), 191–196 (1996)
98. Toth, A., Showalter, K.: Logic gates in excitable media. J. Chem. Phys **103**, 2058–2066 (1995)
99. Ulam, S.M., von Neumann, J.: On combination of stochastic and deterministic processes. Bull. Amer. Math. Soc. **53**(11), 1120 (1947). (abstract 403)
100. Waddington, C.H.: The Strategy of the Genes. Allen and Unwin (1957)
101. Weimar, J.R., Boon, J.P.: Class of cellular automata for reaction-diffusion systems. Phys. Rev. E **49**(2), 1749–1752 (1994)
102. Weyl, H.: Philosophy of Mathematics and Natural Science. Princeton University Press (1949)
103. Winfree, E.: DNA computing by self-assembly. The Bridge **33**(4), 31–38 (2003)
104. Wolfram, S.: Computation theory of cellular automata. Comm. Math. Phys. **96**, 15–57 (1984)
105. Wolfram, S.: Universality and complexity in cellular automata. Physica D **10**, 1–35 (1984)
106. Wolfram, S.: Twenty problems in the theory of cellular automata. Physica Scripta **T9**, 170–183 (1985)
107. Wolfram, S.: Cellular automata fluids: basic theory. J. Stat. Phys. **45**, 471–526 (1986)
108. Wolfram, S.: Random sequence generation by cellular automata. Adv. in Appl. Math. **7**, 123–169 (1986)
109. Wolz, D., de Oliveira, P.P.B.: Very effective evolutionary techniques for searching cellular automata rule spaces. Journal of Cellular Automata **3**(4), 289–312 (2008)
110. Wuensche, A.: Finding gliders in cellular automata. In: Adamatzky [3], chap. 13
111. Wuensche, A., Lesser, M.: The Global Dynamics of Cellular Automata. Addison Wesley (1992)
112. Yates, F.E. (ed.): Self-Organizing Systems: the emergence of order. Plenum (1987)