CHAPTER 4

CONTEXTUAL GENETIC ALGORITHMS[31]

1 Models with both Dynamic and Selective Dimensions

The origin of coded systems with both dynamic and selective dimensions is far from understood. It is in fact the problem of the origin of life. But whether or not we know how coded systems can naturally arise, should not stop us from exploring the dynamics-selection coupling of Selected Self-Organization, as described in chapter 2, in scientific models and computational tools. Especially regarding the latter, we may be able to improve information compression in current evolutionary computation algorithms tremendously by including self-organizing layers between solution encoding and expression.

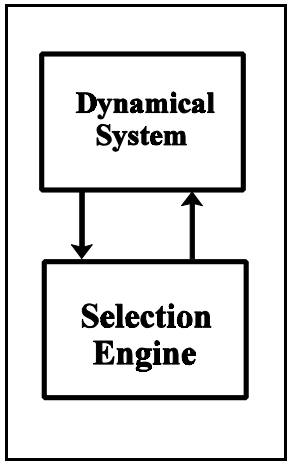

Figure 2: Models of Selected Self-Organization should be based on the coupling of a dynamical system to a selection engine.

The study of evolutionary systems is often divided in two camps: those that are concerned with self-organizing properties of dynamical systems, and those concerned with models of genetically driven natural selection. The former often use boolean networks and cellular automata as their computational models, while the latter use genetic algorithms or other forms of evolutionary computation. It is not very common to build models with both dimensions in order to explore the nature of this fundamental coupling. It is precisely the nature of the relationship between memory tokens of the selection engine, and construction parts of the self-organizing dynamics that I wish to explore formally here (figure 1). This relationship is in living systems implemented by the genetic code. In abstract terms, it is the harnessing of dynamic materiality by symbolic memory [Pattee, 1982, 1995a]. To fully explore it, it may be useful to frame the problem in terms of the science of semiotics.

2 Semiotics of Living Organizations

As introduced in section 1.7 of chapter 2, semiotics concerns the study of signs/symbols in three basic dimensions: syntactics (rule-based operations between signs within the sign system), semantics (relationship between signs and the world external to the sign system), and pragmatics (evaluation of the sign system regarding the goals of their users) [Morris, 1946]. We can understand the semiotics of the genetic

[31] The material presented in this chapter has been previously published in Rocha [1995c, 1995f, 1997d],

system if we consider all processes taking place before translation as the set of syntactic operations; the relation between mRNA (signifier) and folded amino acid chains (signified), through the genetic code, as the implementation of a semantic relation; and finally, the selective pressures on the developed phenotypes as the pragmatic evaluation of the genetic sign system. Figure 2, which is adapted from Peter Cariani's [1987] extensive discussion of the semiotics of living organizations, depicts these relationships.

Computational models of evolutionary systems, such as genetic algorithms, explore only small portions of the semiotics of the genetic system scheme as depicted in figure 2, namely the pragmatics axis between genotype and phenotype through a linear code that establishes simpler semantic relations. They do not generally explore the syntax axis or the development portion of the semantics axis (except for a few exceptions in the field of Artificial Life [Kitano, 1984; Dellaert and Beer, 1984].) I feel that the inclusion of a more complete picture of the semiotics of living organisms in computational models, can contribute to a much better understanding of evolutionary systems, as well as the definition of more efficient tools for adaptive systems theory. I will discuss the inclusion of syntax and a more complete developmental semantics separately in the next sections. Let us start by looking into the semiotics of the genetic system in more detail.

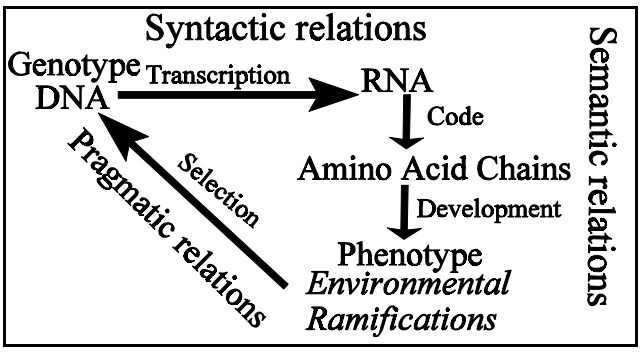

Figure 3: Semiotics of the Genetic System

2.1 Two Type Symbol System: Contextual Environmental Information

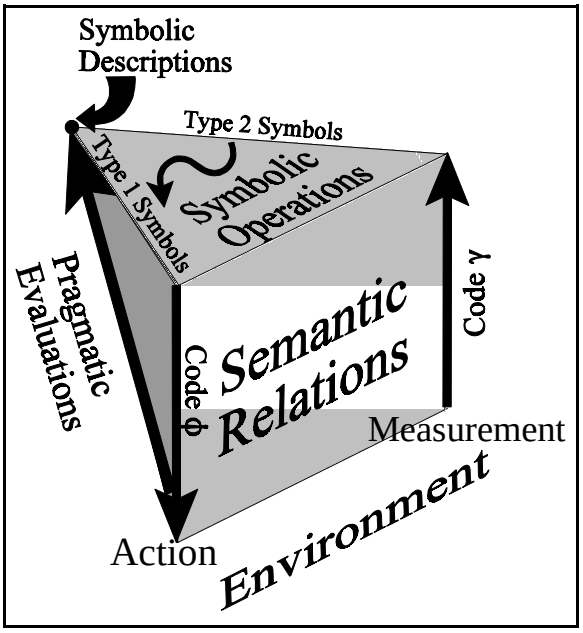

Figure 4: Genetic semiotics with 2 type symbol system

Until now, the semiotics of DNA has been considered to be strictly unidirectional: DNA stands for proteins to be constructed. In other words, the symbolic DNA encodes (through the genetic code) phenotypes with repercussions in some environment. Naturally, through variation and natural selection (pragmatic evaluations) new semantic relations between genes and phenotypes are created which are better adapted to a particular environment. However, real-time contextual measurements are not allowed by this unidirectional semiotics. If in addition to symbols standing for actions to be performed, the genetic sign system is also allowed a second type of symbols standing for environmental, contextual, measurements, then a richer semiotics can be created which may have selective advantage in rapidly changing environments, or in complicated, context dependent, developmental processes. Figure 3 depicts such a sign system. The top plane contains two different types of symbols which are combined in different ways (symbolic operations). *Type 1* symbols stand for actions through a *code* ϕ (e.g. the genetic

code) and *type 2* symbols stand for measurements through a different *code* γ which is being hypothesized here.

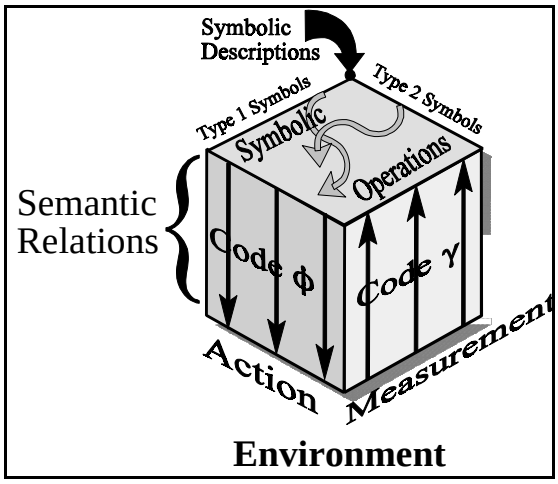

Figure 5: Semantic Closure with 2 symbol types

Notice that *code* γ is proposed here as an abstraction referring to the set of mechanisms which will link environmental measurements (context) to *type 2* symbols. It is **not** expected to function as a proper genetic code with clear cut symbols (nucleotide codons standing for aminoacid chains). Jon Umerez [1995] has stressed the importance of a code in any form of evolving semiotics. In simple terms, what I refer to as a code here is any mechanism able to relate "inert" material structures (signifiers) to other material structures with some functional dynamics (signifieds) "by virtue" of a larger organizational closure. In other words, the function of the material signifiers is not dependent on their particular materiality, but on what they are used to refer to for the imbedding, material semantic closure [Pattee, 1995a], as discussed in chapter 1. Leaving pragmatic evaluations (selection) out of the picture momentarily, the semantic closure with two symbol types, which is able to act as well as perform measurements on its environment can be represented by the cube in figure 4. The semiotic triadic relationship is only complete when individual semantic closures are coupled to an environment which ultimately selects (pragmatic evaluation) the most fit amongst these symbol-matter closures (e.g. in natural selection, those that reproduce the most), as depicted in figure 5. In section 4, genetic systems with 2 types of symbols are presented by both discussing the RNA editing system and proposing a formal counterpart.

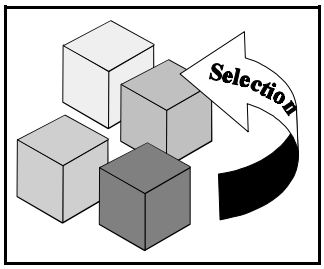

Figure 6: Selection of Semantically Closed Units

2.2 Embodiment and Implementation Dependence: Selected Self-Organization

The issue of materiality is extremely important for two reasons: (i) all which can be represented in this evolutionary semiotics is restricted to what can be constructed by the specific, material, semantically closed system in the first place; and (ii) selection is ultimately performed on this specific material organization capable of performing a number of functions in an environment. The conceptual framework put forward by this material, evolutionary, semiotics forces self-organization and selection together as two indispensable dimensions of evolutionary systems as discussed in chapter 1. Selection takes place on particular dynamics, on the other hand, open-ended evolution is only possible through the existence of a symbolic dimension mediated through a code. Moreover, this code must be built out of some materiality that constrains its representation power and which also ultimately defines an organism's ability to construct and discriminate its environment. This last point raises the issue of implementation-independence and multiple realizability [Umerez, 1995]. A semantically closed system is not implementation independent because matter constrains its classification

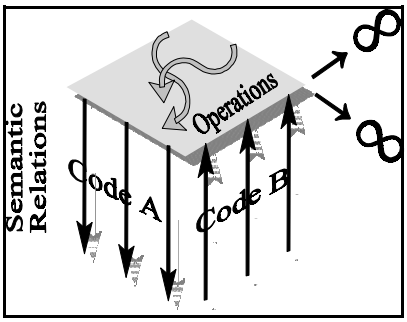

Figure 7: Formal Semiotics

power as well as its evolutionary potential. The second constraint is clear when we realize that two distinct closures which at some point may establish the same representational function, if materially different, will potentially evolve differently given their situated interaction with an environment (see section 2.1.4 in chapter 2). The first constraint is not so clear since we hypothetically allow the idea that two different closures can have the same representational function. However, this equivalence can only be established between formal symbol systems which by definition are not materially constrained and are therefore universal, that is, the set of possible semantic relations is infinite (figure 6). Material symbol systems do not have this property. A coding relation must be formed out of certain available material parts in each domain (e.g. nucleotides and aminoacids in the genetic code), and no semantic relation can escape them (this was discussed as the parts problem in section 1.5 of chapter 2). In the genetic system we can represent any protein, but we cannot represent and construct any other material structure which is not made out of aminoacid chains. Thus, our semiotics is necessarily constrained by matter, not just due to selection pressures, but on account of the parts available for the symbol system itself (figure 7).

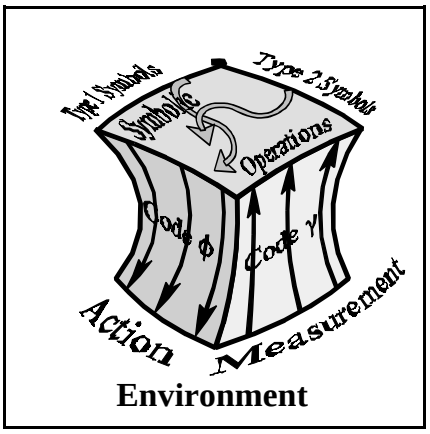

Figure 8: Material Semiotics

Material sign systems are not universal and cannot represent anything whatsoever, but this turns out to be their greatest advantage. The price to pay for the universality of formal symbol systems is complete specificity, that is, full description of its components and behavior. Conversely, material sign systems are based on certain building blocks which do not need a description. For instance, DNA does not need to encode anything other than aminoacid chains, there is no need to include in genetic descriptions information regarding the chemical constituents of aminoacids nor instructions on how to fold an aminoacid chain — folding comes naturally from the dynamical self-organization of aminoacid chains. Notice how a logical simulation of these genetic mechanisms needs to include all this information that comes for free when the self-organizing characteristics of matter are actually used rather than simulated [Moreno et al, 1994]. This information compression is discussed in section 5 by presenting a formal system of development in Genetic Algorithms based on fuzzy logic. A computer simulation of this material semiotics is developed in chapter 5.

3. Contextual Genetic Algorithms

The essence of Genetic Algorithms (GA's) lies on the separation of the description of a solution (e.g. a machine) from the solution itself: variation is applied solely to the descriptions, while the respective solutions are evaluated, and the whole selected according to this evaluation [Holland, 1975]. A genetic algorithm "is primarily concerned with producing variants having a high probability of success in the environment" [Langton, 1989, page 35]. Nonetheless, one important difference between evolutionary computation and biological genetic systems, lies precisely on the connection between descriptions and solutions, between signifier and signified. In genetic algorithms the relation between the two is linear and direct: one description, one solution. While in the biological genetic system there exists a multitude of processes, taking place between the transcription of a description and its expression, responsible for the establishment of an uncertain relation between signifier and signified, that is, a one-to-many relation.

> "The proteins encoded by [DNA] are [...] oxymorphic: their individual shapes are precisely unpredictable. So long as this is true, the genomic language, like our own languages, will not have a logical link between

signifier and signified. This will not prevent its being read or understood; rather, it will assure that DNA remains a language expressing as full a range of meanings through arbitrary signifiers as any other language." [Pollack, 1994, p. 70]

In other words, the same genotype will not always produce the same phenotype; rather, many phenotypes can be produced by one genotype depending on changes in the environmental context. If the effects of changing environmental contexts affecting gene expression within an individual can be harnessed and used to it's selective advantage in a changing environment, then we can say that such an individual has achieved a degree of control over its own genetic expression. It is the objective of this chapter to propose computational schemes which may be able to achieve this degree of control.

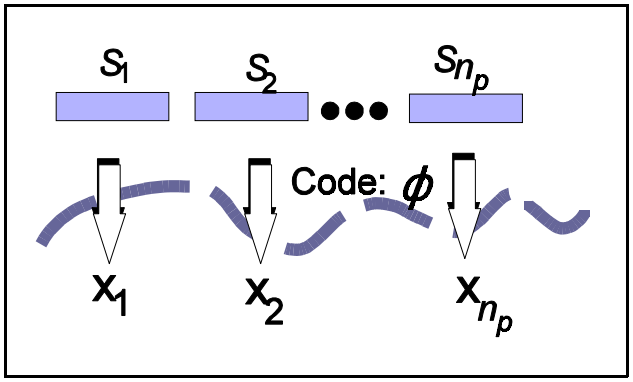

Figure 9: Direct Encoding of traditional GA's

Genetic algorithms explore the semiotics of the genetic system as depicted in figure 2, solely in its pragmatic dimension and in the coded information portion of its semantic dimension. A GA is defined by a population of symbol strings *S* (Chromosomes) which encode a population of solutions *x* to a problem. Variation (e.g. crossover and mutation) is applied to the strings, while the solutions are evaluated regarding some problem. The best chromosome-solution pairs, according to the evaluation preformed by a fitness function, will have more (mutated and crossed over) copies of themselves in the next generation. The fitness evaluation implements the pragmatic dimension, while the encoding of solutions to our problem in descriptional strings implements the coded portion of the semantic dimension. Traditional GA's have a one-to-one mapping between chromosomes and solutions, in other words, there is a direct encoding scheme between genotype and phenotype (figure 8). Code ϕ in figure 8 implements the semantic relation: on the upper side we have descriptions (signifiers), and on the lower side solutions (signifieds), which relate to each other linearly.

In order to expand GA's to model more aspects of the semiotics of the genetic system, we can act on both sides of this code. Expanding the upper portion enhances the syntax (e.g by allowing 2 types of symbols), while expanding the lower portion enhances the semantics of GA's with a developmental stage, yielding GA's with indirect encoding and a more complicated syntax (figure 9). Both of these expansions can introduce nonlinear interactions, in particular the extension of semantic relations with a developmental self-organization system such as boolean networks [Dellaert and Beer, 1994] or L-Systems and Neural Networks [Kitano, 1994, 1995] offers the desired coupling of models of self-organization to models of Natural Selection.

The expansion of the syntactic dimension makes GA's more accurate models of the Genetic System which is rich in nonlinear interactions

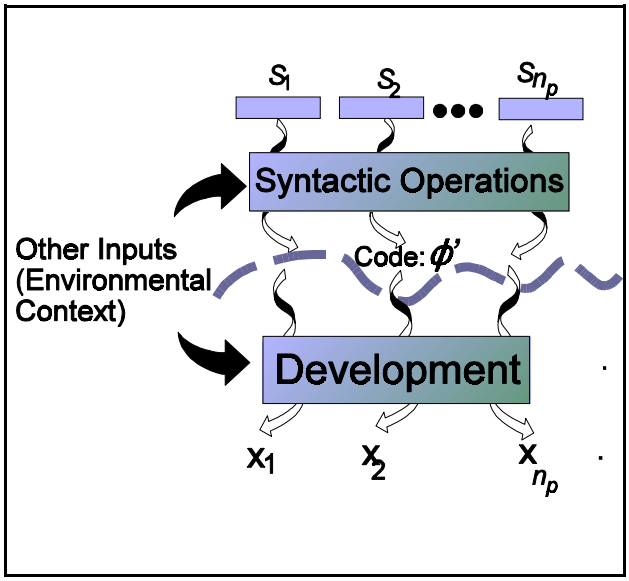

Figure 10: Indirect Encoding and Extended Syntax in Contextual Genetic Algorithms

before translation through the genetic code (e.g. RNA Editing). Furthermore, since the systems which implement the development of solutions and the manipulation of chromosomes (symbol strings) can receive inputs other than chromosomes (e.g. environmental observables) genetic transcription and solution development may be dependent on contextual factors. Hence, I refer to these expanded GA's as *Contextual Genetic Algorithms* (CGA's). In the next sections both of these avenues are pursued separately.

4. Exploring Syntax and RNA Editing

The genetic semiotics described in section 2.1 (figures 3 and 4) expands the syntax of the traditional genetic semiotics of figure 2 by postulating richer symbolic interactions than mere DNA/RNA transcription. If a second type of symbols exists, which operate with genetic messages and in so doing change the latter's encoded meaning, their access to environmental information can provide the genetic system real-time control of genetic expression according to context. This ability would certainly be useful for phenotypical development in changing environments. Some evidence has been presented [Benne, 1993; Stuart, 1993; Simpson and Maslov, 1994; Lomeli et al, 1994] that RNA Editing is used in some genetic systems in different amounts according to different contexts (namely, different stages of a developmental process).

4.1 RNA Editing

The discovery of messenger RNA (mRNA) molecules containing information not coded in DNA, first persuaded researchers in molecular biology that some mechanism in the cell might be responsible for post-transcriptional alteration of genetic information; this mechanism was called 'RNA Editing' [Benne at al, 1986]. "It was coined to illustrate that the alterations of the RNA sequence (i) occur in the protein-coding region and (ii) are most likely the result of a posttranscriptional event" [Benne, 1993, page 16]. The term is used to identify any mechanism which will produce mRNA molecules with information not specifically encoded in DNA. Usually we will have insertion or deletion of particular bases (e.g. uridine), or some sort of base conversion (e.g. adenosine → guanisine).

The most famous RNA editing system is that of the African Trypanosomes [Ibid; Stuart, 1993]. The mitochondrial DNA of this parasite, responsible for sleeping sickness, "consists of several dozen large loops called maxicircles and thousands of smaller ones called minicircles." [Rennie, 1993, page 132] At first, the minicircles were assumed to contain no genetic information, while maxicircles were known to encode mitochondrial rRNA. However, the maxicircles were found to possess strange sequence features such as genes without translational initiation and termination codons, frame shifted genes, etc. Furthermore, observation of mRNA's showed that many of them were significantly different than the maxicircles from which they had been transcribed. These facts suggested that mRNA's were edited post-transcriptionally.

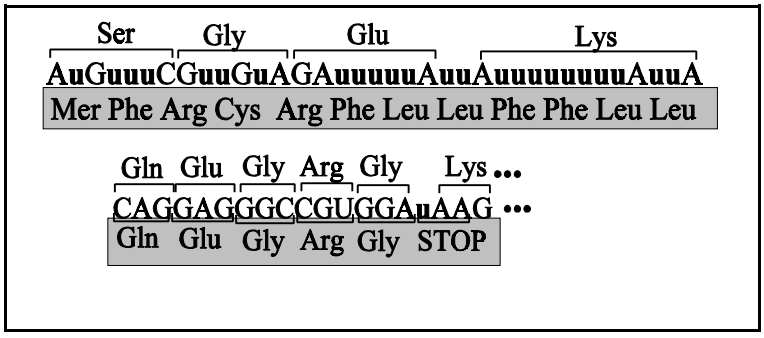

Figure 11: U-insertion in Trypanosomes' RNA

It was later recognized that this editing was performed by *guide RNA's* (gRNA's) coded mostly by the minicircles, the strands of DNA previously assumed to contain no useful

information [Sturn and Simpson, 1990; Blum, Bakalara, and Simpson, 1990]. In this particular genetic system, gRNA's operate by inserting, and sometimes deleting, uridines. To appreciate the effect of this edition consider figure 10. The first example [Benne, 1993, p. 14] shows a massive uridine insertion (lowercase u's); the aminoacid sequence that would be obtained prior to any edition is shown on top of the base sequence, and the aminoacid sequence obtained after edition is shown in the gray box. The second example shows how potentially the insertion of a single uridine can change dramatically the aminoacid sequence obtained; in this case, a termination codon is introduced.

It is unclear how exactly gRNA's insert uridines into mRNA's; basically, the shorter gRNA strings base-pair with stretches of mRNA, and at some point will insert a number of uridines [Seiwert and Stuart, 1994]. An interesting aspect of the gRNA/mRNA base-pairing is that it is more general than the Watson-Crick base-pairing found in DNA and RNA, it is more ambiguous since “uracils in mRNA can be specified by either guanine or adenine in gRNA” [Stuart, 1993, page 36]

But even if the precise mechanisms of RNA editing are not yet know, its importance is unquestionable, since it has the power to dramatically alter gene expression: "cells with different mixes of [editing mechanisms] may edit a transcript from the same gene differently, thereby making different proteins from the same opened gene." [Pollack, 1994, page 78] (one-to-many relations). It is important to retain that a mRNA molecule can be edited in different degrees precisely according to the concentrations of editing operators it encounters. Thus, at the same time, several different proteins coded by the same gene may coexist, if all (or some) of the mRNA's obtained from the same gene, but edited differently, are meaningful to the translation mechanism.

If the concentrations of editing operators can be linked to environmental contexts, the concentrations of different proteins obtained may be selected accordingly, and thus evolve a system which is able to respond to environmental changes without changes in the major part of its genetic information (genome size optimization). One gene, different contexts, different proteins. This may be precisely what the Trypanosome parasites have achieved: control over gene expression during different parts of their complex life cycles.

> "Space is clearly not a problem for mammalian nuclear DNA, so the [previous] rationale is not so obvious for the [editing mechanisms of mammals]. Also there, however, we see one gene encoding two proteins. In mammalian genomes, gene duplication followed by separate evolution of the two copies would be a more obvious way of producing closely related proteins in regulatable amounts. RNA editing, however, does provide the opportunity to introduce highly specific, local changes into only some of the molecules. [...] It could be reasoned that somehow this would be more difficult to achieve via gene duplication, since independently accumulating mutations would make it harder to keep the remainder of the two sequences identical" [Benne, 1993, p. 22]

Thus, RNA editing may be more than just a system responsible for the introduction of uncertainty (one-to-many relations), but also, and paradoxically, a system that may allow the evolution of different proteins constrained by the same genetic string. In other words, even though one gene may produce different mRNA's (and thus proteins), the latter are not allowed heritable variation since they are always constrained by the gene from which they are edited, and which is ultimately selected and transmitted to the offspring of the organism. We can see RNA Editing, especially in the case of gRNA's, as a case of co-adaption of two distinct systems: the stored genetic information (e.g. maxicircles) and the contextual editors (e.g. minicircles), also stored in DNA, but independent and meaningless to the larger semantic loop of the genetic code (figure 11).

The dependent evolution of one gene and several contexts, as expressed by Rob Benne in the previous quote, may allow the introduction of highly specific, local (contextual) changes, more effectively than the independent evolution of several genes. If all of the different expressions were allowed different

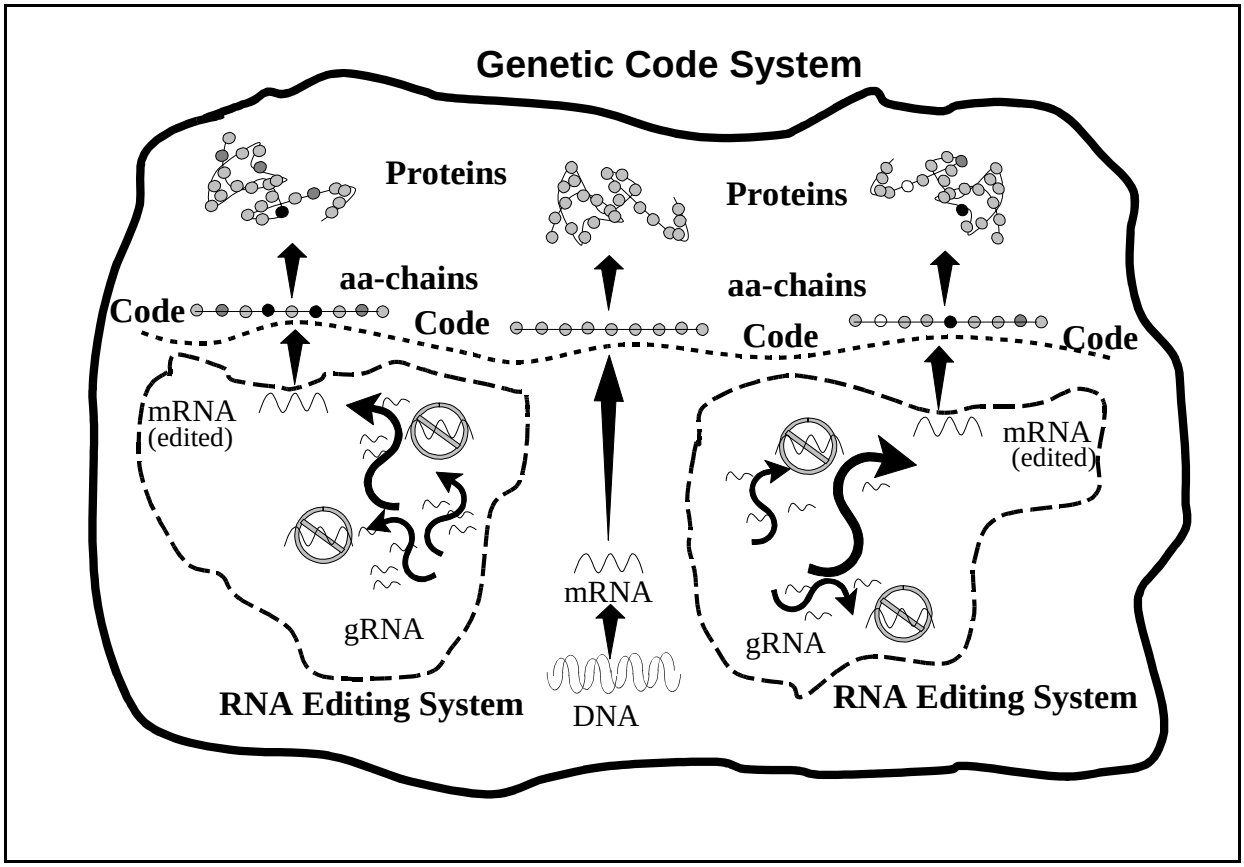

Figure 12: RNA Editing System

genes, they would evolve separately not only increasing the size of the genome, but also, possibly, making it harder to maintain coherent, multicellular, phenotypes as well as coherent developmental processes. For instance, the editing of several genes of the *Trypanosoma Brucei* is developmentally regulated [Stuart, 1993] which may be of evolutionary advantage for these parasites [Simpson and Maslov, 1994]. Though in the course of evolution editing was partially or completely eliminated in many lineages of eukaryotic organisms containing mithocondria, by reverse transcription of partially edited mRNA's, it may be useful for the development of parasitic adaptations as is the case of the developmental regulation of editing in *T. Brucei*, because parasites need to survive in several completely different environments which require very different responses from them[Ibid]. The African Trypanosomes for instance, use the famous Tsé Tsé flies as carriers before infecting mammals; both present the parasite with completely different environments that trigger in it very different stages of development, at least in great part through the workings of the RNA editing system.

The role of RNA editing in the development of multicellular organisms has also been shown to be important, Lomeli at al [1994] have discovered that the extent of RNA editing affecting a type of receptor channels responsible for the mediation of excitatory postsynaptic currents in the central nervous system, increases in rat brain development. As a consequence, the kinetic aspects of these channels will differ according to the time of their creation in the brain's developmental process.

We can think of DNA as a set of symbolic descriptions based on two types of symbols: *type 1* symbols will be expressed in mRNA molecules and will stand for actions to be performed; *type 2* symbols will be expressed in gRNA molecules (or other editing mechanisms) and will stand for contextual observables. RNA editing can be seen as a set of symbolic operations performed with symbols of both types,

resulting in symbols of *type 1* to be translated into actions by the genetic code. This implements the two type symbol semiotics system described in section 2.1.

4.2 A Formal Model of Genetic Editing

In this subsection I develop a formal model of CGA's which expands the syntactic dimension alone (figure 12). It is a conceptual model of RNA Editing, and thus of the two type symbol semiotics of section 2.1.

In GA's, genes are substituted by strings of symbols taken from a binary vocabulary $\boldsymbol{V = \{0, 1\}}$ and called ***V-strings***. The genotype of an individual, referred to as its *symbolic description*, is the set of *V*-strings necessary to produce a phenotype or *solution alternative* [Goldberg, 1989]. The translation of symbolic descriptions into the space of solutions is performed by invariant formal rules which define a code for a particular application. In the following, symbolic descriptions are comprised of only one *V*-string.

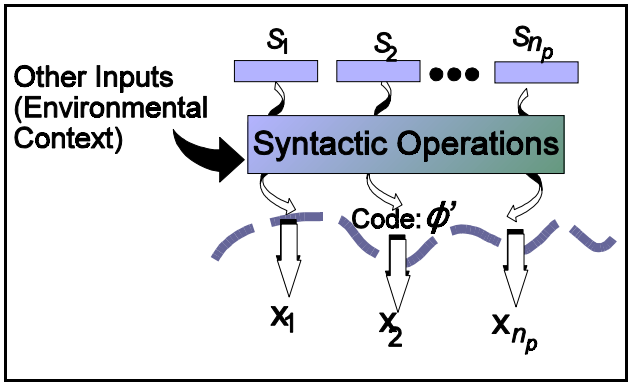

Figure 13: CGA with Expanded Syntax Only

Definitions necessary for a GA:

1. *V* is a vocabulary with two symbols: $V = \{0,1\}$.
2. *S* is a *V*-string of dimension n : $S = s_1 s_2 s_3 \ldots s_n$, $s_i \in V$, $i = 1, 2, ..., n$. Let $\boldsymbol{S}^n$ denote the power set of *V*-strings of dimension *n*.
3. Different operators can be defined to manipulate *V*-Strings, they are functions defined as follows: O: $\boldsymbol{S}^n \times \boldsymbol{S}^n \times ... \times \boldsymbol{S}^n \rightarrow \boldsymbol{S}^n \mid O(S_1, S_2, ... S_k) = S \in \boldsymbol{S}^n$. Examples of such operators are the usual *Crossover* and *Mutation* operators. $Cross(S_1, S_2) = S_3, S_4$, this operator randomly chooses a locus, $1 \leq j \leq n$, of the two input strings and produces two other strings constructed from the first, by exchanging sub-sequences before and after the locus. $Mut: (S_1) = S_2$, this operator randomly flips some of the symbols in the string by other symbols in *V*. This flipping can occur in every position of the string with a very small probability.
4. $\boldsymbol{P}(g) = \{S_i \mid i = 1, ..., n_p\}$, is a population of n_p *V*-strings at generation *g*.
5. $\boldsymbol{X} = X_1 \times X_2 \times ... \times X_d$ is a space of solutions, of dimension *d*, for a particular problem. X_i is the universal set of a relevant variable x_i, $i = 1, 2, ..., d$. ϕ maps *V*-Strings *S* uniquely into solution alternatives $\boldsymbol{x}$. $\phi : \boldsymbol{S}^n \rightarrow \boldsymbol{X} \mid \phi(S) = \boldsymbol{x} \in \boldsymbol{X}$. This mapping establishes the translation rules between symbolic descriptions and solution alternatives: the code. An *individual* is composed of a symbolic description, $S \in \boldsymbol{S}^n$, and a solution alternative, $x \in \boldsymbol{X}$ computed through ϕ.
6. The fitness of a solution *x*, regarding some problem, is given by the function *fit*: $X \rightarrow \mathbb{R}$

The Algorithm (adapted from Mitchell [1996a]):
After defining the code ϕ for some problem:

1. Start with a randomly generated population $\boldsymbol{P}$.
2. Calculate $fit(x)$ for *x* encoded in each *V*-String of $\boldsymbol{P}$.

3. Create a new population ***P***' from ***P***.
4. Go to step 2 (unless some fitness satisfaction criteria is met).

Step 3 is defined by repeating the following sub-steps until n_p *V*-Strings have been created:

a. *Select* a pair of parent *V*-Strings from ***P***. The probability of selection is proportional to the value of fitness of each *V*-String calculated in step 2. The larger the value of *fit*(*x*), the larger the probability of *x* being selected. Selection is done "with replacement", that is, the same V-String can be selected more than once.
b. *Crossover*: with probability p_c (Crossover probability), use the operator *Cross*, from definition 3 above, on the *V*-String pair selected in *a*. In no crossover is performed, form two offspring which are exact copies of the parent *V*-Strings. If crossover does take place, the offspring will be the output *V*-Strings from *Cross*.
c. *Mutate* the two offspring at each *V*-String position with probability p_m (mutation probability). Place the resulting *V*-Strings in the new population ***P***'.

In the traditional GA above, the relation between *S* and *x* is a result of direct application of the mapping ϕ. In the CGA with string editing, however, this mapping is more complicated. Before *S* is translated into the space of solutions, it will possibly be altered through interaction with a different sort of string.

Definitions Necessary for the CGA with String Editing:

1. U is a vocabulary with three symbols: $U = \{0,1,*\}$.
2. E is a string of length m over the vocabulary U, or a U-string of dimension m: $E = e_1e_2e_3 \ldots e_m$, $e_i \in U$, $i = 1, 2, \ldots, m$. Let $\boldsymbol{E}^m$ denote the power set of U-strings of dimension m. These U-strings function as the editing agents of the population of V-strings. The length of U-strings is assumed much smaller than that of the V-strings: $m << n$, usually an order of magnitude. Maintaining the analogy with the RNA editing system of the Trypanosomes, V-strings can be referred to as *maxistrings*, and U-strings as *ministrings*. Here I will assume that the editing agents are constant, that is, the structure of the ministrings will be maintained through the successive generations of ***P***.
3. Let $\mathscr{E}$ denote a finite family (ordered set) of l U-strings: $\mathscr{E} = \{E_1, \ldots, E_l\}$.
4. For each family of U-strings, $\mathscr{E}$, there exists an associated family of mappings $\mathscr{F} = \{f_1, f_2, \ldots, f_l\}$. Each mapping f_i associates its respective U-string in $\mathscr{E}$ with a V-string, and produces another V-string: f_i: $\boldsymbol{E}^m \times \boldsymbol{S}^n \rightarrow \boldsymbol{S}^n$. The associated pair $(\mathscr{E}, \mathscr{F}) = \{(E_1, f_1), (E_2, f_2), \ldots, (E_l, f_l)\}$ is called a *family of editors*. In other words, each editing ministring will have a function which is also dependent on the maxistring to be edited. This function will result in an edited maxistring, and thus specifies how a particular ministring edits maxistrings: when the ministrings match a portion of a maxistring, a number of symbols from the V vocabulary is inserted into or deleted from the maxistring. To introduce the sort of ambiguity the guanine-uracil base pairing creates in the gRNA/mRNA duplex, the U includes an extra symbol '$*$', matching both '1' or '0' in V. Ministrings match more than one subsequence of maxistrings.
5. A U-string $E \in \boldsymbol{E}^m$, *matches* a substring, of size m, of a V-string, $S \in \boldsymbol{S}^n$, at position k if: $\exists_{k \mid 1 \le k \le n}, \forall_{i = 1,2,\ldots,m}: e_i = s_{k+i} \vee e_i = *$

6. Example of a family of mappings f: $E^m \times S^n \rightarrow S^n$. $\mathcal{F} = \{Add\_1(E, S), Del\_1(E, S\})$. ***Add_1*** will add the symbol '1' at position $k+m+1$ if E matches S at position k; all string symbols in S from position $k+m+1$ to $n-1$ are shifted one position to the right (the symbol at position n is lost). ***Del_1*** will instead delete the symbol '1', if it is present at position $k+m+1$ when E matches S at position k; the string symbols are shifted in the inverse direction (the symbol at position n is randomly selected from V).
7. Let the *concentration* of a family of editors $(\mathcal{E}, \mathcal{F})$ be defined by $\mathbb{C} = \{v_1, v_2, ..., v_l\}$, where v_i represents the average number of editors (E_i, f_i) per V-string of a population $\boldsymbol{P}$. If n_p is the number of V-strings in $\boldsymbol{P}$, then there will be $v_i \cdot n_p$ editors (E_i, f_i) randomly distributed by the n_p V-strings of $\boldsymbol{P}(g)$.

The algorithm for the CGA with string editing is essentially the same as the regular GA, except that step 2 is now more complicated. Code ϕ no longer produces solutions x straight from the V-Strings S of $\boldsymbol{P}$, but from intermediate V-Strings S' obtained from $\boldsymbol{P}$ after editing. **Step 2** is now redefined as:

a. Take each V-String S of $\boldsymbol{P}$. Select each editor (E_l, f_l) from $\mathcal{E}$. With probability v_l from $\mathbb{C}$, edit $S$with E_l. If E_l matches S, then operate (edit) S with f_l to obtain S'.
b. Decode x from S' (not S) and calculate its fitness. This is the fitness of individual (S, x).

Figure 13 shows the operational layout of this CGA with string editing. Generally, we have a population $\boldsymbol{P}$ of n_p maxistrings, and a family of l editors with different concentrations. Before the maxistrings can be translated into the space of solutions $\boldsymbol{X}$, by the mapping ϕ, they must "pass" through successive layers of editors, present in different concentrations. At each generation, the same number of editors (given by the concentrations) is randomly distributed over these layers. Thus, in the example of figure 13, editor 1 (E_1, *Add_1*) with a concentration of 0.5, will have $n_p/2$ copies of itself randomly distributed by the n_p positions of its layer; there will be on average 0.5 of such editors 'meeting' each maxistring. When an editor meets a maxistring, and its ministring matches some subsequence of the maxistring, the editor's function is applied and the maxistring is altered.

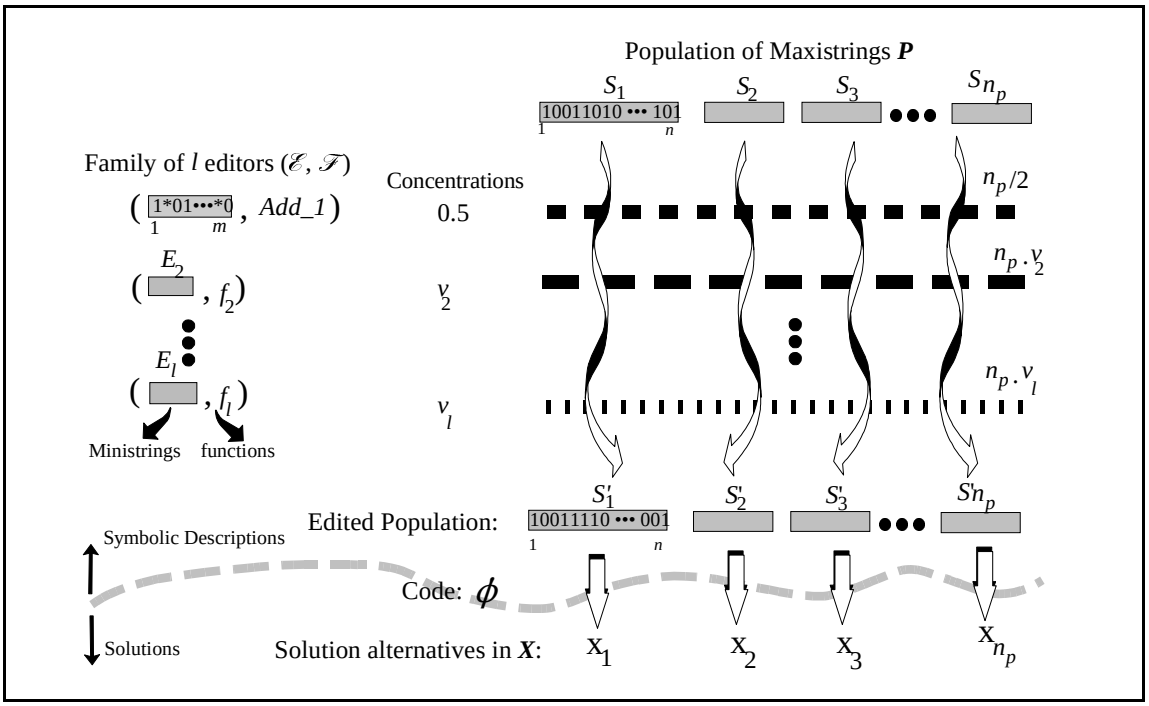

Figure 14: CGA with String Editing

4.3 Context and Evolutionary Systems

In biological genetic systems RNA editing regulates gene expression. Somehow, organisms have used the edition of mRNA molecules to their advantage, perhaps by linking it to environmental context. If a particular external event has the effect of changing the concentrations of editing agents in some genetic system, then those genes which are able to produce fit phenotypes in the different contexts will be selected. Notice that changing environmental context will not merely affect the concentration of editing agents, but also, potentially, the fitness landscape of the genetic system. Thus, the ability to link changes in the environment with internal parameters such as concentrations of editing agents, gives organisms an adaptive advantage as gene expression can become contextually regulated. The idea is the introduction of the second kind of semantic relation leading to a second type of symbol described in section 2.1. The editing strings are now more than symbolic constraints, they are also semantically related to context variation through a (postulated) code γ.

Figure 14 shows precisely this kind of coupling between environmental context and the regulating effects of editor concentrations. Notice, at the bottom of the figure, the dependence of the fitness landscape of the solution alternative space ***X***, on environmental context. When the context changes, not only are the symbolic descriptions edited differently, but the solution alternatives are also evaluated differently. The inclusion of this extra level of semantic relations and pragmatic evaluations establishes the kind of genetic semiotics described in section 2.1.

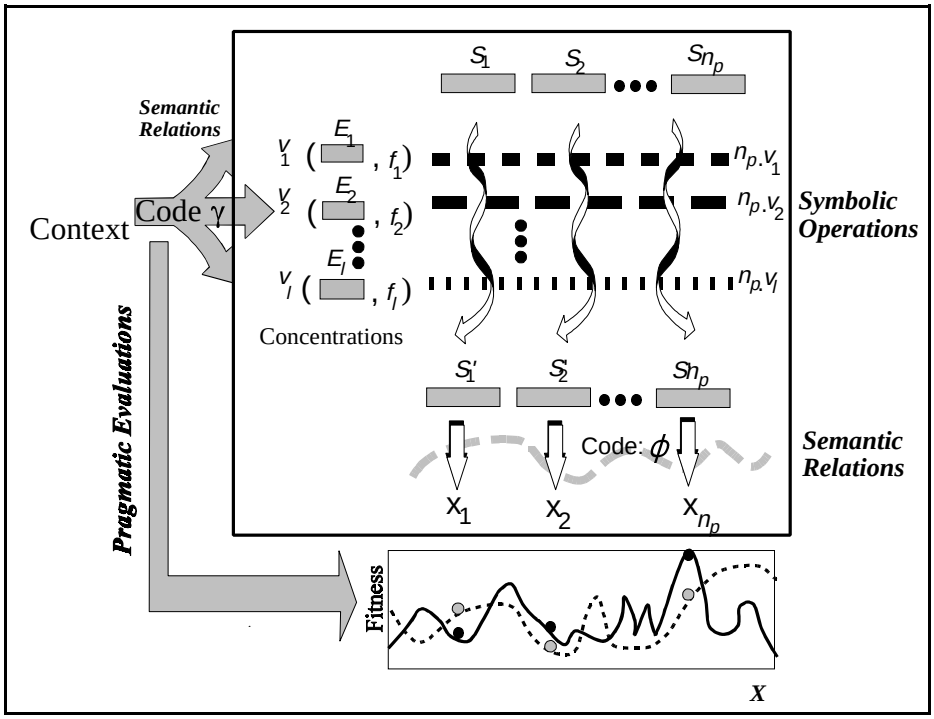

Figure 15: CGA with Editing Strings Linked to Context which affects Fitness

Consider now two sets of concentrations $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ of our family of editors ($\mathscr{E}$, $\mathscr{F}$) linked respectively to two evaluation functions, *fitness1* and *fitness2*. When the first context is at play, we obtain a population of solution alternatives $\mathscr{A}_1$ which will be evaluated by *fitness1*; alternatively, when the second context is at play, $\mathscr{A}_2$ is evaluated by *fitness2*. Notice that both $\mathscr{A}_1$ and $\mathscr{A}_2$ are produced from the same population of symbolic descriptions $\boldsymbol{P}$. Those symbolic descriptions in $\boldsymbol{P}$ which tend to produce fit solution alternatives in both $\mathscr{A}_1$ and $\mathscr{A}_2$ (evaluated by *fitness1* and *fitness2* respectively) will have a higher probability of being selected. This result will of course be dependent on the timing and sequence of application of contexts: if contexts are alternated rapidly, then it will be possible to have symbolic descriptions, with a high probability of selection in the population, which produce fit solutions in only one of the contexts; if contexts are maintained a bit longer before alternating, those symbolic descriptions that tend to produce fit solutions in both contexts will have a higher probability of selection; if the contexts are maintained too long, however, it will be more difficult to evolve symbolic descriptions able to survive in both contexts. These results are trivial to obtain in computer simulations and follow Levins' [1968, chapter 2] strategies of adaptation. This kind of CGA may be useful in situations where the evaluation of solutions changes dramatically according to context. I do not pursue the computational description of this model further in this dissertation, because the computational explorations of CGA's are dedicated to the development portion of CGA's presented in the next section and chapter 5. The existence and utility of syntactic extensions of genetic semiotics are validated with the knowledge we possess of the RNA editing system discussed above.

5. Development and Material Constraints

In chapter 2 (section 1.5.3) I equated the notion of embodiment with von Neumann's parts problem. This aspect of evolutionary systems lies on the semantic area of the semiotics of the genetic system. In figure 2 it is depicted in the bottom part of the semantic relations axis, that is, the development of a phenotype from amino acid chains. Especially in computational realms, we tend to think of the genetic system as a one-to-one mapping of genetic strings to completed phenotypes or solutions to a problem. However, as emphasized earlier (section 2.2 and chapter 1), biological genetic strings encode amino acid strings that will themselves self-organize (fold and subsequently engage in some developmental process) into a final product that is not explicitly genetically encoded — if it were, genotypes would have to be tremendously larger. This fabulous information compression is achieved by utilizing powerful dynamic building blocks, the amino acids, whose physical characteristics do not require encoding.

To explore these ideas computationally, we need to use genetic algorithms that code for some computational building blocks whose (computational) dynamic characteristics do not require genetic encoding, and which will self-organize into a final solution not explicitly encoded. The self-organization of solutions from encoded descriptions is an instance of the process of emergent morphology/classification as discussed in chapter 1. In the following I discuss some approaches to achieve models of this selected self-organization. I also propose a scheme in which the computational building blocks are represented by fuzzy sets to be implemented in chapter 5.

5.1 Development in Artificial Life

Development refers to those processes taking over an organism once it is reproduced and which are responsible for the transformation of its form. Generally, artificial life models of development are based on Wilson's [1988] ideas: a GA will encode "a *production system program* (*PSP*) consisting of a finite number of production (condition-action) rules [...] of the form: $X + K_i \Rightarrow K_j K_k$. The *K*'s stand for cell phenotypes and *X* represents the local context". [Ibid, page 159]. Basically, the symbolic descriptions of the GA code for a population of "mother cells", or "eggs". These "eggs" code for a specific PSP (a set of production rules) dictating how the "cell" develops into some multicellular aggregate, which is then evaluated for its fitness. The more fit aggregates will have the symbolic description of its "egg" reproducing with a larger probability in the population. These ideas have been used mostly to generate neural networks [Kitano, 1990; Belew, 1993; Gruau, 1992] or more generally sensorimotor control systems [for a good overview see Husbands et al, 1994].

Lately much attention has been posited on evolutionary strategies that bring together self-organizing systems and natural selection inspired algorithms. Particularly in the field of Artificial Life, Kitano[1994], and Dellart and Beer [1994], have proposed GA's which do not encode directly their solutions, but rather encode generic rules (through L-Systems) which develop into boolean networks simulating metabolic cycles. With these approaches, GA's no longer model exclusively selection, but also a self-organizing dimension standing for some materiality. The GA does not search the very large space possible solutions, but a space of basic rules which can be manipulated to build different self-organizing networks. These networks are then started (sometimes with some learning algorithm) and will converge to some attractor behavior standing for a solution of our simulation. Rather than directly encoding solutions, the GA harnesses a space of possible self-organizing networks which will themselves converge to a solution — emergent morphology.

Usually such indirect encoding schemes for genetic algorithms are based on the encoding of generic rules for developing dynamic systems, e.g. boolean networks [Dellaert and Beer, 1994] or neural networks [Kitano, 1994, 1995], which will themselves self-organize into final solutions. The primary advantage of indirect encoding GA's is the information compression of encoded solutions into smaller chromosomes. The

GA does not search the very large space of possible solutions, say, the set of weights of a large neural network (see chapter 5), but a space of generic rules which self-organize into solutions (usually L-Systems that produce large neural networks). Indirect encoding in GA's is an attempt to utilize the advantages of embodiment discussed earlier in a computational realm.

The semiotic genetic system does not encode every detail of the obtained solutions, rather it encodes a development scheme which relies on the pre-existence of rich enough building blocks that do not require a description. In biological systems these building blocks are amino acids whose dynamical characteristics genes do not need to encode as they "come for free" with the laws of matter. In the computational realm, we can ease the chromosomes of a GA from having to describe every detail of the solutions through an indirect encoding scheme. However, some form of that description is unavoidable somewhere else in a computer implementation, since computer programs require full specification by definition. It is therefore important to have as simple as possible a description of the dynamic building blocks for the indirectly encoded GA solutions. If a true computational dynamic system, such as a boolean network, is used, every time a chromosome is decoded into a set of rules to build the network that will self-organize into a solution [Dellaert and Beer, 1994], the network will actually have to be implemented and run for a number of cycles in all its details. Thus at each step of the GA, the evaluation of a chromosome relies on a computationally demanding evaluation procedure that must implement and observe the dynamic behavior of a network, which many times develops into long transient cycles. For this reason I have developed an indirect encoding scheme which utilizes fuzzy sets as representations of the states of dynamical systems presented next.

5.2 Fuzzy Development Programs: Emergent Classification in Contextual Genetic Algorithms

5.2.1 Fuzzy Sets as Dynamical States

Fuzzy sets [Zadeh, 1965] are extensions of classical sets which allow members of a set to have a degree of inclusion in it. That is, rather than an element being an element of a set or not, it is an element to a degree defined in the unit interval. '0' represents complete non-inclusion and '1' full inclusion. Mathematically, fuzzy sets are simply unconstrained mappings to the unit interval. Unconstrained because the degree of membership of an element in set is not constrained by the membership value of another element (Fuzzy Sets are introduced in more mathematical detail in Chapter 3). Even though fuzzy sets were developed with a natural language interpretation, for the purposes of the remaining of this chapter, I will consider simply their powerful formal characteristics and give them a totally distinct interpretation. I wish to emphasize that by no means does this imply a disagreement with fuzzy logic's interpretation that I pursued in chapters 3 and 5, but a mere pragmatic desire to use a fully developed formal edifice that I believe to be very useful to tackle the problems discussed in this chapter. In other words, I am using fuzzy sets simply as mappings to the unit interval.

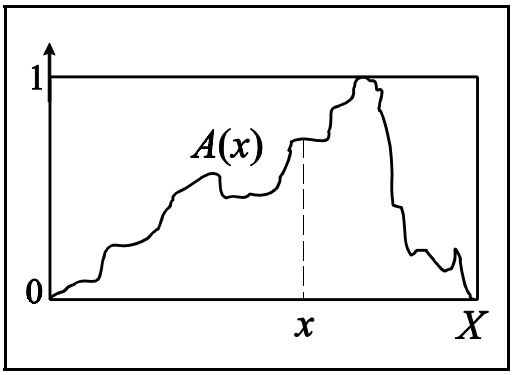

Figure 16: Fuzzy Set as the state of a dynamical system

Since the interval of membership of an element in a fuzzy set is the real unit interval, it is clear that a fuzzy set is a mathematical structure which is able to capture the state of any system whose components's states are defined by

numerical values, inasmuch as any real interval can be mapped to the unit interval. In other words, the elements of the set represent components of some system which take values in the real unit interval. Formally, we start with the definition of a universal set X containing all the elements (variables) x of system X. A fuzzy set A is defined by the (membership) function $A(x)$: $X \rightarrow [0,1]$, and it represents a particular state of system X (figure 15). A dynamical system is a system with a state-determined transformation f between subsequent states, that is, a transformation which relies only on the present state of a system, $A_t(x)$, and optionally some set of numerical inputs I_t, to calculate its next state: $A_{t+1}(x) = f(A_t(x), I_t)$. Therefore, the state of any dynamical system of several components can be represented by a fuzzy set A, and a general dynamical system is defined by a fuzzy set function f on the universal set X.

The function f, to yield a dynamical system with complex, nonlinear, behavior needs to be much more complicated than a simple fuzzy logic connective such as intersection and union, since these are fully linear as they compute the next value of x independently from the values of other elements of X. f must define a complicated convolution of the fuzzy set in terms of some general rule. The conventional convolution operation, primarily used in signal processing, sums the multiplication of each element of a signal by each element of a reference pattern (mask), in other words, it weights the value of each element regarding the mask, and then aggregates all the weights. It is fairly easy to extend this notion to fuzzy logic to obtain nonlinear convolutions [e.g. Lee et al, 1996], by substituting multiplication and summation by appropriate fuzzy operations and signals by fuzzy sets. Thus a fuzzy convolution, FCONV, of fuzzy set $A(x)$ by a fuzzy mask $M(x)$ yields fuzzy set $B(x)$: $\mathrm{FCONV}(A(x), M(x)) = B(x) = \oplus_{n=-\infty}^{+\infty}\left(M(n) \otimes A(x-n)\right)$, where $\otimes$ and $\oplus$ are appropriate fuzzy operations from the large family of fuzzy connectives. FCONV depends on the fuzzy connectives chosen as well as the mask selected. If we construct function f of the definition of dynamical system above as a sequence of fuzzy convolutions with different fuzzy connectives and masks, then such dynamical systems can be as complicated as we desire.

The definition of a dynamical system as a sequence of fuzzy convolutions is interesting and can potentially offer a general mathematical formalism to deal with complex systems, but that is not the scope of the work here pursued. Unless we wish to study the general mathematical properties of dynamical systems, there is little advantage in using the fuzzy set formalism above instead of the real thing. If one desires to implement say, a boolean network, it is much more intuitive to do it directly than with the global perspective offered by the fuzzy convolution scheme. What is relevant for the remaining of this section is that the state of a dynamical system of several components can be represented by a fuzzy set.

5.2.2 Fuzzy Development Programs

Consider an initial fuzzy set $A_0(x) = 0.5$ for all x of X, it is the initial condition of our system X. 0.5 is the furthest point away from 0 and 1. If 0 means 'off' and 1 means 'on', then in fuzzy logic's terms, 0.5 is the most uncertain point. Consider now a sequence of n fuzzy sets $F_1, F_2, \cdots, F_n$, which will be applied to A_0 with the sequence of n fuzzy operations (also referred to as connectives) $\odot_1, \odot_2, \cdots, \odot_n$, that is, F_1 is applied to A_0 with $\odot_1$, yielding A_1, which in turn is applied F_2 with $\odot_2$, yielding A_2, and so forth. This sequence of n fuzzy sets F_i and operations $\odot_i$ develops system X from state A_0 to state A_n. It is therefore a program to develop $A_n(x)$ from $A_0(x)$ in n steps: a *fuzzy development program* (FDP). It leads system X through a sequence of states represented as a sequence of n fuzzy sets $A_1, A_2, \cdots, A_n$ on a universal set X. Notice that if some of the operations $\odot_i$ are non-commutative, then the sequence of the FDP matters, which is desired of developmental models. For instance, the fuzzy logic connective $A \cap \overline{B}$ is not commutative. If it is in the pool of operations $\odot_i$, the permutations of the order of application of the FDP will yield different final states A_n.

For the purposes of this section, the FDP sequence does not need to be the result of a dynamical system defined as a fuzzy convolution function outlined above. All we need to know is that there is a system

X which undergoes a process of development defined by the sequence of n states $A_i(x)$. The precise dynamic mechanism that causes state A_n to develop from A_0 is unknown. We do know however that by applying the sequence of n fuzzy sets F_i and operations $\odot_i$ to A_0 we obtain A_n. It is a higher level knowledge of the observed or desired characteristics of system X, but not of the nature of its lower level dynamics. As explored in section 5.2.1, we can also study the lower-level dynamics in this fuzzy set framework with fuzzy convolution operations, but for my purposes here of linking development and embodiment to evolutionary strategies, it is not necessary to define any dynamics, only its outcomes through a sequence of states represented by fuzzy sets. FDP's are not dynamical systems, but higher level representations of dynamical development.

The advantage of using fuzzy sets to represent states of dynamical systems lies in the ability to use a very large pool of existing fuzzy logic operations in order to define the transformations between developmental stages. That is, operations $\odot_i$ that transform fuzzy set A_0 to A_n can be easily constructed with the whole edifice of fuzzy logic connectives. Furthermore, the fuzzy sets F_i and operations $\odot_i$ can be regarded as higher-level representations of known global transformations to system X observed in our models. In this sense, fuzzy sets may offer a more intuitive way of thinking about the development of system X, than the lower-level (convoluted) interactions of components.

5.2.3 Information Requirements of Fuzzy Development Programs

Consider a small pool of n_F typical fuzzy set shapes referred to as $\mathscr{F}$. These shapes include the traditional trapezoidal fuzzy set shapes of fuzzy control and other simple shapes (some examples are shown in figure 16, details in the implementation of this scheme in Chapter 5). Consider a small pool of n_O fuzzy set operations referred to as $\mathscr{O}$. These operations range from commutative operations such as fuzzy union ($\cup$) and intersection ($\cap$) to non-commutative operations such as $A \cap \overline{B}$ and $\overline{A} \cup B$. A FDP is a sequence (string) S of n fuzzy sets $F_i \in \mathscr{F}$ and n operations $\odot_i \in \mathscr{O}$: $S = \odot_1 F_1 \odot_2 F_2 \cdots \odot_n F_n$ which are to be applied in sequence to the initial state $A_0(x)$ of system X.

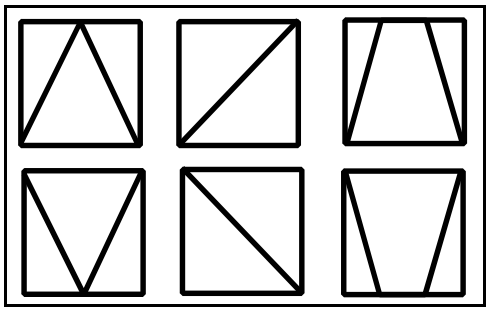

Figure 17: Examples of Simple Fuzzy Set Shapes

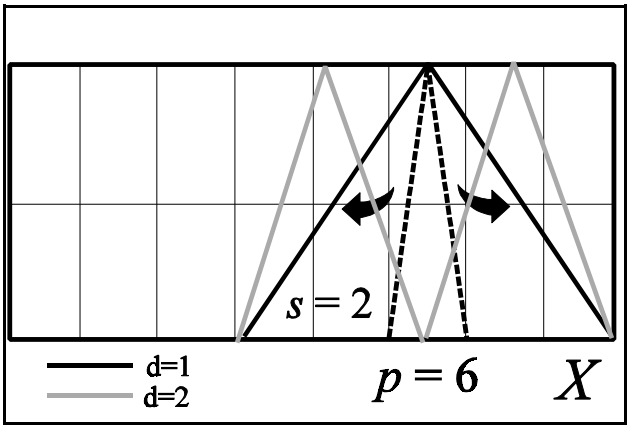

Figure 18: Triangular Fuzzy Set Shape Applied to Sixth Octant of X (p=6) and stretched 2 octants to each side (s=2). Shown for two division parameter (d=1 and d=2).

Notice that the fuzzy sets $F_i \in \mathscr{F}$ are not yet properly defined. That is, they were defined in terms of a set of shapes $\mathscr{F}$, but to be proper fuzzy sets, they must also be defined on a particular universal set. The universal set of our system is X, the set of all components (variables) x of a dynamical system. As a first approach, the fuzzy shapes of $\mathscr{F}$ could be defined over X, but let us increase the flexibility of FDP's by defining a partition of X in an even number n_X of parts. If $n_X = 8$, X is divided in equal octants. Now each F_i of the FDP S is associated with a specific part of X: $p = 1, \cdots n_X$. In other words, each shape F_i is applied to part p of X through operation $\odot_i$. In addition, we can create an extra parameter $s = 1, \cdots n_X/2$, which represents the

number of parts of X that shape F_i should be stretched over. Figure 17 shows the universal set X divided in octants (n_X = 8), and a triangular fuzzy shape (dotted line) being applied to the sixth octant (p = 6) with a stretch of two octants for each side of the sixth octant (s = 2). A final parameter d = 1, ··· $n_X/2$ is defined which represents the number of times shape F_i is going to be repeated in the portion of X given by p and s. In figure 17, the dark line represents d = 1, meaning that the triangular shape is repeated once. If d were 2, then the triangular shape would be narrowed in half, and repeated twice over the interval of X given by parts 4 to 8 (p=6, s=*2*), in figure 17 this is represented by the lighter line.

Each fuzzy set F_i of the FDP S is defined by a fuzzy set shape from $\mathcal{F}$, the part of X, p, the stretch s, and the division factor d. Notice that the information required to describe the FDP S does not depend on the size of X but on the parameters n_X, n_F, and n_O, since we only need to identify the portion of X where the shape of $\mathcal{F}$ is going to be stretched, divided, and applied with an operation of $\mathcal{O}$. To specify the part of X chosen (e.g. octant), no information regarding the elements of X in that part is required. We need only $\log_2 n_F$ and $\log_2 n_O$ bits of information to identify n_F fuzzy set shapes and n_O operations from $\mathcal{F}$ and $\mathcal{O}$ respectively. We further need $\log_2 n_X$ bits to describe the position parameter p, and $2\times\log_2 (n_X/2)$ bits to describe the stretch parameter s and division factor d. Therefore, $\log_2 n_F + \log_2 n_X + 2\times\log_2 (n_X/2)$ bits are required to identify a fuzzy set F_i in a FDP S, and $\log_2 n_O$ bits to identify its associate operation. For example, if there are 16 possible fuzzy set shapes ($|\mathcal{F}|$=16) and 16 possible fuzzy logic operations ($|\mathcal{O}|$=16), and X is divided in 16 parts, then $\log_2 16 + \log_2 16 + 2*\log_2 8 + \log_2 16 = 4 + 4 + 6 + 4 = 18$ bits are required for each pair fuzzy set/operation in the FDP. If the length of the FDP is n = 8, then the FDP S requires 144 bits to be described, that is, a 144 long bit string. Notice that this value is independent of the cardinality of X or its parts. In summary, a FDP S requires the following number of bits of information to be described for any X:

$$\nu = n\left[\log_2 n_F + \log_2 n_X + 2\cdot\log_2\left(\frac{n_X}{2}\right) + \log_2 n_O\right] \qquad \textbf{(1)}$$

Assuming a constant small pool of fuzzy set shapes and fuzzy logic operations, as well as a constant partition of X, it is possible to construct a fuzzy set of X with a constant amount of information ν. The ability to approximate any fuzzy set of X is naturally dependent on the length of the FDP's used and on the richness of its building blocks: the fuzzy set shapes and operations of $\mathcal{F}$ and $\mathcal{O}$. In other words, we can potentially create a language to describe a system X of many components with a relatively small FDP. In the previous example, if the cardinality of system X is 100, we may be able to describe its states (fuzzy sets) with only 144 bits of information. Furthermore, the components of X are continuous type variables[32] defined on the unit interval, while the FDP description is binary (144 bits). In computational terms, the components of X are real-type variables which usually require 4 to 10 bytes (32 to 80 bits) each. The computational description of a small number of components easily surpasses the binary description of a FDP by several orders of magnitude. For example, 100 real type components require 3200 to 8000 bits of information. Specific application examples are discussed in chapter 5.

[32] Discrete type variables can also of course be represented in FDP's as fuzzy sets can be defuzzified into crisp sets. An example is discussed in chapter 5.

5.2.4 General Purpose Genetic Algorithm with Developmental Indirect Encoding: Emergent Classification

Utilizing the scheme above, states $A(x)$ of system X can be evolved (regarding some fitness function) through a genetic algorithm (GA) which codes not for the states themselves, but for FDP's whose sequences of operations produce $A(x)$. The chromosomes of the GA are bit strings of length ν encoding FDP programs, which develop into (continuous) solutions $A(x)$ that are not directly encoded in the bit strings (figure 18). Usually indirect encoding schemes for GA's are based on the encoding of generic rules for developing dynamic systems, e.g. boolean networks [Dellaert and Beer, 1994] or neural networks [Kitano, 1994, 1995], which will themselves self-organize into final solutions. The primary advantage of indirect encoding GA's is the information compression of encoded solutions into smaller chromosomes. The GA does not search the very large space of possible solutions, but a space of generic rules which can be manipulated to build different self-organizing systems. The fuzzy scheme above is more general since it is not restricted to a particular dynamical system (such as boolean networks or neural networks), but can instead represent the states of any dynamical system. Furthermore, it defines continuous variable dynamical systems as opposed to, for instance, strictly discrete boolean networks. FDP's convert binary strings into continuous fuzzy sets given a predefined pool of building blocks which ultimately define the scheme's effectiveness. As shown in figure 18, the decoding of FDP's into fuzzy sets can also be dependent on contextual inputs, if the fuzzy set operations used allow that sort of input. Thus, FDP GA's are also CGA's.

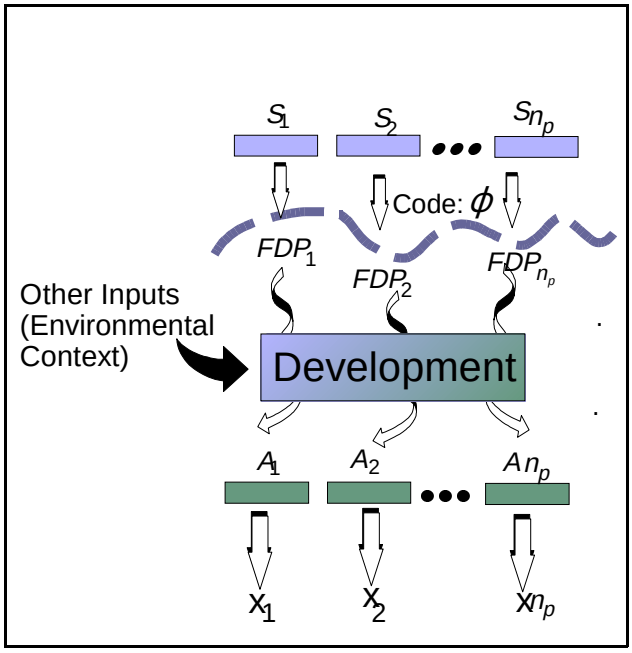

Figure 19: CGA with Developmental Stage based on FDP's. Chromosomes encode FDP's which develop into Fuzzy Sets, standing for Solutions.

Indirect encoding in GA's is an attempt to utilize the advantages of emergent morphology/classification discussed in chapter 1 and section 2.2 of this chapter in a computational realm. The semiotic genetic system does not encode every detail of the obtained solutions, rather it encodes a development scheme which relies on the preexistence of rich enough building blocks. In other words, it harnesses a small subset of dynamical parts into a solution good enough given a fitness function (pragmatics). It relies on the existence of building blocks that do not require a description. In biological systems these building blocks are amino acids whose dynamical characteristics genes do not need to encode. In the computational realm, we can ease the chromosomes of a GA from having to describe every detail of the solutions through an indirect encoding scheme as discussed above. However, some form of that description is unavoidable somewhere else in a computer implementation as everything must be specified in computational environments. In other words, emergent classification in the computer realm, requires the simulation of materiality in order to implement selected self-organization. In this case, the material dynamic constraints are simulated with FDP's.

It is therefore important to have as simple as possible a description of the dynamic building blocks for the indirectly encoded GA solutions. The fuzzy set scheme above is an attempt precisely at that. If a true

computational dynamic system, such as boolean networks, is used, every time a chromosome is decoded into a set of rules to build the network that will self-organize into a solution [Dellaert and Beer, 1994], the network will actually have to be implemented and run for a number of cycles in all its details. Thus at each step of the GA, the evaluation of a chromosome relies on a computationally demanding evaluation procedure that must implement and observe the dynamic behavior of a network. By contrast, the FDP is not a dynamical system, it is a sequence dependent (non-commutative) procedure for constructing the state of a dynamical system without actually running it. FDP's depend on a small vocabulary of simple parts, such as fuzzy set shapes and operations, which are described by very simple equations. Notice that the fuzzy set shapes are not defined on the universal set *X*, thus computationally we do not have to store its precise values regarding a large *X* until the moment they are applied, at which time they are calculated from the simple equations. Moreover, a FDP has precisely *n* steps of operation, unlike true dynamical systems whose components interact in long cycles until some form of stability is reached.

FDP's offer a simulation of true developmental dynamics which preserves some important characteristics of development such as being constructed from a small pool of (simulated material) parts which interact with one-another through a development program in order to define a final state. Since the operations between these parts can be non-commutative, the order of application of parts is important as it leads to different final results. Though not the real thing, they allow a transformation from a small discrete, boolean, domain, to a large continuous domain without very computationally expensive implementations of true dynamical systems. Furthermore, they can represent the state of any dynamical system, discrete or continuous. Coupling them to GA's, allows the establishment of computational counterparts of emergent classification and selected self-organization with the immediate advantage of tremendous genetic information compression. Some computational results of the FDP GA are discussed in chapter 5.

5.2.5 Computational Issues: Fuzzy Indirect Encoding as Solution Approximation

One final word in this section regarding modeling and simulation issues. The indirect encoding GA scheme decodes chromosomes into dynamical systems (or simulations of) which require another simulation code to define what the dynamical states mean. For instance, the nodes of Dellaert and Beer's [1994] boolean networks obtained from a GA, stand for such attributes as cell division or change of state. In the case here developed, fuzzy sets are obtained from the indirect encoding of FDP's in a GA, but these fuzzy sets have to be given an interpretation in our simulation. Indirect encoding requires more attention to the hierarchy of levels of simulation in computer models of selected self-organization. Chromosomes decode into FDP's, which based on a pool of pre-defined parts, construct a fuzzy set describing the state of a particular dynamical system according to a second simulation code.

Depending on the problems to be solved, the language of fuzzy set shapes and operations, as well as its accuracy (e.g. partition of the universal set) will be the dictating factors for the efficiency of this fuzzy indirect encoding scheme. As it will become clear in chapter 5 with the discussion of specific applications, the building blocks of FDP's are rough approximating functions which are combined to yield solutions to a problem. The rougher the approximations, the rougher the final solution. If only very simple fuzzy set shapes are used, or only a very small number of partitions of the universal set is chosen, then naturally the evolved FDP's will be operating on a rough simplification of the space of solutions. In other words, large portions of the space of solutions will be out of reach to the genetic algorithm. In many cases of extremely large, rough, search spaces such simplifications might be the only avenue to reach fairly good solutions due to computational limitations

The dependence on the characteristics of the building blocks is not surprising, since as discussed in chapter 1 and section 2.2 of this chapter, it is the other side of selected self-organization and evolutionary constructivism. The material building blocks used in evolving symbol systems constrain its universe of

classification or construction. The richer these building blocks, the larger this universe. Indirect encoding schemes aim at the simulation of material building blocks in computational realms, likewise, the richness of the computational building blocks is a constraining factor for evolved solutions. However, unlike material systems who merely have to *use* building blocks, computational systems must at some level *describe* those building blocks which poses serious limits to the complexity of usable building blocks. Simple fuzzy sets organized into FDP's seem to offer a simple enough language to describe complicated solutions as shown in chapter 5.