Computation in Cellular Automata: A Selected Review

Melanie Mitchell
Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM 87501 U.S.A.
email: mm@santafe.edu

September 10, 1996

To appear in Schuster, H. G., and Gramss, T. (editors), *Nonstandard Computation.* Weinheim: VCH Verlagsgesellschaft.

1. Introduction

Cellular automata (CAs) are decentralized spatially extended systems consisting of large numbers of simple identical components with local connectivity. Such systems have the potential to perform complex computations with a high degree of efficiency and robustness, as well as to model the behavior of complex systems in nature. For these reasons CAs and related architectures have been studied extensively in the natural sciences, mathematics, and in computer science. They have been used as models of physical and biological phenomena, such as fluid flow, galaxy formation, earthquakes, and biological pattern formation. They have been considered as mathematical objects about which formal properties can be proved. They have been used as parallel computing devices, both for the high-speed simulation of scientific models and for computational tasks such as image processing. In addition, CAs have been used as abstract models for studying "emergent" cooperative or collective behavior in complex systems. For collections of papers in all these areas, see, e.g., Burks (1970a); Fogelman-Soulie, Robert, and Tchuente (1987); Farmer, Toffoli, and Wolfram (1984); Forrest (1990); Gutowitz (1990); Jesshope, Jossifov, and Wilhelmi (1994); and Wolfram (1986).

In this chapter I will review selected topics related to computation in CAs. The presentation will assume an elementary knowledge of the theory of computation, including formal-language theory and computability.

A CA consists of two components. The first component is a *cellular space*[1]: a lattice of N identical finite-state machines (*cells*), each with an identical pattern of local connections to other cells for input and output, along with boundary conditions if the lattice is finite. Let

[1]CA terminology differs among different authors. The terminology in this chapter will be consistent with most modern work.

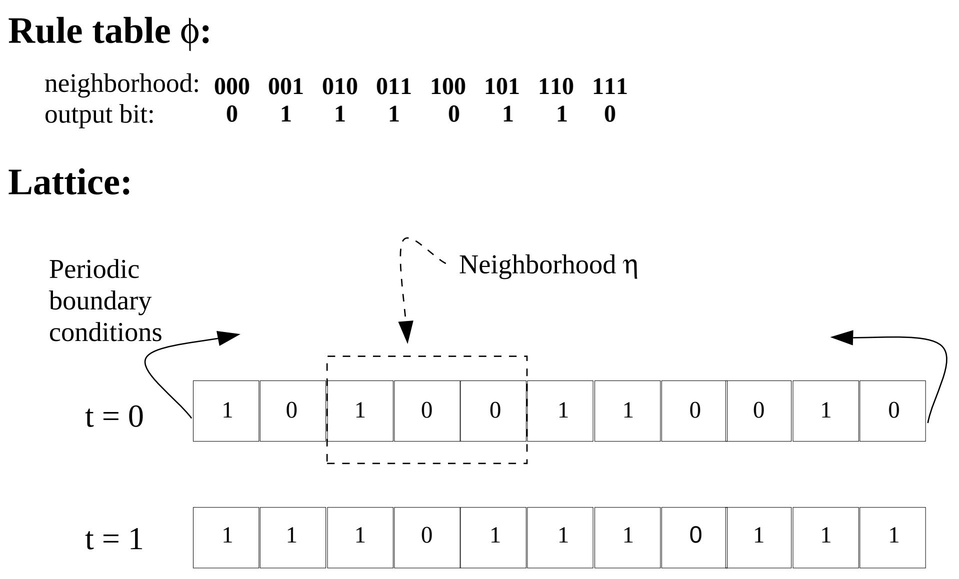

Rule table ϕ:
neighborhood: 000 001 010 011 100 101 110 111
output bit: 0 1 1 1 0 1 1 0
Lattice:
Periodic boundary conditions
Neighborhood η
t = 0
1 0 1 0 0 1 1 0 0 1 0
t = 1
1 1 1 0 1 1 1 0 1 1 1

Figure 1: Illustration of a one-dimensional, binary-state, $r = 1$ CA with periodic boundary conditions shown iterating for one time step. Wolfram (1983) proposed a numbering scheme for one-dimensional k=2 ("elementary") CAs in which the output bits are ordered lexicographically, as in the figure, and are read right-to-left (neighborhood 111 first) to form a binary integer between 0 and 255. In that scheme, the elementary CA pictured here is number 110.

Σ denote the set of states in a cell's finite state machine, and $k = |\Sigma|$ denote the number of states per cell. Each cell is denoted by an index i and its state at time t is denoted s_i^t, where $s_i^t \in \Sigma$. The state s_i^t of cell i together with the states of the cells to which cell i is connected is called the *neighborhood* η_i^t of cell i.

The second component is a *transition rule* (or "CA rule") $\phi(\eta_i^t)$ that gives the update state $s_i^{(t+1)}$ for each cell i as a function of η_i^t.

Typically in a CA a global clock provides an update signal for all cells: at each time step all cells update their states synchronously according to $\phi(\eta_i^t)$.

A one-dimensional, $k = 2$ ($\Sigma = \{0, 1\}$) CA is illustrated in Figure 1. Here, the neighborhood of each cell consists of itself and its two nearest neighbors, and the boundary conditions are periodic—i.e., the leftmost cell is considered to be the right neighbor of the rightmost cell, and vice versa. For one dimensional CAs, the size of the neighborhood η_i (leaving off the t subscript when it is not needed) is often written as $|\eta_i| = 2r + 1$ where r is called the *radius* of the CA. In the one-dimensional case, $\phi : \Sigma^{2r+1} \to \Sigma$. In cases of binary-state CAs where the number of possible neighborhoods is not too large, the CA rule is often displayed as a lookup table (or *rule table*) which lists each possible neighborhood together with its *output bit*, the update value for the state of the central cell in the neighborhood. The 256 one-dimensional, $k = 2, r = 1$ CAs are called *elementary CAs* (ECAs). The one pictured in Figure 1 is ECA 110, according to Wolfram's (1983) numbering scheme.

The behavior of CAs is often illustrated using "space-time diagrams" in which the con-

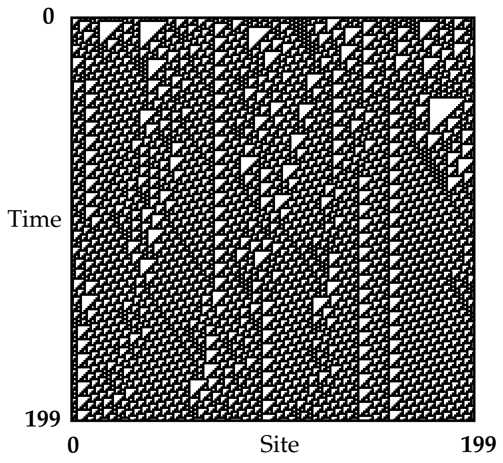


Figure 2: A space-time diagram, illustrating the typical behavior of Elementary CA 110. The lattice, displayed horizontally, starts with a randomly generated initial configuration. Cells in state 1 are displayed as black, and cells in state 0 are displayed as white. Time increases down the page. (This and other space-time diagrams given in this chapter were plotted using the lattice automaton simulation and graphing package la1d, written by James P. Crutchfield.)

figuration of states in the d-dimensional lattice is plotted as a function of time. (Of course, in most cases space-time diagrams are practical only for $d \leq 2$.) Figure 2 gives a space-time diagram of the behavior of ECA 110 on a lattice with $N = 200$, starting from a randomly generated initial configuration (the lattice is displayed horizontally) and iterated over 200 time steps with time increasing down the page.

The ECAs are among the simplest versions of the CA architecture (although, as can be seen in Figure 2, such CAs can give rise to apparently complicated aperiodic behavior). This basic architecture can be modified in many ways—increasing the number of dimensions, the number of states per cell and the neighborhood size; modifying the boundary conditions; making the CA rule stochastic rather than deterministic; and so on.

CAs are included in the general class of "iterative networks" or "automata networks" (see Fogelman-Soulie, Robert, and Tchuente, 1987 for a review of this general class). CAs are distinguished from other examples of this class in their homogeneous and local connectivity among cells, homogeneous update rule across all cells, and (typically) relatively small k.

2. Von Neumann's Self-Reproducing Cellular Automaton

The original concept of cellular automata is most strongly associated with the great scientist and mathematician John von Neumann. According to the history recounted by Burks (1966, 1970b), von Neumann was deeply interested in connections between biology and the (then) new science of computational devices, "automata theory." Paramount in his mind was the biological phenomenon of self-reproduction, and he had posed a fundamental question: "What kind of logical organization is sufficient for an automaton to be able to reproduce

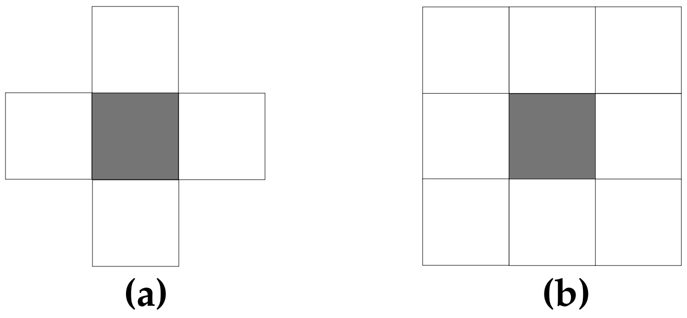


Figure 3: (a) The von Neumann neighborhood. (b) The Moore neighborhood. In both cases, the cell to be updated is shaded.

itself?" The idea of using cellular automata as a framework for answering this question was suggested to von Neumann by Stanislaw Ulam (Burks, 1970b). Thus, the original concept of cellular automata can be credited to Ulam, while early development of the concept can be credited to von Neumann.

Von Neumann strongly believed that a general theory of computation in "complex networks of automata" such as cellular automata would be essential both for understanding complex systems in nature and for designing artificial complex systems. Von Neumann made foundational contributions to such a theory, and it is thus ironic that the standard model of computation, with a CPU, globally accessible memory, serial processing, and so on has been dubbed the "von Neumann style" architecture, and architectures such as cellular automata have been dubbed "non-von Neumann style."

Von Neumann's detailed solution to his question, "What kind of logical organization is sufficient for an automaton to be able to reproduce itself?" was presented in his book *Theory of Self-Reproducing Automata* (von Neumann, 1966). The manuscript was incomplete at the time of von Neumann's death in 1957. The manuscript was edited and completed by Burks, who also gives an excellent detailed overview of von Neumann's system in Essay 1 of his book *Essays on Cellular Automata* (Burks, 1970a).

This question has to be framed carefully so that it does not admit trivial solutions. For example, one can easily design a CA in which 1s reproduce themselves. The self-reproducing automaton must have a certain degree of complexity. Von Neumann required that the automaton in question be equivalent in power to a universal Turing machine.

The self-reproducing automaton that von Neumann constructed, here denoted as M_c, is embedded in a two-dimensional cellular space with a particular CA rule and a particular initial configuration of states. The two-dimensional space is considered to be infinite, and all but a finite number of cells start out in a special "quiescent" state.

There are 29 possible states per cell (including the quiescent state), and the neighborhood of each cell consists of five cells: the cell itself and the four bordering cells to the north, south, east, and west (Figure 3a). This two-dimensional neighborhood is now called the "von Neumann neighborhood." The two-dimensional neighborhood consisting of the cell itself and its 8 neighbors (the von Neumann neighborhood plus the diagonal cells) is called the "Moore neighborhood" (Figure 3b).

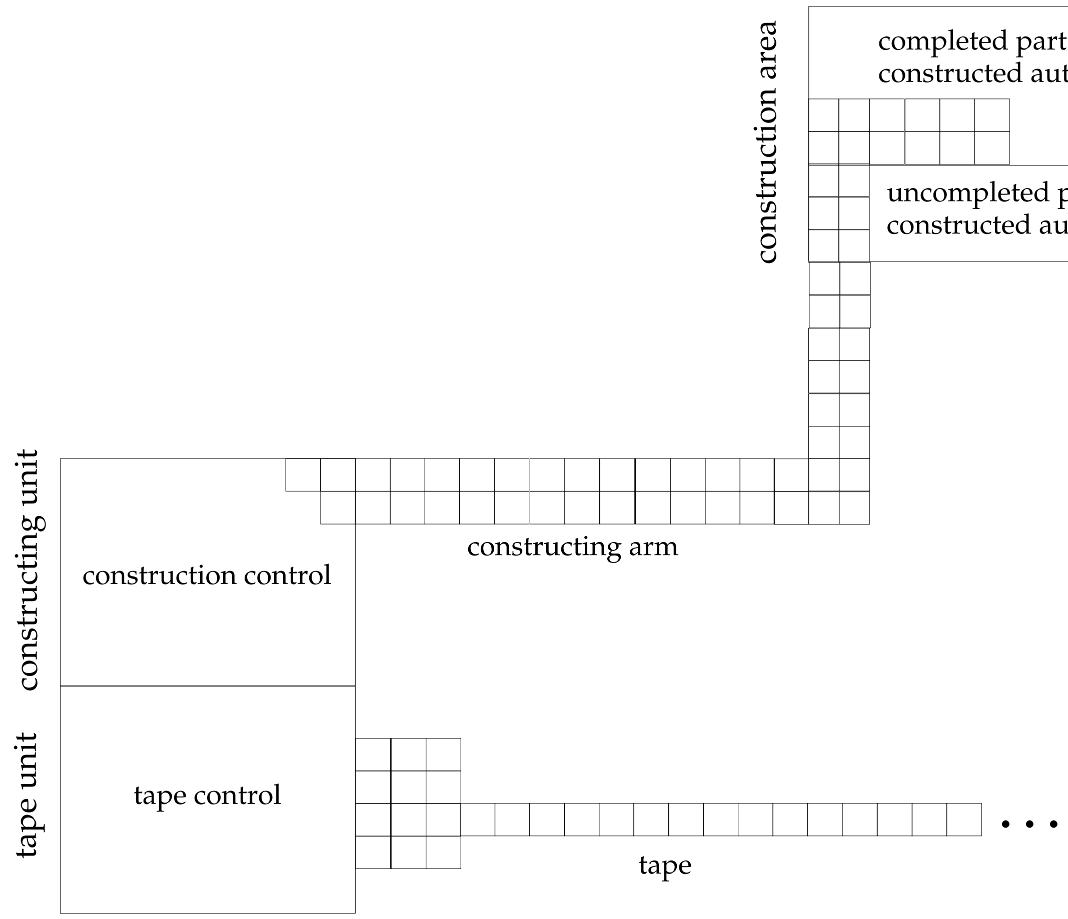


Figure 4: A schematic diagram of the operation of von Neumann's self-reproducing automaton, illustrating the high-level components. (Adapted from Burks, 1970c.)

Figure 4 illustrates schematically the high-level process by which the self-reproducing automaton M_c works. M_c consists of a configuration of states which can be grouped into two functional units: a constructing unit, which constructs the new automaton, and a tape unit, which stores and reads the information needed to construct the new automaton.

The tape unit consists of a tape control and a "tape." The tape is a linear array of cells that contains the information about M, the automaton to be constructed. In particular, it contains (in left-to-right order) (1) the x and y coordinates (x_0, y_0) of the lower left-hand corner of the rectangle in which the new automata is to be constructed (the "construction area"); (2) the width α and height β of that rectangle; and (3) the cell states making up the automaton (listed in reverse order of where they are to be placed in the rectangle) and a asterisk indicating the end of the tape.

The construction of the new automaton M is carried out by the sending of signals (in the form of propagating cell states) between the tape unit and the constructing unit. The constructing unit consists of a construction control and a "constructing arm." As shown in Figure 4, the constructing arm is an array of cell states through which cell states to be constructed can be sent from the construction control to designated places in the construction area.

The original automaton M_c is "turned on" by an external signal sent to the construction control. The construction control then sends a signal to the tape unit, which reads from the tape and sends (via signals) the values of x_0, y_0, and β to the construction control. The construction control then causes the construction "arm"—an expanding and contracting array of cell states—to "move" (by changing the states of intermediate cells) from its initial position to the upper left-hand corner of the construction area. Then the construction control asks for and receives the value of successive states on the tape, and moves the construction arm to place them in their proper position in the construction area.

When construction is complete (i.e., the terminating asterisk is read from the tape), the construction control moves the construction arm to the lower left-hand corner of M and sends the start signal to it, which causes M to begin the self-reproduction process anew. Then M_c moves its construction arm back to its original position.

The above is a rough, high-level description of how von Neumann's self-reproducing automaton works. Von Neumann's solution included both this high-level design and its implementation in a 29-state cellular automaton. The high-level functions invoked above—reading states from the tape, sending different signals, receiving and recognizing different signals, and so on—are all built up in a complicated way by primitive CA operations. At the lowest level, von Neumann used primitive CA operations to synthesize logical operations such as "or," "and," and other simple operations such as delays and signal propagation. These operations were used in turn to synthesize higher-level operations such as signal recognizers, the movable construction arm, and so on. Finally, the higher-level operations were used to put together the entire automaton. It is interesting to note that some of the intermediate-level operations were analogous to the primitive elements von Neumann used in designing the EDVAC—the first "stored program" electronic computer.

It should be clear that M_c is capable of more than self-reproduction—in fact, it can be shown to be a universal constructor, capable of constructing any automaton whose configuration is stored on its tape. (It can also be shown to be capable of universal computation.) Self-reproduction then reduces to the special case where M_c's tape contains a description of Mc itself. It is essential that the constructed automaton be initially "quiescent" (i.e., its states do not change before it receives a start signal). If this were not the case, then M might begin a new construction process before its own construction is completed, and possibly interfere with its own construction process. Von Neumann's automaton is initially quiescent; it does not begin its construction process until it receives an external signal.

Von Neumann insisted that his CA have a reasonably small number of states per cell—otherwise the question of universal computation and universal construction in a CA could be begged by implementing high-level capabilities directly in each cell (with a huge number of states). His 29-state construction was later further simplified by Codd (1968) to a CA with eight states per cell, and later by Banks (1971) to four states per cell. Von Neumann's design itself was implemented on a computer by Pesavento (1996). Langton (1984) studied very simple self-reproducing structures in CAs in order to understand the minimal requirements for non-trivial self-reproduction.

Other types of self-reproducing automata (or computer programs) can be designed—for example, see Hofstadter (1979), Chapter 16, for a discussion of self-reproduction in computer programs and in nature. One commonality between all such systems is the use of the same information in two modes: first as an *interpreted* program that implements a copying engine, and second as *uninterpreted* data to be copied to form the offspring system. In von Neumann's system, the initial configuration of states is interpreted by the CA to implement the copying engine, and also holds the tape containing the uninterpreted initial configuration of states that will make up the copied automaton M. This basic principle, that the same information is used in these two modes, also forms the basis of self-reproduction in molecular biology. It is interesting to note that von Neumann's design of a self-replicating automaton was prior to the discovery of the detailed mechanisms by which DNA replicates itself.

3. Universal Computation in Cellular Automata

3.1 Smith's Two-Dimensional Construction

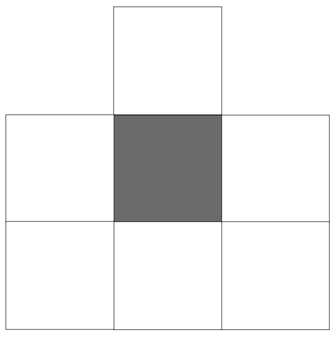
Von Neumann was the first to design a cellular automaton capable of universal computation. Since then many others became interested in the problem and either improved on his original construction or have developed alternative constructions. An early example was Alvy Ray Smith (1971), who constructed a series of progressively simpler cellular automata capable of universal computation. The first in the series was a two-dimensional cellular automaton M_s that simulates a given Turing machine M in "real time"—that is, there is a one-to-one correspondence between the time steps of M_s and the time steps of M.

Smith's construction is quite straightforward. The main cleverness is using the same CA states to represent both tape symbols and states of M. Suppose M has m tape symbols (including the blank symbol) and n internal states. In M_s, the number of states per cell k is one plus the maximum of m and n. The cellular space is an infinite two-dimensional space, with all but a finite number of cells being in a "quiescent" state, denoted by 0. A cell neighborhood consists of 7 cells, with the neighborhood template given in Figure 5a. Without loss of generality, assume that $n < m$. Then $k = m + 1$ (where the $(m + 1)$st state is 0).

The CA works as illustrated in Figure 5b. One row of the space is used to simulate M's tape. At each time step t the cells in this row are in states corresponding to the symbols S_i on M's tape at time step t in M's operation. At any time, all but a finite number of M's tape squares are blank. M_s needs to know the extent of this finite tape area, so the leftmost and rightmost cells of the finite tape are set to M's blank symbol (here called "1"). All other cells outside the current finite tape area are set to the quiescent state 0.

The row of cells above the tape row contains one cell that simulates the tape head (labeled h, containing state P). It is directly above the cell (labeled s) whose symbol S_0 is to be scanned at time t. Two other cells are necessary to label: cells a and b, the cells directly to the left and right of cell h. All cells except the tape cells and h are in the quiescent state.

The actions of M—scan the current tape symbol u in state v, write a new tape symbol p and go into state q, and move either left or right—can be written in the form $(u, v) = pXq$, where X is either L (a left move) or R (a right move). These actions are encoded in M_s's transition rule, part of which is schematically given in Table 1. (This table leaves out some



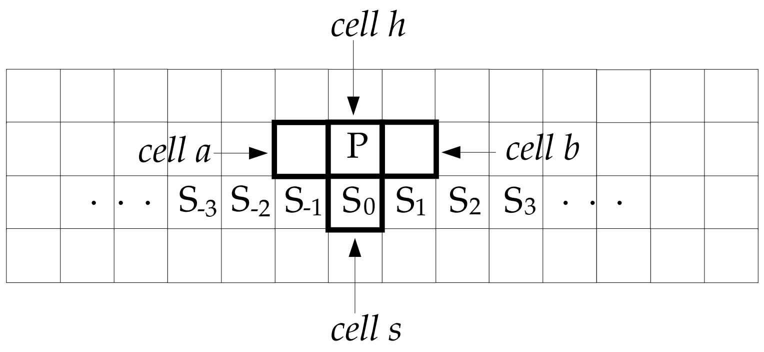


Figure 5: (a) The neighborhood template used in Smith's universal CA. The cell to be updated is shaded. (b). A schematic diagram of the operation of Smith's universal CA. The current cell simulating the tape head is labeled h, with state P. The cells immediately to its left and right are labeled a and b, respectively. The cell currently being scanned is labeled s, and has state S_0. The other non-blank tape cells i have states S_i. (Adapted from Smith, 1971.)

details, such as entries dealing with the ends of the finite tape area; see Smith, 1971 for all details.) The starred symbol in each neighborhood configuration is the symbol to be updated by that transition. For example, the first entry in the table is for the scanned cell s. Assuming its value is u, the tape head's value is v, and M has the action $(u, v) = pXq$, s is assigned the new state p. Likewise, the second entry specifies that the tape head moves by updating the current tape head cell to state 0.

In this way, M_s simulates any given Turing machine M in real time. As a corollary, a CA U_s can be constructed in this way to simulate a universal Turing machine U in real time. (For example, Minsky (1967) described a 6-state, 6-symbol universal Turing machine, so a two-dimensional, 7-state CA can be constructed that simulates it in real time.) In his paper, Smith gives several other variations of the original construction with different number of states, different neighborhood templates, and one-dimensional architectures.

It should be noted that this approach to universal computation in CAs differs from von

Cell c	Neighborhood of c	Next state of c	conditions
s	P	p	$S_0 = u$
	$S_{-1}\ S_0^*\ S_1$		$P = v$
	$0\ \ 0\ \ 0$		
h	0	0	
	$0\ \ P^*\ \ 0$		in all cases
	$S_{-1}\ S_0^*\ S_1$		
a	0	0	if $X = R$
	$0\ \ 0^*\ \ P$	q	if $X = L$
	$S_{-2}\ S_{-1}^*\ S_0$		
b	0	q	if $X = R$
	P $\ 0^*\ \ 0$	0	if $X = L$
	$S_0\ S_1^*\ S_2$		

Table 1: Part of the transition rule for Smith's universal CA, given schematically. A Turing machine move is represented as $(u, v) = pXq$: scan the current tape symbol u in state v, write a new tape symbol p and go into state q, and move X, where X is either L (a left move) or R (a right move). A starred symbol represents the state of cell c, the cell to be updated. (Adapted from Smith, 1971.)

Neumann's in that Smith's construction simulates a particular Turing machine M—a different construction is needed for each different Turing machine—whereas von Neumann's construction can simulate any Turing machine by setting up the initial configuration in the correct way. Von Neumann's construction, on the other hand, does not simulate Turing machines in anything like real time.

Many alternative schemes for simulating Turing machines in cellular automata have been formulated over the years. For example, Lindgren and Nordahl (1990) constructed an $r = 1$, $k = 7$ one-dimensional CA in which tape symbols were represented by stationary cell states (as in Smith's construction) but in which the tape head (with internal state) was represented as a left- or right-moving "particle"—a propagating set of CA states.

3.2 Universal Computation in the Game of Life

A very different approach was used to show that the well-known CA "the Game of Life" is capable of universal computation. "Life" was first invented by John Conway in the 1960s; it is defined and its universality is demonstrated in Berlekamp, Conway, and Guy (1982), Chapter 25. Life is a two-dimensional, binary state CA, where each cell's neighborhood η_i^t is the Moore neighborhood (Figure 3b). The transition rule $s_i^{t+1} = \phi(\eta_i^t)$ is very simple: if $s_i^t = 1$, then $\phi(\eta_i^t) = 1$ if and only if exactly two or three other neighbors are in state 1; otherwise $\phi(\eta_i^t) = 0$. If $s_i^t = 0$, then $\phi(\eta_i^t) = 1$ if and only if exactly three other neighbors are in state 1; otherwise $\phi(\eta_i^t) = 0$. In this discussion, Life is assumed to iterate on an infinite two-dimensional lattice, starting with an initial configuration with a finite number of 1s, and

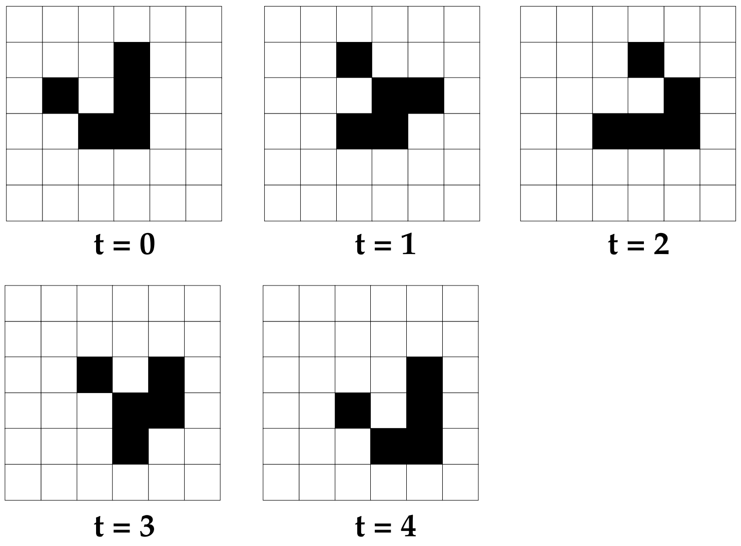


Figure 6: A glider in Life. The structure at $t = 0$ and $t = 4$ is the glider, which moves one square diagonally every four time steps.

all other cells set to 0.

Life is well known because this very simple transition rule often leads to very complicated and interesting patterns in the cellular space. For example, it is very easy to construct initial configurations that will produce simple, propagating, localized structures called "gliders" (Figure 6), out of which more complicated patterns can be formed. Much investigation has gone into discovering other kinds of propagating structures in Life as well (see Berlekamp, Conway, and Guy (1982), Chapter 25 for an overview of some of these structures). Conway asked the question "Can the population [i.e., cells set to 1] of a Life configuration grow without limit?" William Gosper answered the question in 1970 with the invention of a "glider gun," a stationary structure that emits a new glider every 30 time steps. Gliders and glider guns are the key structures used in the construction of a universal computer in Life.

In this construction, rather than simulating a universal Turing machine, basic logical functions are built up from interactions between streams of gliders "shot off" from glider guns. For example, in some cases when two gliders collide, they annihilate. This is used to build a NOT gate, as illustrated in Figure 7a. An input stream A of bits is represented as a stream of gliders, spaced so that the presence of a glider represents a 1 and the absence represents a 0. This stream collides with a perpendicular stream of gliders coming from a glider gun, spaced as indicated in the figure. Since two colliding gliders in this configuration will annihilate, the only gliders in the vertical stream that make it past the horizontal stream are those that "collided" with the absent gliders, or "holes," in the stream—the 0 bits. The resulting vertical output stream is the NOT of the horizontal input stream.

AND and OR gates can be constructed similarly, as illustrated in Figures 7b and 7c. For example, the AND function (Figure 7b) takes two horizontal input streams, labeled A and B, and outputs a horizontal stream with 1s (gliders) corresponding to positions in which both A and B had 1s, and 0s (holes) otherwise. G signifies a glider gun shooting off an upward-moving stream of gliders with no holes. Whenever the B stream contains a 1, there is an

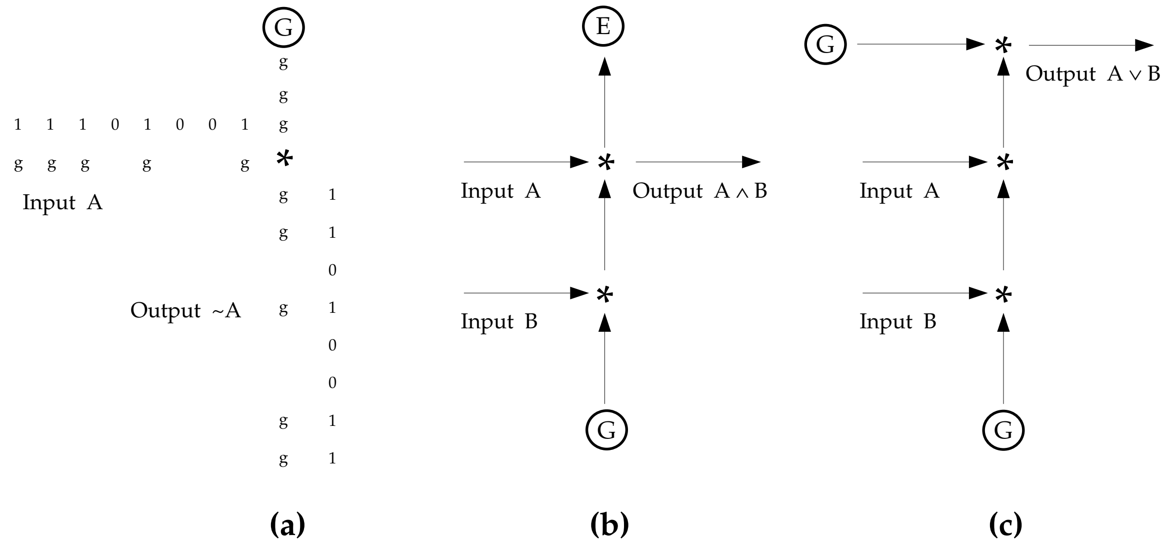


Figure 7: NOT (a), AND (b), and OR (c) gates built out of gliders (labeled g), glider guns (labeled G), and "eaters" (labeled E) in Life. The symbol $*$ indicates a glider collision. (Adapted from Berlekamp, Conway, and Guy, 1982.)

annihilation with the G stream, creating a hole in the G stream. Whenever the B stream contains a 0, the glider in the G stream is allowed to proceed. The A stream is delayed with respect to the B stream just enough so that the hole or glider in G corresponding to the B bit meets the corresponding A bit. A hole in the G stream allows the A bit (hole or glider) to join the output stream. A glider in the G stream always creates a hole in the output stream. It can be seen that this implements the AND function. The part of the vertical G stream that goes past the A stream is consumed by an "eater" (labeled E), a stationary structure that destroys gliders.

The construction also includes clever ways to dispose of unwanted gliders, to copy glider streams, to reposition, delay, and thin glider streams by arbitrary amounts, to store information in the form of circulating glider streams, and to implement auxiliary storage registers via stationary blocks of cells that can be accessed and moved around by glider "fleets." These techniques and the logical functions described above can be used to build up circuits consisting of glider streams that can compute any recursive function. By this means, Life is demonstrated to be universal.

Note that in this construction, space and time resources are considered unbounded. The goal was not to construct a *efficient* computer in Life, or even one that could be practically implemented, but simply to show that in principle Life can compute anything that is computable. This also shows that even CA's (such as Life) with very simple transition rules can be inherently unpredictable—because of Life's universality, there is no general procedure that can predict, say, when an initial configuration will fade away into the all 0s configuration.

Most work on universal computation in CAs consists of devising a CA that can simulate a universal Turing machine or some other computer known to be universal. For example, Margolus (1984) constructed a CA that is closely related to Fredkin and Toffoli's (1982)

"Billiard Ball" model of computation. In contrast, Berlekamp, Conway, and Guy took an already existing CA—Conway's Game of Life—that was interesting for other reasons, and showed that a universal computer could be embedded in it. They went even further in their imaginations, speculating that "It's probable, given a large enough Life space, initially in a random state, that after a long time, intelligent self-reproducing animals will emerge and populate some parts of the space." Whether this is true remains to be seen. An interesting treatment of the Game of Life and its relation to some modern scientific and philosophical problems is given in Poundstone (1985).

In general, universal computation in CAs is interesting only as a proof of principle that this kind of architecture is as powerful as any computer. In practice, none of the constructions of universal computation in CAs is meant to be a practical device that one could actually use to compute something. For one thing, setting up an initial configuration that would result in the desired computation would be extremely difficult. For another, these embedded computers are very slow (even Smith's, which simulates a universal Turing machine in real time) compared with any practical device, and even more so when one realizes that a massively parallel system (a CA) is being used to simulate a slow, serial device.

4. Dynamics and Computation in Cellular Automata

Several researchers have been interested in the relationships between the generic dynamical behavior of cellular automata and their computational abilities, as part of the larger question of relationships between dynamical systems theory and computation theory. Viewing cellular automata as discrete, spatially extended dynamical systems, Stephen Wolfram (1984) proposed a qualitative classification of CA behavior roughly analogous to classifications in dynamical systems theory. (The analogy is rough since concepts such as "chaos," "attractor," "bifurcation," "sensitive dependence on initial conditions," are rigorously defined for continuous-state continuous-time dynamical systems, whereas CAs are discrete-state, discrete-time systems.) Wolfram's four classes of CA behavior are:

> *Class 1:* Almost all initial configurations relax after a transient period to the same fixed configuration (e.g., all 1s).
>
> *Class 2:* Almost all initial configurations relax after a transient period to some fixed point or some temporally periodic cycle of configurations, but which one depends on the initial configuration. (It should be pointed out that on finite lattices, there is only a finite number (2^N) of possible configurations, so all rules ultimately lead to periodic behavior. Class 2 refers not to this type of periodic behavior but rather to cycles with periods much shorter than 2^N.)
>
> *Class 3:* Almost all initial configurations relax after a transient period to chaotic behavior. (The term "chaotic" here refers to apparently unpredictable space-time behavior.)
>
> *Class 4:* Some initial configurations result in complex localized structures, sometimes long-lived. Li and Packard (1990) claimed that ECA 110 (Figure 2), for example, has typical class 4 behavior.

Wolfram (1984) speculated that all class 4 CAs (except the ECAs) are capable of universal computation, and that it could be implemented in a way similar to that described above for Life—by building circuits out of propagating localized structures such as gliders. However, since there can be no general method for proving that a given rule is or is not capable of universal computation, this hypothesis is impossible to verify.

Subsequent to Wolfram's work, several researchers have asked how static properties of CA rules relate to the dynamical behavior of the CAs. Christopher Langton, for example, studied the relationship between the "average" dynamical behavior of cellular automata and a particular statistic (λ) of a CA rule table (Langton, 1990). For binary-state CAs, λ is simply the fraction of 1s in the output bits of the rule table. For CAs with $k > 2$, λ is defined as the fraction of non-quiescent states in the rule table, where one state is arbitrarily chosen to be quiescent.

Langton performed a number of Monte Carlo samples of two-dimensional CAs, starting with $\lambda = 0$ and gradually increasing λ to $1 - 1/k$ (i.e., the most homogeneous to the most heterogeneous rule tables). Langton used various statistics such as single-site entropy, two-site mutual information, and transient length to classify CA "average" behavior at each λ value. The notion of "average behavior" was intended to capture the most likely behavior observed with a randomly chosen initial configuration for CAs randomly selected in a fixed-λ subspace. These studies revealed some correlation between the various statistics and λ. The correlation is quite good for very low and very high λ values. However, for intermediate λ values in finite-state CAs, there is a large degree of variation in behavior.

Langton claimed on the basis of these statistics that as λ is incremented from 0 to $[1 - 1/k]$, the average behavior of CAs undergoes a "phase transition" from ordered behavior (fixed point or limit cycle after some short transient period) to chaotic behavior (apparently unpredictable after some short transient period). As λ reaches a "critical value" λ_c, the claim is that rules tend to have longer and longer transient phases. Additionally, Langton claimed that CAs close to λ_c tend to exhibit long-lived, "complex"—non-periodic, but non-random—patterns. Langton proposed that the λ_c regime roughly corresponds to Wolfram's class 4 CAs.

Langton further hypothesized that CAs able to perform complex computations will most likely be found in this regime, since complex computation in cellular automata requires sufficiently long transients and space-time correlation lengths. A review of this work that is skeptical about the relationships between λ and dynamical and computational properties of CAs is given in Mitchell, Crutchfield, and Hraber (1994a).

Packard attempted to experimentally test Langton's computation hypothesis by using a genetic algorithm (GA) to evolve CAs to perform a particular computation (Packard, 1988). He interpreted the results of his experiment as showing that the GA tends to select CAs with λ close to λ_c—i.e., the "edge of chaos." This experiment was widely interpreted as supporting the hypothesis that cellular automata capable of complex computation are most likely to be found near λ_c. However, my colleagues and I were unable to replicate these results, and were able to argue from empirical and theoretical grounds that Packard's result was almost certainly an artifact of mechanisms in the particular GA that was used rather than a result of any computational advantage conferred by λ_c regions (Mitchell, Hraber, and Crutchfield, 1993).

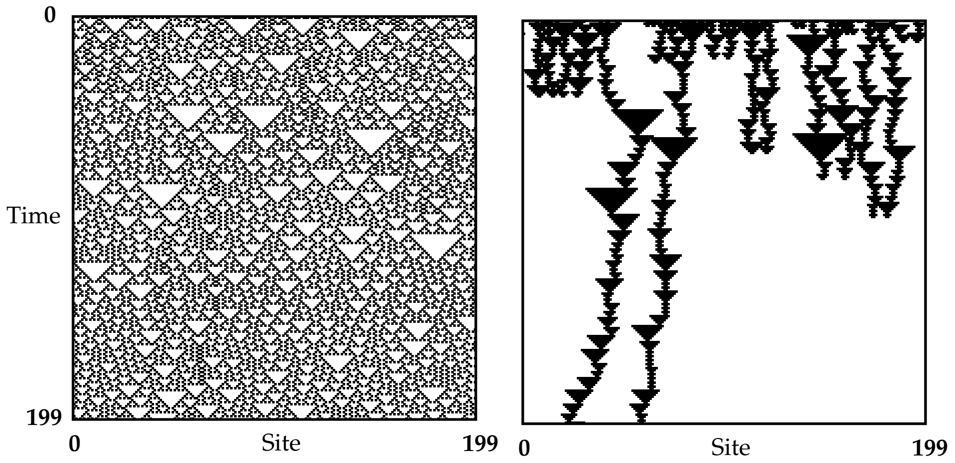


Figure 8: (a) Space-time diagram illustrating the typical behavior of ECA 18, a "chaotic" rule. (b) The same diagram with the regular domains filtered out, leaving only the embedded particles.

Those negative results did not disprove the hypothesis that computational capability can be correlated with phase transitions in CA rule space; they showed only that Packard's results did not provide support for that hypothesis. Relationships between computational capability and phase transitions have been found in other types of dynamical systems. For example, Crutchfield and Young (1989, 1990) looked at the relationship between the dynamics and computational structure of discrete time series generated by the logistic map at different parameter settings. They found that at the onset of chaos there is an abrupt jump in computational class of the time series, as measured by the formal-language class required to describe the time series. This result demonstrated that a dynamical system's computational capability—in terms of the richness of behavior it produces—is qualitatively increased at a phase transition.

However, in the context of cellular automata, our results made it clear that, fixing r and k, any particular task is likely to require CAs with a particular range of λ values for good performance, and the particular range required is a function only of the particular task, r, and k rather than any intrinsic properties of regions of CA rule space. A discussion of the problems with Langton's and Packard's experiments and, more generally, with the "edge of chaos" hypotheses is given in Mitchell, Crutchfield, and Hraber (1994a).

A different approach to understanding relationships between dynamical systems theory and computation in CAs was taken by James Hanson and James Crutchfield (1992; Crutchfield and Hanson, 1993; Hanson, 1993). They noted that attempts like Wolfram's and Langton's to classify a given CA rule in terms of its generic behavior is problematic, since for many rules there is no "generic" behavior either across initial configurations or even for the same initial configuration—e.g., there can be different dynamics going on in different parts of the lattice. Instead, they developed techniques for classifying the different patterns that show up in CA space-time behavior. In particular, they applied Crutchfield's "computational mechanics" framework (Crutchfied, 1994) to the classification of such patterns. Their idea was, given a space-time configuration formed by a CA, to discover an appropriate "pattern basis"—a representation in terms of which the configuration could be understood. Once such a pattern basis is found, the part of the configuration fitting the basis can be seen as forming a background against which coherent structures (defects, walls, singularities, etc.) that do not fit the basis evolve. In this way, coherent structures are identified and their dynamics can be understood. This approach includes both the identification of patterns—finding the most appropriate pattern basis—as well as an analysis of the dynamics in terms of the embedded coherent structures. It is important that this approach work both for ordered configurations and for apparently structureless ("chaotic" or "turbulent") configurations. An example of a turbulent configuration is given in Figure 8a, a space-time diagram of the typical behavior of ECA 18 (which is classified as class 3 under Wolfram's scheme).

To understand the patterns formed by CAs, Hanson and Crutchfield combined tools from computation theory and from nonlinear dynamics. Computation theory comes in because Hanson and Crutchfield define their pattern bases in terms of formal languages and their corresponding automata. For example, ECA 18 has a single pattern basis $\Lambda_{18} = (0\Sigma)^*$. That is, every other site is a zero; the remaining sites can be either 0 or 1. (Such a formal-language description of a set of configurations could possibly be discovered automatically via a "machine reconstruction" algorithm, Crutchfield, 1994).

Crutchfield and Hanson call the space-time regions that conform to the regular-language pattern basis "regular domains." A regular domain is a space-time regions whose spatial configurations consist of words in the regular-language pattern basis. A regular domain must be both temporally invariant—the CA always maps a configuration in the regular domain to another configuration in the same regular domain—and spatially homogeneous—roughly, the graph representing the finite automaton corresponding to the regular language is strongly connected (Hanson and Crutchfield, 1992).

Once the regular domains of a space-time configuration are discovered (either by eye or by an automated induction method), those regions are considered to be understood—their dynamics consists of being mapped to the same pattern basis at each time step. Those regions can then be filtered out of the space-time diagram, leaving only the deviations from regular domains, which are called "embedded particles," whose dynamics can be studied. (These differ from the explicitly designed propagating structures called "particles" in some research on CAs—see below.)

In ECA 18, there is only one stable regular domain—Λ_{18}—but some regions have 0s on even sites and some on odd sites. The boundaries between these "phase-locked" regions are called "defects" and can be thought of as the embedded particles of the system. However, particles defined in this way will be in different places if the configuration is read from left-to-right than if it is read from right-to-left. Instead, Hanson and Crutchfield define blocks of the form $1(00)^n1, n = 0, 1, 2, \ldots$, to be particles since these contain both the left-to-right and the right-to-left defects and the cells in between. Defining the particles in this way is more symmetric and does not lose any information about the defects. Figure 8b shows a filtered version of Figure 8a, in which all the particles are colored black, and all other cells are colored white. For random initial configurations, these particles have been shown to follow a random walk in space-time, and annihilate in pairs whenever they intersect (see Eloranta and Nummelin, 1992, Hanson and Crutchfield, 1992, and Eloranta, 1994).

In short, Rule 18 can be understood of consisting of regular domains defined by the regular language Λ_{18}, and of particles that diffuse randomly and pair-annihilate. Thus a chaotic configuration, such as that of Figure 8a can be understood structurally in a much more informative way than simply classifying it as "chaotic" or "turbulent."

Since this understanding is based on notions from computation theory, Crutchfield and Hanson call the regular domain and particle structure of a CA's behavior its "intrinsic computation." This term refers to structures that emerge in a system's behavior that can be understood in computational terms, even in the absence of any interpretation of what computation is taking place. Crutchfield and Hanson speculate that the complexity of the intrinsic computation gives an upper bound on the complexity of possible "useful" computations that the system can be made to do, where "useful" refers to input/output behavior that is desired by some outside observer, or is needed by the system itself for its "survival" in some environment. For example, in the case of ECA 18, they speculate that the CA cannot be made to do any computation more complex than generating Λ_{18} and generating random diffusion. Note that understanding the CA's intrinsic computation requires more than simply examining space-time configurations—it requires the induction of an appropriate set of pattern bases, and the subsequent discovery of particles that form boundaries between different pattern bases or dislocations between different phases of the same pattern basis. In Section 8 I will describe an application of this framework to describing useful computation by CAs.

Hanson and Crutchfield linked their analysis of intrinsic computation in CAs to nonlinear dynamics by analyzing the "attractor-basin portrait" of ECA 18 (Hanson and Crutchfield, 1992), which gave additional insight into the dynamics of the CA's particles, and showed that small perturbations in the configurations do not appreciably change the CA's behavior or intrinsic computational complexity.

In contrast to work on universal computation in CAs and to general relationships between dynamics and computation such as those described above, much CA research is on how CAs can be used as fast, practical, parallel computers. The next several sections will describe some efforts in this direction.

5. The Firing Squad Synchronization Problem

The firing squad synchronization problem (FSSP) was an early and well-known problem for one-dimensional cellular automata. The first published statement of the problem was a note by Moore (1964), which credited the original statement to Myhill in 1957 and the first solution to McCarthy and Minsky, all apparently unpublished. The problem is as follows: Consider a one-dimensional $r = 1$ CA in which there are special leftmost and rightmost cells. In keeping with the firing squad analogy, the leftmost cell is called the "general" and all the other cells are called "soldiers." The general can receive input from outside the system.

At time step 0, all cells are in the quiescent state. At some time step $t = t_g$, the general (responding to an external input) goes into a special state, interpreted as a "command to fire." Then at some later time step $t = t_f$, all of the soldier cells must go into the "firing" state, and none of them can have been in the firing state at any previous time step. The problem is to devise states and state transitions for the soliders that will accomplish this behavior.

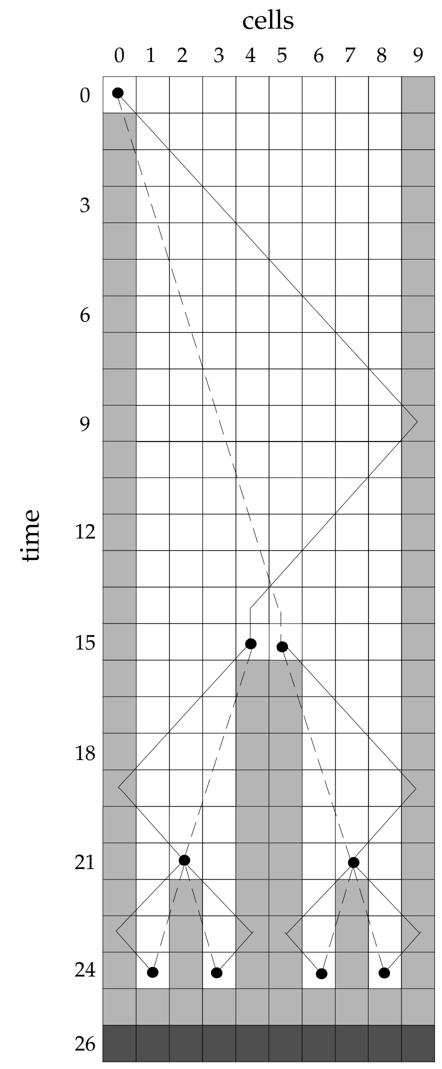


Figure 9: Schematic illustration of one solution to the firing squad synchronization problem, adapted from Fischer (1965). Signal A (solid line) travels at velocity 1, and signal B (dashed line) travels at velocity 1/3. "General" cells contain black dots and boundary cells are shaded.

The original motivation for the FSSP was the problem of how to get all parts of a self-reproducing automaton (like the one devised by von Neumann) to turn on simultaneously when only local communication is possible. Many solutions to the FSSP have been proposed. A simple one, described by Fischer (1965), is illustrated in the schematic space-time diagram given in Figure 9. In this figure, $N = 10$.

Once the general goes into the "command to fire" state (here at time step 0), it sends out two signals (in the form of propagating cell states), signal A (solid line) traveling with velocity 1 (one cell per time step) and signal B (dashed line) traveling with velocity 1/3. It then acts as a "boundary" cell (shaded). The rightmost cell also acts as a boundary cell after time step 0.

When A reaches a boundary cell, it is reflected at a perpendicular angle. This means that the A and B signals will reach the cell at the center of the lattice (or adjoining cells on even-sized lattices) simultaneously. When this happens, the central cell or cells at which A and B meet become "generals" (signified by black dots) which send out new A and B signals and then become boundary cells, as illustrated in the figure. This same signaling pattern and subsequent halving of the lattice is repeated recursively until all cells are boundary cells, and at the following time step all cells go into the firing state (darker shading). This strategy works for any N and the time from the original command to fire to all cells firing will be $3N - 4$. (Fischer, 1965, developed a similar approach, using similar signals and the "sieve of Eratosthenes" to construct a one-dimensional CA that generates primes; in particular, the CA generates a binary sequence in which the tth bit is 1 if and only if t is prime.)

It can be shown that the minimum possible time to fire is $2N-2$. A solution achieving this minimal time was first proposed in an unpublished manuscript by E. Goto (Moore, 1964). The solution described above can be implemented with 15 states per cell (Fischer, 1965). A fair amount of research has gone into finding minimal-time solutions with a minimum number of states (e.g., Waksman, 1966; Balzer, 1967; Mazoyer, 1987; Yunes, 1994) and in finding solutions to generalizations of the FSSP (e.g., Moore and Langdon, 1968; Shinahr, 1974; and Culik, 1989). The large literature on this problem indicates the degree of interest on how to perform synchronization efficiently in decentralized parallel systems. For an overview of work on this problem, see Mazoyer (1988).

The description above was in terms of propagating signals rather than a CA rule, but it is not too difficult to translate this high-level description into such a rule. This kind of high-level description, in which signals and their interactions are the primitive building blocks, is very useful for designing and understanding computation in cellular automata; such descriptions will be given in the sections below. However, very often such high-level descriptions obscure the complexity of the underlying CA rule, in terms of the number of states or the neighborhood size required to implement the desired higher-level signaling.

6. Parallel Formal-Language Recognition by Cellular Automata

The study of formal-language recognition and generation has been a cornerstone of theoretical computer science, and the investigation of computational architectures such as CAs often begins with studies of their formal-language recognition abilities. Several researchers have looked into this question for CAs (e.g., Cole, 1969; Smith, 1972; Pecht, 1983; Seiferas,

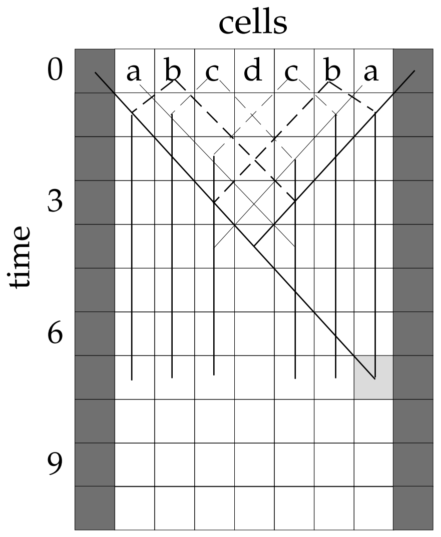


Figure 10: Schematic illustration of Smith's $r = 1$ bounded CA that recognizes the context-free language L_1 (consisting of palindromes) in real time. The lightly shaded rightmost cell at $t = 7$ indicates acceptance of the input.

1977; Sommerhalder and van Westrhenen, 1983; Bucher and Culik, 1984; Choffrut and Culik, 1984; Terrier, 1994), both for theoretical reasons and for the prospect of using CAs as parallel pattern-recognizers. In this section I will review some of the work by Alvy Ray Smith, whose interest was in understanding how to exploit the parallelism of CAs for formal-language and pattern recognition.

Smith (1972) studied one-dimensional $r = 1$ "bounded CAs" (BCAs)—CAs with two special "boundary" cells. The rightmost non-boundary cell was designated to be the "accept" cell. A BCA is said to accept language L if, for all words x in L, there is a time t such that the accept cell goes into an accept state, and for all words x not in L, there is no such time t. A BCA is said to accept language L in real time if, for any word x of length n, the CA can determine that x is in L within n steps.

Smith proved that the class of languages that can be accepted by BCAs is the class of context-sensitive languages, and concluded that "pattern recognition by [such] cellular automata reduces to problems in the theory of context-sensitive languages."

Smith described recognition algorithms in terms of propagating signals, similar to those in the description of the FSSP solution above. He gave as one example the problem of recognizing the context-free language (here called L_1) consisting of all palindromes—words that are the same read forwards as backwards. Figure 10 gives a schematic space-time diagram illustrating the bounded CA constructed by Smith that recognizes L_1 in real time. In Figure 10, the input is the palindrome *abcdcba*. Each cell, including the (shaded) boundary cells, sends out a signal containing its state in both directions at velocity 1. When two signals that do not carry the same state arrive at the same cell, a "no" signal is sent with zero velocity indicating that that cell did not contain the center of the palindrome (solid vertical lines in Figure 10. When the two boundary signals intersect in the center of the lattice, if there

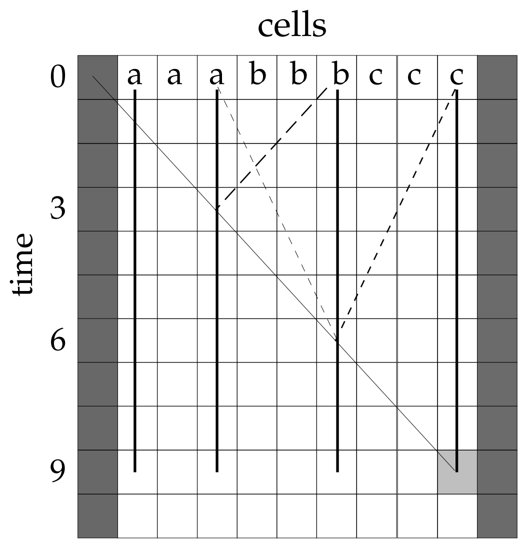


Figure 11: Schematic illustration of Smith's bounded CA that recognizes the context-sensitive language $L_2 = \{a^m b^m c^m | m \geq 1\}$ in real time. The lightly shaded rightmost cell at $t = 9$ indicates acceptance of the input.

center cell does not contain the "no" signal, then the right boundary signal reflects back to the right at velocity 1, and when it arrives at the rightmost (non-boundary) cell, it causes that cell to go into the "accept" state (light shading). If the center cell did contain the "no" signal, then the input would be judged not to be a palindrome, and the rightmost cell would not go into the accept state. (In Figure 10, N is odd. When N is even, two signals are considered to intersect when they occupy adjacent cells, and if two intersecting signals do not carry the same state, the "no" signal is sent at zero velocity by the rightmost of the two cells.)

A second example is a bounded CA that recognizes the context-sensitive language $L_2 = \{a^m b^m c^m | m \geq 1\}$ in real time. Smith's construction is illustrated in Figure 11. The input is $aaabbbccc$, which is in L_2. Let B denote the boundary-cell state. Each cell sends a zero-velocity signal (not shown in the figure) indicating its initial state. In addition, special zero-velocity signals (solid vertical lines in Figure 11) are sent to indicate any Ba boundaries, ab boundaries, bc boundaries, and cB boundaries. A signal from the Ba boundary is sent at velocity 1 to the right, and checks for all a's. A signal from the bc boundary is sent at velocity 1 to the left, and checks for all b's. Both signals stop either when they encounter a wrong letter or when they encounter a zero-velocity boundary signal. If the left-moving Ba-boundary signal encounters only a's, the right-moving bc-boundary signal encounters only b's, and if they reach the ab boundary at the same time, then the input is guaranteed to be of the form $a^m b^m w$ for some suffix w. If this is the case, the right-moving Ba signal continues moving right at velocity 1.

At the same time as this process is going on, a signal is sent from the ab boundary to the right, moving at velocity 1/2 and checking for all b's, and a signal is sent from the cB boundary to the left at velocity 1/2, checking for all c's. Both signals stop either when they encounter a wrong letter or when they encounter a zero-velocity boundary signal. If the *ab* signal encounters only *b*'s and the *cB* signal encounters only *c*'s, and if they reach the *bc* boundary at the same time, then the input is guaranteed to be of the form $w'b^nc^n$ for some prefix w'. Furthermore, if they reach the *bc* boundary at the same time as the right-moving *Ba* signal, then the input is guaranteed to be of the form $a^mb^mc^m$, and the right-moving *Ba* signal proceeds to the rightmost non-boundary cell, which goes into the accept state.

As was the case for the FSSP, the above descriptions of CA formal-language recognizers are given in terms of propagating and intersecting signals, not in terms of CA rules. "Compiling" these high-level description into CA rules takes some thought. For example, multiple signals can occupy the same cell simultaneously; this requires allowing enough states per cell to encode the multiple signals, and then determining the correct transitions for each possible configuration. This can require a large number of states per cell. Smith was not concerned with designing CA rules that can recognize formal languages with a minimum number of states; rather, his purpose was to show that, in principle, there exist bounded CAs in which parallelism can be used so that a large class of languages can be recognized in real time. He compared the language-recognition capability of such CAs with certain other parallel architectures (e.g., "bounded iterative automata" and multitape Turing machines), and proved a number of results showing that the class of bounded CAs is faster in many cases. He also gave several other clever constructions of bounded CAs that can recognize formal languages in real time.

Many of Smith's results on language recognition have been generalized to CAs of more than one dimension by Seiferas (1977) and by Pecht (1983).

7. Parallel Arithmetic by Cellular Automata

In addition to their application as parallel formal-language recognizers, CAs have been investigated as parallel devices for performing arithmetic operations. In this section I describe work in this area by Kenneth Steiglitz and his colleagues, who use various "interacting particle" methods to perform parallel arithmetic in CAs. (Other work on parallel arithmetic in cellular automata has been done by Sheth, Nag, and Hellwarth, 1991; Clementi, De Biase, and Massini, 1994; and Mazoyer, 1996, among others.) (Their notion of particle differs from that of Hanson and Crutchfield as described above. In the next section, their work will be contrasted with "particle computation" methods based on Hanson and Crutchfield's framework.)

In one study, Steiglitz, Irfan Kamal, and Arthur Watson (1988) designed a particular class of CAs—the one-dimensional, binary-state, "parity-rule filter automata"—to perform parallel arithmetic. This class of automata has the property that propagating periodic structures often act as "solitons"—that is, they can pass through each other in space-time without destroying each other, but only by shifting each others phase. It turns out that such a feature can be useful for implementing arithmetic operations in CAs via particle interactions.

One-dimensional filter automata (FAs) differ from standard CAs in that their cells are updated asynchronously from left to right: given radius r, a cell i is updated using the neighborhood,

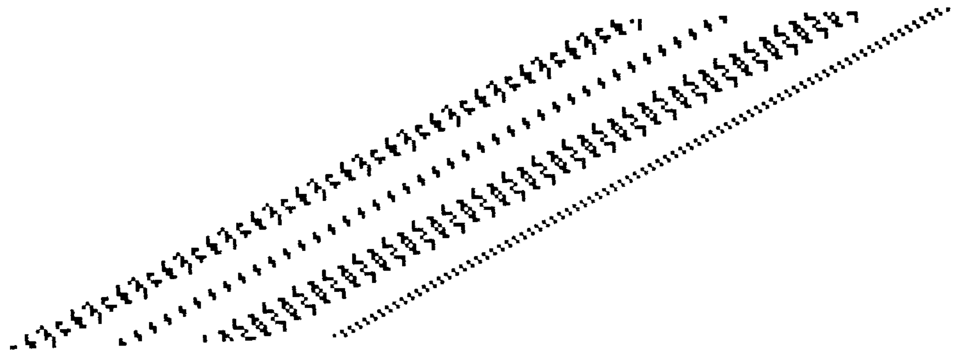


Figure 12: Space-time diagram displaying four typical particles supported by the $r = 5$ BPFA. (Reprinted from Steiglitz, Kamal, and Watson, 1988.)

$$\eta_i^t = s_{i-r}^{t+1} s_{i-r+1}^{t+1} \ldots s_i^t s_{i+1}^t \ldots s_{i+r}^t.$$

In other words, cell i's neighborhood consists of the states of the already updated r cells to the left, cell i itself, and the not-yet-updated r cells to the right. Although this architecture loses the parallelism of CAs (in which all cells are updated simultaneously), implementations of FAs can retain some degree of parallelism by simultaneously updating the cells along a space-time diagonal frontier from the upper right to the lower left (Steiglitz, Kamal, and Watson, 1988).

A binary-state parity-rule FA (BPFA) is defined by the update rule,

$$s_i^{t+1} = \begin{cases} 1 & \text{if } \eta_i^t \text{ has a positive, even number of 1s} \\ 0 & \text{otherwise.} \end{cases}$$

The BPFAs are parameterized by the radius r. The lattice will be thought of as infinite, but the initial configurations will contain only a finite number of nonzero sites.

Steiglitz, Kamal, and Watson defined a "particle" to be a particular kind of periodic sequence in the space-time behavior of a parity-rule FA (see Figure 12). It has been shown that in a BPFA, every configuration with a finite number of nonzero sites will evolve to a periodic sequence, which can be decomposed into particles (Steiglitz, Kamal, and Watson, 1988). It has also been shown that in these FAs particles move only to the left, never to the right.

Steiglitz, Kamal, and Watson devised an algorithm for enumerating all particles of period p that could be formed and could propagate in a BPFA with a given radius r. They also devised a scheme in which information could be encoded in a particle's "phase state"—a combination of its periodic phase and its displacement with respect to the left boundary of the lattice. They were then able to empirically construct tables giving the results of collisions between pairs of particles as a function of their phase states.

All this enabled Steiglitz, Kamal, and Watson to implement a "carry-ripple adder" using $r = 5$ BPFA particles, as illustrated in Figure 13. The two addends n and m are represented

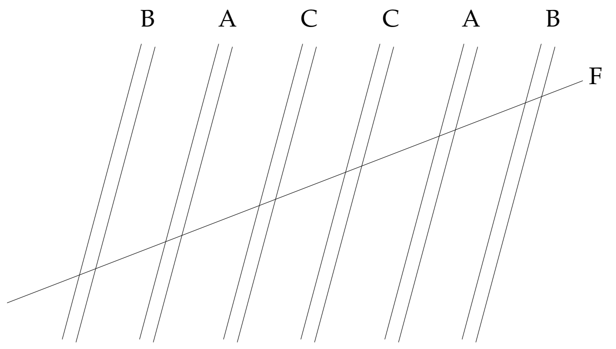


Figure 13: Schematic diagram of the operation of the carry-ripple adder of Steigliz, Kamal, and Watson. The "particle-bundles" (pairs of particles) encode the bits of the two addends, and are labeled A (0,0), B (0,1 or 1,0), and C (1,1). The fast particle, labeled F, propagates the carry bit. (Adapted from Steiglitz, Kamal, and Watson, 1988.)

as equal-length binary numbers (with zeros padding the front of the shorter number). The jth pair (n_j, m_j) of corresponding bits in n and m are represented in the BPFA as a pair of parallel particles (a "particle bundle"), labeled A, B, or C in the figure. Particle bundle A represents the case where $n_j = m_j = 0$; B represents the case where $n_j = 0, m_j = 1$ or vice versa; and C represents the case where $n_j = m_j = 1$. The particle bundles are arrayed in order of j from left to right, with the most significant bit on the left. A "fast particle," labeled F in the figure, travels through A, B, and C, and the collisions effect the binary-addition operation, with the resulting particle bundles encoding the result for each added pair, and the F particle encoding the current carry bit. (It is required here that the speeds of the particles do no change after collisions, a requirement that is not hard to implement in BPFAs.) The final F and the final particle bundles transmit the results of the addition to the leftmost cell, where they are read off by some external interpreter.

The trick to implement this carry-ripple adder was to find a set of particles that have the desired behavior. Steiglitz, Kamal, and Watson sketched a graph-search algorithm (not described here) that searches collision tables for particles implementing any desired logical operation via collisions between a fast particle and slow-particle bundles, as in the carry-ripple addition scheme described above.

Steiglitz, Kamal, and Watson's construction was meant to be an example of how simple parallel arithmetic can be implemented in cellular-automaton-like devices. Other simple logical operations besides addition could presumably be implemented in a similar way. Their work differs from most of the constructions I have described in previous sections in that it is meant to be a practical method for parallel computation rather than a proof of principle or a theoretical result. All the algorithms they described (e.g., enumerating particles and searching collision tables) were actually implemented and were analyzed in terms of complexity to show that this is a feasible approach. The next step is to devise automated methods to efficiently combine these simple arithmetic operations into more complicated desired computations, and to demonstrate that these achieve the requisite parallelism and accompanying speed-up. This is likely to be difficult—as Squier and Steiglitz comment in a later paper, "the behavior of the particles is very difficult to control and their properties only vaguely understood." (Squier and Steiglitz, 1994).

In their later study, Richard Squier and Kenneth Steiglitz (1994) described a possibly more general and more easily programmable approach to computation in CAs using colliding particles. Their "particle machine" approach is closely related to lattice gases, an application of cellular automata to the simulation of fluid dynamics, in which "particles" are primitive cell states rather than more complex space-time structures and local rules are given directly in terms of particle movements and particle collisions and reactions. (For overviews of research on lattice gases, see, e.g., Doolen, 1990; Toffoli and Margolus, 1987; and Lawniczak and Kapral, 1996).

Squier and Steiglitz's particle machines are one-dimensional, bounded binary-state CAs. A certain number n of particle-types is defined ahead of time by the designer—the idea is to define enough types so as to be able to use this same CA for many different computations. The state of each cell encodes an occupancy vector which says which particle type is currently contained by the cell (the number of states per cell is n, the number of particle types). The transition table gives, for every configuration of particles in a neighborhood, what the next state is for the center cell of that neighborhood.

To perform a computation, particles are "injected" by an outside agent at different speeds at the ends of the lattice, propagating along the lattice and colliding with other particles (where a collision occurs when two particles are in adjacent cells) and having reactions. It is also decided ahead of time what constitutes an answer, and the computation is considered to be complete when the answer appears somewhere in the lattice.

Squier and Steiglitz gave several examples to illustrate how particle machines might be programmed. In all the examples, $r = 1$ and $n \approx 14$. The first example is a binary-addition scheme, illustrated in Figure 14a, which is similar to the carry-ripple adder scheme described above. Here, the bits of the two addends, n and m, are represented by four different types of "data" particles: left- and right-moving 0s and 1s. The bits of n are injected in the left end of the lattice, least-significant bit first, and likewise for the bits of m in the right end of the lattice. In the center of the lattice is a stationary "processor" particle p_k, which can take on one of two types, p_0 or p_1, representing a carry bit of 0 or 1, respectively. It starts out in state p_0. When p_k's neighborhood contains a right-moving data particle on its left and a left-moving data particle on its right, the two data particles are annihilated, a new, left-moving "answer" particle is created that encodes the result of addition mod 2 (with the current carry bit), and p_k is set to either p_0 or p_1, depending on the new carry bit. The left-moving answer particle can pass unchanged through any additional right-moving data particles. The answer is read off the left end of the lattice as the answer particles arrive.

A second example is binary multiplication (Figure 14b). Here, as before, data particles encoding the bits of the two multiplicands n and m travel left and right towards the center of the lattice. In the center is a row of processor particles. At each three-way interaction between two data particles and a processor particle, the processor particle is set to encode the sum of the product of the two bits and its previous state (initially set to 0). Any carry bits are encoded by the right-moving particles. After all the data particles have passed through

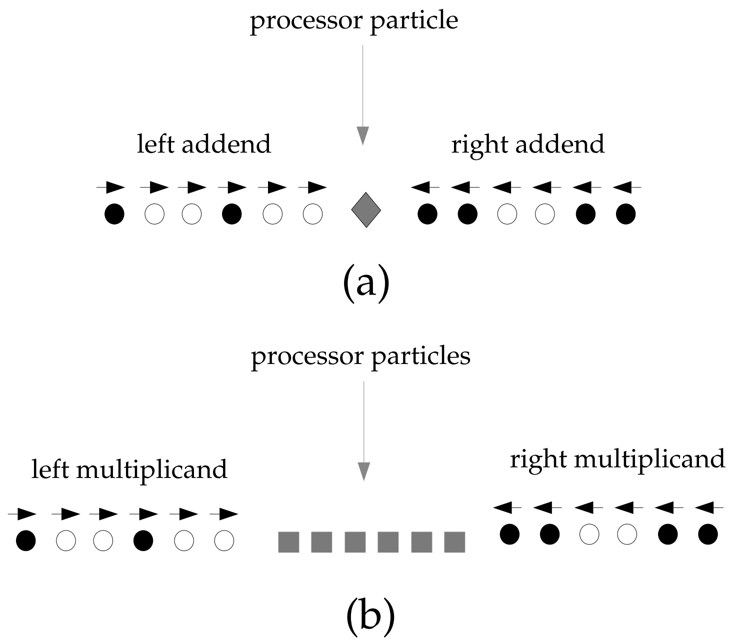


Figure 14: Schematic illustration of Squier and Steiglitz's (a) binary addition and (b) multiplication schemes using particle collisions. (Adapted from Squier and Steiglitz, 1994.)

each other, the processor particles contain the product nm, with the least-significant bit on the left. (Note that this scheme requires knowing ahead of time how many processor particles to use for a given n and m.)

Squier and Steiglitz give similar sketches for particle-collision schemes to perform arbitrarily nested combinations of arithmetic operators as well as schemes to perform digital filtering. These examples are meant to illustrate the general framework, which Squier and Steiglitz claim can provide a programmable CA substrate for general-purpose parallel computation. (They contrast their approach with systolic arrays, e.g., Kung, 1982. Systolic arrays are special-purpose CA-like devices in which each processor—and sometimes the connection topology—is typically more complicated than that in the CAs which we have been discussing). Like the BPFA approach described above, at this point the research into this framework is just beginning; it is as yet unclear how hard it will be to construct hardware and to write "programs" for this framework for performing more complex computations with a useful degree of parallelism. Some discussion of hardware implementation of this approach is given in Squier, Steiglitz, and Jakubowski (1995).

8. Evolving CAs with Genetic Algorithms

(This section is adapted from Mitchell, Crutchfield, and Das, 1996.)

In previous sections, I have described several projects in which cellular automata are cleverly (and sometimes with considerable difficulty) hand-designed to perform computations. The work of myself and my colleagues James Crutchfield, Rajarshi Das, and James Hanson takes a different approach, that of automatically designing CAs with genetic algorithms (GAs).

Genetic algorithms are search methods inspired by biological evolution. In a typical GA, candidate solutions to a given problem are encoded as bit strings. A population of such strings ("chromosomes") is chosen at random and evolves over several generations under selection, crossover, and mutation. At each generation, the fitness of each chromosome is calculated according to some externally imposed fitness function, and the highest-fitness chromosomes are selected preferentially to form a new population via reproduction. Pairs of such chromosomes produce offspring via crossover, where each offspring receives components of its chromosome from each parent. The offspring are then subject to a small probability of mutation at each bit position. After several generations, the population often contains high-fitness chromosomes—high-quality solutions to the given problem. (For overviews of GAs, see Goldberg, 1989 and Mitchell, 1996.)

Some early work on evolving CAs with GAs was done by Packard and colleagues (Packard, 1988; Richards, Meyer, and Packard, 1990). Koza (1992) also applied genetic programming to evolve CAs for simple random-number generation. Our work builds on that of Packard (1988), described in Section 4. We have used a form of the GA to evolve CAs to perform two computational tasks. The first is a density-classification task (Mitchell, Hraber, and Crutchfield, 1993; Mitchell, Crutchfield, and Hraber, 1994b; Crutchfield and Mitchell, 1995; and Das, Mitchell, and Crutchfield, 1994.) (Subsequent work on evolving CAs and related architectures to perform density classification was done by Sipper, to appear, and by Andre, Bennett, and Koza, 1996.) The second is a synchronization task (Das, Crutchfield, Mitchell, and Hanson, 1995). All the work described here uses one-dimensional, binary-state $r = 3$ CAs with periodic boundary conditions.

For the density classification task, the goal was to find a CA that decides whether or not the initial configuration (IC) contains a majority of 1s (i.e., has high density). If it does, the whole lattice should eventually go to the fixed-point configuration of all 1s (i.e., all cells in state 1 for all subsequent iterations); otherwise it should eventually go to the fixed-point configuration of all 0s. More formally, we call this task the "$\rho_c = \frac{1}{2}$" task. Here ρ denotes the density of 1s in a binary-state CA configuration and ρ_c denotes a "critical" or threshold density for classification. Let ρ_0 denote the density of 1s in the IC. If $\rho_0 > \rho_c$, then within M time steps the CA should go to the fixed-point configuration of all 1s (i.e., all cells in state 1 for all subsequent iterations); otherwise, within M time steps it should go to the fixed-point configuration of all 0s. M is a parameter of the task that depends on the lattice size N.

Designing an algorithm to perform the $\rho_c = \frac{1}{2}$ task is trivial for a system with a central controller or central storage of some kind, such as a standard computer with a counter register or a neural network in which all input units are connected to a central hidden unit. However, the task is nontrivial for a small-radius ($r \ll N$) CA, since a small-radius CA relies only on local interactions. It has been argued that no finite-radius, finite-state CA with periodic boundary conditions can perform this task perfectly across all lattice sizes (Land and Belew, 1995; Das, 1996), but even to perform this task well for a fixed lattice size requires more powerful computation than can be performed by a single cell or any linear combination of cells. Since the 1s can be distributed throughout the CA lattice, the CA must transfer information over large distances ($\approx N$). To do this requires the global coordination of cells that are separated by large distances and that cannot communicate directly. How

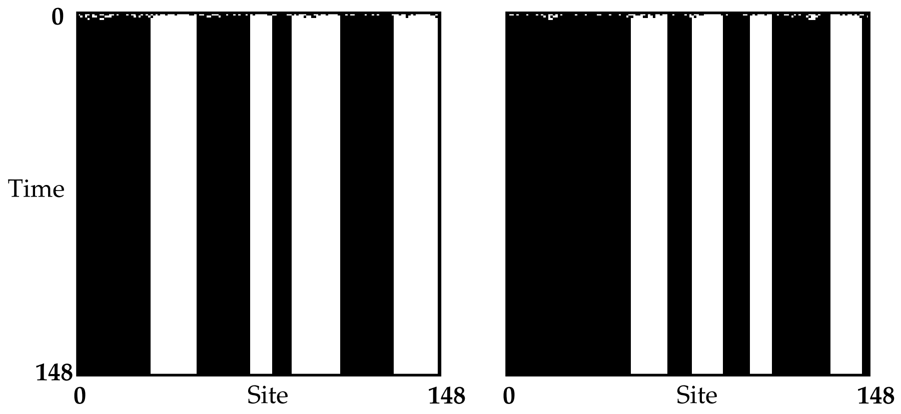


Figure 15: Space-time diagrams for ϕ_{maj}, the $r = 3$ majority rule. In the left diagram, $\rho_0 < \frac{1}{2}$; in the right diagram, $\rho_0 > \frac{1}{2}$.

can this be done while still exploiting the parallelism of CAs? Our interest was to see if the GA could devise one or more methods.

The need for such coordination is illustrated in Figure 15, in which we display the space-time behavior of a naive hand-designed candidate solution for this task—the "majority" rule ϕ_{maj}, in which the output bit for each 7-bit ($r = 3$) neighborhood is decided by a majority vote among the seven cells in the neighborhood. Figure 15 gives two space-time diagrams displaying the behavior of this rule on two initial conditions, one with $\rho_0 < 1/2$ and the other with $\rho_0 > 1/2$. As can be seen, local neighborhoods with majority 1s map to regions of all 1s and similarly for 0s, but when an all-1s region and an all-0s region border each other, there is no way to decide between them, and both persist. Thus, the majority rule (which implements a threshold on a linear combination of states) does not perform the $\rho_c = \frac{1}{2}$ task.

Instead, more sophisticated coordination and information transfer must be achieved. This coordination must, of course, happen in the absence of any central processor or central memory directing the coordination.

We used a genetic algorithm to search for $r = 3$ CA rules to perform the $\rho_c = \frac{1}{2}$ task. Each chromosome in the population represented a candidate CA rule—it consisted of the output bits of the rule table, listed in lexicographic order of neighborhood (cf. ϕ in Figure 1). The chromosomes representing rules were thus of length $2^{2r+1} = 128$. The size of the rule space in which the GA worked was thus 2^{128}—far too large for any kind of exhaustive evaluation.

Our version of the GA worked as follows. First, a population of 100 chromosomes was chosen at random from a distribution that was flat over the density of 1s in the output bits. (This "uniform" distribution differs from the more commonly used "unbiased" distribution in which each bit in the chromosome is independently randomly chosen. We found that using a uniform distribution considerably improved the GA's performance on this task—see Mitchell, Crutchfield, and Hraber, 1994b, for details). The fitness of a rule in the population was computed by (1) randomly choosing 100 ICs that are uniformly distributed over $\rho \in [0.0, 1.0]$, with exactly half with $\rho < \rho_c$ and half with $\rho > \rho_c$, (2) iterating the

CA ($r = 3$)	Rule table (hex)	$\mathcal{P}_{149,10^4}$	$\mathcal{P}_{599,10^4}$	$\mathcal{P}_{999,10^4}$
ϕ_{maj}	000101170117177f 0117177f177f7fff	0.000	0.000	0.000
ϕ_{exp}	0505408305c90101 200b0efb94c7cff7	0.652	0.515	0.503
ϕ_{par}	0504058705000f77 037755837bffb77f	0.769	0.725	0.714

Table 2: Rule tables and measured values of $\mathcal{P}_{N,10^4}(\phi)$ at various N for three different $r = 3$ rules. To recover the 128-bit string giving the output bits of the rule table, expand each hexadecimal digit (the first row followed by the second row) to binary. The output bits are then given in lexicographic order starting from the all-0s neighborhood at the leftmost bit in the 128-bit binary string. ϕ_{maj} (hand-designed) computes the majority of 1s in the neighborhood. ϕ_{exp} (evolved by the GA) expands blocks of 1s. ϕ_{par} (evolved by the GA) uses a "particle-based" strategy.

rule on each IC until it arrives at a fixed point or for a maximum of $M \approx 2N$ time steps, and (3) determining whether the final behavior is correct—i.e., 149 0s for $\rho_0 < \rho_c$ and 149 1s for $\rho_0 > \rho_c$. The rule's fitness, F_{100}, was the fraction of the 100 ICs on which the rule produced the correct final behavior. No partial credit was given for partially correct final configurations. (Like the initial population of rules, each sample of ICs on which to test the rules was chosen from a uniform distribution rather than an unbiased distribution over ρ because the former considerably improved the GA's performance on this problem.)

In each generation, a new set of 100 ICs was generated; F_{100} was computed for each rule in the population; CAs in the population were ranked in order of fitness; the 20 highest fitness ("elite") rules were copied to the next generation without modification; and the remaining 80 rules for the next generation were formed by single-point crossovers between randomly chosen pairs of elite rules. The parent rules were chosen from the elite with replacement—that is, an elite rule was permitted to be chosen any number of times. The offspring from each crossover were each mutated at exactly two randomly chosen positions. This process was repeated for 100 generations for a single run of the GA. (Since a different sample of ICs was chosen at each generation, the fitness function was stochastic.) For a discussion of this algorithm and details of its implementation, see Mitchell, Crutchfield, and Hraber (1994b).

In our experiments, we set $N = 149$, a reasonably large but still computationally tractable odd number (odd, so that the task will be well-defined on all ICs).

Six hundred runs were performed, each starting with a different random-number seed. We examined the fittest evolved rules to understand their computational "strategies" for performing the density classification task. On most runs the GA evolved a rather unsophisticated class of strategies. One example, a CA here called ϕ_{exp} (for "expand"), is illustrated in Figure 16. This rule had $F_{100} \approx 0.9$ in the generation in which it was discovered. Its computational strategy is the following: Quickly reach the fixed point of all 0s unless there is a sufficiently large block of adjacent (or almost adjacent) 1s in the IC. If so, expand that block. (For this rule, "sufficiently large" is 7 or more cells.) This strategy does a fairly good

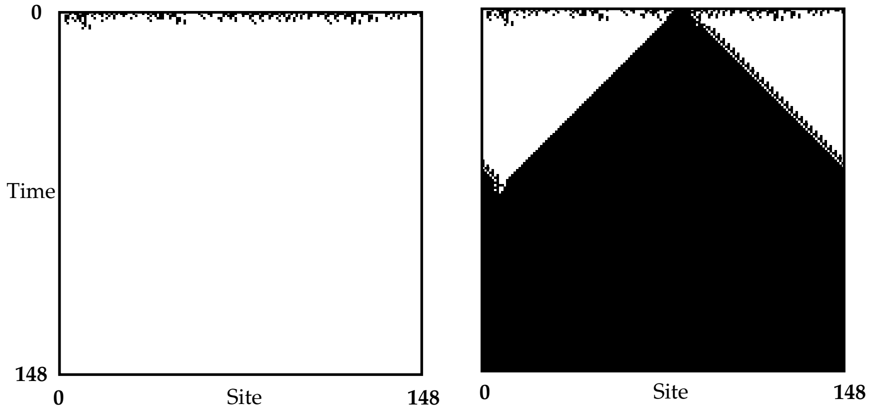


Figure 16: Space-time diagrams for a "block-expanding" rule, ϕ_{exp}. In the left diagram, $\rho_0 < 1/2$; in the right diagram, $\rho_0 > 1/2$. Both ICs are correctly classified.

job of classifying low and high density ICs under F_{100}: it relies on the appearance or absence of blocks of 1s to be good predictors of ρ_0, since high-density ICs are statistically more likely to have blocks of adjacent 1s than low-density ICs.

Similar strategies were evolved in most runs. On approximately half the runs, "expand 1s" strategies were evolved, and on most of the other runs, the opposite "expand 0s" strategies were evolved. These block-expanding strategies, although they obtained $F_{100} \approx .9$ with $N = 149$, do not count as sophisticated examples of computation in CAs: all the computation is done locally in identifying and then expanding a "sufficiently large" block. There is no notion of global coordination or information flow between distant cells—two things we claimed were necessary to perform well on the task. Indeed, such strategies perform poorly under performance measures using different distributions of ICs, and when N is increased. This can be seen in Table 2, which gives the rule tables and performances across different lattice sizes for different rules. The performance $\mathcal{P}_{N,10^4}$ is defined as the fraction of correct classifications over 10^4 ICs chosen at random from the unbiased distribution (each bit in the IC is independently randomly chosen). This is a more difficult test of quality than F_{100}: since they are chosen from an unbiased distribution, these ICs all have ρ_0 close to $1/2$ and are thus the hardest cases to classify. Therefore, $\mathcal{P}_{N,10^4}$ gives a lower bound on other performance measures. As can be seen, $\mathcal{P}_{N,10^4}(\phi_{\text{maj}})$ was measured to be zero for each N. $\mathcal{P}_{N,10^4}(\phi_{\text{exp}})$ is approximately 0.65 for $N = 149$ and drops to approximately 0.5 for the larger values of N.

Mitchell, Crutchfield, and Hraber (1994b) analyzed the detailed mechanisms by which the GA evolved such block-expanding strategies. This analysis uncovered some quite interesting aspects of the GA, including a number of impediments that, on most runs, kept the GA from discovering better-performing CAs. These included the GA's breaking the $\rho_c = \frac{1}{2}$ task's symmetries for short-term gains in fitness, as well as "overfitting" to the fixed lattice size $N = 149$ and the unchallenging nature of the IC samples.

Despite these various impediments and the unsophisticated CAs evolved on most runs, on approximately 3% percent of the runs the GA discovered CAs with more sophisticated strate-

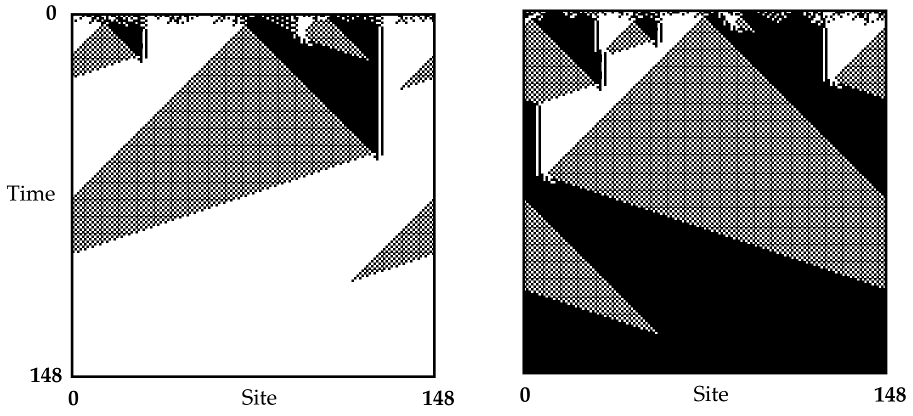


Figure 17: Space-time diagrams for ϕ_{par}, a "particle-based" rule. In the left diagram, $\rho_0 < 1/2$; in the right diagram, $\rho_0 > 1/2$. Both ICs are correctly classified. The gray-looking region consists of a checkerboard pattern of alternating 0s and 1s.

gies that yielded significantly better performance across different IC distributions and lattice sizes than was achieved by block-expanding strategies. The typical space-time behaviors of one such rule, here called ϕ_{par} (for "particle"), are illustrated in Figure 17.

The improved performance of ϕ_{par} can be seen in table 2. ϕ_{par} not only has significantly higher performance than ϕ_{exp} for $N = 149$, but its performance degrades relatively slowly as N is increased, whereas ϕ_{exp}'s performance drops quickly. As we describe in Das, Mitchell, and Crutchfield (1994), ϕ_{par}'s behavior is similar to that of a CA designed by Gacs, Kurdyumov, and Levin (1978).

In Figure 17 it can be seen that, under ϕ_{par}, there is a transient phase during which spatial and temporal transfer of information about the density in local regions takes place. Roughly, over short times, ϕ_{par}'s behavior is locally similar to that of ϕ_{maj} in that local high-density regions are mapped to all 1s, local low-density regions are mapped to all 0s, with a vertical boundary in between them. This is what happens when a region of 1s on the left meets a region of 0s on the right. However, there is a crucial difference from ϕ_{maj}: when a region of 0s on the left meets a region of 1s on the right, rather than a vertical boundary being formed, a checkerboard region (alternating 1s and 0s, appearing "gray" in the figure) is propagated. When the propagating checkerboard region collides with the black-white boundary, the inner region (e.g., the large white region in the right-hand diagram of Figure 17) is cut off and the outer region is allowed to propagate. In this way, the CA uses local interactions and geometry to determine the relative sizes of adjacent low- and high-density regions that are larger than the neighborhood size. For example, in the right-hand space-time diagram, the large inner white region is smaller than the large outer black region—thus the propagating checkerboard pattern reaches the black-white boundary on the white side before it reaches it on the black side; the former is cut off, and the latter is allowed to propagate.

The black-white boundary and the checkerboard region can be thought of as "signals" indicating "ambiguous" regions. The creation and interactions of these signals can be inter-

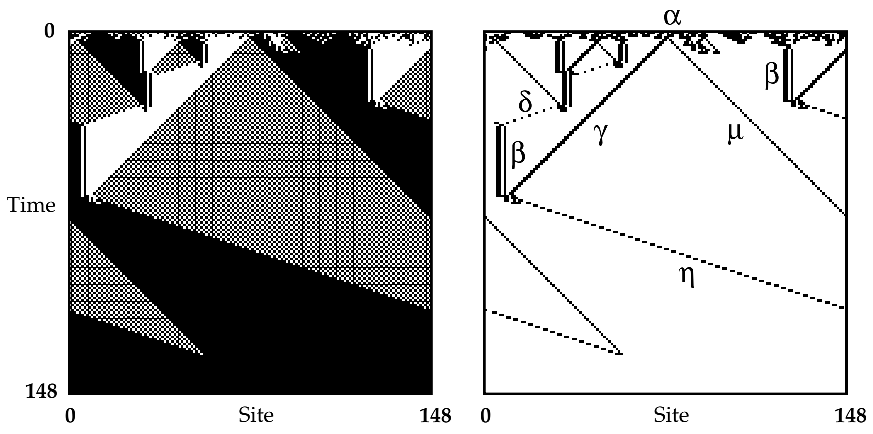


Figure 18: (a) The right-hand spacetime diagram of figure 17. (b) The same diagram with the regular domains filtered out, leaving only the particles (some of which are labeled by here by the Greek letter code of table 3). Note that particle α (unlike other the other particles) lasts for only one time step, after which it decays to particles γ and μ.

preted as the locus of the computation being performed by the CA—they form its emergent "algorithm."

The above explanation of how ϕ_{par} performs the $\rho_c = \frac{1}{2}$ task is informal and incomplete. Our approach to a more rigorous understanding is to examine the space-time behavior exhibited by the CA and to "reconstruct" from that behavior what the emergent algorithm is, using Crutchfield and Hanson's computational mechanics framework (Section 4). Recall that regular domains are (roughly) regions of space-time consisting of words in the same regular language—in other words, they are regions that are computationally homogeneous and simple to describe. E.g., in figure 17, there are three regular domains, corresponding to the regular languages 0^*, 1^*, and $(01)^*$. Particles are defined to be the localized boundaries between those domains. In computational mechanics, particles are identified as information carriers, and collisions between particles are identified as the loci of information processing. Particles and particle interactions form a high-level language for describing computation in spatially extended systems such as CAs. Figure 18 hints at this higher level of description: to produce it we filtered the regular domains from the space-time behavior displayed in the right-hand diagram of figure 17 to leave only the particles and their interactions, in terms of which the emergent algorithm of the CA can be understood. Table 3 gives a catalog of the relevant particles and interactions for ϕ_{par}. In Crutchfield and Mitchell (1995) and in Das (1996) we describe other particle-based rules that were evolved by the GA for this task. In Das, Mitchell, and Crutchfield (1994) we describe the evolutionary stages by which ϕ_{par} was evolved by the GA.

In Das, Crutchfield, Mitchell, and Hanson (1995), we gave a similar analysis for a global synchronization task. The goal for the GA was to find a CA that, from any IC, produces a globally synchronous oscillation between the all-1s and all-0s configurations. This task

Regular Domains		
$\Lambda^0 = 0^*$	$\Lambda^1 = 1^*$	$\Lambda^2 = (01)^*$

Particles (Velocities)	
$\alpha \sim \Lambda^0\Lambda^1$ (0)	$\beta \sim \Lambda^1 01 \Lambda^0$ (0)
$\gamma \sim \Lambda^0\Lambda^2$ (-1)	$\delta \sim \Lambda^2\Lambda^0$ (-3)
$\eta \sim \Lambda^1\Lambda^2$ (3)	$\mu \sim \Lambda^2\Lambda^1$ (1)

Interactions	
decay	$\alpha \rightarrow \gamma + \mu$
react	$\beta + \gamma \rightarrow \eta$, $\mu + \beta \rightarrow \delta$, $\eta + \delta \rightarrow \beta$
annihilate	$\eta + \mu \rightarrow \emptyset$, $\gamma + \delta \rightarrow \emptyset$

Table 3: Catalog of regular domains, particles (domain boundaries), particle velocities (in parentheses), and particle interactions seen in ϕ_{par}'s space-time behavior. The notation $p \sim \Lambda^x\Lambda^y$ means that p is the particle forming the boundary between regular domains Λ^x and Λ^y. $\emptyset$ denotes an annihilation (no particles are formed by the interaction).

differs considerably from the FSSP. First, the FSSP uses cells with many states, allowing for boundary cells, a "general" cell, and a diverse array of signals. In contrast, we are using two-state CAs with periodic boundary conditions, so there can be no special cells or propagating signals based on different local states. Instead, like in the density-classification task, "signals" have to emerge as particles—boundaries between regular domains formed by cells in state 0 or 1. Second, the desired final behavior is not static, as in the FSSP, but is dynamic—an oscillation between all 1s and all 0s. Third, in our task synchronization must take place starting from any IC, rather than all cells starting in the quiescent state as in the FSSP. We believe that all of these features make this task more interesting and make it require solutions that will be more similar to the emergence of spontaneous synchronization that occurs in decentralized systems throughout nature.

Figure 19 illustrates the behavior of one CA, here called ϕ_{sync}, evolved by the GA to perform this task. The behavior is illustrated both as a space-time diagram and as a filtered version of that diagram which reveals the embedded particles. The basic strategy of ϕ_{sync} is similar in some ways to that of ϕ_{par} for the density-classification task. Local regions of synchrony can be formed easily by requiring that the all-0s neighborhood maps to 1 and the all-1s neighborhood maps to 0. However, as can be seen in Figure 19a, those local regions can be out of phase with one another. ϕ_{sync} uses particles—boundaries between the synchronized regions and the jagged regions—to allow one local phase to take over the entire lattice, and to cut off the others.

Again, tools of computational mechanics allowed us to understand ϕ_{sync}'s strategy in the higher-level language of particles and particle interactions (Figure 19b) as opposed to the low-level language of CA rule tables and raw spatial configurations. This is described in detail in Das, Crutchfield, Mitchell, and Hanson (1995).

Our notion of computation via particles and particle-interactions differs considerably from the notions used in the work presented in previous sections. Propagating particle-like signals were used in the solution to the FSSP and in Smith's formal-language recognizers, but those were designed by hand to be the explicit behavior of the CA, and their interactions were effected by a relatively large number of states. Steiglitz, Kamal, and Watson's carry-ripple adder and the universal computer constructed in the Game of Life both used binary-state

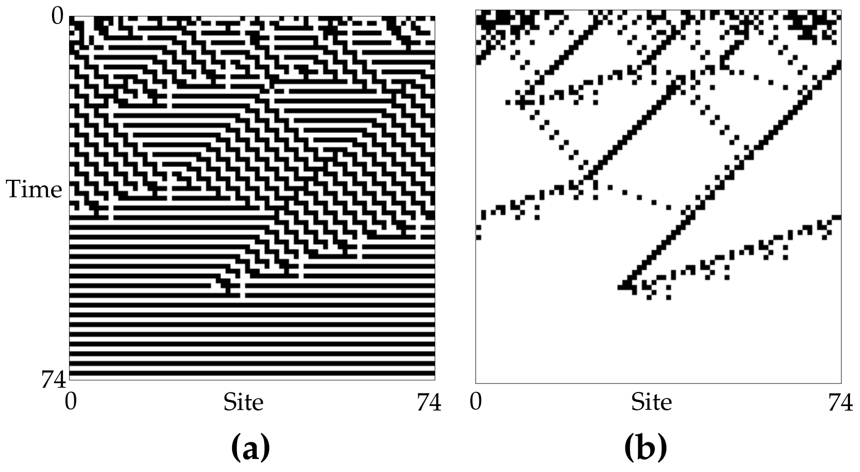


Figure 19: (a) Space-time diagram illustrating the behavior of a CA evolved by the GA to perform the synchronization task. The initial condition was generated at random. (b) The same space-time diagram after filtering out regular domains.

signals consisting of propagating periodic patterns, but again, the particles were explicit against a quiescent background, and their interaction properties were carefully hand coded. Squier and Steiglitz's particles were the primitive states of the cells and their interaction properties were explicit in the CA rule.

In contrast to all these systems, particles in our system are "embedded" as walls between regular domains, and are often apparent only after those domains have been discovered and filtered out. Their structures and interaction properties are "emergent" properties of the patterns formed by the CAs, and although each cell has only two possible states, the structures of embedded particles can be more complex than atomic or simple periodic structures. Finally, in contrast to Crutchfield and Hanson's work on computational mechanics of CAs, in our work particles and their interactions attain *semantics*, in that they contribute to an overall strategy for accomplishing a desired task. For example, in Figure 18(b), the γ particle can be interpreted as signaling a particular kind of ambiguity—a white region bordering a black region each of size greater than $2r+1$. Its interaction with the β particle and the subsequent creation of the η particle can be interpreted as signaling that the white region was smaller than the black region. This kind of interpretation contrasts with the non-semantic particles of Figure 8b. In the case of ϕ_{par}, these kind of semantics can be attached to particles and their interactions only in virtue of the evolutionary process to which the CA's lineage has been subject.

More generally, our philosophy is to view CAs as systems that naturally form patterns (such as regular domains), and to view the GA as taking advantage—via selection and variation—of these pattern-forming propensities so as to shape them to perform desired computations. We then attempt to understand the behavior of the resulting CAs by applying tools (such as the computational mechanics framework) formulated for analyzing pattern-forming systems, which give us a high-level description of the computationally relevant parts of the system's behavior. In doing so, we begin to answer Wolfram's last problem from

"Twenty problems in the theory of cellular automata" (Wolfram, 1985): "What higher-level descriptions of information processing in cellular automata can be given?", and we believe that this framework will be a basis for the "radically new approach" that Wolfram thinks will be needed.

This approach is motivated in part by our interest in developing scientific tools for understanding how systems in nature perform computations. However, we also believe that the best approach to engineering highly complex, massively parallel, and decentralized computing systems will be to take advantage of the natural dynamics of such systems in the same way that natural evolution takes advantage of the intrinsic dynamics of the systems on which it operates.

While our work on evolving CAs with GAs is a highly simplified example of this overall approach, the GA's discoveries of rules such as $\phi_{\rm par}$ and of rules that produce global synchronization is significant, since these are the first examples of a GA's producing sophisticated computation in decentralized, distributed systems. These discoveries are encouraging for the prospect of using GAs to automatically evolve computation for more complex tasks (e.g., image processing or image compression) and in more complex systems; these are the subjects of current work by our group. Moreover, evolving CAs with GAs also gives us a tractable framework in which to study the mechanisms by which an evolutionary process might create complex coordinated behavior in natural decentralized distributed systems. For example, we have already learned how the GA's breaking of symmetries can lead to suboptimal computational strategies (Mitchell, Crutchfield, and Hraber, 1993); eventually we may be able to use such models to test ways in which such symmetry breaking might occur in natural evolution. In general, models such as ours can provide insights on how evolutionary processes can discover structural properties of individuals that give rise to improved adaptation. In our case, such structural properties—regular domains and particles—were identified via the computational mechanics framework, and allowed us to analyze the evolutionary emergence of sophisticated computation.

9. Conclusion

This review has only begun to describe the large literature on computation in cellular automata, and the even larger literature on decentralized parallel computation in general. Some of the recent work on these topics is brought together in the *Parcella* proceedings (Handler, Legendi, and Wolf, 1995; Legendi et al., 1986; Wolf, Legendi, and Schendel, 1988; Wolf, Legendi, and Schendel, 1990; Jesshope, Jossifov, and Wilhelmi, 1994).

Among the important topics that were left out are discussions of hardware implementions of CAs (e.g., Toffoli and Margolus, 1987); reliable computation in CAs (e.g., Gacs, 1986); a larger discussion of analysis of CAs in terms of computation theory (e.g., Wolfram, 1984b; Nordahl, 1989; Moore, 1996), computation in more complex CA architectures, including three-dimensional CAs (e.g., Tsalides, Hicks, and York, 1989), CAs with real-valued states (e.g., Garzon and Botelho, 1993), and systolic arrays (e.g., Kung, 1982), and the many applications CAs and CA-like architectures have found in parallel computation.

The theory and applications of CAs and similar systems will likely attract increasing interest as the capabilities for massive parallel computation expand and robust, decentralized, and highly distributed systems become imperative. CAs and similar systems will also be increasingly important as modeling tools, as scientists probe more deeply into the behavior of natural systems composed of simple components with local communication and emergent collective properties. I hope that this review will contribute to the increase of interest in CAs and to the eventual understanding and wider application of such systems.

Acknowledgments

Thanks to Jim Crutchfield, Howard Gutowitz, Jim Hanson, Chris Langton, Cris Moore, and Mats Nordahl for many discussions about cellular automata and for comments on an earlier draft of this chapter. This work has been supported by the Santa Fe Institute and by grants from the National Science Foundation (IRI-9320200), the Department of Energy (DE-FG03-94ER25231), and the Defense Advanced Projects Research Agency and the Office of Naval Research (N00014-95-1-0975).

References

Andre, D., Bennett, F. H. III, and Koza, J. R. (1996). Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems.* Cambridge, MA: MIT Press.

Balzar, R. (1967). An 8-states minimal time solution to the firing squad synchronization problem. *Information and Control*, 10, 22–42.

Banks, E. R. (1971). *Information and Transmission in Cellular Automata.* Ph.D. Dissertation, Massachusetts Institute of Technology.

Berlekamp, E., Conway, J. H., and Guy, R. (1982). *Winning Ways for Your Mathematical Plays*, volume 2. Academic Press.

Bucher W. and Culik, K. II (1984). On real time and linear time cellular automata. *RAIRO Informatique Théorique*, 18 (4), 307–325.

Burks, A. W. (1966). Editor's introduction. In von Neumann (1966).

Burks, A. W. (editor) (1970a) . *Essays on Cellular Automata.* Urbana, IL: University of Illinois Press.

Burks, A. W. (1970b). Introduction. In Burks (1970a).

Burks, A. W. (1970c). Von Neumann's self-reproducing automata. In Burks (1970a).

Choffrut, C. and Culik, K. II (1984). On real-time cellular automata and trellis automata. *Acta Informatica*, 21, 393–407.

Clementi, A., De Biase, G. A., and Massini, A. (1994). Fast parallel arithmetic on cellular automata. *Complex Systems*, 8(6), 435.

Codd, E. F. (1968). *Cellular Automata.* New York: Academic Press.

Cole, S. N. (1969). Real-time computation by n-dimensional iterative arrays of finite-state machines. *IEEE Transactions on Computers*, 18, 349–365.

Crutchfield, J. P. (1994). The calculi of emergence: Computation, dynamics, and induction. *Physica D* 75: 11-54.

Crutchfield, J. P., and Hanson, J. E. (1993). Turbulent pattern bases for cellular automata. *Physica D* 69: 279–301.

Crutchfield, J. P., and Mitchell, M. (1995). The evolution of emergent computation. *Proceedings of the National Academy of Sciences, USA*, 92 (23): 10742.

Crutchfield, J. P., and Young, K. (1989). Inferring statistical complexity. *Physical Review Letters*, 63, 105.

Crutchfield, J. P., and Young, K. (1990). Computation at the onset of chaos. In W. H. Zurek (editor), *Complexity, Entropy, and the Physics of Information.* Reading, MA: Addison-Wesley.

Culik, K. II (1989). Variations of the firing squad problem and applications. *Information Processing Letters*, 30, 153–157.

Das, R. (1996). *The Evolution of Emergent Computation in Cellular Automata.* Ph.D. Thesis, Computer Science Department, Colorado State University, Ft. Collins, CO.

Das, R., Crutchfield, J. P., Mitchell, M., and Hanson, J. E. (1995). Evolving globally synchronized cellular automata. In L. J. Eshelman, ed., *Proceedings of the Sixth International Conference on Genetic Algorithms*, 336–343. San Francisco, CA: Morgan Kaufmann.

Das, R., Mitchell, M., and Crutchfield, J. P. (1994). A genetic algorithm discovers particle-based computation in cellular automata. In Y. Davidor, H.-P. Schwefel, and R. Männer, eds., *Parallel Problem Solving from Nature—PPSN III*, 244-353. Berlin: Springer-Verlag (Lecture Notes in Computer Science, volume 866).

Doolen, G. (editor) (1990). *Lattice Gas Methods for Partial Differential Equations.* Reading, MA: Addison-Wesley.

Eloranta, K. (1994). The dynamics of defect ensembles in one-dimesional cellular automata. *Journal of Statistical Physics*, 76(5 / 6):1377.

Eloranta, K. and Nummelin, E. (1992). The kink of cellular automaton rule 18 performs a random walk. *Journal of Statistical Physics*, 69, 1131-1136.

Farmer, D., Toffoli, T., and Wolfram, S. (editors) (1984). *Cellular Automata: Proceedings of an Interdisciplinary Workshop.* Amsterdam: North Holland.

Fischer, P. C. (1965). Generation of primes by a one-dimensional real-time iterative array. *Journal of the Association for Computing Machinery*, 12 (3), 388–394.

Fogelman-Soulie, F., Robert, Y., and Tchuente, M (1987). *Automata Networks in Computer Science: Theory and Applications.* Manchester, UK : Manchester University Press.

Forrest, S. (1990). Emergent computation: Self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. *Physica D* 42: 1–11.

Fredkin, E. and Toffoli, T. (1982). Conservative logic. *International Journal of Theoretical Physics*, 21, 219–253.

Gacs, P. (1986). Reliable computation with cellular automata. *Journal of Computer and Systems Sciences*, 32, 15–78.

Gacs, P., Kurdyumov, G. L., and Levin, L. A. (1978). One-dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii* 14: 92–98 (in Russian).

Garzon, M., and Botelho, F. (1993). Real computation with cellular automata. In Boccara, N. et al. (editors), *Cellular Automata and Cooperative Systems*, 191–202.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

Gould, S. J., and Vrba, E. S. (1982). Exaptation: A missing term in the science of form. *Paleobiology* 8: 4–15.

Gutowitz, H. A. (1990). *Cellular Automata.* Cambridge, MA: MIT Press.

Handler, W., Legendi, T., and Wolf, G. (editors) (1985). *International Workshop on Parallel Processing by Cellular Automata and Arrays (Parcella '84).* Berlin : Akademie Verlag

Hanson, J. E. (1993). *Computational Mechanics of Cellular Automata.* Ph.D. Thesis, Physics Department, University of California, Berkeley, CA.

Hanson, J. E., and Crutchfield, J. P. (1992). The attractor-basin portrait of a cellular automaton. *Journal of Statistical Physics* 66, no. 5/6: 1415–1462.

Hofstadter, D. R. (1979). *Gödel, Escher, Bach: an Eternal, Golden Braid.* New York: Basic Books.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley.

Jesshope, C., Jossifov, V., and Wilhelmi, W. (editors) (1994). *International Workshop on Parallel Processing by Cellular Automata and Arrays (Parcella '94).* Berlin : Akademie Verlag

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press.

Kung, H. T. (1982). Why systolic architectures? *Computer*, 15, 1 37–46.

Land, M., and Belew, R. K. (1995). No perfect two-state cellular automata for density classification exists. *Physical Review Letters* 74 (25): 5148.

Langton, C. G. (1984). Self-reproduction in cellular automata. *Physica D*, 10, 135–144.

Langton, C. G. (1990). Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D*, 42, 12-27.

Lawniczak, A. T. and Kapral, R. (editors) (1996). *Pattern Formation and Lattice Gas Automata.* Oxford: Oxford University Press.

Legendi, T. et al. (Editors) (1986). *International Workshop on Parallel Processing by Cellular Automata and Arrays (Parcella '86).* Berlin : Akademie Verlag

Li, W. and Packard, N. (1990). The structure of the elementary cellular automata rule space. *Complex Systems*, 4, 281–297.

Lindgren, K. and Nordahl, M. G. (1990). Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4, 299–318.

Margolus, N. (1984). Physics-like models of computation. *Physica D*, 10, 81–95.

Mazoyer, J. (1987). A six states minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50, 183–238.

Mazoyer, J. (1996). Computations on one-dimensional cellular automata. *Annals of Mathematics and Artificial Intelligence*, 16 (1–4).

Mazoyer, J. (1988). An overview of the firing squad synchronization problem. In C. Choffurt (editor), *Automata Networks*, 82–93. Lecture Notes in Computer Science, Vol. 316. Springer-Verlag.

Minsky, M. (1967). *Computation: Finite and Infinite Machines.* Englewood Cliffs, NJ: Prentice-Hall.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms.* Cambridge, MA: MIT Press.

Mitchell, M., Crutchfield, J. P., and Das, R. (1996). Evolving cellular automata to perform computations: A review of recent work. In *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96).* Moscow, Russia: Russian Academy of Sciences.

Mitchell, M., Crutchfield, J. P., and Hraber, P. T. (1994a). Dynamics, computation, and the "edge of chaos": A re-examination. In G. Cowan, D. Pines, and D. Melzner (editors), *Complexity: Metaphors, Models, and Reality.* Reading, MA: Addison-Wesley.

Mitchell, M., Crutchfield, J. P., and Hraber, P. T. (1994b). Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D* 75: 361–391.

Mitchell, M., Hraber, P. T., and Crutchfield, J. P. (1993). Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems* 7, 89–130.

Moore, C. (1996). *Majority-Vote Cellular Automata, Ising Dynamics, and* **P***-Completeness.* Working Paper 96-08-060, Santa Fe Insitute, Santa Fe, NM.

Moore, E. F. (1964). The firing squad synchronization problem. In E. F. Moore (editor), *Sequential Machines: Selected Papers*, 213–214. Reading, MA: Addison-Wesley.

Moore, F. R. and Langdon, G. G. (1968). A generalized firing squad problem. *Information and Control*, 12, 212–220.

Nordahl, M. G. (1989). Formal languages and finite cellular automata. *Complex Systems*, 3, 63–78.

Packard, N. H. (1988). Adaptation toward the edge of chaos. In J. A. S. Kelso, A. J. Mandell, M. F. Shlesinger, eds., *Dynamic Patterns in Complex Systems*, 293–301. Singapore: World Scientific.

Pecht, J. (1983). On the real-time recognition of formal languages in cellular automata. Acta Cybernetica 6 (1). 33-53.

Pesavento, U. (1996). An implementation of von Neumann's self-reproducing machine. *Artificial Life* 2 (4), 337-354.

Poundstone, W. (1985). *The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge.* New York: Morrow.

Richards, F. C., Meyer, T. P., and Packard, N. H. (1990). Extracting cellular automaton rules directly from experimental data. *Physica D* 45: 189–202.

Seiferas, J. I. (1977). Linear-time computation by nondeterministic multi-dimensional iterative arrays. *SIAM Journal of Computing*, 6, 487–504.

Sheth, B., Nag, P., and Hellwarth, R. W. (1991). Binary addition on cellular automata. *Complex Systems*, 5, 479.

Shinahr, I. (1974). Two and three-dimensional firing squad synchronization problems. *Information and Control*, 24, 163–180.

Sipper, M. (To appear). Co-evolving non-uniform cellular automata to perform computations. To appear in *Physica D*.

Smith, A. R. III (1971). Simple computation-universal cellular spaces. *Journal of the Association for Computing Machinery*, 18, 339–353.

Smith, A. R. III (1972). Real-time language recognition by one-dimensional cellular automata. *Journal of Computer and System Sciences*, 6, 233–253.

Sommerhalder, R. and van Westrhenen, S. C. (1983). Parallel language recognition in constant time by cellular automata. *Acta Informatica*, 19, 397.

Squier, R. K. and Steiglitz, K. (1994). Programmable parallel arithmetic in cellular automata using a particle model. *Complex Systems*, 8, 311–323.

Squier, R. K., Steiglitz, K., and Jakubowski, M. H. (1995). *General Parallel Computation Without CPUs: VLSI Realization of a Particle Machine.* Technical Report TR-484-95, Computer Science Department, Princeton University, Princeton, NJ.

Steiglitz, K., Kamal, I., and Watson, A. (1988). Embedding computation in one-dimensional automata by phase-coding solutions. *IEEE Transactions on Computers*, 37, 138–145.

Terrier, V. (1994). Language recognizable in real time by cellular automata. *Complex Systems*, 8, 325–336.

Toffoli, T., and Margolus, N. (1987). *Cellular Automata Machines: A New Environment for Modeling.* Cambridge, MA: MIT Press.

Tsalides, Ph., Hicks, P. J., and York, T. A. (1989). Three-dimensional cellular automata and VLSI applications. *IEE Proceedings*, 136, Pt. E, No. 6, 490–495

von Neumann, J. (1966). *Theory of Self-Reproducing Automata* (edited and completed by A. W. Burks). Urbana, IL: University of Illinois Press.

Waksman, A. (1966). An optimum solution to the firing squad synchronization problem. *Information and Control*, 9, 66–78.

Wolf, G., Legendi, T., and Schendel, U. (Editors) (1988). *International Workshop on Parallel Processing by Cellular Automata and Arrays (Parcella '88).* Berlin : Springer-Verlag.

Wolf, G., Legendi, T., and Schendel, U. (Editors) (1990). *International Workshop on Parallel Processing by Cellular Automata and Arrays (Parcella '90).* Berlin : Akademie-Verlag.

Wolfram, S. (1983). Statistical mechanics of cellular automata. *Review of Modern Physics*, 55, 601–644,

Wolfram, S. (1984a). Universality and complexity in cellular automata. *Physica D*, 10, 1–35.

Wolfram, S. (1984b). Computation theory of cellular automata. *Communications in Mathematical Physics*, 96, 15–57.

Wolfram, S. (1985). Twenty problems in the theory of cellular automata. *Physica Scripta*, T9, 170–183.

Wolfram, S. (editor) (1986). *Theory and Applications of Cellular Automata.* Singapore: World Scientific.

Yunes, J. B. (1994). Seven-state solutions to the firing squad synchronization problem. *Theoretical Computer Science*, 127, 313–332.