On the Incorporation of a Developmental Process in a System of Self-Replicating Programs

DAI WORKING PAPER No. 258

Tim Taylor
(E-mail: timt@aifh.ed.ac.uk)

Department of Artificial Intelligence
University of Edinburgh
January 1996
(with minor corrections June 1996)

Abstract

This paper describes work I intend to pursue over the course of my PhD (although it presents a discussion of general ideas rather than a detailed research proposal—that will come later). My primary research interest is in the study of evolutionary processes through the use of computational models. In particular, I intend to study evolution in systems of self-replicating programs, especially those in which there is a developmental process to map genotypes to phenotypes. The reasons for this choice, and discussion of the analogy I wish to draw between the proposed digital system and existing biological systems, are explained. I believe that this work will be of interest both from the perspective of making some contribution to our understanding of biological evolution and also from the perspective of the evolution of computer programs in their own right.

Contents

Chapter 1

Introduction

The three characteristics that determine the long-term survival[1] of a self-replicating entity in competition with other self-replicators in an environment with limited resources, are:

- longevity
- fecundity
- copying fidelity

(from [Dawkins 76]). This is just formalizing the obvious; that those entities which exist for a long time, produce many copies of themselves, and can produce these copies with few mistakes (all *relative to the other entities in the competition*) will be the ones whose numbers grow in successive generations and who will come to dominate the system.

If the process of creating a new individual by copying an existing one is not *completely* reliable—if there is always a chance that a mistake will be made and that the copy will be slightly imperfect, then evolution of the self-replicators will occur; there is a slim chance that a new slightly imperfect copy of a self-replicating entity thus formed may actually be slightly *better* than its parent in an evolutionary sense[2].

[1] That is, the survival of a chain of descendants throughout many generations.

[2] That is, it may have greater longevity, fecundity or copying-fidelity.

According to neo-Darwinist theory[3], biological evolution has proceeded from the very first self-replicating molecules to produce all forms of life we see today, solely by the principle of the differential survival of slightly imperfect self-replicators which are competing for a limited resource of molecules necessary for their replication.

Although many people tacitly accept the neo-Darwinist theory, I suspect that a large number of them (myself included) find it hard to imagine that such a simple process is responsible for the vast spectrum of extraordinarily complicated and well-adapted biological species alive today. This is not to say that I think the theory is wrong, merely that I have to admit some limit to my powers of imagination.

One reason for this shortcoming may simply be the sheer scale of the numbers involved—billions of self-replicating entities competing and evolving in parallel over the course of billions of years. There is also the fact that evolution is an incremental process, that over this huge number of individuals and years any change in the self-replicating entities that has been retained, has, by definition, led to an improvement in the comparative performance of these entities. As John von Neumann noted:

> "living organisms are very complicated aggregations of elementary parts, and by any reasonable theory of probability or thermodynamics, highly improbable...[however] if by any peculiar accident there should ever be one of them, from there on the rules of probability do not apply, and there will be many of them, at least if the milieu is reasonable" *quoted from* [Koza 94]

However, even if we can accept this account, there are still many features of observed biological species that are not easily explicable. For example, nearly all modern self-replicators are contained within cells, which are extremely complicated survival machines. On top of this, many species of self-replicators also go through a remarkably time- and energy-consuming process of development and differentiation into multicellular organisms. In human beings, for example, a single zygote divides and differentiates into approximately 10^{15} cells of over 200 different types. In humans, as in many other species, this process of development takes decades before the organism reaches adulthood and is finally able to achieve its goal of replicating its genome.

[3] Neo-Darwinism is a synthesis of Darwin's theory of natural selection and modern population genetics.

[Dawkins 82] has suggested some compelling reasons why developmental processes may be beneficial, and why organisms which adopt a growth/reproduction/death life cycle, where each newly-created individual is forced through a single-celled 'bottleneck' may be better equipped for evolving adaptive complexity.

Following Dawkins' line of reasoning, it is certainly easy to see that a single mutation affecting the process of development of an individual organism has, in principle, the scope to alter anything from, at one extreme, a small detail of the adult phenotype to, at the other extreme, the entire organism, depending on whether the mutation affects the late or early stages (respectively) of development. It is also not too hard to imagine that a mutation acting upon a developing organism has a fair chance of producing a change of phenotype that in some way 'makes sense' to the design of the organism as a whole (e.g. a mutation in the genome of a giraffe, say, may cause an extra vertebra to be produced in the neck). These two mechanisms both produce a *leveraging* effect on the power of a single mutation, creating an environment where *macromutations*, and not only this, but *sensible* macromutations, will be relatively common.

Dawkins' argument is that development can produce complex organs and behaviour patterns, and complex organs and behaviour patterns are favoured in arms races against other species. However, he warns that:

> "We must beware of the heresy of 'biotic' adaptationism ([Williams 66]). We have seen that recurrent reproduction life cycles, i.e. 'organisms', make the evolution of complex organs possible. It is all too easy to treat this as a sufficient explanation for the *existence* of organismal life cycles, on the grounds that complex organs are, in some vague sense, a good idea... The Darwinian must begin by seeking immediate benefits to genes promoting this kind of life cycle, at the expense of their alleles." (*pp. 262-3*)
>
> "A complex developmental sequence has to have evolved from an earlier developmental sequence which was slightly less complex." (*p. 258*)

It is my intention to study the evolution of self-replicating entities in a digital medium, in which the "genome" of these entities is not a direct representation for a computer program which can reproduce itself, but rather a description of a developmental sys-

tem by which a self-replicating computer program may be *constructed* using a *recipe* contained on the genome. It is hoped that the results of such research may be of interest to evolutionary biologists, both in suggesting advantages of organismal over non-organismal life cycles, and in suggesting ways in which complex developmental sequences may have evolved cumulatively from simpler sequences. Additionally, it is anticipated that the results will be useful from the point of view of studying the evolution of computer programs in their own right, particularly through the comparison of performance of such developmental systems to more traditional genetic programming and self-replicating approaches.

Chapter 2

An Overview of Related Work

Within the last few years a number of researchers have begun to look at some closely related issues. Below I present a summary of what I believe to be the most important results.

The research described in this chapter divides into three categories. One distinct category is work relating to the study of self-replicating computer code[1]. The other two categories are fairly closely related to each other; one is the study of developmental systems on computers, and the other is the study of self-organization.

2.1 Research with Self-Replicators

2.1.1 Description of Existing Systems

The motivation underlying most of the work with self-replicating code is to create a digital analogy to the very early stages of biological evolution. Probably the best known work with self-replicating computer code has been produced by Tom Ray and colleagues with their "Tierra" system (see [Ray 91]). Similar systems have been described by Chris Adami and Titus Brown (the "Avida" system [Adami & Brown 94]), and by Jakob Skipper (the "Computer Zoo" [Skipper 92]).

[1] By "self-replicating" I mean code which, when executed, will cause a copy of itself to be created at some other position in the computer's memory. This is in contrast to, for example, a "chromosome" used in a standard genetic algorithm, which generally does not represent a computer program and cannot reproduce itself, instead having to rely on the algorithmic engine to copy it if it is deemed to have sufficient 'fitness'. By the same token, I do not regard cellular automata as self-replicating either.

The basic idea of these systems is that a virtual operating system is provided, complete with a simple machine language. The machine language instructions are designed to be robust under the mutation and evolution of programs written in the language. This robustness is achieved chiefly through the use of relative or template-driven[2] addressing in branching instructions (rather than absolute addressing), and by avoiding the use of explicit memory addresses as operands to instructions. In such an environment, the percentage of functional programs in the space of all possible programs is greatly increased, and the chances of mutating a functional program and coming up with another program that also works are much higher.

Although the details of these systems differ somewhat[3], the general mode of operation is the same. An "evolutionary run" commences with the introduction of an *ancestor* program into the otherwise empty memory. The ancestor program has been hand-written by the researcher to produce another copy of itself in the computer's memory when the program is run. A small element of stochastic behaviour is usually associated with the execution of the machine instructions, e.g. an ADD instruction which usually adds one to its operand may occasionally add zero or two instead, or a COPY instruction may sometimes mutate a byte of the data it is copying. Alternatively, the programs may be subject to point random mutations at a given low rate. In either of these ways, as a run proceeds variations of the ancestor program begin to appear. If a variation retains the ability to produce a copy of itself, then it too may be retained in the population of programs over a large number of generations. As the available memory begins to fill, some sort a reaper operation is performed to kill off some of the programs.

A number of interesting results have been obtained from such evolutionary runs, for example the appearance of "parasites"—short pieces of code which run another program's copying procedure in order to copy themselves. It may be true to say that most of the interesting behaviour that has been seen to evolve has done so because facilities

[2] With template-driven branching, a branching instruction is followed by a template (a sequence of bits). The operating system will search for the nearest matching template in the rest of the code and move the instruction pointer to that position.

[3] Programs in Avida and Computer Zoo also have the notion of a location in a 2D space (in Computer Zoo the programs can actually move around this space). This notion was introduced mainly to provide a form of genetic isolation, and also to facilitate efficient implementation on parallel hardware.

for these behaviours were engineered into the original language specification. For example, the fact that parasites emerge is a little less surprising when the mechanisms of template-driven branching are considered. However, what all of these studies have demonstrated is that it *is* possible to build a robust operating system in which non-trivial self-replicating computer code can evolve. It will be of great interest to follow the development of these systems, particularly Tierra and Avida, whose creators plan to conduct further research on more powerful parallel hardware.

Kurt Theraling and Ray have also published a paper entitled "Evolving Multi-Cellular Artificial Life" in which they describe an extension to the Tierra system to allow programs to run parallel processes ([Thearling & Ray 94]). Although interesting in its own right and for the development of parallel computer code, these programs are multicellular in a different sense to that described in Chapter 1; they are capable of spawning new parallel processes during runtime, but the process of self-replication still involves copying each instruction in the program byte-by-byte to a new memory location. It is the fact that what is being copied is still a high level description of the phenotype[4], rather than a much lower level *recipe* for producing the phenotype that makes this arrangement less likely to be successful in evolving complex adaptations, according to the current thesis (see Section 3.1).

John Koza has proposed a system of self-replicating programs using a slightly different approach ([Koza 94]). Koza's programs are boolean trees whose nodes may be standard logical functions (i.e. AND, OR, NOT) or a number of extra functions which not only produce a boolean result but also have side effects which act upon the program itself or upon other programs in the computer's memory. For example, there is an ADD function which will search the rest of the memory for a program matching a given template, and, if one is found, will have the side effect of substituting this new program in the position where the ADD instruction resided in the original program.

One result of this approach of using boolean functions with side effects is that a program may perform a boolean calculation as well as having the side effect of producing another copy of itself somewhere else in memory. Koza encouraged his programs to perform specific boolean calculations by imposing a fitness function on the system—a program

[4] The phenotype being the *behaviour* of the program, or even just the program itself.

has a good chance of proceeding to the next generation only if it is able to self-replicate, *and* performs well on the target boolean function[5]. In this respect, Koza's system is very similar to Avida, where programs can gain more CPU time by successfully completing user-specified tasks.

While it is perfectly possible for a program on, say, the Tierra system to perform other functions *as well as* reproducing itself, this hasn't been observed yet, presumably because there is no obvious way in which a program's fitness may be increased by doing tasks which do not affect its ability to reproduce[6].

In this important sense, Avida and Koza's system differ from Tierra and Computer Zoo; the latter pair of systems allow programs which perform complicated functions which may not be directly related to self-reproduction, but *only* if, by however roundabout a route, such functions in some way increase that program's longevity, fecundity or copying-fidelity relative to the other programs in the system. In contrast, in Koza's system and in Avida, there is actually a *specified* function (or set of functions) which the programs must perform *as well as* having to be able to reproduce. In this way, there are effectively two fitness measures, the former being completely unrelated to a program's ability to self-replicate.

The reasons for Koza's and Adami et al.'s decisions to build such systems are not stated in their papers, but I suspect they stemmed from a desire to incorporate the potential evolutionary power of self-replicating entities within a framework where the evolving programs were actually performing a task specified by the researcher so that they were doing something *useful* as well as reproducing. This may well be a promising line of research, but it should be remembered that it is without an analogy in neo-Darwinist theory, according to which an entity's fitness is only related to its powers of self-replication[7]. The subject of adding extra functionality to self-replicating programs

[5] The set of primitive computational entities from which programs may be constructed comprised not only eight functions (some of which have been described), but also three terminals, corresponding to three boolean input values. In any given evolutionary run, all the programs are tested for their performance on a specific three-argument boolean function—the *target* function.

[6] In fact, we might expect any such 'additional' functionality to actually *decrease* the program's fitness, as it would have the effect of wasting CPU time which could otherwise have been utilised by the program in producing more copies of itself.

[7] Note, however, that there is an analogy here with the attempts of genetic engineers to implant recombinant DNA into living organisms to farm useful proteins such as insulin. These results

is an important issue, and I shall return to it in Section 3.3.

In addition to the systems already described, this section would be incomplete without mention of the ideas of Douglas Hofstadter, some ten years ahead of Tierra ([Hofstadter 79]). Hofstadter proposed a system called Typogenetics, in which he "tried to capture some ideas of molecular genetics in a typographical system". In this system, a strand of letters encoded a sequence of operations (such as copying, cutting etc.) which were to be performed upon the strand itself. The Typogenetics systems was later developed and analysed by Harold Morris in his PhD dissertation ([Morris 88]), in which he showed that it was possible to produce several types of strand which were capable of self-replication. If the interpretation and processing of strands is handled on a computer, then we have a system of self-replicating computer programs, albeit with a somewhat different representation and interpreting process to those used in the other systems described in this section.

In his original book, Hofstadter also discussed the issues of self-replication and self-reference in great detail (which we will return to in Chapter 3).

2.1.2 Similarities Between Self-Replicating Systems

It is of interest to compare the kinds of machine instructions that have been included in the languages from which each of the described self-replicating systems (except Typogenetics) are built. Such comparison reveals that the following features are common to all the languages:

1. Template facilities
2. A small number of registers (together with associated facilities)
3. A stack[8]

suggest that it is possible to employ self-replicating entities for human-specified tasks, at least in the short term. The interesting question here is whether the genetic engineers can successfully change the fitness landscape of the organisms in the long term (by killing those organisms which do not have the ability to produce the protein) in order to create a new breed of protein-producing organisms which is evolutionarily stable, at least in the environment of the laboratory.

[8] In Koza's system, there are two 'semi-stacks' called MATCHED-TREE and OPEN-SPACE. The former can contain a sequence of instructions which are all placed there at the same time, but which can be read off one by one using the LR instruction. The latter can be written to one instruction at a

4. Very few/no constant operands
5. Basic arithmetic/boolean operators
6. Conditional operator(s)
7. Memory allocation facility
8. Split operator (to signal that a newly written data should now be interpreted as a program in its own right)

(In Koza's system, the last two features of this list are implicit in the processor rather than being explicit instructions in the language.)

The types of operations used in Typogenetics were more biologically inspired, including the following:

1. Move read/write head
2. Find a template (a 'purine' or a 'pyrimidine')
3. Switch copy mode on/off
4. Insert a new unit
5. Delete a unit
6. Cut strand

There is a striking resemblance between this list and this list of operations performable by a Turing machine. This leads us on to the question of whether any or all of these systems are capable of universal computation—this is certainly a critical point, as only those systems which are computationally complete[9] are even *theoretically* possible of

time using the SR instruction, but only when the sequence of writing instructions is deemed to have stopped can the contents of OPEN-SPACE be acted upon, and action at this point will operate on the entire contents of OPEN-SPACE. (To my knowledge, the computational completeness of this system has not been proved.)

[9] In this paper the terms "capable of universal computation", "computationally universal", "computationally complete" and "Turing equivalent" are used interchangeably.

evolving programs to perform *any conceivable computational process*. The computational completeness of Tierra has, in fact, been proved by Carlo Maley [Maley 94].

Both of the lists above share the following properties:

- A small instruction set compared to traditional computer languages[10]
- *Robust* instructions (e.g. no explicit memory addressing)

We will return to thoughts of the kinds of property that suitable representations should include in Section 3.5.

2.1.3 To Self-Replicate Or Not to Self-Replicate?

On the difference between self-replicating and non self-replicating systems, Tom Ray has the following to say:

> "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." [Ray 91]

It seems that there are a number of fairly fundamental differences between systems which support self-replication (e.g. the systems described in this section), and those in which replication of individuals is governed by the algorithmic engine (e.g. genetic algorithms). Three such differences are listed in Table 2.1.

If we are hoping to evolve programs that are useful to us in ways other than being a study of self-replication, we are again drawn to the question of whether self-replicating

[10] The size of the instruction sets of traditional languages is much larger than one might imagine if all possible combinations of operands are taken into account.

Self-Replicating	*Non Self-Replicating*
Individuals are at liberty to define their own fitness functions	Fitness functions are defined externally and are generally constant
Evolution is open-ended	Evolution is not open-ended
Phenotype must include machine instructions to replicate genotype	Phenotype can be anything that represents a solution to the specified problem—it is not confined to machine instructions

Table 2.1: Properties of self-replicating versus non self-replicating systems

code can be forced to evolve solutions to some externally defined fitness measure. This issue is discussed in Section 3.3.

It should be noted that, strictly speaking, classification of systems as either self-replicating and non self-replicating can be a somewhat arbitrary decision, depending on the extent to which the component parts of the system (i.e. the program, data, interpreter and processor) are intertwined, and on which components are subject to reproduction.

To borrow an example from [Hofstadter 79] (*p. 499*), suppose there is a computer language that has the convention that any program beginning with an asterisk has the side effect of copying the entire program in memory before executing the other commands. Then a program consisting of a single asterisk is self-replicating. The main reason why such a self-replicating system seems a 'cheat' is that the processor is performing the whole copying process in response to a single program instruction rather than a long list of program instructions to explicitly perform the self-replicating task. The evolution in such a system would presumably be somewhat constrained as the processor is beyond the influence of natural selection.

We shall return to issues relating to the 'granularity' of the copying process, and to which components of the system are being copied, in Section 3.2.

2.2 Research with Developmental Systems

There has recently been a considerable growth in research looking at various aspects of developmental systems. Several approaches have been taken, which differ in the

extent of biological inspiration employed, and the extent to which the model uses local inter- and intra-cellular interactions on one hand, to more abstract grammar-based descriptions of the developmental process on the other hand.

To my knowledge, the first published work on proposed mechanisms by which biological organisms achieve spatial pattern formation (morphogenesis) was Alan Turing's paper "The Chemical Basis of Morphogenesis" [Turing 52]. Turing had started working on this subject at around this time, but regrettably this was the only publication before his death[11].

The paper began with a memorable quote which is worth bearing in mind when considering the other models proposed in this section as well:

> "This model will be a simplification and an idealization, and consequently a falsification. It is to be hoped that the features retained for discussion are those of greatest importance in the present state of knowledge." [Turing 52]

Turing proposed a fairly simple mathematical model of a system in which a number of chemical substances (which he called morphogens) reacted with one another and diffused through a tissue, and showed that this model was capable of accounting for many phenomena associated with morphogenesis and pattern formation. However, as attractive as such 'reaction-diffusion' models are, there is still very little direct evidence for them in biological organisms.

Quite fittingly, since Turing's day the use of computer simulations has meant that it is now possible to study the dynamics of much more complicated systems (i.e. with fewer simplifications and idealizations) in a great deal of detail. The rest of this section describes some more recent research, some of which concentrates on modelling processes at the cellular level, and some of which uses somewhat more abstract descriptions.

[11] However, some originally unpublished manuscripts, as well as a reprint of the Morphogenesis paper, have recently been published in [Saunders 92].

2.2.1 Cellular Level Models

Kunihiko Kaneko and Tetsuya Yomo have proposed a model of cell differentiation based on simulations with interacting artificial cells [Kaneko & Yomo 95]. The model comprises a metabolic reaction network within each cell (i.e. each cell contains a number of chemicals each at a certain concentration, and rules are given to describe how these concentrations change over time), interaction between cells (i.e. active transport and diffusion of chemicals), and cell division. No spatial variation is included in the model, although the authors are currently working on this, and cell death is not included, although the authors have experimented with this. One of the main results of this work is that cell differentiation is observed to occur after a certain number of cell divisions, and that this differentiation between groups of cells becomes fixed after a time.

Hiroaki Kitano has taken the basic model proposed by Kaneko and Yomo and added a genetic algorithm to evolve the genetic rules that determine the metabolism of cells [Kitano 94]. Extending this work further, he has added a 'Nerve Growth Factor' to the cells, which promotes 'axon' growth when it reaches a certain threshold within a cell. In this way, a neural network develops over the multicellular organism [Kitano 95][12]. Kitano draws three main conclusions from this work; (1) differentiation of cell types through a process of development is possible in this model, (2) the rate of change of chemical balance in each cell slows down as the size of the cluster grows, and (3) interaction between cells is very important in determining the global behaviour of the system. One weakness in the work is the fitness function employed—a function of the weighted sum of the total DNA produced by an organism, the number of cells it contains and the total length of axons. Kitano acknowledges that fitness should ideally be related to the behaviour of an individual in the environment.

Similar work on growing neural networks has been carried out by Stefano Nolfi, Domenico Parisi and Angelo Cangelosi, but instead of modelling the detailed metabolism of the cells, these researchers modelled a set of developmental instructions governing the behaviour of the neurons. This work generally concerns itself with the growth of a neural network to control an artificial organism searching for food in a 2D world. Al-

[12] The growth of cells and axons in this model all occurs in a 2D space.

though evolution acts upon the neural network, the morphology and sensing apparatus of the organism is fixed.

In [Nolfi & Parisi 95] an organism is evolved whose genome contains a description of up to 40 different neurons. The description includes details such as the position of the neuron in 2D space, synaptic weight, branching angle of axons etc. Although the growth of axons is modelled in this work, the neurons themselves can appear at different places in the network at different times during maturation, depending on information contained in the genome—there is no growth of a multicellular organism from a single 'zygote'.

[Cangelosi *et al.* 94] describes an approach which is more interesting from the current perspective. In this work, there are 16 different *types* of cell, and along with a description of each cell's axon growth similar to that used in [Nolfi & Parisi 95], a set of production rules is also encoded on the genome which dictates the way in which each type of cell divides into two daughter cells (as such, this is a hybrid of a cellular level model (description of axon growth for each cell) and a grammar based model (production rules for each cell type)). The encoded information also includes the location where daughter cells are placed in the network relative to their mother, so that a kind of cell migration is modelled in the system. A network begins as a single egg cell of a specified type, and is grown for five cell division cycles according to the production rules. At this stage, the axons are then grown from the final configuration of cells to form the neural network.

Both [Cangelosi *et al.* 94] and [Nolfi & Parisi 95] include a discussion of the differential effects of mutations affecting the early and late stages of development of an organism, and of the evolutionary consequences of both types of mutation. The conclusions are fairly logical; basically that mutations affecting early stages of development have a much greater impact on the final morphology of an organism than do late-acting mutations (they lead to long jumps in the genomic state space), so are less likely to produce viable individuals. Hence, early-acting mutations are less likely to be retained over evolutionary time, but those that do will produce fairly radical changes in morphology. These ideas agree with and are expanded upon by Stuart Kauffman in his book "The Origins of Order" [Kauffman 93] described in Section 2.3.

On a slightly different note, Frank Dellaert and Randall Beer have proposed a model of development with three components; a genetic regulatory network, a cellular simulator, and an organismal level component to deal with intercellular communication and other high level features [Dellaert & Beer 94]. In this model, a genetically encoded Boolean network encodes the development rules for an artificial autonomous agent. The paper describes experiments in which a 64 cell 'creature', composed of a number of different cell types, grows from a single zygote, and how the phenotype evolves towards an externally specified pattern used to judge the fitness of each organism. As a demonstration of how a fairly complex morphology can arise from a fairly simple developmental model, this work is of great interest, although a number of features were added to the model to make it more likely to succeed[13].

2.2.2 Grammar-Based Models

This section describes work using grammar-based models to describe developmental systems. As was apparent in the previous section, the distinction between cellular level models and grammar-based models is sometimes less than clear—they are probably best viewed as different ends of a spectrum of models. Indeed, Kitano suggests in [Kitano 94] that the morphogenesis observed in his cellular level model may be describable by a grammar-based model.

However, some of Kitano's earlier work on growing neural networks was based explicitly on grammar models. In [Kitano 90] a system is described which uses a genetic encoding of 'graph' L-system production rules to produce the weight matrix of a neural network. The topology of the networks is, however, fixed—the only variability being the number of hidden units employed, which may vary up to a given maximum. The L-system has no context sensitivity during morphogenesis, and the whole net is grown before it is subject to input. Even with this fairly simple system, in a comparison against a direct-encoding system on a 4-n-4 encoder/decoder problem, the indirect, grammar-based model is found to have far superior performance, especially as the problem size

[13] For example, although an organism evolved which had a sensible design (eyes at the front, motors at the rear, and neural tissue internally), this was because it was selected for its similarity to a given model organism rather than because its design was deemed to be functionally fit. Also, each cell in the developing model has some global knowledge of the morphology, i.e. knowledge of whether it is near the midline of the organism or whether it is on an exterior boundary.

is scaled.

Frédéric Gruau has developed a somewhat more complicated developmental system for producing neural networks [Gruau 92], [Gruau 93]. In this system, chromosomes encode a description based upon a cellular rewriting system to specify the development of the architecture and weights of a network. The task of the networks is to solve boolean functions, which they evolve to do successfully. Gruau claims that the system works because (1) the representation is abstract, modular[14] and compact, and (2) any chromosome develops a valid phenotype.

In a similar fashion, Richard Belew has described results of using an ONTOL grammar system to encode the development of neural networks to predict time series [Belew 93]. One finding is that after the first few hundred generations, where only first order solutions are produced, there is a sudden increase in complexity of solutions, with higher order (and better) solutions appearing. This phenomenon, which some authors have likened to the debated evolutionary phenomenon of punctuated equilibria, has in fact been observed in several of the other neural network models reported in this section. It appears in this case that as a network evolves, new units and connections are being added to the hidden layers which don't actually have any effect on its performance (and so are no subject to selection pressure). However, these extra units may eventually come to be connected to the inputs and outputs of the net by mutations in the production rules, and when this happens there may be an instantaneous jump in the performance of the net.

Most of the work described in this section has been inspired to a greater or lesser extent by the work of Aristid Lindenmayer, who developed L-system grammars for the description, analysis and developmental simulation of multicellular biological organisms. There are many different varieties of L-system, some of which can model context sensitivity in the sense that a production rule might specify that one type of node be next to another type for the predecessor of the production to be satisfied. It has been demonstrated that a wide variety of complex morphologies can be created using such systems. Moreover, using *map* L-systems, the morphologies are not restricted to tree structures.

[14] Gruau has shown that a single section of the chromosome is capable of producing the same sub-nets in different parts of the final network.

A good introduction to L-systems is given in [Lindenmayer & Prusinkiewicz 88].

2.3 Research with Self-Organization

In the final section of this chapter, I will briefly mention some recent work on self-organizing systems, and on the effects that self-organization may have on the process of evolution.

Stuart Kauffman and colleagues have published a number of papers on this subject, and most of these ideas appear in his book "The Origins of Order" [Kauffman 93]. Kauffman argues that random variation and selection alone are not the only processes affecting biological evolution, but that self-organizational properties inherent at several levels in natural organisms also play a crucial role in the evolution of complex adaptations. These self-organizational properties may be strong enough to act *against* selection and Kauffman suggests that they can account for similarities between species which are sustained *despite* the pressures of selection. His view is that the study of self-organization adds a great deal to our understanding of the neo-Darwinist position, which he considers to be a correct but incomplete account of natural evolution.

Jari Vaario has also been concerned with self-organization. In [Vaario 94a] and [Vaario 94b] he describes a computational modelling method for adaptive self-organization. The model is an object-oriented language in which cells and environment can interact. In his papers, Vaario stresses the need to include developmental processes in the study of evolution. On the developmental system itself, the need for *local* interactions between cells is stressed, as it has been by other authors. Local interactions are regarded as essential by Vaario both because of their facilitation of self-organization in an organism, and also because they ensure that the developmental process is non-linear, so that a single mutation of the genotype can create a completely different phenotype.

Vaario and Katsunori Shimohara have also published a paper which describes a method of studying formation of structures based on a network of autonomous units, which has some correspondence with biological morphogenetic processes [Vaario & Shimohara 95]. The interactions in the network are modelled by attractive and repulsive forces, the

production of these forces being controlled in each unit by genetic information and environmental factors. The genetic component enables the use of evolutionary algorithms to evolve the interactions and thus to create new structures.

This work, along with other work described in this chapter, demonstrates the growth of interest in studying *local interactions* between subunits of a self-organizing system. I expect such a perspective to be very useful in my proposed research.

In the next chapter, I expand upon some of the ideas that have been highlighted by this review, and introduce some others which I also consider to be relevant for the study of developmental systems in the context of self-replicating computer code.

Chapter 3

Some Thoughts about Development and Self-Replication on a Digital Medium

3.1 The Analogy and Associated Nomenclature

Before proceeding, it will be helpful to clarify the meanings I wish to attach to some of the terms used in the remainder of this paper. The way I define the following terms in the current context indicates the level of analogy being drawn between biological and digital self-reproduction and development.

- **SYSTEM**. A term to encompass a whole collection of self-replicating organisms (which may be of one or many species), together with the physical environment in which they compete, and any necessary INTERPRETING and PROCESSING ENGINE not associated with individual ORGANISMS.

- **ORGANISM**. An individual self-replicator. This term encompasses all components of the self-replicator which are exposed to selection, mutation, recombination etc. For example, in a traditional genetic algorithm, an organism is just an encoded 'chromosome' representing a solution to a problem, whereas in the system proposed in Section 3.5.5, the organism comprises a GENOME, a phenotypic PROGRAM, and some associated PROCESSING MACHINERY.

- **GENOME**. A blueprint for an individual ORGANISM which is interpreted by the INTERPRETING MACHINERY to produce a PROGRAM which is the phenotype of the ORGANISM. The GENOME is passed from one generation of an ORGANISM to the next, subject to possible genetic recombination with other GENOMES, copying errors, mutations etc.

- **PROGRAM**. A set of instructions in the language of the PROCESSING MACHINE of the ORGANISM.

- **INTERPRETING MACHINERY/ENGINE**. The mechanism which reads genetically encoded instructions and translates them into instructions that the PROCESSING MACHINERY can understand. The terms *machinery* and *engine* are somewhat interchangeable, although I generally use the former in situations where the INTERPRETER is part of each ORGANISM, and the latter where it is a hard-coded component of the SYSTEM, unattached to individual ORGANISMS, not subject to evolutionary change and selection, and where it interprets every ORGANISM in the SYSTEM according to the same set of rules.

- **PROCESSING MACHINERY/ENGINE**. The mechanism which executes language instructions according to the interpreted genetic information of an ORGANISM.

- **META-PROCESSOR**. There will be a need, especially when running a SYSTEM on a serial machine, to perform various high-level housekeeping tasks such as time-slicing, memory management etc. The META-PROCESSOR will not be subject to evolution, and may be regarded as being outside of the SYSTEM.

- **SIMPLE SELF-REPLICATOR**. A self-replicator in which there is a one-to-one mapping from genotype to phenotype. Most classical genetic algorithm 'chromosomes' fall into this category.

- **SINGLE-CELLED SELF-REPLICATOR**. A self-replicator in which the genotype is a 'recipe' for constructing the phenotype according to a given developmental scheme. The work described in [Nolfi & Parisi 95] involves such a developmental scheme (although the individual organisms in the system are not self-replicators).

- **MULTI-CELLED SELF-REPLICATOR**. A self-replicator in which, not only is the genotype a 'recipe' for constructing the phenotype, but where the method of construction entails the processes of division and differentiation from an initial 'seed' (or zygote) to a final connected collection of TERMINAL NODES. The systems described in [Cangelosi *et al.* 94], [Dellaert & Beer 94] and [Kitano 95] all involve this type of developmental scheme (although the individual organisms in the systems are not self-replicators).

- **TERMINAL NODE**. A discrete component of a MULTI-CELLED SELF-REPLICATOR, either possessing internal information processing capacities of its own, or representing a single instruction in the language of the PROCESSING MACHINE for the whole ORGANISM. An ORGANISM has a finite set of TERMINAL NODE types.

Note that in the above definitions, the distinction I am drawing between single-celled and multi-celled self-replicators is that the latter have a process of division and differentiation to produce phenotypes, whereas the former do not. The current thesis holds that multi-celled self-replicators are successful at evolving complex adaptations because their genotypes are *recipes* for their phenotypes, *and* they have a developmental process entailing the processes of division and differentiation.

I am not concerned with the fact that in multicellular biological organisms *every* cell has a copy of the same genome and they all act fairly autonomously (they all contain their own processors) even in adulthood. It would appear to be unnecessary in a computer system to have local copies of the genome, because any part of the organism can have direct access to a single master copy of the genome, which is a much more efficient arrangement. As for each cell having its own processor, this is a highly significant feature of biological organisms. However, on a digital medium, having many processors acting individually *within a single organism* would be very expensive in terms of computer processing power. Given the limited computer resources at my disposal, it is therefore probably best to stick to simpler systems and leave enough computer processing power to be able to run large populations of organisms over many generations during an evolutionary run.

3.2 Overcoming the Paradox of Self-Replication

An organism that has the power to replicate itself would seem to be paradoxical, in the manner explained by Harold Morris in the following quote:

> "It seems that an entity with reproductive capacity must possess information descriptive of what it is to generate, as a factory's automated chemical-synthesizing system is programmed with the recipe for the compound it is to synthesize. But now an entity *S0* which is going to replicate *itself* would seem to need information descriptive of itself; we might suppose it is equipped with a blueprint of itself. Yet its offspring *S1*, being identical to *S0*, also has the capacity to self-replicate, so *S1* must possess a self-descriptive blueprint. *S0*'s blueprint then *describes* a blueprint-equipped self-replicator, which means the self-replicator's blueprint describes, among other things, a blueprint. Yet it doesn't stop there, because *S1*'s blueprint must contain a description of *its* blueprint-equipped self-replicating offspring *S2*. Following this transitive line of reasoning out it appears the blueprint of *S0* embeds an infinity of self-descriptions." [Morris 88] *p. 382*

In practice, of course, we see self-replication all around us; in biological organisms and in the systems described in Chapter 2. The apparent paradox of self-replication would appear not to be a real constraint at all—but what is the 'trick' used to overcome it?

The systems of self-replication described in the previous chapter all share the feature that individual organisms contain information which is sometimes treated as *instructions* in the language of the system, and sometimes treated as *data* to be copied to a different part of the computer's memory in the act of self-replication. This duality of meaning—and the separation of the processes of *copying* information and of *processing* it as instructions—is the key to solving the problem.

However, a whole computer system is composed of more than just instructions and data—an *interpreter* is required to translate the instructions into the machine language of the hardware, and a *processor* is required to supervise the execution of these machine language instructions and to perform various associated 'housekeeping' duties.

Looking at any of the described systems from an evolutionary point of view, the fact that it is only the programs and data that are subject to evolution, while the system's interpreter and processor remain hard coded and outside the influence of mutation and selection, suggests that the potential for the system to evolve complex adaptations must be somewhat diminished. In biological systems, DNA, RNA, ribosomes and all the associated cellular machinery constitute the program, data, interpreter and processor, and all of these components are subject to natural selection.

The more self-contained a system is, the more potential it has for truly open-ended evolution. However, the more self-contained the self-replicating system is, the more imposing the paradox of self-replication appears; in a completely self-contained system, we must concern ourselves not only with the replication of program and data, but also of interpreter and processor as well.

The key to overcoming this problem is to include a sufficiently strong support system (to borrow a phrase from [Hofstadter 79] *p. 529*), as exemplified by the biological case. In eukaryotic sexual organisms, the single-celled zygote inherits cell machinery from the maternal gamete (the egg cell). This machinery is sufficient to interpret and process the zygotic DNA and begin the process of cell division and differentiation. As the organism grows, sections of its DNA are decoded which produce more cellular machinery. By the time this organism comes to reproduce, some of *its* cellular machinery is contained within germ line cells (if it is female) to 'bootstrap' the next generation in the same way. Thus we have two systems, A (the cellular machinery) and B (the DNA). A replicates B and B replicates A.

An important goal of my research is to use this analogy to produce a system on a digital medium which is as self-contained as possible.

In such a system, where all components are subject to selection, the possibility arises, when set against a backdrop of coevolution and a changing environment, of a kind of high level selection for species which are good at *evolving*. Richard Dawkins has suggested just this [Dawkins 88]. He believes that if selection among embryologies for the property of evolvability can proceed in a cumulative manner, then

> "Perhaps there is a sense in which a form of natural selection favours, not just adaptively successful phenotypes, but a tendency to evolve in certain directions, or even just a tendency to evolve at all." (*p. 219*)

It will be of great interest to look for evidence of such high level selection during the research proposed in this paper. Note that, if we are to be able to observe this phenomenon (and also to observe phenomena associated with co-evolution, discussed in Section 3.6), we will need a number of distinct species[1] of program to be co-existing. There should, however, be no need to engineer this into the environment, as existing systems (biological or digital) which allow open-ended evolution always seem to produce and be capable of sustaining a rather large number of species.

An additional advantage of a self-contained system is that, as the interpreter and processor of a genome are under the control of selection, it is possible that evolution may be able to fine-tune parameters of the system such as the mutation rate during copying, the degree of epistasis[2] etc. Kauffman has suggested that the values of such parameters may be critical if a population is to be able to maintain high fitness as the complexity of the system increases [Kauffman 93].

3.3 Selfish Genes and Useful Programs

When studying evolution in a system of self-replicating entities, it has been suggested by Dawkins that the only sensible level at which to study natural selection is that of the gene [Dawkins 76]. The definition of a gene is somewhat fuzzy, but is generally regarded as a section of the genome which can produce a differential effect on a particular phenotypic trait, depending on which allele of the gene is present. This reflects the fact that, after the long process of development of an organism, many different genetic and environmental factors will have influenced any given phenotypic trait, so that the

[1] Strictly speaking, we just need considerable genetic diversity rather than a number of *distinct* species (that is, the genomes of individuals in the population could vary smoothly rather then being grouped into species). However, a number of possible reasons have been proposed to explain why distinct species do tend to form in practice, the most likely being evolution in a system where genes may become (geographically) isolated from each other and therefore unable to fully intermix. See [Maynard Smith 89].

[2] The degree of epistasis of genes in a genome is the extent to which the fitness of a given allele depends upon the presence or absence of alleles of *other* genes at different positions in the genome.

notion of a single gene being responsible for a particular trait is a gross simplification. However, it is still valid to talk about *different alleles* of a gene producing *variations* in a phenotypic trait. The picture is complicated by the fact that, in a system of interacting organisms, phenotypic effects of one organism might extend beyond its body even to the extent where it is sensible to talk about a gene in one organism producing a phenotypic effect in *a different* organism (see [Dawkins 82]).

If we look at evolution in a population from this perspective (i.e. as a competition between genes in the gene pool), then we see that it is the *genes which are good at promoting the production of more copies of themselves in the gene pool* that will come to dominate. This doesn't necessarily square with the development of optimally efficient individual organisms[3], which is a very important consideration if we are hoping to evolve self-replicating computer programs that can perform useful tasks as well as reproduction. Three important questions are:

- Is it possible to direct self-replicating programs to perform some useful task as well as reproduction?
- How long can such behaviour be sustained in an evolutionary run?
- How likely is it that a program evolved in such a way will be particularly efficient?

These are open questions, and it is hoped that the work proposed in Chapter 4 may begin to illuminate them.

3.4 Self and Non-Self

If evolution is really all about competition among selfish genes in the gene pool, we are drawn back to the question expressed in Chapter 1 of why it is that, in nature, genes generally group themselves into massive conglomerations—multicellular organisms—which have the sole responsibility for the replication of all of the genes they contain in

[3] In fact, the very notion of 'efficiency' of an individual organism becomes meaningless from this perspective—it is the *genes* which are efficient or inefficient at promoting their own reproduction through their phenotypic effects on *all of the organisms in the system*, and on the environment.

their genomes. In that chapter, several possible answers were suggested, mostly based upon the work of Dawkins.

If we take the digital analogue of a single organism to be a single computer program, then the extent to which this program is an autonomous individual is defined by which parts of the computer's memory it has permission to read and write. Most of the research with self-replicating code has allowed programs the potential of read access to all parts of memory, but of write access only to memory owned by the program. Unsurprisingly, it has generally been found that allowing unrestricted write access leads to a fairly chaotic and destructive environment (see, for example, [Dewdney 84], [Dewdney 85] and [Dewdney 87]).

A program that can read the genome of another program in memory can be regarded as having incorporated this new genetic information with its own. In other words, such a program is performing a kind of genetic recombination. There are a number of ways in which genetic recombination (or, more generally, the mixing of genetic material from a number of organisms) is performed in nature, the most obvious being crossover during reproduction, but also through parasites and symbionts such as bacteriophages, plasmids and transposons.

It has been proposed that sexual reproduction and these other methods of sharing genetic information may confer advantages to a species in conditions where the environment is changeable (be this the physical environment or the other species with which the species interacts and co-evolves), but this is a debated topic. For example, the following quote is from [Fontanari & Meir 90]:

> "Two points which are usually cited in favour of sexual reproduction: (1) sexual organisms evolve more quickly than asexual ones, due to the possibility of two (or more) favourable genes coming together, and (2) sexual organisms are less likely to accumulate harmful mutations as these can be reconstituted through recombination. In the asexual case, a reverse mutation would have to take place. These points should be weighed against the advantage of asexual reproduction, namely that favourable genes are transmitted directly to offspring without the potential harm of mixing them

with unfavourable genes of the mate."

John Maynard Smith has discussed these issues at some length in [Maynard Smith 89].

My research will concentrate, at least initially, on the development of asexual self-replicating code (i.e. a program will only be able to read and write to memory that it owns). Later on, I hope to reach a stage where I am able to study aspects of genetic recombination, either through introducing handwritten ancestral programs that can inspect the genetic information of other programs, or, ideally, by witnessing such an ability spontaneously emerge from an existing asexual population[4].

In order to attack the more fundamental question of why a multicellular organism is selectively advantageous to a single strand of self-replicating material, I also plan to study competition between programs which grow from an indirectly encoded genome (as described above) and simpler programs (analogous to many described in the previous chapter) that contain a one-to-one mapping from genotype to phenotype (i.e. the 'genome' is the program itself).

3.5 Representation

In the preceding discussion, mention has been made of several factors which should be taken into account when deciding upon a suitable representation for genotypes and phenotypes in the proposed system. These include issues concerning:

- the robustness of self-replicating code
- the size of the instruction set
- the granularity of the copying process
- the features of the developmental system
- the degree to which an individual genome contains the power to interpret and process *itself*

[4] Such spontaneous emergence would, of course, only happen if genetic recombination confers an advantage to programs in the environment in which they are evolving.

- the computational completeness of the final system

Trying to accommodate all of these factors in one system will be the first task I must tackle, and I do not underestimate its complexity. It may be that compromises will have to be made with some features—although my intuitive feeling at present is that I would prefer to have a system that tackles *all* of the issues at least to some degree, rather than a system that tackles some issues well while completely ignoring others. The reason for this is that I believe that all of the features in this list will be important components of the goals I am aiming for, and furthermore that each feature will contribute to the final system in a complex and highly interconnected way.

Below I discuss shortcomings of existing representations in the context of the current task, and add a few more points to the discussion, which come to mind even before starting to attack the problem in earnest.

3.5.1 S-expressions, L-systems, Boolean Graphs: A Limited Silver Braid

The various research projects with self-replicating code and developmental systems described in Chapter 2 have employed a variety of representations. Although each representation was appropriate for the particular research in hand, none of them deals with more than 3 or 4 of the points in the list at the start of Section 3.5.

Robert Collins and David Jefferson have reviewed a number of candidate representations for evolving artificial organisms (including parameterized functions, Lisp S-Expressions, deterministic Finite State Automata and primitive rule-based organisms), and came to the conclusion that none of the candidates was particularly suitable [Collins & Jefferson 91]. They suggest that a good representation should have the following properties:

1. (approximate) *closure* of the set of legal genotypes under the action of genetic operators, i.e. the genetic operators must always (or almost always) produce syntactically legal genotypes when applied to other syntactically legal genotypes (where the function that maps from genotype to phenotype defines the legal syntax);

2. *smoothness* of the phenotype under the action of genetic operators, i.e. the phenotype should tend to change smoothly as the genotype is changed by mutation and recombination;

3. the ability to *scale* to large phenotypes;

4. the ability to *evolve* phenotypes that exhibit *both continuous and discrete behaviours* as a function of their inputs, e.g. the ability to *switch* to any one of a repertoire of (continuous) behaviours depending upon environmental conditions; and,

5. a *uniform computational model*, i.e. the programming paradigm in which the behaviour functions are expressed should not contain features that include any kind of explicit or implicit knowledge of the environment, nor bias toward a particular evolutionary trajectory.

Although issues of self-replication and development were not considered by Collins and Jefferson, most of these properties are valid in the current context; (2) is not always necessary—it depends upon whether mutations act early or late in the process of development, as discussed earlier, and the importance of (4) in the current context is unclear—although the ability to *switch* between behaviours is an inbuilt ability of the kind of computer languages we are considering.

Frédéric Gruau has proposed a similar list of features in the context of genetic encodings of neural networks (see [Gruau 92] and [Gruau 94]). The list includes:

1. *completeness*—any neural network can be encoded. This is equivalent to our desire for computational completeness in the current context;

2. *closure* (as in previous list);

3. *compactness* of encoding;

4. *modularity*—a repeated pattern in the network can be represented by a single encoding of the pattern in the genome. This would be a desirable feature for repeated phenotypic patterns in any developmental system;

5. *scalability* (as in the previous list);

In the present context, the features from the two lists above which appear to be most relevant are:

- completeness
- closure
- scalability
- modularity
- uniformity of model

In the rest of this section, I expand upon a few points which have already been raised, and end with a brief sketch of the type of representation I have in mind to attack the problem.

3.5.2 Computational Completeness

As we saw in Chapter 2, it is important to think about the computational completeness of the final system as well as its ability to evolve robustly. This is really what Collins and Jefferson were getting at by insisting on a uniform computational model—we should not try to engineer any assumptions or features into the system which we think might help individuals evolving within it, because in populations of any size, particularly in situations where the environment is changing or co-evolution is occurring, interactions between components in the system will be too complex for us to completely understand. Hence, if we engineer features which we think may be beneficial, we run the risk that these features may actually be *limiting* evolution in some way that we have not accounted for. It is therefore preferable to have a system that contains few or no assumptions about the kinds of adaptation that it may support, but rather make it as *unconstrained* as possible to evolve any and all conceivable (and inconceivable!) organisms.

3.5.3 Granularity of the Copying Process

It has already been mentioned that the number of components of a complete self-replicating system (i.e. program, data, interpreter and processor) which are actually being replicated affects both our intuitive feel for whether 'interesting' self-replication is happening, and also the likelihood of truly open-ended evolution occurring.

Another way to look at this is that it is the extent to which an individual organism contains *explicit* instructions for self-replication that affects its potential for evolvability. A construct that relies too heavily on a hard-coded processor, so that self-replication can be performed by calling a single command of the language, can be said to have *coarse copying granularity*. On the other hand, a construct that contains explicit instructions to direct self-replication (either within the program, or within attached processing machinery which is also subject to evolution), can be said to have *fine copying granularity*, and would be expected to be capable of far more open ended evolution than a construct with coarse granularity.

Of all of the systems described in Chapter 2, Typogenetics comes the closest to replicating program, data, interpreter and processor. Hofstadter's suggested self-replicating Typogenetics strand (as demonstrated in [Morris 88]) encodes everything but the interpreter—the mechanism which reads strands and produces enzymes which are encoded on them. The interpreter is human in Hofstadter's system, or a hard-coded programmatic interpreting engine in Morris's derived system.

Hofstadter compares his system to biological genetic systems, and notes that they are both 'tangled hierarchies' where there is a 'two-way street' of information exchange where neither strands (DNA) nor enzymes can be thought of as being on a higher level than the other (see [Hofstadter 79] p. 513).

3.5.4 Features of the Developmental System

As mentioned in Chapter 2, studies of pattern formation under developmental systems, together with work on self-organization, suggest that *local* interactions between components of the developing phenotype play an important role in the production of complexity in spatial pattern formation and cell differentiation (see, for example,

[Hofstadter 79] (*p. 544*), [Belew 93], [Dellaert & Beer 94] and [Vaario 94a]).

It would appear that a system that exhibits a high degree of local interaction during the developmental stage would probably be highly epistatic at the genetic level. As has been mentioned already, Stuart Kauffman and colleagues have suggested that, as the complexity of a system increases, an optimum degree of epistasis emerges such that the ability of organisms to maintain high fitness values is dented if the actual degree of epistasis in the system is either below *or above* this level (see [Kauffman 93]).

In the proposed work, it will therefore be of interest to look for possible effects of local interactions during development on the evolution observed in the system.

3.5.5 A Brief Sketch of a Possible Solution

My major problem will be in finding some hybrid of the representations discussed in Chapter 2 that is capable of performing the functions of self-replication *and* of development in a way that accommodates the points raised in this section. It would seem that the system will comprise at least three components—the chosen representation must have the capability to describe all of them:

- A *Genome*—an area of memory containing the blueprint for the rest of the system.
- Some *Processing Machinery*—initially inherited from the parent program, this component contains sufficient information to map the genome into its phenotype.
- A *Phenotype*—a program which is formed by developmental rules contained in the processing machinery acting upon the genome until a stage where all of the nodes of the developing pattern are *terminal nodes*, i.e. *instructions in the language of the system.* For self-replication, this program will have to perform at least two functions: (1) it should contain its own copy of the processing machinery to be inherited by its children, and (2) it should contain the necessary code to copy (with high, but not necessarily perfect, accuracy) this processing machinery, together with its genome, into a new part of memory, and somehow signal to the meta-processor that this new creation should now be treated as any other

program.

The processing machinery would begin by writing a 'seed' pattern[5] (contained in the genome) to some area of memory, then proceed to expand this pattern to a larger area of memory according to production rules contained within the machinery. This process of development would stop when the whole pattern has been expanded into a set of terminal nodes, which are instructions in the system's language set.

This is a fairly complicated scheme (to say the least), and raises the question of how an ancestral self-replicating organism can be obtained. In existing self-replicating systems such as Tierra, the ancestral program is hand-coded by the author; this is possible because the ancestor is a *simple* self-replicator, in the sense defined in Section 3.1.

A number of possibilities come to mind:

1. Use a conventional genetic algorithm to evolve organisms that develop programmatic phenotypes. The fitness function employed will reward those programs which get all or some of the way towards self-replication. Such a route does not offer much external control over whether the three component architecture just discussed will be adopted—which may not be a bad thing. A major difficulty with such an approach would be in defining a fitness function which would reward programs which have got 'some way towards' self-replication.

2. A second approach would be to hand-craft the ancestral program. The task could be made tractable by including sufficiently powerful commands in the system's language set. However, we would want to bear in mind the issues concerning the granularity of the copying process, discussed in Section 3.5.3.

3. Thinking of the biological analogy, where, according to neo-Darwinist theory, multicellular organisms evolved from single-celled organisms, it may be that the best approach is to initially create a system of 'single-celled' organisms and look for the emergence (whether unprompted or otherwise) of multicellularity from such a system. It may well turn out that multicellular organisms are, in fact,

[5] A pattern of bits.

only viable in an environment where there is actually a preponderance of single-celled organisms.

3.6 Other Ideas

Reading through the literature, a few other ideas come to mind. These are not concerned directly with self-replication or development, but are issues relevant to any evolutionary system.

3.6.1 Co-Evolution

Brief mention has already been made of systems where a number of different species are *co-evolving*. Assuming that the species are interacting and competing with each other in any manner, then the effective 'environment' in which an individual organism finds itself includes all of the members of these different species (as well as its own). These other species are also evolving, so the environment of an organism in the system changes over evolutionary time. It would appear that co-evolution can accelerate the pace of evolution in each species involved, in a way depending on the ecological relations between the interacting species. Such relations may be competitive, exploitative or mutualistic. Co-evolution is discussed from a biological point of view in [Maynard Smith 89] and [Dawkins 82]. For a more computational view, see [Kauffman & Johnsen 91] and [Koza 91].

3.6.2 The Interaction of Learning and Evolution

Geoffrey Hinton and Steven Nowlan published a short paper entitled "How Learning Can Guide Evolution" [Hinton & Nowlan 87] which has provoked renewed interest in this and related issues.

In their paper, Hinton and Nowlan show, by using a simple computational model, that

> "Learning can provide an easy evolutionary path towards co-adapted alleles in environments that have no good evolutionary path for non-learning organisms."

Their work is the first computer simulation and analysis of this effect, which was actually first proposed exactly 100 years ago by J. Mark Baldwin [Baldwin 96] (and hence often referred to as the Baldwin Effect).

John Maynard Smith, in a comment about Hinton and Nowlan's work, summarizes the Baldwin Effect hypothesis as follows:

> "If individuals vary genetically in their capacity to learn, or to adapt developmentally, then those most able to adapt will leave most descendants, and the genes responsible will increase in frequency. In a fixed environment, when the best thing to learn remains constant, this can lead to the genetic determination of a character that, in earlier generations, had to be acquired afresh each generation." [Maynard Smith 87]

In his paper, Maynard Smith also points out some interesting differences between sexual and asexual populations with respect to learning and evolution. Fontanari and Meir have published work relating specifically to learning and evolution in asexual populations [Fontanari & Meir 90]. In an interesting paper, Richard Belew has extended Hinton and Nowlan's work to look at interactions between evolution, learning and *culture* [Belew 90]. Among other things, Belew notes that evolution, culture and learning are adaptive systems acting on vastly different timescales, and, as such, they allow a lineage to track changes in the environment over each of these different timescales.

It should be noted, however, that the Balwin Effect depends upon a *fixed environment*, which is an unrealistic assumption in most scenarios of interest. David Ackley and Michael Littman have reported an investigation of a system where the environment is not fixed, but the criterion for success of phenotypic behaviours is also evolvable [Ackley & Littman 91]. Their work suggests that in these more realistic conditions, the interactions between learning and evolution become complicated in a number of surprising ways which are discussed in the paper.

In my later work, it would be of interest to include in the system's language set some features that could facilitate learning abilities (if such abilities have not been observed to have evolved spontaneously). This would allow the possibility of a more detailed study of the Baldwin Effect.

In this chapter, I have tried to summarise my current thoughts about the task in hand. In the final chapter, I try to decompose the work into a number of manageable subtasks, each with clear goals, and take an initial stab at defining a timetable for this work.

Chapter 4

Proposed Research Schedule

In this chapter, an initial attempt is made to divide the work I wish to pursue into a number of fairly well defined stages. I have tried to devise stages which each have some clear goal. In the final section I suggest a preliminary timetable.

4.1 Stage 1

- Decide upon a set of principles for assessing the outcome of this work. This will entail deciding which are the questions that I wish to pose and the theories that I wish to test, and thinking about the kinds of answers that I am looking for.

4.2 Stage 2

- Specify the required features of the language to be used, and decide upon an appropriate platform(s) for implementation.
- Implement the meta-processor, interpreter and any other features of the virtual machine that are required in order to be able to use the language.
- Review literature for analysis tools appropriate for evolving systems.

4.3 Stage 3

- Evolve a system that develops a programmatic phenotype from a genotype using a standard genetic algorithm. In other words, have some pre-defined task which we

want the programs to perform, and evolve a population of (non-self-replicating) programs under a standard GA where the fitness of each program is some measure of its performance on the task.

4.4 Stage 4

- Build upon the system developed in Stage 3 to try to evolve a self-replicating program. A number of possible routes to this goal were briefly outlined in Section 3.5.5.

4.5 Stage 5

Experiment with the self-replicating system developed in Stage 4 in order to look at some of the following questions:

- What are the advantages of development? Are multicellular self-replicators fitter than single-celled or simple self-replicators (and in what circumstances)? If so, why?
- Has genetic recombination been seen to emerge in the system? If not, in what ways may it be introduced? In what ways does evolution in the system change when genetic recombination is introduced?
- Is there any evidence for programs evolving the ability to learn? If not, is it possible to add extra functionality to the set of programming language instructions to facilitate the evolution of learning? In what ways does learning interact with development and evolution? Is there any evidence of programs using artifacts (non-replicating parts of the system) as cultural devices?
- What are the features of co-evolution which are apparent in the system?

4.6 A Preliminary Timetable

These figures take into account a number of background activities such as attending EPCC courses, reading and writing papers, attending seminars and conferences etc.

Stage	*Approximate Duration*
1	1 month
2	6 months
3	3 months
4	3 months
5	8 months
Total	21 months

Table 4.1: Preliminary Timetable

Assuming that work starts in February 1996, the project should be complete by the end of November 1997. This seems like a reasonable target, as it leaves 10 months of funding remaining, 6-8 months of which will probably be spent writing the final thesis. This still leaves a few months free for possible overruns.

Bibliography

[Ackley & Littman 91] D Ackley and M Littman. Interactions between learning and evolution. In CG Langton, C Taylor, JD Farmer, and S Rasmussen, editors, *Artificial Life II*, volume X of *SFI Studies in the Sciences of Complexity*, pages 487–509. Addison-Wesley, Redwood City, CA, 1991.

[Adami & Brown 94] C Adami and CT Brown. Evolutionary learning in the 2D artificial life system 'Avida'. In R Brooks and P Maes, editors, *Artificial Life IV*, pages 377–381. The MIT Press, 1994.

[Baldwin 96] JM Baldwin. A new factor in evolution. *American Naturalist*, 30:441–451, 1896.

[Belew 90] RK Belew. Evolution, learning and culture: Computational metaphors for adaptive algorithms. *Complex Systems*, 4:11–49, 1990.

[Belew 93] RK Belew. Interposing an ontogenetic model between genetic algorithms and neural networks. In J Cowan, editor, *Advances in Neural Information Processing (NIPS5)*. Morgan Kaufmann, San Mateo, CA, 1993.

[Cangelosi *et al.* 94] A Cangelosi, S Nolfi, and D Parisi. Cell division and migration in a 'genotype' for neural networks. *Network: Computation in Neural Systems*, 5:497–515, 1994.

[Collins & Jefferson 91] RJ Collins and DR Jefferson. Representations for artificial organisms. In J-A Meyer and SW Wilson, editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behaviour*, pages 382–390, Cambridge, MA, 1991. MIT Press.

[Dawkins 76] R Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, 1976.

[Dawkins 82] R Dawkins. *The Extended Phenotype*. WH Freeman, Oxford, 1982.

[Dawkins 88] R Dawkins. The evolution of evolvability. In C Langton, editor, *Artificial Life*, volume VI of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 201–220, Reading, MA, 1988. Addison-Wesley.

[Dellaert & Beer 94] F Dellaert and RD Beer. Toward an evolvable model of development for autonomous agent sythesis. In R Brooks and P Maes, editors, *Artificial Life IV: Proceedings of the Fourth Internation Workshop on the Synthesis and Simulation of Living Systems*, Cambridge, MA, 1994. MIT Press.

[Dewdney 84] AK Dewdney. Computer Recreations: In the game called Core War hostile programs engage in a battle of bits. *Scientific American*, 250(5):15–19, May 1984.

[Dewdney 85] AK Dewdney. Computer Recreations: A Core War bestiary of viruses, worms and other threats to computer memories. *Scientific American*, 252(3):14–18, March 1985.

[Dewdney 87] AK Dewdney. Computer Recreations: A program called MICE nibbles its way to victory at the first Core War tournament. *Scientific American*, 256(1):8–11, January 1987.

[Fontanari & Meir 90] JF Fontanari and R Meir. The effect of learning on the evolution of asexual populations. *Complex Systems*, 4:401–414, 1990.

[Gruau 92] F Gruau. Genetic synthesis of boolean neural networks with a cell rewriting development process. In D Whitley and JD Schaffer, editors, *Combination of Genetic Algorithms and Neural Networks*, pages 55–74. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[Gruau 93] F Gruau. Genetic synthesis of modular neural networks. In S Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA93)*, pages 318–325, San Mateo, CA, 1993. Morgan Kaufmann.

[Gruau 94] F Gruau. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. Unpublished PhD thesis, Laboratoire de l'Informatique

du Parallélisme, l'Ecole Normale Supérieure de Lyon, 1994.

[Hinton & Nowlan 87] GE Hinton and SJ Nowlan. How learning can guide evolution. *Complex Systems*, 1:495–502, 1987.

[Hofstadter 79] DR Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, New York, 1979.

[Kaneko & Yomo 95] K Kaneko and T Yomo. A theory of differentiation with dynamic clustering. In F Morán, A Moreno, JJ Merelo, and P Chacón, editors, *Advances in Artificial Life: Third Euopean Conference on Artificial Life*, Lecture Notes in Artificial Intelligence, pages 329–340. Springer, 1995.

[Kauffman & Johnsen 91] SA Kauffman and S Johnsen. Co-evolution at the edge of chaos: Coupled fitness landscapes, poised states, and co-evolutionary avalanches. In CG Langton, C Taylor, JD Farmer, and S Rasmussen, editors, *Artificial Life II*, volume X of *SFI Studies in the Sciences of Complexity*, pages 325–369, Redwood City, CA, 1991. Addison-Wesley.

[Kauffman 93] SA Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

[Kitano 90] H Kitano. Designing neural networks using genetic algorothms with graph generation system. *Complex Systems*, 4(4):461–476, August 1990.

[Kitano 94] H Kitano. Evolution of metabolism for morphogenesis. In *Proceedings of Alife-IV*, 1994.

[Kitano 95] H Kitano. Cell differentiation and neurogenesis in evolutionary large scale chaos. In F Morán, A Moreno, JJ Merelo, and P Chacón, editors, *Advances in Artificial Life: Third Euopean Conference on Artificial Life*, Lecture Notes in Artificial Intelligence, pages 341–352. Springer, 1995.

[Koza 91] JR Koza. Genetic evolution and co-evolution of computer programs. In CG Langton, C Taylor, JD Farmer, and S Rasmussen, editors, *Artificial Life II*, volume X of *SFI Studies in the Sciences of Complexity*, pages 603–629. Addison-Wesley, Redwood City, CA, 1991.

[Koza 94] JR Koza. Artificial life: Spontaneous emergence of self-replicating and evolutionary self-improving

computer programs. In C Langton, editor, *Artificial Life III*, volume XVII of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 225–262. Addison-Wesley, 1994.

[Lindenmayer & Prusinkiewicz 88] A Lindenmayer and P Prusinkiewicz. Developmental models of multicellular organisms: A computer graphics perspective. In C Langton, editor, *Artificial Life*, volume VI of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 221–249, Reading, MA, 1988. Addison-Wesley.

[Maley 94] CC Maley. The computational completeness of Ray's Tierran assembly language. In C Langton, editor, *Artificial Life III*, volume XVII of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 503–514. Addison-Wesley, 1994.

[Maynard Smith 87] J Maynard Smith. When learning guides evolution. *Nature*, 329:761–762, 1987.

[Maynard Smith 89] J Maynard Smith. *Evolutionary Genetics*. Oxford University Press, 1989.

[Morris 88] HC Morris. Typogenetics: A logic for artificial life. In C Langton, editor, *Artificial Life*, volume VI of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 369–395, Reading, MA, 1988. Addison-Wesley.

[Nolfi & Parisi 95] S Nolfi and D Parisi. Evolving artificial neural networks that develop in time. In F Morán, A Moreno, JJ Merelo, and P Chacón, editors, *Advances in Artificial Life: Third Euopean Conference on Artificial Life*, Lecture Notes in Artificial Intelligence, pages 353–367. Springer, 1995.

[Ray 91] TS Ray. An approach to the synthesis of life. In Langton, Taylor, Farmer, and Rasmussen, editors, *Artificial Life II*, pages 371–408. Addison-Wesley, Redwood City, CA, 1991.

[Saunders 92] PT Saunders, editor. *Morphogenesis*. Collected Works of AM Turing. North-Holland, Amsterdam, 1992.

[Skipper 92] J Skipper. The computer zoo—evolution in a box. In FJ Varela and P Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 355–364, Cambridge, MA, 1992. MIT Press.

[Thearling & Ray 94] K Thearling and TS Ray. Evolving multi-cellular artificial life. In R Brooks and P Maes, editors, *Artificial Life IV*, pages 283–288. The MIT Press, 1994.

[Turing 52] AM Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London*, 237 B:37–72, August 1952.

[Vaario & Shimohara 95] J Vaario and K Shimohara. On formation of structures. In F Morán, A Moreno, JJ Merelo, and P Chacón, editors, *Advances in Artificial Life: Third Euopean Conference on Artificial Life*, Lecture Notes in Artificial Intelligence, pages 421–435. Springer, 1995.

[Vaario 94a] J Vaario. From evolutionary computation to computational evolution. *Informatica*, 18(4):417–434, 1994.

[Vaario 94b] J Vaario. Modeling adaptive self-organization. In R Brooks and P Maes, editors, *Artificial Life IV*, pages 313–318. The MIT Press, 1994.

[Williams 66] GC Williams. *Adaptation and Natural Selection.* Princeton University Press, Princeton, NJ, 1966.