Christopher G. Langton
Center for Nonlinear Studies, Los Alamos National Laboratory, Los Alamos, NM 87545
internet address cgl@lanl.gov

Artificial Life

Vitalism amounted to the assertion that living things do not behave as though they were nothing but mechanisms constructed of mere material components; but this presupposes that one knows what mere material components are and what kind of mechanisms they can be built into.
— C.H. Waddington, *The Nature of Life.*

Artificial Life is the study of man-made systems that exhibit behaviors characteristic of natural living systems. It complements the traditional biological sciences concerned with the *analysis* of living organisms by attempting to *synthesize* life-like behaviors within computers and other artificial media. By extending the empirical foundation upon which biology is based *beyond* the cabon-chain life that has evolved on Earth, Artificial Life can contribute to theoretical biology by locating *life-as-we-know-it* within the larger picture of *life-as-it-could-be.*

Artificial Life, SFI Studies in the Sciences of Complexity,
Ed. C. Langton, Addison-Wesley Publishing Company, 1988

THE BIOLOGY OF POSSIBLE LIFE

Biology is the scientific study of life—in principle anyway. In practice, biology is the scientific study of life based on carbon-chain chemistry. There is nothing in its charter that restricts biology to the study of carbon-based life; it is simply that this is the only kind of life that has been available for study. Thus, theoretical biology has long faced the fundamental obstacle that it is difficult, if not impossible, to derive general theories from single examples.

Certainly life, as a dynamic physical process, could "haunt" other physical material: the material just needs to be organized in the right way. Just as certainly, the dynamic processes that constitute life—in whatever material bases they might occur—must share certain universal features—features that will allow us to recognize life by its dynamic *form* alone, without reference to its *matter*. This *general* phenomenon of life—life writ-large across all possible material substrates—is the true subject matter of biology.

Without other examples, however, it is extremely difficult to distinguish essential properties of life—properties that must be shared by any living system *in principle*—from properties that are incidental to life, but which happen to be universal to life on Earth due *solely* to a combination of local historical accident and common genetic descent. Since it is quite unlikely that organisms based on different physical chemistries will present themselves to us for study in the foreseeable future, our only alternative is to try to synthesize alternative life-forms ourselves—*Artificial Life*: life made by man rather than by nature.

ARTIFICIAL LIFE

Only when we are able to view *life-as-we-know-it* in the larger context of *life-as-it-could-be* will we really understand the nature of the beast. Artificial Life (AL) is a relatively new field employing a *synthetic* approach to the study of *life-as-it-could-be*. It views life as a property of the *organization* of matter, rather than a property of the matter which is so organized.

Whereas biology has largely concerned itself with the *material* basis of life, Artificial Life is concerned with the *formal* basis of life. Biology has traditionally started at the top, viewing a living organism as a complex biochemical machine, and worked *analytically* downwards from there—through organs, tissues, cells, organelles, membranes, and finally molecules—in its pursuit of the mechanisms of life. Artificial Life starts at the bottom, viewing an organism as a large population of *simple* machines, and works upwards *synthetically* from there—constructing large aggregates of simple, rule-governed objects which interact with one another nonlinearly in the support of life-like, global dynamics.

The "key" concept in AL is *emergent behavior*. Natural life emerges out of the organized interactions of a great number of nonliving molecules, with no global controller responsible for the behavior of every part. Rather, every part is a behav*or* itself, and life is the behav*ior* that emerges from out of all of the local interactions among individual behav*ors*. It is this bottom-up, distributed, local determination of behavior that AL employs in its primary methodological approach to the generation of lifelike behaviors.

ARTIFICIALITY

The dictionary[1] defines the term "artificial" as "made by man, rather than occurring in nature," but there is another sense of the term that is more appropriate for the study of Artificial Life. This sense was best captured by Simon in his excellent monograph *The Sciences of the Artificial*[38]:

> Artificiality connotes perceptual similarity but essential difference, resemblance from without rather than within. The artificial object imitates the real by turning the same face to the outer system... imitation is possible because distinct physical systems can be organized to exhibit nearly identical behavior... Resemblance in behavior of systems without identity of the inner systems is particularly feasible if the aspects in which we are interested arise out of the *organization* of the parts, independently of all but a few properties of the individual components.

Thus, Artificial Life studies natural life by attempting to capture the behavioral essence of the constituent components of a living system, and endowing a collection of artificial components with similar behavioral repertoires. If organized correctly, the aggregate of artificial parts should exhibit the same dynamic behavior as the natural system.

This bottom-up modeling technique can be applied at any level of the hierarchy of living systems in the natural world—from modeling molecular dynamics on millisecond time-scales to modeling evolution in populations over millennia. At any such level, behavioral primitives are identified, rules for their behavior in response to local conditions are specified, the primitive behavors are organized similarly to their natural counterparts, and the behavior of interest is allowed to emerge "on the shoulders" of all of the myriad local interactions among low-level primitives taken collectively.

The ideal tool for this synthetic approach to the study of life is the computer. However, the traditional computer program—a centralized control structure with global access to a large set of predefined data-structures—is inappropriate for synthesizing life within computers. A new approach to computation is required, one that focuses on *ongoing dynamic behavior* rather than on any *final result*.

The essential features of computer-based Artificial Life models are:

- They consist of populations of simple programs or specifications.
- There is no single program that directs all of the other programs.
- Each program details the way in which a simple entity reacts to local situations in its environment, including encounters with other entities.
- There are *no* rules in the system that dictate global behavior.

- Any behavior at levels higher than the individual programs is therefore emergent.

To illustrate, consider modeling a colony of ants. We would provide simple specifications for the behavioral repertoires of different *castes* of ants, and create lots and lots of instances of each caste. We would start up this population of "antomata" (a term coined by Doyne Farmer) from some initial configuration within a simulated two-dimensional environment. From then on, the behavior of the system would depend entirely on the collective results of all of the local interactions between individual antomata and between individual antomata and features of their environment. There would be no single "drill-sergeant" antomaton choreographing the ongoing dynamics according to some set of high-level rules for colony-behavior. The behavior of the colony of antomata would emerge from out of the behaviors of the individual antomata themselves, just as in a real colony of ants.

Each of these antomata is a behavor. We will identify such behavors as simple *machines*. Artificial Life is concerned with tuning the behaviors of such low-level machines so that the behavior that emerges at the global level is *essentially* the same as some behavior exhibited by a natural living system.

THE ANIMATION OF MACHINES

animate *tr.v.* **1.** To give life to; fill with life.[1]

How are we to go about *animating* machines? How are we to go about bringing machines to life—or bringing life to machines?

The etymological ancestry of the term "animate" goes back to the Indo-European root ane meaning "to breathe," which is also ancestral to the Latin *animus*, denoting reason, mind, soul, spirit, life, breath, and etc. This corresponds to the notion that life is some kind of "energy," "force," or "essence"—that which leaves the physical body upon death, and lacking which mere material flesh and bone could not live. Thus, throughout historical time, the notion of giving life to some material object involved the act of "breathing" this mystical force or essence into an otherwise inanimate material body.

This notion that life was an extra "something" necessary *over and above* the detailed organization of a material organism is known as *vitalism*. Vitalism was championed especially strongly during the last two centuries as a defense against the growth of materialism and the scientific method which, especially after Darwin, threatened to explain everything in nature—including man. Worse yet, they threatened to do so without recourse to the supernatural or to God, but only by reference to everyday physical phenomena and materials, thereby removing man from his exalted position in this universe of otherwise mere material objects.

Biologists today reject vitalism, believing rather that life as we know it will eventually be explainable completely within the context of biochemistry. Thus, most biologists would agree—in principle anyway—with the following statement:

living organisms are nothing more than complex biochemical machines. However, they are different from the machines of our everyday experience. A living organism is ***not*** a single, complicated biochemical machine. Rather it must be viewed as a large ***population*** of relatively simple machines. The complexity of its behavior is due to the highly nonlinear nature of the interactions between all of the members of this polymorphic population. To animate machines, therefore, is not to "bring" life to *a* machine; rather it is to organize a population of machines in such a way that their interactive dynamics is "alive."

THE BEHAVIOR GENERATION PROBLEM

Artificial Life is concerned with generating *lifelike* behavior. Thus, it focuses on the problem of creating *behavior generators*. A good place to start is to identify the mechanisms by which behavior is generated and controlled in natural systems, and to recreate these mechanisms in artificial systems. This is the course we will take later in this paper.

The related field of Artificial Intelligence is supposedly concerned with generating *intelligent* behavior. It, too, focuses on the problem of creating behavior generators. However, although it initially looked to natural intelligence to identify its underlying mechanisms, these mechanisms were not known, nor are they today. Therefore, following an initial flirt with neural nets, AI became wedded to the only other known vehicle for the generation of complex behavior: the technology of serial computer programming. As a consequence, from the very beginning Artificial Intelligence embraced an underlying methodology for the generation of intelligent behavior that bore no demonstrable relationship to the method by which intelligence is generated in natural systems. In fact, AI has focused primarily on the production of intelligent *solutions* rather than on the production of intelligent *behavior*. There is a world of difference between these two possible foci.

By contrast, Artificial Life has the great good fortune that many of the mechanisms by which behavior arises in natural living systems are now known. There are still many holes in our knowledge, but the general picture is in place. Therefore, Artificial Life can remain true to natural life, and has no need to resort to the sort of infidelity that is only now coming back to haunt AI. Furthermore, Artificial Life is not concerned with building systems that reach some sort of solution. For AL systems, the *ongoing dynamics* is the behavior of interest, not the state ultimately reached by that dynamics.

The key insight into the natural method of behavior generation is gained by noting that *nature is fundamentally parallel*. This is reflected in the "architecture" of natural living organisms, which consist of many millions of parts, each one of which has its own behavioral repertoire. Living systems are highly distributed, and quite massively parallel. If our models are to be true to life, they must also be highly distributed and quite massively parallel. Indeed, it is unlikely that any other approach will prove viable.

PREVIEW

In the remainder of the paper, we will discuss a number of different aspects of the field of Artificial Life. First we will review the history of man's attempts to simulate life, trying to identify major threads of intellectual development that have proven essential to the enterprise.

Second, we will review the genotype/phenotype distinction in living organisms, viewing the genotype as a specification for *machinery*, and the phenotype as the behavior of the machinery so specified. We will then generalize the concepts of genotype and phenotype, so that we may apply them to the task of generating behavior in artificial systems.

Next, we will review the methodology of *recursively generated objects*, which makes natural use of the genotype/phenotype distinction, and we will give examples of its application to the generation of specific lifelike behaviors. Finally we discuss the problem of *generating* behavior generators, for which we turn to the process of evolution, and a discussion of Genetic Algorithms.

Throughout, the focus will be on machines and the behaviors that they are capable of generating. The field of Artificial Life is unabashedly mechanistic and reductionist. However, this *new mechanism*—based as it is on multiplicities of machines and on recent results in the fields of nonlinear dynamics, chaos theory, and the formal theory of computation—is vastly different from the mechanism of the last century.

HISTORICAL ROOTS OF ARTIFICIAL LIFE

Mankind has a long history of attempting to map the mechanics of his contemporary technology onto the workings of nature, trying to understand the latter in terms of the former.

The earliest mechanical technologies provided tools that extended man's physical abilities and greatly reduced the labor required to make a living. Early technologies yielded tools for moving water, for manipulating stone and timber, and for obtaining and processing food. Tools allowed mankind to alter the natural order of things to suit his purposes and needs.

However, there was much about nature that could not be altered—such as the progression of the seasons—in the face of which man had to alter *his* behavior to fit the natural order of things. In order to do so, it was useful to be able to build *models* of nature that allowed predictions to be made about when certain events would—or should—take place. Models were developed that allowed the anticipation of floods, the determination of when to plant and when to harvest food, and the prediction of the motion of the sun, moon, and planets through the heavens. Models allowed man to alter *his* behavior in order to take fuller advantage of the natural order of things.

Building a model is a little bit like building a machine of some sort. The art of modeling is a technology in itself, one which produced tools that extended man's mental abilities; tools of thought which greatly reduced the mental labor required to make a living. When the mechanical technology of the time was sufficiently advanced, these tools of thought were eventually committed to hardware, becoming physical machines. Thus, the history of machines involves a continuing process of rendering in hardware progressively more complicated sequences of actions—physical and/or mental—previously carried out solely by recourse to muscle and brain.

It is not surprising, therefore, that early models of life reflected the principal technology of their era. The earliest models were simple statuettes and paintings—works of art which captured the static form of living things. Later, these statues were provided with articulated arms and legs in the attempt to capture the dynamic form of living things. These simple statues incorporated no internal dynamics, requiring human operators to make them behave.

The earliest mechanical devices that were capable of generating their own behavior were based on the technology of water transport. These were the early Egyptian water clocks called *Clepsydra*. These devices made use of a rate-limited process —in this case the dripping of water through a fixed orifice—to indicate the progression of another process—the position of the sun. Ctesibius of Alexandria developed a water-powered mechanical clock around 135 B.C. which employed a great deal of the available hydraulic technology—including floats, a siphon, and a water-wheel-driven train of gears.

In the first century A.D., Hero of Alexandria produced a treatise on *Pneumatics*, which described, among other things, various gadgets in the shape of animals and humans that utilized pneumatic principles to generate simple movements.

However, it was really not until the age of mechanical clocks that artifacts exhibiting complicated internal dynamics became possible. Around 850 A.D., the *mechanical escapement* was invented, which could be used to regulate the power provided by falling weights. This invention ushered in the great age of clockwork technology. The earliest mechanical clock to make use of this regulation scheme seems to have been developed by Richard of Wallingford in 1326. Later, following Galileo, came pendulum clocks, and further ingenious developments in escapements for the regulation of rate. Throughout the Middle Ages and the Renaissance, the history of technology is largely bound up with the technology of clocks. Clocks often constituted the most complicated and advanced application of the technology of an era.[1]

Perhaps the earliest clockwork simulations of life were the so-called "Jacks" mechanical "men" incorporated in early clocks which would swing a hammer to strike the hour on a bell. The word "jack" is derived from "jaccomarchiadus," which means "the man in the suit of armour." These accessory figures retained their popularity even after the spread of clock dials and hands—to the extent that clocks

[1]This association of machinery with the inexorable flow of time may be largely responsible for the spectre of predestination associated with the early philosophy of mechanism.

were eventually developed in which the function of time-keeping was secondary to the control of large numbers of figures engaged in various activities, even acting out entire plays.

Finally, clockwork mechanisms appeared which had done away altogether with any pretense at time-keeping. These "automata" were entirely devoted to imparting lifelike motion to a mechanical figure or animal. These mechanical automaton simulations of life included such things as elephants, peacocks, singing birds, musicians, and even fortune tellers.

This line of development reached its peak in the famous duck of Vaucanson, described as "an artificial duck made of gilded copper who drinks, eats, quacks, splashes about on the water, and digests his food like a living duck."[2]

> There has never been a more famous automaton than Vaucanson's duck. In 1735 Jacques de Vaucanson arrived in Paris at the age of 26. Under the influence of contemporary philosophic ideas, he had tried, it seems, to reproduce life artificially.

Unfortunately, neither the duck itself nor any technical descriptions or diagrams remain that would give the details of its construction. The complexity of the mechanism is attested to by the fact that one single wing contained over 400 articulated pieces.

One of those called upon to repair Vaucanson's duck was a "mechanician" named Reichsteiner, who was so impressed with it that he went on to build a duck of his own—also now lost—which was exhibited in 1847. Here is an account of this duck's operation from the newspaper *Das Freie Wort*:

> After a light touch on a point on the base, the duck in the most natural way in the world begins to look around him, eyeing the audience with an intelligent air. His lord and master, however, apparently interprets this differently, for soon he goes off to look for something for the bird to eat. No sooner has he filled a dish with oatmeal porridge than our famished friend plunges his beak deep into it, showing his satisfaction by some characteristic movements of his tail. The way in which he takes the porridge and swallows it greedily is extraordinarily true to life. In next to no time the basin has been half emptied, although on several occasions the bird, as if alarmed by some unfamiliar noises, has raised his head and glanced curiously around him. After this, satisfied with his frugal meal, he stands up and begins to flap his wings and to stretch himself while expressing his gratitude by several contented quacks.
>
> But most astonishing of all are the contractions of the bird's body clearly showing that his stomach is a little upset by this rapid meal and the effects of a painful digestion become obvious. However, the brave little bird holds out, and after a few moments we are convinced in the most concrete manner

[2]See Chapuis[7] regarding all quotes concerning these mechanical ducks.

> that he has overcome his internal difficulties. The truth is that the smell which now spreads through the room becomes almost unbearable. We wish to express to the artist inventor the pleasure which his demonstration gave to us.

Figure 1 shows two views of one of the ducks—there is some controversy as to whether it is Vaucanson's or Reichsteiner's.

THE DEVELOPMENT OF CONTROL MECHANISMS

Out of the technology of the clockwork regulation of automata came the more general—and perhaps ultimately more important—technology of *process control*. As attested to in the descriptions of the mechanical ducks, some of the clockwork mechanisms had to control remarkably complicated actions on the part of the automata, not only *powering* them but *sequencing* them as well.

Control mechanisms evolved from early, simple devices—such as a lever attached to a wheel which converted circular motion into linear motion—to later, more complicated devices—such as whole sets of cams upon which would ride many interlinked mechanical arms, giving rise to extremely complicated automaton behaviors.

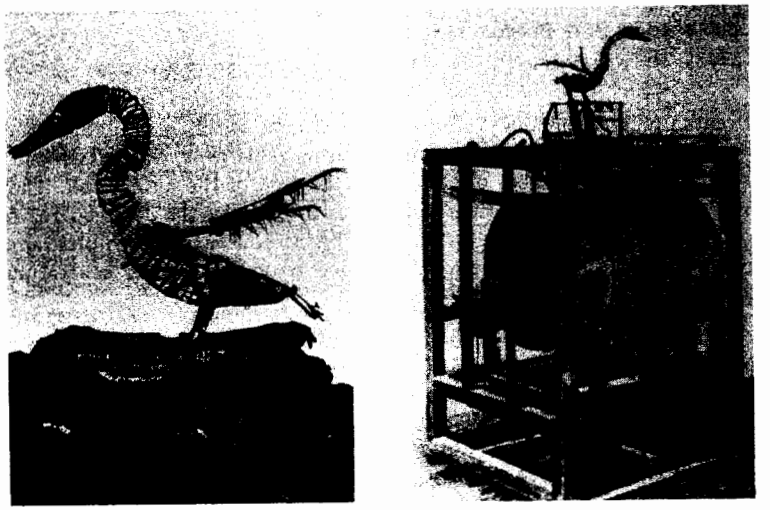


FIGURE 1 Two views of the mechanical duck attributed to Vaucanson. Printed in *Automata: A Historical and Technological Study* by Alfred Chapuis and Edmond Droz, published by B. A. Batsford Ltd.

FIGURE 2 Two views of a drawing automaton built by the Jaquet-Droz family. Printed in *Automata: A Historical and Technological Study* by Alfred Chapuis and Edmond Droz, published by B. A. Batsford Ltd.

Eventually *programmable controllers* appeared, which incorporated such devices as interchangeable cams, or drums with movable pegs, with which one could program arbitrary sequences of actions on the part of the automaton. The writing and picture drawing automata of Figure 2, built by the Jaquet-Droz family, are examples of programmable automata. The introduction of such programmable controllers was one of the primary developments on the road to general purpose computers.

ABSTRACTION OF THE LOGICAL "FORM" OF MACHINES

During the early part of the 20^{th} century, the formal application of logic to the mechanical process of arithmetic lead to the abstract formulation of a "procedure." The work of Church, Kleene, Gödel, Turing, and Post formalized the notion of a logical sequence of steps, leading to the realization that the essence of a mechanical process—the "thing" responsible for its dynamic behavior—is not a thing at all, but an abstract control structure, or "program"—a sequence of simple actions selected from a finite repertoire. Furthermore, it was recognized that the essential

features of this control structure could be captured within an abstract set of rules—a formal specification— without regard to the material out of which the machine was constructed. The "logical form" of a machine was separated from its material basis of construction, and it was found that "machineness" was a property of the former, not of the latter. Of course, the principle assumption made in Artificial Life is that the "logical form" of an organism can be separated from its material basis of construction, and that "aliveness" will be found to be a property of the former, not of the latter.

Today, the formal equivalent of a "machine" is an *algorithm:* the logic underlying the dynamics of an automaton, regardless of the details of its material construction. We now have many formal methods for the specification and operation of abstract machines, such as programming languages, formal language theory, automata theory, recursive function theory, etc. Many of these have been shown to be logically equivalent.

Once we have learned to think of machines in terms of their abstract, formal specifications, we can turn around and view abstract, formal specifications as potential machines. In mapping the machines of our common experience to formal specifications, we have by no means exhausted the space of *possible* specifications. Indeed, most of our individual machines map to a very small subset of the space of specifications—a subset largely characterized by methodical, boring, uninteresting dynamics. When placed together in aggregates, however, even the simplest machines can participate in *extremely* complicated dynamics.

GENERAL PURPOSE COMPUTERS

Various threads of technological development—programmable controllers, calculating engines, and the formal theory of machines—have come together in the general purpose, stored program computer. Programmable computers are extremely general behavior generators. They have no intrinsic behavior of their own. Without programs, they are like formless matter. They must be told how to behave. By submitting a program to a computer—that is: by giving it a formal specification for a machine—we are telling it to behave as if it were the machine specified by the program. The computer then "emulates" that more specific machine in the performance of the desired task. Its great power lies in its plasticity of behavior. If we can provide a step-by-step specification for a specific kind of behavior, the chameleon-like computer will exhibit that behavior. Computers should be viewed as *second-order* machines—given the formal specification of a first-order machine, they will "become" that machine. Thus, the space of possible machines is directly available for study, at the cost of a mere formal description: computers "realize" abstract machines.

FORMAL LIMITS OF MACHINE BEHAVIORS

Although computers—and by extension other machines—are capable of exhibiting a bewilderingly wide variety of behaviors, we must face two fundamental limitations on the kinds of behaviors that we can expect of computers.

The first limitation is one of *computability in principle.* There are certain behaviors that are "uncomputable"—behaviors for which *no* formal specification can be given for a machine which will exhibit that behavior. The classic example of this sort of limitation is Turing's famous *halting problem:* can we give a formal specification for a machine which, when provided with the description of *any* other machine together with its initial state, will—by inspection alone—determine whether or not that machine will reach its halt state? Turing proved that no such machine can be specified. Rice and others[23] have extended this undecidability result to the determination—by inspection alone—of *any* non-trivial property of the future behavior of an arbitrary machine.

The second limitation is one of *computability in practice.* There are many behaviors for which we do not know how to specify a sequence of steps which will cause the computer to exhibit that behavior. We can automate what we know how to do already, but there is much that we do not know how to do. Thus, although a formal specification for a machine which will exhibit a certain behavior may be possible *in principle*, we have no formal procedure for producing that formal specification in practice, short of a trial-and-error search through the space of possible descriptions.

We need to separate the notion of a formal specification of a machine—that is, a specification of the *logical structure* of the machine—from the notion of a formal specification of a machine's behavior—that is, a specification of the *sequence of transitions* that the machine will undergo. We have formal systems for the former, but not for the latter. In general, we can neither derive behaviors from specifications nor derive specifications from behaviors.

The moral is: in order to determine the behavior of some machines, there is no recourse but to run them and see how they behave! This has consequences for the methods by which we (or nature) go about *generating* behavior generators themselves, which we will take up in the section on evolution.

FROM MECHANICS TO LOGIC

With the development of the general purpose computer, attention turned from the *mechanics* of life to the *logic* of life. The computer's tremendous capacity for emulation made it possible to explore the behaviors of a great many possible machines—machines which would probably never have been committed to hardware. The 1950's and 1960's saw an explosion of interest in computer and electro-mechanical models of life.

VON NEUMANN AND AUTOMATA THEORY The first computational approach to the generation of lifelike behavior was due to the brilliant Hungarian mathematician John von Neumann. In the words of his colleague Arthur W. Burks, von Neumann was interested in the general question[6]:

> What kind of logical organization is sufficient for an automaton to reproduce itself? This question is not precise and admits to trivial versions as well as interesting ones. Von Neumann had the familiar natural phenomenon of self-reproduction in mind when he posed it, but he was not trying to simulate the self-reproduction of a natural system at the level of genetics and biochemistry. *He wished to abstract from the natural self-reproduction problem its logical form.* [emphasis added]

In von Neumann's initial thought experiment (his "kinematic model"), a machine floats around on the surface of a pond, together with lots of machine parts. The machine is a *universal constructor:* given the description of any machine, it will locate the proper parts and construct that machine. If given a description of itself, it will construct a copy of itself. This is not quite self-reproduction, however, because the offspring machine will not have a description of itself and hence could not go on to construct another copy. So, von Neumann's machine also contains a *description copier:* once the offspring machine has been constructed, the "parent" machine constructs a copy of the description that it worked from and attaches it to the offspring machine. This constitutes genuine self-reproduction. However, von Neumann decided that this model did not properly distinguish the logic of the process from the material of the process, and looked about for a completely formal system within which to model self-reproduction.

Stan Ulam—one of von Neumann's colleagues at Los Alamos who also investigated dynamic models of pattern production and competition[47]—suggested an appropriate formalism, which has come to be known as a *cellular automaton* (CA). In brief, a CA model consists of a regular lattice of *finite automata*, which are the simplest formal models of machines. A finite automaton can be in only one of a finite number of states at any given time, and its transitions between states from one time step to the next are governed by a *state-transition table:* given a certain input and a certain internal state, the state-transition table specifies the state to be adopted by the finite automaton at the next time step. In a CA, the necessary input is derived from the states of the automata at neighboring lattice points. Thus, the state of an automaton at time $t + 1$ is a function of the states of the automaton itself and its immediate neighbors at time t. All of the automata in the lattice obey the same transition table and every automaton changes state at the same instant, time step after time step. CA's are good examples of the kind of computational paradigm sought after by Artificial Life: bottom-up, parallel, local-determination of behavior.

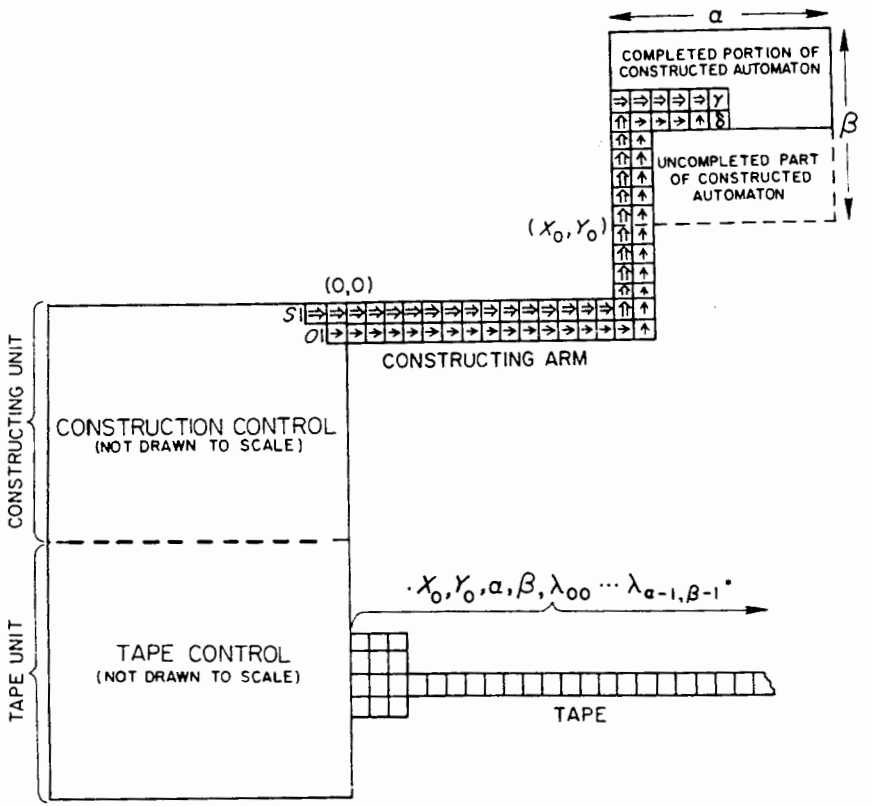


FIGURE 3 Schematic diagram of von Neumann's self-reproducing CA configuration. From Burks,[6] reprinted courtesy of University of Illinois Press.

Von Neumann was able to embed the logical equivalent of his kinematic model as an initial pattern of state assignments within a large CA lattice using 29 states per cell (Figure 3). Although von Neumann's work on self-reproducing automata was left incomplete at the time of his death, Arthur Burks organized what had been done, filled in the remaining details, and published it together with a transcription of von Neumann's 1949 lectures at the University of Illinois entitled "Theory and Organization of Complicated Automata," in which he gives his views on various problems related to the study of complex systems in general.[48]

Von Neumann's CA model was a constructive proof that an essential characteristic behavior of living things—self-reproduction—*was* achievable by machines.

Furthermore, he determined that any such method must make use of the information contained in the description of the machine in two fundamentally different ways:

- INTERPRETED, as instructions to be executed in the construction of the offspring.
- UNINTERPRETED, as passive data to be duplicated to form the description given to the offspring.

Of course, when Watson and Crick unraveled the mystery of DNA, they discovered that the information contained therein was used in precisely these two ways in the processes of transcription/translation and replication.

In describing his model, von Neumann pointed out that[6]:

> By axiomatizing automata in this manner, one has thrown half of the problem out the window, and it may be the more important half. One has resigned oneself not to explain how these parts are made up of real things, specifically, how these parts are made up of actual elementary particles, or even of higher chemical molecules.

Whether or not the baby has been disposed of depends on the questions we are asking. If we are concerned with explaining how the life that we know emerges from the known laws of physics and organic chemistry, then indeed the baby has been tossed out. But, if we are concerned with the more general problem of explaining how lifelike behaviors emerge out of low-level interactions within a population of logical primitives, the baby is still with us.

WIENER AND CYBERNETICS The technology of process control—which in its discrete form lead to von Neumann's automaton approach—lead in its continuous form to *Cybernetics*, proposed by Norbert Wiener as "the study of control and communication in the animal and the machine."[30,51]

The term "cybernetics" is derived from the Greek $\chi\upsilon\beta\varepsilon\rho\nu\acute{\eta}\tau\eta\varsigma$—or *steersman*—which was used by Plato in the sense of "government." For Wiener, the word imparted a sense of goal-oriented, purposeful control of behavior.

Cybernetics had its origin in Wiener's war-related work on the control of anti-aircraft fire. An anti-aircraft gun must fire, not at the current position of the target, but at the spot to which the aircraft will have moved during the flight of the shell. Thus, the controller must predict, or "anticipate," the future path of the airplane. In working out a general mathematical basis for predicting the probable future course of an observed time-series, Wiener and his colleague Julian Bigelow realized that it was important to collect information about the *deviations* between predicted motion and actual motion. These deviations could then be *fed-back* as input to the predictor and treated as corrections to further predictions.

Wiener and Bigelow also realized that improper treatment of the corrective feedback could result in two different forms of "pathological" behavior on the part

of the controller. If the controller is not sufficiently sensitive to the corrective feedback, the corrections will not keep pace with the deviations, and the gap between predicted motion and actual motion will continue to grow. On the other hand, if the controller is *overly* sensitive to the feedback, each corrective maneuver will be too large, resulting in larger and larger deviations, first to one side and then to the other. Eventually, this will result in the system becoming hopelessly engaged in wild oscillations.

The first form of pathological behavior was similar to the condition in humans and animals known as *Ataxia*, in which internal sensory feedback from a limb is insufficient or absent. Wiener and Bigelow asked Arturo Rosenbluth whether the second form of pathology was also known to occur in humans or animals. Rosenbluth answered immediately that "purpose tremor," sometimes observed in patients who had suffered injuries to the cerebellum, was just such a pathological condition.

Wiener, Bigelow, and Rosenbluth were thus lead to the realization that *feedback* played a similar role in a wide variety of natural and artificial systems, and that a comprehensive program of interdisciplinary research into the functions and especially the *dysfunctions* of goal-oriented—or "teleological"—*machines* could reveal a great deal about the nature of similar mechanisms operating in living organisms.

Von Neumann's program of the application of *discrete* mathematics to the *synthesis* of behavior and Wiener's program of the application of *continuous* mathematics to the *analysis* of behavior are entirely complementary endeavors, and there is quite a large area of potential overlap between them. Indeed, many of the same phenomena can be represented equally well within either of the two methodological approaches, and it was one of von Neumann's dreams to develop a continuous version of his discrete, automaton approach.

THE POST-WAR PERIOD In the years following the publication of von Neumann's and Wiener's approaches, other researchers followed up on the basic ideas—extending them, simplifying them, and proposing alternative models for the explanation and synthesis of lifelike behaviors

James Thatcher completed a simplified version of von Neumann's self-replicating CA model.[44] E.F. Codd developed a version using only eight states per cell.[8] Richard Laing demonstrated a clever variation on the von Neumann plan in which a machine first constructs a description of itself by self-inspection, and then uses that description to construct a copy of itself.[26] This latter model would be capable of passing on acquired characteristics in Lamarckian fashion, unlike von Neumann's model. Laing also developed a system of self-reproducing artificial organisms based on what he called *artificial molecular machines*—dynamic "program tapes" interacting within a sort of "soup." This model attempted to combine in one system the best features of von Neumann's CA and kinematic models.[25]

Others developed self-reproducing models based on different primitive elements. Michael Arbib[2] developed a 2D-lattice model of self-reproduction in which each lattice point consists of a set of registers in which instructions are stored. The *contents* of these register sets may be shifted into the registers of neighboring lattice

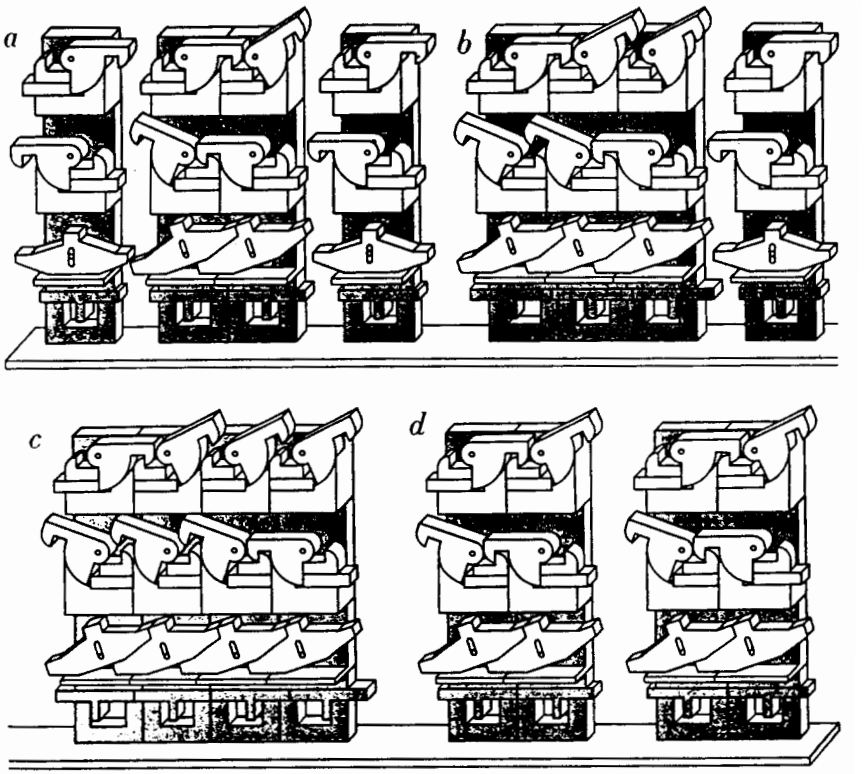


FIGURE 4 One of Penrose's devices for illustrating self-reproduction. Reprinted courtesy of Scientific American.

points. In fact, whole sets of lattice points may shift their contents in any one c the four cardinal directions simultaneously, the contents moving as a rigid unit, a if they were held together by chemical bonds.

L.S. Penrose built a series of clever mechanical models illustrating a kind of self reproduction.[33] The basic system consists of a box filled with tilting blocks. Th blocks have hooks which can engage other blocks in several different arrangements When a "seed"—consisting of a pair of blocks hooked together in one of the possibl arrangements—is placed into a box full of unhooked blocks and the box is shake vigorously, the seed will induce the rest of the blocks to hook up in pairs exhibitin the same conformation as the seed. One of his models is illustrated in Figure 4.

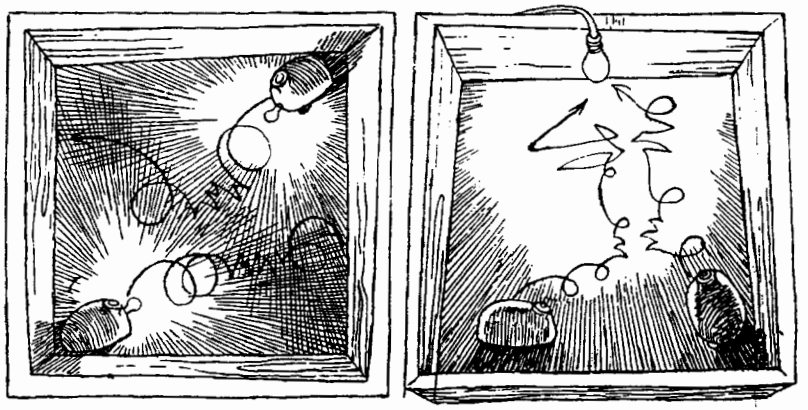


FIGURE 5 Schematic diagram of interactions between two of Grey Walter's electronic turtles. Reprinted courtesy of Scientific American.

Homer Jacobson built a self-reproducing train set.[24] In this model, linked cars chug around an oval track together with unlinked cars. A linked set of cars will pull off onto a siding and direct the construction of a similarly linked set of cars by switching passing unlinked cars onto an adjacent siding in the correct order. Once the construction is complete, both linked sets of cars will reenter the oval track.

Grey Walter built a pair of electronic "turtles" named Elmer and Elsie: "imitations of life" which exhibited "free will."[49,50] These turtles would wander around, attracted to dim light, but repelled by bright light,[3] until their batteries got low, in which case they would home in on their brightly lit "kennels," plug into a recharger, and recharge their batteries. When Walter attached lights to the turtles themselves, the resulting interactive dynamics became quite complex (Figure 5).

Walter dubbed his turtles *Machina speculatrix*. In his words[49]:

> These machines are perhaps the simplest that can be said to resemble animals. Crude though the are, they give an eerie impression of purposefulness, independence, and spontaneity... Perhaps we flatter ourselves in thinking that man is the pinnacle of an estimable creation. Yet as our imitation of life becomes more faithful our veneration of its marvelous processes will not necessarily become less sincere.

Samuel's famous checker playing program incorporated a learning algorithm based on adaptation by natural selection.[38] This program quickly learned to play

[3]Cf. Braitenberg's *Vehicles*.[5]

checkers better than Samuel. Holland[21,22] has investigated many applications of adaptation by natural selection, and proposed the class of machine-learning techniques known as "genetic algorithms," of which we will have more to say in the section on evolution.

Of course, much of the early work in Artificial Life was also ancestral to Artificial Intelligence. This is certainly true of Samuel's and Holland's work. Other common ancestors include McCulloch and Pitts' nerve-net models,[31] Rosenblatt's work on perceptrons,[37] and Minsky and Papert's book on perceptrons.[32]

Walter Stahl built several models of cellular activity in which "Turing machines are used to model 'algorithmic enzymes' which transform biochemicals represented as letter strings."[40,41] In one work, an entire artificial cell "metabolizes" energy strings and reproduces itself.[43] Stahl also looked into unsolvable problems for a cell automaton.[42]

In the late 1960's, Aristid Lindenmayer introduced his mathematical models of cellular interaction in development, now known simply as *L-systems*. These relatively simple models are capable of exhibiting remarkably complex developmental histories, supporting intercellular communication and differentiation. Many applications have been found, especially in modeling the development of the branching structure of plants. Some simple examples of L-systems are given in the section on Recursively Generated Objects, as well as in Lindenmayer's contribution to these proceedings.

Since 1970, Michael Conrad and various collaborators have developed an increasingly sophisticated series of "artificial world" models for the study of adaptation, evolution, and population dynamics within artificial ecosystems (see Conrad and Strizich[9] and Rizki and Conrad[36] Later models have focused on individual fitness as an emergent property of the system.

One unfortunate consequence of the explosive progress in the technology of computation was that as more and more energy was devoted to developing practical applications for discoveries that were originally made in the attempt to model natural processes, less and less energy was devoted to the sorts of studies that had lead to these discoveries in the first place.

Thus, Chomsky's formal language theory was applied to the specification of programming languages and in the development of compilers. Cellular automata were applied to the task of image processing and, in general, the pursuit of nature was set aside in favor of developing practical applications of the original products of that pursuit. As a consequence, the initial tidal wave of research involving the computer-based study of life receded, leaving behind various, isolated "tidal pools" of research, which hung on largely due to the persistence of individual researchers who made their living doing something eminently more practical from an engineering point of view.

From about the mid-1970's until quite recently, although there has been a good deal of work involving computer-based models of living systems, much of this research has taken place within the confines of a wide variety of disciplines, largely in isolation from other such efforts. Diffusion of results across these disciplinary

boundaries has been slow or nonexistant. Furthermore, models that produced lifelike behavior but which did not model some specific aspect of natural life were generally treated as oddities—interesting to be sure, but of questionable scientific relevance. There was no general recognition that such systems might be worthy of study on their own rights; that the study of *possible* life might be every bit as relevant to the scientific understanding of life as the study of *actual* life. Instead, individuals have pursued such models out of their own personal interest, on their own personal time, and—recently—on their own personal computers.

Many of these pursuits were reported to the larger scientific community by Martin Gardner in his "Mathematical Games" column in *Scientific American*. One such system worthy of note is John Conway's cellular automaton game of LIFE.[19,20] In this system, a cell will turn "on" if exactly three of its eight neighbors are "on" and it will stay "on" so long as either 2 or 3 of its neighbors are "on," otherwise it will turn "off." This CA system has been experimented with extensively.[3,34] Many of the configurations seem to have a life of their own. Perhaps the single most remarkable structure is known as the *glider*, a quasi-periodic configuration of period 4 which displaces itself diagonally with respect to the fixed lattice of cells (see Figure 6).

The glider is one instance of the general class of propagating structures in CA. These propagating information structures are effectively simple machines—*virtual machines*—which crawl around the lattice like so many ants, interacting with other such machines and with the more passive structures in the array. Their behavior is reminiscent of the actions of biomolecules—especially enzymes—in their capacity for recognizing and altering other structures they encounter in their wanderings, including other *propagating* structures.[29]

Since Martin Gardner's retirement, A.K. Dewdney has taken up the cause of reporting work in Artificial Life in his *Scientific American* column "Computer Recreations." Although many of the systems reported were initially conceived as simple computer games, several—such as Core Wars,[11,13,15] Wator,[12] Flibs,[14] 3D-LIFE,[16] etc.—involve the bottom-up determination of lifelike behaviors, and are worthy of more serious investigation.

There are many other works that could be discussed, but we have reached the present day and the current state of the field, which these proceedings as a whole are meant to review. Therefore, we will bring this historical survey to a close with the following summary.

FIGURE 6 Glider propagating with respect to a fixed cell (○).

THE ROOTS OF COMPLEX BEHAVIOR

Since the beginning of recorded history, man has attempted to build imitations of living things. Early attempts captured the "form" of living things in statuary and paintings, while later attempts sought to "animate" these static forms by the use of hidden machinery.

It is quite clear from a study of the history of attempts to build "living" artifacts that the material out of which the artifact was constructed was considered irrelevant—it was the model's dynamic behavior that mattered. The elusive holy grail was the construction of a mechanism which, regardless of its constituent material, *behaves* like a living thing.

Most of the more serious attempts, particularly during the long history of clockwork automata, involved a central "program" of some kind which was responsible for the model's dynamic behavior. Whether it was a rotating drum with pegs tripping levers in sequence, a set of motor driven cams, or some other mechanism—the tune to which the automaton danced was "called" by central control machinery.

Therein lay the source of the failure of these models and, in my view, the source of failure of the whole program of modeling complex systems that followed, right up to—and most especially including—much of the work in Artificial Intelligence. The most promising approaches to modeling complex systems like life or intelligence are those which have dispensed with the notion of a centralized global controller, and have focused instead on mechanisms for the *distributed* control of behavior.

BIOLOGICAL AUTOMATA

Organisms have been compared to extremely complicated and finely tuned biochemical machines. Since we know that it is possible to abstract the logical form of a machine from its physical hardware, it is natural to ask whether it is possible to abstract the logical from of an organism from its biochemical wetware. The field of Artificial Life is devoted to the investigation of this question.

In the following sections we will look at the manner in which behavior is generated in bottom-up fashion in living systems. We then generalize the mechanisms by which this behavior generation is accomplished, so that we may apply them to the task of generating behavior in artificial systems.

We will find that the essential machinery of living organisms is quite a bit different from the machinery of our own invention, and we would be quite mistaken to attempt to force our preconceived notions of abstract machines onto the machinery of life. The difference, once again, lies in the exceedingly parallel and distributed nature of the operation of the machinery of life, as contrasted with the singularly serial and centralized control structures associated with the machines of our invention.

GENOTYPES AND PHENOTYPES

The most salient characteristic of living systems, from the behavior generation point of view, is the *genotype/phenotype* distinction. The distinction is essentially one between a specification of machinery—the genotype—and the behavior of that machinery—the phenotype.

The *genotype* is the complete set of genetic instructions encoded in the linear sequence of nucleotide bases that makes up an organism's DNA. The *phenotype* is the physical organism itself —the structures that emerge in space and time as the result of the interpretation of the genotype in the context of a particular environment. The process by which the phenotype develops through time under the direction of the genotype is called *morphogenesis*. The individual genetic instructions are called *genes*, and consist of short stretches of DNA. These instructions are "executed"—or *expressed*—when their DNA sequence is used as a template for transcription. In the case of protein synthesis, transcription results in a duplicate nucleotide strand known as a *messenger RNA*—or *mRNA*—constructed by the process of base-pairing. This mRNA strand may then be modified in certain ways before it makes its way out to the cytoplasm where, at bodies known as *ribosomes*, it serves as a template for the construction of a linear chain of *amino acids*. The resulting *polypeptide* chain will fold up on itself in some complex manner, forming a tightly packed molecule known as a *protein*. The finished protein detaches from the ribosome and may go on to serve as a passive structural element in the cell, or may have a more active role as an *enzyme*. Enzymes are *the* functional molecular "operators" in the logic of life.

One may consider the genotype as a largely unordered "bag" of instructions, each one of which is essentially the specification for a "machine" of some sort—passive or active. When instantiated, each such machine will enter into the ongoing logical fray in the cytoplasm, consisting largely of local interactions between other such machines. Each such instruction will be "executed" when its own triggering conditions are met and will have specific, local effects on structures in the cell. Furthermore, each such instruction will operate within the context of all of the other instructions that have been—or are being—executed.

The phenotype, then, consists of the structures and dynamics that emerge through time in the course of the execution of the parallel, distributed "computation" controlled by this genetic bag of instructions. Since gene's interactions with one another are highly nonlinear, the phenotype is a nonlinear function of the genotype, and the label for that nonlinear function is "development."

GENERALIZED GENOTYPES AND PHENOTYPES

In the context of Artificial Life, we need to generalize the notions of *genotype* and *phenotype*, so that we may apply them in non-biological situations. We will use the term *generalized genotype*—or GTYPE—to refer to any largely unordered set of low-level rules, and we will use the term *generalized phenotype*—or PTYPE—to refer to the behaviors and/or structures that emerge out of the interactions among these low-level rules when they are activated within some specific environment.

The GTYPE, essentially, is the specification for a set of machines, while the PTYPE is the behavior that results as the machines interact with one another in the context of a specific environment. This is the bottom-up approach to the generation of behavior. A set of entities is defined and each entity is endowed with a specification for a simple behavioral repertoire—a GTYPE—which contains instructions that detail its reactions to a wide range of *local* encounters with other such entities or with specific features of the environment. Nowhere is the behavior of the set of entities as a whole specified. The global behavior of the aggregate—the PTYPE— emerges out of the collective interactions among individual entities.

It should be noted that the PTYPE is a multilevel phenomenon. First, there is the PTYPE associated with each particular instruction—the effect that instruction has on the entity's behavior when it is expressed. Second, there is the PTYPE associated with each individual entity—its individual behavior within the aggregate. Third, there is the PTYPE associated with the behavior of the aggregate as a whole.

This is true for natural systems as well. We can talk about the phenotypic trait associated with a particular gene, we can identify the phenotype of an individual cell, and we can identify the phenotype of an entire multicellular organism—its body, in effect. PTYPES *should* be complex and multilevel. If we want to simulate life, we should expect to see hierarchical structures emerge in our simulations. In general, phenotypic traits at the level of the whole organism will be the result of many nonlinear interactions between genes, and there will be no single gene to which one can assign responsibility for the vast majority of phenotypic traits.

In summary, GTYPES are low-level rules for behav*ors*—i.e., abstract specifications for "machines"—which will engage in local interactions within a large aggregate of other such behav*ors*. PTYPES are the behav*iors*—the structures in time and space—that *develop* out of these nonlinear, local interactions (Figure 7).

UNPREDICTABILITY OF PTYPE FROM GTYPE

Nonlinear interactions between the objects specified by the GTYPE provide the basis for an extremely rich variety of possible PTYPES. PTYPES draw on the full combinatorial potential implicit in the set of possible interactions between low-level rules. The other side of the coin, however, is that we cannot predict the PTYPES that will emerge from specific GTYPES given specific initial structures. If we wish to maintain the property of predictability, then we must restrict severely the nonlinear dependence of PTYPE on GTYPE, but this forces us to give up the combinatorial richness of possible PTYPES. Therefore, a trade-off exists between behavioral richness and predictability.

As discussed previously, we know that it is impossible in the general case to determine *any* nontrivial property of the future behavior of a sufficiently powerful computer from a mere inspection of its program and its initial state alone.[23] A

Turing machine—the formal equivalent of a general purpose computer—can be captured within the scheme of GTYPE/PTYPE systems by identifying the machine's transition table as the GTYPE and the resulting computation as the PTYPE. From this we can deduce that in the general case it will not be possible to determine, by inspection alone, any nontrivial feature of the PTYPE that will emerge from a given GTYPE in the context of a particular initial configuration. In general, the only way to find out anything about the PTYPE is to start the system up and watch what happens as the PTYPE develops under control of the GTYPE.

Similarly, it is not possible in the general case to adduce which specific alterations must be made to a GTYPE to effect a desired change in the PTYPE. The problem is that any specific PTYPE trait is, in general, an effect of many, many nonlinear interactions between the behavioral primitives of the system. Consequently, given an arbitrary proposed change to the PTYPE, it may be impossible to determine by any formal procedure exactly what changes would have to be made to the GTYPE to effect that—and *only* that— change in the PTYPE. It is not a practically computable problem. There is no way to calculate the answer—short of exhaustive search—*even though there may be an answer!*

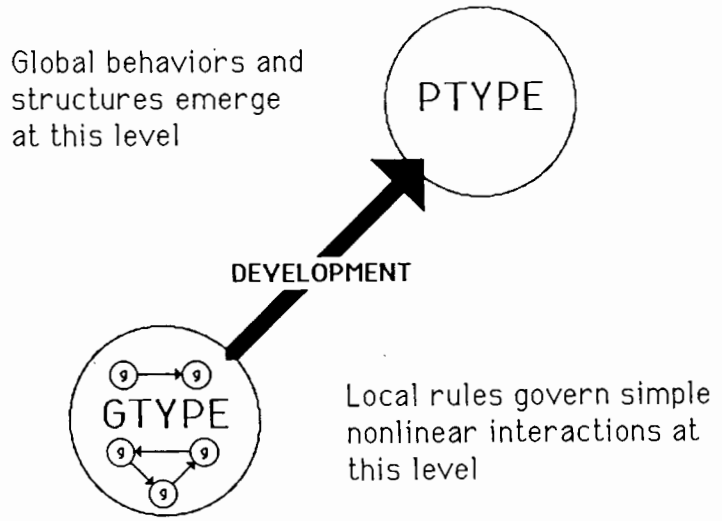


FIGURE 7 The relationship between GTYPE and PTYPE.

The only way to proceed in the face of such an unpredictability result is by a process of trial and error. However, some processes of trial and error are more efficient than others. In natural systems, trial and error are interlinked in such a way that error guides the choice of trials under the process of evolution by natural selection. It is quite likely that this is the *only* efficient, *general* procedure that could find GTYPES with specific PTYPE traits.

RECURSIVELY GENERATED OBJECTS

In the previous section, we described the distinction between genotype and phenotype, and we introduced their generalizations in the form of GTYPE's and PTYPE's. In this section, we will review a general approach to building GTYPE/PTYPE systems based on the methodology of *recursively generated objects.*

A major appeal of this approach is that it arises naturally from the GTYPE/PTYPE distinction: the local developmental rules—the recursive description itself—constitute the *GTYPE*, and the developing structure—the recursively generated object or behavior itself—constitutes the *PTYPE*.

Under the methodology of recursively generated objects, the "object" is a structure that has sub-parts. The rules of the system specify how to modify the most elementary, "atomic" sub-parts, and are usually sensitive to the *context* in which these atomic sub-parts are embedded. That is, the "neighborhood" of an atomic sub-part is taken into account in determining which rule to apply in order to modify that sub-part. It is usually the case that there are no rules in the system whose context is the entire structure; that is, there is no use made of *global* information. Each piece is modified solely on the basis of its own state and the state of the pieces "nearby."

Of course, if the initial structure consists of a single part—as might be the case with the initial seed—then the context for applying a rule is necessarily global. The usual situation is that the structure consists of *many* parts, only a local sub-set of which determine the rule that will be used to modify any one sub-part of the structure.

A recursively generated object, then, is a kind of PTYPE, and the recursive description that generates it is a kind of GTYPE. The PTYPE will emerge under the action of the GTYPE, developing through time via a process of morphogenesis.

We will illustrate the notion of recursively generated objects with examples taken from the literature on L-systems, cellular automata, and computer animation.

EXAMPLE 1: LINDENMAYER SYSTEMS

Lindenmayer systems (L-systems) consist of sets of rules for rewriting strings of symbols, and bear strong relationships to the formal grammars treated by Chomsky. We will give several examples of L-systems illustrating the methodology of

recursively generated objects (for a more detailed review, see the paper by Lindenmayer and Prusinkiewicz in these proceedings).

In the following "$X \rightarrow Y$" means that one replaces every occurrence of symbol "X" in the structure with string "Y." Since the symbol "X" may appear on the right as well as the left sides of some rules, the set of rules can be applied "recursively" to the newly rewritten structures. The process can be continued *ad infinitum* although some sets of rules will result in a "final" configuration when no more changes occur.

SIMPLE LINEAR GROWTH

Here is an example of the simplest kind of L-system. The rules are *context free*, meaning that the context in which a particular part is situated is *not* considered when altering it. There must be only one rule per part if the system is to be deterministic.

The rules: (the "recursive description" or GTYPE):

```
1)  A -> CB
2)  B -> A
3)  C -> DA
4)  D -> C
```

When applied to the initial seed structure "A," the following structural history develops (each successive line is a successive time step):

time	structure	rules applied (L to R)
0	A	initial "seed"
1	C B	rule 1 replaces A with CB
2	D A A	rule 3 replaces C with DA & rule 2 replaces B with A
3	C C B C B	rule 4 replaces D with C & rule 1 replaces the two
4	....(etc).....	A's with CB's

And so forth.

The "PTYPE" that emerges from this kind of recursive application of a simple, local rewriting rule can get extremely complex. These kinds of grammars (whose rules replace single symbols) have been shown to be equivalent to the operation of a finite state machine. With appropriate restrictions, they are also equivalent to the "regular languages" defined by Chomsky.

BRANCHING GROWTH L-systems incorporate meta-symbols to represent branching points, allowing a new line of symbols to branch off from the main "stem."

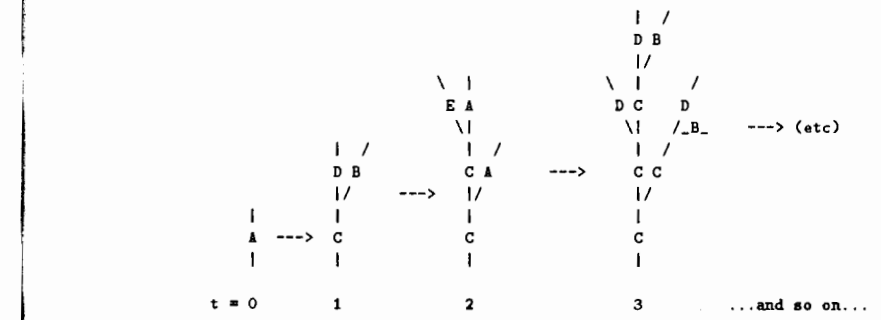
The following grammar produces branching structures. The "( )" and "[ ]" notations indicate left and right branches, respectively, and the strings within them indicate the structure of the branches themselves. The rules—or GTYPE:


```
1) A -> C[B]D
2) B -> A
3) C -> C
4) D -> C(E)A
5) E -> D
```

When applied to the starting structure "A," the following sequence develops (using linear notation):

time	structure	rules applied (L to R)
0	A	initial "seed".
1	C[B]D	rule 1.
2	C[A]C(E)A	rules 3,2,4.
3	C[C[B]D]C(D)C[B]D	rules 3,1,3,5,1.
4	C[C[A]C(E)A]C(C(E)A)C[A]C(E)A	rules 3,3,2,4,3,4,3,2,4.

In two dimensions, the structure develops as follows:



Note that at each step, *every symbol is replaced*, even if just by another copy of itself. This allows all kinds of complex phenomena, such as signal propagation along the structure, which will be demonstrated in the next example.

SIGNAL PROPAGATION In order to propagate signals along a structure, one must have something more than just a single symbol on the left-hand side of a rule. When there is more than one symbol on the left-hand side of a rule, the rules are *context sensitive*, i.e., the "context" within which a symbol occurs (the symbols next to it) are important in determining what the replacement string is. The next example illustrates why this is critical for signal propagation.

In the following example, the symbol in "{ }'s" is the symbol (or string of symbols) to be replaced, the rest of the left-hand side is the context, and the symbols "[" and "]" indicate the left and right ends of the string, respectively. Suppose the rule set contains the following rules:

```
1) [{C} -> C     a "C" at the left-end of the string remains a "C"
2) C{C} -> C     a "C" with a "C" to its left remains a "C"
3) *{C} -> *     a "C" with an "*" to its left becomes an "*"
4) {*}C -> C     an "*" with a "C" to its right becomes a "C"
5) {*}] -> *     an "*" at the right end of the string remains an "*"
```

Under these rules, the initial structure "*CCCCCCC" will result in the "*" being propagated to the right, as follows:

time	structure
0	*CCCCCCC
1	C*CCCCCC
2	CC*CCCCC
3	CCC*CCCC
4	CCCC*CCC
5	CCCCC*CC
6	CCCCCC*C
7	CCCCCCC*

This would not be possible without taking the "context" of a symbol into account. In general, these kinds of grammars are equivalent to Chomsky's "context-sensitive" or "Turing" languages, depending on whether or not there are any restrictions on the kinds of strings on the left and right hand sides.

The capacity for signal propagation is extremely important, for it allows arbitrary computational processes to be embedded within the structure, which may directly affect the structure's development. The next example demonstrates how embedded computation can affect development.

EXAMPLE 2: CELLULAR AUTOMATA

Cellular automata (CA) provide another example of the recursive application of a simple set of rules to a structure. In a CA, the structure that is being updated is the entire universe: a lattice of finite automata. The local rule set—the GTYPE—in this case is the transition function obeyed homogeneously by every automaton in the lattice. The local context taken into account in updating the state of each automaton is the state of the automata in its immediate neighborhood. The transition function for the automata constitutes a *local physics* for a simple, discrete space/time universe. The universe is updated by applying the local physics to each "cell" of its structure over and over again. Thus, although the physical structure itself doesn't develop over time, its *state* does.

Within such universes, one can embed all manner of processes, relying on the context sensitivity of the rules to local neighborhood conditions to propagate information around within the universe "meaningfully." In particular, one can embed general purpose computers. Since these computers are simply particular configurations of states within the lattice of automata, *they can compute over the very set of symbols out of which they are constructed.* Thus, structures—or PTYPES—in this

universe can compute and construct other structures, which also may compute and construct.

For example, here is the simplest known structure that can reproduce itself:[4]

```
  2 2 2 2 2 2 2 2
2 1 7 0 1 4 0 1 4 2
2 0 2 2 2 2 2 2 0 2
2 7 2         2 1 2
2 1 2         2 1 2
2 0 2         2 1 2
2 7 2         2 1 2
2 1 2 2 2 2 2 2 1 2 2 2 2 2
2 0 7 1 0 7 1 0 7 1 1 1 1 1 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Each number is the state of one of the automata in the lattice. Blank space is presumed to be in state "0." The "2"-states form a sheath around the "1"-state data-path. The "7 0" and "4 0" state pairs constitute signals embedded within the data-path. They will propagate counter-clockwise around the loop, cloning off copies which propagate down the extended tail as they pass the T-junction between loop and tail. When the signals reach the end of the tail, they have the following effects: each "7 0" signal extends the tail by one unit, and the two "4 0" signals construct a left-hand corner at the end of the tail. Thus, for each full cycle of the instructions around the loop, another side and corner of an "offspring loop" will be constructed. When the tail finally runs into itself after four cycles, the collision of signals results in the disconnection of the two loops as well as the construction of a tail on each of the loops.

After 151 time steps, this system will evolve to the following configuration:

```
                2
              2 1 2
              2 7 2
              2 0 2
              2 1 2
  2 2 2 2 2 2 2 7 2     2 2 2 2 2 2 2 2
2 1 1 1 7 0 1 7 0 2   2 1 7 0 1 4 0 1 4 2
2 1 2 2 2 2 2 2 1 2   2 0 2 2 2 2 2 2 0 2
2 1 2         2 7 2   2 7 2         2 1 2
2 1 2         2 0 2   2 1 2         2 1 2
2 4 2         2 1 2   2 0 2         2 1 2
2 1 2         2 7 2   2 7 2         2 1 2
2 0 2 2 2 2 2 2 2 0 2   2 1 2 2 2 2 2 2 1 2 2 2 2 2
2 4 1 0 7 1 0 7 1 2   2 0 7 1 0 7 1 0 7 1 1 1 1 1 2
  2 2 2 2 2 2 2 2 2     2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Thus, the initial configuration has succeeded in reproducing itself.

Each of these loops will go on to reproduce itself in a similar manner, giving rise to an expanding *colony* of loops, growing out into the array. Color plates 1

[4]Note added in proof: this structure has been simplified by John Byl in a report to appear in *Physica D.*

through 8 show the development of a colony of loops from a single initial loop (for details, see Langton[27,28]).

These embedded self-reproducing loops are the result of the recursive application of a rule to a seed structure. In this case, the primary rule that is being recursively applied constitutes the "physics" of the universe. The initial state of the loop itself constitutes a little "computer" under the recursively applied physics of the universe: a computer whose program causes it to construct a copy of itself. The "program" within the loop computer is also applied recursively to the growing structure. Thus, this system really involves a double level of recursively applied rules. The mechanics of applying one recursive rule within a universe whose physics is governed by another recursive rule had to be worked out by trial and error. This system makes use of the signal propagation capacity to embed a structure that itself *computes* the resulting structure, rather than the "physics" being directly responsible for developing the final structure from a passive seed.

This captures the flavor of what goes on in natural development: the genotype codes for the constituents of a dynamic process in the cell, and it is this dynamic process that is primarily responsible for mediating—or "computing"—the expression of the genotype in the course of development.

EXAMPLE 3: COMPUTER ANIMATION

The previous examples were largely concerned with the growth and development of *structural* PTYPES. Here, we give an example of the development of a *behavioral* PTYPE.

Craig Reynolds has implemented a simulation of flocking behavior.[35] In this model—which is meant to be a general platform for studying the qualitatively similar phenomena of flocking, herding, and schooling—one has a large collection of autonomous but interacting objects (which Reynolds refers to as "Boids"), inhabiting a common simulated environment.

The modeler can specify the manner in which the individual Boids will respond to *local* events or conditions. The global behavior of the aggregate of Boids is strictly an emergent phenomenon, none of the rules for the individual Boids depend on global information, and the only updating of the global state is done on the basis of individual Boids responding to local conditions.

Each Boid in the aggregate shares the same behavioral "tendencies":

1. to maintain a minimum distance from other objects in the environment, including other Boids,
2. to match velocities with Boids in its neighborhood, and
3. to move toward the perceived center of mass of the Boids in its neighborhood.

These are the only rules governing the behavior of the aggregate.

These rules, then, constitute the generalized genotype (GTYPE) of the Boids system. They say nothing about structure, or growth and development, but they determine the behavior of a set of interacting objects, out of which very natural motion emerges.

With the right settings for the parameters of the system, a collection of Boids released at random positions within a volume will collect into a dynamic flock, which flies around environmental obstacles in a very fluid and natural manner, occasionally breaking up into sub-flocks as the flock flows around both sides of an obstacle. Once broken up into sub-flocks, the sub-flocks reorganize around their own, now distinct and isolated, centers of mass, only to re-merge into a single flock again when both sub-flocks emerge at the far-side of the obstacle and each sub-flock feels anew the "mass" of the other sub-flock (Figure 8).

The flocking behavior itself constitutes the generalized-phenotype (PTYPE) of the Boids system. It bears the same relation to the GTYPE as an organism's morphological phenotype bears to its molecular genotype. The same distinction between the *specification* of machinery and the *behavior* of machinery is evident.

DISCUSSION OF EXAMPLES

In all of the above examples, the recursive rules apply to *local structures* only, and the PTYPE—structural or behavioral—that results at the global level emerges from out of all of the local activity taken collectively. Nowhere in the system are there rules for the behavior of the system at the global level. This is a much more powerful and simple approach to the generation of complex behavior than that typically taken in AI, for instance, where "expert systems" attempt to provide global rules for global behavior. Recursive, "bottom up" specifications yield much more natural, fluid, and flexible behavior at the global level than typical "top-down" specifications, and they do so *much* more parsimoniously.

It is worthwhile to note that *context-sensitive* rules in GTYPE/PTYPE systems provide the possibility for nonlinear interactions among the parts. Without context sensitivity, the systems would be linearly decomposable, information could not "flow" throughout the system in any meaningful manner, and complex long-range dependencies between remote parts of the structures could not develop.

There is also a very important feedback mechanism *between* levels in such systems: the interactions among the low-level entities give rise to the global level dynamics which, in turn, affects the lower levels by *setting the local context* within which each entity's rules are invoked. Thus, local behavior supports global dynamics, which shapes local context, which affects local behavior, which supports global dynamics, and so forth.

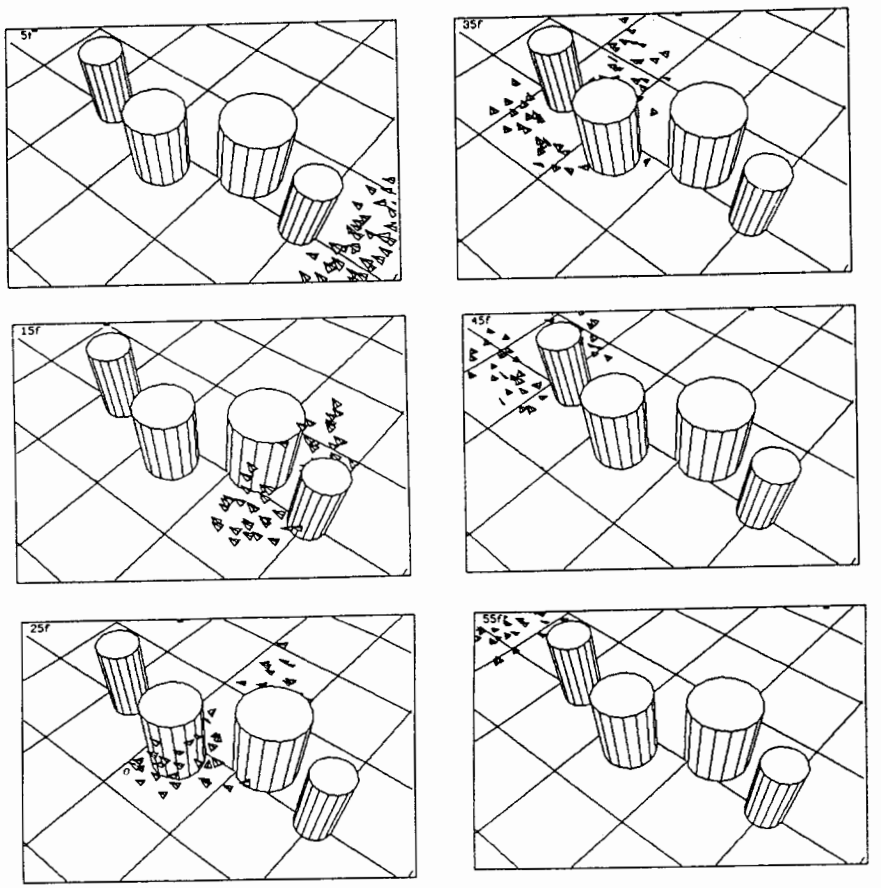


FIGURE 8 A flock of "Boids" negotiating a field of columns. Sequence generated by Craig Reynolds.

GENUINE LIFE IN ARTIFICIAL SYSTEMS It is important to distinguish the ontological status of the various levels of behavior in such systems. At the level of the individual behav*ors*, we have a clear difference in kind: boids are *not* birds; they are not even remotely like birds; they have no cohesive physical structure, but rather exist as information structures—processes—within a computer. But—and this is *the* critical "but"—at the level of behav*iors*, *flocking Boids and flocking birds are two instances of the same phenomenon:* flocking.

The behavior of a flock as a whole does not depend on the internal details of the entities of which it is constituted—only on the details of the way in which these entities behave in each other's presence. Thus, flocking in Boids is true flocking, and may be counted as another empirical data point in the study of flocking behavior in general, right up there with flocks of geese and flocks of starlings.

This is *not* to say that flocking Boids capture *all* the nuances upon which flocking behavior depends, or that the Boid's behavioral repertoire is sufficient to exhibit all the different modes of flocking that have been observed—such as the classic "V" formation of flocking geese. The crucial point is that we have captured—within an aggregate of artificial entities—a *bona-fide* lifelike behavior, and that the behavior emerges within the artificial system in the same way that it emerges in the natural system.

The same is true for L-systems and the self-reproducing loops. The constituent parts of the artificial systems are different kinds of things from their natural counterparts, but the emergent behavior that they support is the same kind of thing: genuine morphogenesis and differentiation for L-systems, and genuine self-reproduction in the case of the loops.

The claim is the following: The "artificial" in Artificial Life refers to the component parts, not the emergent processes. If the component parts are implemented correctly, the processes they support are *genuine*—every bit as genuine as the natural processes they imitate.

The *big* claim is that a properly organized set of artificial primitives carrying out the same functional roles as the biomolecules in natural living systems will support a process that will be "alive" in the same way that natural organisms are alive. Artificial Life will therefore be *genuine* life—it will simply be made of different stuff than the life that has evolved here on Earth.

EVOLUTION

In the preceding sections, we have mentioned several times the formal impossibility of predicting the behavior of an arbitrary machine by mere inspection of its specification and initial state. We must run the machine in order to determine its behavior in the general case.

The consequence of this unpredictability for GTYPE/PTYPE systems is that we cannot determine the PTYPE that will be produced by an arbitrary GTYPE by inspection alone. We must "run" the GTYPE, and let the PTYPE develop in order to determine the resulting structure and its behavior.

Since, for any interesting system, there will exist an enormous number of potential GTYPES, and since there is no formal method for deducing the PTYPE from the GTYPE, how do we go about finding GTYPES that will generate lifelike PTYPES?

Up till now, the process has largely been one of guessing at appropriate GTYPES, and modifying them by trial and error until they generate the appropriate PTYPES. However, this process is limited by our preconceptions of what the appropriate PTYPES would be, and by our restricted notions of how to generate GTYPES. We need to automate the process so that our preconceptions and limited ability to conceive of machinery do not overly constrain the search for GTYPES that will yield the appropriate behaviors.

NATURAL SELECTION AMONG POPULATIONS OF MACHINES

Nature, of course, has hit upon the proper mechanism: *evolution by the process of natural selection among variants.* The scheme is a very simple one. However, in the face of the formal impossibility of predicting behavior from machine description alone, it may well be the only efficient, general scheme for searching the space of possible GTYPES.

The mechanism of evolution is as follows. A set of GTYPES is interpreted within a specific environment, forming a population of PTYPES which interact with one another and with features of the environment in various complex ways. On the basis of the relative performance of their associated PTYPES, *some* of the GTYPES are reproduced in such a way that the copies are similar to—but not exactly the same as—the originals. These new GTYPES develop PTYPES which enter into the complex interactions within the environment, and the process is continued *ad infinitum* (Figure 9). As expected from the formal limitations on predictability, GTYPES must be "run" in an environment and their behaviors must be evaluated explicitly, their implicit behavior cannot be determined in any other way.

Evolution, therefore, works by selecting *descriptions* of machines which exhibit the appropriate behaviors when they are run, and it progresses by creating new descriptions from those existing descriptions which produced machinery with the most appropriate behaviors.

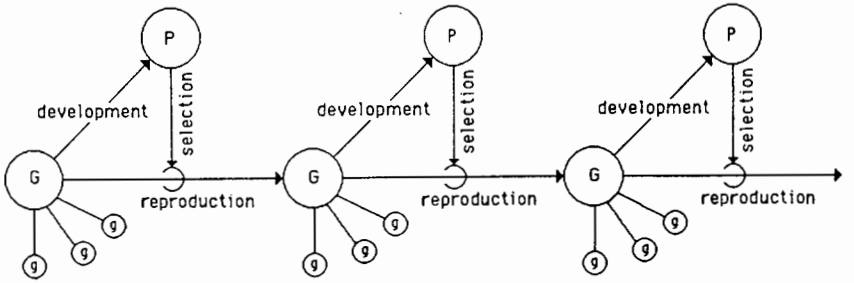


FIGURE 9 The process of evolution by natural selection.

CRITERIA FOR EVOLUTION Evolution by the process of natural selection will operate within a population of reproducing machines provided that the following three criteria are met:

- CRITERION OF HEREDITY—Offspring are similar to their parents: the copying process maintains high fidelity.
- CRITERION OF VARIABILITY—Offspring are not *exactly* like their parents or each other: the copying process is not perfect.
- CRITERION OF FECUNDITY—Variants leave different numbers of offspring: specific variations have an effect on behavior, and behavior has an effect on reproductive success.

Of these three criteria, the first two apply primarily to the process by which GTYPES are copied and modified, and the third applies to the manner in which PTYPES determine which GTYPES are selected for copying.

GENETIC ALGORITHMS

John Holland has pioneered the application of the process of natural selection to the problem of machine learning in the form of what he calls the "genetic algorithm" (GA).[4,21,22] The GA is a specific method for generating a set of offspring from a parent population, and is primarily concerned with producing variants having a high probability of success in the environment. The GA generates variants by applying *genetic operators* to the GTYPES of the most successful PTYPES in the population. The genetic operators consist of (in relative order of importance) *crossover, inversion,* and *mutation.*

The basic outline of the genetic algorithm is as follows:

1. Select pairs of GTYPES according to the success of their respective PTYPES. The more successful the PTYPE, the more likely that its GTYPE is selected.
2. Apply *genetic operators* to the pairs of GTYPES selected to create "offspring" GTYPES.
3. Replace the least successful GTYPES with the offspring generated in step 2.

Despite the seeming simplicity of the GA, Holland has been able to prove several remarkable theorems about its performance. GA's, it turns out, are capable of making optimal use of the past experience of the population, as stored in the distribution of "alleles" in the GTYPE pool and in the relative success of the PTYPES associated with the GTYPES in the population.

Based on the number of positions on which they may vary, there are a great many GTYPES that could potentially be constructed. In a system of any complexity, the number of potential GTYPES is astronomical. If $\mathcal{L}$ is the number of positions at which two GTYPES might exhibit differences, and $\mathcal{N}$ is the average number of values one might find at each such position, then the size of GTYPE space is of order $\mathcal{N}^{\mathcal{L}}$.

There is an even larger number of potential PTYPES, since each GTYPE could determine different PTYPES in different environmental contexts. It is important to note that a major part of this environmental context is the population of other PTYPES. Thus, just as the rest of the GTYPE provides an important part of the context within which a particular part of the GTYPE is interpreted, the rest of the PTYPES in the population provide an important part of the context within which a particular PTYPE develops.

What the GA does—and does very well—is to explore this very large space of possible PTYPES in an intelligent manner. It does so by hunting out the GTYPE building blocks most often associated with the most successful PTYPES, and biasing the sampling of GTYPE space in favor of offspring which use these highly rated building blocks in new combinations.

The *crossover* operator is responsible for most of the "intelligence" in the operation of the GA. Given two strings which represent GTYPES, the crossover operator works by swapping segments of the strings from each to the other, as illustrated in Figure 10. The reason this is so effective an operator is that it tends to maintain combinations of building blocks that have worked well together in the past, because it swaps whole groups of building blocks at a time.

Thus, the crossover operator works by producing *new combinations* of building blocks, the inversion operator works by permuting the *linkage* relation between building blocks, and the mutation operator works by introducing *new* building blocks. Taken together, these three operators constitute an extremely general and powerful mechanism for searching large and unpredictable description spaces, one which is highly immune to getting hung up on local maxima, because it is "climbing" many local gradients in parallel and quite often produces new sample points that fall between local maxima.

The set of all possible subsets of building blocks for constructing GTYPES is referred to as *schema space*. A schema is a particular subset of the set of building blocks that might occur in a particular GTYPE. For instance, the set consisting of the specific values at sites 2, 3, and 10 of a GTYPE is a schema. A whole GTYPE, the specific value at every position considered, is another schema, as is the set consisting of just a specific value at position 22. The set of all possible subsets of a set is formally referred to as the *power set* of the set. Thus, schema space is formally identical to the power set of GTYPES: the space of possible GTYPE building blocks.

Holland has been able to prove that, under the action of the genetic algorithm, every schema represented in the population—that is, every represented element of the power set—will propagate throughout the population in direct proportion to its own intrinsic fitness. Furthermore, this is achieved without explicitly collecting information on the fitness of each represented building block. Since each GTYPE is really an instance of $2^{\mathcal{L}}$ distinct schemas, by physically testing a population of only M GTYPES, the GA is actually gaining information on between $2^{\mathcal{L}}$ and $M2^{\mathcal{L}}$ schemas.

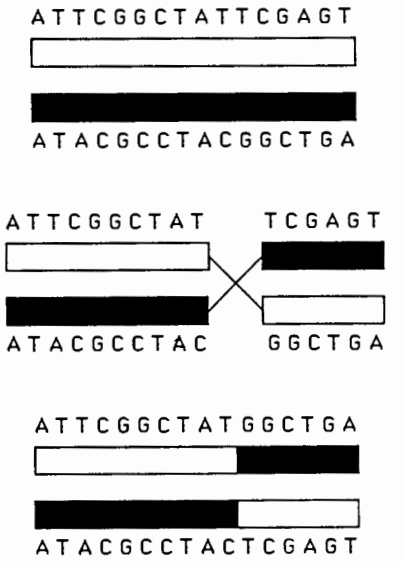


FIGURE 10 Action of the crossover operator.

This "implicit parallelism" yields robust evolutionary potential. As Holland puts it[4]:

> [The GA] samples each schema with above-average instances with increasing intensity, thereby further confirming (or disconfirming) its usefulness and exploiting it (if it remains above-average). This also drives the overall average [fitness of the population] upward, providing an ever-increasing criterion that a schema must meet to be above average. Moreover, the heuristic employs a distribution of instances, rather than working only from the "most recent best" instance. This yields both robustness and insurance against being caught on "false peaks" (local optima) that misdirect development. Overall, the power of this heuristic stems from its rapid accumulation of better-than-average building blocks.

EMERGENT FITNESS FUNCTIONS

A problem common to many computer models employing evolutionary processes is that it is very easy to underestimate the complexity of environmental interactions. Most such models provide overly simple environments within which certain behaviors are preordained as "fit" and others as "unfit." Such models often contain

clear-cut boundaries between the environments and the "living" systems they nurture, and environments are often specified top-down, even when the primary actors in the model are specified bottom-up.

In nature, it is often extremely difficult to draw such sharp distinctions between the living system and its environment, and interactions with and within the environment are often as complicated as interactions within the living-system. Rigid, pre-specified, "unnatural" environments foster rigid, predictable, "unlifelike" evolutionary progression.

Rather, the environment itself should be specified at the lowest possible level, in a bottom-up fashion. The "artificial nature" within which the artificial life-forms of a model are to evolve must be only implicit in the rules of the system, allowing for much more subtle interactions between the life-forms and features of the environment. Such systems have much greater potential for demonstrating genuine evolutionary progression. The fitness function, the set of criteria that determines whether an organism is "fit" in its environment, should itself be an emergent property of the system (see the article by Packard in these proceedings).

THE ROLE OF COMPUTERS IN STUDYING LIFE AND OTHER COMPLEX SYSTEMS

Artificial Intelligence and Artificial Life are each concerned with the application of computers to the study of complex, natural phenomena. Both are concerned with generating complex behavior. However, the manner in which each field employs the technology of computation in the pursuit of its respective goals is strikingly different.

AI has based its underlying methodology for generating intelligent behavior on the computational paradigm. That is, AI uses the technology of computation as a model of intelligence. AL, on the other hand, is attempting to develop a new computational paradigm based on the natural processes that support living organisms. That is, AL uses the technology of computation as a tool to explore the dynamics of interacting information structures. It has not adopted the computational paradigm as its underlying methodology of behavior generation, nor does it attempt to "explain" life as a kind of computer program.

One way to pursue the study of Artificial Life would be to attempt to create life *in vitro*, using the same kinds of organic chemicals out of which we are constituted. Indeed, there are numerous exciting efforts in this direction. This would certainly teach us a lot about the possibilities for alternative life-forms *within* the carbon-chain chemistry domain that could have (but didn't) evolve here.

However, biomolecules are extremely small and difficult to work with, requiring rooms full of special equipment, replete with dozens of "postdocs" and graduate students willing to devote the larger part of their professional careers to the perfection of electrophoretic gel techniques. Besides, although the creation of life *in vitro* would certainly be a scientific feat worthy of note—and probably even a Nobel prize—it would not, in the long run, tell us much more about the space of *possible* life than we already know.

Computers provide an alternative medium within which to attempt to synthesize life. Modern computer technology has resulted in machinery with tremendous potential for the creation of life *in silico*.

Computers should be thought of as an important laboratory tool for the study of life, substituting for the array of incubators, culture dishes, microscopes, electrophoretic gels, pipettes, centrifuges and other assorted wet-lab paraphernalia, one simple-to-master piece of experimental equipment devoted exclusively to the incubation of information structures.

The advantage of working with information structures is that information has no intrinsic size. The computer is *the* tool for the manipulation of information, whether that manipulation is a consequence of our actions or a consequence of the actions of the information structures themselves. Computers themselves will not be alive, rather they will support informational universes within which dynamic populations of informational "molecules" engage in informational "biochemistry."

This view of computers as workstations for performing scientific experiments within artificial universes is fairly new, but it is rapidly becoming accepted as a legitimate—even necessary—way of pursuing science. In the days before computers, scientists worked primarily with systems whose defining equations could be solved analytically, and ignored those whose defining equations could *not* be so solved. This was largely the case because, in the absence of analytic solutions, the equations would have to be integrated over and over again—essentially simulating the time behavior of the system. Without computers to handle the mundane details of these calculations, such an undertaking was unthinkable except in the simplest cases.

However, with the advent of computers, the necessary mundane calculations could be relegated to these idiot-savants, and the realm of numerical simulation was opened up for exploration. "Exploration" is an appropriate term for the process, because the numerical simulation of systems allows one to "explore" the system's behavior under a wide range of parameter settings and initial conditions. The heuristic value of this kind of experimentation cannot be overestimated. One often gains tremendous insight into the essential dynamics of a system by observing its behavior under a wide range of initial conditions.

Most importantly, however, computers are beginning to provide scientists with a new paradigm for modeling the world. When dealing with essentially unsolvable governing equations, the primary reason for producing a formal mathematical model—the hope of reaching an analytic solution by symbolic manipulation—is lost. Systems of ordinary and partial differential equations are not very well suited for implementation as computer algorithms. One might expect that other modeling technologies would be more appropriate when the goal is the *synthesis*, rather than the *analysis*, of behavior (see Toffoli[45] for a good exposition).

This expectation is easily borne out. With the precipitous drop in the cost of raw computing power, computers are now available that are capable of simulating physical systems from first principles. This means that it has become possible,

for example, to model turbulent flow in a fluid by simulating the motions of its constituent particles—not just approximating *changes* in concentrations of particles at particular points, but actually computing their motions exactly.[18,46,52]

What does all of this have to do with the study of life? The most surprising lesson we have learned from simulating complex physical systems on computers is that *complex behavior need not have complex roots.* Indeed, tremendously interesting and beguilingly complex behavior can emerge from collections of *extremely* simple components.

This leads directly to the exciting possibility that much of the complex behavior exhibited by nature—especially the complex behavior that we call life—*also* has simple generators. Since it is very hard to work backwards from a complex behavior to its generator, but very simple to create generators and synthesize complex behavior, a promising approach to the study of complex natural systems is to undertake the general study of the kinds of behavior that can emerge from distributed systems consisting of simple components (Figure 11).

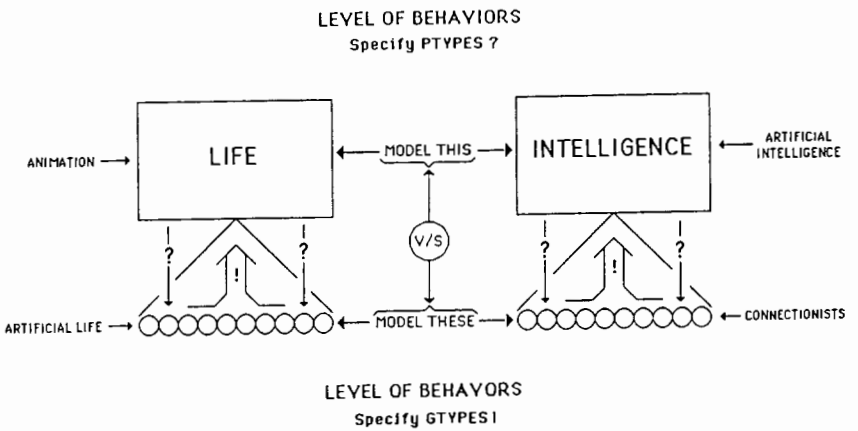


FIGURE 11 The bottom-up *versus* the top-down approach to modeling complex systems.

NONLINEARITY AND LOCAL DETERMINATION OF BEHAVIOR

LINEAR VS. NONLINEAR SYSTEMS

The distinction between linear and nonlinear systems is fundamental, and provides excellent insight into why the mechanisms of life should be so hard to find. The simplest way to state the distinction is to say that *linear systems* are those for which the behavior of the whole is just the sum of the behavior of its parts, while for *nonlinear systems*, the behavior of the whole is *more* than the sum of its parts.

Linear systems are those which obey the *superposition principle.* We can break up complicated linear systems into simpler constituent parts, and analyze these parts *independently.* Once we have reached an understanding of the parts in isolation, we can achieve a full understanding of the whole system by *composing* our understanding of the isolated parts. This is the key feature of linear systems: by studying the parts in isolation, we can learn everything we need to know about the complete system.

This is not possible for nonlinear systems, which do *not* obey the superposition principle. Even if we could break such systems up into simpler constituent parts, and even if we could reach a complete understanding of the parts in isolation, we would not be able to combine our understandings of the individual parts into an understanding of the whole system. The key feature of nonlinear systems is that their primary behaviors of interest are properties of the *interactions between parts*, rather than being properties of the parts themselves, and these interaction-based properties necessarily disappear when the parts are studied independently.

Thus, analysis is most fruitfully applied to linear systems. Such systems can be taken apart in meaningful ways, the resulting pieces solved, and the solutions obtained from solving the pieces can be put back together in such a way that one has a solution for the whole system.

Analysis has *not* proved anywhere near as effective when applied to nonlinear systems: the nonlinear system must be treated as a whole.

A different approach to the study of nonlinear systems involves the inverse of analysis: *synthesis.* Rather than start with the behavior of interest and attempting to analyze it into its constituent parts, we start with constituent parts and put them together in the attempt to *synthesize* the behavior of interest.

Life is a property of *form*, not *matter*, a result of the organization of matter rather than something that inheres in the matter itself. Neither nucleotides nor amino acids nor any other carbon-chain molecule is alive—yet put them together in the right way, and the dynamic behavior that emerges out of their interactions is what we call life. It is effects, not things, upon which life is based—life is a kind of behavior, not a kind of stuff—and as such, it is constituted of simpler behaviors, not simpler stuff. *Behaviors themselves* can constitute the fundamental parts of nonlinear systems—*virtual parts*, which depend on nonlinear interactions between physical parts for their very existence. Isolate the physical parts and the virtual parts cease to exist.[29] It is the *virtual parts* of living systems that Artificial Life is after: the fundamental atoms and molecules of behavior.

THE PARSIMONY OF LOCAL DETERMINATION OF BEHAVIOR

It is easier to generate complex behavior from the application of simple, *local* rules than it is to generate complex behavior from the application of complex, *global* rules. This is because complex global behavior is usually due to nonlinear interactions occurring at the local level. With bottom-up specifications, the system computes the local, nonlinear interactions explicitly and the global behavior—which was implicit in the local rules—emerges spontaneously without being treated explicitly.

With top-down specifications, however, local behavior must be implicit in global rules. This is really putting the cart before the horse! The global rules must "predict" the effects on global structure of many local, nonlinear interactions— something which we have seen is intractable, even impossible, in the general case. Thus, top-down systems must take computational shortcuts and explicitly deal with special cases, which results in inflexible, brittle, and unnatural behavior.

Furthermore, in a system of any complexity the number of possible global states is astronomically enormous, and grows exponentially with the size of the system. Systems that attempt to supply *global* rules for *global* behavior simply *cannot* provide a different rule for every global state. Thus, the global states must be classified in some manner; categorized using a coarse-grained scheme according to which the global states within a category are indistinguishable. The rules of the system can only be applied at the level of resolution of these categories. There are many possible ways to implement a classification scheme, most of which will yield different partitionings of the global state-space. Any rule based system must necessarily *assume* that finer-grained differences don't matter, or must include a finite set of tests for "special cases," and then must assume that no *other* special cases are relevant.

For most complex systems, however, fine differences in global state can result in enormous differences in global behavior, and there may be no way in principle to partition the space of global states in such a way that specific fine differences have the appropriate global impact.

On the other hand, systems that supply *local* rules for *local* behaviors, *can* provide a different rule for each and every possible local state. Furthermore, the size of the local state-space can be completely independent of the size of the system. In local rule-governed systems, each local state, and consequently the global state, can be determined exactly and precisely. Fine differences in global state will result in very specific differences in local state, and consequently will affect the invocation of local rules. As fine differences affect local behavior, the difference will be felt in an expanding patch of local states, and in this manner—propagating from local neighborhood to local neighborhood—fine differences in global state can result in large differences in global behavior. The only "special cases" explicitly dealt with in locally determined systems are exactly the set of all possible local states, and the rules for these are just exactly the set of all local rules governing the system.

CONCLUSION: THE EVOLUTION OF WATCHMAKERS

As complex biochemical machines, living organisms have been compared to fine mechanical watches. In the famous "Argument from Design" this analogy has been used as proof of the existence of God—the "Watchmaker" whose existence we must infer from the evident "design" exhibited by these fine biochemical clockworks. The most famous formulation of this argument was put forth by William Paley in the first years of the nineteenth century (see Richard Dawkins' excellent exposition of this argument,[10] as well as his contribution to these proceedings).

By the middle of the nineteenth century, Darwin had given a better explanation for the existence of design in nature. During the three-and-one-half billion years from the pre-biotic soup to the present, the biochemical springs, gears, and balance-wheels of living organisms have been slowly crafted and fitted together by a "Blind Watchmaker": the process of evolution by natural selection.

However, this first great era of evolution is drawing to a close and another one is beginning. The process of evolution has lead—in us—to "watches" which understand what makes them "tick," which are beginning to tinker around with their own mechanisms, and which will soon have mastered the "clockwork" technology necessary to construct watches of their own design. The Blind Watchmaker has produced *seeing watches*, and these "watches" have seen enough to become watchmakers themselves. Their vision, however, is extremely limited, so much so that perhaps they should be referred to as *near-sighted watchmakers*.

With the discovery of the structure of DNA and the interpretation of the genetic code, a feedback loop stretching from molecules to men and back again has finally closed. The process of biological evolution has yielded genotypes that code for phenotypes capable of manipulating their own genotypes directly: copying them, altering them, or creating new ones altogether in the case of Artificial Life.

By the middle of this century, mankind had acquired the power to extinguish life on Earth. By the middle of the next century, he will be able to create it. Of the two, it is hard to say which places the larger burden of responsibility on our shoulders. Not only the specific kinds of living things that will exist, but the very course of evolution itself will come more and more under our control. The future effects of changes we make now are, in principle, unpredictable—we cannot foresee all of the possible consequences of the kinds of manipulations we are now capable of inflicting on the very fabric of inheritance, whether in natural or artificial systems. Yet if we make changes, we are responsible for the consequences.

How can we justify our manipulations? How can we take it upon ourselves to create life, even within the artificial domain of computers, and then snuff it out again by halting the program or pulling the plug? What right to existence does a physical process acquire when it is a "living process," whatever the medium in which it occurs? Why should these rights accrue only to processes with a particular material constitution and not another? Whether these issues have correct answers or not, they must be addressed, honestly and openly.

Artificial Life is more than just a scientific or technical challenge, it is also a challenge to our most fundamental social, moral, philosophical, and religious beliefs. Like the Copernican model of the solar system, it will force us to re-examine our place in the universe and our role in nature.

ACKNOWLEDGMENTS

I am grateful for discussions with Richard Bagley, Richard K. Belew, Arthur Burks, Peter Cariani, A. K. Dewdney, Doyne Farmer, Stephanie Forrest, Ron Fox, John Holland, Greg Huber, Dan Kaiser, Stuart Kauffman, Richard Laing, David Langton, Norman Packard, Steen Rasmussen, Craig Reynolds, Paul Scott, and, of course, the participants of the Artificial Life workshop.

REFERENCES

1. American Heritage Dictionary of the English Language.
2. Arbib, M. A. (1966), "Simple Self-Reproducing Universal Automata," *Information and Control* **9**, 177–189.
3. Berlekamp, E., J. Conway, and R. Guy (1982), *Winning Ways for your Mathematical Plays* (New York: Academic Press).
4. Booker, L., D. E. Goldberg, and J. H. Holland (1988), "Classifier Systems and Genetic Algorithms," *Artificial Intelligence*, in press.
5. Braitenberg, V. (1984), *Vehicles: Experiments in Synthetic Psychology* (Cambridge: MIT Press).
6. Burks, A. W. (ed) (1970), *Essays on Cellular Automata* (Urbana, IL: University of Illinois Press).
7. Chapuis, A., and E. Droz (1958), *Automata: A Historical and Technological Study*, trans. A. Reid (London: B.T. Batsford Ltd.).
8. Codd, E. F. (1968), *Cellular Automata* (New York: Academic Press).
9. Conrad, M., and M. Strizich (1985), "EVOLVE II: A Computer Model of an Evolving Ecosystem," *Biosystems* **17**, 245–258.
10. Dawkins, R. (1986), *The Blind Watchmaker* (London: W. W. Norton).
11. Dewdney, A. K. (1984), "Computer Recreations: In the Game Called Core War Hostile Programs Engage in a Battle of Bits," *Scientific American* **250(5)**, 14–22.
12. Dewdney, A. K. (1984), "Computer Recreations: Sharks and Fish Wage an Ecological War on the Toroidal Planet Wa-Tor," *Scientific American* **251(6)** 14–22.
13. Dewdney, A. K. (1985), "Computer Recreations: A Core War Bestiary of Viruses, Worms and Other Threats to Computer Memories," *Scientific American* **252(3)**, 14–23.
14. Dewdney, A. K. (1985), "Computer Recreations: Exploring the Field of Genetic Algorithms in a Primordial Computer Sea Full of Flibs," *Scientific American* **253(5)**, 21–32.
15. Dewdney, A. K. (1987), "Computer Recreations: A Program Called MICE Nibbles its Way to Victory at the First Core War Tournament," *Scientific American* **256(1)**, 14–20.
16. Dewdney, A. K. (1987), "Computer Recreations: The Game of Life Acquires Some Successors in Three Dimensions," *Scientific American* **256(2)**, 16–24.
17. Farmer, J. D., T. Toffoli, and S. Wolfram (1984), "Cellular Automata: Proceedings of an Interdisciplinary Workshop, Los Alamos, New Mexico, March 7-11, 1983," *Physica D* **10(1-2)**.
18. Frisch, U., B. Hasslacher, and Y. Pomeau (1986), "Lattice Gas Automata for the Navier-Stokes Eequation," *Physical Review Letters* **56**, 1505–1508.
19. Gardner, M. (1970), "The Fantastic Combinations of John Conway's New Solitaire Game 'Life,'" *Scientific American* **223(4)**, 120–123.

20. Gardner, M. (1971), "On Cellular Automata, Self-Reproduction, The Garden of Eden and the Game of 'Life,'" *Scientific American* **224(2)**, 112–117.
21. Holland, J. H. (1975), *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press).
22. Holland, J. H. (1986), "Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-Based Systems.," *Machine Learning II*, Eds. R. S. Mishalski, J. G. Carbonell, and T. M. Mitchell (New York: Kauffman), 593–623.
23. Hopcroft, J. E., and J. D. Ullman (1979), *Introduction to Automata Theory, Languages, and Computation* (Menlo Park, CA: Addison-Wesley).
24. Jacobson, H. J. (1958), "On Models of Reproduction," *American Scientist* **46(3)**, 255–284.
25. Laing, R. (1975), "Artificial Molecular Machines: A Rapproachment between Kinematic and Tessellation Automata," *Proceedings of the International Symposium on Uniformly Structured Automata and Logic, Tokyo, August, 1975.*
26. Laing, R. (1977), "Automaton Models of Reproduction by Self-Inspection," *J. Theor. Biol.* (1977) **66**, 437–456.
27. Langton, C. G. (1984), "Self-Reproduction in Cellular Automata," *Physica D* **10(1-2)**, 135–144.
28. Langton, C. G. (1986), "Studying Artificial Life with Cellular Automata," *Physica D* **22**, 120–149.
29. Langton, C. G. (1987), "Virtual State Machines in Cellular Automata," *Complex Systems* **1**, 257–271.
30. Masani, P. (1985), *Norbert Wiener: Collected Works* (Cambridge, MA: Massachusetts Institute of Technology Press), Vol. IV.
31. McCulloch, W. S., and W. Pitts (1943), "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics* **5**, 115–133.
32. Minsky, M., and S. Papert (1969), *Perceptrons: An Introduction to Computational Geometry* (Cambridge, MA: MIT Press).
33. Penrose, L. S. (1959), "Self-Reproducing Machines," *Scientific American* **200(6)**, 105–113.
34. Poundstone, W. (1985), *The Recursive Universe* (New York: William Morrow).
35. Reynolds, C. W. (1987), "Flocks, Herds, and Schools: A Distributed Behavioral Model (Proceedings of SIGGRAPH '87)," *Computer Graphics* **21(4)**, 25–34.
36. Rizki, M. M., and M. Conrad (1986), "Computing the Theory of Evolution," *Physica D* **22**, 83–99.
37. Rosenblatt, F. (1962), *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* (Washington, D.C.: Spartan Books).
38. Samuel, A.L. (1959), "Some Studies in Machine Learning using the Game of Checkers," *IBM J. Res. Dev.* **3**, 210–229.
39. Simon, Herbert A. (1969), *The Sciences of the Artificial* (Boston: MIT Press).

40. Stahl, W. R. and Goheen, H. E. (1963), "Molecular Algorithms," *J. Theoret. Biol.* **5**, 266–287.
41. Stahl, W. R., R. W. Coffin, and H. E. Goheen (1964), "Simulation of Biological Cells by Systems Composed of String-Processing Finite Automata," *AFIPS Conference Proceedings - 1964 Spring Joint Computer Conference*, vol. 25, 89–102.
42. Stahl, W. R. (1965), "Algorithmically Unsolvable Problems for a Cell Automaton," *J. Theoret. Biol.* **8**, 371–394.
43. Stahl, W. R. (1967), "A Computer Model of Cellular Self-Reproduction," *J. Theoret. Biol.* **14**, 187–205.
44. Thatcher, J. (1970), "Universality in the von Neumann Cellular Model," *Essays on Cellular Automata*, Ed. A. W. Burks (Urbana, IL: University of Illinois Press).
45. Toffoli, T. (1984), "Cellular Automata as an Alternative to (Rather than an Approximation of) Differential Equations in Modeling Physics," *Physica D* **10(1-2)**.
46. Toffoli, T., and N. Margolus (1987), *Cellular Automata Machines.* (Cambridge: MIT Press).
47. Ulam, S. (1962), "On some Mathematical Problems Connected with Patterns of Growth of Figures," *Proceedings of Symposia in Applied Mathematics* **14**, 215–224; reprinted in *Essays on Cellular Automata*, Ed. A. W. Burks (Urbana, IL: University of Illinois Press).
48. Von Neumann, J. (1966), *Theory of Self-Reproducing Automata*, edited and completed by A.W. Burks (Urbana, IL: U. of Illinois Press).
49. Walter, W. G. (1950), "An Imitation of Life," *Scientific American* **182(5)**, 42–45.
50. Walter, W. G. (1951), "A Machine That Learns," *Scientific American* **51**, 60–63.
51. Weiner, N. (1961), *Cybernetics, or Control and Communication in the Animal and the Machine* (New York:, John Wiley); original print in 1948.
52. Wolfram, S. (1986), "Cellular Automaton Fluids 1: Basic Theory," *Journal of Statistical Physics* **45**, 471–526.