Struktur- und Funktionsmodellierung: Grundlagen zur Sprache XL

Ersetzungsregel bei L-Systemen

Einfache Form

Linke Regelseite ==> rechte Regelseite; z.B. F0 ==> F0 F0; F0 wird ersetzt durch zwei F0

Komplexe Form

(* Kontext *) Linke Regelseite, (Bedingung) ==> rechte Regelseite { prozeduraler XL-Code }

(* b:Baum *), t:Trieb, (b[alter] > 10) ==>
t [RU(45) Knospe] [RU(-45) Knospe] { t[length] *= 1.1 }

Syntaxelemente

Axiom	Startobjekt
F0	Zylinder mit variabler Länge und variablem Durchmesser
F(10)	Zylinder mit Länge 10 und variablem Durchmesser
F(10, 1)	Zylinder mit Länge 10 und Durchmesser 1
F(10, 1, 14)	Zylinder mit Länge 10, Durchmesser 1 und Farbe 14 (gelb) (Bedeutung der Farbindices siehe unten)
RU(10)	Rotation um Aufwärts-Achse (up) um 10 Grad gegen den Uhrzeigersinn
RL(10)	Rotation um Links-Achse (left) um 10 Grad gegen den Uhrzeigersinn
RH(10)	Rotation um Vorwärts-Achse (head) um 10 Grad gegen d. Uhrzeigersinn
[	Verzweigungsanfang (Subgraph mit Verzweigungsrelation beginnen)
]	Verzweigungsende (Subgraph beenden, zurück zum Ausgangspunkt)
M(3)	Bewegung ohne das Zeichnen eines Elementes entlang der head-Achse. Negative Werte geben eine Rückwärtsbewegung an.
Translate(0, 0, 3)	Bewegung ohne Zeichnen eines Elementes im globalen Koordinaten-system, (x, y, z) = (0, 0, 3), relativ zur bisherigen Position.
L(10) [MRel(0.7) ...]	Symbolkombination für eine relative Bewegung entlang des Mutter-elementes. L(10) setzt die Länge für die nächsten F0-Befehle. MRel(0.7) gibt die relative Position (0.7 der aktuellen Länge, also hier: 7) entlang des Mutterelementes an, ausgehend von dessen Basis (unten), von wo das nächste F0 konstruiert wird.

Parameter

A(x) ==> A(x+1) Rechnen mit Parameter des Objektes vom Typ A (welches ersetzt wird)

a:A ==> a { a[carbonpool] += 2.5 } Rechnen mit Eigenschaft des Objektes a vom Typ A, dieses Objekt bleibt ansonsten unverändert

Moduldefinition

Alle verwendeten Module (Objekttypen) müssen definiert sein. Vordefiniert sind Basismodule in GroIMP wie F0, F(), M(), RU() etc. Eigene Module können in der RGG-Datei selbst definiert werden. Mit der module-Definition wird eine neue Objektklasse erzeugt. Eine Objektklasse kann als Bauplan für Objekte verstanden werden, in der Eigenschaftenfelder und Methoden (das sind spezielle Fähigkeiten bzw. Funktionen) definiert sind. Jedes erzeugte Objekt ist eine Instanz einer Klasse (eines Modultyps) und wird nach dem entsprechenden Bauplan angelegt. Diesen Vorgang nennt man "Konstruktion" oder "Instanzierung". Die Eigenschaftsfelder sind durch den Bauplan vorgegeben; die entsprechenden Werte werden bei der Konstruktion vorgegeben, können im Laufe der Berechnung aber geändert werden. So hat die Klasse Mensch z.B. die Eigenschaft „Körpergröße“, die Werte sind aber für jede Instanz der Klasse – d.h. für jeden Menschen – unterschiedlich. Werden bestimmte Eigenschaften bestimmt, die bei der Konstruktion angegeben werden müssen, so nennt man diese "Aufrufparameter". **Selbst definierte Module können wie folgt angegeben werden:**

module Trieb extends Null;

Einfaches, nicht gezeichnetes Objekt ohne Aufrufparameter.

module Trieb (int alter) extends Null;

Einfaches, nicht gezeichnetes Objekt mit einem Aufrufparameter. D.h., beim Anlegen eines Objektes dieses Typs muss eine Ganzzahl mit angegeben werden, z.B. Trieb(20).

module Trieb (int alter) extends F(10);

Einfaches, gezeichnetes Objekt auf Basis des Standardmoduls F(). Ein Objekt Trieb erbt alle Eigenschaften und Methoden, die im Bauplan des Moduls F() definiert sind. Dies ist z.B. die Eigenschaft „length“. Aufrufparameter für Trieb siehe oben. Der Aufrufparameter für F(10) ist konstant auf 10 gesetzt, d.h. alle Trieb-Objekte werden mit der Anfangslänge 10 erzeugt. Die Objekteigenschaft „length“ hat also den Wert 10. "alter" hat nichts mit der Länge zu tun.

module Trieb (int alter, float laenge) extends F(laenge);

Die Definition des Trieb-Objektes wird hier um einen weiteren Aufrufparameter (float laenge) erweitert. Dieser Parameter wird beim Anlegen des Trieb-Objektes an F(laenge) weitergegeben. D.h., das Objekt wird mit einer entsprechenden Anfangslänge erzeugt. Das Objekt hat also sowohl eine selbstdefinierte Eigenschaft „laenge“, als auch die von F geerbte Eigenschaft „length“. Diese sind beim Erzeugen (der Konstruktion) des Objektes gleich. Wird eine der Eigenschaften im Laufe der Modellausführung geändert, bleibt die andere Eigenschaft davon aber unberührt. Nur eine Änderung von „length“ führt zu einer sichtbaren Veränderung des Objektes.

module Trieb (int alter, super.length) extends F(length);

Diese Definition des Trieb-Objektes verknüpft den zweiten Aufrufparameter direkt mit der geerbten Eigenschaft „length“ der F-Klasse. „super.“ ist ein Verweis auf den Bauplan der Klasse, die mit "extends" angegeben wird.

Ansprache:

r.alter oder r[alter] Ansprache einer Eigenschaft des Objektes r
(bei der 2. Variante erfolgt bei Änderung automatisches Neuzeichnen)

r.setTransform(0, 0, 1) Ansprache einer Methode setTransform(x, y, z) des Objektes r

Erinnerung:

r:Trieb, (r[alter] > 10) ==> ... sucht nach allen Instanzen der Klasse Trieb. r ist ein Bezeichner (Name) für die jeweilige Instanz. Er dient nur der Ansprache (um z.B. die Bedingung formulieren zu können).

Regeltypen

Es gibt mehrere Regeltypen.

==> **Regel im Lindenmayer-Stil.** Einfache Ersetzungsregel, bei der ein Teilgraph (im Allgemeinen nur ein Objekt) durch einen anderen Graphen ersetzt wird. Die Relationen werden hierbei wieder hergestellt. D.h.: Wird in einem eindimensionalen Graphen [A B C D] der Knoten B durch einen Knoten G ersetzt, so wird G mit den ursprünglichen Relationen von B in den Graphen eingesetzt [A G C D]. Das Leerzeichen gibt in XL stets eine Nachfolgekante an.

```
Axiom ==> A B C D;
B ==> G;
```

==>> **Regel im SPO-Stil.** Komplexe Ersetzungsregel, bei der ein Teilgraph durch einen anderen Graphen ersetzt wird. Die Relationen müssen hierbei vom Programmierer selbst wieder hergestellt werden. D.h.: wird in einem eindimensionalen Graphen [A B C D] der Knoten B durch einen Knoten G ersetzt, so werden die ursprünglichen Relationen von B nicht im Graphen beibehalten. Wird die Relation zum Element C im Beispiel nicht explizit angegeben, so existiert anschließend keine Verbindung der Knoten C D zum Graphen, mit der Folge, dass die Knoten nicht mehr sichtbar sind, also praktisch gelöscht werden. Es bleiben also dann nur die (unverbundenen) Knoten A und G.

```
Axiom ==> A B C D;
B ==>> G;
```

Um dasselbe Ergebnis zu erhalten wie im oberen Beispiel (Lindemayer-Stil), müssen die Relationen zu A und C explizit auf der rechten Regelseite aufgeführt werden:

```
Axiom ==> A  B  C  D;
a:A B c:C  ==>>  a G c;
```

Dieser Regeltyp wird häufig genutzt, um Subgraphen zu löschen.

```
Axiom ==>  A A [ B A A A ] A A [ C A A ] A

B ==>> ;
```

Der resultierende Graph lautet “A A A A [C A A] A”.

::> **Aktualisierungsregel.** Dieser Regeltyp verändert die Struktur des Graphen nicht. Er wird genutzt, um die Eigenschaften der Objekte (Knoten) im Graphen zu verändern.

```
c:C  ::>  { c[length] = c[length] * 20; }
```

Anhang:

Farbtabelle für den F-Befehl:

0	black
1	blue
2	green
3	cyan
4	red
5	magenta
6	brown
7	light grey
8	dark grey
9	light blue
10	light green
11	light cyan
12	light red
13	light magenta
14	yellow
15	white