Complex Systems **1** (1987) 257–271

Virtual State Machines in Cellular Automata

Chris Langton *
Center for Nonlinear Studies
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

Abstract. One of the most important properties of cellular automata is their capacity to support *propagating structures*. Propagating structures are employed as *signals* in many CA models of computation. Treating propagating structures as *automata* elevates the notion of a signal to something that is a computational entity in its own right. These propagating signal-automata are embedded in the very "tape" upon which they compute, and are constituted of the very symbols that they write on the tape in the course of their computation. Thus, signal-automata are both process and data at the same time. They can create, modify, or erase other such automata, and can support distributed computations wherein the operators also serve as operands. As their existence is rather ephemeral when compared to that of the "physical" cells of the lattice that get caught up in their propagation, we refer to signal-automata as "virtual" automata, or *virtual state machines* (VSM's). In this paper, we look at some examples of VSM systems, explore a programming methodology based on the process of protein synthesis, and discuss the implications for "virtual structure" in the physical world.

1. Introduction

Cellular automata can be viewed either as computers themselves or as *logical universes* within which computers may be embedded. On the first view, an initial configuration constitutes the data that the physical computer is working on, and the transition function implements the algorithm that is to be applied to the data. This is the approach taken in most current applications of cellular automata, such as image processing.

The second approach, however, is much more interesting from a theoretical point of view. In this case, the initial configuration itself constitutes a computer of some sort, and the transition function is seen as the "physics" obeyed by the parts of the computer. The algorithm being run and the

*Computer address: cgl@lanl.gov.

data being manipulated are functions of the precise state of the initial configuration of the embedded computer. In the most general case, the initial configuration will constitute a universal computer.

As is the case for their physical counterparts, computers embedded in CA depend on *signals* to communicate information between different parts of the machine. Because of their local-connectivity, CA provides new insights into the nature of signals. We are led to ask: "What is a signal? Does it merely carry information between two distinct parts of a computer or can it be something more?"

In the remainder of this paper we explore an approach to embedded computation based on the composition of signals that compute. These signals are based on propagating structures in CA that have the capacity to make decisions, to effect changes in the medium through which they propagate, and to interact with other propagating structures in such a manner that the global effect of their dynamic interaction is a highly parallel, distributed computation. We end with a discussion of what the existence of such structures might imply about processes occurring in the natural world.

2. Signals as automata

In a physical computer, a signal is a wave-front of voltage-change that propagates along a wire at approximately the speed of light. Sometimes a signal only carries information, but in many cases it carries both information and the *power* to cause something to happen, as when a line that controls a gate is raised to logical one. In computers embedded within CA, signals are also typically implemented as waves of state-change propagating along a pathway that acts as a wire. This propagating "wave" can be quite small, typically only a cell or two in length. Since such a signal is propagating through a *logical* universe, considerations of power or dissipation are not necessary: logic suffices.

One views a computer embedded in a CA not so much as a device composed of lots of *active* parts *communicating* via signals, but as a device composed of *passive* parts *being operated on* by signals. A computation is viewed as the net effect of many signals working in parallel within the structure of the computer and altering its state: signals can be active computational entities in their own right.

It is natural to generalize the notion of a signal from something that is merely a passive bearer of information to something that is a more active *causal* entity. This generalization is most effectively made if we model signal-waves in CA as *automata* propagating within an n-dimensional "tape" that is the structure of the computer.

Generalizing signals to automata provides a useful framework within which to interpret the actions and potentialities of signals. In addition to a capacity to carry information, automaton-signals have the capacity to recognize and/or modify the various structures they encounter in the course of their propagation. Most importantly, since some of the struc-

tures encountered by automaton-signals may be other propagating signals, automaton-signals have the capacity to *recognize and modify each other* as well as to recognize and modify the more passive structures in their environment.

Viewing signals as automata also forces us to reconsider our notions of what kinds of structures constitute computers. On the above view, the very structure of the computer is itself data, subject to manipulation by the processes which are "running" within it. Indeed, we can dispense altogether with *any* particular passive configuration within which signals are constrained to propagate, and computations that consist solely of signals interacting with one another can be embedded in CA. In this case, the only role left to the physical "hardware" is to provide a medium within which signals can propagate: an Æther.

3. Virtual state machines

Propagating structures in CA are typically small, periodic configurations that constantly displace themselves with respect to the fixed cellular background. Although they have often been employed as mere carriers of information, they are also capable of performing arbitrary computational tasks.

The simplest computational entities formally recognized in computer science are *finite state machines.* A finite state machine (FSM) consists of a finite set of states Σ, a finite input alphabet α, and a transition function Δ that maps a *(state,input)* pair to the next state: $\Delta : (\Sigma \times \alpha) \rightarrow \Sigma$. An FSM may also produce an output symbol, selected from a finite output alphabet, at every state transition. An FSM "receives" a single input stream consisting of a sequence of symbols from its input alphabet. An FSM cannot review its own input or output, thus its "memory" is limited to its set of states.

The most powerful formally recognized computational entities are *Turing machines.* A Turing machine (TM) extends the concept of an FSM by incorporating an indefinitely extensible tape that it can read from, write on, and move over in either direction. Thus, a TM can review both its input and its output, and its "memory" is in principle unbounded.

We will regard any configuration in a CA that can function as an FSM to be an automaton. How simple can such a configuration be? As simple as a single cell. By definition, each cell in a CA contains a finite state machine. Let us call these fixed base-machines "zero-order" or *physical* automata.

In contrast, any configuration that can function as a TM must consist of more than one cell, since each cell has only a finite number of states. We will call locally periodic configurations that occupy *more* than one cell in space *or* time "higher-order" or *virtual* automata. Virtual automata are supported "on the shoulders", so to speak, of the lattice of zero-order, physical automata that constitute the cellular array. Propagating structures, such as signals, are prime examples of virtual automata.

If a propagating structure can ever encounter physical cells that it has

traversed before, then it is potentially a *virtual Turing machine*, if not, then it is a *virtual finite automaton*. This distinction is complicated by the fact that it is possible that the output of a virtual automaton may *itself* propagate through the array, in which case a virtual automaton may encounter some of its own previous output even if it never retraces its absolute physical path.

There are a number of observations that follow when we view propagating structures as virtual automata. In order to have a convenient name, we will refer to virtual automata as *virtual state machines* (VSM's), whether they are functioning as finite-state machines or as Turing machines.

- *VSM's are embedded in the very tape upon which they are operating.* Both machine and data are represented as states of the same medium: an array of cells. Thus, VSM's are both *processes* and *data* at the same time.
- Since VSM's are both processes and data at the same time, *writing on the 'tape' of the environment is equivalent to construction.*
- VSM's can *erase* as well as construct.
- VSM's can be *self-erasing*, which is the ultimate form of *halting*.
- Since constructed configurations can also be viewed as either data or processes, *VSM's can construct other VSM's.* Likewise, VSM's can *erase* other VSM's.
- Because they are both processes and data, *VSM's can treat other VSM's as 'data' and read or modify their structure.*
- Because configurations can occur on all scales, *VSM's can be embedded in other VSM's.* Thus VSM's can be hierarchically composed of smaller VSM's.

Thus, the rigidly-fixed, homogeneous lattice of *physical* automata that constitutes the cellular array can support a heterogeneous population of *virtual* automata that are relatively free to migrate around (like ants) in the lattice. Furthermore, this population of virtual automata can vary in size and composition with time as VSM's are created, modified, and destroyed by other VSM's and processes occurring in the array.

4. Examples of VSM's

The *glider* of Conway's cellular automaton game of "Life" [1] is an example of a VSM. The glider is a configuration that propagates with respect to the background of fixed physical automata, cycling through four "states" over and over again, as shown in figure 1.

Figure 1: A Glider propagating with respect to a fixed cell (○).

The glider is just one example of a general class of structures that propagate with respect to the background of fixed physical cells. They are solitary, particle-like waves of state-change, rippling across the cellular background.

Figure 2 shows more examples of VSM's. Figure 2a shows the simplest propagating structure that can be implemented in a rotation-symmetric, two-dimensional cellular automaton. It will propagate to the right. Figure 2b shows a simple automaton traveling to the right, writing an alternating pattern as it propagates. Figure 2c shows a simple automaton traveling to the left, making an inverse copy of a row of cells as it does so. Figure 2d shows a VSM propagating to the right that produces a regular "string" of VSM's that propagate upward.

Propagating structures have been used as *signals* in the construction of embedded computers. Both von Neumann and Codd used signals propagating along "wires" in their designs for universal constructing machines embedded in cellular automata [2,3]. Conway [4] used the glider both as the carrier of a bit of information and as a specialized operator in constructing the proof that the "Life" transition rule can support universal computation. Margolus [5] has implemented Fredkin's Billiard Ball Model of Computation (BBMC) [6] in a cellular automaton. In the BBMC, hard spheres (billiard balls) collide elastically with each other and with mirrors–reflective walls–to implement computations. Any location where two balls might collide constitutes a *Fredkin gate* where the potential collision is taken as implementing a logical function–such as 'AND'–in a *reversible* manner. In Margolus' implementation of the BBMC, the billiard balls are modeled by propagating structures that interact as if they were involved in elastic collisions.

In all of these models, propagating structures carry information and interact with other propagating structures or with static structures in the array. In all cases, the propagating structures are essentially autonomous, are capable of making "decisions" based on their local environments, and can affect, or be affected by, the state of the physical cells through which they are propagating. In short, they are functioning as automata.

5. VSM synthesis model

In the examples mentioned above, a few specialized automata interact in rather restricted ways. We would like a programming methodology for

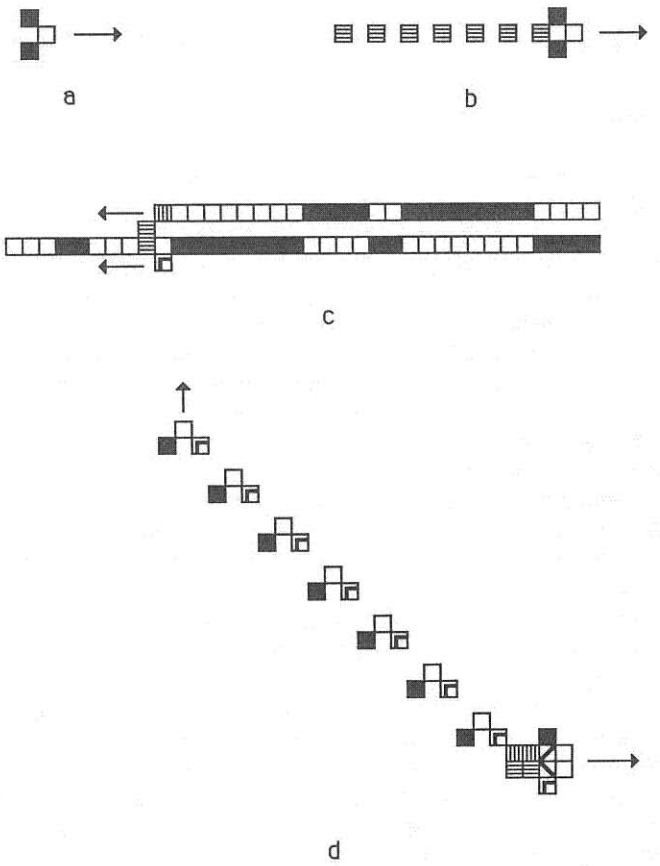


Figure 2: Examples of virtual state machines (VSM's). (a) A simple VSM traveling to the right. (b) A simple VSM traveling to the right and writing a regular string. (c) A VSM propagating to the left, making an inverse copy of a string that it is traversing. (d) A VSM propagating to the right, producing as output a regular sequence of VSM's traveling upward.

VSM's that allows the production of arbitrary automata when and where needed in the course of parallel, distributed computation. We take the process of protein synthesis in living cells as our model.

In living cells, *descriptions* for a great many different molecular automata- enzymes- are encoded in the DNA. These descriptions are turned into functioning molecular machines in the process of protein synthesis. This process is initiated when a polymerase enzyme, propagating along the DNA, recognizes a particular base-sequence label. When it encounters the label it is seeking, the enzyme triggers the transcription of the following stretch of DNA into messenger RNA (mRNA). When the polymerase enzyme reaches a special "stop" sequence on the DNA, transcription is terminated and the mRNA is cut loose. At this point, various editing procedures can be invoked on the mRNA strand, including the excision of several segments—*introns*—which may go on to function as enzymes themselves [7]. The edited mRNA is then transferred to a ribosome in the cytoplasm where it is used as a template for building a protein.

On this model, we can construct a "string" of cells in the array that will function like DNA: as the repository for *descriptions* of automata. Each description will be preceded by a *label* that can be recognized by an enzyme-like automaton that is traversing the string. To simplify things, we will have the result of the recognition of a label be the direct construction of the automaton described, rather than go through the extra steps of first transcribing the description and then transferring the transcription to another site before constructing the automaton.

In the following example, we have constructed a simple Turing machine by using the model described above. Figure 3 gives an overview of the machine and the interpretation of the symbols used in the diagrams of the machine's operation. There are two linear strings of cells in the array that constitute "tapes". One tape holds the rules and the other holds the data. A rule consists of four consecutive cells, which encode the rule as follows. The first cell is a label that uniquely identifies the rule. The first cell after the label indicates what to *print* at the "current-cell marker", which sits directly above the current cell on the data-tape and reflects its state. The second cell after the label indicates which direction to *move* the current-cell marker. The third cell after the label indicates what the *next-state* should be.

The machine is started with a single-cell VSM sitting next to the current-cell marker on the data-tape (figure 4a). This VSM holds the starting state of the machine. By sensing the state of the current-cell marker, the VSM holding the starting state can determine the current state and the current tape-symbol. Thus, it can determine what rule to search for on the rule-tape. The VSM moves to the rule-tape and starts to search for the proper rule-label (figure 4b), passing over rules with different labels. When it encounters the proper label (figure 4c), it activates the rule following the label. When activated, three VSM's are produced, in sequence, one from each of the three cells following the label cell (figure 4d). Each VSM

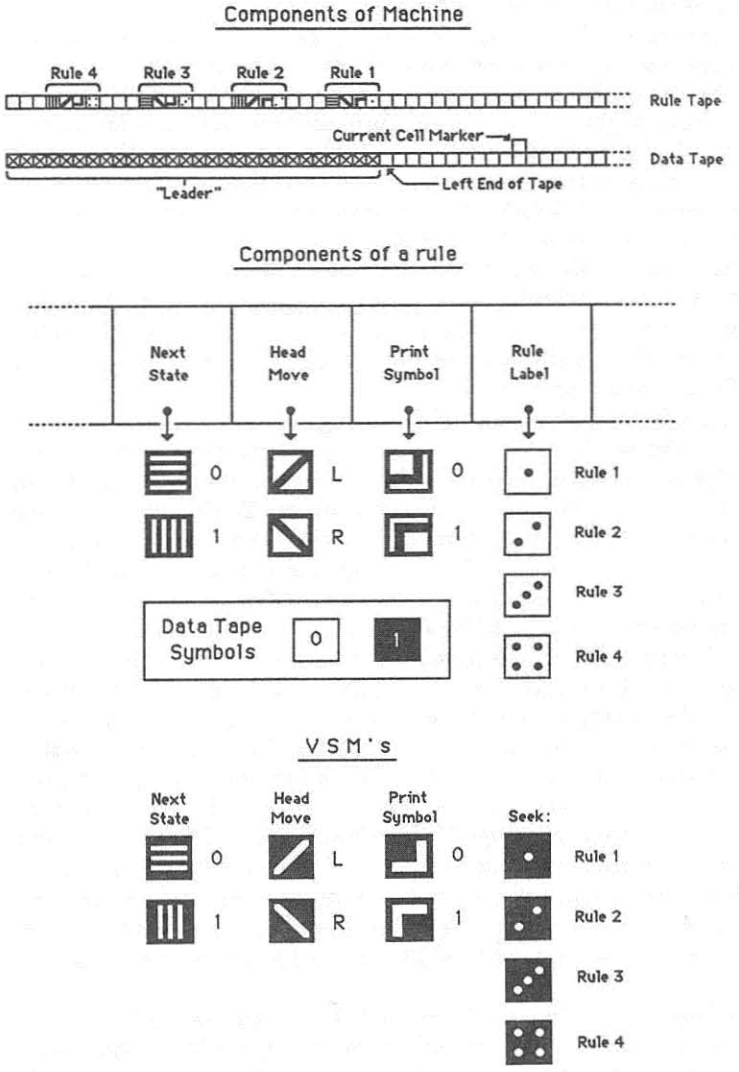
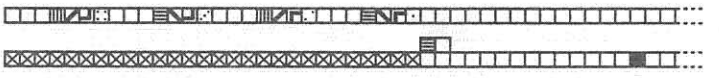
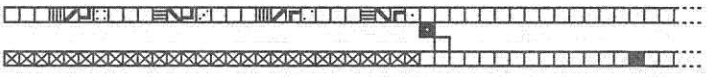


Figure 3: Plan of VSM Turing machine and key to figure 4.



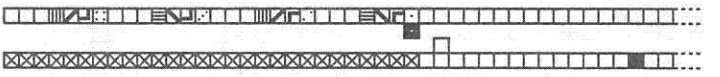
a

Initial-state VSM (▤) sits next to current cell marker.



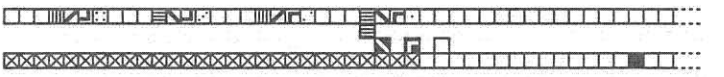
b

Rule-1 seeking VSM (■) moves to rule tape.



c

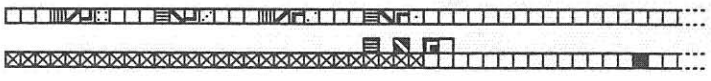
Rule-1 seeking VSM matches rule-1 label.



d

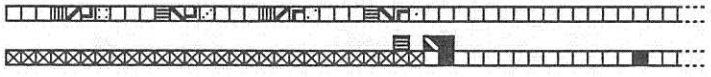
The three rule VSM's are generated and move to data tape.

Figure 4: VSM Turing Machine operation. (a–d) Determination of rule to seek and triggering of VSM's.



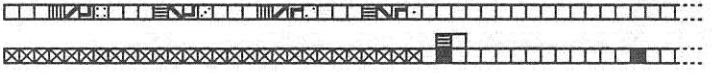
e

Rule VSM's propagate to current-cell marker.



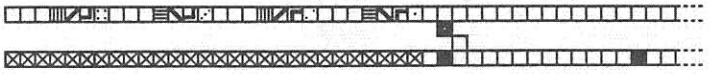
f

Print-symbol VSM has written a 1 at current cell.



g

Move VSM has moved current-cell marker to the right.



h

Next-state VSM has determined next rule to seek.

Figure 4: (e–h) Application of VSM's, determination of next rule to seek, and iteration of cycle.

moves to the data tape and propagates along it to the current-cell marker, where it performs its action (figure 4e). The first VSM alters the state of the current-cell marker to the state dictated as the print symbol by the rule (figure 4f). The current-cell, sitting directly beneath the current-cell marker, changes itself accordingly. The second VSM moves the current-cell marker to the right or to the left one cell, where it assumes the state of the new current-cell beneath it (figure 4g). Finally, the third VSM carries the next-state information necessary to iterate the cycle: when it encounters the current-cell marker it determines the proper rule to apply next and moves to the rule-tape in order to search for the proper label (figure 4h). And so the process continues.

Halting is easily implemented by *not* including a description for the *next-state* in some rule. Then, when the rule for halting is activated, the rule will produce only two VSM's—one for printing the new symbol at the current-cell marker and one for moving the current-cell marker. No next-state VSM will be produced and hence no new rule will be sought and the process comes to a halt.

Note that it does not matter in what order the rules are stored on the rule tape, and that there is no necessary connection between a label and the specific rule that follows it. The association between rules and labels is arbitrary. Any combination of print-symbol, current-cell marker move, and next-state can follow any label. Thus any Turing machine can be "programmed" on the rule-tape, within the limits of the rule code (maximum number of states, finite input and print alphabets, and etc). If the rules are those of a *universal Turing machine*, then any Turing machine at all may be simulated. One portion of the data-tape will then encode the rules for the specific Turing machine to be simulated.

The basic form of this simple Turing machine may easily be extended in a number of ways to implement a more general, but still centrally-located, control structure. For instance, the "data-tape" could be extended to include the whole of the n-dimensional array in which the rule-tape is embedded. Then there could be many VSM "heads" acting in parallel, sensing various currently-active sites distributed throughout the n-dimensional space, determining the rules to apply, and propagating to the rule-tape to search for the proper labels.

There could also be multiple rule-tapes, some of which could contain duplicate labels. The rules associated with identical labels on different rule-tapes might or might not be identical–allowing multiple rule "alleles" or *paralleles*. Furthermore, the rule-tapes themselves could be acted upon as if they were data-tapes, either by direct modification of rules or labels, or by modification in such a way that rules could be made unavailable for further activation until subsequently remodified.

Finally, the rules themselves could be distributed over the n-dimensional array like the data, either statically or as VSM's. In the former case the rules would be sought out locally, while in the latter case the rules themselves would actively seek out the situations to which they applied.

Note that there might also be other processes going on in the array. The VSM's produced by the rule tapes may be sensing the current status of these other processes. The other processes might also produce VSM's themselves, which would either act on the rule tapes directly or trigger rules to be activated. These other processes may themselves be directed by rule tapes, giving the effect of multiple "cells" communicating with one another. In this manner, higher-order control structures can be built. Alternatively, the other processes might be considered to be "outside" the rule-tape system, and thus constitute elements of the "environment". Such a model might prove very effective in understanding the cellular processes that support life [8].

6. Social automata: computation in colonies of co-operating automata

Life, intelligence, and even computation, are behavioral phenomena that emerge from the interaction of many inanimate, unintelligent, and even illogical parts. There is much to be gained, therefore, from the study of the spectrum of global behaviors that can emerge from the aggregation together of many separate entities, each with its own behavioral repertoire. In some cases, the global activity will be just the sum of the activities of the individuals. In other cases, however, the global activity is much more complex than the behavior of isolated individuals would lead us to expect.

Cellular automata provide us with several hierarchical levels of "individuals" out of which aggregates may be composed. The first level consists of the individual cells, each of which is occupied by the same finite automaton. What makes CA so interesting is that the global behavior supported by a lattice of such automata is much more complex and varied than the sum of the behaviors of the individual automata. Propagating structures—VSM's—constitute another level of individuals. Although they are directly supported by the rigidly fixed, homogeneous cells of the lattice, VSM's are free to migrate around in the lattice, constantly changing their set of neighboring virtual automata. Furthermore, a *homogeneous* lattice of automata can support a *heterogeneous* population of VSM's, and this population can vary in size and composition with time, as VSM's are created, modified, and destroyed by processes occurring in the lattice. Thus, a rigidly-fixed, uniform population can support a polymorphic *society* of relatively free-ranging individuals. Higher levels in the hierarchy of individuals are due to the interaction of the processes that are constituted of individuals of lower levels, in much the manner of the biological hierarchy of molecules: cells, tissues, organs, organisms, societies, and so forth.

The individual automata belonging to these societies can interact in several different ways. As finite-state or Turing machines, these automata are computing some function on the tape of their environment. Each individual automaton can be thought of as both recognizing a language and writing a language, as it is both being affected by and affecting its envi-

ronment. Since it is a shared environment, individual automata can be affected by the computations of others either *indirectly* via their effects on the environment, or by operating on each other *directly*.

To the extent that automata recognize portions of each other's output, we can say that a channel of communicaton exists between the automata. Furthermore, the "string" being produced as a result of a VSM's computation can itself turn into one or more VSM's. Thus, VSM's can be seen as implementing a *meta-grammar*, where the "strings" produced may turn into machines that implement yet other grammars. These, in turn, may spawn other such machines, and so forth. The analysis of such meta-grammars is problematic, since machines at all levels of the grammatical hierarchy can interact with one another. It is even possible to have "autocatalytic cycles" of VSM's, where machine $\mathcal{A}$ produces machine $\mathcal{B}$, which produces machine $\mathcal{C}$, which produces....., which produces machine $\mathcal{N}$, which closes the cycle by producing machine $\mathcal{A}$.

The important point to be made about these hierarchies of individuals is that VSM-like structures can occur at any level of the hierarchy: systems of VSM's can support higher-order VSM's. This suggests that we, as interacting individuals immersed in a social aggregate, may often get caught up in the propagation of VSM-like social processes sweeping through our local neighborhood of the society, without really being aware of the VSM itself, only of the particular task it induces us to undertake. There is much to be learned about the ebb and flow of such social processes. Systems of VSM's in CA could give us a new way to investigate social dynamics.

7. Virtual structure

Physics has primarily concerned itself with the *analysis* of matter. Physicists analyze matter by taking it apart and finding its *subparts*, and then iterating the process on the subparts. *Synthesis* is a process directly opposed to analysis. It consists in putting together parts to form *superparts*. Just as we can discover the way in which matter decomposes into subparts, we can explore the way in which matter can be composed into superparts. Thus, just as there are such things as sub-particles that constitute material things, there are such things as *super-particles* that material things can constitute.

What propagating VSM's in CA demonstrate clearly is that there are really *two* classes of "things" that sub-parts can be composed into: *physical things* and *virtual things*. The physical things are those which exist on the same time scale as their constituent parts. The virtual things are those which persist over longer time scales than their constituent parts. In general, analysis has not dealt with virtual things very well for two reasons: (1) virtual things are harder to identify than physical things, and (2) by the time you get hold of a virtual thing to take it apart, you have only got the parts that constitute the virtual thing and not the virtual thing itself. If you "fix" a virtual thing for study you have lost it, because its

very existence depends on its dynamic structure.

Virtual things depend on the flux of their constituent parts. They are open systems, continuously exchanging matter with the environment. Thus, virtual things must have the capacity to create the conditions for their own transfer—they pave their own way. If a virtual structure is to persist in time, it must have the capacity to impose its structure on an ever-changing set of constituent subparts.

It is not such a great step from this to *replication*. All that is required is that the pattern hang onto its constituent parts for a while *after* it has imposed its structure on a new set of parts, rather than relinquishing the old parts in the process. Thus, it is conceivable that propagation may be the ancestor of replication: propagating virtual structures in the pre-biotic soup may have been the precursers of templating nucleotides.

Note that to identify something as virtual is *not* to remove its causal efficacy. Virtual things interact with physical things all the time, they are patterns of organization that overtake physical things and sweep them up in a local hurricane of organization, eventually discarding them as they move on to overtake other physical things and recruit them into their physical basis of support. Thus, super-particles can be virtual and yet still have physical, causal efficacy: they constitute the "force" behind the dynamic organization of material things. There is nothing vitalistic in such a statement, it merely acknowledges that structure can be dynamic as well as static.

8. Summary

We have discussed a view of computation in CA that generalizes the notion of a signal to the notion of a propagating automaton. On this view, a computation is the global result of the local interactions of a set of signal-automata, working within and altering the structure of the computer. In the extreme case, the structure of the computer can consist *solely* of such automata, propagating within the logical Æther provided by the lattice of physical cells. "Programs" in such spaces must consist of spatially distributed structures. A programming model based on the process of protein synthesis was proposed that involves enzyme-like, propagating automata that translate passive descriptions of automata into functional automata, whose effects may result in the translation of yet other automata. Such parallel sets of propagating automata may be viewed as colonies or "societies" of automata. Such a viewpoint should facilitate both the understanding of social dynamics and the application of useful principles of social organization to the task of computation. Furthermore, the combination of ephemeral existence and *physically* causal efficacy exhibited by virtual automata in CA suggests that similar processes may be responsible for much of the dynamic nature of the world around us. The study of the behavior possible in systems of virtual state machines in cellular automata should provide us with unique insights into the nature of virtual structure in the

physical world.

References

[1] Martin Gardner, "The Fantastic Combinations of John Conway's New Solitaire Game 'Life'", *Scientific American*, **223** (October 1970) 120–123.

[2] John von Neumann, *Theory of Self-Reproducing Automata*, edited and completed by Arthur W. Burks (University of Illinois Press, 1966).

[3] E.F. Codd, *Cellular Automata*, (Academic Press, 1968).

[4] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy, *Winning Ways for your Mathematical Plays* Volume 2, (Academic Press, 1982).

[5] Norman Margolus, "Physics-Like Models of Computation", *Physica D*, **10** (1984) 81–95.

[6] Edward Fredkin and Thommaso Toffoli, "Conservative Logic", *International Journal of Theoretical Physics*, **21** (1982) 219–253.

[7] Thomas R. Cech, "RNA as an Enzyme", *Scientific American*, **255** (November 1986) 64–75.

[8] Christopher G. Langton, "Studying Artificial Life with Cellular Automata", *Physica D*, **22** (1986) 120–149.