Representation of genotype and phenotype in a coherent framework based on extended L-Systems

Ole Kniemeyer[1], Gerhard H. Buck-Sorlin[1 2], and 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
[2] Institute of Plant Genetics and Crop Plant Research, Dept. Cytogenetics, Corrensstr. 3, D-06466 Gatersleben, Germany

Abstract. A formal language approach for the specification of ALife models is presented. "Relational Growth Grammars" incorporate rule-based, procedural and object-oriented concepts. By enabling parametric Lindenmayer systems to rewrite multiscaled graphs, it becomes possible to represent genes, plant organs and populations as well as developmental aspects of these entities in a common formal framework. Genetic operators (mutation, crossing-over, selection) take the form of simple graph rewrite rules. This is illustrated using Richard Dawkins' "biomorphs", whereas other applications are briefly sketched. The formalism is implemented as part of an interactive software platform.

1 Introduction

High-level, rule-based languages exhibit several features that make them suitable for biological modelling: Being both transparent in use and not requiring constant re-compilation of the code after modification (thanks to powerful string parsers), they could potentially be developed into a universal language 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(indenmayer)-systems in 1968 [12], these string rewriting grammars have been primarily used to model growth and architecture of individual plants [15]. 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 Nineties that "globally sensitive" and "open" L-systems took the environment into account [10, 13]). Nevertheless, no genetics whatsoever was usually represented in such L-systems, despite of the double-string structure of DNA which seems to predestinate it to be described in a string-rewriting formalism. The "interactive barley" model by Buck-Sorlin and Bachmann [1] is one exception, despite its not properly exploiting the string nature of the genome. Dassow and Mitrana [2] have devised a formal string

rewriting language called "evolutionary grammars" to represent genetic operators (e.g., mutation, recombination). However, their formalism is restricted to the genetic level and is unable to model architectural or behavioral aspects of the phenotype. Kim [9] has developed `transsys`, a software tool enabling the specification and dynamic simulation of gene expression, transcription factors and L-system-based morphogenetic processes. Though this is a step in the right direction, `transsys` treats genetic activity and macroscopic morphogenesis with different formalisms.

Godin and Caraglio [8] show how the integration of several scaling levels in one coherent framework could be done: They defined "multiple-scaled tree graphs" (MTGs) to describe branching structures simultaneously at several degrees of spatial resolution. Although their approach is basically static and does not encompass the genetic level, it served as a model for our "relational growth grammar" data structures in which binary relations play a key role.

In the following, we will sketch our improved formal language and show its applicability to ALife model specification by "reincarnating" the biomorphs from Richard Dawkins' book "The Blind Watchmaker" [3]. In the Discussion and Conclusions, our framework will be exemplified using other ALife formalisms, and the perspectives of our approach will be shown.

2 Relational Growth Grammars

2.1 The data structures

Typical data structures in biological models can be roughly categorized as follows:

Multisets, i.e. unordered collections where the same element can appear several times (examples: unstructured populations, molecules in a solution). L-systems operating on multisets were introduced in [11].

Strings (equivalent to 1-dimensional *arrays* or *lists*) with strict linear order amongst the elements. In our graph-related approach we will use special directed edges of type "successor" to connect elements which 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 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 [8].—Biological examples are genomes and linearly ordered architectural substructures (e.g., tree branches).

Axial trees allow to combine strings by a "lateral branch" relation, corresponding to Godin's "+" and symbolized by the bracket notation from classical L-systems [15] where the brackets are used to encode branching in a string and only become important in the (turtle) interpretation, whilst in our model new branches instantly arise at rule application. Axial trees in nature correspond to botanical trees (at the macroscopic level).

Relational structures (or edge-labelled directed graphs) generalize all these basic structures. Here, arbitrary user-defined types of edges are permitted, representing relationships not yet covered by the above basic structures—e.g., the relation between genotype and phenotype, or the "contains" relation connecting compound entities with their parts (Godin's "/"). Using the "successor" and the "contains" relation, a string can be represented as in Fig. 1.

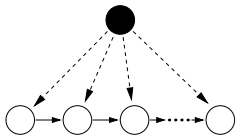

Fig. 1. A string represented by an edge-labelled graph. Continuous edges denote the successor relation, dashed edges the "contains" relation. The filled "basal" node represents an object at a lower level of resolution (multiscaled approach, cf. [8]).

The structures are implemented with Java: Each graph node is an object in the sense of object-oriented programming. Every node pair can be connected by a set of labelled edges. This flexible data structure permits communication between nodes, by sending messages along a certain edge type. It is also used in our interactive application to constitute an easy-to-use user interface, e.g. interaction controls can be connected to data nodes by edges.

2.2 The formalism

A documentation with complete definitions of the concepts used here will be published elsewhere. Here we list only the main items characterising the Relational Growth Grammar calculus:

- *Graph grammars* are used to rewrite the data structures described above. Contrary to classical L-systems, several nodes (generalising the symbols in L-systems) can appear on the left side of a rule. In the most general case, entire subgraphs are rewritten. Our graph grammars, using both node and edge labels, were partly inspired by the PROGRES system (see [18]).
- Commands of *turtle geometry* (originally from parametric L-systems and in the systematized variant used by the Grogra software [10]) are allowed as nodes and serve to interpret the generated graphs geometrically if necessary. E.g., the command "**F**" (without parameters, with a length, or with displacement coordinates as parameters) causes the creation of a solid cylinder representing, e.g., a (botanical) internode.
- Variables, functions and relations can be defined by the user. (The declaration of several types of random variables was already implemented in Grogra [10].) Procedural programs (scripts) in a Java-like language can be inserted

into a rule and will be executed when the rule is applied (cf. [16] for C fragments in L-systems).
- We use a variant of *table L-systems* ([17]; cf. "sub-L-systems", [14]) to control directly the order in which certain groups of rules are to be applied—mainly for transparency and convenience reasons; the order of rule application could also be encoded in additional parameters and conditions.
- Essential to all rule-based languages are the processes of *matching* and *replacement*, into which the application of a rule can be decomposed. Three 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, in other transformation systems, e.g. in CA, and also in our formalism. When using this mode one has to be aware of possible resolution conflicts with overlapping matches. The other two modes are *sequential matching and replacement immediately after each match* and *sequential replacement*, i.e. in each rewriting step only one match is exploited. For our examples only parallel matching and replacement are needed as resolution conflicts do not occur.
- *Conditions* can be specified to control the matching process, *probabilities* act upon replacement (cf. [15]).
- *Graph contexts* can be defined in a general manner, in particular they are not confined to left and right contexts in the sense of string matching.

2.3 Example: Crossing-over

To show the potential of our new formalism, we chose the example of crossing-over (co), an important biological process essentially consisting in the exchange of aligned, homologous substrings (alleles) of matching chromosomes (cf. Fig. 2).

This process can be modelled using classical L-systems if the number of genes involved is low. With more than a few genes, the exhaustive listing of all possible multiple co events will become too long. In our new formal framework, we will represent the genomes by strings which are associated with the organisms by edges of a graph. Co will be modelled by one single graph-rewrite rule.

To encode co events, we need two additional relations (i.e., edge types): "Mating" between the two strings which are (potential) objects of co, and "alignment" between homologous gene loci. The latter can be defined on a pair j, k of nodes representing a single gene using the built-in function "index", which returns the position of an item within a string:

$$j \overset{\text{align}}{\text{———}} k \quad := \quad \text{index}(j) == \text{index}(k)$$

Co can then be described by the graph rewrite rule shown in Figure 2a, which can be written down as follows:

$$j\,k,\ m\,n,\ j \overset{\text{align}}{\text{———}} m, \big(\text{match}(\text{base}(j) \overset{\text{mate}}{\text{———}} \text{base}(m))\big) \longrightarrow \big(\text{prob}(p_r(\text{dist}(j,k)))\big)\ ?\ j\,n,\ m\,k$$

The left hand side lists the "successor" relation between j and k, 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 Fig. 2a, can be omitted in the condition because it follows from $j \overset{\text{align}}{\frown} m$. 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 2a, otherwise the rule is not applied. —Matching of the rule is checked at all possible positions, thus making double or multiple co's in extra rules (as would have to be done with classical L-systems) dispensable. In order to prevent the "mirror match" resulting from the (j,k)-(m,n)-symmetry of the left hand side no two matches may consist of the same sets of matched nodes.

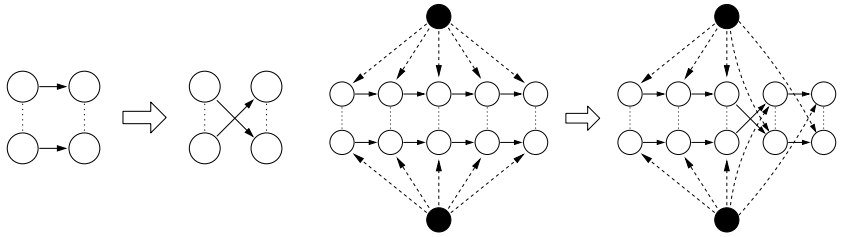

Fig. 2. A relational growth grammar rule representing co (a) and its application to a pair of strings (b). Continuous edges denote the "successor" relation, dashed edges containment, and point-edges alignment.

3 The example of Biomorphs

We will now apply the new formalism, including the co rule from the last section, to a well-known example, the "Blind Watchmaker" programme, written by the zoologist Richard Dawkins [3] and inspired by the seemingly blind phenomenon of evolution by mutation and selection. Based on a simple genotype-phenotype model [4], it originally consists of two Pascal procedures: The first reproduces the genotype, thereby making new individuals and introducing some random chance mutation. In the second the development of the phenotype is modelled using a recursive tree-drawing routine with the genes' values as parameters (depth of recursion, direction of branches). The genotype consists of nine genes, eight of which have 19 alleles ranging from -9 over 0 to +9, the ninth a range from 0 to 9, in integral steps. The latter gene determines recursion depth. The other genes are translated into components of a matrix d which represent the horizontal, resp. vertical setoffs in a global coordinate system to be used in the development of the growing binary tree. The programme is started off with an initial set of

alleles, which 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 selects one of the offspring for further reproduction. By piling-up mutations in a certain direction a shortcut is taken through the multidimensional genotypic parameter space with its billions of possibilities, thereby arriving at astonishing "biomorphs". Our relational growth grammar will furthermore provide a possibility to select *two* parent individuals from which offspring is generated using the co rule shown above.

The following table of relational growth grammar rules contains both alternatives, simple vegetative reproduction (using "SelectOne") and reproduction from two parents involving co (using "SelectTwo" and "Recombine" instead).

Initialize: $\alpha \longrightarrow \mathit{population} \xrightarrow{\text{contains}} \mathit{genome}\ [1\ 1\ 1\ 1\ 1\ 0\ {-1}\ {-1}\ 5]$

Reproduce: $p{:}\mathit{population} \xrightarrow{\text{contains}} g{:}\mathit{genome}$
$\longrightarrow\ p\ \&\big(i{=}0..14,\ [\xrightarrow{\text{contains}}\ \mathit{germ}(200{*}i) \xleftarrow{\text{encodes}} g.\text{cloneTree}]\big)$

Mutate: $\text{int} \longrightarrow \big(\text{prob}(p_m)\big)\ ?\ \text{irandom}(-9,\ 9)$

Grow: $\mathit{germ}(x)\ (*\xleftarrow{\text{encodes}} g{:}\mathit{genome}\ *)$
$\longrightarrow\ \mathit{biomorph}\ \mathbf{f}(x,0,0)\ \mathit{biom}(\text{abs}(g[8])+1,\ 2);$
$b{:}\mathit{biom}(\mathit{depth},\mathit{dir}),$
$\big(\text{match}(\text{root}(b) \xleftarrow{\text{encodes}} g{:}\mathit{genome})\ \&\&\ (\mathit{depth} > 0)\big)$
$\longrightarrow\ \big\{\text{int}[\,][\,]\ d = \big\{\{-g[1], g[5]\}, \{-g[0], g[4]\}, \{0, g[3]\}, \{g[0], g[4]\},$
$\{g[1], g[5]\}, \{g[2], g[6]\}, \{0, g[7]\}, \{-g[2], g[6]\}\big\};\big\}$
$\mathbf{F}(\mathit{depth}{*}d[\mathit{dir} \bmod 8][0],\ 0,\ \mathit{depth}{*}d[\mathit{dir} \bmod 8][1])$
$[\mathit{biom}(\mathit{depth}{-}1,\ \mathit{dir}{-}1)]\ [\mathit{biom}(\mathit{depth}{-}1,\ \mathit{dir}{+}1)]$

SelectOne: $\mathit{population} \xrightarrow{\text{contains}} b{:}\mathit{biomorph} \xleftarrow{\text{encodes}} g{:}\mathit{genome},\ \big(\text{isSelected}(b)\big)$
$\longrightarrow\ \hat{}\,\mathit{population} \xrightarrow{\text{contains}} g$

SelectTwo: $\mathit{population}\ [\xrightarrow{\text{contains}} b_1{:}\mathit{biomorph} \xleftarrow{\text{encodes}} g_1{:}\mathit{genome}]$
$[\xrightarrow{\text{contains}} b_2{:}\mathit{biomorph} \xleftarrow{\text{encodes}} g_2{:}\mathit{genome}],$
$\big(\text{isSelected}(b_1)\ \&\&\ \text{isSelected}(b_2)\big)$
$\longrightarrow\ \hat{}\,\mathit{population}\ [\xrightarrow{\text{contains}} g_1]\ [\xrightarrow{\text{contains}} g_2],\ g_1 \xrightarrow{\text{mate}} g_2$

Recombine: $\text{int}\ j,\ k,\ m,\ n;$
$j\ k,\ m\ n,\ j \xrightarrow{\text{align}} m,\ \big(\text{match}(\text{base}(j) \xrightarrow{\text{mate}} \text{base}(m))\big)$
$\longrightarrow\ \big(\text{prob}(p_r(\text{dist}(j,k)))\big)\ ?\ j\ n,\ m\ k;$
$g{:}\mathit{genome} \xrightarrow{\text{mate}} \mathit{genome} \longrightarrow g$

Rule "Initialize" replaces the axiom α by a new *population* containing a single *genome* (a string consisting of nine integer nodes). In the next step, "Reproduce", this configuration matches the left hand side. The population p is retained in the new graph, but there it 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. The rule makes use of the repetition operator &(range, ...).

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, i.e., each *germ* is replaced by a new *biomorph*. The actual geometric structure of a full-grown biomorph, its phenotype, will be represented by subordinate nodes, the first of which is the **f**(x,0,0)-node which acts as a coordinate displacement in the specified direction (thus resembling the turtle command "**f**" in classical L-systems) and ensures non-overlapping biomorphs (the parameter x was set in "Reproduce" to 200*i with the biomorph index i). The *biom*-node next to **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 programme. The **F**(x,y,z)-node corresponds to the **F**-command of turtle graphics, i.e., 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' programme. Upon substituting "SelectOne" by "SelectTwo" and "Recombine" the grammar lets the user select two biomorphs and performs co 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 co is potentially to be carried out. The first rule of "Recombine" performs

this co as described in section 2.3. 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".

This sequence of rule application can be specified formally as a table:

```
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 Fig. 3.

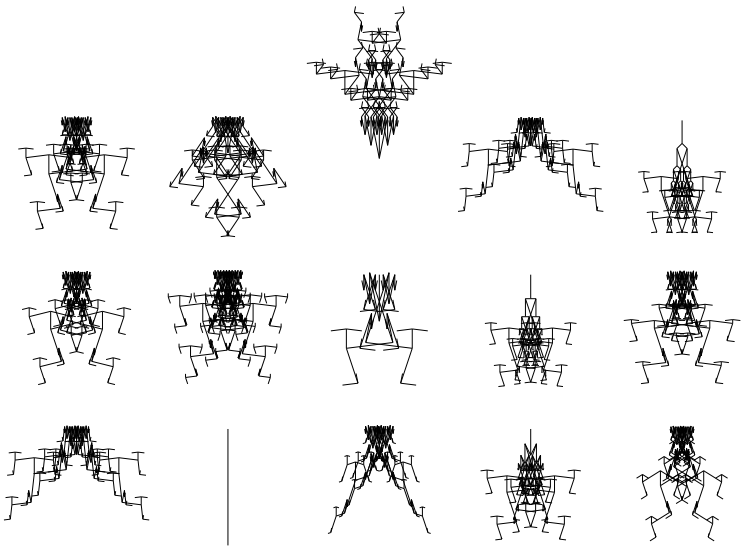

Fig. 3. Population of mutants of the genome $(1, -3, 8, -4, 5, -9, -4, 9, 8)$

4 Discussion and Conclusions

Relational growth grammars can model complex genetic operations like multiple co, while at the same time being flexible enough to model the ramified architecture of the phenotype (see biomorph example). This universality is confirmed by the successful re-implementation of other classical ALife scenarios in our formal framework. Notably, our grammars are proper extensions of L-systems, hence all plant models obtained with L-systems (e.g., [15]) can be reproduced.

Another calculus which got some attention in the ALife community is Artificial Chemistry (cf. [6]), i.e., a model of the dynamics of large numbers of artificial

"molecules" in a virtual solution. A simple example is a "chemical" prime number generator [19], where the molecules are integers and their interaction in the case of collision is expressed by the following grammar rule:

$$a{:}\text{int},\ b{:}\text{int},\ (b \bmod a == 0) \longrightarrow a,\ b/a$$

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%.

Cellular Automata (CA) are another good example: The underlying grid of a CA could be constructed in our approach, e.g., 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 neighbourhood definition of the grid. E.g., the transition rule of Conway's famous Game of Life [7] can be expressed as follows:

$$\begin{aligned} x{:}\text{int},\ &\big(x == 1\ \&\&\ x.\text{context.sum in } \{2,3\}\big) &&\longrightarrow 0\\ x{:}\text{int},\ &\big(x == 0\ \&\&\ x.\text{context.sum} == 3\big) &&\longrightarrow 1 \end{aligned}$$

If one were to criticize the wealth of computational tools created as an answer to the flood of biological information produced in the recent past, one should mention the lack of universality and transparency characterising many of them: They are often tailored to a specific problem and cannot be applied to a different discipline or a different organism. This rigidity is mostly due to some implicit structural rigidity "hardwired" within the source code. The extended L-system approach presented here tries to overcome this problem by providing a universal modelling language which is still transparent enough to be understood and applied by biologists or agronomists. We believe that universality and transparency are essential prerequisites to tackle future problems in biological modelling.

5 Acknowledgements

This research was funded by the DFG under grant Ku 847/5-1 (research group "Virtual Crops"). The second author thanks Dr. Patrick Schweizer (IPK) for providing office facilities. All support is gratefully acknowledged.

References

1. Buck-Sorlin, G.H., Bachmann, K.: Simulating the morphology of barley spike phenotypes using genotype information. Agronomie: Plant Genetics and Breeding **20** (2000) 691–702; s.a. http://taxon.ipk-gatersleben.de.
2. Dassow, J., Mitrana, V.: Evolutionary grammars: A grammatical model for genome evolution. - In: R. Hofestädt et al. (eds.): Bioinformatics. German Conference on Bioinformatics, GCB'96. September 30 - October 2, 1996, Leipzig, Germany. Springer, Berlin (1997) 199–209.
3. Dawkins, R.: The Blind Watchmaker. Longman, Harlow (1986).

4. Dawkins, R.: The Evolution of Evolvability. - In: C. Langton (ed.): Artificial Life, SFI Studies in the Sciences of Complexity, Addison-Wesley, Reading (1988) 201–220.
5. de Boer, M.J.M., Lindenmayer, A.: Map OL-Systems with Edge Label Control: Comparison of Marker and Cyclic Systems. - In: Proc. 3rd Int. Workshop on Graph-Grammars and Their Application to Computer Science, LNCS **291** (1987) 378–392.
6. Fontana, W.: Algorithmic chemistry. - In: C. G. Langton, C. Taylor, J. D. Farmer, S. Rasmussen (eds.), Artificial Life II, SFI Studies in the Sciences of Complexity, Addison-Wesley, Reading (1991) 159–209.
7. Gardner, M.: Wheels, Life, and Other Mathematical Amusements. W. H. Freeman. New York (1983).
8. Godin, C., Caraglio, Y.: A multiscale model of plant topological structures. J. Theor. Biol. **191** (1998) 1–46.
9. Kim, J.: `transsys`: A Generic Formalism for Modelling Regulatory Networks in Morphogenesis. In: J. Kelemen, P. Sosk (Eds.): Advances in Artificial Life. LNAI **2159** (2001) 242–251.
10. Kurth, W.: 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** (1994) 190 p.
11. Lane, B., Prusinkiewicz, P.: Generating spatial distributions for multilevel models of plant communities. Proceedings of Graphics Interface 2002, Calgary (Can.), May 27–29 (2002) 69–80.
12. Lindenmayer, A.: Mathematical models for cellular interactions in development, Parts I and II. J. Theor. Biol. **18** (1968) 280–315.
13. Mech, R., Prusinkiewicz, P.: Visual models of plants interacting with their environment. ACM SIGGRAPH 1996, New Orleans, ACM (1996) 397–410.
14. Prusinkiewicz, P., Hanan, J., Mech, R.: An L-system-based plant modeling language. - In: M. Nagl, A. Schürr, M. Münch (eds.): Proceedings of the International Workshop AGTIVE'99. LNCS **1779**, Springer, Berlin (2000) 395–410.
15. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer-Verlag, New York (1990).
16. Prusinkiewicz, P., Mündermann, L., Karwowski, R., Lane, B.: The use of positional information in the modeling of plants. ACM SIGGRAPH 2001, Los Angeles, ACM (2001) 289–300.
17. Rozenberg, G.: T0L systems and languages. Information and Control **23** (1973) 357–381.
18. Schürr, A.: Programmed Graph Replacement Systems. - In: G. Rozenberg (ed.): Handbook on Graph Grammars: Vol. 1, Foundations. World Scientific, Singapore (1997) 479–546.
19. Skusa, A., Banzhaf, W., Busch, J., Dittrich, P., Ziegler, J.: Künstliche Chemie. Künstliche Intelligenz **1/00** (2000) 12–19.