2 The Entities of Gene Expression Programming

In contrast to its analogous cellular gene expression, GEP gene expression is rather simple. The main players in gene expression programming are only two: the chromosomes and the expression trees, the latter consisting of the expression of the genetic information encoded in the former. The process of information decoding (from the chromosomes to the expression trees) is called translation. And this translation implies obviously a kind of code and a set of rules. The genetic code is very simple: a one-to-one relationship between the symbols of the chromosome and the functions and terminals they represent. The rules are also quite simple: they determine the spatial organization of the functions and terminals in the expression trees and the type of interaction between sub-expression trees in multigenic systems.

Therefore, there are two languages in gene expression programming: the language of the genes and the language of the expression trees, and we will see that the sequence or structure of one of these languages is more than sufficient to infer exactly the other. In nature, although the inference of the sequence of proteins given the sequence of genes and vice versa is possible, very little is known about the rules that determine the folding of the protein. And the expression of a protein gene is not complete before the folding of the protein, that is, strings of amino acids only become proteins when they are correctly folded into their native three-dimensional structure. The only thing we know for sure about protein folding is that the sequence of the amino acids determines the folding. However, the rules that orchestrate the folding are still unknown. Fortunately for us, in GEP, thanks to the simple rules that determine the structure of expression trees and their interactions, it is possible to infer immediately the phenotype (the final structure, which is equivalent to the folded protein molecule) given the sequence of a gene, and vice versa. This bilingual and unequivocal system is called *Karva* language. The details of this language are explored in this chapter.

Cândida Ferreira**:** *Gene Expression Programming*, Studies in Computational Intelligence (SCI) **21**, 29–54 (2006)
www.springerlink.com

2.1 The Genome

In gene expression programming, the genome or chromosome consists of a linear, symbolic string of fixed length composed of one or more genes. Despite their fixed length, we will see that GEP chromosomes code for expression trees with different sizes and shapes.

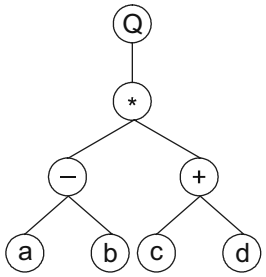
2.1.1 Open Reading Frames and Genes

The structural organization of GEP genes is better understood in terms of open reading frames (ORFs). In biology, an ORF, or coding sequence of a gene, begins with the start codon, continues with the amino acid codons, and ends at a termination codon. However, a gene is more than the respective ORF, with sequences upstream of the start codon and sequences downstream of the stop codon. Although in GEP the start site is always the first position of a gene, the termination point does not always coincide with the last position of a gene. It is common for GEP genes to have noncoding regions downstream of the termination point. (For now we will not consider these noncoding regions, as they do not interfere with the product of expression.)

Consider, for example, the algebraic expression:

$$\sqrt{(a-b)\times(c+d)} \tag{2.1}$$

It can also be represented as a diagram or expression tree (ET):



where "Q" represents the square root function.

This kind of diagram representation is in fact the phenotype of GEP genes, the genotype being easily inferred from the phenotype as follows:

```
01234567
Q*-+abcd
```
(2.2)

which is the straightforward reading of the ET from left to right and from top

to bottom (exactly as we read a page of text). The expression (2.2) is an ORF, strarting at "Q" (position 0) and terminating at "d" (position 7). I named these ORFs *K-expressions* (from Karva language).

Note that this notation differs from both the postfix and prefix representations used in different GP implementations with arrays or stacks (Keith and Martin 1994). Figure 2.1 compares Karva notation both with postfix and prefix expressions.

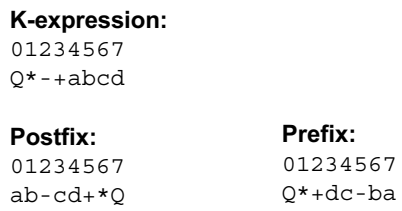


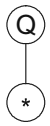
Figure 2.1. Comparison of Karva notation with both postfix and prefix representations. In all cases, the expression (2.1) is represented.

Consider another ORF, the following K-expression:

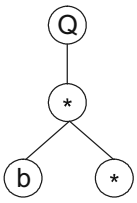
```
01234567890
Q*b**+baQba                                    (2.3)
```

Its expression as an ET is also very simple and straightforward. For its complete expression, the rules governing the spatial distribution of functions and terminals must be followed. First, the start of the ORF corresponds to the root of the ET (this root, though, is at the top of the tree), forming this node the first line of the ET. Second, depending on the number of arguments of each element (functions may have a different number of arguments, whereas terminals have an arity of zero), in the next line are placed as many nodes as there are arguments to the elements in the previous line. Third, from left to right, the new nodes are filled consecutively with the elements of the ORF. This process is repeated until a line containing only terminals is formed. So, for the K-expression (2.3) above, the root of the ET will be formed by "Q", the symbol at position 0:

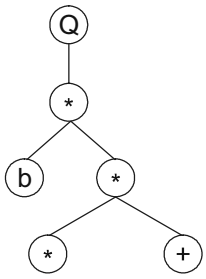
The square root function has only one argument, so the next line requires only one node, which is filled with the next symbol at position 1, that is, "*":



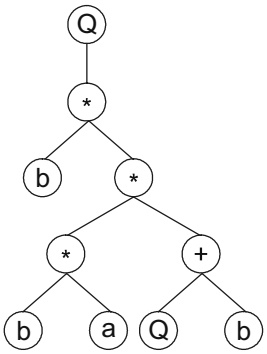
The multiplication function takes two arguments and, therefore, in the next line are placed two more nodes. These nodes are then filled with the symbols at positions 2 and 3, that is, “b” and “*”, obtaining:



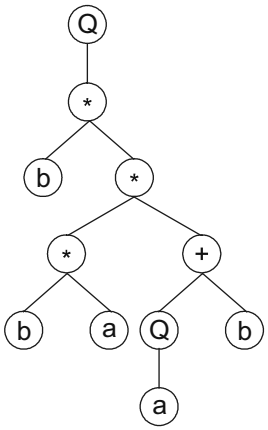
In this line we have a leaf node and a bud node representing a function of two arguments (multiplication). Therefore two more nodes are required to build the next line. And in this case, they are filled with the elements at positions 4 and 5, namely “*” and “+”, giving:



Now we have a line with two function buds, each expanding into two more branches. Thus, four new nodes are required in the next line. And they are filled with the elements “b”, “a”, “Q”, and “b”, occupying respectively positions 6, 7, 8, and 9 in the ORF, obtaining:



In this new line, although there are four nodes, just one of them is a bud node, whereas the remaining three are leaf nodes. From the leaf nodes obviously no more growth will be sprouting, but from the bud node another branch will be formed. Thus, the required branch is placed below the bud node and filled with the next element in the ORF, the symbol "a" at position 10:



With this step, the expression of the K-expression (2.3) is complete as the last line contains only nodes with terminals. We will see that, thanks to the structural organization of GEP genes, the last line of all ETs generated by this technique will contain only terminals. And this is equivalent to say that all programs evolved by GEP are syntactically correct. Indeed, in GEP, there is no such thing as an invalid expression or computer program.

Looking at the structure of ORFs only, it is difficult or even impossible to see the advantages of such a representation, except perhaps for its simplicity and elegance. However, when open reading frames are analyzed in the context of a gene, the advantages of this representation become obvious. As I said before, the chromosomes of gene expression programming have fixed length, and they are composed of one or more genes of equal length. Therefore the length of GEP genes is also fixed. Thus, in gene expression programming, what varies is not the length of genes, which is constant, but the length of the ORFs. Indeed, the length of an ORF may be equal to or less than the length of the gene. In the first case, the termination point coincides with the end of the gene, and in the latter the termination point is somewhere upstream of the end of the gene. And this obviously means that GEP genes have, most of the times, noncoding regions at their ends.

And what is the function of these noncoding regions at the end of GEP genes? We will see that they are the essence of gene expression program-

ming and evolvability, for they allow the modification of the genome through the use of virtually any kind of genetic operator without any kind of restriction, always producing syntactically correct programs. And this opens up new grounds for the exploration of the search space as all the recesses of the fitness landscape are now accessible. Thus, in gene expression programming, the fundamental property of genotype/phenotype systems – syntactic closure – is intrinsic, allowing the totally unconstrained manipulation of the genotype and, consequently, an efficient evolution. Indeed, this is the paramount difference between gene expression programming and previous GP implementations, with or without linear genomes, all of them either limiting themselves to inefficient genetic operators or checking exhaustively all the newly created programs for syntactic errors (for a review on genetic programming with linear genomes see Banzhaf et al. 1998).

Let us now analyze the structural organization of GEP genes in order to understand how they invariably code for syntactically correct computer programs and why their revolutionary structure allows the unconstrained application of virtually any search operator and, therefore, guarantees the generation of the quintessential genetic variation fundamental for the evolution of good solutions.

2.1.2 Structural and Functional Organization of Genes

So, what is so special about the structure of GEP genes? Well, they are composed of two different domains – a head and a tail domain – each with different properties and functions. The head domain is used mainly to encode the functions chosen for the problem at hand, whereas the tail works as a buffer or reservoir of terminals in order to guarantee the formation of only valid structures. Thus, the head domain contains symbols that represent both functions and terminals, whereas the tail is composed of only terminals.

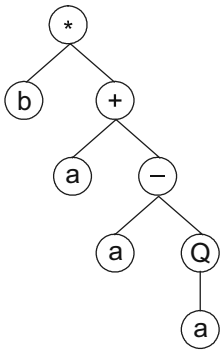
For each problem, the length of the head h is chosen, whereas the length of the tail t is a function of h and the number of arguments of the function with more arguments n_{max} (also called maximum arity) and is evaluated by the equation:

$$t = h \cdot (n_{max} - 1) + 1 \tag{2.4}$$

Consider a gene for which the set of functions F = {Q, *, /,-, +} and the set of terminals T = {a, b}, thus giving $n_{max} = 2$. And if we chose an $h = 15$, then $t = 15 \cdot (2 - 1) + 1 = 16$ and the length of the gene g is 15 + 16 = 31. One such gene is shown below (the tail is shown in bold):

```
01234567890123456789012345678 90
*b+a-aQab+//+b+babbabbbababbaaa
```
(2.5)

It codes for the following ET:

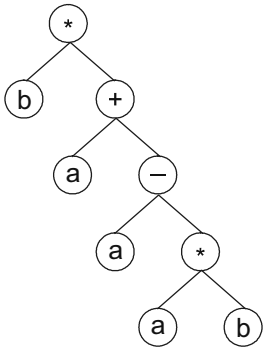


Note that in this case the ORF ends at position 7, whereas the gene ends at position 30 and, therefore, a noncoding region of 23 elements, apparently doing nothing, exists downstream of the termination point.

Suppose now a mutation occurred in the coding region of the gene, specifically at position 6, changing the "Q" into "*". Then the following gene is obtained:

```
01234567890123456789012345678 90
*b+a-a*ab+//+b+babbabbbababbaaa
```
(2.6)

And its expression gives:

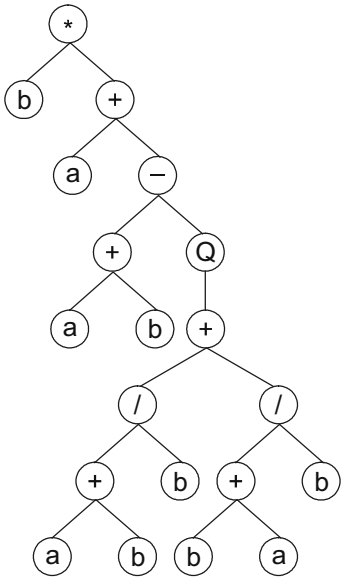


In this case, the termination point shifts just one position to the right (position 8), with the new expression tree managing to keep most of the characteristics of the original one.

Consider another mutation in chromosome (2.5) above, the substitution of "a" at position 5 by "+". In this case, the following daughter chromosome is obtained:

```
01234567890123456789012345678 90
*b+a-+Qab+//+b+babbabbbababbaaa
```
(2.7)

And its expression results in the following ET:

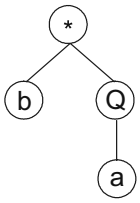


In this case, the termination point shifts 12 positions to the right (position 19), enlarging significantly the new expression tree.

Obviously the opposite might also happen, and the expression tree might shrink. For instance, suppose that a mutation occurred at position 2 in chromosome (2.5) above, changing the "+" into "Q":

```
01234567890123456789012345678 90
*bQa-aQab+//+b+babbabbbababbaaa
```
(2.8)

Now its expression results in the following ET:



In this case, the ORF ends at position 3, shortening the original ET in four nodes and creating a new daughter tree very different from the original one.

So, despite their fixed length, each gene has the potential to code for expression trees of different sizes and shapes, being the simplest composed of

only one node (when the first element of a gene is a terminal) and the largest composed of as many nodes as the length of the gene (when all the elements of the head are functions with maximum arity).

It is evident from the examples above, that any modification made in the genome, no matter how profound, always results in a structurally correct expression tree. Obviously, the structural organization of genes must be preserved, always maintaining the boundaries between head and tail and not allowing symbols from the function set on the tail. We will pursue these matters further in the next chapter (section 3.3) where the mechanisms and the effects of different genetic operators are thoroughly analyzed.

2.1.3 Multigenic Chromosomes

In nature, chromosomes usually code for more than one gene, as complex individuals require complex genomes. Indeed, the evolution of more complex entities is only possible through the creation of multigenic genomes. Not surprisingly, gene expression programming also explores the advantages of multigenic systems.

The chromosomes of gene expression programming are usually composed of more than one gene of equal length. For each problem, the number of genes, as well as the length of the head, are chosen a priori. Each gene codes for a sub-ET and the sub-ETs interact with one another forming a more complex entity. The details of such interactions will be fully explained in section 2.2, Expression Trees and the Phenotype. For now we will focus exclusively on the construction of sub-ETs from their respective genes.

Consider, for example, the following chromosome with length 39, composed of three genes, each with $h = 6$ and a length of 13 (the tails are shown in bold):

```
012345678901201234567890120123456789012
*Qb+*/bbbabab-a+QbQbbababa/ba-/*bbaaaaa          (2.9)
```

It has three open reading frames, and each ORF codes for a particular sub-ET (Figure 2.2). We know already that the start of each ORF coincides with the first element of the gene and, for the sake of clarity, for each gene it is always indicated by position zero; the end of each ORF, though, is only evident upon construction of the respective sub-ET. As shown in Figure 2.2, the first ORF ends at position 9 (sub-ET_1); the second ORF ends at position 6 (sub-ET_2); and the last ORF ends at position 2 (sub-ET_3).

a. 012345678901201234567890120123456789012
Qb+/**bbbabab**-a+QbQ**bbababa**/ba-/***bbaaaaa**

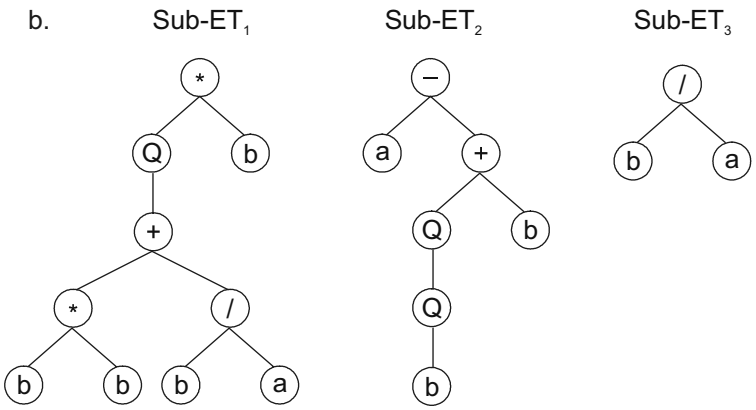


Figure 2.2. Expression of GEP genes as sub-ETs. **a)** A three-genic chromosome with the gene tails shown in bold. Position 0 marks the start of each gene. **b)** The sub-ETs codified by each gene.

In summary, the chromosomes of gene expression programming contain several ORFs, each ORF coding for a structurally and functionally unique sub-ET. We will see that, depending on the problem at hand, these sub-ETs may be selected individually according to their respective fitness (for example, in problems with multiple outputs), or they may form a more complex, multi-subunit ET and be selected according to the fitness of the whole, multi-subunit ET. The patterns of expression and the details of selection will be often discussed in this book. However, keep in mind that each sub-ET is both a separate entity and part of a more complex, hierarchical structure, and, as in all complex systems, the whole is more than the sum of its parts.

2.2 Expression Trees and the Phenotype

In nature, the phenotype has multiple levels of complexity, being the most complex the organism itself. But tRNAs, proteins, ribosomes, cells, and so forth, are also products of expression and all of them are ultimately encoded in the genome.

In contrast to nature, in gene expression programming, the expression of the genetic information is very simple. Nonetheless, as we have seen in

section 2.1, GEP chromosomes can have different structural organizations and, therefore, the individuals encoded in those structures have obviously different degrees of complexity. The simplest individuals we have encountered so far are encoded in a single gene, and the "organism" is, in this case, the product of one gene – an ET composed of only one subunit. In other cases, the "organism" is a multi-subunit ET in which the different subunits are linked together by a particular linking function. And in other cases, the "organism" is composed of different sub-ETs in which the different sub-ETs are responsible for a particular facet of the problem at hand. In this section we will discuss different aspects of the expression of the genetic information in gene expression programming, drawing attention to the different levels of phenotypic complexity.

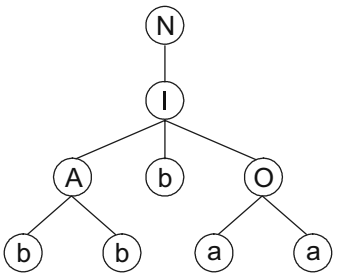
2.2.1 Information Decoding: Translation

From the simplest individual to the most complex, the expression of the genetic information in gene expression programming starts with translation or, in other words, with the construction of all the sub-ETs.

Consider the following chromosome composed of just one gene (the tail is shown in bold):

```
0123456789012345
NIAbObbaaaabaabb                    (2.10)
```

The symbols {A, O, N, I} represent, respectively, the Boolean functions AND, OR, NOT, and IF (if $a = 1$, then b; else c), where the first two functions take two arguments, NOT takes one argument, and the IF function takes three arguments. In this case, the product of translation is the following expression tree with nine elements:



which, as you can easily check, is a perfect, albeit rather verbose, solution to the NOR function.

Thus, for this simple individual, a single unit ET is the "organism" and, in this case, the expression of the genetic information ends with translation.

When the genome codes for more than one gene, each gene is independently translated as a sub-ET, and two different kinds of "organism" might be formed: in the first kind, the sub-ETs are physically connected to one another by a particular linking function; in the second kind, the sub-ETs work together to solve the problem at hand but there are no direct connections between them. Let's now make this clearer with two examples.

Consider, for instance, the following chromosome composed of three different genes (the tails are shown in bold):

```
012345678901234567890123456789
AOa abaaaab Nab aaaaaab INN bababaa          (2.11)
```

It codes for three different sub-ETs (Figure 2.3), each one representing a particular Boolean expression. Usually, these sub-ETs or sub-programs are part of a bigger program in which the sub-ETs are linked by a particular linking function. For instance, if we linked the sub-ETs one after the other by the Boolean function OR or AND, two different programs would be represented by chromosome (2.11) above. Thus, for this individual, the expression of the genetic information starts with the translation of the sub-ETs, but

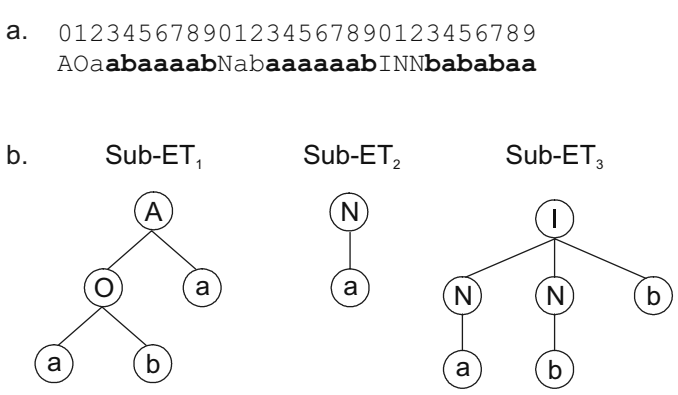


Figure 2.3. Translation of GEP genes as Boolean sub-ETs. **a)** A three-genic chromosome with the gene tails shown in bold. **b)** The sub-ETs codified by each gene. Note that the full expression of the chromosome will require some kind of interaction between the sub-ETs. Indeed, the program encoded in the chromosome only makes sense if the interactions between sub-programs are specified. For instance, three different programs would be obtained if the linking were done by OR, AND, or IF.

ends only after the linking of the sub-ETs by a particular linking function. And therefore the "organism" in this case will consist of a multi-subunit expression tree composed of three smaller subunits.

For problems with multiple outputs, however, the different sub-ETs encoded in the genome are engaged in the identification of just one kind of output and, therefore, they are not physically connected to one another: they remain more or less autonomous agents working together to solve the problem at hand. For instance, in classification problems a particular sub-ET is responsible for the identification of a particular class.

Consider, for instance, the chromosome below composed of three different genes created to solve a classification task with three distinct classes:

```
012345678901201234567890120123456789012
-/dac/dacaccd//-aacbbbabcd-d/+c*dbdbacd          (2.12)
```

It codes for three different sub-ETs, each one representing a rather complex algebraic expression (Figure 2.4).

a.
```
012345678901201234567890120123456789012
-/dac/dacaccd//-aacbbbabcd-d/+c*dbdbacd
```

b.

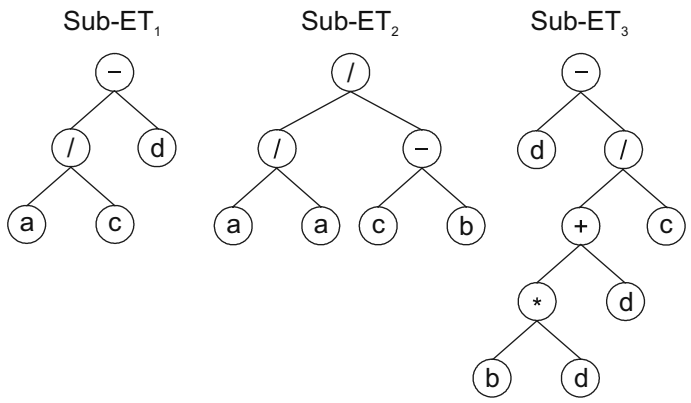


Figure 2.4. Translation of GEP genes as algebraic sub-ETs. **a)** A three-genic chromosome with the tails shown in bold. **b)** The sub-ETs codified by each gene. Note that, after translation, the sub-ETs might either form multi-subunit expression trees composed of smaller sub-ETs or remain isolated as a single-unit expression tree. For instance, in problems with just one output, the sub-ETs might be linked by a particular linking function or, in problems with multiple outputs, each sub-ET is responsible for the identification of a particular output.

In this classification task, for gene expression to be complete, the output of each sub-ET must first be converted into 0 or 1 using a rounding threshold, below which the output is converted into 0, 1 otherwise. Then the sub-ETs must be subjected to the following rules in order to determine the class:

IF (Sub-ET_1 = 1 AND Sub-ET_2 = 0 AND Sub-ET_3 = 0), THEN Class 1;
IF (Sub-ET_1 = 0 AND Sub-ET_2 = 1 AND Sub-ET_3 = 0), THEN Class 2;
IF (Sub-ET_1 = 0 AND Sub-ET_2 = 0 AND Sub-ET_3 = 1), THEN Class 3.

Let's make this more concrete with a simple example, a small sub-set of the iris dataset (Fisher 1936) where the first five samples of each type of iris in the original dataset are used (Table 2.1). Indeed, the program (2.12) above was created using the training samples shown in Table 2.1. And, as you can easily confirm with the help of Figure 2.4, this model classifies all the samples correctly using a rounding threshold of 0.5.

So, for problems with multiple outputs, multiple sub-programs are encoded in the chromosome and the "organism" is the result of an intricate collaboration between all sub-programs, in which each sub-program is engaged in the discovery of a particular facet of the global problem.

Table 2.1. The iris sub-set.

Sepal length (a)	Sepal width (b)	Petal length (c)	Petal width (d)	Type
5.1	3.5	1.4	0.2	class 1 (seto.)
4.9	3.0	1.4	0.2	class 1 (seto.)
4.7	3.2	1.3	0.2	class 1 (seto.)
4.6	3.1	1.5	0.2	class 1 (seto.)
5.0	3.6	1.4	0.2	class 1 (seto.)
7.0	3.2	4.7	1.4	class 2 (vers.)
6.4	3.2	4.5	1.5	class 2 (vers.)
6.9	3.1	4.9	1.5	class 2 (vers.)
5.5	2.3	4.0	1.3	class 2 (vers.)
6.5	2.8	4.6	1.5	class 2 (vers.)
6.3	3.3	6.0	2.5	class 3 (virg.)
5.8	2.7	5.1	1.9	class 3 (virg.)
7.1	3.0	5.9	2.1	class 3 (virg.)
6.3	2.9	5.6	1.8	class 3 (virg.)
6.5	3.0	5.8	2.2	class 3 (virg.)

2.2.2 Posttranslational Interactions and Linking Functions

We have already seen that translation results in the formation of sub-ETs with different sizes and shapes, and that the complete expression of the genetic information requires the interaction of these sub-ETs with one another. Only then will the individual be fully expressed. A very common and useful strategy consists in the linking of sub-ETs by a particular linking function. Indeed, most mathematical and Boolean applications are problems of just one output and, therefore, can benefit from this strategy, in which more complex programs are designed by linking together smaller sub-programs.

When the sub-ETs are algebraic expressions or Boolean expressions, any mathematical or Boolean function with more than one argument can be used to link the sub-ETs in a final, multi-subunit ET. For algebraic expressions, the most frequently chosen functions to link the sub-ETs are addition, subtraction, multiplication, or division. For Boolean expressions, the most frequently chosen linking functions are all the interesting functions of two arguments (functions 1, 2, 4, 6, 7, 8, 9, 11, 13, and 14, or, more intelligibly, NOR, LT, GT, XOR, NAND, AND, NXOR, LOE, GOE, and OR, respectively), or functions of three arguments such as the already familiar IF function (if $a = 1$, then b; else c) or the 3-multiplexer (also easily described as an IF THEN ELSE statement: if $a = 0$, then b; else c, which, as you can see, is very similar to the IF function).

However, the linking of sub-ETs with functions of two arguments is much simpler, as any number of sub-ETs can be linked together one after the other (see Figure 2.5). On the other hand, the linking of sub-ETs with linking functions of more than two arguments, say n arguments, is more problematic as it requires n^n sub-ETs for a correct linking (see Figure 2.6). Let's now make this clearer with two examples.

For instance, consider the following chromosome, encoding three algebraic sub-ETs linked by addition (the tails are shown in bold):

```
012345678901201234567890120123456789012
QaQ+-Q bbaaaba +Q+ab+ abababa *-**b+ aabbaba          (2.13)
```

As you can see in Figure 2.5, the sub-ETs are linked together one after the other in an orderly fashion. Note that the multi-subunit ET encoded in chromosome (2.13) could be linearly encoded as the following K-expression:

```
01234567890123456789012
++*Q+-*aQ+*b+aab+abbaab                              (2.14)
```

a.
```
012345678901201234567890120123456789012
QaQ+-Qbbaaaba+Q+ab+abababa*-**b+aabbaba
```

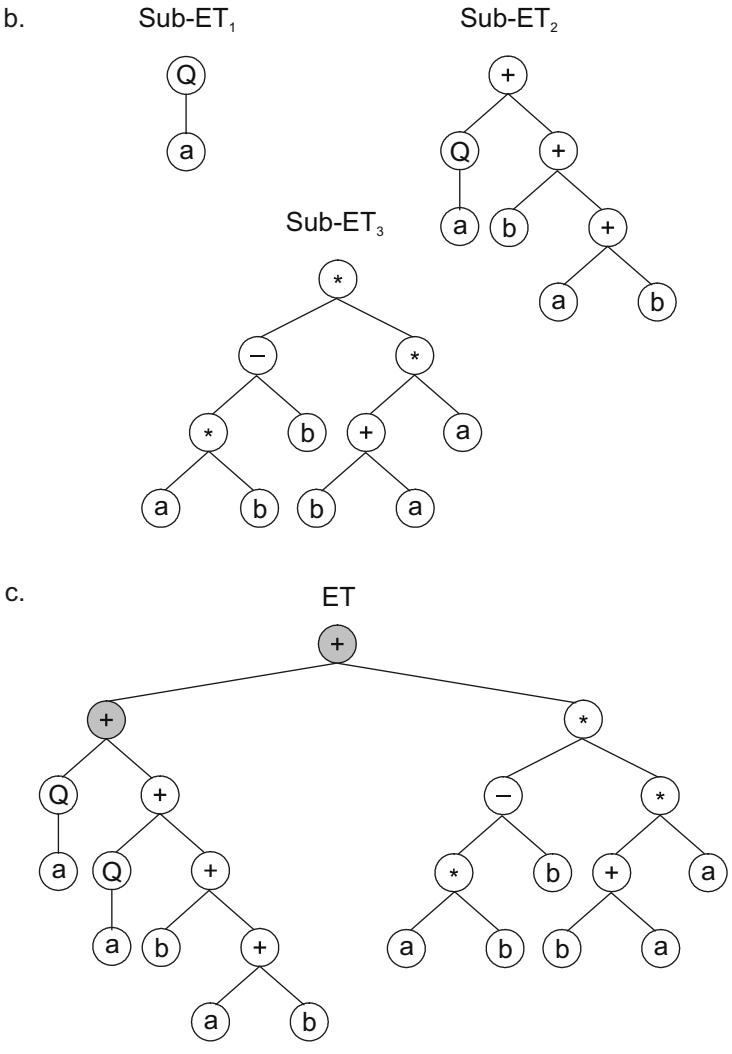


Figure 2.5. Expression of multigenic chromosomes encoding sub-ETs linked by a two-argument function. **a)** A three-genic chromosome with the tails shown in bold. **b)** The sub-ETs codified by each gene. **c)** The result of posttranslational linking with addition. The linking functions are shown in gray.

However, the use of multigenic chromosomes is more appropriate to evolve solutions to complex problems, for they permit the modular construction of more complex, hierarchical structures, where each gene codes for a smaller building block (see the section Testing the Building Blocks Hypothesis in chapter 12). These small building blocks are separated from each other and, therefore, can evolve with a certain degree of independence.

And now consider another chromosome, this time encoding three Boolean sub-ETs, linked by a three-argument function (the tails are shown in bold):

```
012345678901234501234567890123450123456789012345
IOaIA<b>cbaaacaacac</b>AOcaI<b>ccabcbccbac</b>IONAA<b>bbbbacbcbbc</b>          (2.15)
```

As you can see in Figure 2.6, at least three genes are required to link the sub-ETs with the IF(*a*,*b*,*c*) function. Note also that if more sub-ETs were needed, the simplest organization would require at least nine sub-ETs so that they could be linked properly by the three-argument function. Again, the multi-subunit ET encoded in chromosome (2.15) could be linearized, forming the following K-expression:

```
012345678901234567890123456789O
IIAIOaIOcONAAcbaaaIAbbbbacccaac                    (2.16)
```

In summary, to express fully a chromosome, the information concerning the kind of interaction between the sub-ETs must also be provided. Therefore, for each problem, the type of linking function or type of interaction between sub-ETs is chosen a priori. We can start with addition for algebraic expressions or OR for Boolean rules but, in some cases, another linking function might be more appropriate (like multiplication or AND, for instance). The idea, of course, is to find a good solution, and different linking functions can be used to explore different recesses of the fitness landscape, increasing the odds of finding Mount Everest. Notwithstanding, the basic gene expression algorithm can be easily modified to enable the evolution of linking functions. And an elegant and interesting way of solving this problem consists in the creation of homeotic genes that encode a developmental program or cell in which different combinations of sub-ETs are brought together by following the linking interactions operating in that particular cell (see how this is achieved in the next section).

a. 012345678901234501234567890123450123456789012345
IOaIA**cbaaacaacac**AOcaI**ccabcbccbac**IONAA**bbbbacbcbbc**

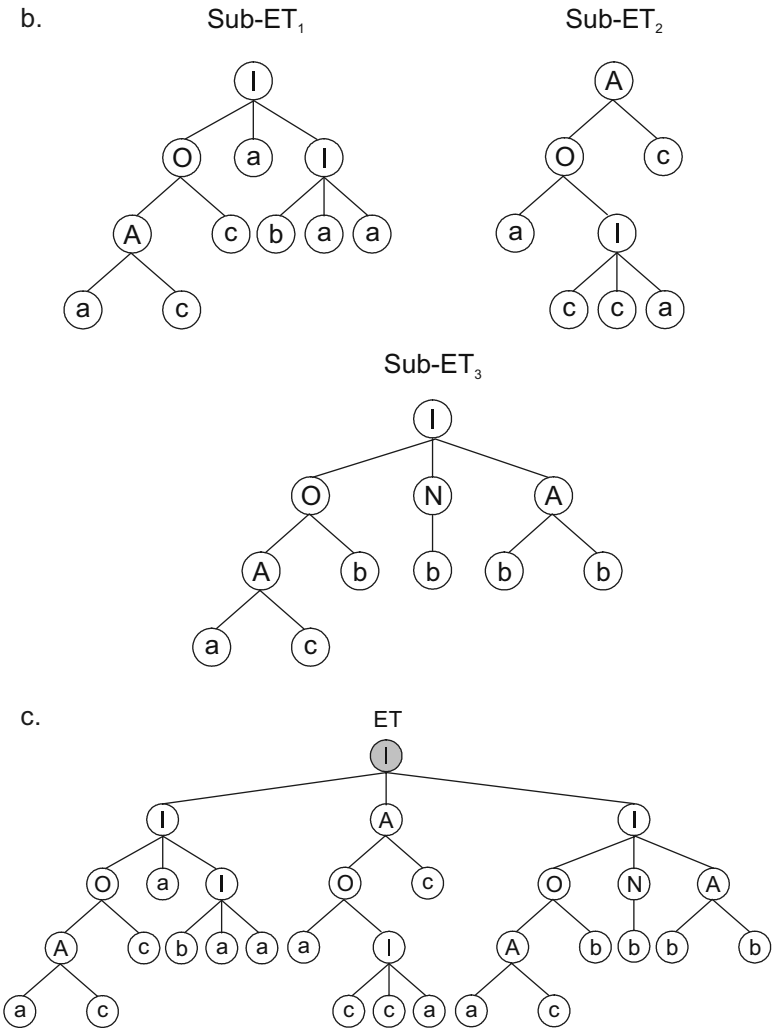


Figure 2.6. Expression of multigenic chromosomes encoding sub-ETs linked by a three-argument function. **a)** A three-genic chromosome with the gene tails shown in bold. **b)** The sub-ETs codified by each gene. **c)** The result of posttranslational linking with IF. The linking function is shown in gray.

2.3 Cells and the Evolution of Linking Functions

The linking of sub-ETs by a particular linking function is simple and highly efficient in evolutionary terms. Indeed, from unigenic to multigenic systems, efficiency increases considerably (see, for instance, section 5 of chapter 12 for a discussion of The Higher Organization of Multigenic Systems). Despite this artificial increase in complexity ("artificial" in the sense that it was not evolved by the system itself), evolution in multigenic systems still occurs efficiently and therefore they can be efficiently used to evolve good solutions to virtually all kinds of problems.

In principle, it is possible to impose from outside higher levels of complexity, but this is no guarantee that an increase in performance will be achieved. Natural evolutionary systems are not created this way: higher levels of complexity are created above lower levels and the evolution of complexity occurs more or less continuously.

Notwithstanding, the linking of sub-ETs in gene expression programming can be implemented by using a higher level of complexity. For that purpose a special class of genes was created – homeotic genes – that control the development of the individual. The expression of such genes results in different main programs or cells, that is, they determine which genes are expressed in each cell and how the sub-ETs of each cell interact with one another. Or stated differently, homeotic genes determine which sub-ETs are called upon (and how often) in which main program or cell and what kind of connections they establish with one another. How this is done is explained in the next section.

2.3.1 Homeotic Genes and the Cellular System

Homeotic genes have exactly the same kind of structural organization as conventional genes and they are built using an identical process. This means that they also contain a head and tail domain, with the heads containing, in this case, linking functions (so called because they are in fact used to link the different sub-ETs encoded in the conventional genes) and a special class of terminals – genic terminals – representing conventional genes, which, in the cellular system, encode different sub-ETs or sub-programs; the tails contain obviously only genic terminals.

Consider, for instance, the following chromosome:

```
01234560101234560101234560101234560123
/+a/abbba+*-abbabb/b*+abaab+Q/0*210212          (2.17)
```

It codes for three conventional genes and one homeotic gene (shown in bold). The three conventional genes code, as usual, for three different sub-ETs, with the difference that now these sub-ETs may be invoked multiple times from different places, or, stated differently, they will act as automatically defined functions (ADFs). And the homeotic gene controls the interactions between the different sub-ETs or, in other words, determines which functions are used to link the sub-ETs or ADFs and how the linking is established (Figure 2.7).

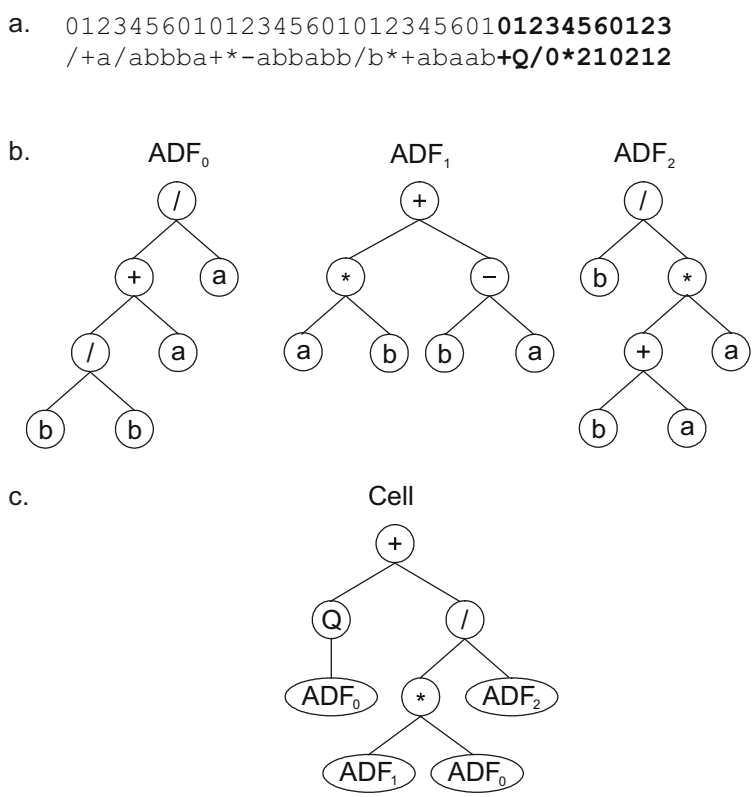


Figure 2.7. Expression of chromosomes with a single homeotic gene encoding a main program or cell. **a)** The chromosome composed of three conventional genes and one homeotic gene (shown in bold). **b)** The sub-ETs or ADFs codified by each conventional gene. **c)** The final main program or cell. Note that the cellular system allows code reuse as each ADF might be called several times by the main program.

It is worth emphasizing how flexible and dynamic the linking of sub-ETs became with this new method, allowing not only the use of any kind of linking function (even functions of just one argument can now be used as linkers) but also the use of different linking functions at a time as opposed to just one static linker. And to top it off, this new system is totally unsupervised, being the linking functions totally sorted out by the evolutionary process itself.

As Figure 2.7 clearly shows, this kind of representation not only allows the evolution of linking functions but also allows code reuse. Thus, this is also an extremely elegant form of implementing ADFs in gene expression programming. Indeed, any ADF in this cellular representation can not only be used as many times as necessary but also establish different interactions with the other ADFs in the main program or cell. For instance, in the particular case of Figure 2.7, ADF_0 is used twice in the main program, whereas ADF_1 and ADF_2 are both used just once.

It is worth pointing out that homeotic genes have their specific length and their specific set of functions. And these functions can take any number of arguments (functions with one, two, three, ..., *n* arguments). For instance, in the particular case of chromosome (2.17), the head length of the homeotic gene h_H is equal to five, whereas for the conventional genes $h = 4$; and the function set of the homeotic gene consists of $F_H = \{+, *, /, Q\}$, whereas for the conventional genes the function set consists of $F = \{+, -, *, /\}$.

In summary, as Figure 2.7 emphasizes, the cellular system of gene expression programming is not only a form of elegantly allowing the totally unconstrained evolution of linking functions in multigenic systems, but also an extremely elegant and flexible way of encoding automatically defined functions that can be called an arbitrary number of times from an arbitrary number of different places.

2.3.2 Multicellular Systems with Multiple Main Programs

The use of more than one homeotic gene results obviously in a multicellular system where each homeotic gene puts together a different combination of sub-expression trees.

Consider, for instance, the following chromosome:

```
01234560101234560101234560101234560123456
/Q+*babab/+a/abbab*Q-bbaaab*1+1020*Q*1202          (2.18)
```

It codes for three conventional genes and two homeotic genes (shown in

bold). And its expression results in two different cells or main programs, each expressing different genes in different ways (Figure 2.8). And as you can see in Figure 2.8, in this particular case, ADF_0 is called both from $Cell_0$ and $Cell_1$; ADF_1 is called twice from $Cell_0$ and just once from $Cell_1$; and ADF_2 is only called from $Cell_1$.

The applications of these multicellular systems are multiple and varied and, like the multigenic systems, they can be used both in problems with just one output and in problems with multiple outputs. In the former case, the best program or cell accounts for the fitness of the individual; in the latter, each cell is responsible for a particular facet of a multiple output task such as a classification task with multiple classes. We will pursue these questions further in chapter 6, where automatically defined functions are discussed in more detail.

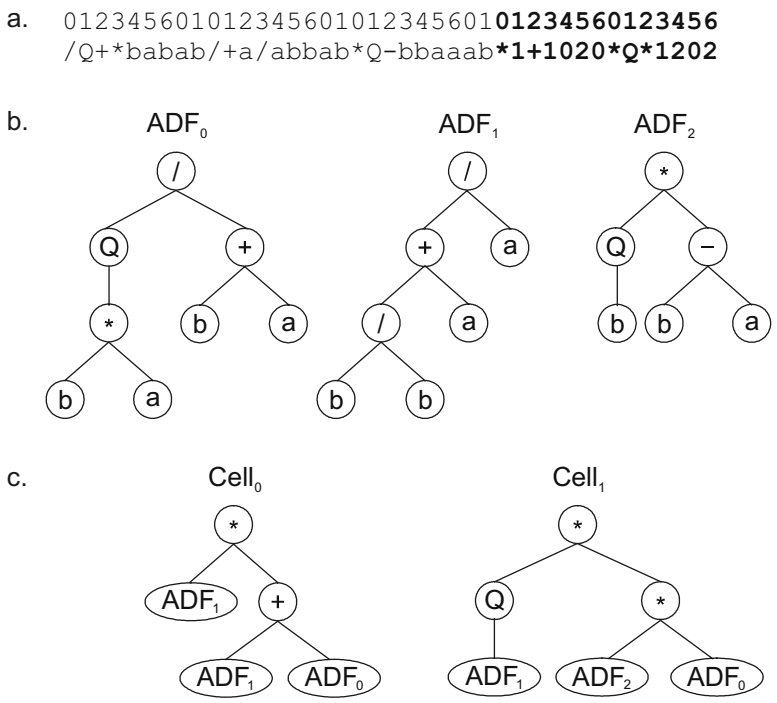


Figure 2.8. Expression of chromosomes with two homeotic genes encoding two different main programs or cells. **a)** The chromosome composed of three conventional genes and two homeotic genes (shown in bold). **b)** The sub-ETs or ADFs codified by each conventional gene. **c)** Two different main programs expressed in two different cells. Note how different cells put together different consortiums of genes.

2.4 Other Levels of Complexity

As we have seen in the examples above, although a very simple system, gene expression programming exhibits already a complex development and also uses different chromosomal organizations. So far, we have been dealing with genes (both conventional and homeotic) containing only head/tail domains. But gene expression programming regularly uses other chromosomal organizations that are more complex than the basic head/tail domain. These complex chromosomal structures consist of clusters of functional units composed of conventional head/tail domains plus one or more extra domains. The extra domains usually code for several one-element sub-ETs. And all the sub-ETs encoded in the different domains interact with one another, forming a more complex entity with a complex network of interactions.

One such architecture was developed to manipulate random numerical constants for function finding problems (Ferreira 2001, 2003). For instance, the following chromosome contains an extra domain Dc (shown in bold) encoding random numerical constants:

```
01234567890123456789012
+*?+?*+a??aaa??09081345                    (2.19)
```

As you can see in Figure 2.9, the translation of the head/tail domain is done in the usual fashion, but, after translation, additional processing is needed in order to replace the question marks in the tree by the numerical constants they represent. In chapter 5, Numerical Constants and the GEP-RNC Algorithm, we will learn how these sub-ETs interact with one another so that the individual is fully expressed.

Multiple domains are also used to design neural networks totally encoded in a linear genome. These neural networks are one of the most complex individuals evolved by GEP. In this case, the neural network architecture is encoded in a conventional head/tail domain, whereas the weights and thresholds are encoded in two extra domains, Dw and Dt, each encoding several one-element sub-ETs. For instance, the chromosome below contains two extra domains encoding the weights and the thresholds of the neural network encoded in the head/tail domain (the domains are shown in different shades):

```
0123456789012345678901234567890123456789010
DUDTUDcdabdcabacbad429984097914824092675841      (2.20)
```

As you can see in Figure 2.10, the translation of the head/tail domain encoding the neural network architecture is also done in the usual fashion, but the

a.
```
0123456789012345678901 2
+*?+?*+a??aaa??09081345
```

b.

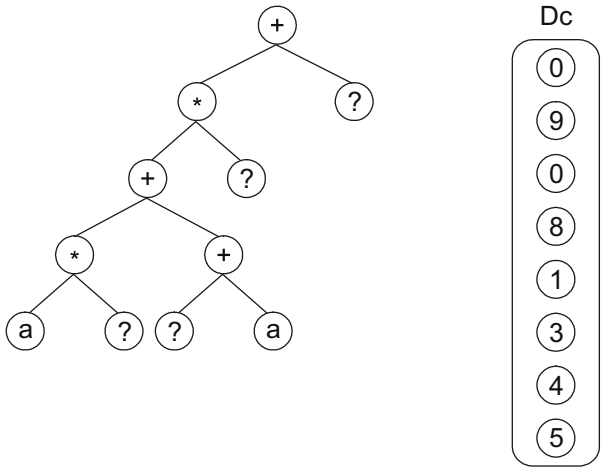


Figure 2.9. Translation of chromosomes with an additional domain for handling random numerical constants. **a)** The chromosome composed of a conventional head/tail domain and an extra domain (Dc) encoding random numerical constants represented by the numerals 0-9 (shown in bold). **b)** The sub-ETs codified by each domain. The one-element sub-ETs encoded in Dc are placed apart together. "?" represents the random numerical constants encoded in the numerals of Dc. How all these sub-ETs interact will be explained in chapter 5.

weights of the connections and the thresholds of the neurons must be assigned posttranslationally. In chapter 10, Design of Neural Networks, we will learn the rules of their complete development and how populations of these complex individuals evolve, finding solutions to problems in the form of adaptive neural networks totally encoded in linear genomes.

2.5 Karva Language: The Language of GEP

We have already seen that each gene codes for a particular sub-ET, and that each sub-ET corresponds to a specific K-expression or open reading frame. Due to the simplicity and elegance of this correspondence, K-expressions are, per se, extremely compact, intelligible computer programs. We have already seen how multi-subunit expression trees can be easily converted into linear K-expressions, and this can be easily done for any algebraic or Boolean

a.
```
012345678901234567890123456789012345678901234567890123456789010
DUDTUDcdabdcabacbad42998409791482409267584 1
```

b.

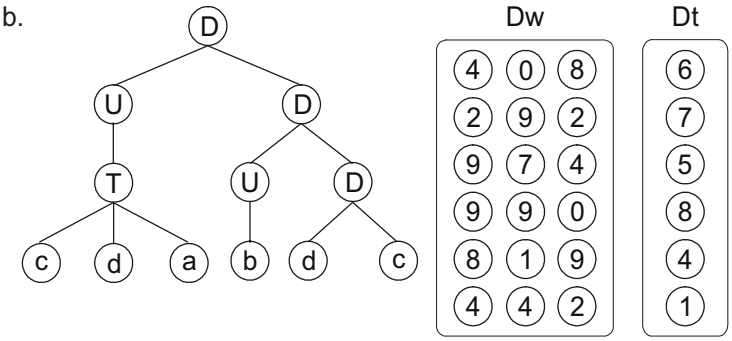


Figure 2.10. Translation of chromosomes with two extra domains for handling the weights and thresholds of neural networks. **a)** A multi-domain chromosome composed of a conventional head/tail domain encoding the neural network architecture, and two extra domains – one encoding the weights (Dw) of the neural network encoded in the head/tail domain and another the thresholds (Dt). Dw and Dt are shown in different shades. **b)** The sub-ETs codified by each domain. The one-element sub-ETs encoded in Dw and Dt are placed apart together. "U", "D", and "T" represent, respectively, neurons or functions with connectivity one, two, and three. How all these sub-ETs interact will be shown in chapter 10.

expression. Indeed, the language of gene expression programming – Karva language – is a versatile representation that can be used to evolve relatively complex programs as simple, extremely compact, symbolic strings. In fact, there is already commercially available software such as Automatic Problem Solver by Gepsoft that automatically converts K-expressions and GEP chromosomes into several programming languages, such as C, C++, C#, Visual Basic, VB.NET, Java, Fortran, VHDL, Verilog, and others.

Another advantage of Karva notation is that it can be used to evolve highly sophisticated programs using any programming language. Indeed, the original GEP implementation was written in C++, but it can be done in virtually any programming language, as it does not rely on any quirks of a particular programming language. As a comparison, it is worth pointing out that early GP implementations relied greatly on LISP because the sub-tree swapping that occurs during reproduction in that system is very simple to implement in that programming language.

In the next chapter, the implementation details of the gene expression algorithm will be fully analyzed, starting with the creation of the initial population and finishing with selection and reproduction to generate the new individuals of the new generation.