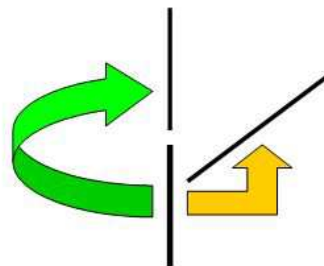
Relationale Wachstumsgrammatiken

Der Schritt zu relationalen Wachstumsgrammatiken

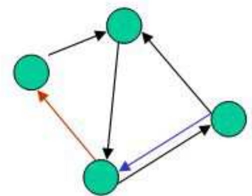
Nachteil von L-Systemen:

- in L-Systemen mit Verzweigungen (über Turtle-Kommandos) nur 2 mögliche Relationen zwischen Objekten: "direkter Nachfolger" und "Verzweigung"



Erweiterungen:

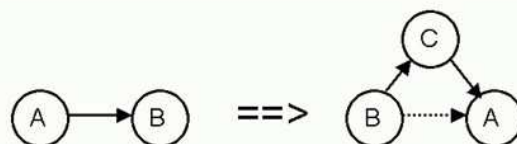
- Zulassen weiterer Relationstypen (beliebig wählbar)
- Zulassen von Zyklen   (→ Graph-Grammatik)



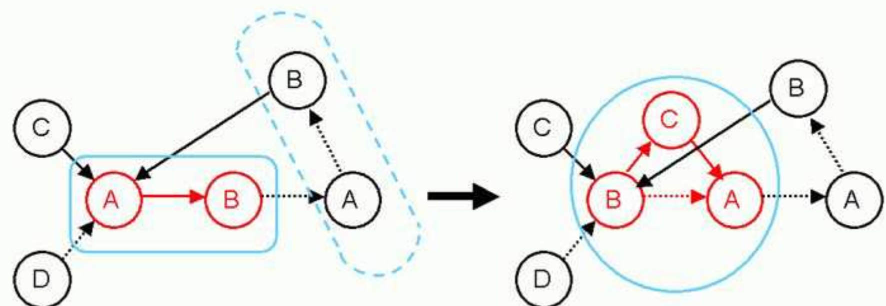
ebenfalls regelbasierter Mechanismus:

Graph-Grammatiken

Regel:

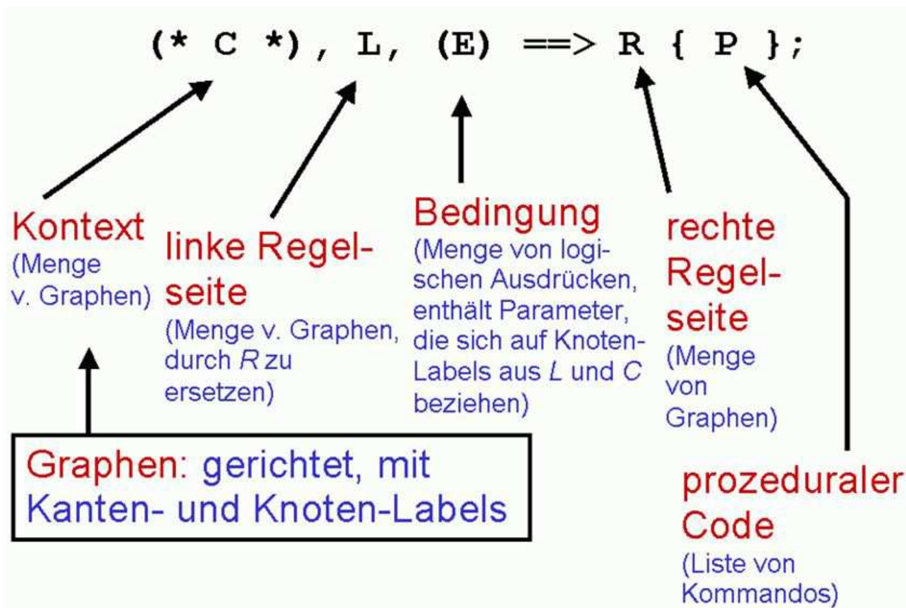


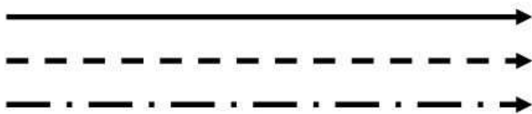
Anwendung:



RELATIONALE WACHSTUMSGRAMMATIKEN
(RGG: Relational Growth Grammars, parallele Graph-Gramm.)

Aufbau einer Regel einer RGG:





Kanten-Markierungen repräsentieren verschiedene Arten von Relationen:

- ist Nachbar von
- enthält
- trägt
- codiert (genetisch)
- ist gepaart mit
- (...)

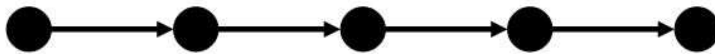
→auch möglich: Darstellung von multiskalierten Strukturen

Standard-Kantentypen:

successor (> oder *blank*), branch (+> oder erste Kante bei Klammern [...]), refinement (/>)

RGG als Verallgemeinerungen von L-Systemen:

Zeichenketten entsprechen speziellen Graphen



In Textform schreiben wir allgemeine (selbstdefinierte) Kanten als `-kantensorte->`

Kanten des speziellen Typs "Nachfolger" werden meist als Leerzeichen geschrieben (statt `-successor->`)

Sonderformen von RGG-Regeln:

Aktualisierungsregeln (Regelpfeil `::>`): es werden nur Parameter verändert

Beispiel: `s:Sphere ::> s[radius] += increment;`

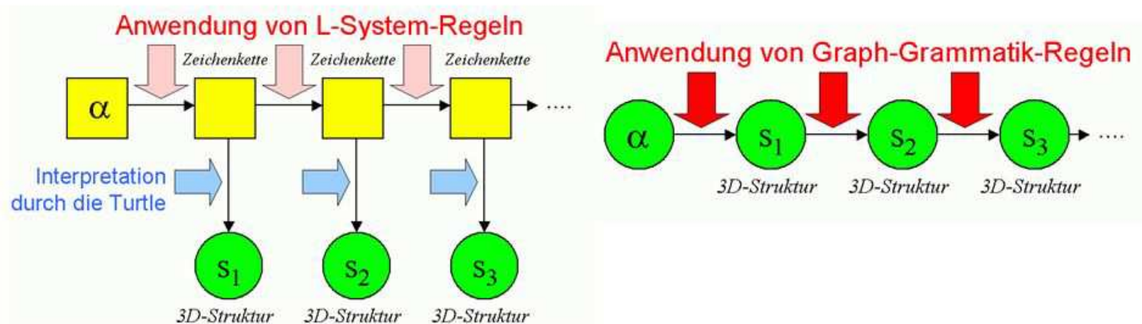
Instanzierungsregeln: einzelne Zeichen werden in Substrukturen aufgelöst, ohne Einfluss auf den nächsten Entwicklungsschritt
(Regel muss dann direkt in der Moduldeklaration stehen)

Beispiel:

```
module S(float value) extends Null ==>
   { float x = value;}
   Sphere(0.1).(setShader(new RGBAShader(1,1-x,1-x)));
```

Charakteristika von RGGs:

• Grammatik modifiziert direkt den Graphen, Umweg über String-Codierung entfällt (bzw. wird nur noch für Regel-Input gebraucht)



außerdem Nachteil der Turtle-Interpretation von L-Systemen: Segmente sind nur Zylinder, keine Objekte im Sinne der OOP

→ Erweiterungen:

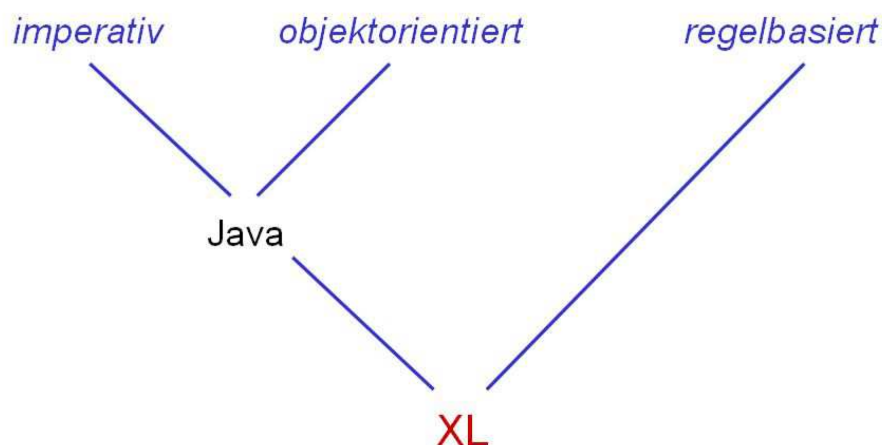
• Knoten des Graphen können beliebige Objekte sein (auch Grafikobjekte)

• Einbettung von Code einer höheren, imperativen oder objektorientierten Programmiersprache in die Regeln (für uns: Java)

Programmiersprache zur Implementation von RGG: XL

„eXtended L-system language“

Programmiersprache, die parallele Graph-Grammatiken (RGG) einfach verfügbar macht



Die Sprache XL

Sprachspezifikation: Kniemeyer (2007/08)

Ole Kniemeyer: Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling. Dissertation, BTU Cottbus 2008.
http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:kobv:co1-opus-5937

- Erweiterung von Java
- erlaubt zugleich Spezifikation von L-Systemen und RGG in intuitiv verständlicher Regelschreibweise

prozedurale Blöcke, ähnlich Java: { ... }

regelorientierte Blöcke (RGG-Teil): [...]

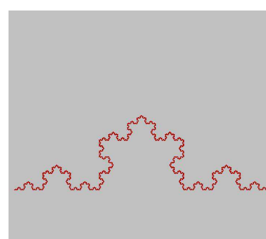
- Knoten der Graphen sind Java-Objekte, auch Geometrie-Objekte

Beispiel: XL-Programm für die Koch'sche Kurve

```
public void derivation()
  [
  Axiom ==> RU(90) F(10);
  F(x) ==> F(x/3) RU(-60) F(x/3) RU(120) F(x/3) RU(-60) F(x/3);
  ]
```

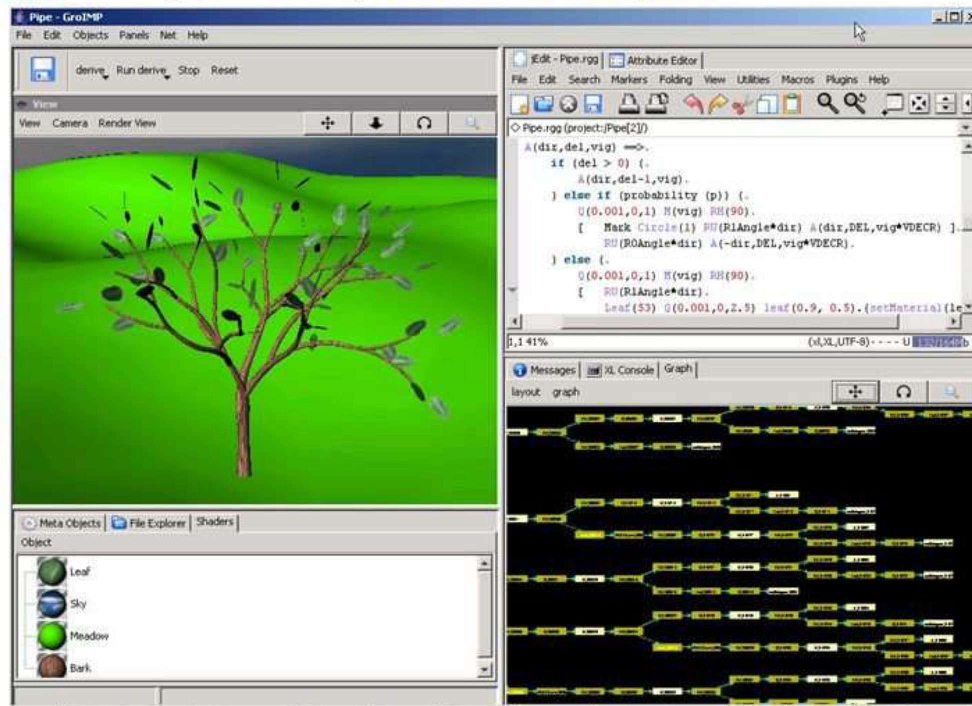
Knoten des Graphen

Kanten (Typ „Nachfolger")



zum Arbeiten mit XL:

Interaktive 3D-Plattform ***GroIMP*** (Growth-grammar related Interactive Modelling Platform) mit XL-Compiler



- GroIMP ist ein Open Source-Projekt

http://www.grogra.de

http://sourceforge.net/projects/groimp/

siehe auch das Wiki:
http://sourceforge.net/p/groimp/wiki/Home/

Anwendungsbeispiel: Modellierung von Parklandschaften

(Rogge & Moschner 2007, für Stiftung Branitzer Park, Cottbus)

mit GroIMP generierte Erle in VRML-Welt