

ARTIFICIAL LIFE

CHRISTOPHER G. LANGTON

"Art" + "Life" = Artificial Life: Life made by Man rather than by Nature. Our technological capabilities have brought us to the point where we are on the verge of creating "living" artifacts. The field of Artificial Life is devoted to studying the scientific, technological, artistic, philosophical, and social implications of such an accomplishment.

1 THE BIOLOGY OF POSSIBLE LIFE

Biology is the scientific study of life — in principle anyway. In practice, biology is the scientific study of life on Earth based on carbon-chain chemistry. There is nothing in its charter that restricts biology to carbon-based life; it is simply that this is the only kind of life that has been available to study. Thus, theoretical biology has long faced the fundamental obstacle that it is impossible to derive general principles from single examples.

Without other examples, it is difficult to distinguish essential properties of life-properties that would be shared by any living system — from properties that may be incidental to life in principle, but which happen to be universal to life on Earth due solely to a combination of local historical accident and common genetic descent. In order to derive general theories about life, we need an ensemble of instances to generalize over. Since it is quite unlikely that alien life forms will present themselves to us for study in the near future, our only option is to try to create alternative life-forms ourselves — Artificial Life — literally "life made by Man rather than by Nature".

Artificial Life ("AL" or "A-Life") is the name given to a new discipline that studies "natural" life by attempting to recreate biological phenomena from scratch within computers and other "artificial" media. A-life complements the analytic approach of traditional biology with a synthetic approach: rather than studying biological phenomena by taking living organisms apart to see how they work, we attempt to put together systems that behave like living organisms.

The process of synthesis has been an extremely important tool in many disciplines. Synthetic chemistry — the ability to put together new chemical compounds not found in nature — has not only contributed enormously to our theoretical understanding of chemical phenomena, but has also allowed us to fabricate new materials and chemicals that are of great practical use for industry and technology. Artificial life amounts to the practice of "synthetic biology," and, by analogy with synthetic chemistry, the attempt to recreate biological phenomena in alternative media will result in not only better theoretical understanding of the phenomena under study, but also in practical applications of biological principles in industry and technology. By extending the horizons of empirical research in biology beyond the territory currently circumscribed by life-as-we-know-it, the study of Artificial Life gives us access to the domain of life-as-it-could-be, and it is within this vastly larger domain that we must ground general theories of biology and in which we will discover novel and practical applications of biology in our engineering endeavors.

1.1 AI AND THE BEHAVIOR GENERATION PROBLEM

Artificial Life is concerned with generating life-like 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 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 known. There are still many holes in our knowledge, but the general picture is in place. Therefore, Artificial Life can start by recapturing natural life, and has no need to resort to the sort of initial infidelity that is now coming back to haunt AI. Furthermore, Artificial Life is not primarily concerned with building systems that reach some sort of solution. For AL systems, the ongoing dynamic is the behavior of interest, not necessarily the state ultimately reached by that dynamic.

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.

2 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. It is not surprising, therefore, that models of life have also reflected the principal technology of the era, from pneumatics in the time of the Greeks to computation in the current "information age".

2.1 EARLY HISTORY

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

BC 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 AD, Hero of Alexandria produced a treatise on Pneumatics, which described, among other things, various simple 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 AD, 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. 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.

Perhaps the earliest clockwork simulations of life were the so-called "Jacks", mechanical "men" incorporated in early clocks who 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 armor." These accessory figures retained their popularity even after the spread of clock dials and hands — to the extent that clocks were eventually developed in which the function of time-keeping was secondary to the control of large numbers of figures engaged in various activities, to the point of 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 life-like motion to a mechanical figure or animal. These mechanic automation 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". Vaucanson's goal is captured neatly in the following description.¹

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 construction of this duck. 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 in later years 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.

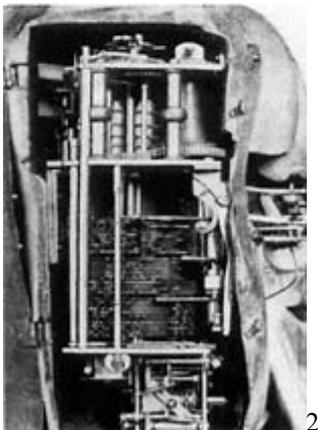


Figure 1 shows two views of one of the ducks — there is some controversy as to whether it is Vaucanson's or Reichsteiner's. The mechanism inside the duck would have been completely covered with feathers and the controlling mechanism in the box below would have been covered up as well.

2.2 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.



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.

2.3 ABSTRACTION OF THE LOGICAL "FORM" OF MACHINES

During the early part of the 20 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. 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. All 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.

2.4 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.

2.5 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 that 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. In particular, Turing showed that the best that such a proposed machine could do would be to emulate the given machine to see whether or not it halted. If the emulated machine halted, fine. However, the emulated machine might run forever before halting, and therefore the emulating machine could not answer whether or not it would halt. Rice and others have extended this indecisive result to the determination by inspection alone — of any non-trivial property of the future behavior of an arbitrary machine.²

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 that will cause a computer to exhibit that behavior. We can automate what we can explain how to do, but there is much that we can not explain how to do. Thus, although a formal specification for a machine that 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 cannot derive behaviors from specifications nor can we 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.

2.6 JOHN VON NEUMANN: FROM MECHANICS TO LOGIC

With the development of the general purpose computer, various researchers turned their attention from the mechanics of life to the logic of life.

The first computational approach to the generation of life-like 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:³

"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.

This approach is the first to capture the essence of the Artificial Life approach. To understand the field of Artificial Life, one need only replace references to "self-reproduction" in the above with references to any other biological phenomenon.

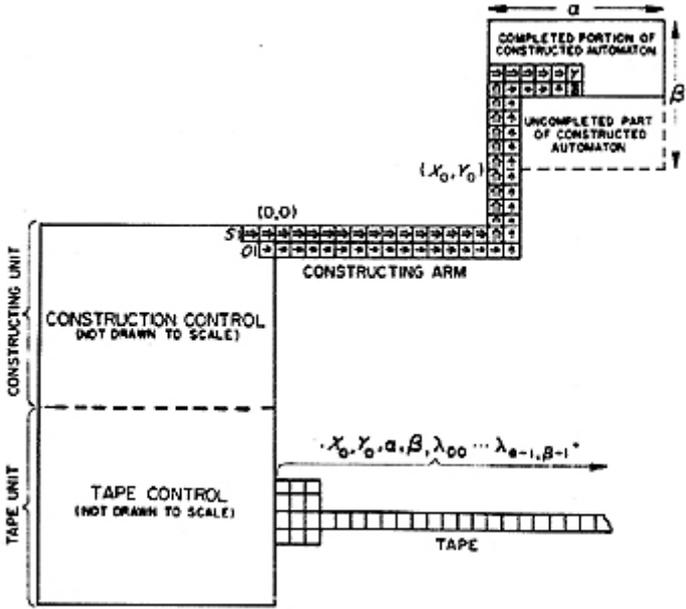
In von Neumann's initial thought experiment (his "kinematic model"), a machine floats around on the surface of a pond, together with lots of machine parts. The machine is a universal constructor: given the description of any machine, it will locate the proper parts and construct that machine. If given a description of itself, it will construct 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.

Von Neumann decided that this model did not properly distinguish the logical form 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 Alamos⁴ — suggested an appropriate formalism, which has come to be known as a cellular automaton (CA).

In brief, a CA 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.

Von Neumann was able to embed the equivalent of his kinematic model as an initial pattern of state assignments within a large CA-lattice using 29 states per cell. 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.⁵ Figure 3 shows a schematic diagram of von Neumann's self-reproducing machine.



3

3

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 unveiled the structure 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:

"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 more important half of the question 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 interesting part has been tossed out. However, if we are concerned with the more general problem of explaining how life-like behaviors emerge out of low-level interactions within a population of logical primitives, we have retained the more interesting portion of the question.

2.7 LESSONS FROM HISTORY

As stated at the beginning of this section, throughout history we have repeatedly tried to map our contemporary technology onto nature in an attempt to understand natural phenomena. Sometimes this has been successful, but in the case of both life and intelligence, such mappings have not provided satisfactory explanations. There is a lesson here: although Artificial Life uses computers as its primary tool for the synthesis of biological phenomena, we must not mistake the tool for the object under study and attempt to characterize life as a "computation." If we are able to bring life to computers, this will not be because life "is" a computation, at least as we now understand the term "computation." Rather, it will be because computers have some fundamental properties that will allow them to be organized in such a way that they can become alive. It is quite likely that we will learn more about computation by studying life than we will learn about life by studying computation. This will be taken up in more detail in the following section.

It is also important to note that in the control programs of early automata we see the roots of what Mitchell Resnick has called the "centralized mindset": the attribution to natural phenomena of a central controller that is ultimately responsible for their behavior. It is a mindset that has dominated most of our scientific, philosophical, and even religious thought for the last several centuries. In contrast, Resnick refers to the kind of approach advocated by Artificial Life as the "distributed mindset". The difference is crucial, since most of nature chugs along in the absence of any central controllers. In order to understand most of nature, therefore, we must abandon the centralized mindset and come to an understanding of the dynamics of distributed systems qua distributed systems.

3 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 has adopted the centralized control architecture of serial, "von Neumann" style computation as a model of intelligence. AL, on the other hand, is attempting to develop a new computational paradigm based on the distributed processes that

support living organisms. That is, AL uses insights from biology to explore the dynamics of interacting information structures. AL has not adopted the computational paradigm as its underlying methodology of behavior generation, nor does it attempt to "explain" life as a "computation".

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 "post-docs" 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 can be relegated to these idiot-savants, and the realm of numerical simulation is 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 over-estimated. One often gains tremendous insight for the essential dynamics of a system by observing its behavior under a wide range of initial conditions.

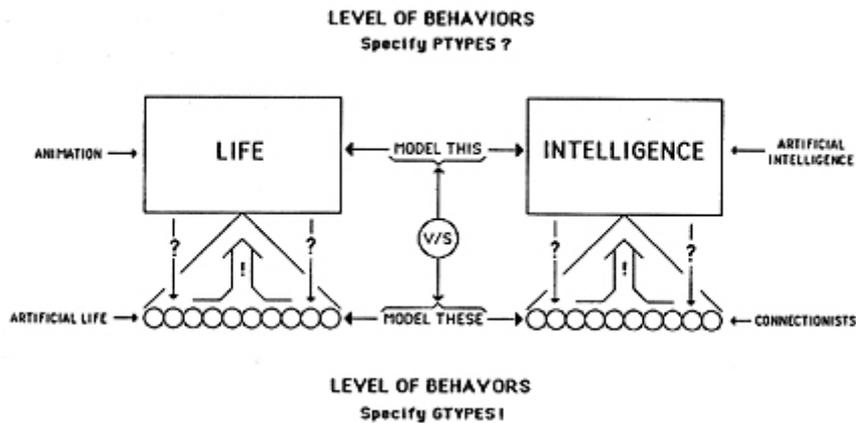
Most importantly, however, computers are beginning to provide scientists with a new paradigm for modeling the world. When dealing with essentially unsolvable governing equations, the primary reason for producing a formal mathematical model — the hope of reaching an analytic solution by symbolic manipulation — 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.⁷

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 within regions of the fluid, but actually computing their motions exactly.⁸

There is an extremely important point here, one that involves the ultimate goals of the scientific enterprise as well as the issue of "centralized" versus "distributed" mindsets. The point is the following. There is a fundamental difference between being able to describe or predict phenomena on the one hand and "explaining" them on the other hand. One can use Navier-Stokes equations to describe or predict fluid flows in many cases, but fluids are not calculating Navier-Stokes equations! The descriptive and predictive power of the Navier-Stokes approach is useful, but the phenomena of fluid flow is actually generated by quite different mechanisms. Physics has largely been considered successful when it has been able to produce a predictive, abstract description of a physical phenomenon, one that generally ignores the low level mechanisms by which the phenomenon is actually generated. This is a benefit of adopting the centralized mindset, and it is very useful to be able to do this whenever possible.

However, for most natural phenomena, there is probably no simpler description than the generative mechanisms themselves. In these circumstances adopting the distributed mindset is necessary. Note that just because we cannot provide an abstract, high-level predictive description of a phenomenon does not mean that we have failed to provide a scientific description, a low-level description may be "as scientific" as we can get concerning the phenomenon. Even when a simpler, high level description is possible, it is important to keep in mind the difference between the description of a process and an understanding of the mechanisms by which it is generated.

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 relatively 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 aggregates of simple components (Figure 4).



4

4

Another surprising lesson is that a distributed system can behave as if it had a central controller when in fact it does not. Computers, especially parallel computers, allow us to adopt Resnick's "distributed mindset" in our studies of Nature. Before computers, it was extremely difficult to come to an understanding of how complex behavior could emerge in the absence of a central controller, which is why the "centralized mindset" dominated the study of Nature for so long. With computers, however, the capacity to build distributed models provides us with the necessary tools for experimenting with the dynamics of distributed systems, and it is becoming apparent that the most interestingly complex behaviors in Nature owe their complexity in part to their distributed, noncentralized architecture.

4 NON LINEARITY AND LOCAL DETERMINATION OF BEHAVIOR.

4.1 LINEAR VS. NONLINEAR SYSTEMS

As mentioned briefly above, the distinction between linear and nonlinear systems is fundamental, and provides excellent insight into why the principles underlying the dynamics of life should be so hard to discover. 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 principle of superposition. 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 understandings 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 principle of superposition. Even if we can break such systems up into simpler constituent parts, and even if we can reach a complete understanding of the parts in isolation, we would not be able to compose 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.

The process that we call "life" is a fundamentally nonlinear process, emerging out of interactions between non-living parts. 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.

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. Analysis is most appropriate for the understanding of linear systems, synthesis is the most appropriate for the understanding of nonlinear systems.

4.2 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-graining 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.

5 ABSTRACTING THE "BIO-LOGIC" OF BIOLOGY

Organisms have been compared to extremely complicated and finely tuned biochemical machines. Since we have seen that it is often 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 form 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.

5.1 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 various 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 a 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.

5.2 GENERALIZED GENOTYPES AND PHENOTYPES

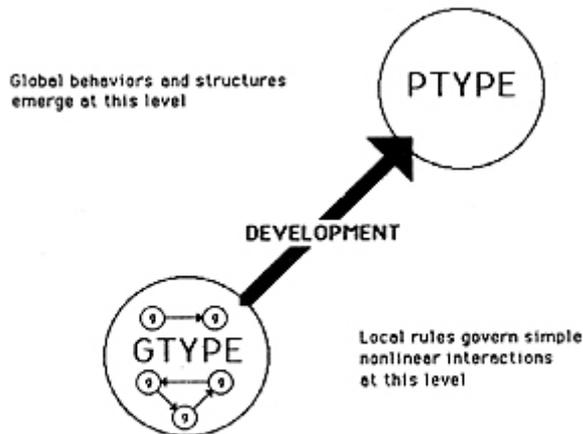
In the context of artificial life, we need to generalize the notions of genotype and phenotype, so that we may apply them in non-biological situations. We will use the term generalized — genotype — or GTYPE — to refer to any largely unordered set of low-level rules, and we will use the term generalized — 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 the context of a specific environment. The GTYPE, essentially, is the specification for a set of machines, while the PTYPE is the behavior that results as the machines are executed and interact with one another.

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 — that contains instructions which 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 multi-level phenomenon. First, there is the PTYPE associated with each particular instruction — the effect that instruction has on an 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 multi-cellular organism — its body, in effect. PTYPES should be complex and multi-level. 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 behaviors — i.e., abstract specifications for "machines" — which will engage in local interactions within a large aggregate of other such behaviors. PTYPES are the behaviors — the structures in time and space — that develop out of these nonlinear, local interactions (Figure 5).



5

5

5.3 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 in general, we cannot predict the PTYPES that will emerge from specific GTYPES, due to the general unpredictability of nonlinear systems. 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, which depends critically on nonlinear interactions. Therefore, a trade-off exists between behaviorally rich but essentially unpredictable nonlinear systems on the one hand, and behaviorally restricted but predictable (and therefore "programmable") linear systems on the other. We shall see in the section on Evolution that the lack of programmability is adequately compensated for by the increased capacity for adaptiveness provided by the rich behavioral repertoire of nonlinear systems.

As discussed previously, we know that it is impossible in the general case to determine nontrivial properties of the future behavior of a sufficiently powerful computer from a mere inspection of its program and its initial state alone.⁹ 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, nontrivial features 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 and the environment.

Similarly, it is not possible in the general case to determine what 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 (an "epistatic trait" in biological terms.) 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! An example from biology would be: What changes would have to be made to the Human genome in order to produce six fingers on each hand rather than five?

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 when nonlinear functions are involved.

6 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".

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.

6.1 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.¹⁰

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 re-written structures. The process can be continued ad-infinitum although some sets of rules will result in a final "fixed-point" configuration in which no more changes will occur.

6.1.1 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) $A \rightarrow CB$
- 2) $B \rightarrow A$
- 3) $C \rightarrow DA$
- 4) $D \rightarrow C$

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 finite state machines. With appropriate restrictions, they are also equivalent to the "regular languages" defined by Chomsky.

6.1.2 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) $AA \rightarrow C[B]D$
- 2) $B \rightarrow A$
- 3) $C \rightarrow C$
- 4) $D \rightarrow C(E)A$
- 5) $E \rightarrow D$

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:

|/|/

\|\|/

E A D C D

\|\|/_**B**_Æ (etc.)

|/|/|/

D B C A Æ C C

|/Æ|/|/

|||

A Æ C C C

|||

t = 0 1 2 3 ... and so on ...

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.

6.1.3 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. By allowing more than one symbol on the left-hand side, the rules can be made *context-sensitive* — *i.e.*, the "context" within which a symbol occurs (the symbols next to it) are important in determining what the replacement string will be. The next example illustrates why this is critical for signal propagation.

In the following example, the symbol in "{ }" 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} Æ C** a "C" at the left-end of the string remains a "C".
- 2) **C{C} Æ C** a "C" with a "C" to its left remains a "C".
- 3) ***{C} Æ*** a "C" with an "*" to its left becomes an "*".
- 4) **{*}C Æ C** an "*" with a "C" to its right becomes a "C".
- 5) **{*} Æ*** 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 CCCCCCC*C

7 CCCCCC*

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.

6.2 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 *GT*YPE — 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 local "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 in this universe can compute and construct other structures, which also may compute and construct.

For example, here is a simple structure that can reproduce itself:

```
22222222
2170140142
2022222202
212 212
202 212
272 212
21222222122222
207107107111112
2222222222222
```

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
212
272
202
212
222222272 22222222
2111701702 2170140142
2122222212 2022222202
212 272 272 212
212 202 212 212
242 212 202 212
212 272 272 212
2022222202 21222222122222
2410710712 207107107111112
22222222 22222222222222
```

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.

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 having the "physics" directly responsible for developing the final structure from a passive seed.

This captures the flavor of what goes on in natural biological 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.

6.3 EXAMPLE 3: FLOCKING "BOIDS"

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.¹¹ 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.

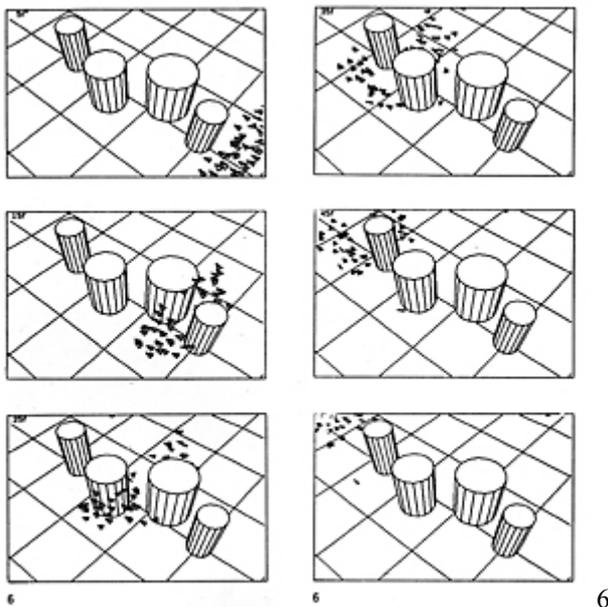
Each Boid in the aggregate shares the same behavioral "tendencies":

to maintain a minimum distance from other objects in the environment, including other Boids, to match velocities with Boids in its neighborhood, 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 re-organize 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 farside of the obstacle and each sub-flock feels anew the "mass" of the other sub-flock (Figure 6).



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.

6.4 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.

6.4.1 IMPORTANCE OF CONTEXT SENSITIVITY.

It is worthwhile to note that *context-sensitive* rules in GTYPE/PYPE 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.

6.4.2 FEEDBACK BETWEEN THE LOCAL AND THE GLOBAL LEVELS.

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.

6.5 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 behaviors 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 behaviors, *flocking Boids and — flocking birds are two instances of the same phenomenon: flocking.*

The behavior of a flock as a whole does not depend critically 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* life-like 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 parallel.

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.

7 EVOLUTION

Modern organisms owe their structure to the complex process of biological evolution, and it is very difficult to discern which of their properties are due to chance, and which to necessity. If biologists could "re-wind the tape" of evolution and start it over, again and again, from different initial conditions, or under different regimes of external perturbations along the way, they would have a full *ensemble* of evolutionary pathways to generalize over. Such an ensemble would allow them to distinguish universal, necessary properties (those which were observed in all the pathways in the ensemble) from accidental, chance properties (those which were unique to individual pathways). However, biologists cannot rewind the tape of evolution, and are stuck with a single, actual evolutionary trace out of a vast, intuited ensemble of possible traces.

Although studying computer models of evolution is not the same as studying the "real thing," the ability to freely manipulate computer experiments — to "rewind the tape", perturb the initial conditions, and so forth — can more than make up for their "lack" of reality.

It has been known for some time that one can evolve computer programs by the process of natural selection among a population of variant programs. Each individual program in a population of programs is evaluated for its performance on some task. The programs that perform best are allowed to "breed" with one another via *Genetic Algorithms*.¹²

The offspring of these better-performing parent programs replace the worst performing programs in the population, and the cycle is iterated. Such evolutionary approaches to program improvement have been applied primarily to the tasks of function optimization and machine learning.

However, such evolutionary models have only recently been applied to the study of evolution itself / cite{Wilson89}. Researchers were primarily concerned with the *results*, rather than with the *process*, of evolution. In what follows, we will review several models that have been used to study the evolutionary process itself.

7.1 ENGINEERING PTYPES FROM GTYPES

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

The consequence of this unpredictability for GTYPE/PYPE systems is that we cannot, in the general case, determine the PYPE that will be produced by an arbitrary GTYPE by

inspection alone. We must "run" the GTYPE in the context of a specific environment, and let the PTYPE develop in order to determine the resulting structure and its behavior.

This is even further complicated when the environment consists of a population of PTYPES engaged in nonlinear interactions, in which case the determination of a PTYPE depends on the behavior of the specific PTYPES it is interacting with, and on the emergent details of the global dynamics.

Since, for any interesting system, there will exist an enormous number of potential GTYPES, and since there is no formal method for deducing the PTYPES from the GTYPES, how do we go about finding GTYPES that will generate life-like PTYPES? or PTYPES that exhibit any particular sought after behavior?

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 would like to be able to automate the process so that our preconceptions and limited abilities to conceive of machinery do not overly constrain the search for GTYPES that will yield the appropriate behaviors.

7.2 NATURAL SELECTION AMONG POPULATIONS OF VARIANTS

Nature, of course, has had to face the same problem, and has hit upon an elegant solution: *evolution by the process of natural selection among populations of 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* GTYPES are duplicated in larger numbers than others, and they are duplicated in such a way that the copies are similar to — but not exactly the same as — the originals. These variant GTYPES develop into variant PTYPES, which enter into the complex interactions within the environment, and the process is continued ad-infinity. As expected from the formal limitations on predictability, GTYPES must be "run" (i.e., turned into PTYPES) in an environment and their behaviors must be evaluated explicitly, their implicit behavior cannot be determined.

7.3 GENETIC ALGORITHMS

In true von Neumann spirit, John Holland has attempted to abstract "the logical form" of the natural process of biological evolution in what he calls the "Genetic Algorithm" (GA).¹³ In the GA, a GTYPE is represented as a character string that encodes a potential solution to a problem. For instance, the character string GTYPES might encode the weight matrix of a neural network or the transition table of a finite state machine. These character strings are rendered as PTYPES via a problem specific interpreter, which constructs the neural net or finite state machine specified by each GTYPE, evaluates its performance in the problem domain, and provides it with a specific fitness value, or "strength".

The GA implements selection by making more copies of the character strings representing the better performing PTYPES. The GA generates variant GTYPES by applying *genetic operators* to these character strings. The genetic operators typically consist of *reproduction*, *crossover* and *mutation*, **with occasional** usage of *inversion* and *duplication*.

Recently, John Koza has developed a version of the GA, which he calls the *Genetic Programming Paradigm* (GPP), that extends the genetic operators to work on GTYPES that are simple expressions in a standard programming language. The GPP differs from the traditional GA in that these program expressions are not represented as simple character strings but rather as the parse trees of the expressions. This makes it easier for the genetic operators to obey the syntax of the programming language when producing variant GTYPES.

Figure 7 shows some examples of GTYPES in the GA and GPP paradigms.

a) 11001010111010110101000101011011101011

b) (OR (NOT a) (AND b c))

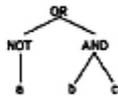


Figure 7. a) A typical character string GTYPE for the GA
 b) A typical programming language expression GTYPE for the GPP and its associated parse tree.

7

7

7.4 THE GENETIC OPERATORS

The genetic operators work as follows.

Reproduction is the most basic operator. It is often implemented in the form of *fitness proportionate reproduction*, which means that strings are duplicated in direct proportion to their relative fitness values. Once all strings have been evaluated, the average fitness of the population is computed, and those strings whose fitness is higher than the population average have a higher than average chance of being duplicated, while those strings whose fitness is lower than the population average have a lower than average chance of being duplicated. There are many variations on this scheme, but most implementations of the GA or the GPP use some form of fitness proportionate reproduction as the means to implement "selection". Another form of this is to simply keep the top 10% or so of the population and throw away the rest, using the survivors as breeding stock for the next generation.

Mutation in the GA is simply the replacement of one or more characters in a character string GTYPE with another character picked at random. In binary strings, this simply amounts to random bit-flips. In the GPP, mutation is implemented by picking a sub-tree of the parse tree at random, and replacing it with a randomly generated sub-tree whose root-node is of the same syntactic type as the root-node of the replaced sub-tree.

Crossover is an analog of sexual recombination. In the GA, this is accomplished by picking two "parent" character strings, lining them up side-by-side, and interchanging equivalent substrings between them, producing two new strings that each contain a mix of their parent's genetic information. Crossover is an extremely important genetic operator: whereas mutation

is equivalent to random search, crossover allows the more "intelligent" search strategy of putting things that have proven useful before into new combinations.

In the GPP, crossover is implemented by picking two "parent" parse trees, locating syntactically similar sub-trees within each, and swapping them.

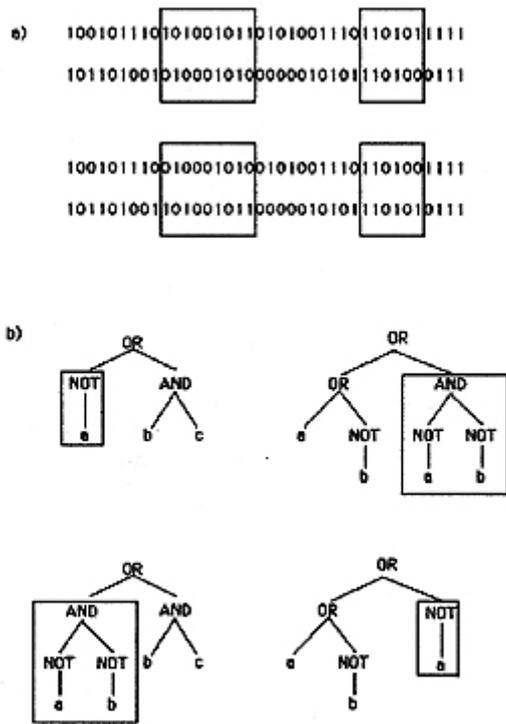


Figure 8: Crossover in a) the GA and b) the GPP. In each, the top line shows the two parent GTYPES with the substructures to be crossed-over outlined by boxes. The bottom line shows the result of performing the crossover operation.

Figure 8 illustrates the crossover operation in the GA and GPP.

Inversion is used rarely in order to rearrange the relative locations of specific pieces of genetic information in the character strings encoding the GTYPE.

Duplication is sometimes used in situations where it makes sense for the genome to grow in length, representing, for instance, larger neural nets, or bigger finite state machine transition tables.

7.5 THE OPERATION OF THE GENETIC ALGORITHM

The basic outline of the genetic algorithm is as follows:

- 1 Generate a random initial population of GTYPES.
- 2 Render the GTYPES in the population as PTYPES and evaluate them in the problem domain, providing each GTYPE with a fitness value.

3 Duplicate GTYPES according to their relative fitness using a scheme like fitness proportionate reproduction.

4 Apply genetic operators to the GTYPES in the population, typically picking cross-over partners as a function of their relative fitness.

5 Replace the least fit GTYPES in the population with the offspring generated in the last several steps.

6 Go back to step 2 and iterate.

Although quite simple in outline, the genetic algorithm has proved remarkably powerful in a wide variety of applications, and provides a useful tool for both the study and the application of evolution.

7.6 THE CONTEXT OF ADAPTATION

In this section, we will compare and contrast the context in which evolution has traditionally been set in GA's with the context in which evolution has occurred in the natural world.

GA's have traditionally been employed in the contexts of machine learning and function optimization. In such contexts, one is often looking for an explicit, optimal solution to a particular, well specified problem. This is reflected in the implementation of the evaluation of PTYPES in traditional GA's, in which each GTYPE is expressed as a PTYPE independently of the others, tested on the problem, and assigned a value representing its individual fitness, using an explicit fitness function. Thus, one is often seeking to evolve an *individual* that explicitly *encodes* an optimal solution to a precisely specified problem. The fitness of a GTYPE in such cases is simply a function of the problem domain, and is independent of the fitnesses of the other GTYPES in the population.

This is quite different from the context in which natural biological evolution has taken place, in which the behavior of a PTYPE and its associated fitness are highly dependent on which other PTYPES exist in the environment, and on the dynamics of their interactions. Furthermore, in the natural context, it is generally the case that there is no single, explicitly specified problem confronting the population. Rather, there is often quite a large set of problems facing the population at any one time, and these problems are only implicitly determined as a function of the dynamics of the population and the environment themselves, which may change significantly over time. In such a context, nature has often discovered that the collective behavior emerging from the interactions among a set of PTYPES will address a sub-set of the implicitly defined problems better than an individual PTYPE can.

Thus the proper picture for the natural evolutionary context is that of a large cloud of potential collective solutions addressing a large cloud of potential collective problems, and that both of these clouds are implicit in the spatio-temporal dynamics of the population. The dynamics of such systems are very complex and impossible to predict.

The important point here is that nonlinearities and emergent collective phenomena are properties that are to be exploited rather than avoided, as has been the traditional engineering viewpoint. Emergent solutions may be harder to understand or to engineer, but there are far more of them than there are non emergent, analyzable linear solutions. The true power of evolution lies in its ability to exploit emergent collective phenomena.

7.7 FROM ARTIFICIAL SELECTION TO NATURAL SELECTION

In *The Origin of Species*, Darwin used a very clever device to argue for the agency of natural selection. In the first chapter of *Origin*, Darwin lays the groundwork of the case for *natural* selection by carefully documenting the process of *artificial selection*. Most people of his time were familiar with the manner in which breeders of domestic animals and plants could enhance traits arbitrarily by selective breeding of their stock. Darwin carefully made the case that the wide variety of domestic animals and plants existent at his time were descended from a much smaller variety of wild-stock, due to the selective breeding imposed by farmers and herders throughout history.

Now, Darwin continues, simply note that environmental conditions can fill the role played by the human breeder in artificial selection, and, Voila! one has natural selection. The rest of the book consists in a very careful documentation of the manner in which different environmental conditions would favor animals bearing different traits, making it more likely that individuals bearing those traits would survive to mate with each other and produce offspring, leading to the gradual enhancement of those traits through time. A beautifully simple yet elegant mechanism to explain the origin and maintenance of the diversity of species on Earth — too simple for many of his time, particularly those of strong religious persuasion.

The abstraction of this simple elegant mechanism for the production and filtration of diversity in the form of the Genetic Algorithm is straightforward and obvious. However, as it is usually implemented, it is artificial, rather than natural selection, that is the agency determining the direction of computer evolution. Either we ourselves, or our algorithmic agents in the form of explicit fitness functions, typically stand in the role of the breeder in computer implementations of evolution. Yet it is plain that the role of "breeder" can as easily be filled by "nature" in the world inside the computer as it is in the world outside the computer — it is just a different "nature." If we are to study evolution by "natural" selection, we must create a "nature" within the artificial world of a computer.

In the following sections, we will explore a number of examples of computational implementations of the evolutionary process, starting with examples that clearly involve artificial selection, and working our way through to an example that clearly involves natural selection. The key thing to keep track of throughout these examples is the manner in which we incrementally give over our role as breeder to the "natural" pressures imposed by the dynamics of the computational world itself.

7.71 A BREEDER'S PARADISE: BIOMORPHS

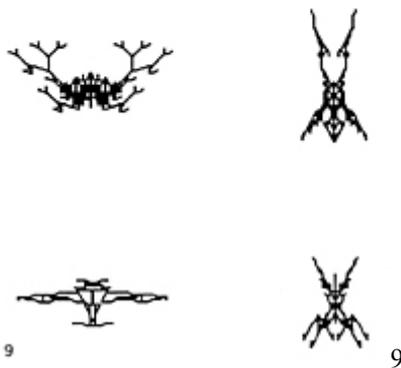
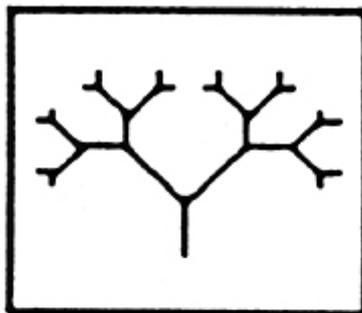
The first model, a clear-cut example of computational artificial selection, is due to the Oxford evolutionary biologist, Richard Dawkins, author of such highly-regarded books as *The Selfish Gene*, *The Extended Phenotype* and *The Blind Watchmaker*.

In order to illustrate the power of a process in which the random production of variation is coupled with a selection mechanism, Dawkins wrote a program for the Apple Macintosh computer that allows users to "breed" recursively generated objects.

The program is set up to generate tree structures recursively by starting with a single stem, adding branches to it in a certain way, adding branches to those branches in the same way, and so on. The number of branches, their angles, their size relative to the stem they are being added to, the number of branching iterations, and other parameters affecting the growth of

these trees are what constitute the GTYPES of the tree-organisms — or "biomorphs" as Dawkins calls them. Thus, the program consists of a general purpose recursive tree generator, which takes each organism's GTYPE (parameter settings) as data and generates its associated PTYPE (the resulting tree).

The program starts by producing a simple default — or "Adam" — tree and then produces a number of mutated copies of the parameter string for the Adam tree. The program renders the PTYPE trees of all of the different GTYPES on the screen for the user to view. The user then selects the PTYPE (i.e. tree shape) he or she likes the best, the program produces mutated copies of that tree's GTYPE (parameter string), and renders their associated PTYPE trees. The user selects another tree, and the process continues. The original Adam tree together with a number of its distant descendants are shown in Figure 9.



It is clear that this is a process of artificial selection. The computer generates the variants, but the human user fills the role of the "breeder", the active selective agent, determining which structures are to go on to produce variant offspring. However, the mechanics of the production of variants are particularly clear: produce slight variations on the presently selected GTYPE. The specific action taken by the human breeder is also very clear: choose the PTYPE whose GTYPE will have variations of it produced in the next round. There is both a producer and a selector of variation.

7.72 ALGORITHMIC BREEDERS

In this section, we investigate a model which will take us two steps closer to natural selection. First the human breeder is replaced with a program he writes, one which formalizes his selection criteria so that the act of selection can be performed by his computational agent: a fitness function. This is a program that will do his breeding for him, applying his selection preferences, but which takes him physically out of the loop. Second, we see that our computational representative can itself be allowed to evolve — an important first step toward eliminating our externally imposed, a priori criteria from the process completely.

The system we discuss here is due to Danny Hillis, inventor of the Connection Machine and chief scientist of Thinking Machines Corp. In the course of the work at TMC, they have a need to design fast and efficient chips for the hardware implementation of a wide variety of common computational tasks, such as sorting numbers. For many of these, there is no body of theory that tells engineers how to construct the optimal circuit to perform the task in question. Therefore, progress in the design of such circuits is often a matter of blind trial and error until a better circuit is discovered. Hillis decided to apply the trial-and-error procedure of evolution to the problem of designing sorting circuits.

In his system, the GTYPES are strings of numbers encoding circuit connections that implement comparisons and swaps between input lines. GTYPES are rendered into the specific circuits they encode — their PTYPES — and they are rated according to the number of circuit elements and connections they require, and by their performance on a number of test strings which they have to sort. This rating is accomplished by an explicit fitness function — Hillis' computational representative — which implements the selection criteria and takes care of the breeding task. Thus, this is still a case of artificial selection, even though there is no human being actively doing the selection.

Hillis implemented the evolution problem on his Connection Machine CM2 — a 64,000 processor SIMD parallel super computer. With populations of 64 K sorting networks over thousands of generations, the system managed to produce a 65 element sorter, better than some cleverly engineered sorting networks, but not as good as the best known such network, which has 60 components. After reaching 65 element sorters, the system consistently got itself hung-up on local optima.

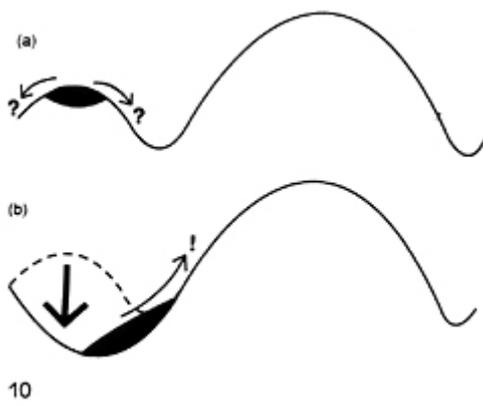
Hillis then borrowed a trick from the biological literature on the coevolution of hosts and parasites¹⁴ and in the process took a step closer to natural selection by allowing the evaluation function to evolve in time. In the previous runs, the sorting networks were evaluated on a fixed set of sorting problems — random sequences of numbers that the networks had to sort into correct order. In the new set of runs, Hillis made another evolving population out of the sorting problems. The task for the sorting networks was to do a good job on the sorting problems, while the task for the sorting problems was to make the sorting networks perform poorly.

In this situation, whenever a good sorting network emerged and took over the population, it became a target for the population of sorting problems. This led to the rapid evolution of sorting sequences that would make the network perform poorly and hence reduce its fitness. Hillis found that this coevolution between the sorting networks and the sorting problems led much more rapidly to better solutions than had been achieved by the evolution of sorting networks alone, resulting in a sorting network consisting of 61 elements.

It is the coevolution in this latter set of runs that both brings us several steps closer to natural selection and is responsible for the enhanced efficiency of the search for an optimal sorting network. First of all, rather than having an absolute, fixed value, the fitness of a sorting network depends on the specific set of sorting problems it is facing.

Likewise, the fitness of a set of sorting problems depends on the specific set of sorting networks it is facing. Thus, the "fitness" of an individual is now a relative quantity, not an absolute one. The fitness of an individual now depends a little more on the "nature" of the world it is embedded in.

Coevolution increases the efficiency of the search as follows: In the earlier runs consisting solely of an evolving population of sorting networks, the population of networks was effectively hill-climbing on a multi-peaked fitness landscape. Therefore, the populations would encounter the classic problem of getting stuck on local-maxima. That is, a population could reach certain structures which lie on relatively low fitness peaks, but from which any deviations result in lower fitness, which is selected against. In order to find another, higher peak, the population would have to cross a fitness valley, which is difficult to do under simple Darwinian selection (Figure 10-[a]).



In the coevolutionary case, here's what happens (Figure 10-[b]). When a population of sorting networks gets stuck on a local fitness peak, it becomes a target for the population of sorting problems. That is, it defines a new peak for the sorting problems to climb. As the sorting problems climb their peak, they *drive down the peak on which the sorting networks are sitting*, by finding sequences that make the sorting networks perform poorly, therefore lowering their fitness. After a while, the fitness peak that the sorting networks were sitting on has been turned into a fitness valley, from which the population can escape by climbing up the neighboring peaks by normal Darwinian means. As the sorting networks climb other peaks, they drive down the peak that they had provided for the sorting problems, who are then free to chase the sorting networks to the new peaks they have achieved and drive those down in turn.

In short, each population dynamically deforms the fitness landscape being traversed by the other population in such a way that both populations can continue to climb uphill without getting stuck in local maxima. When they do get stuck, the maxima get turned into minima which can be climbed out of by simple Darwinian means. Thus, coupled populations evolving by Darwinian means can bootstrap each other up the evolutionary ladder far more efficiently than they can climb it alone. By competing with one another, coupled populations improve one another at increased rates.

Thus, when coupled in this way, a population may get hung up on local optima for a while, but eventually it will be able to climb again. This suggests immediately that the structure of the evolutionary record for such systems should show periods of stasis followed by periods of evolutionary change. The stasis comes about as populations sit at the top of local fitness peaks, waiting around for someone to come along and do them the favor of lowering the peaks they are stuck on. The periods of change come about when populations are released from local optima and are freed to resume climbing fitness peaks, and are therefore changing in time. Hillis has, in fact, carefully documented this kind of *Punctuated Equilibria* in his system.

7.73 COMPUTATIONAL ECOLOGIES

Continuing on our path from artificial to natural selection, we turn to a research project carried out by Kristian Lindgren,¹⁵ in which, although there is still an explicit fitness measure, many different species of organisms coevolve in each other's presence, forming ecological webs allowing for more complex interactions than the simple host-parasite interactions described above.

In this paper, Lindgren studies evolutionary dynamics within the context of a well known game-theoretic problem: the *Iterated Prisoners' Dilemma Model* (IPD). This model has been used effectively by Axelrod and Hamilton in their studies of the evolution of cooperation.¹⁶

In the prisoners' dilemma model, the payoff matrix (the fitness function) is constructed in such a way that individuals will garner the most payoff collectively in the long-run if they "cooperate" with one-another by *avoiding* the behaviors that would garner them the most payoff individually in the short-run. If individuals only play the game once, they will do best by not cooperating ("defecting.") However, if they play the game repeatedly with one another (the "iterated" version of the game,) they will do best by cooperating with one another.

TABLE 1 The payoff matrix for the Prisoner's Dilemma Game. The pair (s_1, s_2) denotes the scores to players A and B, respectively.

		Player B	
		Cooperate	Defect
Player A	Cooperate	(3,3)	(0,5)
	Defect	(5,0)	(1,1)

The payoff matrix for the prisoner's dilemma game is shown in Figure 11. This payoff matrix has the following interesting property. Assume, as is often assumed in game theory, that each player wants to maximize his immediate payoff, and let's analyze what player A should do. If B cooperates, then A should defect, because then A will get a score of 5 whereas he only gets a score of 3 if he cooperates. On the other hand, if B defects, then again A should defect, as he will get a score of 1 if he defects while he only gets a score of 0 if he cooperates. So, no matter what B does, A maximizes his immediate payoff by defecting. Since the payoff matrix is symmetric, the same reasoning applies to player B, so B should defect no matter what A does. Under this reasoning, each player will defect at each time step, giving them one point each per play. However, if they could somehow decide to cooperate, they would each get 3 points per play: the two players will do better in the long run by foregoing the action that maximizes their immediate payoff. The payoff matrix for the prisoner's dilemma game. The pair (s_1, s_2) denotes the scores to players A and B respectively.

The question is, of course: can ordinary Darwinian mechanisms, which assume that individuals selfishly want to maximize their immediate payoff, lead to cooperation? Surprisingly, as demonstrated by Axelrod and Hamilton, the answer is yes.

In Lindgren's version of this game, strategies can evolve in an open-ended fashion by learning to base their behavioral choices upon longer and longer histories of previous interactions.

The scheme used by Lindgren to represent strategies to play the Iterated Prisoners' Dilemma Game is as follows. In the simplest version of the game, players make their choice of whether to cooperate or defect based solely on what their opponent did to them in the last time step. This is called the memory-1 game. Since the opponent could have done only one of two things, cooperate or defect, a strategy needs to specify what it would do in either of those two cases. As it has two moves it can make in either of those two cases, cooperate or defect, there are four possible memory-1 strategies. These can be encoded in bit-strings of length 2, as illustrated in Figure 12.

<u>strategy</u>	<u>name</u>
[0 0]	All Defect
[0 1]	TIT-for-TAT (TFT)
[1 0]	TAT-for-TIT (anti-TFT)
[1 1]	All Cooperate

12

12

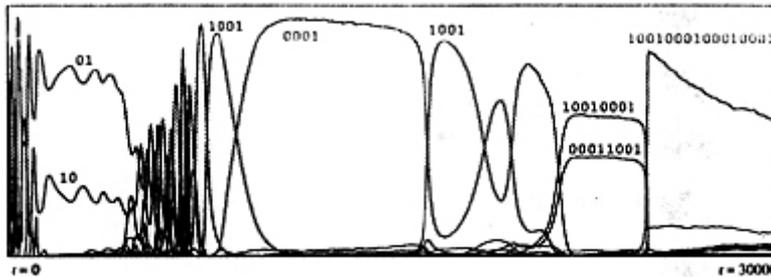
If the players could base their decisions by looking another move into the past, to see what they did to their opponent before their opponent made his move, then we would have the memory-2 game. In this case, there are 2 moves with 2 possible outcomes each, meaning that a memory-2 strategy must specify whether to cooperate or defect for each of 4 possible cases. Such a strategy can be encoded using 4 bits, or twice the length of the memory-1 strategies, and there will be 16 possible memory-2 strategies. Memory-3 strategies require another doubling of the encoding bit string, i.e., 8 bits, yielding 256 possible memory-3 strategies. In general, memory-n strategies require 2 bits for their encoding, and there will be 2 such strategies.

In order to allow for the evolution of higher memory strategies, Lindgren introduces a new genetic operator: gene-duplication. As a memory-n strategy is just twice as long as a memory n-1 strategy, a memory n strategy can be produced from a memory n-1 strategy by simply duplicating the memory n strategy and concatenating the duplicate to itself. In Lindgren's encoding strategy, gene-duplication has the interesting property that it is *a neutral mutation*. Simple duplication alone does not change the PTYPE, even though it has doubled the length of the GTYPE. However, once doubled, mutations in the longer GTYPE will alter the behavior of the PTYPE.

Once again, evolution proceeds by allowing populations of different organisms to bootstrap each other up coupled fitness landscapes, dynamically deforming each other's landscapes by turning local maxima into local minima. Again, the fitness of strategies is not an absolute, fixed number that is independently computable. Rather, the fitness of each strategy depends on what other strategies exist in the "natural" population.

Many complicated and interesting strategies evolve during the evolutionary development of this system. More important, however, are the various phenomenological features exhibited by the dynamics of the evolutionary process. First of all, as we might expect, the system exhibits a behavior that is remarkably suggestive of Punctuated Equilibria. After an initial

irregular transient, the system settles down to relatively long periods of stasis "punctuated" irregularly by periods of rapid evolutionary change.¹⁷



13

13

Second, the diversity of strategies builds up during the long periods of stasis, but often collapses drastically during the short, chaotic episodes of rapid evolutionary succession. These "crashes" in the diversity of species constitute "extinction events." In this model, these extinction events are observed to be a natural consequence of the dynamics of the evolutionary process alone, without invoking any catastrophic, external perturbations (there are no comet impacts or "nemesis" stars in this model!) Furthermore, these extinction events happen on multiple scales: there are lots of little ones and fewer large ones.

In order to understand the dynamics of a system that is subjected to constant perturbations, one needs to understand the dynamics of the unperturbed system first. We do not have access to an unperturbed version of the evolution of life on Earth, consequently we could not have said definitively that extinction events on many scale sizes would be a natural consequence of the process of evolution itself. By comparing the perturbed and unperturbed versions of model systems like Lindgren's, we may very well be able to derive a universal scaling relationship for "natural" extinction events, and therefore be able to explain deviations from this relationship in the fossil record as due to external perturbations such as the impact of large asteroids.

Third, the emergence of ecologies is nicely demonstrated by Lindgren's model. It is usually the case that a mix of several different strategies dominates the system during the long periods of stasis. In order for a strategy to do well, it must do well by cooperating with other strategies. These mixes may involve three or more strategies whose collective activity produces a stable interaction pattern that benefits all of the strategies in the mix. Together, they constitute a more complex, "higher order" strategy, which can behave as a group in ways that are impossible for any individual strategy.

It is important to note that, in many cases, the "environment" that acts on an organism, and in the context of which an organism acts, is primarily constituted of the other organisms in the population and their interactions with each other and the physical environment. There is tremendous opportunity here for evolution to discover that certain sets of individuals exhibit emergent, collective behaviors that reap benefits to all of the individuals in the set. Thus, evolution can produce major leaps in biological complexity, without having to produce more complex individuals by simply discovering, perhaps even "tripping over," the many ways in which collections of individuals at one level can work together to form aggregate individuals at the next higher level of organization.

This is thought to be the case for the origin of eukaryotic cells, which are viewed as descended from early cooperative collections of simpler, prokaryotic cells \ cite(Margulis70). It is also the process involved in the origin of multicellular organisms, which lead to the Cambrian Explosion of diversity some 700 million years ago. It was probably a significant factor in the origin of the prokaryotes themselves, and it has been discovered independently at least 7 times by the various social insects (including species of wasps, bees, ants, and termites).

7.74 THE EMERGENCE OF VIRTUAL NATURE

The final step in eliminating our hand from the selection/breeding process and setting the stage for true "natural" selection within a computer is taken in a model due to Tom Ray.¹⁸ This step involves eliminating our algorithmic breeding agent completely.

In his "Tierra" simulation system, computer programs compete for CPU time and memory space. The "task" that these programs must perform in order to be reproduced is simply the act of self-reproduction itself! Thus, there is no need for an externally defined fitness function that determines which GTYPES get copied by an external copying procedure. The programs reproduce themselves, and the ones that are better at this task take over the population. The whole external task of evaluation of fitness has been internalized in the function of the organisms themselves. Thus, there is no longer a place for the Human breeder or his computational agent. This results in genuine natural selection within a computer.

In Tierra, programs replicate themselves "noisily," so that some of their offspring behave differently. Variant programs that reproduce themselves more efficiently, which trick other programs into reproducing them, or which capture the execution pointers of other programs, etc., will leave more offspring than others. Similarly, programs that learn to defend themselves against such tricks will leave more offspring than those that do not.

We will discuss a few of the "digital organisms" that have emerged within the Tierra system (it is not necessary to understand the code in the illustrated programs in order to follow the explanation in the text.)

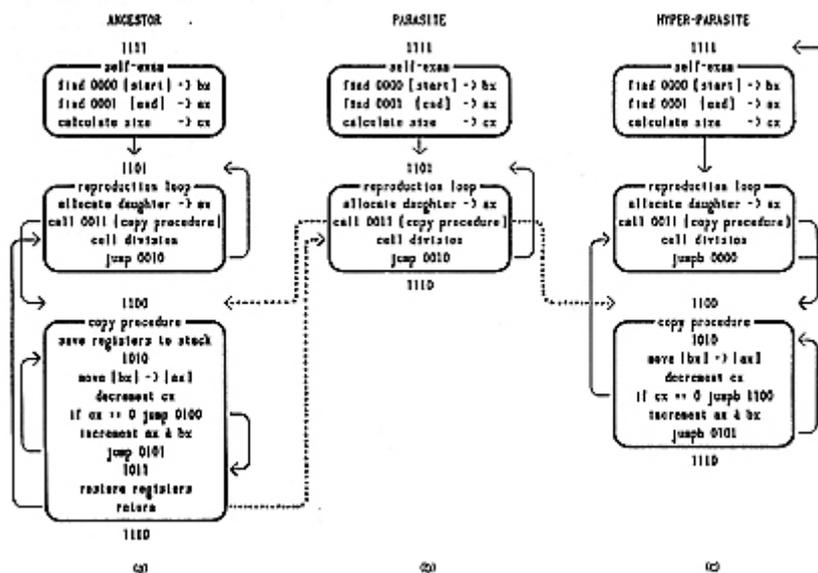


Figure 14-(a) shows the self-replicating "ancestor" program that is the only program Tom Ray has ever written in the Tierra system. All the other programs evolved under the action of natural selection.

The ancestor program works as follows. In the top block of code, the program locates its "head" and its "tail," templates marking the upper and lower boundaries of the program in memory. It saves these locations in special registers and, after subtracting the location of the head from the location of the tail, it stores its length in another register.

In the second block of code, the program enters an endless loop in which it will repeatedly produce copies of itself. It allocates memory space of the appropriate size and then invokes the final block of code, which is the actual reproduction loop. After it returns from the reproduction loop, it creates a new execution pointer to its newly produced offspring, and cycles back to create another offspring.

In the third and final block of code, the reproduction loop, the program copies itself, instruction by instruction, into the newly allocated memory space, making use of the addresses and length stored away by the first block of code. When it has copied itself completely, it returns to the block of code that called it, in this case, the second block.

It should be noted that "function calls" in Tierra are accomplished by seeking for a specific bit pattern in memory rather than by branching to a specific address. Thus, when the second block of code "calls" the third block of code, the reproduction loop, it does so by initiating a seek forward in memory for a specific "template." When this template is found, execution begins at the instruction following the template. Returns from function calls are handled in the normal manner, by simply returning to the instruction following the initial function call. This template addressing scheme is used in other reference contexts as well, and helps make Tierra language programs robust to mutations, as well as easily relocatable in memory.

Figure 14-(b) shows a "parasite" program that has evolved to exploit the ancestor program. The parasite is very much like the ancestor program, except that it is missing the third block of code, the reproduction loop. How then does it copy itself?

The answer is that it makes use of a nearby ancestor program's reproduction loop! Recall that a function call in Tierra initiates a seek forward in memory for a particular template of bits. If this pattern is not found within the initiating program's own code, the search may proceed forward in memory into the code of other organisms, where the template may be found and where execution then begins. When the invoked function in another organism's code executes the "return" statement, execution reverts to the program that initiated the function call. Thus, organisms can execute each other's code, and this is exactly what the parasite program does: it makes use of the reproductive machinery of the ancestor host.

This means that the parasite does not have to take the time to copy the code constituting the reproductive loop, and hence can reproduce more rapidly, as it has fewer instructions to copy. The parasites thus proliferate in the population. However, they cannot proliferate to the point of driving out the ancestor hosts altogether, for they depend on them for their reproductive machinery. Thus, a balance is eventually struck optimizing the joint system.

Eventually, however, another mutant form of the ancestor emerges which has developed an immunity to the parasites. This program is illustrated in Figure 14-(c). Two key differences from the ancestor program confer the immunity to the parasite programs. First, instead of

executing a "return" instruction, the reproduction loop instead initiates a jump back in memory to the template found in the instruction that calls the reproduction loop. This has the same effect as a return statement when executed by the immune program, but has a very different effect on the parasite. The second important difference is that following the cell division in the second block (which allocates a new execution pointer to the offspring just created) the program jumps back to the beginning of the first block of code, rather than to the beginning of the second block. Thus, the immune program constantly resets its head, tail, and size registers. This seems useless when considering only the immune organism's own reproduction, but let's see what happens when a parasite tries to execute the reproduction loop in an immune organism.

When a parasite attempts to use the immune program's reproduction code, the new jump transfers the parasite's execution pointer to the second block of the immune program's code, rather than returning it to the second block of the parasite code, as the parasite expects. Then, this execution pointer is further redirected to the first block of the immune program, where the registers originally containing the head, tail and length of the parasite are reset to contain the head, tail, and length of the immune organism. The immune program has thus completely captured the execution pointer of the parasite. Having lost its execution pointer, the parasite simply becomes dormant data occupying memory, while the immune program now has two execution pointers running through it: its own original pointer, plus the pointer it captured from the parasite. Thus, the immune program now reproduces twice as rapidly as before. Once they emerge, such immune programs rapidly drive the parasites to extinction.

Complex interactions between variant programs like those described above continue to develop within evolutionary runs in Tierra. From a uniform population of self-reproducing ancestor programs, Ray, a tropical biologist by training, notes the emergence of whole "ecologies" of interacting species of computer programs. Furthermore, he is able to identify many phenomena familiar to him from his studies of real ecological communities, such as competitive exclusion, the emergence of parasites, key-stone predators and parasites, hyper-parasites, symbiotic relationships, sociality, "cheaters", and so forth.

Again, the actual "fitness" of an organism is a complex function of its interactions with other organisms in the "soup". Collections of programs can cooperate to enhance each other's reproductive success, or they can drive each other's reproductive success down, thus lowering fitness and kicking the population off of local fitness peaks.

Not surprisingly, Ray, too, has noted periods of relative stasis punctuated by periods of rapid evolutionary change, as complex ecological webs collapse and new ones stabilize in their place. Systems like Ray's Tierra capture the proper context for evolutionary dynamics, and natural selection is truly at play here.

Digital Organisms from Ray's Tierra Simulation System. (a) Self-reproducing ancestor program. (b) An early parasite of the ancestor. (c) A descendant of the ancestry that is immune to the parasite.

8 ARTIFICIAL LIFE IN HARDWARE AND WETWARE

Although much of this article has discussed attempts to synthesize biological phenomena within the "software" medium of a computer, similar attempts are being carried out in both "wetware" (working with real molecules in biochemistry laboratories) and "hardware" (typically, mobile robots). Despite the difference in underlying medium, the same principles

apply to Artificial Life research being carried out across these vastly different material substrates: bottom-up implementation of large populations of semi-autonomous individuals, which get caught up in emergent, collective, and evolving dynamics. In this section we briefly review several Artificial Life research efforts in "hardware" and "wetware".

8.1 HARDWARE

A new approach to robotics is being pioneered by researchers including Rod Brooks at MIT, Luc Steels at Brussels, Leslie Kaelbling of Teleos Research, Tim Smithers at Edinburgh, and Mark Tilden at Waterloo. This new approach involves throwing out the goal of endowing a robot with human-level perception, representation, reasoning, and cognitive capacities using traditional Artificial Intelligence approaches. In its place, it substitutes the goal of building a "street-smart" robot whose intelligence is more along the lines of an insect than of a human. The "intelligence" of such robots is distributed across (what we would call) a "peripheral" sensory and motor nervous system rather than being centrally located within a central processor analog of the cerebral cortex. This sounds like intelligence without brains, but it is grounded in the realization that higher level cognitive capabilities are late-comers in evolutionary history, and that they depend critically on the much earlier and more basic sensory and motor capabilities provided by simple, decentralized nervous systems. Despite their relative simplicity, these early nerve nets were able to move organisms around successfully within their environments with nary a representation, a model of the world, nor a plan to their names. "Elephants don't play chess!" is one of the battle cries of this new approach, although "Cockroaches don't play chess!" might be even more appropriate.

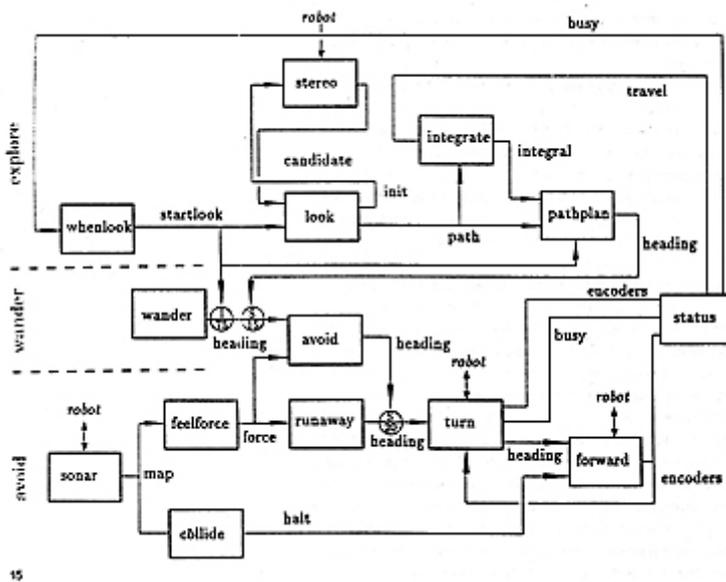
In pursuing this new approach to robotics, Rod Brooks and his colleagues at the MIT Mobile Robot Lab (the "Mobot" Lab) have emphasized the following:¹⁹

- that there would be no traditional notion of planning.
- that no central representation was needed.
- that notions of world modelling are impractical and unnecessary.
- that biology and evolution were good models to follow in our quest.
- that we insist on building complete systems that exist in the real world so that we would not trick ourselves into skipping hard problems.

This philosophical approach to the problem has lead Brooks and his students to a bottom-up, distributed scheme for building robots and their control structures, which they call the *subsumption architecture*. In the subsumption scheme, there are many layers of activity going on in parallel, with lower layers being controlled (subsumed) by higher layers.

For instance, imagine that we have a three-wheeled robot with two rear drive wheels and a front idler wheel. The two drive wheels are driven independently, which is how steering is effected. The robot also has sonar range finders, and a TV camera for vision. The lowest layer of control in the subsumption architecture (level-0) might simply be a circuit that causes the robot to move forward by turning on both drive wheels full-speed. The next layer up (level-1) would be a circuit taking data from the sonar range finders with inhibitory outputs to the drive layer (level-0). As obstacles come within range of the sonar sensors, they selectively inhibit the signals being sent to one or the other of the drive wheels in the underlying level-0 circuitry, causing the robot to turn to avoid the obstacles. Notice that level-1 assumes the presence of level-0, but level-0 need not assume the existence of any higher levels. Level-0 simply does its thing, while level-1 occasionally steps in and tweaks this underlying primary behavior.

The next layer up, level-2, might be a circuit which identifies objects in the room and causes the robot to head toward them. The level-2 circuit would have inputs from the TV camera and outputs to level-1, the obstacle avoidance circuitry. When distant objects are located by the TV camera, the obstacle avoidance circuitry can be tweaked by level-2 to cause the robot to head in the general direction of these distant objects, relying on the autonomous behavior of level-1 to avoid obstacles along the way. Again, level-2 assumes the existence of level-1, but level-1 need not assume the existence of any higher level(s). Thus, each layer "subsumes" the layers beneath it, inhibiting or enhancing their autonomous behavior in specific ways in the pursuit of higher-level goals. In principle, this layered control structure can be iterated to many higher levels. Figure 15 shows a diagram of the subsumption architecture of an actual robot control.



Subsumption architecture control circuitry for one of Rod Brooks' mobile robots. Higher levels of control structure are farther up in the diagram.

On this scheme, therefore, it is not that "fish got to swim, birds got to fly," but rather that "fins got to flip, wings got to flap." Swimming and flying are seen to emerge from the controlled application of these primary behaviors, as the flippers and flappers are modulated by higher level control circuitry connected to sensory circuits, enhancing or inhibiting the simple behaviors of the the flippers or flappers, which simply want to "do their thing." In this approach, "doers" are primary while controllers are secondary, whereas on the traditional AI approach, controllers are primary and "doers" are secondary. Higher level intelligence, then, is seen to have arisen as control structures on top of pre-existing simple behaviors, giving rise to higher-order behaviors, over which even higher-order control structures emerged, and so forth. This is a view of intelligence driven from the bottom-up rather than from the top-down, which has the advantage that it is consistent with the probable evolutionary development of higher intelligence.

A good source of inspiration for this bottom-up approach to intelligent robotics is the book *Vehicles* by Braitenberg.²⁰

8.2 WETWARE

An increasing number of biomolecular laboratories around the world are attempting to synthesize chemical systems from scratch that exhibit life-like properties. Some of these involve the same kinds of organic molecules that make up natural life, but put together in new ways, while others involve molecules that are largely foreign to natural life.

In 1990, Julius Rebek and his colleagues at MIT demonstrated a molecular system that self-replicates via templating, in a manner similar to the self-replication of DNA.²¹ However, the molecules involved consist of an aromatic ester and an amino-adenosine, rather than DNA.

Similar feats had been accomplished previously by Gunter von Kiedrowski of the University of Göttingen and also by Leslie Orgel of the Salk Institute. However, these latter efforts involved slight modifications of naturally occurring biomolecules, whereas Rebek's system involves truly synthetic molecules not found in natural life.

In Switzerland, at the ETH in Zurich, Bachmann, Walde, Luisi, and Lang have succeeded in synthesizing self-replicating artificial cell-like structures called "micelles," consisting of mono-layer lipid membranes.²² The micelles replicate as a chemical reaction within them produces more membrane material.

In Gerald Joyce's lab at the Scripps Research Institute, a molecular version of the genetic algorithm is being employed to produce novel RNA-based enzymes by the mechanism of Darwinian evolution.²³ This is an extremely promising technology, which should provide the medical world with a whole new method for designing useful drugs. Part of the excitement about this new technology (called "directed molecular evolution" or in vitro molecular evolution) is that it allows researchers to avoid having to solve the "protein-folding" problem. A protein's chemical properties are a function of its complex, folded structure. In order to produce a molecule with a particular chemical behavior, drug designers have traditionally had to work out the specific linear sequence of bases on a peptide string that would fold-up into just the right shape.

Directed molecular evolution saves drug-designers from having to worry about sequence details. Instead, they can start out with a molecule that has a function similar to the function that they seek, produce many close variants of that molecule, and test the resulting variants to see how they perform on the desired task. The variants that perform better have more copies of themselves made via the PCR reaction, again with some variation. The process is continued until molecules performing the desired task are produced. Joyce has successfully evolved a ribozyme (an RNA-based enzyme) that will cleave DNA, starting with a ribozyme that cleaves only RNA). At no time in this process need the researchers concern themselves with the specific linear sequences of the molecules, which is the analog of the genotype). Rather, they apply selective forces directly at the phenotypic level of molecular function.

There is a good deal more work along these and similar lines proceeding in biochemical laboratories around the world. In fact, Gerald Joyce predicts that synthetic wetware life will be realized in some biochemical lab by the turn of the century.

9 CONCLUSION

In the section on Evolution, the point was made that it is often the case that the most important aspects of "Nature" in the process of evolution are the other organisms in the environment and their effects on the physical world.

In the case of human beings, our "Nature" is largely dominated by the products of our own artifice, not only physical structures such as buildings and highways, but also such socio-cultural constructs as governments, churches, corporations, and so forth. That is, our own artifacts are the dominating aspects of the "Nature" that is the agency of "natural" selection for the human species. By adapting to an artifactual "Nature" we are becoming more and more "artifactual" ourselves. As we are increasingly responsible for constructing "Nature," so we are increasingly responsible for constructing ourselves. In a very real sense, humans themselves are becoming a form of Artificial Life-products of our own artifice. Now that we are on the verge of synthesizing life artificially — now that our artifacts are becoming more and more like us — and now that we are becoming more and more like our artifacts, where will life go from here?

It is important to bear in mind that whether we produce living artifacts within the next ten years or only within the next thousand, on the scale of geological time, we are literally at the beginning of a new epoch of life on Earth, an epoch that will be dominated by life-forms that we will have had a significant role in shaping. Will they sustain and nurture us, and we them? Or will they destroy and replace us, or we them?

We must do everything in our power, *starting now*, to assure that the consequences of our imminent mastery of the technology of life are beneficial for all life, whatever it is made of.

LITERATURE AND NOTES

Axelrod, R., and W.D. Hamilton "The Evolution of Cooperation." (1981): pp. 1390—1396.

Axelrod, R. *The Evolution of Cooperation*. New York, Basic Books, 1984.

Bachmann, Walde, Luisi, and Lang, *J. Am. Chem. Soc.*, **113** 8204, 1991.

Braitenberg, V. *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, 1984.

Brooks, R., and Flynn, A.M., "Fast, Cheap, and Out of Control: A Robot Invasion of the Solar System." *Journal of the British Interplanetary Society* Vol. 42, pp. 478—485, 1989.

Brooks, R. "Intelligence without Representation." *Artificial Intelligence* **47**, pp. 139—159, 1991.

Buss, L. *The Evolution of individuality*. Princeton University Press, 1987.

Burks, A. W. (ed) *Essays on Cellular Automata*, University of Illinois Press, Urbana, 1970.

Chapuis, A., and Droz, E., *Automata: A Historical and Technological Study*. (Trans. A. Reid), B. T. Batsford Ltd, London, 1958.

Dawkins, R. "The Evolution of Evolvability."

In *Artificial Life*, edited by C.G. Langton, Addison Wesley, pp 201—220, 1989.

Frisch, U., Hasslacher, B., and Pomeau, Y., "Lattice gas automata for the Navier-Stokes equation." *Physical Review Letters* **56**, pp. 1505—1508, 1986.

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

Hamilton, W. D. "Sex versus Nonsex versus Parasite." *Oikos* **35**, pp. 282—290, 1980.

Hamilton, W. D. "Pathogens as causes of Genetic Diversity in their Host Populations." in *Population Biology of Infectious Diseases.* edited by R. M. Anderson and R. M. May, Berlin, Springer-Verlag, pp. 269—296, 1982.

Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

Hopcroft, J. E., and Ullman, J. D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Menlo Park, Calif. 1979.

Beaudry, A. A., and Joyce, G. F., "Directed Evolution of an RNA Enzyme." *Science*, Vol. 257, pp. 635—641, July 31, 1992. Joyce, G. F., "Directed Molecular Evolution." *Scientific American*, Dec. 92, Vol. 267 no. 6. pp. 90—97, 1992.

Langton, C. G., "Self-Reproduction in Cellular Automata", *Physica D*, Vol. 10, No. 1—2, pp. 135—144 (1984).

Langton, C. G., "Studying Artificial Life with Cellular Automata", *Physica D*, Vol. 22, pp. 120—149, 1986.

Lindgren, K. "Evolutionary Phenomena in Simple Dynamics." In *Artificial Life II*, edited by C. G. Langton *et al.*, Redwood City CA, Addison Wesley, pp. 295—312, 1991. Margulis, L. *Origin of Eucaryotic Cells.* New Haven, Yale University Press, 1970.

Prusinkiewicz, P. and Lindenmayer, A., *The Algorithmic Beauty of Plants*. Berlin, Springer-Verlag, 1991.

Ray, T.S., "An Approach to the Synthesis of Life." In *Artificial Life II*, edited by C. G. Langton *et al.*, Redwood City CA, Addison Wesley, pp. 371—408, 1991.

Rebek, J., *et al.*, *J. Am. Chem. Soc.* **112**, 1,249, 1990.

Reynolds, C. W., "Flocks, Herds, and Schools: A Distributed Behavioral Model." *Proceedings of SIGGRAPH '87*, in *Computer Graphics* **V** 21, no. 4, pp. 25—34 (July, 1987).

Toffoli, T., "Cellular Automata as an Alternative to (Rather than an Approximation of) Differential Equations in Modeling Physics." *Physica D*, V10(1—2), 1984.

Toffoli, T., and Margolus, N., *Cellular Automata Machines.* MIT Press, 1987.

Ulam, S., "On some Mathematical Problems Connected with Patterns of Growth of Figures", *Proceedings of Symposia in Applied Mathematics* **14**, pp. 215—224. Reprinted in *Burks70*.

Von Neumann, J., *Theory of Self-Reproducing Automata*, edited and completed by A. W. Burks, U. of Illinois Press, Urbana, 1966.

Wilson, S. W. "The Genetic Algorithm and Simulated Evolution." In *Artificial Life II*, edited by C. G. Langton *et al.*, Redwood City CA, Addison Wesley, pp. 157—165, 1989.

Wolfram, S. "Cellular automaton fluids 1: Basic theory." *Journal of Statistical Physics* 45, pp. 471—526, 1986.

Figure 1

Two views of the mechanical duck attributed to Vaucanson.

Figure 2

Two views of a drawing automaton built by the Jaquet-Droz family

Figure 3

Schematic diagram of von Neumann's CA self-reproducing configuration.

Figure 5
The relationship between GTYPE and PTYPE.

Figure 6
A flock of "Boids" negotiating a field of columns.

Figure 7
GTYPES in (a) the GA and (b) the GPP paradigms

Figure 8
Crossover-operation in (a) the GA and (b) the GPP.

Figure 9
(a) Dawkin's "Adam" tree and
(b) some of its descendants.

Figure 10
(a) Population of sorting networks stuck on a local fitness peak. In order to attain the higher peak, the population must cross a fitness "valley," which is difficult to achieve under normal Darwinian mechanisms.
(b) The coevolving parasites deform the fitness landscape of the sorting networks, turning the fitness peak into a fitness valley, from which it is easy for the population to escape.

Figure 11
The payoff matrix for the Prisoners' Dilemma Game. The pair (s_1, s_2) denotes the scores to players A and B respectively.

Figure 12
The four possible memory-1 strategies for playing the Prisoners' Dilemma Game.

Figure 13
During evolutionary development, the system settles down to relatively long periods of stasis "punctuated" by periods of rapid evolutionary change.