

Thomas S. Ray

School of Life & Health Sciences, University of Delaware, Newark, Delaware 19716,
ray@brahms.udel.edu

An Approach to the Synthesis of Life

Ray, T. In Press. An approach to the synthesis of life. In: *Artificial Life II*, Santa Fe Institute Studies in the Sciences of Complexity, vol. XI, (Farmer, J. D., C. Langton, S. Rasmussen, & C. Taylor, eds). Redwood City, CA: Addison-Wesley, 1991.

———. In Press. Population dynamics of digital organisms. In: "Artificial Life II Video Proceedings" C.G. Langton (Editor), Redwood City, CA: Addison Wesley, 1991.

Marcel, a mechanical chessplayer... his exquisite 19th-century brainwork — the human art it took to build which has been flat lost, lost as the dodo bird ... But where inside Marcel is the midget Grandmaster, the little Johann Allgeier? where's the pantograph, and the magnets? Nowhere. Marcel really is a mechanical chessplayer. No fakery inside to give him any touch of humanity at all.

— Thomas Pynchon, *Gravity's Rainbow*.

INTRODUCTION

Ideally, the science of biology should embrace all forms of life. However in practice, it has been restricted to the study of a single instance of life, life on earth. Life on earth is very diverse, but it is presumably all part of a single phylogeny. Because biology is based on a sample size of one, we can not know what features of life are peculiar to earth, and what features are general, characteristic of all life. A truly comparative natural biology would require inter-planetary travel, which is light years away. The ideal experimental evolutionary biology would involve creation of multiple planetary systems, some essentially identical, others varying by a parameter of interest, and observing them for billions of years.

A practical alternative to an inter-planetary or mythical biology is to create synthetic life in a computer. The objective is not necessarily to create life forms that would serve as models for the study of natural life, but rather to create radically different life forms, based on a completely different physics and chemistry, and let these life forms evolve their own phylogeny, leading to whatever forms are natural to their unique physical basis. These truly independent instances of life may then serve as a basis for comparison, to gain some insight into what is general and what is peculiar in biology. Those aspects of life that prove

to be general enough to occur in both natural and synthetic systems can then be studied more easily in the synthetic system. “Evolution in a bottle” provides a valuable tool for the experimental study of evolution and ecology.

The intent of this work is to synthesize rather than simulate life. This approach starts with hand crafted organisms already capable of replication and open-ended evolution, and aims to generate increasing diversity and complexity in a parallel to the Cambrian explosion.

To state such a goal leads to semantic problems, because life must be defined in a way that does not restrict it to carbon based forms. It is unlikely that there could be general agreement on such a definition, or even on the proposition that life need not be carbon based. Therefore, I will simply state my conception of life in its most general sense. I would consider a system to be living if it is self-replicating, and capable of open-ended evolution. Synthetic life should self-replicate, and evolve structures or processes that were not designed-in or pre-conceived by the creator (Pattee, 1989).

Core Wars programs, computer viruses, and worms (Cohen, 1984; Dewdney, 1984, 1985a, 1987, 1989; Denning, 1988; Rheingold, 1988; Spafford et al., 1989) are capable of self-replication, but fortunately, not evolution. It is unlikely that such programs will ever become fully living, because they are not likely to be able to evolve.

Most evolutionary simulations are not open-ended. Their potential is limited by the structure of the model, which generally endows each individual with a genome consisting of a set of pre-defined genes, each of which may exist in a pre-defined set of allelic forms (Holland, 1975; Dewdney, 1985b; Dawkins, 1987, 1989; Packard, 1989; Ackley & Littman, 1990). The object being evolved is generally a data structure representing the genome, which the simulator program mutates and/or recombines, selects, and replicates according to criteria designed into the simulator. The data structures do not contain the mechanism for replication, they are simply copied by the simulator if they survive the selection phase.

Self-replication is critical to synthetic life because without it, the mechanisms of selection must also be pre-determined by the simulator. Such artificial selection can never be as creative as natural selection. The organisms are not free to invent their own fitness functions. Freely evolving creatures will discover means of mutual exploitation and associated implicit fitness functions that we would never think of. Simulations constrained to evolve with pre-defined genes, alleles and fitness functions are dead ended, not alive.

The approach presented here does not have such constraints. Although the model is limited to the evolution of creatures based on sequences of machine instructions, this may have a potential comparable to evolution based on sequences of organic molecules. Sets of machine instructions similar to those used in the Tierra Simulator have been shown to be capable of “universal computation” (Aho et al., 1974; Minsky, 1976; Langton, 1989). This suggests that evolving machine codes should be able to generate any level of complexity.

Other examples of the synthetic approach to life can be seen in the work of Holland (1976), Farmer et al. (1986), Langton (1986), Rasmussen et al. (1990), and Bagley et al. (preprint). A characteristic these efforts generally have in common is that they parallel the origin of life event by attempting to create prebiotic conditions from which life may emerge

spontaneously and evolve in an open ended fashion.

While the origin of life is generally recognized as an event of the first order, there is another event in the history of life that is less well known but of comparable significance: the origin of biological diversity and macroscopic multicellular life during the Cambrian explosion 600 million years ago. This event involved a riotous diversification of life forms. Dozens of phyla appeared suddenly, many existing only fleetingly, as diverse and sometimes bizarre ways of life were explored in a relative ecological void (Gould, 1989; Morris, 1989).

The work presented here aims to parallel the second major event in the history of life, the origin of diversity. Rather than attempting to create prebiotic conditions from which life may emerge, this approach involves engineering over the early history of life to design complex evolvable organisms, and then attempting to create the conditions that will set off a spontaneous evolutionary process of increasing diversity and complexity of organisms. This work represents a first step in this direction, creating an artificial world which may roughly parallel the RNA world of self-replicating molecules (still falling far short of the Cambrian explosion).

The approach has generated rapidly diversifying communities of self-replicating organisms exhibiting open-ended evolution by natural selection. From a single rudimentary ancestral creature containing only the code for self-replication, interactions such as parasitism, immunity, hyper-parasitism, sociality and cheating have emerged spontaneously. This paper presents a methodology and some first results.

Here was a world of simplicity and certainty no acidhead, no revolutionary anarchist would ever find, a world based on the one and zero of life and death. Minimal, beautiful. The patterns of lives and deaths.... weightless, invisible chains of electronic presence or absence. If patterns of ones and zeros were “like” patterns of human lives and deaths, if everything about an individual could be represented in a computer record by a long string of ones and zeros, then what kind of creature would be represented by a long string of lives and deaths? It would have to be up one level at least — an angel, a minor god, something in a UFO.

— Thomas Pynchon, *Vineland*.

METHODS

THE METAPHOR

Organic life is viewed as utilizing energy, mostly derived from the sun, to organize matter. By analogy, digital life can be viewed as using CPU (central processing unit) time, to organize memory. Organic life evolves through natural selection as individuals compete for resources (light, food, space, etc.) such that genotypes which leave the most descendants increase in frequency. Digital life evolves through the same process, as replicating algorithms compete for CPU time and memory space, and organisms evolve strategies to exploit one another.

CPU time is thought of as the analog of the energy resource, and memory as the analog of the spatial resource.

The memory, the CPU and the computer's operating system are viewed as elements of the "abiotic" environment. A "creature" is then designed to be specifically adapted to the features of the environment. The creature consists of a self-replicating assembler language program. Assembler languages are merely mnemonics for the machine codes that are directly executed by the CPU. These machine codes have the characteristic that they directly invoke the instruction set of the CPU and services provided by the operating system.

All programs, regardless of the language they are written in, are converted into machine code before they are executed. Machine code is the natural language of the machine, and machine instructions are viewed by this author as the "atomic units" of computing. It is felt that machine instructions provide the most natural basis for an artificial chemistry of creatures designed to live in the computer.

In the biological analogy, the machine instructions are considered to be more like the amino acids than the nucleic acids, because they are "chemically active". They actively manipulate bits, bytes, CPU registers, and the movements of the instruction pointer (see below). The digital creatures discussed here are entirely constructed of machine instructions. They are considered analogous to creatures of the RNA world, because the same structures bear the "genetic" information and carry out the "metabolic" activity.

A block of RAM memory (random access memory, also known as "main" or "core" memory) in the computer is designated as a "soup" which can be inoculated with creatures. The "genome" of the creatures consists of the sequence of machine instructions that make up the creature's self-replicating algorithm. The prototype creature consists of 80 machine instructions, thus the size of the genome of this creature is 80 instructions, and its "genotype" is the specific sequence of those 80 instructions.

THE VIRTUAL COMPUTER — TIERRA SIMULATOR

The computers we use are general purpose computers, which means, among other things, that they are capable of emulating through software, the behavior of any other computer that ever has been built or that could be built (Aho et al., 1974; Minsky, 1976; Langton, 1989). We can utilize this flexibility to design a computer that would be especially hospitable to synthetic life.

There are several good reasons why it is not wise to attempt to synthesize digital organisms that exploit the machine codes and operating systems of real computers. The most urgent is the potential threat of natural evolution of machine codes leading to virus or worm type of programs that could be difficult to eradicate due to their changing "genotypes". This potential argues strongly for creating evolution exclusively in programs that run only on virtual computers and their virtual operating systems. Such programs would be nothing more than data on a real computer, and therefore would present no more threat than the data in a data base or the text file of a word processor.

Another reason to avoid developing digital organisms in the machine code of a real computer is that the artificial system would be tied to the hardware and would become

obsolete as quickly as the particular machine it was developed on. In contrast, an artificial system developed on a virtual machine could be easily ported to new real machines as they become available.

A third issue, which potentially makes the first two moot, is that the machine languages of real machines are not designed to be evolvable, and in fact might not support significant evolution. Von Neuman type machine languages are considered to be “brittle”, meaning that the ratio of viable programs to possible programs is virtually zero. Any mutation or recombination event in a real machine code is almost certain to produce a non-functional program. The problem of brittleness can be mitigated by designing a virtual computer whose machine code is designed with evolution in mind. Farmer & Belin (in press) have suggested that overcoming this brittleness and “Discovering how to make such self-replicating patterns more robust so that they evolve to increasingly more complex states is probably the central problem in the study of artificial life.”

The work described here takes place on a virtual computer known as Tierra (Spanish for Earth). Tierra is a parallel computer of the MIMD (multiple instruction, multiple data) type, with a processor (CPU) for each creature. Parallelism is imperfectly emulated by allowing each CPU to execute a small time slice in turn. Each CPU of this virtual computer contains two address registers, two numeric registers, a flags register to indicate error conditions, a stack pointer, a ten word stack, and an instruction pointer. Each virtual CPU is implemented via the C structure listed in Appendix A. Computations performed by the Tierran CPUs are probabilistic due to flaws that occur at a low frequency (see Mutation below).

The instruction set of a CPU typically performs simple arithmetic operations or bit manipulations, within the small set of registers contained in the CPU. Some instructions move data between the registers in the CPU, or between the CPU registers and the RAM (main) memory. Other instructions control the location and movement of an “instruction pointer” (IP). The IP indicates an address in RAM, where the machine code of the executing program (in this case a digital organism) is located.

The CPU perpetually performs a fetch-decode-execute-increment-IP cycle: The machine code instruction currently addressed by the IP is fetched into the CPU, its bit pattern is decoded to determine which instruction it corresponds to, and the instruction is executed. Then the IP is incremented to point sequentially to the next position in RAM, from which the next instruction will be fetched. However, some instructions like JMP, CALL and RET directly manipulate the IP, causing execution to jump to some other sequence of instructions in the RAM. In the Tierra Simulator this CPU cycle is implemented through the `time_slice` routine listed in Appendix B.

THE TIERRAN LANGUAGE

Before attempting to set up an Artificial Life system, careful thought must be given to how the representation of a programming language affects its adaptability in the sense of being robust to genetic operations such as mutation and recombination. The nature of the virtual computer is defined in large part by the instruction set of its machine language. The approach in this study has been to loosen up the machine code in a “virtual bio-computer”, in order to create a computational system based on a hybrid between biological and classical

von Neumann processes.

In developing this new virtual language, which is called “Tierran”, close attention has been paid to the structural and functional properties of the informational system of biological molecules: DNA, RNA and proteins. Two features have been borrowed from the biological world which are considered to be critical to the evolvability of the Tierran language.

First, the instruction set of the Tierran language has been defined to be of a size that is the same order of magnitude as the genetic code. Information is encoded into DNA through 64 codons, which are translated into 20 amino acids. In its present manifestation, the Tierran language consists of 32 instructions, which can be represented by five bits, *operands included*.

Emphasis is placed on this last point because some instruction sets are deceptively small. Some versions of the redcode language of Core Wars (Dewdney, 1984, 1987; Rasmussen et al., 1990) for example are defined to have ten operation codes. It might appear on the surface that the instruction set is of size ten. However, most of the ten instructions have one or two operands. Each operand has four addressing modes, and then an integer. When we consider that these operands are embedded into the machine code, we realize that they are in fact a part of the instruction set, and this set works out to be about 10^{11} in size. Inclusion of numeric operands will make any instruction set extremely large in comparison to the genetic code.

In order to make a machine code with a truly small instruction set, we must eliminate numeric operands. This can be accomplished by allowing the CPU registers and the stack to be the only operands of the instructions. When we need to encode an integer for some purpose, we can create it in a numeric register through bit manipulations: flipping the low order bit and shifting left. The program can contain the proper sequence of bit flipping and shifting instructions to synthesize the desired number, and the instruction set need not include all possible integers.

A second feature that has been borrowed from molecular biology in the design of the Tierran language is the addressing mode, which is called “address by template”. In most machine codes, when a piece of data is addressed, or the IP jumps to another piece of code, the exact numeric address of the data or target code is specified in the machine code. Consider that in the biological system by contrast, in order for protein molecule A in the cytoplasm of a cell to interact with protein molecule B, it does not specify the exact coordinates where B is located. Instead, molecule A presents a template on its surface which is complementary to some surface on B. Diffusion brings the two together, and the complementary conformations allow them to interact.

Addressing by template is illustrated by the Tierran JMP instruction. Each JMP instruction is followed by a sequence of NOP (no-operation) instructions, of which there are two kinds: NOP_0 and NOP_1. Suppose we have a piece of code with five instruction in the following order: JMP NOP_0 NOP_0 NOP_0 NOP_1. The system will search outward in both directions from the JMP instruction looking for the nearest occurrence of the complementary pattern: NOP_1 NOP_1 NOP_1 NOP_0. If the pattern is found, the instruction pointer will move to the end of the pattern and resume execution. If the pattern is not found, an error condition (flag) will be set and the JMP instruction will be ignored (in practice, a

limit is placed on how far the system may search for the pattern).

The Tierran language is characterized by two unique features: a truly small instruction set without numeric operands, and addressing by template. Otherwise, the language consists of familiar instructions typical of most machine languages, e.g., MOV, CALL, RET, POP, PUSH etc. The complete instruction set is listed in Appendix B.

THE TIERRAN OPERATING SYSTEM

The Tierran virtual computer needs a virtual operating system that will be hospitable to digital organisms. The operating system will determine the mechanisms of interprocess communication, memory allocation, and the allocation of CPU time among competing processes. Algorithms will evolve so as to exploit these features to their advantage. More than being a mere aspect of the environment, the operating system together with the instruction set will determine the topology of possible interactions between individuals, such as the ability of pairs of individuals to exhibit predator-prey, parasite-host or mutualistic relationships.

Memory Allocation — Cellularity

The Tierran computer operates on a block of RAM of the real computer which is set aside for the purpose. This block of RAM is referred to as the “soup”. In most of the work described here the soup consisted of 60,000 bytes, which can hold the same number of Tierran machine instructions. Each “creature” occupies some block of memory in this soup.

Cellularity is one of the fundamental properties of organic life, and can be recognized in the fossil record as far back as 3.6 billion years (Barbieri, 1985). The cell is the original individual, with the cell membrane defining its limits and preserving its chemical integrity. An analog to the cell membrane is needed in digital organisms in order to preserve the integrity of the informational structure from being disrupted easily by the activity of other organisms. The need for this can be seen in AL models such as cellular automata where virtual state machines pass through one another (Langton, 1986, 1987), or in core wars type simulations where coherent structures demolish one another when they come into contact (Dewdney, 1984, 1987; Rasmussen et al., 1990).

Tierran creatures are considered to be cellular in the sense that they are protected by a “semi-permeable membrane” of memory allocation. The Tierran operating system provides memory allocation services. Each creature has exclusive write privileges within its allocated block of memory. The “size” of a creature is just the size of its allocated block (e.g., 80 instructions). This usually corresponds to the size of the genome. While write privileges are protected, read and execute privileges are not. A creature may examine the code of another creature, and even execute it, but it can not write over it. Each creature may have exclusive write privileges in at most two blocks of memory: the one that it is born with which is referred to as the “mother cell”, and a second block which it may obtain through the execution of the MAL (memory allocation) instruction. The second block, referred to as the “daughter cell”, may be used to grow or reproduce into.

When Tierran creatures “divide”, the mother cell loses write privileges on the space of the daughter cell, but is then free to allocate another block of memory. At the moment of division, the daughter cell is given its own instruction pointer, and is free to allocate its own

second block of memory.

Time Sharing — The Slicer

The Tierran operating system must be multi-tasking in order for a community of individual creatures to live in the soup simultaneously. The system doles out small slices of CPU time to each creature in the soup in turn. The system maintains a circular queue called the “slicer queue”. As each creature is born, a virtual CPU is created for it, and it enters the slicer queue just ahead of its mother, which is the active creature at that time. Thus the newborn will be the last creature in the soup to get another time slice after the mother, and the mother will get the next slice after its daughter. As long as the slice size is small relative to the generation time of the creatures, the time sharing system causes the world to approximate parallelism. In actuality, we have a population of virtual CPUs, each of which gets a slice of the real CPU’s time as it comes up in the queue.

The number of instructions to be executed in each time slice is set proportional to the size of the genome of the creature being executed, raised to a power. If the “slicer power” is equal to one, then the slicer is size neutral, the probability of an instruction being executed does not depend on the size of the creature in which it occurs. If the power is greater than one, large creatures get more CPU cycles per instruction than small creatures. If the power is less than one, small creatures get more CPU cycles per instruction. The power determines if selection favors large or small creatures, or is size neutral. A constant slice size selects for small creatures.

Mortality — The Reaper

Self-replicating creatures in a fixed size soup would rapidly fill the soup and lock up the system. To prevent this from occurring, it is necessary to include mortality. The Tierran operating system includes a “reaper” which begins “killing” creatures when the memory fills to some specified level (e.g., 80%). Creatures are killed by deallocating their memory, and removing them from both the reaper and slicer queues. Their “dead” code is not removed from the soup.

In the present system, the reaper uses a linear queue. When a creature is born it enters the bottom of the queue. The reaper always kills the creature at the top of the queue. However, individuals may move up or down in the reaper queue according to their success or failure at executing certain instructions. When a creature executes an instruction that generates an error condition, it moves one position up the queue, as long as the individual ahead of it in the queue has not accumulated a greater number of errors. Two of the instructions are somewhat difficult to execute without generating an error, therefore successful execution of these instructions moves the creature down the reaper queue one position, as long as it has not accumulated more errors than the creature below it.

The effect of the reaper queue is to cause algorithms which are fundamentally flawed to rise to the top of the queue and die. Vigorous algorithms have a greater longevity, but in general, the probability of death increases with age.

Mutation

In order for evolution to occur, there must be some change in the genome of the creatures. This may occur within the lifespan of an individual, or there may be errors in passing along the genome to offspring. In order to insure that there is genetic change, the operating system randomly flips bits in the soup, and the instructions of the Tierran language are imperfectly executed.

Mutations occur in two circumstances. At some background rate, bits are randomly selected from the entire soup (60,000 instructions totaling 300,000 bits) and flipped. This is analogous to mutations caused by cosmic rays, and has the effect of preventing any creature from being immortal, as it will eventually mutate to death. The background mutation rate has generally been set at about one bit flipped for every 10,000 Tierran instructions executed by the system.

In addition, while copying instructions during the replication of creatures, bits are randomly flipped at some rate in the copies. The copy mutation rate is the higher of the two, and results in replication errors. The copy mutation rate has generally been set at about one bit flipped for every 1,000 to 2,500 instructions moved. In both classes of mutation, the interval between mutations varies randomly within a certain range to avoid possible periodic effects.

In addition to mutations, the execution of Tierran instructions is flawed at a low rate. For most of the 32 instructions, the result is off by plus or minus one at some low frequency. For example, the increment instruction normally adds one to its register, but it sometimes adds two or zero. The bit flipping instruction normally flips the low order bit, but it sometimes flips the next higher bit or no bit. The shift left instruction normally shifts all bits one bit to the left, but it sometimes shifts left by two bits, or not at all. In this way, the behavior of the Tierran instructions is probabilistic, not fully deterministic.

It turns out that bit flipping mutations and flaws in instructions are not necessary to generate genetic change and evolution, once the community reaches a certain state of complexity. Genetic parasites evolve which are sloppy replicators, and have the effect of moving pieces of code around between creatures, causing rather massive rearrangements of the genomes. The mechanism of this ad hoc sexuality has not been worked out, but is likely due to the parasites' inability to discriminate between live, dead or embryonic code.

Mutations result in the appearance of new genotypes, which are watched by an automated genebank manager. In one implementation of the manager, when new genotypes replicate twice, producing a genetically identical offspring at least once, they are given a unique name and saved to disk. Each genotype name contains two parts, a number and a three letter code. The number represents the number of instructions in the genome. The three letter code is used as a base 26 numbering system for assigning a unique label to each genotype in a size class. The first genotype to appear in a size class is assigned the label aaa, the second is assigned the label aab, and so on. Thus the ancestor is named 80aaa, and the first mutant of size 80 is named 80aab. The first parasite of size 45 is named 45aaa.

The genebanker saves some additional information with each genome: the genotype name of its immediate ancestor which makes possible the reconstruction of the entire phylogeny; the time and date of origin; "metabolic" data including the number of instructions executed

in the first and second reproduction, the number of errors generated in the first and second reproduction, and the number of instructions copied into the daughter cell in the first and second reproductions (see Appendix C); some environmental parameters at the time of origin including the search limit for addressing, and the slicer power, both of which affect selection for size.

THE TIERRAN ANCESTOR

The Tierran language has been used to write a single self-replicating program which is 80 instructions long. This program is referred to as the “ancestor”, or alternatively as genotype 0080aaa (Fig. 1). The ancestor is a minimal self-replicating algorithm which was originally written for use during the debugging of the simulator. No functionality was designed into the ancestor beyond the ability to self-replicate, nor was any specific evolutionary potential designed in. The commented Tierran assembler and machine code for this program is presented in Appendix C.

The ancestor examines itself to determine where in memory it begins and ends. The ancestor’s beginning is marked with the four no-operation template: 1 1 1 1, and its ending is marked with 1 1 1 0. The ancestor locates its beginning with the five instructions: ADRB, NOP_0, NOP_0, NOP_0, NOP_0. This series of instructions causes the system to search backwards from the ADRB instruction for a template complementary to the four NOP_0 instructions, and to place the address of the complementary template (the beginning) in the ax register of the CPU (see Appendix A). A similar method is used to locate the end.

Having determined the address of its beginning and its end, it subtracts the two to calculate its size, and allocates a block of memory of this size for a daughter cell. It then calls the copy procedure which copies the entire genome into the daughter cell memory, one instruction at a time. The beginning of the copy procedure is marked by the four no-operation template: 1 1 0 0. Therefore the call to the copy procedure is accomplished with the five instructions: CALL, NOP_0, NOP_0, NOP_1, NOP_1.

When the genome has been copied, it executes the DIVIDE instruction, which causes the creature to lose write privileges on the daughter cell memory, and gives an instruction pointer to the daughter cell (it also enters the daughter cell into the slicer and reaper queues). After this first replication, the mother cell does not examine itself again; it proceeds directly to the allocation of another daughter cell, then the copy procedure is followed by cell division, in an endless loop.

Forty-eight of the eighty instructions in the ancestor are no-operations. Groups of four no-operation instructions are used as complementary templates to mark twelve sites for internal addressing, so that the creature can locate its beginning and end, call the copy procedure, and mark addresses for loops and jumps in the code, etc. The functions of these templates are commented in the listing in Appendix C.

RESULTS

GENERAL BEHAVIOR OF THE SYSTEM

Evolutionary runs of the simulator are begun by inoculating the soup of 60,000 instructions with a single individual of the 80 instruction ancestral genotype. The passage of time in a run is measured in terms of how many Tierran instructions have been executed by the simulator. Most software development work has been carried out on a Toshiba 5200/100 laptop computer with an 80386 processor and an 80387 math co-processor operating at 20 Mhz. This machine executes over 12 million Tierran instructions per hour. Long evolutionary runs are conducted on mini and mainframe computers which execute about one million Tierran instructions per minute.

The original ancestral cell which inoculates the soup executes 839 instructions in its first replication, and 813 for each additional replication. The initial cell and its replicating daughters rapidly fill the soup memory to the threshold level of 80% which starts the reaper. Typically, the system executes about 400,000 instructions in filling up the soup with about 375 individuals of size 80 (and their gestating daughter cells). Once the reaper begins, the memory remains roughly 80% filled with creatures for the remainder of the run.

Once the soup is full, individuals are initially short lived, generally reproducing only once before dying, thus individuals turn over very rapidly. More slowly, there appear new genotypes of size 80, and then new size classes. There are changes in the genetic composition of each size class, as new mutants appear, some of which increase significantly in frequency, sometimes replacing the original genotype. The size classes which dominate the community also change through time, as new size classes appear (see below), some of which competitively exclude sizes present earlier. Once the community becomes diverse, there is a greater variance in the longevity and fecundity of individuals.

In addition to an increase in the raw diversity of genotypes and genome sizes, there is an increase in the ecological diversity. Obligate commensal parasites evolve, which are not capable of self-replication in isolated culture, but which can replicate when cultured with normal (self-replicating) creatures. These parasites execute some parts of the code of their hosts, but cause them no direct harm, except as competitors. Some potential hosts have evolved immunity to the parasites, and some parasites have evolved to circumvent this immunity.

In addition, facultative hyper-parasites have evolved, which can self-replicate in isolated culture, but when subjected to parasitism, subvert the parasites energy metabolism to augment their own reproduction. Hyper-parasites drive parasites to extinction, resulting in complete domination of the communities. The relatively high degrees of genetic relatedness within the hyper-parasite dominated communities leads to the evolution of sociality in the sense of creatures that can only replicate when they occur in aggregations. These social aggregations are then invaded by hyper-hyper-parasite cheaters.

Mutations and the ensuing replication errors lead to an increasing diversity of sizes and genotypes of self-replicating creatures in the soup. Within the first 100 million instructions of elapsed time, the soup evolves to a state in which about a dozen more-or-less persistent size classes coexist. The relative abundances and specific list of the size classes varies over time. Each size class consists of a number of distinct genotypes which also vary over time.

EVOLUTION

Micro-Evolution

If there were no mutations at the outset of the run, there would be no evolution. However, the bits flipped as a result of copy errors or background mutations result in creatures whose list of 80 instructions (genotype) differs from the ancestor, usually by a single bit difference in a single instruction.

Mutations in and of themselves, can not result in a change in the size of a creature, they can only alter the instructions in its genome. However, by altering the genotype, mutations may affect the process whereby the creature examines itself and calculates its size, potentially causing it to produce an offspring that differs in size from itself.

Four out of the five possible mutations in a no-operation instruction convert it into another kind of instruction, while one out of five converts it into the complementary no-operation. Therefore 80% of mutations in templates destroy the template, while one in five alters the template pattern. An altered template may cause the creature to make mistakes in self examination, procedure calls, or looping or jumps of the instruction pointer, all of which use templates for addressing.

parasites

An example of the kind of error that can result from a mutation in a template is a mutation of the low order bit of instruction 42 of the ancestor (Appendix C). Instruction 42 is a NOP_0, the third component of the copy procedure template. A mutation in the low order bit would convert it into NOP_1, thus changing the template from 1 1 0 0 to: 1 1 1 0. This would then be recognized as the template used to mark the end of the creature, rather than the copy procedure.

A creature born with a mutation in the low order bit of instruction 42 would calculate its size as 45. It would allocate a daughter cell of size 45 and copy only instructions 0 through 44 into the daughter cell. The daughter cell then, would not include the copy procedure. This daughter genotype, consisting of 45 instructions, is named 0045aaa.

Genotype 0045aaa (Fig. 1) is not able to self-replicate in isolated culture. However, the semi-permeable membrane of memory allocation only protects write privileges. Creatures may match templates with code in the allocated memory of other creatures, and may even execute that code. Therefore, if creature 0045aaa is grown in mixed culture with 0080aaa, when it attempts to call the copy procedure, it will not find the template within its own genome, but if it is within the search limit (generally set at 200–400 instructions) of the copy procedure of a creature of genotype 0080aaa, it will match templates, and send its instruction pointer to the copy code of 0080aaa. Thus a parasitic relationship is established (see ECOLOGY below). Typically, parasites begin to emerge within the first few million instructions of elapsed time in a run.

immunity to parasites

At least some of the size 79 genotypes demonstrate some measure of resistance to parasites. If genotype 45aaa is introduced into a soup, flanked on each side with one individual of genotype 0079aab, 0045aaa will initially reproduce somewhat, but will be quickly eliminated

from the soup. When the same experiment is conducted with 0045aaa and the ancestor, they enter a stable cycle in which both genotypes coexist indefinitely. Freely evolving systems have been observed to become dominated by size 79 genotypes for long periods, during which parasitic genotypes repeatedly appear, but fail to invade.

circumvention of immunity to parasites

Occasionally these evolving systems dominated by size 79 were successfully invaded by parasites of size 51. When the immune genotype 0079aab was tested with 0051aao (a direct, one step, descendant of 0045aaa in which instruction 39 is replaced by an insertion of seven instructions of unknown origin), they were found to enter a stable cycle. Evidently 0051aao has evolved some way to circumvent the immunity to parasites possessed by 0079aab. The fourteen genotypes 0051aaa through 0051aan were also tested with 0079aab, and none were able to invade.

hyper-parasites

Hyper-parasites have been discovered, (e.g., 0080gai, which differs by 19 instructions from the ancestor, Fig. 1). Their ability to subvert the energy metabolism of parasites is based on two changes. The copy procedure does not return, but jumps back directly to the proper address of the reproduction loop. In this way it effectively seizes the instruction pointer from the parasite. However it is another change which delivers the coup de grâce: after each reproduction, the hyper-parasite re-examines itself, resetting the bx register with its location and the cx register with its size. After the instruction pointer of the parasite passes through this code, the CPU of the parasite contains the location and size of the hyper-parasite and the parasite thereafter replicates the hyper-parasite genome.

social hyper-parasites

Hyper-parasites drive the parasites to extinction. This results in a community with a relatively high level of genetic uniformity, and therefore high genetic relationship between individuals in the community. These are the conditions that support the evolution of sociality, and social hyper-parasites soon dominate the community. Social hyper-parasites (Fig. 2) appear in the 61 instruction size class. For example, 0061acg is social in the sense that it can only self-replicate when it occurs in aggregations. When it jumps back to the code for self-examination, it jumps to a template that occurs at the end rather than the beginning of its genome. If the creature is flanked by a similar genome, the jump will find the target template in the tail of the neighbor, and execution will then pass into the beginning of the active creature's genome. The algorithm will fail unless a similar genome occurs just before the active creature in memory. Neighboring creatures cooperate by catching and passing on jumps of the instruction pointer.

It appears that the selection pressure for the evolution of sociality is that it facilitates size reduction. The social species are 24% smaller than the ancestor. They have achieved this size reduction in part by shrinking their templates from four instructions to three instructions. This means that there are only eight templates available to them, and catching each others jumps allows them to deal with some of the consequences of this limitation as well as to make dual use of some templates.

cheaters: hyper-hyper-parasites

The cooperative social system of hyper-parasites is subject to cheating, and is eventually invaded by hyper-hyper-parasites (Fig. 2). These cheaters (e.g., 0027aab) position themselves between aggregating hyper-parasites so that when the instruction pointer is passed between

them, they capture it.

a novel self-examination

All creatures discussed thus far mark their beginning and end with templates. They then locate the addresses of the two templates and determine their genome size by subtracting them. In one run, creatures evolved without a template marking their end. These creatures located the address of the template marking their beginning, and then the address of a template in the middle of their genome. These two addresses were then subtracted to calculate half of their size, and this value was multiplied by two (by shifting left) to calculate their full size.

Macro-Evolution

When the simulator is run over long periods of time, hundreds of millions or billions of instructions, various patterns emerge. Under selection for small sizes there is a proliferation of small parasites and a rather interesting ecology (see below). Selection for large creatures has usually lead to continuous incrementally increasing sizes (but not to a trivial concatenation of creatures end-to-end) until a plateau in the upper hundreds is reached. In one run, selection for large size lead to apparently open ended size increase, evolving genomes larger than 23,000 instructions in length. This evolutionary pattern might be described as phyletic gradualism.

The most thoroughly studied case for long runs is where selection, as determined by the slicer function, is size neutral. The longest runs to date (as much as 2.86 billion Tierran instructions) have been in a size neutral environment, with a search limit of 10,000, which would allow large creatures to evolve if there were some algorithmic advantage to be gained from larger size. These long runs illustrate a pattern which could be described as periods of stasis punctuated by periods of rapid evolutionary change, which appears to parallel the pattern of punctuated equilibrium described by Eldredge & Gould (1972) and Gould & Eldredge (1977).

Initially these communities are dominated by creatures with genome sizes in the eighties. This represents a period of relative stasis, which has lasted from 178 million to 1.44 billion instructions in the several long runs conducted to date. The systems then very abruptly (in a span of 1 or 2 million instructions) evolve into communities dominated by sizes ranging from about 400 to about 800. These communities have not yet been seen to evolve into communities dominated by either smaller or substantially larger size ranges.

The communities of creatures in the 400 to 800 size range also show a long-term pattern of punctuated equilibrium. These communities regularly come to be dominated by one or two size classes, and remain in that condition for long periods of time. However, they inevitably break out of that stasis and enter a period where no size class dominates. These periods of rapid evolutionary change may be very chaotic. Close observations indicate that at least at some of these times, no genotypes breed true. Many self-replicating genotypes will coexist in the soup at these times, but at the most chaotic times, none will produce offspring which are even their same size. Eventually the system will settle down to another period of stasis dominated by one or a few size classes which breed true.

Two communities have been observed to die after long periods. In one community, a chaotic period led to a situation where only a few replicating creatures were left in the soup, and these were producing sterile offspring. When these last replicating creatures died (presumably from an accumulation of mutations) the community was dead. In these runs, the mutation rate was not lowered during the run, while the average genome size increased by an order of magnitude until it approached the average mutation rate. Both communities died shortly after the dominant size class moved from the 400 range to the 700 to 1400 range. Under these circumstances it is probably difficult for any genome to breed true, and the genomes may simply have “melted”. Another community died abruptly when the mutation rate was raised to a high level.

DIVERSITY

Most observations on the diversity of Tierran creatures have been based on the diversity of size classes. Creatures of different sizes are clearly genetically different, as their genomes are of different sizes. Different sized creatures would have some difficulty engaging in recombination if they were sexual, thus it is likely that they would be different species. In a run of 526 million instructions, 366 size classes were generated, 93 of which achieved abundances of five or more individuals. In a run of 2.56 billion instructions, 1180 size classes were generated, 367 of which achieved abundances of five or more.

Each size class consists of a number of distinct genotypes which also vary over time. There exists the potential for great genetic diversity within a size class. There are 32^{80} distinct genotypes of size 80, but how many of those are viable self-replicating creatures? This question remains unanswered, however some information has been gathered through the use of the automated genebank manager.

In several days of running the genebanker, over 29,000 self-replicating genotypes of over 300 size classes accumulated. The size classes and the number of unique genotypes banked for each size are listed in Table 1. The genotypes saved to disk can be used to inoculate new soups individually, or collections of these banked genotypes may be used to assemble “ecological communities”. In “ecological” runs, the mutation rates can be set to zero in order to inhibit evolution.

ECOLOGY

The only communities whose ecology has been explored in detail are those that operate under selection for small sizes. These communities generally include a large number of parasites, which do not have functional copy procedures, and which execute the copy procedures of other creatures within the search limit. In exploring ecological interactions, the mutation rate is set at zero, which effectively throws the simulation into ecological time by stopping evolution. When parasites are present, it is also necessary to stipulate that creatures must breed true, since parasites have a tendency to scramble genomes, leading to evolution in the absence of mutation.

0045aaa is a “metabolic parasite”. Its genome does not include the copy procedure, however it executes the copy procedure code of a normal host, such as the ancestor. In an environment favoring small creatures, 0045aaa has a competitive advantage over the ancestor,

however, the relationship is density dependent. When the hosts become scarce, most of the parasites are not within the search limit of a copy procedure, and are not able to reproduce. Their calls to the copy procedure fail and generate errors, causing them to rise to the top of the reaper queue and die. When the parasites die off, the host population rebounds. Hosts and parasites cultured together demonstrate Lotka-Volterra population cycling (Lotka, 1925; Volterra, 1926; Wilson & Bossert, 1971).

A number of experiments have been conducted to explore the factors affecting diversity of size classes in these communities. Competitive exclusion trials were conducted with a series of self-replicating (non-parasitic) genotypes of different size classes. The experimental soups were initially inoculated with one individual of each size. A genotype of size 79 was tested against a genotype of size 80, and then against successively larger size classes. The interactions were observed by plotting the population of the size 79 class on the x axis, and the population of the other size class on the y axis. Sizes 79 and 80 were found to be competitively matched such that neither was eliminated from the soup. They quickly entered a stable cycle, which exactly repeated a small orbit. The same general pattern was found in the interaction between sizes 79 and 81.

When size 79 was tested against size 82, they initially entered a stable cycle, but after about 4 million instructions they shook out of stability and the trajectory became chaotic with an attractor that was symmetric about the diagonal (neither size showed any advantage). This pattern was repeated for the next several size classes, until size 90, where a marked asymmetry of the chaotic attractor was evident, favoring size 79. The run of size 79 against size 93 showed a brief stable period of about a million instructions, which then moved to a chaotic phase without an attractor, which spiraled slowly down until size 93 became extinct, after an elapsed time of about 6 million instructions.

An interesting exception to this pattern was the interaction between size 79 and size 89. Size 89 is considered to be a “metabolic cripple”, because although it is capable of self-replicating, it executes about 40% more instructions than normal to replicate. It was eliminated in competition with size 79, with no loops in the trajectory, after an elapsed time of under one million instructions.

In an experiment to determine the effects of the presence of parasites on community diversity, a community consisting of twenty size classes of hosts was created and allowed to run for 30 million instructions, at which time only the eight smallest size classes remained. The same community was then regenerated, but a single genotype (0045aaa) of parasite was also introduced. After 30 million instructions, 16 size classes remained, including the parasite. This seems to be an example of a “keystone” parasite effect (Paine, 1966).

Symbiotic relationships are also possible. The ancestor was manually dissected into two creatures, one of size 46 which contained only the code for self-examination and the copy loop, and one of size 64 which contained only the code for self-examination and the copy procedure (Figure 3). Neither could replicate when cultured alone, but when cultured together, they both replicated, forming a stable mutualistic relationship. It is not known if such relationships have evolved spontaneously.

DISCUSSION

The “physical” environment presented by the simulator is quite simple, consisting of the energy resource (CPU time) doled out rather uniformly by the time slicer, and memory space which is completely uniform and always available. In light of the nature of the physical environment, the implicit fitness function would presumably favor the evolution of creatures which are able to replicate with less CPU time, and this does in fact occur. However, much of the evolution in the system consists of the creatures discovering ways to exploit one-another. The creatures invent their own fitness functions through adaptation to their biotic environment.

Parasites do not contain the complete code for self-replication, thus they utilize other creatures for the information contained in their genomes. Hyper-parasites exploit parasites in order to increase the amount of CPU time devoted to the replication of their own genomes, thus hyper-parasites utilize other creatures for the energy resources that they possess. These ecological interactions are not programmed into the system, but emerge spontaneously as the creatures discover each other and invent their own games.

Evolutionary theory suggests that adaptation to the biotic environment (other organisms) rather than to the physical environment is the primary force driving the auto-catalytic diversification of organisms (Stanley, 1973). It is encouraging to discover that the process has already begun in the Tierran world. It is worth noting that the results presented here are based on evolution of the first creature that I designed, written in the first instruction set that I designed. Comparison to the creatures that have evolved shows that the one I designed is not a particularly clever one. Also, the instruction set that the creatures are based on is certainly not very powerful (apart from those special features incorporated to enhance its evolvability). It would appear then that it is rather easy to create life. Evidently, virtual life is out there, waiting for us to provide environments in which it may evolve.

Emergence

Cariani (1991) has suggested a methodology by which emergence can be detected. His analysis is described as “emergence-relative-to-a-model”, where “the model... constitutes the observer’s expectations of how the system will behave in the future.” If the system evolves such that the model no longer describes the system, we have emergence.

Cariani recognizes three types of emergence, in semiotic terms: syntactic, semantic and pragmatic. Syntactic operations are those of computation (symbolic). Semantic operations are those of measurement (e.g., sense perception) and control (e.g., effectors), because they “determine the relation of the symbols in the computational part of the device to the world at large”. Pragmatic (“intentional”) operations are those that are “performance-measuring”, and hence “the criteria which control the selection”.

Cariani has developed this analysis in the context of robotics, and considers that the semantic operations should act at the interface between the symbolic (computational) and the nonsymbolic (real physical world). I can not apply his analysis in precisely this way to my simulation, because there is no connection between the Tierran world and the real physical

world. I have created a virtual universe that is fully self-contained within the computer, thus I must apply his analysis in this context.

In the Tierran world, symbolic operations (syntactic), computations, take place in the CPU. The “nonsymbolic”, “real physical world” is the soup (RAM) where the creatures reside. The measurement (semantic) operations are those that involve the location of templates; the effector operations are the copying of instructions within the soup, and the allocation of memory (cells). Fitness functions (pragmatic) are implicit, and are determined by the creatures themselves because they must effect their own replication.

Any program which is self-modifying can show syntactic emergence. As long as the organization of the executable code changes, we have syntactic emergence. This occurs in the Tierran world, as the executable genetic code of the creatures evolves.

Semantic emergence is more difficult to achieve, as it requires the appearance of some new meaning in the system. This is found in the Tierran world in the evolution of templates and their meanings. When a creature locates a template, which has a physical manifestation in the “real world” of the soup, the location of the template appears in the CPU in the form of a symbol representing its address in the soup. For example, the beginning and end of the ancestor are each marked by templates. That one “means” beginning and the other “means” end is apparent from the computation made on the symbols for them in the CPU: the two are subtracted to calculate the size of the creature, and copying of the genome starts at the beginning address. Through evolution, a class of creatures appeared which did not locate a template at their end, but rather one in their center. That the new template “means” center to these creatures is again apparent from the computations made on its associated symbol in the CPU: the beginning address is subtracted from the center address, the difference is then multiplied by two to calculate the size.

Pragmatic emergence is considered “higher” by Cariani, and certainly it is the most difficult to achieve, because it requires that the system evolve new fitness functions. In living systems, fitness functions always reduce to: genotypes which leave a greater number of their genes in future generations will increase in frequency relative to other genotypes and thus have a higher fitness. This is a nearly tautological observation, but tautology is avoided in that the fitness landscape is shaped by specific adaptations that facilitate the passing on of genes.

For a precambrian marine algae living before the appearance of herbivores, the fitness landscape consists in part, of a multi-dimensional space of metabolic parameter affecting the efficiency of the conversion of sun light into useable energy, and the use of that energy in obtaining nutrients and converting them into new cells. Regions of this metabolic phase space that yield a greater efficiency at these operations also have higher associated fitnesses.

In order for pragmatic emergence to occur, the fitness landscape must be expanded to include new realms. For example, if a variant genotype of algae engulfs other algae, and thereby achieves a new mechanism of obtaining energy, the fitness landscape expands to include the parameters of structure and metabolism that facilitate the location, capture and digestion of other cells. The fitness landscapes of algae lacking these adaptations will become altered when they develop defense mechanisms, as they will then include the parameters of

mechanisms to avoid being eaten. Pragmatic emergence occurs through the acquisition of a new class of adaptation for enhancing the passing on of genes.

Pragmatic emergence occurs in the Tierran world as creatures which initially do not interact, discover means to exploit one another, and in response, means to avoid exploitation. The original fitness landscape of the ancestor consists only of the efficiency parameters of the replication algorithm, in the context of the properties of the reaper and slicer queues. When by chance, genotypes appear that exploit other creatures, selection acts to perfect the mechanisms of exploitation, and mechanisms of defense to that exploitation. The original fitness landscape was based only on adaptations of the organism to its physical environment (the slicer and reaper). The new fitness landscape retains those features, but adds to it adaptations to the biotic environment, the other creatures. Because the fitness landscape includes an ever increasing realm of adaptations to other creatures which are themselves evolving, it can facilitate an auto-catalytic increase in complexity and diversity of organisms.

In any computer model of evolution, the fitness functions are determined by the entity responsible for the replication of individuals. In genetic algorithms and most simulations, that entity is the simulator program, thus the fitness function is defined globally. In the Tierran world, that entity is the creatures themselves, thus the fitness function is defined locally by each creature in relation to its environment (which includes the other creatures). It is for this reason that pragmatic emergence occurs in the Tierran world.

In Tierra, the fitness functions are determined by the creatures themselves, and evolve with the creatures. As Cariani states “Such devices would not be useful for accomplishing *our* purposes as their evaluatory criteria might well diverge from our own over time”. This was the case from the outset in the Tierran world, because the simulator never imposed any explicit selection on the creatures. They were not expected to solve my problems, other than satisfying my passion to create life. Pragmatic emergence can only occur when we give up control of the destiny of the system.

After describing how to recognize the various types of emergence, Cariani concludes that Artificial Life can not demonstrate emergence because of the fully deterministic and replicable nature of computer simulations. This conclusion does not follow in any obvious way from the preceding discussions and does not seem to be supported. Furthermore, I have never known “indeterminate” and “unreplicable” to be considered as necessary qualities of life.

As a thought experiment, suppose that we connect a Geiger counter near a radioactive source to our computer, and use the interval between clicks to determine the values in our random number generator. The resulting behavior of the simulation would no longer be deterministic or repeatable. However, the results would be the same in any significant respect, to those obtained by using an algorithm to select the random numbers. Determinism and repeatability are irrelevant to emergence and to life. In fact repeatability is a highly desirable quality of synthetic life because it facilitates study of life’s properties.

Synthetic Biology

One of the most uncanny of evolutionary phenomena is the ecological convergence of

biota living on different continents or in different epochs. When a lineage of organisms undergoes an adaptive radiation (diversification), it leads to an array of relatively stable ecological forms. The specific ecological forms are often recognizable from lineage to lineage. For example among dinosaurs, the *Pterosaur*, *Triceratops*, *Tyrannosaurus* and *Ichthyosaurus* are ecological parallels respectively, to the bat, rhinoceros, lion and porpoise of modern mammals. Similarly, among modern placental mammals, the gray wolf, flying squirrel, great anteater and common mole are ecological parallels respectively, to the Tasmanian wolf, honey glider, banded anteater and marsupial mole of the marsupial mammals of Australia.

Given these evidently powerful convergent forces, it should perhaps not be surprising that as adaptive radiations proceed among digital organisms, we encounter recognizable ecological forms, in spite of the fundamentally distinct physics and chemistry on which they are based. Ideally, comparisons should be made among organisms of comparable complexity. It may not be appropriate to compare viruses to mammals. Unfortunately, the organic creatures most comparable to digital organisms, the RNA creatures, are no longer with us. Since digital organisms are being compared to modern organic creatures of much greater complexity, ecological comparisons must be made in the broadest of terms.

In describing the results, I have characterized classes of organisms such as hosts, parasites, hyper-parasites, social, and cheaters. While these terms apply nicely to digital organisms, it can be tricky to examine the parallels between digital and organic organisms in detail. The parasites of this study cause no direct harm to their host, however they do compete with them for space. This is rather like a vine which depends on a tree for support, but which does not directly harm the tree, except that the two must compete for light. The hyper-parasites of this study are facultative and subvert the energy metabolism of their parasite victims without killing them. I can not think of an organic example that has all of these properties. The carnivorous plant comes close in that it does not need the prey to survive, and in that its prey may have approached the plant expecting to feed on it. However, the prey of carnivorous plants are killed outright.

We are not in a position to make the most appropriate comparison, between digital creatures and RNA creatures. However, we can apply what we have learned from digital organisms, about the evolutionary properties of creatures at that level of complexity, to our speculations about what the RNA world may have been like. For example, once an RNA molecule fully capable of self-replication evolved, might other RNA molecules lacking that capability have parasitized its replicatory function?

In studying the natural history of synthetic organisms, it is important to recognize that they have a distinct biology due to their non-organic nature. In order to fully appreciate their biology, one must understand the stuff of which they are made. To study the biology of creatures of the RNA world would require an understanding of organic chemistry and the properties of macro-molecules. To understand the biology of digital organisms requires a knowledge of the properties of machine instructions and machine language algorithms. However, to fully understand digital organisms one must also have a knowledge of biological evolution and ecology. Evolution and ecology are the domain of biologists and machine languages are the domain of computer scientists. The knowledge chasm between biology and computer science is likely to hinder progress in the field of Artificial Life for some time. We

need more individuals with a depth of knowledge in both areas in order to carry out the work.

Trained biologists will tend to view synthetic life in the same terms that they have come to know organic life. Having been trained as an ecologist and evolutionist, I have seen in my synthetic communities, many of the ecological and evolutionary properties that are well known from natural communities. Biologists trained in other specialties will likely observe other familiar properties. It seems that what we see is what we know. It is likely to take longer before we appreciate the unique properties of these new life forms.

Artificial Life and Biological Theory

The relationship between Artificial Life and biological theory is two-fold: 1) Given that one of the main objectives of AL is to produce evolution leading to spontaneously increasing diversity and complexity, there exists a rich body of biological theory that suggests factors that may contribute to that process. 2) To the extent that the underlying life processes are the same in AL and organic life, AL models provide a new tool for experimental study of those processes, which can be used to test biological theory that can not be tested by traditional experimental and analytic techniques (discussed in Ray, unpublished).

Furthermore, there exists a complementary relationship between biological theory and the synthesis of life. Theory suggests how the synthesis can be achieved, while application of the theory in the synthesis is a test of the theory. If theory suggests that a certain factor will contribute to increasing diversity, then synthetic systems can be run with and without that factor. The process of synthesis becomes a test of the theory.

At the molecular level, there has been much discussion of the role of transposable elements in evolution. It has been observed that most of the genome in eukaryotes (perhaps 90%) originated from transposable elements, while in prokaryotes, only a very small percentage of the genome originated through transposons (Thomas, 1971; Orgel & Crick, 1980; Doolittle & Sapienza, 1980). It can also be noted that the eukaryotes, not the prokaryotes, were involved in the Cambrian explosion of diversity (Barbieri, 1985). It has been suggested that transposable elements play a significant role in facilitating evolution (Green, 1988; Jelinek & Schmid, 1982; Syvanen, 1984). These observations suggest that it would be an interesting experiment to introduce transposable elements into digital organisms.

The Cambrian explosion consisted of the origin, proliferation and diversification of macroscopic multi-cellular organisms. The origin and elaboration of multi-cellularity was an integral component of the process. Buss (1987) provides a provocative discussion of the evolution of multi-cellularity, and explores the consequences of selection at the level of cell lines. From his discussion the following idea emerges (although he does not explicitly state this idea, in fact he proposes a sort of inverse of this idea, p. 65): the transition from single to multi-celled existence involves the extension of the control of gene regulation by the mother cell to successively more generations of daughter cells. This is a concept which transcends the physical basis of life, and could be profitably applied to synthetic life in order to generate an analog of multi-cellularity.

The Red Queen hypothesis (Van Valen, 1973) suggests that in the face of a changing

environment, organisms must evolve as fast as they can in order to simply maintain their current state of adaptation. “In order to get anywhere you must run twice as fast as that” (Carroll, 1865). A critical component of the environment for any organism is the other living organisms with which it must interact. Given that the species that comprise the environment are themselves evolving, the pace is set by the maximal rate that any species may change through evolution, and it becomes very difficult to actually get ahead. A maximal rate of evolution is required just to keep from falling behind. This suggests that interactions with other evolving species provide the primary driving force in evolution.

Much evolutionary theory deals with the role of biotic interactions in driving evolution. For example, it is thought that these are of primary importance in the maintenance of sex (Maynard Smith, 1971; Charlesworth, 1976; Bell, 1982, 1989; Michod & Levin, 1988). Stanley (1973) has suggested that the Cambrian explosion was sparked by the appearance of the first organisms that ate other organisms. These new herbivores enhanced diversity by preventing any single species of algae from dominating and competitively excluding others. These kinds of biotic interactions must be incorporated into synthetic life in order to move evolution.

Similarly, many abiotic factors are known to contribute to determining the diversity of ecological communities. Island biogeography theory considers how the size, shape, distribution, fragmentation, and heterogeneity of habitats contribute to community diversity (MacArthur & Wilson, 1967). Various types of disturbance are also believed to significantly affect diversity (Huston, 1979; Petraitis et. al., 1989). All of these factors may be introduced into synthetic life in an effort to enhance the diversification of the evolving systems.

The examples just listed are a few of the many theories that suggest factors that influence biological diversity. In the process of synthesizing increasingly complex instances of life, we can incorporate and manipulate the states of these factors. These manipulations, conducted for the purposes of advancing the synthesis, will also constitute powerful tests of the theories.

Extending the Model

The approach to AL advocated in this work involves engineering over the first 3 billion years of life’s history to design complex evolvable artificial organisms, and attempting to create the biological conditions that will set off a spontaneous evolutionary process of increasing diversity and complexity of organisms. This is a very difficult undertaking, because in the midst of the Cambrian explosion, life had evolved to a level of complexity in which emergent properties existed at many hierarchical levels: molecular, cellular, organismal, populational and community.

In order to define an approach to the synthesis of life paralleling this historical stage of organic life, we must examine each of the fundamental hierarchical levels, abstract the principal biological properties from their physical representation, and determine how they can be represented in our artificial media. The simulator program determines not only the physics and chemistry of the virtual universe that it creates, but the community ecology as well. We must tinker with the structure of the simulator program in order to facilitate the existence of the appropriate “molecular”, “cellular”, and “ecological” interactions to generate a spontaneously increasing diversity and complexity.

The evolutionary potential of the present model can be greatly extended by some modifications. In its present implementation, parasitic relationships evolve rapidly, but predation involving the direct usurpation of space occupied by cells is not possible. This could be facilitated by the introduction of a FREE (memory de-allocation) instruction. However, it is unlikely that such predatory behavior would be selected for because in the current system there is always free memory space available, thus there would be little to be gained through seizing space from another creature. However, predation could be selected for by removing the reaper from the system.

Perhaps a more interesting way to favor predatory type interactions would be to make instructions expensive. In the present implementation, there is no “conservation of instructions”, because the MOV_IAB instruction creates a new copy of the instruction being moved during self-replication. If the MOV_IAB instruction were modified such that it obeyed a law of conservation, and left behind all zeros when it moved an instruction, then instructions would not be so cheap. Creatures could be allowed to synthesize instructions through a series of bit flipping and shifting operations, which would make instructions “metabolically” costly. Under such circumstance, a soup of “autotrophs” which synthesize all of their instructions could be invaded by a predatory creature which kills other creatures to obtain instructions.

Additional richness could be introduced to the model by modifying the way that CPU time is allocated. Rather than using a circular queue, creatures could deploy special arrays of instructions or bit patterns (analogous to chlorophyll) which capture potential CPU time packets raining like photons onto the soup. In addition, with instructions being synthesized through bit flipping and shifting operations, each instruction could be considered to have a “potential time” (i.e., potential energy) value which is proportional to its content of one bits. Instructions rich in ones could be used as time (energy) storage “molecules” which could be metabolized when needed by converting the one bits to zeros to release the stored CPU time. The introduction of such an “informational metabolism” would open the way for all sorts of evolution involving the exploitation of one organism by another.

Separation of the genotype from the phenotype would allow the model to move beyond the parallel to the RNA world into a parallel of the DNA-RNA-protein stage of evolution. Storage of the genetic information in relatively passive informational structures, which are then translated into the “metabolically active” machine instructions would facilitate evolution of development, sexuality and transposons. These features would contribute greatly to the evolutionary potential of the model.

These enhancements of the model represent the current directions of my continuing efforts in this area, in addition to using the existing model to further test ecological and evolutionary theory.

ACKNOWLEDGMENT

I thank Marc Cygnus, Robert Eisenberg, Doyne Farmer, Walter Fontana, Stephanie Forrest, Chris Langton, Dan Pirone, Stephen Pope, and Steen Rasmussen, for their discussions or readings of the manuscripts. Contribution No. 142 from the Ecology Program, School of Life and Health Sciences, University of Delaware.

REFERENCES

- Ackley, D. H. & Littman, M. S. "Learning from natural selection in an artificial environment." In: *Proceedings of the International Joint Conference on Neural Networks, Volume I, Theory Track, Neural and Cognitive Sciences Track*, IJCNN Winter 1990, Washington, DC. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1990.
- Aho, A. V., Hopcroft, J. E. & Ullman, J. D. *The design and analysis of computer algorithms*. Reading, Mass.: Addison-Wesley Publ. Co, 1974.
- Bagley, R. J., Farmer, J. D., Kauffman, S. A., Packard, N. H., Perelson, A. S. & Stadnyk, I. M. "Modeling adaptive biological systems." *Biosystems* **23** (1989): 113–138.
- Barbieri, M. *The semantic theory of evolution*. London: Harwood Academic Publishers, 1985.
- Bell, G. *The masterpiece of nature: the evolution and genetics of sexuality*. Berkeley: University of California Press, 1982.
- Bell, G. *Sex and death in protozoa: the history of an obsession*. Cambridge: University Press, 1989.
- Buss, L. W. *The evolution of individuality*. Princeton: University Press, 1987.
- Cariani, P. "Emergence and artificial life." In: *Artificial Life II*, edited by C. Langton, D. Farmer and S. Rasmussen. Redwood City, CA: Addison-Wesley, 1991, 000–000.
- Carroll, L. *Through the Looking-Glass*. London: MacMillan, 1865.
- Charlesworth, B. "Recombination modification in a fluctuating environment." *Genetics* **83** (1976): 181–195.
- Cohen, F. *Computer viruses: theory and experiments*. Ph. D. dissertation, U. of Southern California, 1984.
- Dawkins, R. *The blind watchmaker*. New York: W. W. Norton & Co., 1987.
- Dawkins, R. "The evolution of evolvability." In: *Artificial life: proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems*, edited by C. Langton. Redwood City, CA: Addison-Wesley, 1989, 201–220.
- Denning, P. J. "Computer viruses." *Amer. Sci.* **76** (1988): 236–238.
- Dewdney, A. K. "Computer recreations: In the game called Core War hostile programs engage in a battle of bits." *Sci. Amer.* **250** (1984): 14–22.
- Dewdney, A. K. "Computer recreations: A core war bestiary of viruses, worms and other threats to computer memories." *Sci. Amer.* **252** (1985a): 14–23.
- Dewdney, A. K. "Computer recreations: Exploring the field of genetic algorithms in a primordial computer sea full of flibs." *Sci. Amer.* **253** (1985b): 21–32.

- Dewdney, A. K. "Computer recreations: A program called MICE nibbles its way to victory at the first core war tournament." *Sci. Amer.* **256** (1987): 14–20.
- Dewdney, A. K. "Of worms, viruses and core war." *Sci. Amer.* **260** (1989): 110–113.
- Doolittle, W. F. & Sapienza, C. "Selfish genes, the phenotype paradigm and genome evolution." *Nature* **284** (1980): 601–603.
- Eldredge, N. & Gould, S. J. "Punctuated equilibria: an alternative to phyletic gradualism." In: *Models in Paleobiology*, edited by J. M. Schopf. San Francisco: Greeman, Cooper, 1972, 82–115.
- Farmer, J. D., Kauffman, S. A., & Packard, N. H. "Autocatalytic replication of polymers." *Physica D* **22** (1986): 50–67.
- Farmer, J. D. & Belin, A. "Artificial life: the coming evolution." Proceedings in celebration of Murray Gell-Man's 60th Birthday. Cambridge: University Press. In press. Reprinted in this volume.
- Gould, S. J. *Wonderful life, the Burgess shale and the nature of history*. New York: W. W. Norton & Company, 1989.
- Gould, S. J. & Eldredge, N. "Punctuated equilibria: the tempo and mode of evolution reconsidered." *Paleobiology* **3** (1977): 115–151.
- Green, M. M. "Mobile DNA elements and spontaneous gene mutation." In: *Eukaryotic transposable elements as mutagenic agents*, edited by Lambert, M. E., McDonald, J. F., & Weinstein, I. B. Banbury Report 30, Cold Spring Harbor Laboratory, 1988, 41–50.
- Holland, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor: Univ. of Michigan Press, 1975.
- Holland, J. H. "Studies of the spontaneous emergence of self-replicating systems using cellular automata and formal grammars." In: *Automata, Languages, Development*, edited by Lindenmayer, A., & Rozenberg, G. New York: North-Holland, 1976, 385–404.
- Huston, M. "A general hypothesis of species diversity." *Am. Nat.* **113** (1979): 81–101.
- Jelinek, W. R. & Schmid, C. W. "Repetitive sequences in eukaryotic DNA and their expression." *Ann. Rev. Biochem.* **51** (1982): 813–844.
- Langton, C. G. "Studying artificial life with cellular automata." *Physica* **22D** (1986): 120–149.
- Langton, C. G. "Virtual state machines in cellular automata." *Complex Systems* **1** (1987): 257–271.
- Langton, C. G. "Artificial life." In: *Artificial life: proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems*, edited by Langton, C. Vol. 6 in the series: Santa Fe Institute studies in the sciences of complexity. Redwood City, CA: Addison-Wesley, 1989, 1–47.

- Lotka, A. J. *Elements of physical biology*. Baltimore: Williams and Wilkins, 1925, reprinted as *Elements of mathematical biology*, Dover Press, 1956.
- Mac Arthur, R. H. & Wilson, E. O. *The theory of island biogeography*. Princeton: University Press, 1967.
- Maynard Smith, J. "What use is sex?" *J. Theoret. Biol.* **30** (1971): 319–335.
- Michod, R. E. & Levin, B. R. (eds). *The evolution of sex*. Sutherland, MA: Sinauer Associates Inc, 1988.
- Minsky, M. L. *Computation: finite and infinite machines*. Englewood Cliffs, N.J.: Prentice-Hall, 1976.
- Morris, S. C. "Burgess shale faunas and the cambrian explosion." *Science* **246** (1989): 339–346.
- Orgel, L. E. & Crick, F. H. C. "Selfish DNA: the ultimate parasite." *Nature* **284** (1980): 604–607.
- Paine, R. T. "Food web complexity and species diversity." *Am. Nat.* **100** (1966): 65–75.
- Packard, N. H. "Intrinsic adaptation in a simple model for evolution." In: *Artificial life: proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems*, edited by C. Langton. Redwood City, CA: Addison-Wesley, 1989, 141–155.
- Pattee, H. H. "Simulations, realizations, and theories of life." In: *Artificial life: proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems*, edited by C. Langton. Redwood City, CA: Addison-Wesley, 1989, 63–77.
- Petratits, P. S., Latham, R. E. & Niesenbaum, R. A. "The maintenance of species diversity by disturbance." *Quart. Rev. Biol.* **64** (1989): 393–418.
- Rasmussen, S., Knudsen, C., Feldberg, R. & Hindsholm, M. "The coreworld: emergence and evolution of cooperative structures in a computational chemistry" *Physica D* **42** (1990): 111–134.
- Rheingold, H. (1988). Computer viruses. *Whole Earth Review* **Fall** (1988): 106.
- Ray, T. S. "Synthetic Life: evolution and ecology of digital organisms." Unpublished, 1990.
- Spafford, E. H., Heaphy, K. A. & Ferbrache, D. J. *Computer viruses, dealing with electronic vandalism and programmed threats*. ADAPSO, 1300 N. 17th Street, Suite 300, Arlington, VA 22209, 1989.
- Stanley, S. M. "An ecological theory for the sudden origin of multicellular life in the late precambrian." *Proc. Nat. Acad. Sci.* **70** (1973): 1486–1489.
- Syvanen, M. "The evolutionary implications of mobile genetic elements." *Ann. Rev. Genet.* **18** (1984): 271–293.
- Thomas, C. A. "The genetic organization of chromosomes." *Ann. Rev. Genet.* **5** (1971): 237–256.

- Van Valen, L. "A new evolutionary law." *Evolutionary Theory* **1** (1973): 1–30.
- Volterra, V. "Variations and fluctuations of the number of individuals in animal species living together." In: *Animal Ecology*, edited by R. N. Chapman. New York: McGraw-Hill, 1926, 409–448.
- Wilson, E. O. & Bossert, W. H. *A primer of population biology*. Stamford, Conn: Sinauer Associates, 1971.

Figure 1. Metabolic flow chart for the ancestor, parasite, hyper-parasite, and their interactions: ax, bx and cx refer to CPU registers where location and size information are stored. [ax] and [bx] refer to locations in the soup indicated by the values in the ax and bx registers. Patterns such as 1101 are complementary templates used for addressing. Arrows outside of boxes indicate jumps in the flow of execution of the programs. The dotted-line arrows indicate flow of execution between creatures. The parasite lacks the copy procedure, however, if it is within the search limit of the copy procedure of a host, it can locate, call and execute that procedure, thereby obtaining the information needed to complete its replication. The host is not adversely affected by this informational parasitism, except through competition with the parasite, which is a superior competitor. Note that the parasite calls the copy procedure of its host with the expectation that control will return to the parasite when the copy procedure returns. However, the hyper-parasite jumps out of the copy procedure rather than returning, thereby seizing control from the parasite. It then proceeds to reset the CPU registers of the parasite with the location and size of the hyper-parasite, causing the parasite to replicate the hyper-parasite genome thereafter.

Figure 2. Metabolic flow chart for social hyper-parasites, their associated hyper-hyper-parasite cheaters, and their interactions. Symbols are as described for Fig. 1. Horizontal dashed lines indicate the boundaries between individual creatures. On both the left and right, above the dashed line at the top of the figure is the lowermost fragment of a social-hyper-parasite. Note (on the left) that neighboring social hyper-parasites cooperate in returning the flow of execution to the beginning of the creature for self-re-examination. Execution jumps back to the end of the creature above, but then falls off the end of the creature without executing any instructions of consequence, and enters the top of the creature below. On the right, a cheater is inserted between the two social-hyper-parasites. The cheater captures control of execution when it passes between the social individuals. It sets the CPU registers with its own location and size, and then skips over the self-examination step when it returns control of execution to the social creature below.

Figure 3. Metabolic flow chart for obligate symbionts and their interactions. Symbols are as described for Fig. 1. Neither creature is able to self-replicate in isolation. However, when cultured together, each is able to replicate by using information provided by the other.

Table 1: Genebank. Table of numbers of size classes in the genebank. Left column is size class, right column is number of self-replicating genotypes of that size class. 305 sizes, 29,275 genotypes.

0034 1	0092 362	0150 2	0205 5	0418 1	5213 2
0041 2	0093 261	0151 1	0207 3	0442 10	5229 4
0043 12	0094 241	0152 2	0208 2	0443 1	5254 1
0044 7	0095 211	0153 1	0209 1	0444 61	5888 36
0045 191	0096 232	0154 2	0210 9	0445 1	5988 1
0046 7	0097 173	0155 3	0211 4	0456 2	6006 2
0047 5	0098 92	0156 77	0212 4	0465 6	6014 1
0048 4	0099 117	0157 270	0213 5	0472 6	6330 1
0049 8	0100 77	0158 938	0214 47	0483 1	6529 1
0050 13	0101 62	0159 836	0218 1	0484 8	6640 1
0051 2	0102 62	0160 3229	0219 1	0485 3	6901 5
0052 11	0103 27	0161 1417	0220 2	0486 9	6971 1
0053 4	0104 25	0162 174	0223 3	0487 2	7158 2
0054 2	0105 28	0163 187	0226 2	0493 2	7293 3
0055 2	0106 19	0164 46	0227 7	0511 2	7331 1
0056 4	0107 3	0165 183	0231 1	0513 1	7422 70
0057 1	0108 8	0166 81	0232 1	0519 1	7458 1
0058 8	0109 2	0167 71	0236 1	0522 6	7460 7
0059 8	0110 8	0168 9	0238 1	0553 1	7488 1
0060 3	0111 71	0169 15	0240 3	0568 6	7598 1
0061 1	0112 19	0170 99	0241 1	0578 1	7627 63
0062 2	0113 10	0171 40	0242 1	0581 3	7695 1
0063 2	0114 3	0172 44	0250 1	0582 1	7733 1
0064 1	0115 3	0173 34	0251 1	0600 1	7768 2
0065 4	0116 5	0174 15	0260 2	0683 1	7860 25
0066 1	0117 3	0175 22	0261 1	0689 1	7912 1
0067 1	0118 1	0176 137	0265 2	0757 6	8082 3
0068 2	0119 3	0177 13	0268 1	0804 2	8340 1
0069 1	0120 2	0178 3	0269 1	0813 1	8366 1
0070 7	0121 60	0179 1	0284 16	0881 6	8405 5
0071 5	0122 9	0180 16	0306 1	0888 1	8406 2
0072 17	0123 3	0181 5	0312 1	0940 2	8649 2
0073 2	0124 11	0182 27	0314 1	1006 6	8750 1
0074 80	0125 6	0184 3	0316 2	1016 1	8951 1
0075 56	0126 11	0185 21	0318 3	1077 5	8978 3
0076 21	0127 1	0186 9	0319 2	1116 1	9011 3
0077 28	0130 3	0187 3	0320 23	1186 1	9507 3
0078 409	0131 2	0188 11	0321 5	1294 7	9564 3
0079 850	0132 5	0190 20	0322 21	1322 7	9612 1
0080 7399	0133 2	0192 12	0330 1	1335 1	9968 1
0081 590	0134 7	0193 4	0342 5	1365 11	10259 31

0082 384	0135 1	0194 4	0343 1	1631 1	10676 1
0083 886	0136 1	0195 11	0351 1	1645 3	11366 5
0084 1672	0137 1	0196 19	0352 3	2266 1	11900 1
0085 1531	0138 1	0197 2	0386 1	2615 2	12212 2
0086 901	0139 2	0198 3	0388 2	2617 9	15717 3
0087 944	0141 6	0199 35	0401 3	2671 7	16355 1
0088 517	0143 1	0200 1	0407 1	3069 3	17356 3
0089 449	0144 4	0201 84	0411 22	4241 1	18532 1
0090 543	0146 1	0203 1	0412 3	5101 15	23134 14
0091 354	0149 1	0204 1	0416 1	5157 9	

APPENDIX A

Structure definition to implement the Tierra virtual CPU. The source code or executables for the Tierra Simulator can be obtained by contacting the author by mail, either email or snail mail.

```
struct cpu { /* structure for registers of virtual cpu */
    int  ax; /* address register */
    int  bx; /* address register */
    int  cx; /* numerical register */
    int  dx; /* numerical register */
    char fl; /* flag */
    char sp; /* stack pointer */
    int  st[10]; /* stack */
    int  ip; /* instruction pointer */
} ;
```

APPENDIX B

Abbreviated code for implementing the CPU cycle of the Tierra Simulator.

```
void main(void)
{   get_soup();
    life();
    write_soup();
}

void life(void) /* doles out time slices and death */
{   while(inst_exec_c < alive) /* control the length of the run */
    {   time_slice(this_slice); /* this_slice is current cell in queue */
        incr_slice_queue(); /* increment this_slice to next cell in queue */
        while(free_mem_current < free_mem_prop * soup_size)
            reaper(); /* if memory is full to threshold, reap some cells */
    }
}

void time_slice(int  ci)
{   Pcells  ce; /* pointer to the array of cell structures */
    char    i; /* instruction from soup */
    int     di; /* decoded instruction */
    int     j, size_slice;
    ce = cells + ci;
    for(j = 0; j < size_slice; j++)
    {   i = fetch(ce->c.ip); /* fetch instruction from soup, at address ip */
        di = decode(i); /* decode the fetched instruction */
    }
```

```

        execute(di, ci);      /* execute the decoded instruction */
        increment_ip(di,ci); /* move instruction pointer to next instruction */
        system_work(); /* opportunity to extract information */
    }
}

void execute(int di, int ci)
{
    switch(di)
    {
        case 0x00: nop_0(ci);      break; /* no operation */
        case 0x01: nop_1(ci);      break; /* no operation */
        case 0x02: orl(ci);         break; /* flip low order bit of cx, cx ^= 1 */
        case 0x03: shl(ci);         break; /* shift left cx register, cx <= 1 */
        case 0x04: zero(ci);        break; /* set cx register to zero, cx = 0 */
        case 0x05: if_cz(ci);       break; /* if cx==0 execute next instruction */
        case 0x06: sub_ab(ci);       break; /* subtract bx from ax, cx = ax - bx */
        case 0x07: sub_ac(ci);       break; /* subtract cx from ax, ax = ax - cx */
        case 0x08: inc_a(ci);        break; /* increment ax, ax = ax + 1 */
        case 0x09: inc_b(ci);        break; /* increment bx, bx = bx + 1 */
        case 0x0a: dec_c(ci);        break; /* decrement cx, cx = cx - 1 */
        case 0x0b: inc_c(ci);        break; /* increment cx, cx = cx + 1 */
        case 0x0c: push_ax(ci);      break; /* push ax on stack */
        case 0x0d: push_bx(ci);      break; /* push bx on stack */
        case 0x0e: push_cx(ci);      break; /* push cx on stack */
        case 0x0f: push_dx(ci);      break; /* push dx on stack */
        case 0x10: pop_ax(ci);       break; /* pop top of stack into ax */
        case 0x11: pop_bx(ci);       break; /* pop top of stack into bx */
        case 0x12: pop_cx(ci);       break; /* pop top of stack into cx */
        case 0x13: pop_dx(ci);       break; /* pop top of stack into dx */
        case 0x14: jmp(ci);          break; /* move ip to template */
        case 0x15: jmpb(ci);         break; /* move ip backward to template */
        case 0x16: call(ci);         break; /* call a procedure */
        case 0x17: ret(ci);          break; /* return from a procedure */
        case 0x18: mov_cd(ci);       break; /* move cx to dx, dx = cx */
        case 0x19: mov_ab(ci);       break; /* move ax to bx, bx = ax */
        case 0x1a: mov_iab(ci);      break; /* move instruction at address in bx
                                           to address in ax */
        case 0x1b: adr(ci);          break; /* address of nearest template to ax */
        case 0x1c: adrb(ci);         break; /* search backward for template */
        case 0x1d: adrf(ci);         break; /* search forward for template */
        case 0x1e: mal(ci);          break; /* allocate memory for daughter cell */
        case 0x1f: divide(ci);       break; /* cell division */
    }
    inst_exec_c++;
}

```

APPENDIX C

Assembler source code for the ancestral creature.

```
genotype: 80 aaa  origin: 1-1-1990  00:00:00:00  ancestor
parent genotype: human
1st_daughter:  flags: 0  inst: 839  mov_daught: 80
2nd_daughter:  flags: 0  inst: 813  mov_daught: 80

nop_1    ; 01    0 beginning template
nop_1    ; 01    1 beginning template
nop_1    ; 01    2 beginning template
nop_1    ; 01    3 beginning template
zero     ; 04    4 put zero in cx
or1      ; 02    5 put 1 in first bit of cx
shl      ; 03    6 shift left cx
shl      ; 03    7 shift left cx, now cx = 4
          ;      ax =                bx =
          ;      cx = template size  dx =
mov_cd   ; 18    8 move template size to dx
          ;      ax =                bx =
          ;      cx = template size  dx = template size
adrb     ; 1c    9 get (backward) address of beginning template
nop_0    ; 00   10 compliment to beginning template
nop_0    ; 00   11 compliment to beginning template
nop_0    ; 00   12 compliment to beginning template
nop_0    ; 00   13 compliment to beginning template
          ;      ax = start of mother + 4  bx =
          ;      cx = template size        dx = template size
sub_ac   ; 07   14 subtract cx from ax
          ;      ax = start of mother  bx =
          ;      cx = template size    dx = template size
mov_ab   ; 19   15 move start address to bx
          ;      ax = start of mother  bx = start of mother
          ;      cx = template size    dx = template size
adrf     ; 1d   16 get (forward) address of end template
nop_0    ; 00   17 compliment to end template
nop_0    ; 00   18 compliment to end template
nop_0    ; 00   19 compliment to end template
nop_1    ; 01   20 compliment to end template
          ;      ax = end of mother  bx = start of mother
          ;      cx = template size  dx = template size
inc_a    ; 08   21 to include dummy statement to separate creatures
sub_ab   ; 06   22 subtract start address from end address to get size
          ;      ax = end of mother  bx = start of mother
```

```

;      cx = size of mother   dx = template size
nop_1   ; 01 23 reproduction loop template
nop_1   ; 01 24 reproduction loop template
nop_0   ; 00 25 reproduction loop template
nop_1   ; 01 26 reproduction loop template
mal     ; 1e 27 allocate memory for daughter cell, address to ax
;      ax = start of daughter   bx = start of mother
;      cx = size of mother      dx = template size
call    ; 16 28 call template below (copy procedure)
nop_0   ; 00 29 copy procedure compliment
nop_0   ; 00 30 copy procedure compliment
nop_1   ; 01 31 copy procedure compliment
nop_1   ; 01 32 copy procedure compliment
divide  ; 1f 33 create independent daughter cell
jmp     ; 14 34 jump to template below (reproduction loop, above)
nop_0   ; 00 35 reproduction loop compliment
nop_0   ; 00 36 reproduction loop compliment
nop_1   ; 01 37 reproduction loop compliment
nop_0   ; 00 38 reproduction loop compliment
if_cz   ; 05 39 this is a dummy instruction to separate templates
;      begin copy procedure
nop_1   ; 01 40 copy procedure template
nop_1   ; 01 41 copy procedure template
nop_0   ; 00 42 copy procedure template
nop_0   ; 00 43 copy procedure template
push_ax ; 0c 44 push ax onto stack
push_bx ; 0d 45 push bx onto stack
push_cx ; 0e 46 push cx onto stack
nop_1   ; 01 47 copy loop template
nop_0   ; 00 48 copy loop template
nop_1   ; 01 49 copy loop template
nop_0   ; 00 50 copy loop template
mov_iab ; 1a 51 move contents of [bx] to [ax]
dec_c   ; 0a 52 decrement cx
if_cz   ; 05 53 if cx == 0 perform next instruction, otherwise skip it
jmp     ; 14 54 jump to template below (copy procedure exit)
nop_0   ; 00 55 copy procedure exit compliment
nop_1   ; 01 56 copy procedure exit compliment
nop_0   ; 00 57 copy procedure exit compliment
nop_0   ; 00 58 copy procedure exit compliment
inc_a   ; 08 59 increment ax
inc_b   ; 09 60 increment bx
jmp     ; 14 61 jump to template below (copy loop)
nop_0   ; 00 62 copy loop compliment
nop_1   ; 01 63 copy loop compliment

```

```

nop_0      ; 00 64 copy loop compliment
nop_1      ; 01 65 copy loop compliment
if_cz      ; 05 66 this is a dummy instruction, to separate templates
nop_1      ; 01 67 copy procedure exit template
nop_0      ; 00 68 copy procedure exit template
nop_1      ; 01 69 copy procedure exit template
nop_1      ; 01 70 copy procedure exit template
pop_cx     ; 12 71 pop cx off stack
pop_bx     ; 11 72 pop bx off stack
pop_ax     ; 10 73 pop ax off stack
ret        ; 17 74 return from copy procedure
nop_1      ; 01 75 end template
nop_1      ; 01 76 end template
nop_1      ; 01 77 end template
nop_0      ; 00 78 end template
if_cz      ; 05 79 dummy statement to separate creatures

```