Genetic L-System Programming: Breeding and Evolving Artificial Flowers with *Mathematica*

C. Jacob, jacob@informatik.uni-erlangen.de,

Chair of Programming Languages, Department of Computer Science, University of Erlangen-Nuremberg, D-91058 Erlangen, Germany

Abstract

Parallel rewrite systems in the form of stringbased L-systems are used for modeling and visualizing growth processes of artificial plants. A package implementing context-sensitive (m,n)-L-systems is presented which takes full advantage of *Mathematica´*s expression manipulation and graphics capabilities. Furthermore, it is demonstrated how to use evolutionary algorithms for inferring L-systems encoding structures with some characteristic properties. We describe our *Mathematica* based genetic programming system *MathEvolvica*, present an L-system encoding via expressions, and explain how to generate, modify and breed L-systems through simulated evolution techniques.

1 Modeling growth processes by L-systems

Rewriting has proved to be a useful technique for defining complex objects by successively replacing parts of simple initial objects using a set of rewrite rules or productions. In the scope of this article we focus on a special type of character based rewrite systems, commonly termed L-systems (Lindenmayer systems), which are used in theoretical biology for describing and simulating natural growth processes [5]. All letters in a given word are replaced in parallel and simultaneously. This feature makes L-systems especially suitable for describing, e.g., fractal structures, cell divisions in multicellular organisms [1,2], or flowering stages of herbaceous plants [6], as we will demonstrate in the sequel.

D0L-systems (D0 means deterministic with no context) are the simplest type of L-systems. Formally a D0L-system can be defined as a triple

IMS´95, First Int. Mathematica Symposium, Southampton, UK, 1995, Computational Mechanics Pub.

$G_{\text{ArtFlower}} = (\Sigma, P = \{p_1, \ldots, p_9\}, \alpha)$

$\Sigma = \{f, \text{pd}, \text{pu}, \text{sprout}, \text{stalk}, \text{leaf}, \text{Leaf}, \text{bloom}\}$

α: *sprout*(4)

P: **Sprout developing leaves and flower:**

p_1: *sprout*(4) → f *stalk*(2) [pd(60) *leaf*(0)] pu(20) [pu(25) *sprout*(0)] [pd(60) *leaf*(0)] pd(20) [pu(25) *sprout*(2)] f stalk(1) *bloom*(0)

Riping sprout:

p_2: *sprout*(t < 4) → *sprout*(t+1)

Stalk elongation:

p_3: *stalk*(t > 0) → f f *stalk*(t-1)

Changing leaf sizes:

p_4: *leaf*(t) → *leaf*(t + 1.5)
p_5: *leaf*(t > 7) → *Leaf*(7)
p_6: *Leaf*(t) → *Leaf*(t - 1.5)
p_7: *Leaf*(t < 2) → *leaf*(0)

Growing bloom:

p_8: *bloom*(t) → *bloom*(t + 1)
p_9: *bloom*(7) → *bloom*(1)

Figure 1: Example of a parametrized D0L-system modelling flowering stages of an artificial plantlike structure (adapted from [6], p. 83ff.)

$G = (\Sigma, P, \alpha)$ where $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ is an alphabet, α, referred to as the axiom, is an element of Σ^*, the set of all finite words over the alphabet Σ. The structure preserving mapping $P : \Sigma \rightarrow \Sigma^*$ is defined by a set of productions or rewrite rules $\sigma \rightarrow P(\sigma)$ for each $\sigma \in \Sigma$.

Figure 1 shows a simple example L-system describing growth sequences of sprouts, leaves and blooms of an artificial flower an animation sequence of which is depicted in fig. 2. The DOL-System encodes macros for generating graphical representations of the leaves, blooms and stalks. All the non-italic terms (f, pud, pd) represent commands to move (forward, backward) and ori-

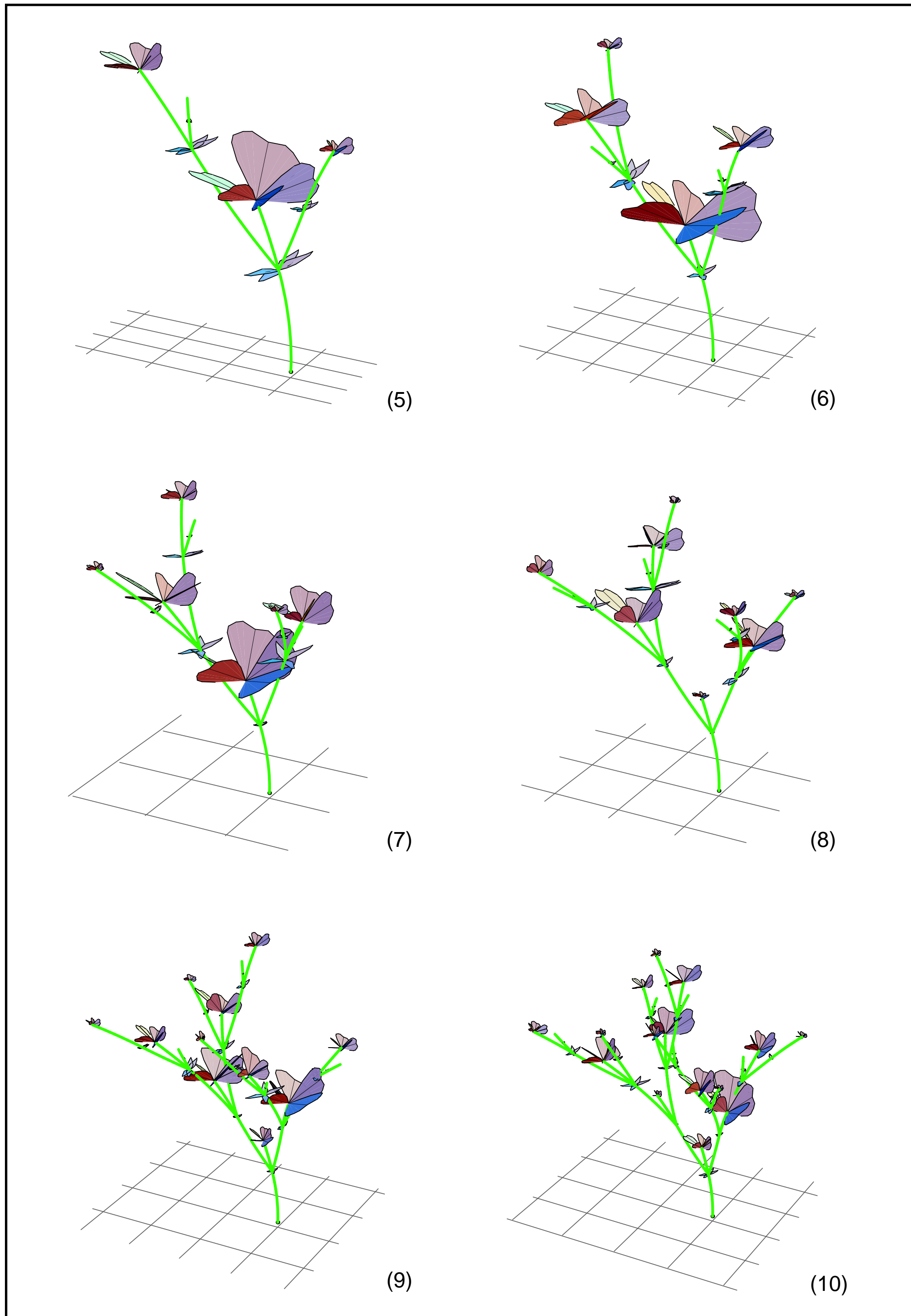

Figure 2: Visualizing growth stages of an artificial plant modelled by the D0L-system of fig. 1, the numbers in brackets mark the number of production iterations starting from the axiom

entate a drawing tool, known as a turtle, in three-dimensional space (rotate, yaw, pitch), thus translating a one-dimensional string into a 3D-object resemb–ling a plant.

2 Genetic L-system programming

Genetic Programming (GP) has been introduced as a method to automatically develop populations of computer programs through simulated evolution [3,4]. Considering L-systems as rulebased development programs it is easy to define program evolution. Each program encoded as a symbolic expression has to be interpreted and is assigned a fitness value dependent on the optimization task to be solved. On the basis of these fitnesses the individual programs struggle for „survival of the fittest“ and for the chance to become members of the next generation. In order to introduce variations into the program encoding structures genetic operators like mutation or crossover – gleaned from nature´s mutating and recombining operators on cell genomes – are applied. The evolution process develops new populations of programs from generation to generation, the interplay of modifying operators and selection hopefully lead–ing to ever better programs.

2.1 Encoding L-systems

In order to use expression evolution for L-systems a proper encoding scheme has to be defined which will be done for the more general case of IL-systems (with I referring to the number of context symbols). In context-sensitive IL-systems the rewriting of a letter depends on *m* of its left and *n* of its right neighbors, where *m* and *n* are fixed integers. These systems are denoted as (*m*,*n*)L-systems which resemble context-sensitive Chomsky grammars, but – as L-system rewriting is parallel in nature – every symbol is rewritten in each derivation step; this is especially important whenever there is an overlap of context strings. The following L-system representations are used in our IL-system package which is part of *MathEvolvica*.

Each IL-system rule has the form $l < p > r \rightarrow s$ with l, p, r and s denoting the *left context*, *predecessor*, *right context* and *successor*, respectively. The symbols "<" and ">" separate context and predecessor strings. Thus each rule can be represented by an expression of the form

```
LRule[ LEFT[ l ], PRED[ p ], RIGHT[ r ], SUCC[ s ] ].
```

Accordingly, an L-system with its axiom and rule set is encoded by an expression of the form

```
LSystem[ _AXIOM, LRULES[ __LRule] ]
```

using *Mathematica* pattern notation.

2.2 Stochastic generation of L-system codings

Normally an evolution loop starts from a population of randomly generated individual genomes – encodings of L-systems in our case. Templates serve as high-level building blocks for the expression generation routines (fig. 3). Each expression is constructed from a start pattern (here: LSys–tem[_AXIOM,_LRULES]) by recursively inserting matching expressions from the expression pool until all pattern blanks have been replaced by proper expressions. Of course, one has to take care that this construction loop eventually ends.

```
LSystem[_AXIOM,_LRULES],                                          (1)

AXIOM[sprout[4]],                                                 (1)

LRULES[
  LRule[LEFT[], PRED[ sprout[4] ], RIGHT[],
        SUCC[
          SEQ[
            SEQ[f],SEQ[stalk[2]], STACK[PD[60],leaf[0]], ...,
            __SEQ
            SEQ[f],SEQ[stalk[1]], bloom[0]]]],
  LRule[LEFT[], PRED[ sprout["t_/;t<4"] ],
        RIGHT[], SUCC[sprout[t+1]]],
  ...
  LRule[LEFT[], PRED[ bloom[6] ], RIGHT[], SUCC[ bloom[1]]],
  ___LRule
],                                                                (1)

LRule[LEFT[],_PRED,RIGHT[],_SUCC],                                (1)

PRED[sprout[aIndex]],                                             (1)

SUCC[_SEQ | _STACK],                                              (1)

SEQ[BlankSequence[_sprout | _stalk | _leaf | _bloom | _f |
                 _YL | _YR | _PU | _PD | _RL | _RR | _SEQ]],      (1)
SEQ[BlankSequence[_sprout | _stalk | _leaf | _bloom | _f |
                 _YL | _YR | _PU | _PD | _RL | _RR ]],            (4)
...

leaf[leafIndex], bloom[bloomIndex]                           (2),  (2)
```

Figure 3: Using *Mathematica´*s pattern notation for the representation of expression templates serving as building blocks to generate L-system encodings

The templates from fig. 3 basically describe the DOL-system of fig. 1. However, the additional __SEQ pattern within the first LRule expression en–ables the generation system to create variations by inserting new command sequences. Accordingly, the L-system description can be enhanced by new rules

through the ___LRule pattern. A sequence of expressions is constructed via alternative templates (SEQ[BlankSequence[...|...]]). Whenever there are several expressions matching one pattern, the expressions are selected with probabilities proportional to their weights (see the bracketed numbers right from the expressions in fig. 3).

2.3 Variations on L-system expressions

The following is a small collection of operators used for generating variations on expressions (fig. 4):

- *Mutation* replaces a randomly selected subexpression by an expression with the same head generated from the expression pool.
- *Crossover* is a recombination operator between two or more expressions. Subexpressions with the same head are selected within the expressions and interchanged.
- *Deletion* erases expression arguments whenever this is possible according to restrictions of the number of arguments for the selected expression.
- *Permutation* interchanges the sequence of arguements of an expression.

These operators are defined for general expressions and are not especially tailored to L-system encodings. Subexpressions are chosen according to operator specific selection schemes based on *Mathematica´*s pattern matching mechanisms. Thus it is, e.g., possible to restrict recombinations to SUCC term subexpressions, or permute only LRule expressions changing the ordering of rules. So this is where problem specificity can be taken into account.

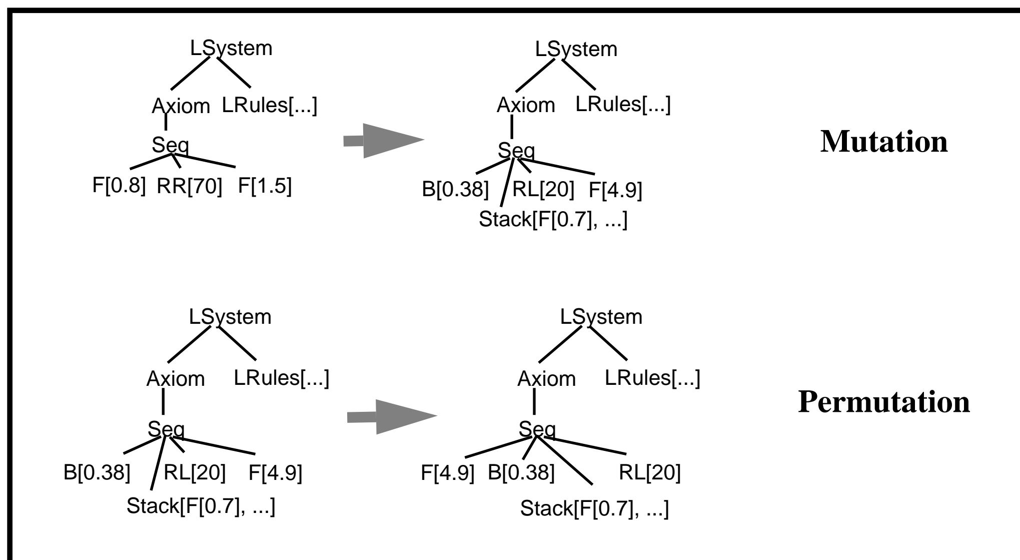

Figure 4: Examples of expression variations through genetic operators

3 Breeding artificial flowers

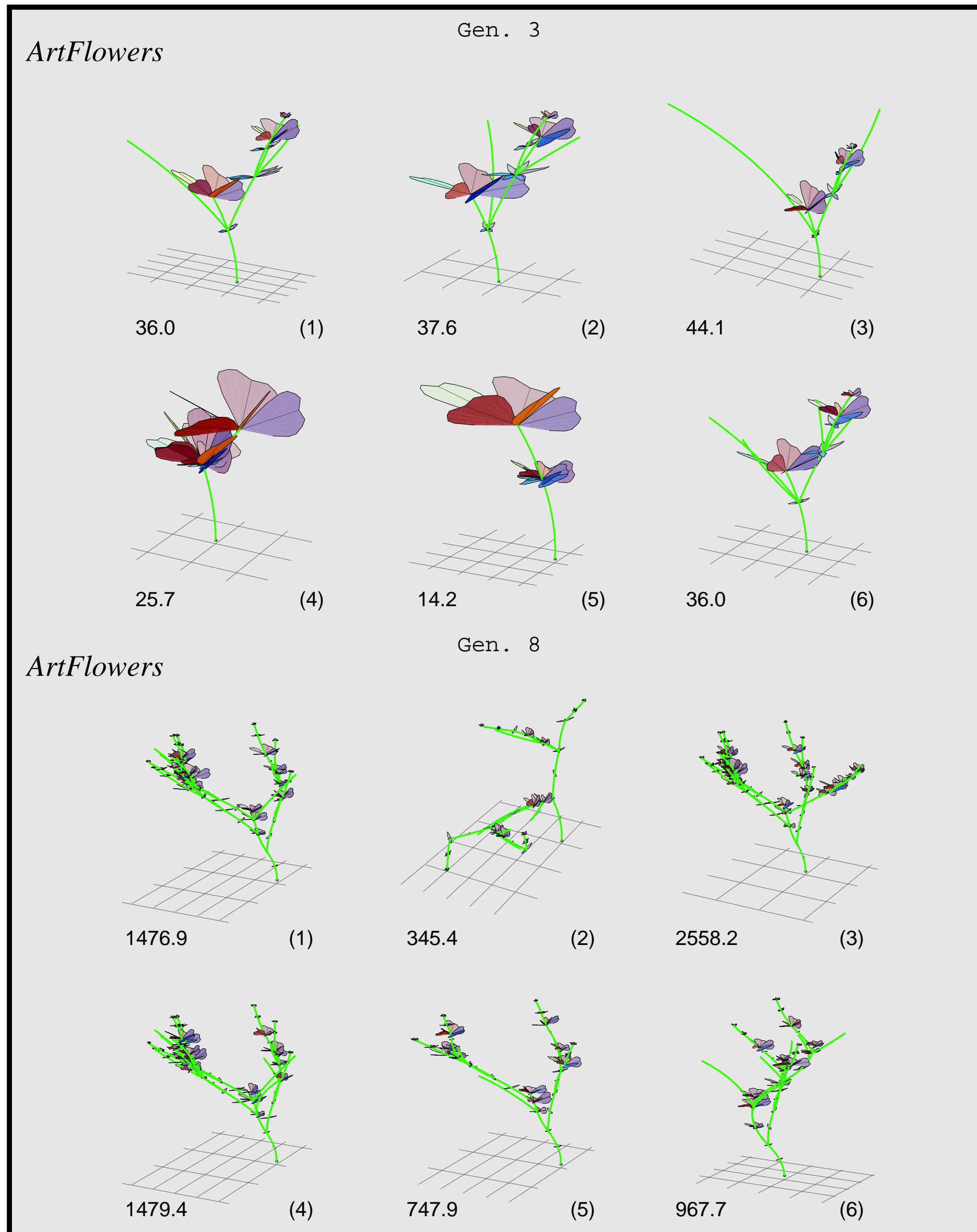

Figure 5: Generation snapshots from an evolution of L-system encoded plants

With all the ingredients described above we are now able to demonstrate how plantlike structures with specific characteristics can be developed with the help of L-system evolution. Suppose we want to breed flowers which spread out far in x-, y- and z-direction and carry as many blooms as possible. If these criteria are incorporated into the fitness evaluation function, we arrive at an

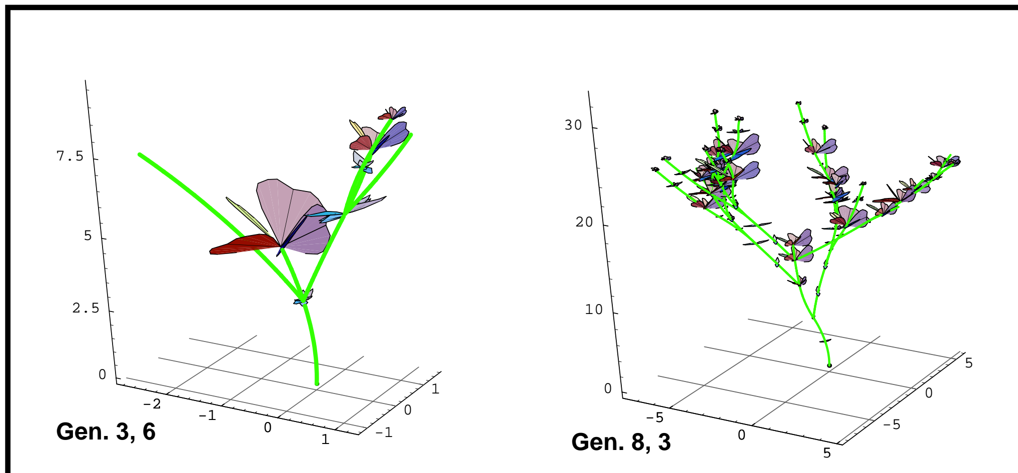

Figure 6: Comparing the best individuals from generations 3 and 8

evolution sequence as depicted in fig. 5 and fig. 6. From initially rather small individuals widespread plants evolve carrying bunches of blooms and leaves. Due to the huge amount of system memory required for generating the graphics we currently are restricted to only very small populations over 10 to 20 generations. However, we are currently implementing a package for distributed evaluation on several *Mathematica* kernels so that we should be able to tackle on more advanced breeding problems.

What makes *Mathematica* especially useful for evolutionary algorithm applications as described above are its capabilities of easy expression manipulation which are important for defining genetic operators and L-system interpretations in a comfortable and flexible way. Finally, without *Mathematica´s* graphics and animation tools we would not have enjoyed playing evolu–tion and gained so much insight into the fascinating area of L-system design.

References

1. Jacob, C., *Genetic L-System Programming*, Parallel Problem Solving from Nature - PPSN III,
   Lecture Notes in Computer Science 866, Springer, Berlin, 1994.
2. Jacob, C., *Modeling Growth with L-Systems & Mathematica*,
   to appear in: *Mathematica in Education*, TELOS Springer, 1995.
3. Koza, J., *Genetic Programming*, MIT-Press, 1993.
4. Koza, J., *Genetic Programming II*, MIT-Press, 1994.
5. Lindenmayer, A., *Mathematical models for cellular interaction in development, Parts I and II*, Journal of Theoretical Biology, 18:280-315, 1968.
6. Prusinkiewicz, P., and Lindenmayer, A., *The Algorithmic Beauty of Plants*, Springer, New York, 1990.