Von L-Systemen zu Graphgrammatiken: Ein neuer Ansatz für die Modellierung von Artificial Life

Arbeit im Rahmen eines Projektes der DFG-Forschergruppe *Virtual Crops*

Ole Kniemeyer

okn@informatik.tu-cottbus.de

Cottbus, 24. Juni 2003

Einige Aufgaben des Projektes

- Modellierung der morphologischen Entwicklung von Gerstenpflanzen unter verschiedenen Regimes
- Repräsentation von Genregelungsnetzwerken für die Steuerung von Morphologie und Phänologie
- Entwurf einer problemorientierten Modellierungssprache, die auch Modelle anderer Projekte der Forschergruppe abbilden kann

Von der Warte der Informatik aus betrachtet bedeutet dies zunächst:

- Abstraktion der verschiedenen Objekte zu einer einheitlichen Datenstruktur
- Abstraktion der verschiedenen, teils sehr komplexen Dynamiken zu einer einheitlichen Modellierungssprache

Dabei sind Datenstruktur und Sprache nicht unabhängig voneinander.

Artificial Life als Ideengeber

In Artificial Life sind bereits viele verschiedene Modelle behandelt worden. Hier im Seminar wurden u. a. angesprochen:

- Zelluläre Automaten
- L-Systeme
- Künstliche Chemie
- Sugarscape

Zum "Ausprobieren in der Praxis" gibt es für jeden Ansatz jeweils eine Vielzahl an spezialisierter Software.

Wünschenswert ist auch hier eine *Vereinheitlichung*, d. h. *eine* Modellierungssprache und dementsprechend *eine* Software.

Zusammen mit den konkreten Aufgaben des Projektes bilden ALife-Modelle eine gute Basis zur Abstraktion der Anforderungen und schließlich zum Entwurf einer geeigneten Sprache.

L-System-Formalismus als Modellierungssprache

Bei den seit 1968 untersuchten und in biologischen Modellen angewandten L-Systemen sind Datenstruktur und Sprache untrennbar miteinander verbunden:

- Die Datenstruktur ist eine Folge von (parametrischen) Symbolen, etwa

  ```
  F(1) [ + A(1) ] [ - A(1) ].
  ```

  Sie besitzt a priori noch keine weitergehenden Eigenschaften, diese kommen erst durch eine Interpretation der Symbole ins Spiel. Diese (meist geometrische) Interpretation ist nicht im L-System selbst spezifiziert, sondern extern in der benutzten L-System-Software.
- Die Sprache besteht aus Ersetzungsregeln, die parallel auf die Symbolfolge angewandt werden, etwa

  ```
  A(x)   ⟶   F(x/2) [ + A(x/2) ] [ - A(x/2) ].
  ```

Nachteile der L-Systeme

L-Systeme sind gut geeignet, pflanzenmorphologische Dynamiken in *Strukturmodellen* abzubilden. Darüber hinaus zeigen sich jedoch einige Nachteile:

- Die zugrunde liegenden Objekte sind einfache Symbole ohne weitere Eigenschaften. Eine Erweiterung hin zu echten *Objekten* im Sinne der objektorientierten Programmierung, d. h. zu Objekten mit Eigenschaften und Verhalten, ist wünschenswert.
- Die lineare Datenstruktur kann *Netzwerke* (etwa Genregelungsnetzwerke, Interaktionsnetzwerke mit der Umwelt, zelluläre Automaten) nicht abbilden.
- Ersetzungsregeln sind darauf angelegt, *wachsende* Strukturen zu erzeugen. Bei Netzwerken sind aber oftmals die Strukturen festgelegt, es ändern sich nur die Parameter der Objekte (z. B. Konzentrationen bei physiologischen Prozessen wie Transportvorgängen, Stoffwechsel, Zustandsänderungen eines zellulären Automaten).
- Die Einbeziehung des *Kontextes* ist bei L-Systemen nur eingeschränkt möglich. Zur Modellierung von Interaktionen benötigt man aber flexible, vielseitige Abfragemöglichkeiten des Kontextes.

Bisheriger Ausweg bei L-Systemen: Mehrere Modellebenen

Zum Ausgleich der Nachteile hat man bisher mit mehreren Modellebenen – und damit auch mehreren Sprachen – gearbeitet:

- *Global sensitive* L-Systeme erlauben die Modellierung von Interaktionen aufgrund räumlicher Nähe (z. B. Beschattung). Die Abfragefunktionen dieses "globalen Kontextes" sind fester Bestandteil der L-System-Software und somit nicht im L-System selbst definierbar.
- Die *Kopplung* eines L-Systems mit einem in einer anderen Sprache (C, Java, ...) programmierten Modul macht die Flexibilität dieser anderen Sprache nutzbar.

Diese Aufteilung in mehrere getrennte Ebenen führt aber zu einem wenig transparenten Modell. Aus Sicht des L-Systems hat das externe Modul stets "Black-Box-Charakter".

Graphgrammatiken

Die erwähnten Nachteile von L-Systemen lassen sich wesentlich eleganter beim Übergang zu *Graphgrammatiken* beheben:

- *Datenstruktur*: Aus Symbolfolgen werden echte *Graphen* im Sinne der Mathematik, d. h. *Knoten*, die über attributierte *Kanten* miteinander verbunden sein können. Die Knoten sind dabei nicht mehr nur Symbole, sondern echte Objekte der zugrunde liegenden Programmiersprache (Java).
- *Dynamik*: Aus den einfachen Ersetzungsregeln der L-Systeme werden Ersetzungsregeln von Graphen.

Graphgrammatiken sind in den letzten Jahrzehnten vielfach untersucht worden.

- Bisher geschah dies meist aus Sicht der *Theorie*.
- "Unsere" Klasse von Graphgrammatiken soll sich dagegen in der *Praxis* bewähren, nämlich bei der Formulierung von Modellen aus dem DFG-Projekt sowie aus ALife.

XL: Graphgrammatiken + Java

Die im Projekt entworfene Sprache *XL* vereint Graphgrammatiken mit wesentlichen Teilen der modernen Allzweck-Programmiersprache Java:

- Es können komplexe *Graphtransformationen* angegeben werden. Hierin sind klassische L-Systeme als Spezialfall enthalten.
- Eine leistungsfähige *Abfragesprache* ermöglicht den einfachen und gezielten Zugriff auf Objekte im Graph. So können etwa Einflüsse durch die Nachbarschaft modelliert werden.
- Als Knoten können *beliebige Java-Klassen* genommen werden, Felder und Methoden von Java-Klassen können angesprochen werden. Dies vereinfacht die Anbindung von in Java implementierten (Teil-)Modellen an XL-Modelle.
- XL enhält selbst wesentliche Konstrukte von Java.
- XL ist eingebettet in die interaktive 3D-Plattform *GroImp*, die Objekte zur *3D-Visualisierung* zur Verfügung stellt. Diese können in XL genutzt werden und spielen dort die Rolle der Turtle-Grafik klassischer L-Syteme.

Beispiel: Mutation und Rekombination

Als Beispiel für Graphtransformationen betrachten wir zwei XL-Produktionen, die die Vorgänge *Mutation* und *Crossing-Over/Rekombination* beschreiben. Das Genom besteht dabei aus einer Folge von `int`-Knoten:

```
production mutate()
{
    k:int ==> (prob(0.2)) ? irandom(-9, 9);
    // entsprechende L-System-Regel in Grogra: g(k) # g(irandom(-9, 9)) ? 0.2
}
```

```
production recombine()
{
    int j, k, l, m;
    j k, l m, (match(parent($j) -mate- parent($l)) && index($j) == index($l))
        ==>> (prob(0.3)) ? j m, l k {System.out.println ("co bei " + index($j));};
}
```

Beispiel: Einbeziehung des Genotyps beim Wachstum

Der soeben Mutation und Rekombination unterzogene Genotyp soll nun das Wachstum des Phänotyps steuern. Hierzu betrachten wir die *Biomorphe* von Richard Dawkins:

```
production grow ()
{
    b:biom(depth, dir), (match(root(b) <-encodes- g:genome) && depth > 0)
        ==> { int[][] d = {{-g[2], g[6]}, {-g[1], g[5]}, {0, g[4]}, {g[1], g[5]},
                           {g[2], g[6]}, {g[3], g[7]}, {0, g[8]}, {-g[3], g[7]}}; }
            Line(depth * d[dir][0], 0, depth * d[dir][1])
                [biom(depth-1, (dir-1) & 7)] [biom(depth-1, (dir+1) & 7)];
    // Hier wächst ein binärer Baum: biom # F [ biom ] [ biom ]
}
```

- Die im Genom g enthaltene Information beeinflusst Länge und Winkel der `Line`-Objekte.
- g kann im Graphen weit von `b` entfernt sein.

```
// Beispiel für zelluläre Automaten: Conways Game Of Life
class GameOfLife extends XLSystem
{
    iterated GameOfLifeCell context(GameOfLifeCell c1)
    {
        (* c1 -+- c2:GameOfLifeCell, (c1.basis.distanceLinf(c2.basis) < 1.1) *);
        iterated return c2;
    }

    production init()
    {
        Axiom
            ==>> ^ &(i=0..9, &(j=0..9, GameOfLifeCell(i, j, 0,
                                                      i in {2..7} && j in {2..7}
                                                      && ((i < 5) ^ (j < 5))) ));
    }

    production transition()
    {
        x:GameOfLifeCell(state), (state == 1 && !(sum(context(x).state) in {2, 3}))
            ==> x(0);

        x:GameOfLifeCell(state), (state == 0 && sum(context(x).state) == 3)
            ==> x(1);
    }
}
```

Laufende Studienarbeiten im Zusammenhang mit GroImp und XL

- Interaktiver L-System-Editor
- Import/Export von 2D- und 3D-Daten, interaktiver Textureditor
- Visualisierung metabolischer Netzwerke
- Realisierung von Sugarscape-Erweiterungen: Vergleich einer direkten Implementation mit einer XL-Implementation