Rule-based modelling with the XL/GroIMP software

Ole Kniemeyer
okn@informatik.tu-cottbus.de

Brandenburgische Technische Universität Cottbus
Institut für Informatik
Lehrstuhl Praktische Informatik / Grafische Systeme

Co-Workers: Gerhard Buck-Sorlin, Winfried Kurth
Funded by Deutsche Forschungsgemeinschaft, Research Group Virtual Crops
Models of biology or ALife are often specified in a rule-based manner.

This *natural choice* of specification is lost in a conventional implementation:

A rule-based language reflects the nature of such models in a much more concise way.
Rule-based modelling: L-systems

L-systems, established in 1968, provide a rule-based formalism. They are used in biological modelling.

- Data structure: Linear *string* of symbols, e.g.,
  \[ F [ + F ] [ - F ] \]

- Turtle graphics interpretation leads to 3D-structures:

- String replacement rules implement dynamics:
  \[ A \rightarrow F [ + A ] [ - A ] \]
  They are applied in parallel.

The XL/GroIMP software – 3/22
Von Koch curve as an L-system

The von Koch (snowflake) curve arises out of the following L-system

\[
\begin{align*}
\alpha & \rightarrow F + + F + + F \\
F & \rightarrow F - F + + F - F
\end{align*}
\]

The software \textit{GroIMP} displays these derivation steps.

- The input is the L-system itself: Thus, specification and implementation are congruent.
- User interactions are possible.
L-system examples of biology

Main field of application of L-systems: Biological modelling, especially of *plant morphology*.

- Plant growth can suitably be described as a *rule-based* process.
- Realistic images can be produced.

```
* # f(-30) RL(90) f(800) RL(-90) a(10,1)
a(l,w) # D(w) F(l) [ RL(a0) b(l*r2, w*wr) ] RH(d) a(l*r1, w*wr)
b(l,w) # D(w) F(l) [ RU(a2) $ c(l*r2, w*wr) ] RH(d) c(l*r1, w*wr)
c(l,w) # D(w) F(l) [ RU(-a2) $ b(l*r2, w*wr) ] RH(d) b(l*r1, w*wr)
```

A parametric tree
Some L-system-generated images
Disadvantages of L-systems

L-systems have a number of disadvantages:

- Linear (1D string) data structure.
- Objects are simple symbols.
- Expressiveness of rules is limited.

These drawbacks become essential when modelling complex structures and interactions:

- Complex structures do not fit well into the linear world of strings.
- Complex interactions on string-encoded structures are not reasonably realizable.
Current solutions

Current solutions “outsource” the complexity:

- Prusinkiewicz introduced *open* L-systems which can be coupled to external programmes (software cpfg, L-Studio).
- Kurth introduced *sensitive* L-systems which provide a number of predefined environment functions (software GROGRA).

A better solution would be the enhancement of expressiveness of the rule-based language! This can be achieved by the transition from *strings* to *graphs*.
Graphs and graph grammars

**Graphs** are sets of nodes and connecting edges.

- This includes L-system-like strings as a subcase.
- General structures can be encoded immediately.
- Graph query languages extract information.

**Graph grammars** are rewriting systems that operate on graphs.

- Again, L-systems are a subcase.
- Complex model dynamics can be implemented in a concise way using graph transformations.
Relational Growth Grammars (RGG) are graph grammars, primarily tailored to the needs of modelling plant growth.

- Edge-labelled directed graphs build the data structure, nodes are objects of the underlying programming language.
- A conventional programming language is included.
- Graph queries can be formulated.
- Relations can be defined and used in queries.
- This general framework is also suitable for ALife.
XL: An implementation of RGG

XL is a Java-based implementation of RGG for the use in practice.

- Most constructs of Java have been integrated (classes, methods, variables, loops, ...).
- Graph transformation rules and queries can easily be formulated.
- Certain Java classes serve as turtle commands.
- All existing Java runtime libraries can be used.
- XL is integrated in the software GroIMP.
Von Koch curve as an RGG-system

The snowflake L-system as an RGG-system:

class Koch extends RGGSystem {
    void run() {
        Axiom ==> F(1) RU(120) F(1) RU(120) F(1);
        F(x) ==> F(x/3) RU(-60) F(x/3) RU(120) F(x/3) RU(-60) F(x/3);
    }
}

Specification and implementation are nearly congruent:

- An enclosing Java-like code is needed.
- Apart from that, the L-system is implemented in a 1:1 fashion.
Conway’s Game Of Life is specified as follows:

- The world is a 2D two-state (dead or alive) cellular automaton with Moore-neighbourhood.
- A living cell dies if it has less than two, or more than three living neighbours.
- A dead cell becomes alive if it has exactly three living neighbours.

The specification can be written down in three lines.
Geometrical definition of neighbourhood:

iterating Cell neighbours(Cell c1) {
    yield (* c1 --+ #c2:Cell, (c1.distanceLinf (c2) < 1.1) *);
}

Transition rules:

void transition() [ 
    x:Cell(1), (! (sum(neighbours(x).state) in {2..3})) ==>> x(0); 
    x:Cell(0), (sum(neighbours(x).state) == 3) ==>> x(1); 
]

- The original specification is reflected concisely in the implementation.
Making use of relations

Definition of neighbourhood as a relation:

```java
boolean neighbour(Cell c1, Cell c2) {
    return (c1 != c2) && (c1.distanceLinf(c2) < 1.1);
}
```

Modified transition rules:

```java
void transition() [ 
    x:Cell(1), (!sum((* x -neighbour-> #Cell *).state) in {2..3}))
        !==> x(0);
    x:Cell(0), (sum((* x -neighbour-> #Cell *).state) == 3) !==> x(1);
]
```

- Such a relational view may help to increase the readability of implementation code.
The modelling platform GroIMP

GroIMP is designed as an integrated modelling platform:

- XL grammars can be interpreted.
- Classes useful in modelling are provided: Turtle commands, further geometrical classes, cells, ...
- The outcome of a model is visualised.
- User interaction during the simulation is possible, even over a network.

GroIMP is still under development.
A simple ant model

A simplistic ant simulation is implemented easily:

- Ants live in a rectangular grid world.
- Ants release pheromone while moving.
- Released pheromone decays by and by.
- Ants remember the last twenty cells visited.
- The movement is influenced by the reachable pheromone values, a direction-preserving tendency, the memory and a random effect.
- There are food sources which stimulate the pheromone deposition of ants.
class Ant extends Cylinder {
  float dx, dy;  // current moving direction
}

class AntSimulation extends RGGSystem {
  const int memory = MIN_USER_EDGE; // a user-defined edge

  boolean placeAntAt(int i, int j) { where to place ants initially... }

  float evaluate(float pheromone, float dirdelta2) { evaluation of a possible move...
    // pheromone: Pheromone content of the reached cell,
    // dirdelta2: squared difference of the direction vectors }

  void run() {
    // move ant a to next cell, keep current cell c in memory with counter value 20
    c:Cell a:Ant ==>> n:nextCell(c, a) a -memory-> 20 c
    { a.dx := n.x - c.x; a.dy := n.y - c.y;  // update moving direction
      float p = (a.length + c.state) * C_ANT;  // amount of pheromone laid down
      a.length := c.state - p;  // update ant excitation status
      c.length := p;  // lay down pheromone;
    }
    // decrease the memory counter; if it has reached zero, the memory node is removed
    m:int ==>> if (m > 0) m(m-1);
    c:Cell ::> c.length := c.length * C_CELL;  // decay of cell pheromone
  }

  Cell nextCell(Cell c1, Ant a) {
    float dx, dy;
    // find the neighbouring cell c2 not in memory with maximum evaluate-value:
    Cell next = select((c1 -+- #c2:Cell,
      (c1.distanceLinf (c2) < 1.1),
      (!exist(* a -memory-> int c2 *)) *),
      (dx = (c2.x - c1.x) - a.dx, dy = (c2.y - c1.y) - a.dy,
      evaluate(c2.length, dx * dx + dy * dy) -> max);
    return (next != null) ? next : c1;  // all neighbours are in memory, ant doesn’t move
  }

  void init() {
    // create a 25 * 25 grid of cells, place the ants
    Axiom ==>> ` for(i = 0 .. 24) for(j = 0 .. 24) (j#Cell(i, j) if(placeAntAt(i, j)) Ant);
  }
}
Back to biology: ABC model

The ABC model predicts flower morphogenesis on the basis of a genetic regulatory network.

- Three genes, A, B and C, are used.
- Their transcription factors determine the type of flower organ to be formed.
- Factor concentrations change in time.

\[ [b] > 80, [c] > [a] \quad \rightarrow \quad \text{stamen} \]
\[ [b] > 80, [c] \leq [a] \quad \rightarrow \quad \text{petal} \]
\[ \ldots \]
The XL implementation of the ABC model benefits from the new features of RGGs:

- The regulatory network is represented as a graph.

  \[
  \text{agene:Gene(0.1) \rightarrow a:Factor(0, 0.3),}
  \text{a Activate(1e-9, 50) agene, ...}
  \]

- Its dynamics is implemented using update rules.

  \[
  \text{f:Factor(c, d) \rightarrow f.concentration := c \cdot d;}
  \text{f:Factor \leftarrow \text{g:Gene(ct)} \rightarrow f.concentration := \text{Math.max}(0,}
  \text{\sum((\text{* Factor(c2,}) \text{ Activate(s, m) g *}, m^c2 / (s+c2))) + ct));}
  \]

- Morphogenesis is modelled in an L-system style.

  \[
  \text{m:Meristem (* \text{factors} \rightarrow \text{Factor(a, Factor(b, Factor(c,)) *)} \rightarrow}
  \text{\{ int t = (b > 80) ? ((c > a) ? \text{STAMEN} : \text{PETAL}) : ...; \}}
  \text{if (t == \text{SHOOT}) (F(0.5, 0.6)) else if (t == \text{PEDICEL}) (...) ... m;}
  \]
A barley model

- Morphogenesis is modelled in an L-System style.
- Diploid genome controls ear morphogenesis.
- A metabolic network in each internode organ controls internode elongation.
Conclusion and outlook

In principle, RGGs provide a concise way of implementing biological or ALife models:

- Necessity of technical code has been reduced.
- The structural view of L-systems is preserved.
- Further models have to be checked.

For practice, runtime efficiency has to be acceptable.

- Graph grammars introduce a runtime overhead.
- Matching algorithm has to be improved, e.g., search order optimization or caching of matches.
- Java byte-code generation is desirable.