A Graph Grammar Approach to Artificial Life

Ole Kniemeyer[1]
Gerhard H. Buck-Sorlin[1,2]
Winfried Kurth[1]

[1]Brandenburgische Technische Universität Cottbus
Department of Computer Science
Chair for Practical Computer Science/Graphics Systems
P.O. Box 101344
D-03013 Cottbus
Germany
okn@informatik.tu-cottbus.de
wk@informatik.tu-cottbus.de

[2]Institute of Plant Genetics and Crop Plant Research
Department of Cytogenetics
Corrensstrasse 3
D-06466 Gatersleben
Germany
buck@ipk-gatersleben.de

Abstract We present the high-level language of *relational growth grammars* (RGGs) as a formalism designed for the specification of ALife models. RGGs can be seen as an extension of the well-known parametric Lindenmayer systems and contain rule-based, procedural, and object-oriented features. They are defined as rewriting systems operating on graphs with the edges coming from a set of user-defined relations, whereas the nodes can be associated with objects. We demonstrate their ability to represent genes, regulatory networks of metabolites, and morphologically structured organisms, as well as developmental aspects of these entities, in a common formal framework. Mutation, crossing over, selection, and the dynamics of a network of gene regulation can all be represented with simple graph rewriting rules. This is demonstrated in some detail on the classical example of Dawkins' biomorphs and the ABC model of flower morphogenesis: other applications are briefly sketched. An interactive program was implemented, enabling the execution of the formalism and the visualization of the results.

Keywords
Graph grammars, L systems, genotype-phenotype mapping, morphogenesis, models of evolution

1 Introduction

High-level, rule-based languages exhibit several virtues that make them suitable for biological modeling: Being both transparent in use and not requiring constant recompilation of an often rather lengthy and obscure code after modification (thanks to powerful string parsers), they are particularly apt to become the foundation for a programming language that is well suited for biology and artificial life—enabling easy specifications of models from the level of biochemistry and genetics up to the level of ecological interactions. We undertake one further step towards such a specialized formal language.

Since the advent of L systems (Lindenmayer systems) in 1968 [23], these string-rewriting grammars have been primarily used to model the growth and architecture of individual plants [26]. By using fixed rule tables and lineage-controlled replacement, most classical models of this sort emphasize the genetically based aspects of morphogenesis (it was not until the nineteen-nineties that *globally sensitive* and *open* L systems took the environment into account [20, 24]). Nevertheless, neither the genome itself nor lists of specific genes were usually represented in such L systems, despite the double-string structure of DNA, which seems to predestine it to be described in a string-rewriting formalism. The *interactive barley* model by Buck-Sorlin and Bachmann [1] is one exception, but even this model does not properly exploit the string nature of the genome. Dassow and Mitrana [7] have devised a formal string-rewriting language called *evolutionary grammars* to represent genetic operators like mutation, recombination, and

transposition. However, their formalism is restricted to the genetic level and lacks the capability to model architectural or behavioral aspects of the phenotype. Kim [16] has developed transsys, a software tool enabling the specification and dynamic simulation of gene expression, transcription factors, and L-system-based morphogenetic processes. But despite representing an important step towards an integrated tool, transsys treats genetic activity and macroscopic morphogenesis with different formalisms.

A valuable hint as to how to integrate several levels of scaling in one coherent framework came from Godin and Caraglio [15]. They defined *multiple-scaled tree graphs* (MTGs) to describe branching structures simultaneously at several degrees of spatial resolution. Each scale of resolution corresponds to a modular structure which can formally be represented by a (graph-theoretical) tree. Entities at one scale are decomposed into entities at finer scales. Three basic relations were defined which are used to build an MTG: The relation < is valid between two subsequent morphological entities that are defined at the same scale of resolution (e.g., between two annual shoots following each other and being part of an axis of a botanical tree). The relation + holds between an entity and a branching daughter entity at the same scale. Finally, the *decomposition,* or *contains*, relation, denoted / in [3], holds between an entity at a given scale and its components defined at the next finer scale (e.g., between a shoot of a herbaceous plant and an internode that is part of that shoot). Although this approach is basically static and does not encompass the genetic level, it served as a model for our *relational growth grammar* (RGG) data structures, in which binary relations play a key role.

Alternative powerful formalisms for representing ALife models in a grammar framework are cooperating distributed grammar systems [5] and eco-grammar systems [4, 12, 6]. They are specifically tailored to represent multi-agent models and can also be seen as extensions of the L-system approach. However, their suitability for describing genetic or evolutionary processes has not yet been demonstrated.

In the following, we will first demonstrate how crossing over can already be represented in a classical L-system setting. This will show the power of the L-system formalism, but also some of its current limitations. We will then sketch our improved formal language and demonstrate its applicability to ALife model specification by "reincarnating" the well-known biomorphs that were popularized by Dawkins in his work *The Blind Watchmaker* [8]. Our final example will be a reimplementation of the ABC model of flower morphogenesis previously formalized in transsys by Kim [16]. In the discussion and conclusions, we will mention how other well-known ALife formalisms can be represented in our framework with ease, and what further perspectives will be opened by our approach [17].

2 Crossing Over with Classical L Systems

Let us consider a theoretical genome with just a small number of genes, the latter having again few alleles: In such a situation, crossing over of loci can be simulated using a combinatorics approach and parametric L systems, by simply stating all possible exchange events (single, double, and triple crossing over) as rules. For four genes the number of possible crossing-over events is seven, including three simple, three double, and one triple crossing-over events. The probability of a crossing-over event between two loci is a function of the distance between the two genes, a notion that permits the creation of marker maps in the first place: The closer the alleles of certain genes are located to each other, the higher are their chances of being transmitted together to the same organism. The distance between any two genes on a marker map is not to be understood as physical distance, but as a measure of allele recombination

frequency, usually stated in centimorgans (cM)[1]: the latter can thus be used to calculate the probability of a single crossing-over event, using a version of the Kosambi function:

$$p_r = \frac{1 - e^{-0.02d}}{2} \tag{1}$$

where p_r is the recombination frequency and d the map distance. Double and triple crossing-over events are usually much rarer, their probability being the product of the probabilities of the single events constituting them.

The L system in Table 1, written in a syntax similar to that used by the GROGRA software [20], visualizes crossing over in an F2 population of a theoretical diploid organism with four genes on a single chromosome. The genes are declared as constants in the header, with one constant per locus for each parent of the crossing population. The constant in this case adopts values of 1 or 0, which are set manually. In the first few steps (axiom and P_1), the two parents plus the F1 (which represents a heterozygous combination of the parental genotypes) are depicted. A set of eight stochastic rules that produce sixty F2 offspring $g(g_1, \ldots, g_4, h_1, \ldots, h_4, t)$, thus containing the parental genotype as parameters (Figure 1) is specified. Crossing over is provided for, too, in the rules of P_2: the probability of execution of each rule is equal to one of three global constants dis_1, dis_2, dis_3, or to some product of two or all three of them.

This example demonstrates the capability of classical L systems to represent genetic operations. However, in the case of more than four loci, the exhaustive listing of all possible multiple crossing-over events will become lengthy and the parameter lists will grow intolerably. In our new formal framework, we will represent the genomes by extra strings, which are associated with the organisms by edges of a graph. Crossing over will be modeled by one single graph-rewrite rule.

3 Relational Growth Grammars

3.1 The Data Structures

L systems operate on strings, that is, on finite sequences of symbols—the symbols either being atomic units from a finite alphabet or carrying a finite list of numerical parameters ("parametric L-systems" [26]). However, in biological models we quite often encounter other typical data structures, which can be roughly categorized together with the strings in the following scheme in order of increasing complexity (see [19] for a related discussion):

1. *Multiset*: In broad terms, a multiset is an unordered collection where the same element can appear several times—the usual mathematical notion of *set* is a special case thereof. Examples with biological significance are unstructured populations and molecules in a solution. L systems operating on multisets were introduced by Lane and Prusinkiewicz [22].

2. *String* (or one-dimensional *array*, or *list*, which we regard as synonyms here): In this case we have in addition to the multiset a strict linear order among the elements. In our graph-related approach we will use special directed edges of successor type to connect nodes that follow each other directly in this order relation. As classical L systems will be embedded as a special case in our formalism, we will assume that two subsequent symbols in a grammar-generated string will automatically be connected by a successor edge. (Exceptions are the symbols [and], which will no longer be treated as

1 100 cM corresponds to the entire length of the chromosome. In other words, if the number of (single-point) crossovers that separate two genes, divided by the number of all possible single-point crossovers, is 1%, then the distance between the genes is 1 cM.

Table 1. L-system listing representing crossing-over in a population of a theoretical diploid organism.

```
\const NOOFF 60, /* number of offspring in F2
                    (maximum = 60 at the moment) */
\const G1 1, /* Locus G1, 1st parent */
[ditto for G2 to G4]
\const H1 0, /* Locus G1, 2nd parent */
[ditto for H2 to H4]
\const dis1 0.2, /* distance between G1 and G2 */
[ditto for dis2 and dis3, distances between G2 and G3, and G3 and G4]
axiom:[ g(G1,G2,G3,G4) ] [ g(G1,G2,G3,G4) ] [ g(H1,H2,H3,H4) ]
      [ g(H1,H2,H3,H4) ] hyb,
P1: hyb -> [ g(G1,G2,G3,G4) ] [ g(H1,H2,H3,H4) ]
            gam(G1,G2,G3,G4,H1,H2,H3,H4,0),
/* crossing-over rules for the F2: */
/* no crossing-over: */
P2: (t < NOOFF) gam(g1,g2,g3,g4,h1,h2,h3,h4,t) ->
  g(g1,g2,g3,g4,h1,h2,h3,h4,t) gam(g1,g2,g3,g4,h1,h2,h3,h4,t+1)
  ?(1-(dis1*(1+dis2+dis2*dis3+dis3)+dis2*(dis3+1)+dis3)),
/* single crossing-over at G1: */
(t < NOOFF) gam(g1,g2,g3,g4,h1,h2,h3,h4,t) -> g(h1,g2,g3,g4,g1,h2,h3,h4,t)
  gam(g1,g2,g3,g4,h1,h2,h3,h4,t+1) ?(dis1),
/* double crossing-over at G2: */
(t < NOOFF) gam(g1,g2,g3,g4,h1,h2,h3,h4,t) -> g(g1,h2,g3,g4,h1,g2,h3,h4,t)
  gam(g1,g2,g3,g4,h1,h2,h3,h4,t+1) ?(dis1*dis2),
/* double crossing-over at G3: */
(t < NOOFF) gam(g1,g2,g3,g4,h1,h2,h3,h4,t) -> g(g1,g2,h3,g4,h1,h2,g3,h4,t)
  gam(g1,g2,g3,g4,h1,h2,h3,h4,t+1) ?(dis2*dis3),
/* single crossing-over at G4: */
(t < NOOFF) gam(g1,g2,g3,g4,h1,h2,h3,h4,t) -> g(g1,g2,g3,h4,h1,h2,h3,g4,t)
  gam(g1,g2,g3,g4,h1,h2,h3,h4,t+1) ?(dis3),
/* single crossing-over at G2: */
(t < NOOFF) gam(g1,g2,g3,g4,h1,h2,h3,h4,t) -> g(h1,h2,g3,g4,g1,g2,h3,h4,t)
  gam(g1,g2,g3,g4,h1,h2,h3,h4,t+1) ?(dis2),
/* triple crossing-over at G1 and G3: */
(t < NOOFF) gam(g1,g2,g3,g4,h1,h2,h3,h4,t) -> g(h1,g2,h3,g4,g1,h2,g3,h4,t)
  gam(g1,g2,g3,g4,h1,h2,h3,h4,t+1) ?(dis1*dis2*dis3),
/* double crossing-over at G1 and G4: */
(t < NOOFF) gam(g1,g2,g3,g4,h1,h2,h3,h4,t) -> g(h1,g2,g3,h4,g1,h2,h3,g4,t)
  gam(g1,g2,g3,g4,h1,h2,h3,h4,t+1) ?(dis1*dis3),
```

parts of a string but encode branches in a more direct way than in classical L systems; see below.) The successor relation corresponds to Godin's < relation [15]. By the term "successor" in the context of graphs, we always mean this topological, directed neighbor relation between linearly ordered nodes, not a numerical relation between possible attributes of the nodes. Biological examples of string structures are genomes and linearly ordered architectural substructures such as the vegetative axes of plants.

3. *Axial tree*: Here we permit the attachment of lateral branches to the elements of a string. Lateral branches are indicated by the bracket notation known from graphically interpreted L systems [26]: There, the brackets are used to encode branching in a string and only become important in the (turtle) interpretation step, whereas in our model new branches are immediately created during rule application. We use two more standard

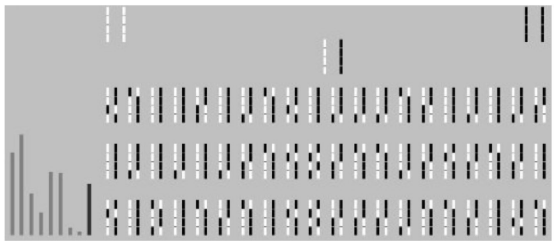

Figure 1. Graphical output of the L system of Table 1: The genotypes of a theoretical diploid organism are represented as double bars with four different genes. Two possible alleles per locus are displayed in black and white. The display includes two parental genotypes (above), the heterozygous F1, and a population of 60 segregating F2. To the left, a histogram is given that depicts the frequency of application of crossing over rules, from left to right: no crossing over (co); single co at G1 (G1-G2); double co at G2 (G1-G2, G2-G3); double co at G3 (G2-G3,G3-G4); single co at G4 (G3-G4); single co at G2 (G2-G3); triple co at all loci; double co at G1 and G4 (G1-G2, G3-G4); total number of co (not to scale!).

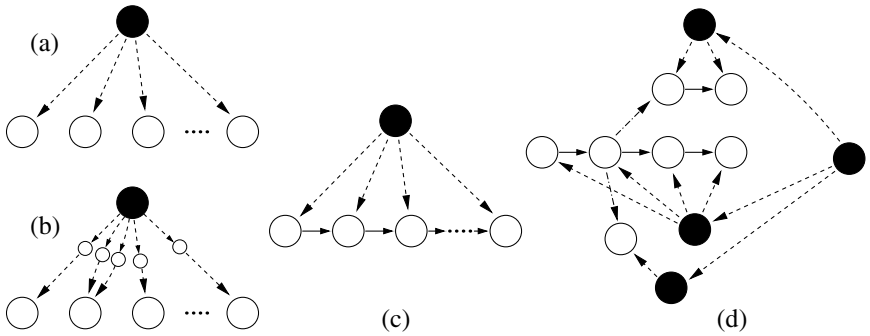

Figure 2. Set (a), multiset (b), string (c), and axial tree (d) as data structures represented by edge-labeled graphs. Continuous edges denote the successor relation, dashed edges the contains relation, dot-dash edges the branch relation. Filled nodes represent objects at a lower level of resolution (multiscaled approach; cf. [15]). The multiset requires auxiliary nodes—the middle layer of nodes in (b)—to represent multiple edges between the same pair of nodes.

relations to represent this data structure: the *branch* relation (corresponding to Godin's +), and the *contains* relation (/), which connects each axis with its elements and a universal node representing the whole structure with all existing axes. The biological counterparts of axial trees are real botanical trees (at the macroscopic level).

Graphical representations of these data structures are depicted in Figure 2.

For each basic data structure, a toolbox of standard functions will be available—for example, in the case of axial trees, functions like `insertChild`, `getFirstChild`, `getNextSibling`, `parent`, `root`, `higher`, and `index`. The first four of these functions refer to child nodes in the sense of the branch relation. `root` returns the root node of a tree composed of "successor" and "branch" edges. `higher` returns the node that is obtained by following one "contains" edge in the reverse direction (this node is assumed to be unique). This means that when we apply `higher` to an element of a string, the node representing the whole string (filled circle in Figure 2c) is the result. `index` returns the number of the position of a node inside a string. The basic data structures are special cases of our most general notion, the relational structure.

4. *Relational structure* (or edge-labeled directed graph). Here we permit arbitrary user-defined types of edges and relations, representing relationships that are not yet covered by the above basic structures—for example, the relation between genotype and phenotype, or the *alignment* relation between elements of two homologous genome

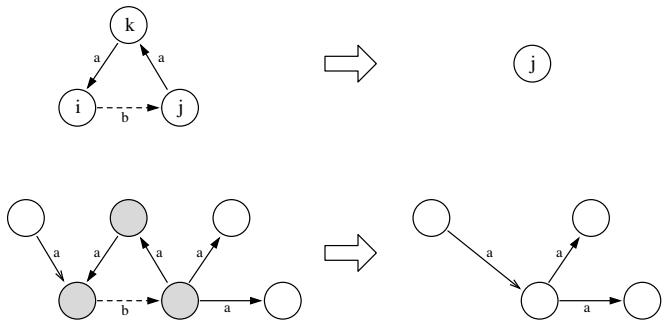

Figure 3. Example of a graph rewriting rule (upper part) and its application to a graph (lower part). The shaded nodes are matched by the left side of the rule. A possible text notation for this rule is "$i \xrightarrow{b} j, j \xrightarrow{a} k, k \xrightarrow{a} i \Longrightarrow j$".

strings. Between a given ordered pair of nodes, an edge with a given label is allowed to appear at most once—this ensures that the edge types do in fact mathematically correspond to relations. The nodes themselves are not merely symbols as in L systems; they are objects in the sense of object-oriented programming.

3.2 The Formalism

A thorough documentation with complete definitions of the concepts used in our new formalism will be published elsewhere. Here we list only the main items characterizing the RGG calculus:

- *Graph grammars* are used to rewrite the data structures described above. In contrast to classical L systems, several nodes, generalizing the symbols in L systems, can appear on the left side of a rule. (Here we do not mean a possible context in the specification of a context-sensitive rule, but the left side proper, i.e., the part that is replaced when the rule is applied.) In the most general case, entire subgraphs are rewritten (Figure 3). Our variant of graph grammars, using both node and edge labels, was partly inspired by the PROGRES system (see [30]).
- On the other hand, features from parametric L systems are exploited. Particularly, commands of *turtle geometry* (in the systematized variant used by the GROGRA software [20]) are allowed as nodes and serve to interpret the generated graphs geometrically if necessary. For example, the command **F** (either with no parameters or with a length or with displacement coordinates as parameters) causes the creation of a solid cylinder representing, for example, a plant cell or (botanical) internode.
- A pure rule-based approach is not feasible in all situations. Programmed graph replacement systems [29] solve this problem by supporting additional programming structures. Within RGGs, these consist in a conventional object-oriented programming language. This language serves as a framework and allows the user to define *variables* (already a convenient feature of GROGRA [20]), *classes*, and *methods*. Pieces of code can be inserted into a rule and are executed when the rule is applied (see [27] for fragments of C code in L systems). Methods may, for example, implement the control logic of table L systems ([28, 13]; cf. "sub-L-systems" [25]), namely, control the order in which certain groups of rules are to be applied. In some cases, this direct specification of application order is more transparent and convenient than an encoding in additional parameters and rule conditions.

- New *relations* between nodes can be defined by the user. These can be used in graph query expressions as a generalization of edges.
- Essential to all rule-based languages are the processes of *matching* and *replacement*, into which the application of a rule can be decomposed. Two modes of rule application can be distinguished, according to the way these processes are evoked within a single rewriting step:

  Parallel matching and parallel replacement is the default method in classical L systems and in other transformation systems such as in cellular automata. In the case of graph grammars, this concurrency may lead to conflicting modifications when there are overlapping matches. One has to be aware of this fact and make additional arrangements if necessary in order to resolve such conflicts. Future extensions of the RGG calculus will contain explicit support for the most common conflict resolutions.

  Sequential replacement—that is, in each rewriting step only one match is exploited. Chomsky grammars and "decomposition rules" (in cpfg [25, 27]) work this way. In our formalism it is possible to specify explicitly how many times a rule (or a block of rules) may be applied during a rewriting step.

  The rewriting mechanism of each individual match follows the single-pushout approach [10], also known as the algebraic or Berliner approach. In addition, rules can be equipped with an embedding mechanism that establishes connections between the old and the new part of the graph. Currently, this mechanism simply transfers incoming (outgoing) edges of the textually leftmost (rightmost) nodes of the left-hand side to the textually leftmost (rightmost) nodes of the right-hand side. For example, the rule in Figure 3 makes use of this embedding.
- *Conditions* can be specified that have to be checked as part of the matching process (cf. [26]).
- *Graph contexts* can be defined within these conditions and are not confined to left and right contexts in the sense of string matching.
- *Probabilities* can further modify the applicability of rules, leading to stochastic grammars [26].

3.3 Crossing Over in the New Model

If we assume that the genomes of our organisms can be represented by "strings" in the sense of the above standard data structure, we have to take care that after a reorganization of the string (by application of a rule) the relations of type *contained*, which connect the elements of the string with the "higher" node representing the entire string (cf. Figure 2), are updated properly. We consider it inadequate to leave this technical task to the designer of the grammar. Therefore, in our system a *refresh* method is implicitly associated with each string. It observes the elements of the string and is automatically activated if some changes in the successor relations (or other modifications, like insertions of elements) occur. By this mechanism, the correct configuration of "contained-type" relations is always ensured.

To encode crossing-over events like those discussed in Section 2, we need two additional relations which are both undirected: a *mating* edge between the two strings that are (potential) objects of crossing over, and an *alignment* between homologous gene loci. The latter can be defined in a *relation definition* statement using the built-in

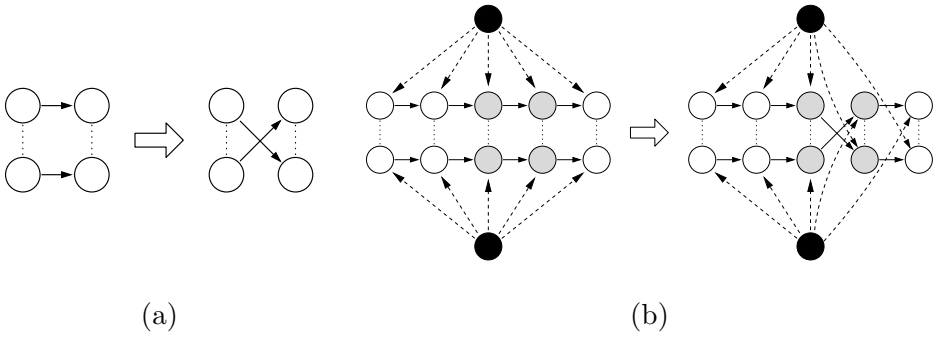

Figure 4. (a) A RGG rule representing crossing over, and (b) its application to a pair of strings. Continuous edges denote the successor relation, dashed edges containment, and dotted edges alignment. In (b), the containment relations have been updated.

function `index`, which returns the position of an item within a string:

$$j \overset{\text{align}}{\text{---}} k := \texttt{index}(j) == \texttt{index}(k)$$

Crossing over can then be described by the graph rewrite rule shown in Figure 4a, which can be written as follows:

$$j\ k,\ m\ n,\ j \overset{\text{align}}{\text{---}} m,\ \left(\texttt{match}(\texttt{higher}(j) \overset{\text{mate}}{\text{---}} \texttt{higher}(m))\right)$$
$$\Longrightarrow\ \texttt{if}\ \left(\texttt{prob}(p_r(dist(j,k)))\right)\ (j\ n,\ m\ k)\ \texttt{else break}$$

The left-hand side lists the successor relation between j and k and between m and n, which is expressed by blanks (as in GROGRA), and the alignment between j and m. The alignment between k and n, although indicated in Figure 4a, can be omitted in the condition, because it follows from $j \overset{\text{align}}{\text{---}} m$. If a matching subgraph is found and the mating condition is met (i.e., the basal nodes are connected by a mate edge), the rule applies: With the recombination probability p_r depending on the genetic distance *dist* between two loci, the successor relations are redirected as in Figure 4a; otherwise the rule is not applied. Matching of the rule is checked at all possible positions, thus making double or multiple crossovers in extra rules dispensable (as would have to be done with classical L-systems). In order to prevent the mirror match resulting from the (j, k)-(m, n) symmetry of the left-hand side, no two matches can consist of the same sets of matched nodes.

The embedding of this crossing-over rule in a complete grammar describing a simple artificial organism consisting of genotype and phenotype will be demonstrated in Section 4.

3.4 Run-Time Behavior

The usability of the RGG calculus strongly depends on its run-time behavior. In general, locating all possible matches of a given left hand side in a graph (the subgraph-isomorphism problem) is NP-complete, which renders a graph grammar approach computationally expensive. However, most of the usual rules consist of simple left-hand sides which can be located quickly. In the examples presented here the run-time behavior posed no problem. Further efforts at improving the matching algorithm could be made in order to lift computational limits; for example, matching of a graph pattern could begin at the least frequently matching node pattern, or frequently appearing

subpatterns could be located and cached before the actual matching algorithm starts. Such sophisticated internal enhancements would be of immediate use for every RGG, without the need of modifying it.

4 The Example of Biomorphs

In the following, we are going to apply the new growth grammar formalism, including the crossing-over rule from the last section, to the well-known Blind Watchmaker program, an algorithm written by the zoologist Richard Dawkins [8] and inspired by the seemingly blind and undirected yet enormously effective phenomenon of evolution by mutation and selection. The algorithm is based on a rather simple genotype-phenotype model and was specified by Dawkins in Pascal [9]. In this original version, it consists of two procedures: The first is responsible for the reproduction of the genotype, thereby producing a number of new individuals and introducing some random mutation. The second provides a developmental scheme in which the genotype is expressed following a simplified recursive tree-drawing routine with the genes' values used as parameters (depth of recursion, directions of branches). The genotype consists of nine genes, eight of which have 19 alleles ranging from -9 through 0 to $+9$, and the ninth a range from 0 to 9, in integer steps. The latter gene determines the depth of recursion. The other genes are translated into components of a matrix d, which represent the horizontal, and the vertical setoffs in a global coordinate system to be used in the development of the growing binary tree. The program is started off with an initial set of alleles and is then modified in the offspring by applying random mutations with given probability.

The parent and its offspring are displayed on the screen, and the user, by applying what Dawkins calls Darwinian (artificial) selection, chooses one of the offspring that on its turn is reproducing. By piling up mutations in a certain direction, which is completely at the discretion of the user, a shortcut is taken through the multidimensional genotypic and phenotypic parameter space with its billions of combinatorial possibilities, thereby arriving at astonishing *biomorphs*—structures that closely resemble organismal morphologies. Our RGG will furthermore provide a possibility to select *two* parent individuals from which offspring is generated using the crossing-over rule shown above.

The following table of RGG rules contains both alternatives: simple vegetative reproduction (using the rule block *SelectOne*), and reproduction from two parents involving crossing over (using *SelectTwo* and *Recombine* instead).

$$\begin{aligned}
&\textit{Initialize}: \ \alpha \\
&\quad \Longrightarrow \ \textit{Population} \xrightarrow{\text{contains}} \textit{Genome}\,[\xrightarrow{\text{first}} 1\ 1\ 1\ 1\ 1\ 0\ {-1}\ {-1}\ 5] \\
&\textit{Reproduce}: \ p : \textit{Population} \xrightarrow{\text{contains}} g : \textit{Genome} \\
&\quad \Longrightarrow \ p\ \texttt{for}(i = 0..13) \\
&\qquad ([\xrightarrow{\text{contains}} \textit{Germ}(200 * i) \xleftarrow{\text{encodes}} \texttt{cloneTree}(g)]) \\
&\textit{Mutate}: \ \texttt{int} \Longrightarrow \texttt{if}\big(\texttt{prob}(p_m)\big)\ \texttt{irandom}(-9, 9)\ \texttt{else break} \\
&\textit{Grow}: \ \textit{Germ}\ (x)\ (* \xleftarrow{\text{encodes}} g : \textit{Genome}\ *) \\
&\quad \Longrightarrow \ \textit{Biomorph}\ \ \mathbf{f}(x, 0, 0)\ \textit{Biom}(\texttt{abs}(g[8]) + 1, 2);
\end{aligned}$$

$$
\begin{aligned}
& b : \mathit{Biom}(\mathit{depth}, \mathit{dir}), \\
& \big(\texttt{match}(\texttt{root}(b) \xleftarrow{\text{encodes}} g : \mathit{Genome}) \;\&\&\; (\mathit{depth} > 0)\big) \\
\Longrightarrow\ & \big\{\texttt{int[][]}\ d = \big\{\{-g[1], g[5]\}, \{-g[0], g[4]\}, \{0, g[3]\}, \\
& \qquad \{g[0], g[4]\}, \{g[1], g[5]\}, \{g[2], g[6]\}, \\
& \qquad \{0, g[7]\}, \{-g[2], g[6]\}\big\};\ \big\} \\
& \mathbf{F}(\mathit{depth} * d[\mathit{dir} \bmod 8][0], 0, \mathit{depth} * d[\mathit{dir} \bmod 8][1]) \\
& [\mathit{Biom}(\mathit{depth} - 1, \mathit{dir} - 1)][\mathit{Biom}(\mathit{depth} - 1, \mathit{dir} + 1)]
\end{aligned}
$$

$$
\begin{aligned}
\mathit{SelectOne}: \ & \mathit{Population} \xrightarrow{\text{contains}} b : \mathit{Biomorph} \xleftarrow{\text{encodes}} g : \mathit{Genome}, \\
& \big(\texttt{isSelected}(b)\big) \\
\Longrightarrow\ & \hat{}\,\mathit{Population} \xrightarrow{\text{contains}} g
\end{aligned}
$$

$$
\begin{aligned}
\mathit{SelectTwo}: \ & \mathit{Population}\ [\xrightarrow{\text{contains}} b_1 : \mathit{Biomorph} \xleftarrow{\text{encodes}} g_1{:}\mathit{Genome}] \\
& \qquad\qquad [\xrightarrow{\text{contains}} b_2 : \mathit{Biomorph} \xleftarrow{\text{encodes}} g_2{:}\mathit{Genome}], \\
& \big(\texttt{isSelected}(b_1)\ \&\&\ \texttt{isSelected}(b_2)\big) \\
\Longrightarrow\ & \hat{}\,\mathit{Population}[\xrightarrow{\text{contains}} g_1][\xrightarrow{\text{contains}} g_2],\ g_1 \xrightarrow{\text{mate}} g_2
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Recombine}: \ & \texttt{int}\ j, k, m, n; \\
& j\ k, m\ n, j \xrightarrow{\text{align}} m, \big(\texttt{match}(\texttt{higher}(j) \xrightarrow{\text{mate}} \texttt{higher}(m))\big) \\
\Longrightarrow\ & \texttt{if}\big(\texttt{prob}(p_r(\mathit{dist}(j, k)))\big)(j\ n, m\ k)\ \texttt{else break}; \\
& g : \mathit{Genome} \xrightarrow{\text{mate}} \mathit{Genome} \Longrightarrow g
\end{aligned}
$$

Rule *Initialize* replaces the axiom α by a new *Population* containing a single *Genome*. The latter is represented as a string of nine integer nodes as in Figure 2c, the link is established by a $\xrightarrow{\text{first}}$ edge from the *Genome* to its first gene node, and the containment relation follows from this. In the next step, *Reproduce*, this configuration matches the left-hand side. The population p is retained in the new graph. There p no longer contains the genome g, but 15 biomorph *Germs* encoded by a copy (`cloneTree`) of g. The "encodes" edge symbolizes the genotype-phenotype relationship.

After reproduction all genes are subject to mutation: *Mutate* replaces a gene (represented as an integer value) with a random value, uniformly distributed over the integer range $-9, \ldots, 9$. The mutation probability is p_m.

In the first rule of *Grow* germination occurs, that is, each *Germ* is replaced by a new *Biomorph*. The actual geometric structure of a fully grown biomorph, its *phenotype*, will be represented by subordinate nodes, the first of which is $\mathbf{f}(x,0,0)$, which acts as a coordinate displacement in the specified direction (thus resembling the turtle command **f** in classical L systems) and ensures nonoverlapping biomorphs (the parameter x was set in *Reproduce* to $200i$ with the biomorph index i). The *Biom* node next to $\mathbf{f}(x, 0, 0)$ initiates the recursive growth process of the morphogenetic second rule of *Grow*. The initial *depth* parameter of *Biom* is taken from the genome g encoding the *Germ*: Essentially, the child node of g with index 8 is selected; the actual expression `abs`$(g[8]) + 1$ ensures positive recursion depth values. `abs` is built in like any other usual math function. The initial *dir* parameter is set to 2. Since g is enclosed in context parentheses (*...*), it is not considered as a part of the replacement process and remains in the graph, though not listed as a node on the right-hand side. However, the encoded object changes from *Germ* to *Biomorph*.

The morphogenetic second rule of *Grow* grows a biomorph according to Dawkins' model: A *Biom* node splits into two *Biom* branches with opposite changes of *dir*, preceded by the creation of a twig, which is implemented by an **F** node in our graph and by a line-drawing instruction in Dawkins' Pascal program. The $\mathbf{F}(x,y,z)$ node corresponds to the **F** command of turtle graphics, that is, it acts as a coordinate displacement for subordinate nodes and is also a visible line segment. Its (phenotypic) vector coordinates are determined via an intermediate step (cf. the calculation of the auxiliary variable d in the Java-like script section) by the genotype g, which is connected by an "encodes" edge to the root of the growing biomorph tree, namely the *Biomorph*. The growth process terminates when *depth* reaches 0.

SelectOne provides user interaction: If the user selects a *Biomorph*, the whole *Population* is replaced by a new one containing only the *Genome g* of the selected *Biomorph*. ^ represents the root node of the whole scene, so the new population becomes a child of this root node. Now the situation resembles that after *Initialize*, but with a possibly mutated genome. Biomorph evolution proceeds with *Reproduce* as described.

So far the procedure reproduces Dawkins' program. Upon replacing *SelectOne* with *SelectTwo* and *Recombine*, the grammar lets the user select two biomorphs and performs a crossing over of their genomes before creating a new biomorph population out of the mated genome: After the user has selected two biomorphs, rule *SelectTwo* replaces the whole population with a new one containing the genomes of the two selected biomorphs, connected by a "mate" edge indicating that crossing over is potentially to be carried out. The first rule of *Recombine* performs this crossing over as described in Section 3.3. The second rule discards one of the two genomes; the other one remains in the graph and leads to a new biomorph population when the grammar application proceeds with *Reproduce*.

The described order of rule application has to be specified as an imperative control structure:

Initialize : 1 *step*
`for`($i = 0$; $i <$ *nbgenerations*; $i++$){
  Reproduce : 1 *step*
  Mutate : 1 *step*
  Grow : * *steps*
  SelectOne : 1 *step* /* *or SelectTwo* : 1 *step*; *Recombine* : 1 *step* */
}

A sample population of 15 biomorph mutants is shown in Figure 5.

5 The *ABC* Model

One of the examples used by Kim [16] to illustrate the capabilities of transsys was the so-called *ABC model* of flower morphogenesis [2]: This model predicts the formation of the constituting organs of an angiosperm flower (sepals, petals, stamens, and carpels) as the result of the interaction between three homeobox genes and their products, that is, transcription factors. Put simply, genes *A* and *C* are mutually exclusive in their effect, whereas *B* "cooperates" with *A* and *C*: This leads to four zones or whorls along the floral primordium, in which either *A* alone is expressed (sepals), or *A* and *B* together (petals), or *B* and *C* together (stamens), or *C* alone (carpels). The corresponding regulatory network is depicted in Figure 6. This model correctly predicts the phenotype of the wild type as well as of mutants exhibiting a loss or gain of function of *A*, *B*, or *C* (Figure 7).

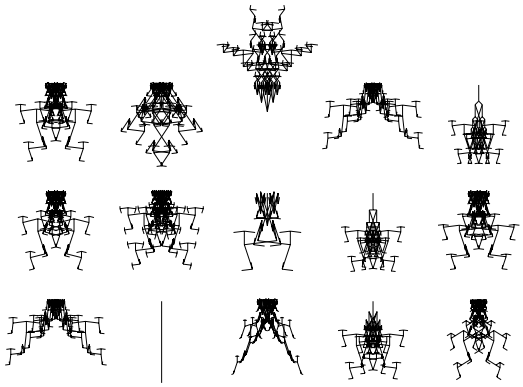

Figure 5. Population of mutants of the genome (1, −3, 8, −4, 5, −9, −4, 9, 8).

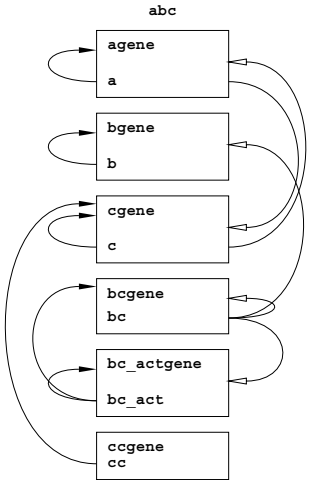

Figure 6. *ABC* model of flower morphogenesis: regulatory network. Upregulating (activating) effects of transcription factors are displayed as black-headed arrows to the left; downregulating (repressing) effects, as white-headed arrows to the right. From [16].

In order to implement the *ABC* model with RGG, it was necessary to break it down into three compartments (in analogy to the method used in transsys): (1) establishment of the network, (2) concentration dynamics of the network (these first two steps being done in [16] using the transsys formalism), and (3) translation of the concentrations of transcription factors into the floral phenotype using rules of development (this being carried out in [16] using a normal parametric L system accepting the temporarily changing concentration values of the transcription factors as the three parameters of a meristem node). Excerpts of the grammar used to produce the flowers of Figure 7 are as follows:

- Establishment of the network...

$$\begin{aligned}
\alpha \implies\ & a : \mathit{Factor}(0, 0.3) \xleftarrow{\text{encodes}} \mathit{agene} : \mathit{Gene}(0.1), \\
& b : \mathit{Factor}(0, 0.3) \xleftarrow{\text{encodes}} \mathit{bgene} : \mathit{Gene}(0.00000001), \\
& c : \mathit{Factor}(0, 0.3) \xleftarrow{\text{encodes}} \mathit{cgene} : \mathit{Gene}(0), \\
& \mathit{bc} : \mathit{Factor}(0, 0.6) \xleftarrow{\text{encodes}} \mathit{bcgene} : \mathit{Gene}(0), \\
& \mathit{bc_act} : \mathit{Factor}(0, 0.1) \xleftarrow{\textit{encodes}} \mathit{bc_actgene} : \mathit{Gene}(0.1), \\
& \mathit{cc} : \mathit{Factor}(0, 0) \xleftarrow{\text{encodes}} \mathit{ccgene} : \mathit{Gene}(0.01),
\end{aligned}$$

...activation/repression of the genes through factors...

a	$\mathit{Activate}(0.000000001, 50)$	agene,
c	$\mathit{Activate}(50, -100)$	agene,
b	$\mathit{Activate}(0.0003, 50)$	bgene,
bc	$\mathit{Activate}(80, -5000)$	bgene,
c	$\mathit{Activate}(10, 50)$	cgene,
cc	$\mathit{Activate}(0.5, 4)$	cgene,
a	$\mathit{Activate}(1500, -20)$	cgene,
$\mathit{bc\_act}$	$\mathit{Activate}(1, 5)$	bcgene,
bc	$\mathit{Activate}(1, -1)$	bcgene,
$\mathit{bc\_act}$	$\mathit{Activate}(0.05, 3)$	$\mathit{bc\_actgene}$,
bc	$\mathit{Activate}(0.1, -5)$	$\mathit{bc\_actgene}$,

...and establishment of a germ: The meristem is carrying the relevant factors (and, indirectly via the previously established network edges, the whole regulatory network):

$$\mathit{ShootPiece}(0, 0, 0, \mathit{SHOOT}, 0)\ \mathit{Meristem}[\xrightarrow{\text{first}} a\ b\ c\ cc]$$

- Dynamics of the network: Decrease in factor concentration due to chemical decay (as before, updating of variables takes place in parallel):

$$f : \mathit{Factor} \dashrightarrow f.\mathit{concentration} -= f.\mathit{concentration} * f.\mathit{decay}$$

- Dynamics of the network: Increase in factor concentration due to gene activity:

$$f : \mathit{Factor} \xleftarrow{\text{encodes}} g : \mathit{Gene}(ct) \dashrightarrow f.\mathit{concentration} += \texttt{max}(0, \texttt{sum}(((*\mathit{Factor}(c_2,)\ \mathit{Activate}(s, m)\ g\ *), m * c_2/(s + c_2) + ct)))$$

- Growth rule of the plant: Classification of the next *ShootPiece*

$$\begin{aligned}
& s : \mathit{ShootPiece}(, , , st,)m : \mathit{Meristem} \\
& \qquad (* \xrightarrow{\text{first}} \mathit{Factor}(a,)\mathit{Factor}(b,)\mathit{Factor}(c,)*) \\
& \Longrightarrow \{ \\
& \qquad \texttt{int}\ t = (b > 80)\ ?\ ((c > a)\ ?\ \mathit{STAMEN} : \mathit{PETAL})
\end{aligned}$$

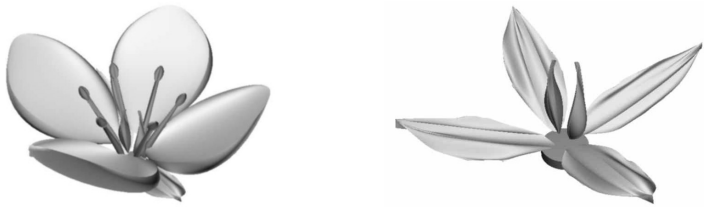

Figure 7. Simulated phenotype of wild type (left) and loss-of-*B* mutant (right) flower.

$$: (a > 80)\ ?\ ((c > 80)\ ?\ \mathit{SHOOT} : \mathit{SEPAL})$$
$$: (c > 80)\ ?\ \mathit{CARPEL} : \mathit{PEDICEL};$$
}

distinction between types of *ShootPiece*. If it has the same type as its predecessor and it is not *SHOOT*, then increase its mass:

`if` $((t\ != \mathit{SHOOT})\ \&\&\ (t == \mathit{st}))$
$(\{s.\mathit{mass}\ +\!= 1;\}$ `break`$)$

else insert a new *ShootPiece* ...

`else`
$(s\ \mathit{ShootPiece}(a, b, c, t, 0)\ m)$

- Termination of growth at certain concentrations:
$\mathit{Meristem}\ (* \xrightarrow{\text{first}} \mathit{Factor}\ \mathit{Factor}\ \mathit{Factor}(c,)\ \mathit{Factor}(cc,\ )\ *),$
$(c > 80\ \&\&\ cc > 1)$
$\Longrightarrow \varepsilon$

- Phenotypical expression of *ShootPiece* if its successor is not a further meristem:

$s : \mathit{ShootPiece}(a, b, c, st, mass),$ (! `match`$(s\ \mathit{Meristem})$)
$\Longrightarrow$ *graphical representation of floral organs*

The establishment of the regulatory network in our formalism consists of the explicit description of the genes *agene*, *bgene*, and *cgene* (as well as three further genes: *bcgene*, *bc_actgene*, and *ccgene*, which are necessary to start off the dynamics of the network) and their products by declaring explicit instances of the classes *Gene* and *Factor*, where the one parameter in *Gene* as well as the first parameter of *Factor* represents the initial concentration, and the second parameter of *Factor* its rate of decay. The production of a transcription factor is represented by an $\xrightarrow{\text{encodes}}$ edge from an instance of *Gene* to an instance of *Factor*.

In the next step, the module *Activate*[2] with two parameters, *specificity* (of a factor to a gene) and *max* (maximum activation possible), is used to set the activating or repressing relationships of the transcription factors *a*, *b*, *c*, *cc*, *bc*, and *bc_act* to the

2 The module *Activate* is an auxiliary RGG node that transmits relevant information along its proper edges (established at the same time) between gene and factor nodes, thereby determining the nature (activation or repression) and dimension of the ensuing Michaelis-Menten reaction. Thus, from the perspective of RGG, the module *Activate* is a type of parametric edge.

six genes. The two parameters of *Activate* are analogous to the Michaelis-Menten parameters K_M and $v_{\max}$ [16]. When representing a repressing relationship, *max* is given a negative value, thereby eliminating the need for a second function, *Repress*. In the last part of the initialization production, a meristem is produced and connected to the transcription factor network via a $\xrightarrow{\text{first}}$ edge.

The next two rules represent the dynamics of the network: (1) the linear decrease in concentration of any factor due to chemical decay, and (2) the increase of the concentration of a given factor due to the transcription activity of its coding gene, the latter on the basis of the Michaelis-Menten equation [16]. Note that—due to the use of the special transformation rules ($\dashrightarrow$)—the actual nodes and edges of the existing network are not modified, only the values of (some of) their parameters. In fact, the semantics of these rules is just the execution of their right-hand sides as conventional imperative code for every occurrence of the left-hand side pattern in the graph.

The next rule constitutes the actual growth rule: It looks for an unspecific module of type *ShootPiece* (i.e., a whorl of the later flower) with a floral *Meristem* on its top, the latter having a context consisting of the three transcription factors. The concentrations of these three factors (*a*, *b*, and *c*)[3] are the first three parameters of the module *ShootPiece*, the fourth being an integer variable specifying the type of organ produced, and the fifth the size or mass of the produced shoot. Thus, depending on the factor concentrations, the organ type is set to one of five possibilities, corresponding to the four floral organs, plus *SHOOT*, which is the organ at the base.

The penultimate rule stops further development of the module *Meristem* at a given concentration of the factors *c* and *cc* by producing the empty word ε; the last one determines the phenotypical expression or translation of a *ShootPiece* module into any of the four floral organs, if it is not succeeded by a further meristem.

6 Discussion and Conclusions

The examples of the biomorph and the *ABC* model have both shown that RGGs can model complex genetic operations such as multiple crossing over and equally complex functional genetic networks while at the same time being flexible enough to encode the ramified architecture of the phenotype. This universality—in a practical sense—is confirmed by the successful reimplementation of other classical ALife scenarios in our formal framework. First of all, our grammars are proper extensions of L systems—hence all plant models obtained with L systems (see, e.g., [26]) can be reproduced with our system.

Another calculus that has received some attention in the ALife community is *artificial chemistry* (cf. [11]), that is, the attempt to model the dynamics of large numbers of artificial "molecules" (which can be numbers, code fragments, graphs, or other abstract objects with nontrivial pairwise interaction) in a virtual solution. A simple example is a "chemical" prime number generator [31], where the molecules are integers and their interaction in the case of collision is expressed by the following grammar rule (in our notation):

$$a{:}\texttt{int},\ b{:}\texttt{int},\ (b\ \texttt{mod}\ a == 0) \longrightarrow a,\ b/a$$

3 The factors *bc* and *bc_act* are used to initialize the network. Their concentrations are not relevant for the process proper of floral morphogenesis and are therefore not passed on to the meristem symbol. However, the concentration of factor *cc* is passed on to the meristem to control the end of floral development.

Given an initial "soup" (multiset) of random integers, the iterated and nondeterministic application of the above rule leads to an increase of the "concentration" of prime numbers in the solution, finally approaching 100%.

One of the most prominent formalisms in ALife research is the cellular automaton (CA). The underlying grid of a CA can be constructed in our approach either by using a purely geometric definition of a neighborhood relation or by iterating rules associated with the symmetry group of the grid. Then the cells of the CA are the nodes of our graph, and the *context* of a cell can be defined as a multiset- or vector-valued function using the neighborhood definition of the grid. For example, the transition rule of Conway's famous game of life [14] can be expressed as follows:

$$\begin{aligned} &x{:}\mathit{Cell}(1),\ \big(!(\texttt{sum}(\mathit{context}(x))\ \texttt{in}\ \{2,3\})\big) &\Longrightarrow\quad & x(0)\\ &x{:}\mathit{Cell}(0),\ \big(\texttt{sum}(\mathit{context}(x)) == 3\big) &\Longrightarrow\quad & x(1) \end{aligned}$$

where x remains unchanged in all cases not covered by the conditions.

A last example refers to ecological modeling. Interactions between species (e.g., food webs) can also be expressed as graphs. A flow of resources between two individuals—for example, between a plant and a herbivore grazing on it—can be represented by a rule with a pair of connected objects on the left side. The third author has implemented an individual-based model of plant and animal populations with traditional L systems [21], using some tricks. In our formalism, the *grazing* rule takes a very simple form:

$$\mathit{Animal}(u) \xrightarrow{\text{grazes}} \mathit{Plant}(v) \Longrightarrow \ldots \mathit{Animal}(u + \mathit{eat}) \ldots \mathit{Plant}(v - \mathit{eat}) \ldots ;$$

where u and v represent the biomasses. (The exact structure of the right side depends on the concrete geometry and behavior of the considered plants and animals.) Traits like consumption efficiency or foraging strategy can again be encoded in associated genome strings and can be subject to mutation and selection. In contrast to the biomorph example, selection will be based on an intrinsic fitness function, such as competitiveness in finding and exploiting resources.

RGG thus promise to be a nearly universal tool, allowing one to specify models of biological objects at diverse levels, from the molecule to the ecosystem.[4]

The ever increasing flood of information coming from both traditional biological disciplines such as morphology or classical genetics and newly created subjects such as transcriptomics, proteomics, or metabolomics has led in the recent past to a similar wealth of computational tools. Though this is essentially a positive development, many of these tools lack universality and transparency: Usually being most appropriate for a given task within a given context, they often cannot be applied to a different discipline, a different organism, or a different hierarchical scale (cell, tissue, organ, organism). This contextual rigidity is most often due to some implicit structural rigidity in the software. As a consequence, a simple interfacing of different models via the exchange of certain data types in an interdisciplinary fashion becomes insufficient as soon as the (usually high) complexity of the observed biological system with its new qualities of both parameters and mutual relationships is considered more completely—beyond some abstracted, idealized, or "purified" perspective. Furthermore, as the focus in current molecular biological research is shifting away from the analysis of DNA and protein sequences to the analysis of development at different scales, a new generation of programming tools is required. Kumar and Bentley [18] list five main processes involved in biological development: cleavage divisions, pattern formation, morphogenesis, cel-

4 Further information and links to a Java-based implementation of RGGs can be found at http://www.grogra.de.

lular differentiation, and growth. All these processes have in common that they may act or be perceived at different hierarchical levels and that their effects are hard (if not impossible) to predict by biological-analytical tools alone. Consequently, according to these two authors, the new levels of knowledge reached in the postgenomic era have to be matched by a parallel paradigm in artificial life, which they call *computational development* (CD) and which is the more or less abstracted modeling and simulation of development in silico, at the level of both individual organogenesis and organismal evolution. The difference between CD and the usual genetic algorithms or other genotype-phenotype models is that in CD models one deliberately introduces structural complexity (e.g., by representing protein domains as fractals), thereby prolonging the path between the genotype and phenotype, which in usual AL models is often too short or missing altogether. Precursors or parallel developments of CD models were, among others, reaction-diffusion systems, activator-inhibitor models, random Boolean networks, cellular encoding, evolutionary neurogenesis, evolutionary 2D morphogenesis, and, of course, L systems. The formalism of extended L systems, presented here, tries to overcome some of the problems of these classical approaches by loosening the rigidity in providing a universal modeling language that is at the same time transparent enough to be understood and applied by computational amateurs such as biologists or agronomists. We believe that these two features, universality and transparency, are essential prerequisites to tackle future problems in biological modeling and computational development, which will more and more involve the direct interaction between computational and biological scientists.

Acknowledgments

This research was funded by the DFG under grant Ku 847/5-1 in the framework of the research group "Virtual Crops." All support is gratefully acknowledged. The second author thanks IPK, in particular Dr. Patrick Schweizer, for providing office facilities while the author was a guest researcher at the institute. We also thank the two anonymous reviewers for valuable comments.

References

1. Buck-Sorlin, G. H., & Bachmann, K. (2000). Simulating the morphology of barley spike phenotypes using genotype information. *Agronomie: Plant Genetics and Breeding, 20,* 691–702. http://taxon.ipk-gatersleben.de.
2. Coen, E.S., & Meyerowitz, E.M. (1991). The war of the whorls: Genetic interactions controlling flower development. *Nature, 353,* 31–37.
3. Costes, E., Sinoquet, H., Kelner, J.J., & Godin, C. (2003). Exploring within-tree architectural development of two apple tree cultivars over 6 years. *Annals of Botany, 91,* 91–104.
4. Csuhaj-Varjú, E. (1995). Eco-grammar systems: Recent results and perspectives. In G. Păun, (Ed.), *Artificial life: Grammatical models* (pp. 79–103). Bucharest, Romania: Black Sea University Press.
5. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., & Păun, G. (1994). *Grammar systems: A grammatical approach to distribution and cooperation*. Yverdon, Switzerland: Gordon and Breach.
6. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., & Păun, G. (1997). Eco-grammar systems: A grammatical framework for studying lifelike interactions. *Artificial Life, 3,* 1–28.
7. Dassow, J., & Mitrana, V. (1997). Evolutionary grammars: A grammatical model for genome evolution. In R. Hofestädt, T. Lengauer, M. Löffler, & D. Schomburg, (Eds.), *Bioinformatics. German Conference on Bioinformatics, GCB'96. September 30–October 2, 1996, Leipzig, Germany* (pp. 199–209). Berlin: Springer-Verlag.
8. Dawkins, R. (1986). *The Blind Watchmaker*. Harlow, UK: Longman.

9. Dawkins, R. (1988). The Evolution of Evolvability. In C. Langton, (Ed.), *Artificial life, SFI studies in the sciences of complexity* (pp. 201–220). Reading, MA: Addison-Wesley.

10. Ehrig, H., Heckel, R., Korff, M., Löwe, M. Ribeiro, L., Wagner, A., & Corradini, A. (1997). Algebraic approaches to graph transformation—Part II: Single pushout approach and comparison with double pushout approach. In G. Rozenberg, (Ed.), *Handbook of graph grammars and computing by graph transformation, Vol. 1, Foundations* (pp. 247–312). Singapore: World Scientific.

11. Fontana, W. (1991). Algorithmic chemistry. In C. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen, (Eds.), *Artificial life II, SFI studies in the sciences of complexity* (pp. 159–209). Reading, MA: Addison-Wesley.

12. Freund, R. (1995). Multi-level eco-array grammars. In G. Păun, (Ed.), *Artificial life: Grammatical models* (pp. 175–201). Bucharest, Romania: Black Sea University Press.

13. Frijters, D., & Lindenmayer, A. (1974). A model for the growth and flowering of *Aster novaeangliae* on the basis of table (1,0) L-systems. In G. Rozenberg, & A. Salomaa, (Eds.), *L systems. Lecture Notes in Computer Science, 15* (pp. 24–52). Berlin: Springer-Verlag.

14. Gardner, M. (1983). *Wheels, life, and other mathematical amusements.* New York: W. H. Freeman.

15. Godin, C., & Caraglio, Y. (1998). A multiscale model of plant topological structures. *Journal of Theoretical Biology, 191,* 1–46.

16. Kim, J. (2001). A generic formalism for modelling regulatory networks in morphogenesis. In J. Kelemen, & P. Sosík, (Eds.), *Advances in artificial life. Lecture Notes in Artificial Intelligence, 2159* (pp. 242–251). Berlin: Springer-Verlag.

17. Kniemeyer, O., Buck-Sorlin, G.H., & Kurth, W. (2003). Representation of genotype and phenotype in a coherent framework based on extended L-systems. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, & J. Ziegler (Eds.), *Advances in Artificial Life. Lecture Notes in Artificial Intelligence, 2801* (pp. 625–634). Berlin: Springer-Verlag.

18. Kumar, S., & Bentley, P.J. (2003). An introduction to computational development. In S. Kumar, & P. J. Bentley, (Eds.), *On growth, form and computers* (pp. 1–43). Amsterdam: Elsevier Academic Press.

19. Kurth, W. (2002). Spatial structure, sensitivity and communication in rule-based models. In F. Hölker, (Ed.), *Scales, hierarchies and emergent properties in ecological models. Theorie in der Ökologie, 6* (pp. 29–46). Frankfurt, Germany: Peter Lang Verlag.

20. Kurth, W. (1994). Growth grammar interpreter GROGRA 2.4: A software tool for the 3-dimensional interpretation of stochastic, sensitive growth grammars in the context of plant modelling. Introduction and reference manual. *Berichte des Forschungszentrums Waldökosysteme der Universität Göttingen, Ser. B, 38.*

21. Kurth, W., & Sloboda, B. (2001). Sensitive growth grammars specifying models of forest structure, competition and plant-herbivore interaction. In *IUFRO 4.11 Congress "Forest Biometry, Modelling and Information Science."* Greenwich, UK. http://cms1.gre.ac.uk/conferences/iufro/proceedings/.

22. Lane, B., & Prusinkiewicz, P. (2002). Generating spatial distributions for multilevel models of plant communities. In *Proceedings of Graphics Interface 2002, Calgary, May 27–29* (pp. 69–80). Mississauga, Canada: Canadian Information Processing Society.

23. Lindenmayer, A. (1968). Mathematical models for cellular interactions in development, Parts I and II. *Journal of Theoretical Biology, 18,* 280–315.

24. Mech, R., & Prusinkiewicz, P. (1996). Visual models of plants interacting with their environment. In *Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH* (pp. 397–410).

25. Prusinkiewicz, P., Hanan, J., & Mech, R. (2000). An L-system-based plant modeling language. In M. Nagl, A. Schürr, & M. Münch, (Eds.), *Proceedings of the International Workshop AGTIVE'99. Lecture Notes in Computer Science, 1779* (pp. 395–410). Berlin: Springer-Verlag.

26. Prusinkiewicz, P., & Lindenmayer, A. (1990). *The algorithmic beauty of plants*. New York: Springer-Verlag.

27. Prusinkiewicz, P., Mündermann, L., Karwowski, R., & Lane, B. (2001). The use of positional information in the modeling of plants. In *Proceedings of SIGGRAPH 2001* (pp. 289–300). Los Angeles.

28. Rozenberg, G. (1973). T0L systems and languages. *Information and Control, 23,* 357–381.

29. Schürr, A. (1997). Programmed graph replacement systems. In G. Rozenberg, (Ed.), *Handbook of graph grammars and computing by graph transformation, Vol. 1: Foundations* (pp. 479–546). Singapore: World Scientific.

30. Schürr, A., Winter, A.J., & Zündorf, A. (1999). The PROGRES approach: Language and environment. In G. Rozenberg, (Ed.), *Handbook of graph grammars and computing by graph transformation, Vol. 2: Applications, languages and tools* (pp. 487–550). Singapore: World Scientific.

31. Skusa, A., Banzhaf, W., Busch, J., Dittrich, P., & Ziegler, J. (2000). Künstliche Chemie. *Künstliche Intelligenz, 1/00,* 12–19.