4. Regelbasierte Systeme

regelbasiertes Programmierparadigma:

Computer = Transformationsmaschine für Strukturen

Es gibt eine aktuelle Struktur, die solange transformiert wird, wie dies möglich ist.

Arbeitsprozess: Such- und Anwendungsprozess.
matching: Suchen einer passenden Regel,
rewriting: Anwendung der Regel, um die Struktur umzuschreiben.

Programm = Menge von Transformationsregeln
Programmfindung = Spezifikation der Regeln

Transformation von Strukturen:
- Aufbau komplexer Strukturen (z.B. Modelle für natürliche Objekte)
- Vereinfachung (z.B. Theorembeweis); *Reduktion* auf Normalform

Reduktionssysteme

Definition

Ein **Reduktionssystem** $(D, \succ)$ besteht aus einer nichtleeren Menge D und einer beliebigen, binären Relation $\succ$ auf D.

Wir benutzen die folgenden Bezeichnungen:

- $\rightarrow$ die reflexive, transitive Hülle von $\succ$
- $\overset{+}{\rightarrow}$ die transitive Hülle von $\succ$
- $\leftrightarrow$ die reflexive, transitive, symmetrische Hülle von $\succ$

Definition

1. Ein Reduktionssystem $(D, \succ)$ heißt **konfluent**, wenn für jedes Tripel $s, s_1, s_2 \in D$ mit $s \to s_1$, $s \to s_2$ ein $t \in D$ existiert mit $s_1 \to t$ und $s_2 \to t$.
2. $(D, \succ)$ heißt **lokal konfluent**, wenn für alle $s, s_1, s_2 \in D$ mit $s \succ s_1$, $s \succ s_2$ ein $t \in D$ mit $s_1 \to t$ und $s_2 \to t$ existiert.
3. $(D, \succ)$ heißt **noethersch** (oder **wohlfundiert** oder **terminierend**), wenn es keine unendlichen Folge $s_0 \succ s_1 \ldots \succ s_i \succ \ldots$ gibt.
4. Ein konfluentes und noethersches Reduktionssystem heißt **kanonisch**.
5. Ein Element $s \in D$ heißt **irreduzibel** (oder eine **Normalform**) in $(D, \succ)$, wenn kein $t \in D$ existiert mit $s \succ t$.
6. Sei $s \in D$. Ein Element $s_0 \in D$ heißt eine **Normalform für s** in $(D, \succ)$, wenn s_0 irreduzibel ist und $s \to s_0$ gilt.

Satz:

Sei $(D, \succ)$ ein kanonisches Reduktionssystem. Dann gilt:

1. *Zu jedem $s \in D$ gibt es eine eindeutige Normalform. Diese bezeichnen wir mit $irr(s)$.*
2. *Für $s, t \in D$ gilt*

$$s \leftrightarrow t \text{ gdw } irr(s) = irr(t)$$

3. *$(D, \succ)$ sei berechenbar im folgenden Sinne: Es gibt einen Algorithmus, der zu jedem $t \in D$ ein t' mit $t \succ t'$ liefert, wenn ein solches existiert, und andernfalls ausgibt „t ist irreduzibel“, Dann ist die Relation $\leftrightarrow$ entscheidbar.*

Beweis zu (1):

Angenommen es gäbe für $s \in D$ zwei Normalformen s_1, s_2.
D.h. es gilt $s \to s_1$ und $s \to s_2$.
Wegen der Konfluenz von $(D, \succ)$ gibt es $t \in D$ mit
$s_1 \to t$ und $s_2 \to t$.
Das widerspricht der Irreduzibilität von s_1, s_2.
Existenz einer Normalform für $s \in D$,
Setze $s_0 = s$ und wählen ein s_{i+1} mit $s_i \succ s_{i+1}$, solange s_i nicht irreduzibel ist.
Da $(D, \succ)$ noethersch ist, wird nach endlich vielen Schritten ein irreduzibles s_i erreicht.

Beweis zu (2):

Die Implikation von rechts nach links ist trivial.
Gelte jetzt $s \leftrightarrow t$.
Nach Definition von $\leftrightarrow$ gibt es eine Folge $s = s_0, s_1, \ldots, s_n = t$, so daß für alle $0 \leq i < n$ entweder $s_i \succ s_{i+1}$ oder $s_{i+1} \succ s_i$ gilt.
Der Nachweis von $irr(s) = irr(t)$ geschieht durch Induktion über n.
Der Induktionsanfang $n = 0$, d.h. $s = t$ ist trivial.
Sei also die Behauptung für Folgen der Länge $n - 1$ schon bewiesen. Also gilt $irr(s_1) = irr(t)$.
Im Fall $s_0 \succ s_1$ gilt offensichtlich $irr(s_0) = irr(s_1)$, und wir sind fertig.
Falls $s_1 \succ s_0$ gilt, folgt aus der Konfluenz, daß ebenfalls $irr(s_0) = irr(s_1)$ gelten muß.

Beweis zu (3):

Zu gegebenem s, t wird wie folgt entschieden, ob $s \leftrightarrow t$.
Beginnend mit $s_0 := s$, liefert der vorausgesetzte Algorithmus Elemente s_i mit $s_0 \succ s_1 \succ s_2 \succ \ldots$, bis hierbei ein irreduzibles s_m erreicht ist.
Da $(D, \succ)$ noethersch ist, tritt das auf jeden Fall ein und wird durch „s_m ist irreduzibel“ mitgeteilt, ferner gilt $s_m = irr(s)$.
Entsprechend erhält man $irr(t)$ aus t.
Nach (2) ist $s \leftrightarrow t$ genau dann, wenn $irr(s) = irr(t)$.

Beweisverfahren der "noetherschen Induktion":

Satz

Für ein noethersches Reduktionssystem $(D, \succ)$ gilt das folgende Beweisprinzip der Noetherschen Induktion:
Es sei $X \subseteq D$, so daß für alle $a \in D$ gilt

$$\{b \mid a \succ b\} \subseteq X \Rightarrow a \in X.$$

Dann ist $X = D$.

Beweis:

Angenommen es gibt $a_0 \in D \backslash X$. Nach Annahme über X gilt $\{b \mid a_0 \succ b\} \not\subseteq X$.
Es gibt also ein a_1 mit

$$a_0 \succ a_1, a_1 \notin X$$

Nach Annahme über X gilt wieder $\{b \mid a_1 \succ b\} \not\subseteq X$ und es gibt ein a_2 mit

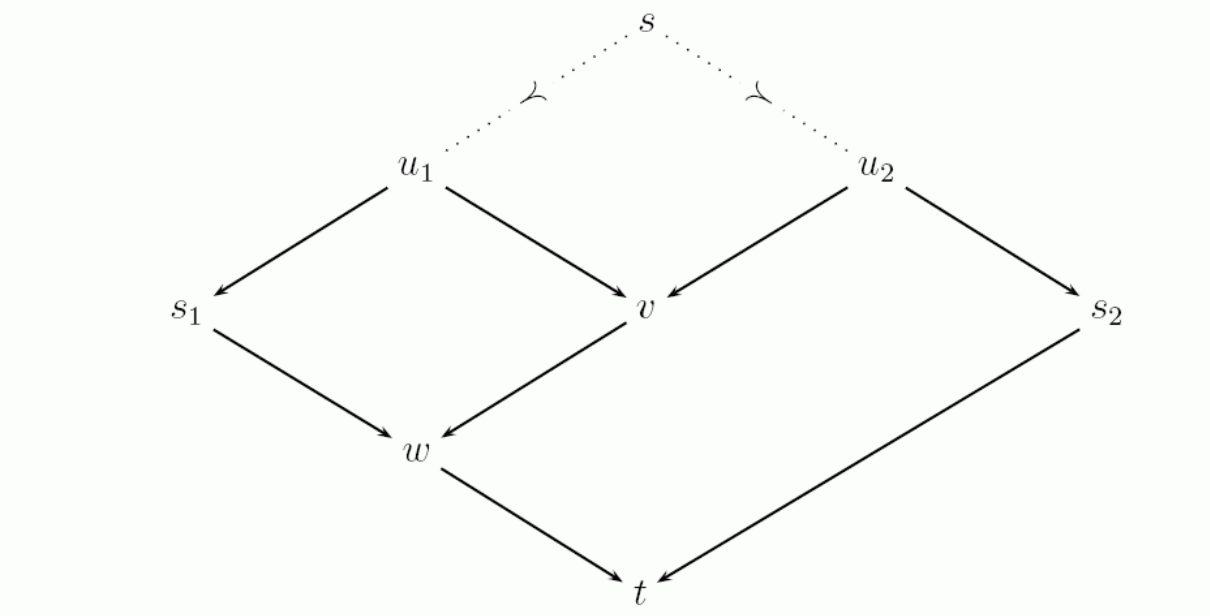
$$a_0 \succ a_1 \succ a_2, a_2 \notin X$$

Fährt man in dieser Weise fort, so erhält man eine unendliche Folge $(a_i)_{i \in N}$ mit $a_i \succ a_{i+1}$ für alle i. Das ist ein Widerspruch, denn $(D, \succ)$ war als noethersch vorausgesetzt. □

Schluss von lokaler Konfluenz auf Konfluenz:

Satz

Wenn $(D, \succ)$ ein noethersches und lokal konfluentes Reduktionssystem ist, dann ist $(D, \succ)$ konfluent, d. h. kanonisch.



Beispiele für Reduktionssysteme:

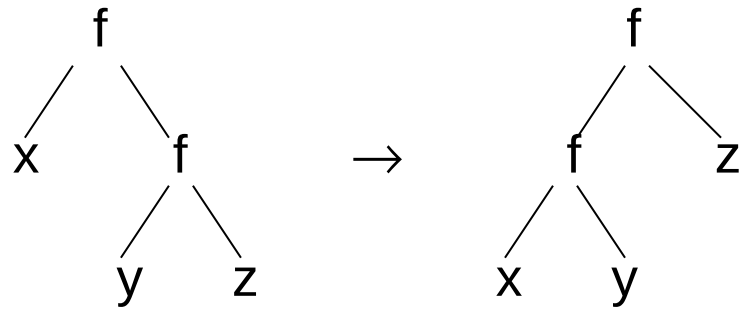
Termersetzungssysteme

Sei Σ eine Signatur wie in der Prädikatenlogik (vgl. Kap. 1), $Term_\Sigma$ die Menge der Terme über Σ und $Ground_\Sigma$ die Menge der Grundterme (d.h. ohne Variablen) über Σ.

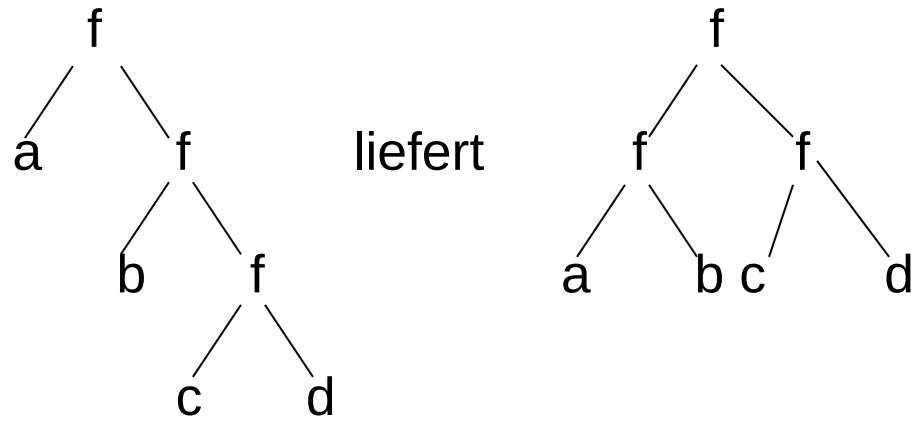
Ein *Termersetzungssystem* über Σ ist eine Teilmenge von $Term_\Sigma \times Term_\Sigma$. Die einzelnen Elemente (die *Regeln*) werden in der Form $T_1 \rightarrow T_2$ geschrieben. Die *Anwendung* einer solchen Regel (Reduktionsrelation) auf einen Grundterm besteht

aus der Ersetzung eines Subterms, der bei passender Substitution σ mit T_1 matcht, durch $\sigma(T_2)$.
Beispiel:

Regel



Anwendung auf



Spezialfall: ***Stringersetzungssysteme***
(auch: Semi-Thue-Systeme)

Die Grundterme sind hier die endlichen Wörter über einem Alphabet Σ.
Die Regeln sind von der Form $\alpha \rightarrow \beta$, wobei $\alpha, \beta \in \Sigma^*$.
Eine *Anwendung* einer Regel (Reduktionsschritt) besteht aus der Ersetzung eines Teilwortes α durch die zugehörige rechte Regelseite, β.

Beispiel:
Σ = { A; B }, einzige Regel BA $\rightarrow$ AABB.
Mit Startwort BAA erhält man sukzessive:

BAA
AABBA
AABAABB
AAAABBABB
AAAABAABBBB
...

(matchende linke Regelseite jeweils rot markiert)

$\Rightarrow$ dieses Stringersetzungssystem ist offensichtlich nicht terminierend.

Anderes Beispiel:
Σ = { A; B }, einzige Regel BBA $\rightarrow$ AAAB.

Dieses System ist terminierend, da die Anzahl der B bei jedem Reduktionsschritt abnimmt (und nicht kleiner als 0 werden kann).

Es ist bisher nicht bekannt, ob die Termination von String-ersetzungssystemen mit nur einer Regel generell entscheidbar ist.

Implementation eines Stringersetzungssystems in der Sprache XL (Beispiel):

```
module A extends F(10, 1)
   {{ setShader(RED);
      setTransform(0, 0, 0.5); }};

module B extends F(10, 1)
   {{ setShader(GREEN);
      setTransform(0, 0, 0.5); }};

protected void init()
   [
   Axiom ==> RU(90) B A A;
   ]

public void apply()
{
setDerivationMode(SEQUENTIAL_MODE);
   [
   B A ==> A A B B;
   ]
}
```

Formale Grammatiken

Eine *Chomsky-Grammatik* (Typ-0-Grammatik) ist ein Stringersetzungssystem, für das im Alphabet eine Teilmenge *N* von Nichtterminalzeichen ausgezeichnet ist und bei der in jeder linken Regelseite mindestens 1 Nichtterminalzeichen auftritt.

Eine Typ-1-Grammatik (auch: kontextsensitive Grammatik) ist eine Typ-0-Grammatik, in der alle Regeln die Form $\alpha A \beta \rightarrow \alpha \gamma \beta$ (mit $A \in N$; α, β, $\gamma \in \Sigma^*$) haben.

Eine Typ-2-Grammatik (auch: kontextfreie Grammatik) ist eine Typ-0-Grammatik, in der die linke Seite bei allen Regeln nur aus genau einem Nichtterminalzeichen besteht.

Eine Typ-3-Grammatik (auch: reguläre Grammatik) ist eine Typ-2-Grammatik, bei der auf der rechten Regelseite höchstens ein Terminalzeichen auftreten darf und – wenn dies der Fall ist – allenfalls noch *ein* Nichtterminalzeichen. Die erlaubte Stellung solcher Nichtterminalzeichen muss außerdem über alle Regeln hinweg einheitlich *immer vor* oder *immer hinter* dem Terminalzeichen sein.

Als Bestandteil einer formalen Grammatik zählt außerdem ein Startzeichen (auch *Axiom* genannt) aus der Menge *N* der Nichtterminalzeichen.

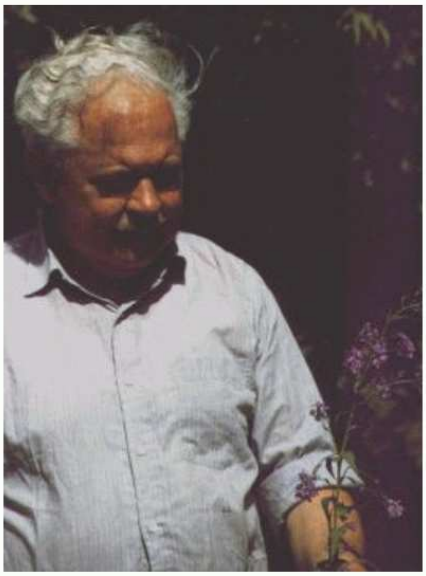
Die von einer Grammatik erzeugte Sprache ist die Menge aller Wörter, die aus dem Startzeichen durch sequenzielle Anwendung der Regeln in endlich vielen Schritten abgeleitet werden können und die nur aus Terminalzeichen bestehen.

Chomsky-Hierarchie:
Typ-0-Sprachen ⊃ kontextsensitive Sprachen ⊃ kontextfreie Sprachen ⊃ reguläre Sprachen.

"Quer" zur Chomsky-Hierarchie liegen die Sprachen, die man durch Lindenmayer-Systeme erhalten kann: Verwendung von paralleler statt sequenzieller Ersetzung.

L-Systeme (Lindenmayer-Systeme)

Modellierung von Pflanzenarchitekturen
als Regelanwendungen: L-Systeme



Aristid Lindenmayer
(1925 – 1989)

Definition:

Ein L-System ist ein Tripel (Σ, α, R); darin ist:

Σ eine Menge von Zeichen, das *Alphabet*,

α eine Zeichenkette mit Zeichen aus Σ, das *Startwort* (auch "Axiom"),

R eine Menge von Regeln der Form

Zeichen → Zeichenkette;

darin sind das Zeichen auf der linken Regelseite und die Zeichenkette aus Σ entnommen.

Ein *Ableitungsschritt* (rewriting) einer Zeichenkette besteht aus der Ersetzung aller ihrer Zeichen, die in linken Regelseiten vorkommen, durch die entsprechenden rechten Regelseiten.

Man vereinbart: Zeichen, auf die keine Regeln anwendbar sind, werden unverändert übernommen.

Ergebnis:

Ableitungskette von Zeichenketten, die sich durch wiederholte Anwendung des *rewriting*-Vorgangs aus dem Startwort ergeben.

$$\alpha \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \ldots.$$

Beispiel:

Alphabet {A, B}, Startwort A

Regelmenge:

A → B
B → AB

Ableitungskette:

A → B → AB → BAB → ABBAB → BABABBAB
→ ABBABBABABBAB → BABABBABABBABBABABBAB
→ ...

wie lang ist die n-te Zeichenkette in dieser Ableitung?

was für die Modellierung von grafischen und biologischen Strukturen noch fehlt:

eine *geometrische Interpretation*

Füge also hinzu:

eine Abbildung, die jeder Zeichenkette
eine Teilmenge des 3-dimensionalen Raumes zuordnet

dann: "interpretierte" L-System-Abarbeitung

$$\begin{array}{ccccccc} \alpha & \rightarrow & \sigma_1 & \rightarrow & \sigma_2 & \rightarrow & \sigma_3 \rightarrow \ldots. \\ & & \downarrow & & \downarrow & & \downarrow \\ & & S_1 & & S_2 & & S_3 \quad \ldots. \end{array}$$

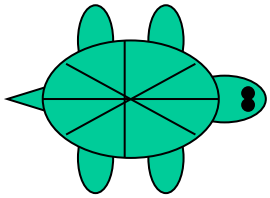
S_1, S_2, S_3, ... können als Entwicklungsstufen eines Objekts, einer Szene oder eines Organismus interpretiert werden.

Für die Interpretation der Zeichenketten:

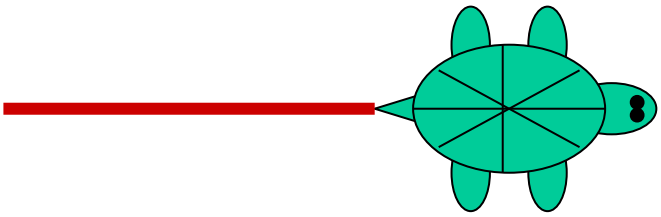
Turtle-Geometrie

- Beschreibung statischer, stückweise linearer geometrischer Strukturen
- imperative Sprache zur Steuerung eines Zeichengeräts
- relative Längen- und Richtungsinformationen

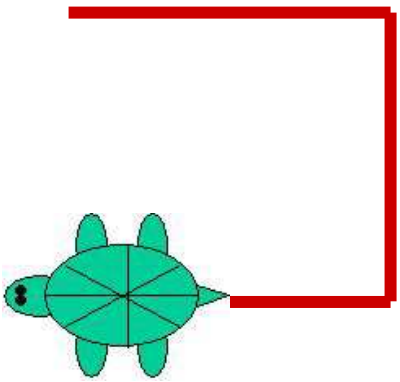
Turtle:
zeichnende Schildkröte, die auf Befehle hört



F0



F0 RU(90) F0 RU(90) LMul(0.5) F0



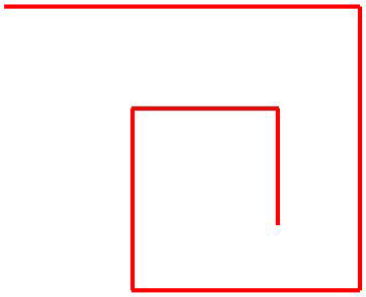
Wiederholung von Abschnitten der Zeichenkette möglich mit "**for**"

z.B. **for ((1:3)) (A B C)**
liefert **A B C A B C A B C**

was ist das Ergebnis der Interpretation von

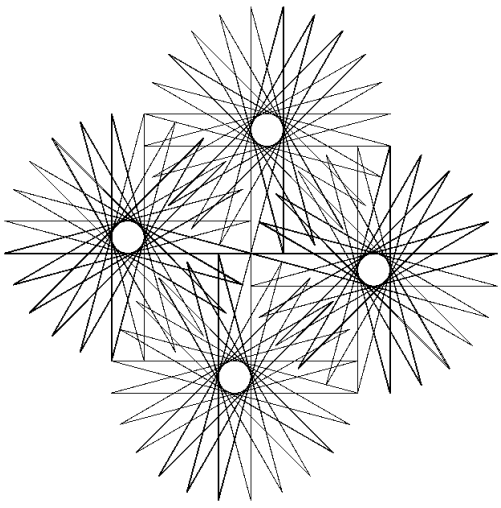
L(10) for ((1:6))
(F0 RU(90) LMul(0.8)) *?*

```
L(10) for ((1:6))
        ( F0 RU(90) LMul(0.8) )
```



anderes Beispiel:

```
for ((1:20))( for ((1:36))
                ( F0 RU(165) F0 RU(165) ) RU(270) )
```



Turtle geometry ("Schildkrötengeometrie")

befehlsgesteuertes, lokales Navigieren im 2D- oder 3D-Raum (Abelson & diSessa 1982; vgl. Programmier-sprache "LOGO")

"Turtle": Zeichen- oder Konstruktionsgerät (virtuell)

- speichert (grafische und nicht-grafische) Informationen

- mit einem Zustandsspeicher assoziiert (wichtig für Verzweigungen)

- aktueller Zustand der Turtle enthält z.B. Information über aktuelle Liniendicke, Schrittweite, Farbe, weitere Eigenschaften des als nächstes zu konstruierenden Objekts

Befehle (Auswahl):

`F0` "Forward", mit Konstruktion eines Elements (Linienstück, Segment, Gebäudetrakt...), benutzt wird die aktuelle Schrittweite für die Länge (die Null steht für "keine explizite Längenfestlegung")

`M0` forward ohne Konstruktion (*Move*-Befehl)

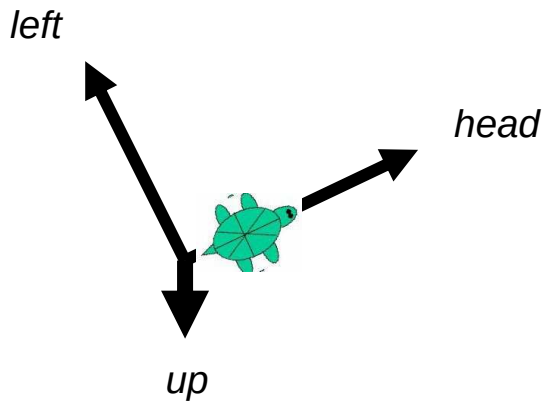
`L(x)` ändere die aktuelle Schrittweite (Länge) zu *x*

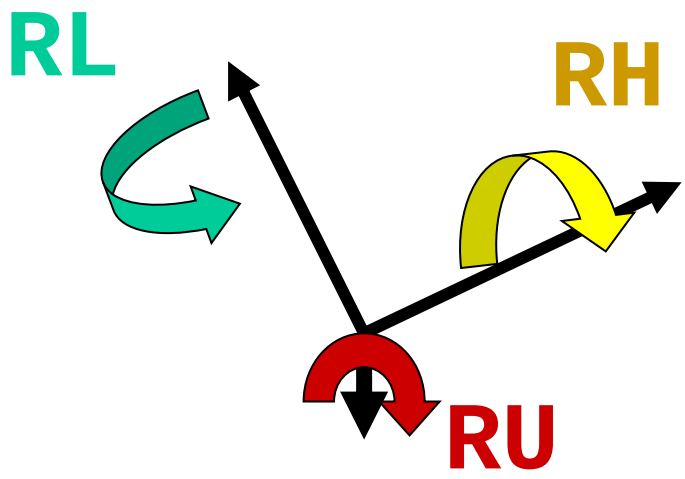
`LAdd(x)` inkrementiere die aktuelle Schrittweite um *x*

`LMul(x)` multipliziere die aktuelle Schrittweite mit *x*

`D(x)`, `DAdd(x)`, `DMul(x)` analog für die aktuelle Dicke

Erweiterung auf 3D-Grafik:
Turtle-Rotationen um 3 Achsen





3D-Befehle:

`RU(45)` Drehung der *turtle* um die "up"-Achse um 45°

`RL(...)`, `RH(...)` analog um "left" und "head"-Achse

up-, *left*- und *head*-Achse bilden ein rechtwinkliges, räumliches Koordinatensystem, das von der *turtle* mitgeführt wird

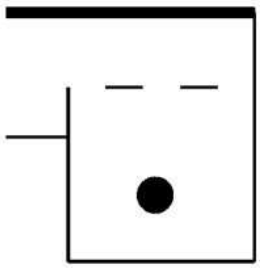
`RV(x)` Rotation "nach unten" mit durch *x* vorgegebener Stärke

`RG` Rotation ganz nach unten (Richtung (0, 0, -1))

Beispiel:

```
L(100) D(3) RU(-90) F(50) RU(90) M0 RU(90) D(10) F0 F0
       D(3) RU(90) F0 F0 RU(90) F(150) RU(90) F(140) RU(90)
       M(30) F(30) M(30) F(30) RU(120) M0 Sphere(15)
```

erzeugt



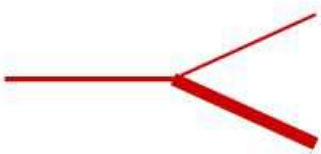
was ist das Ergebnis der Interpretation der Zeichenkette

`L(10) F0 RU(45) F0 RU(45) LMul(0.5) F0 M0 F0` *?*

Verzweigungen: Realisierung mit Speicher-Befehlen

[	lege aktuellen Zustand auf Speicher ("Ablage", Stack)
]	nimm obersten Zustand von der Ablage und mache diesen zum aktuellen Zustand (damit: Ende der Verzweigung)

```
F0 [ RU(-20) F0 ] RU(20) DMul(2) F0
```



Für die Interpretation der Zeichenketten:

Turtle-Geometrie

Der Turtle-Befehlsvorrat wird zu einer Untermenge der Zeichenmenge des L-Systems.

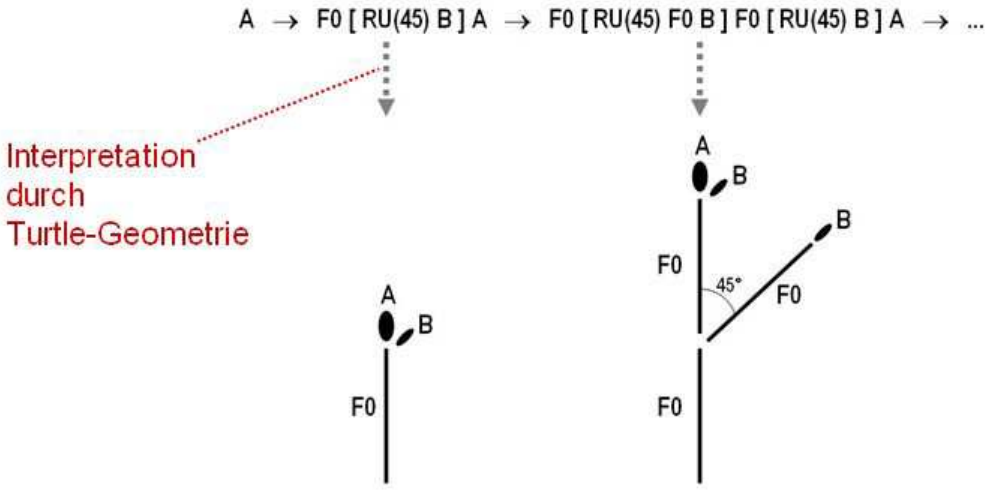
Symbole, die nicht Turtle-Befehle sind, werden von der Turtle ignoriert.

Beispiel:

Regeln

```
A → F0 [ RU(45) B ] A ;
B → F0 B ;
```

Startwort `A`



(`A` und `B` werden normalerweise nicht geometrisch interpretiert.)

was für eine Struktur liefert das L-System

```
A → [ LMul(0.25) RU(-45) F0 ] F0 B;
B → [ LMul(0.25) RU(45) F0 ] F0 A;
```

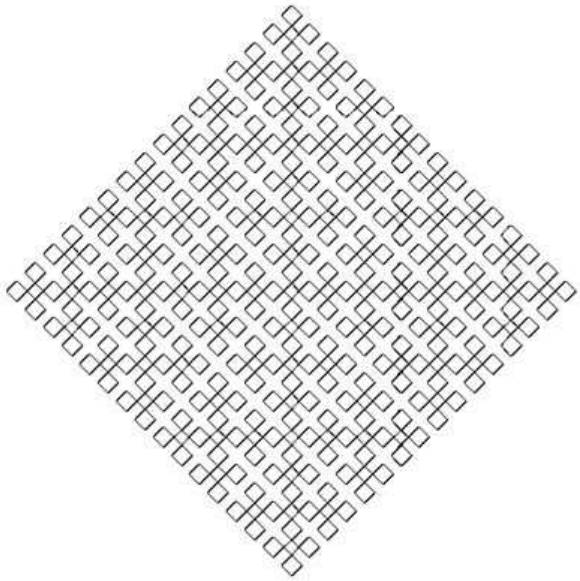
mit Startwort `L(10) A` ?

äquivalente Regel:

```
A → [ LMul(0.25) RU(-45) F0 ] F0 RH(180) A;
```

Flächenfüllende Kurve:

```
Start ==> L(10) RU(-45) X RU(-45) F(1) RU(-45) X;
X ==> X F0 X RU(-45) F(1) RU(-45) X F0 X
```

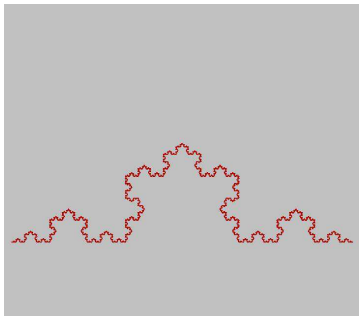


indisches Kolam-Muster
„Anklets of Krishna“

Beispiel für ein Fraktal:

Koch'sche Kurve

```
Start → RU(90) F(10);
F(x) → F(x/3) RU(-60) F(x/3) RU(120) F(x/3) RU(-60) F(x/3)
```



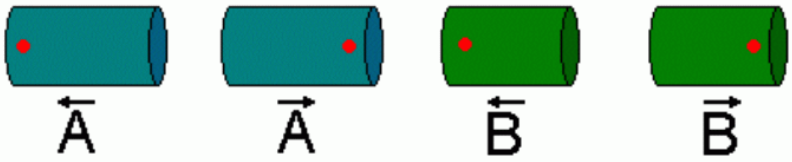
Anwendungsgesichtspunkte von L-Systemen:

- Verwendung hauptsächlich für Vegetationsmodelle
  aber auch: Webmuster, Gebäude, Roboter, Tiere (Vermehrung, Nahrungsaufnahme), Melodien
- 2D- und 3D-Varianten
- Emulation von iterierten Funktionssystemen möglich (siehe Prusinkiewicz & Lindenmayer 1990)
- *dynamische* Simulationen (⇒ Möglichkeit der Animation)
- Anbindung physikalisch oder biologisch begründeter Simulationsmodelle möglich (Kombination von Modell-Ansätzen)

Softwaresysteme zur Umsetzung:

- cpfg / LStudio
  http://algorithmicbotany.org/virtual_laboratory/
- Grogra **http://www.grogra.de**
- Graphtal
- LParser *(nicht mehr im Web)*
- LGrammar **http://www.ee.uwa.edu.au/~braunl/lgrammar**
- GroIMP **http://www.grogra.de**

Erstes Beispiel: Anabaena Catenula



Blaugrünes Bakterium, existiert in vier Varianten:

- „groß": Rechts-A und Links-A
- „klein": Rechts-B und Links-B

Beobachtungen:

- R-A teilt sich in L-A und R-B und R / L vertauscht
- R-B wird spontan zu R-A, L-B zu L-A

L-System für Anabaena Catenula

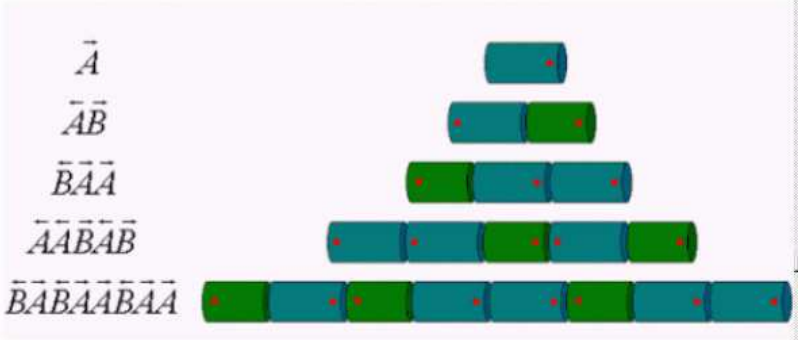
ω : $\vec{A}$

p_1 : $\vec{A} \rightarrow \vec{A}\vec{B}$

p_2 : $\vec{A} \rightarrow \vec{B}\vec{A}$

p_3 : $\vec{B} \rightarrow \vec{A}$

p_4 : $\vec{B} \rightarrow \vec{A}$



Simultane Ersetzung („parallel rewriting")

Grundversion gut für alle Strukturen mit *lokal 1-dimensionalem Grundgerüst* (Verzweigungssysteme).

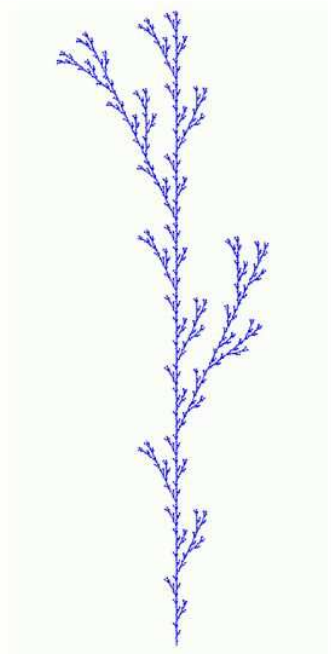
Beispiele

Verzweigungsbeispiel:

`F0 → F0 [ RU(25.7) F0 ] F0 [ RU(-25.7) F0 ] F0 ;`

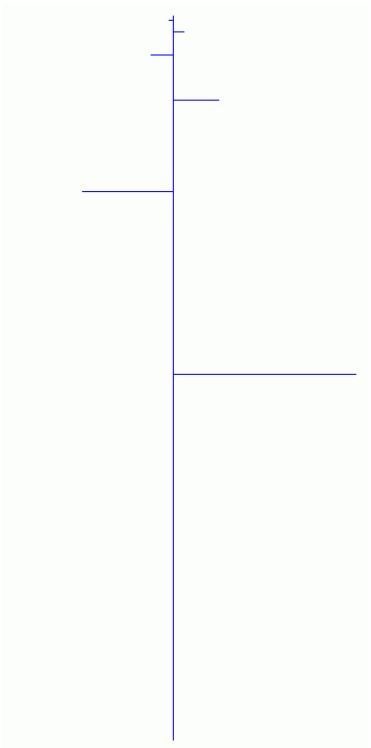
Ergebnis nach 7 Schritten:

(Startwort `L(10) F0`)



botanisch "realistischeres" Beispiel:
Verzweigung, alternierende Zweigstellung und Verkürzung.

```
Axiom ==> F0 A;
A     ==> LMul(0.5) [ RU(90) F0 ] F0 RH(180) A;
```



welche Struktur liefert

```
α → F(10) A ;

A → [ RU(-60) F(6) RH(180) A Sphere(3) ]
    [ RU(40) F(10) RH(180) A Sphere(3) ];
Sphere → Z;   ?
```

(`F(n)` liefert Linie der vorgegebenen Länge *n*,
`Sphere(n)` eine Kugel mit Radius *n*)

Stochastische L-Systeme: Einbau von Zufallsauswahl

Nichtdeterministische L-Systeme: ein Beispiel

$\omega: \quad F$

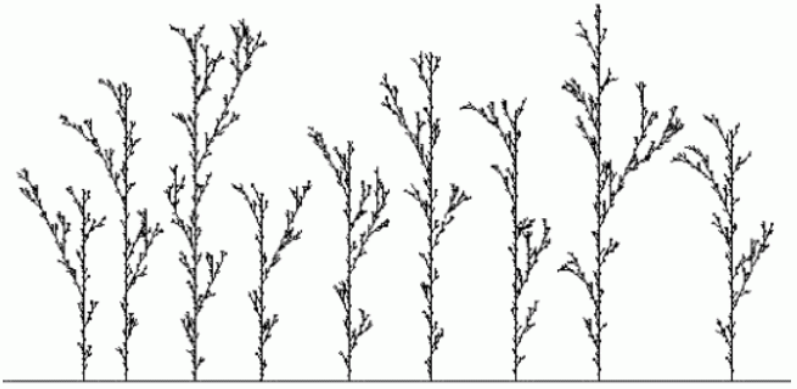
$p_1: \quad F \xrightarrow{0.33} F[+F]F[-F]F$

$p_2: \quad F \xrightarrow{0.33} F[+F]F$

$p_3: \quad F \xrightarrow{0.33} F[-F]F$

$\delta = 30°$

Regeln werden zufällig (hier mit gleichem p) ausgewählt



(in diesem Beispiel – aus Prusinkiewicz & Lindenmayer (1990) – steht "+" für "RU(δ)" und "–" für "RU(–δ)". "F" steht für "F0".)

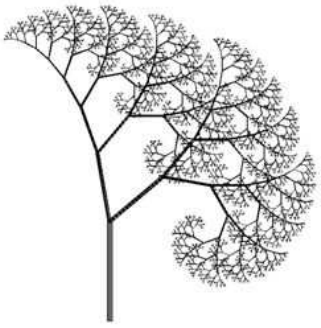
Stochastische L-Systeme

Verwendung von Pseudozufallszahlen

Beispiel:

deterministisch

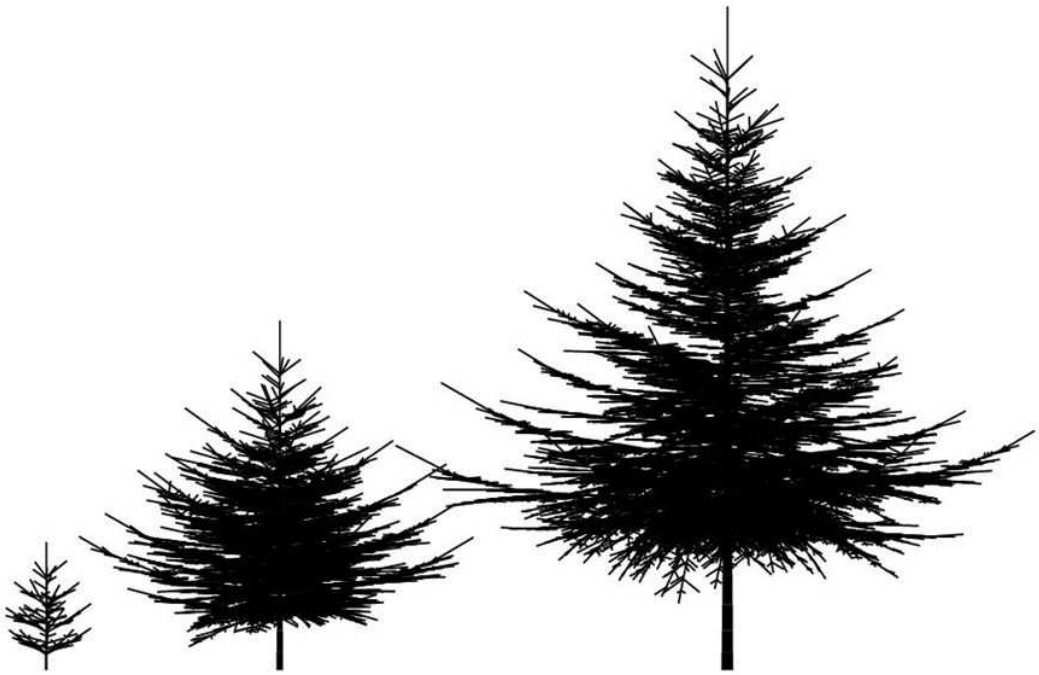
```
Start ==> L(100) D(5) A;
A ==> F0 LMul(0.7) DMul(0.7)
   [ RU(50) A ] [ RU(-10) A ];
```



stochastisch

```
Start ==> L(100) D(5) A;
A ==> F0 LMul(0.7) DMul(0.7)
  if (probability(0.5))
   ( [ RU(50) A ] [ RU(-10) A ] )
  else
   ( [ RU(-50) A ] [ RU(10) A ] );
```

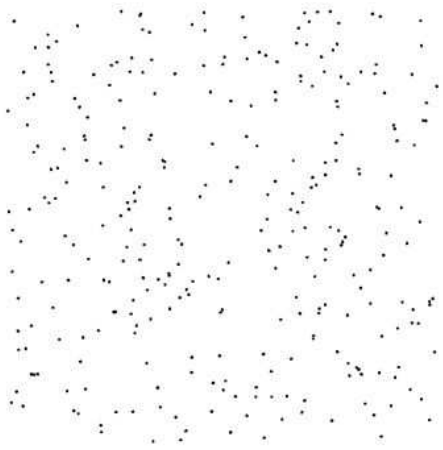
Beispiel: Fichtenmodell in 3D
mit L-System erzeugt



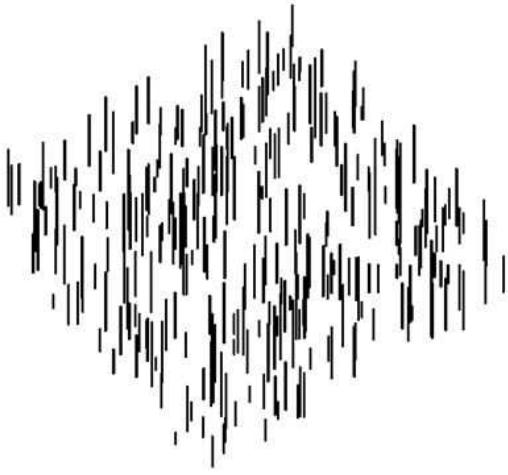
Erzeugung einer Zufallsverteilung in der Ebene:

```
Axiom ==> D(0.5) for ((1:300))
              ( [ Translate(random(0, 100), random(0, 100), 0)
                  F(random(5, 30)) ] );
```

Ansicht von oben



schräg von der Seite



Erweiterung des Symbol-Konzepts:

Lasse reellwertige Parameter nicht nur bei Turtle-Kommandos wie "`RU(45)`" und "`F(3)`" zu, sondern bei allen Zeichen

→ *parametrische L-Systeme*

beliebig lange, endliche Parameterlisten
Parameter werden bei Regel-Matching mit Werten belegt

Beispiel:

Regel `A(x, y) → F(7*x+10) B(y/2)`

vorliegendes Zeichen z.B.: `A(2, 6)`
nach der Regelanwendung: `F(24) B(3)`

Parameter können in Bedingungen abgeprüft werden
(logische Bedingungen mit Java-Syntax):

`A(x, y) (x >= 17 && y != 0) → ....`

Welche Struktur wird von folgendem L-System erzeugt?

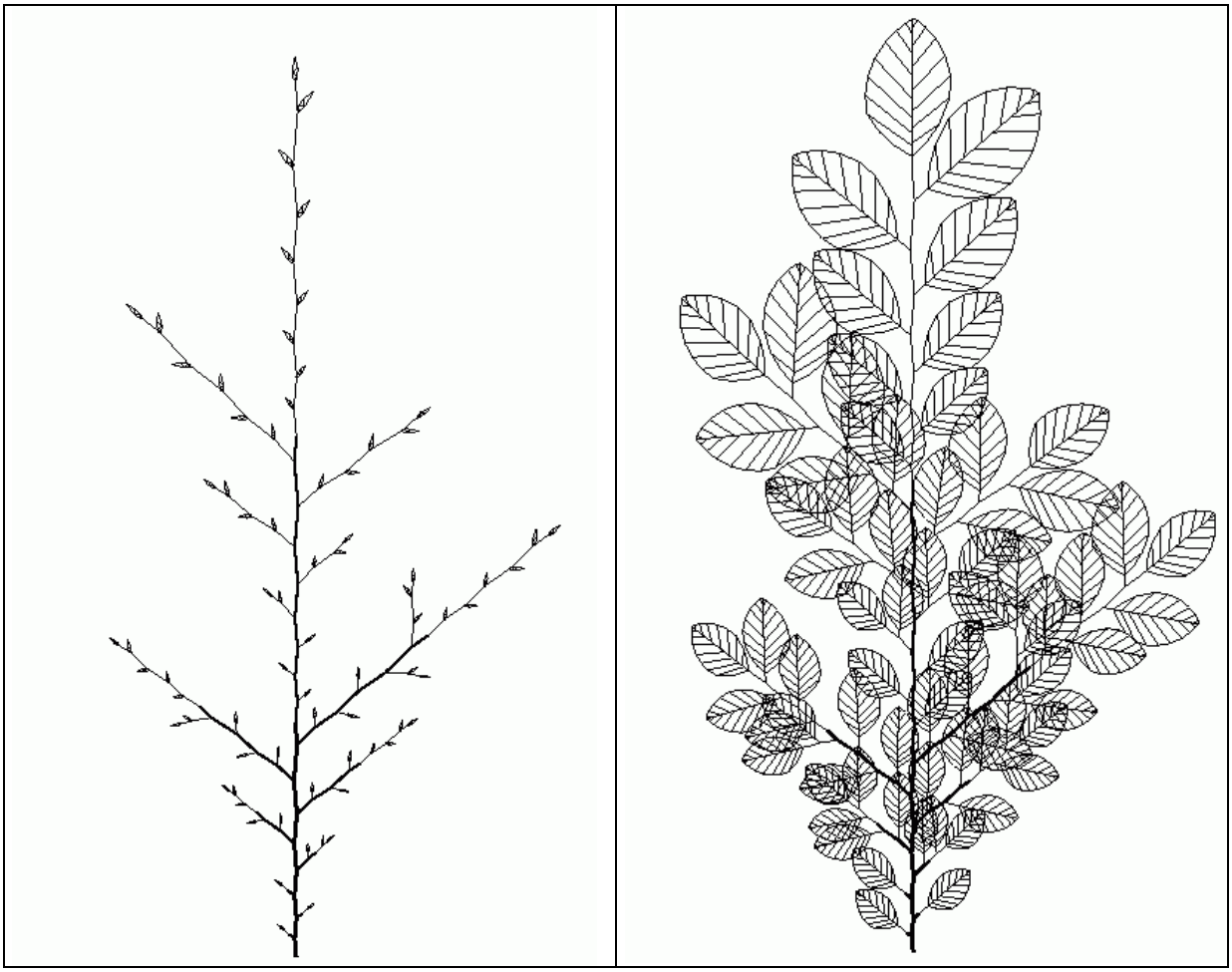
```
α    → [ RU(90) M(1) RU(90) A(1) ] A(1);
A(n) → F(n) RU(90) A(n+1);
```

Variante:

in der zweiten Regel "`RU(90)`" etwa durch "`RU(92)`" ersetzen.

Beispiel Minze (von Prusinkiewicz & Lindenmayer):

Beispiel Buchenzweige:



Nachteil der bisher vorgestellten L-Systeme:
Kontrolle nur durch Vorgänger-Symbol ("lineage control") oder stochastisch

⇒ fehlende Interaktion innerhalb des modellierten Objekts oder mit der Umwelt

⇒ Determinismus oder stochastische Modelle ohne kausale Komponenten

Abhilfe: Einführung von *Sensitivität* bei der Regelanwendung.

(a) *Kontext*sensitivität

(schon altes Konzept, Beispiele bereits bei Lindenmayer...):

Abhängigkeit einer Regelanwendung vom linken und / oder rechten Kontext im String:
leftcontext < **a** > *rightcontext* → β.
stringbasiert!

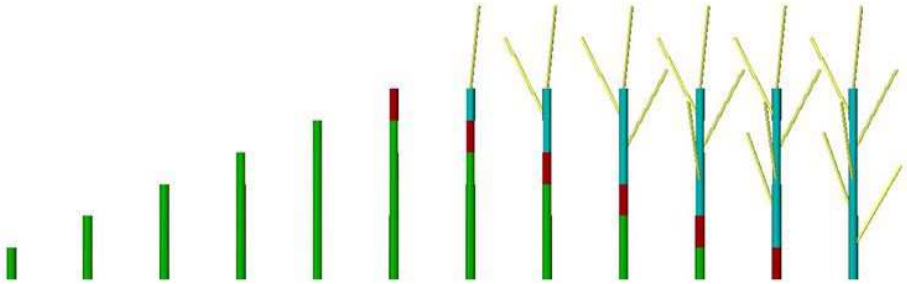
Kontextsensitivität

Abfrage eines Kontexts, der vorhanden sein muss, damit eine Regel anwendbar ist

Angabe des Kontexts in (* .... *)

Beispiel:

```
module A(int age);
module B(super.length, super.color) extends F(length, 3, color);
Axiom ==> A(0);
A(t), (t < 5) ==> B(10, 2) A(t+1);
A(t), (t == 5) ==> B(10, 4);
B(s, 2) (* B(r, 4) *) ==> B(s, 4);
B(s, 4) ==> B(s, 3) [ RH(random(0, 360)) RU(30) F(30, 1, 14) ];
```



Verwendung:

- Weiterleitung von Signalen innerhalb der modellierten Struktur

  s < a > → a s,

  s → , /* leeres Wort */
- Konzentration von Substanzen (Hormonen)
- Bewegung von Objekten (z.B. Insekten auf der Pflanze)

Beispiel: Entwicklung von Blütenständen häufig hormonal gesteuert

häufig: Zusammenwirken von 2 Signalen (von unten und von oben)

- das Modell kann Aufschluss geben, ob für eine in der Natur beobachtete morphologische Sequenz der Blütenbildung 1, 2 oder mehr Pflanzenhormone notwendig sind.

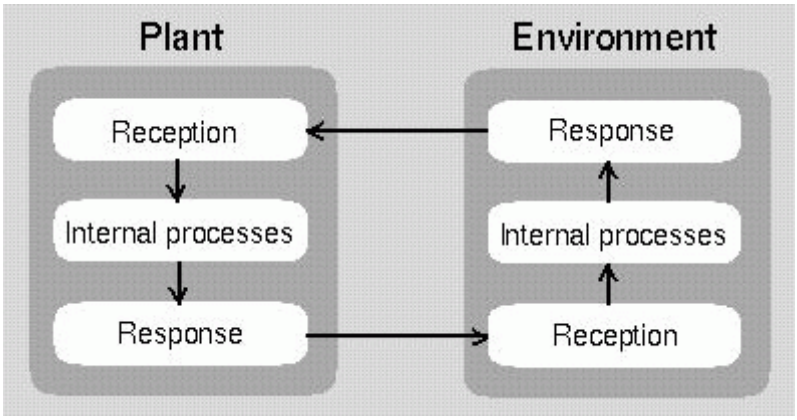
weiteres Beispiel von Prusinkiewicz & Lindenmayer (1990): *Lychnis coronaria*

(b) globale Sensitivität
(auch: *environmentally sensitive L-systems*)

- Kommunikation mit der Umgebung über spezielle Kommunikationsmodule oder über sensitive Funktionen
- Regelanwendung hängt (potentiell) von der gesamten, aktuell vorhandenen Struktur im Objektraum (und von eventuellen externen Eingriffen) ab (nicht nur von der String-repräsentation)
- Schnittstelle zu physikalisch oder biologisch basierten Simulationsmodellen (man spricht dann von *open L-systems*)

Zwei Herangehensweisen:

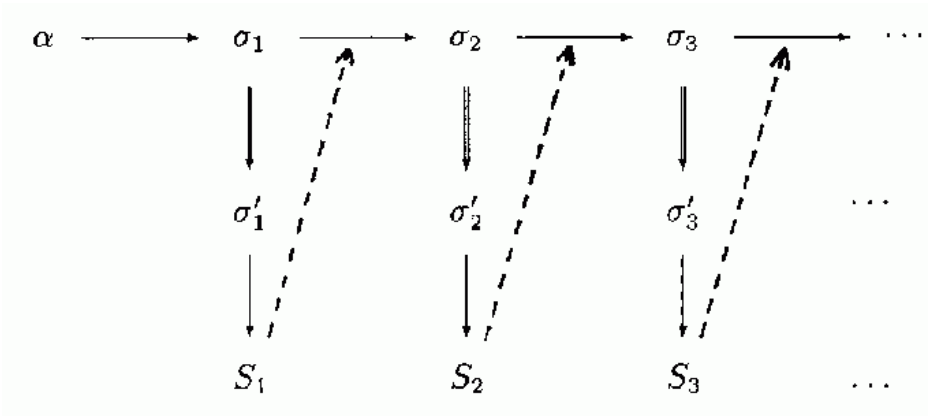
(a) scharfe Trennung von Organismus und Umgebung; der lebende Organismus wird mittels L-System modelliert, die Umgebung mittels anderer Modelle, Implementation als parallele, kommunizierende Prozesse



(Prusinkiewicz)

(b) durch Erweiterung des L-System-Formalismus Aspekte der Umgebung ins L-System "hineinholen"

Prinzip der Anwendung global sensitiver Funktionen:



gestrichelte Pfeile: Informationsfluss von der erzeugten Objektstruktur zur Regelanwendung (Regelauswahl, Parametrisierung) auf die Strings.

⇒ konzeptionell Annäherung an Graph-Grammatiken (dort würden nur noch S_1, S_2 usw. existieren, keine Strings mehr).

Beachte:

- kontextsensitive L-Systeme entsprechen den "selbst-regulierten Mustern" mit endogener Kontrolle,
- global sensitive Systeme den "sehenden Mustern" mit exogener Kontrolle (in der Systematik der Morphogenese-Modelle nach Adrian D. Bell).

einfaches Beispiel:

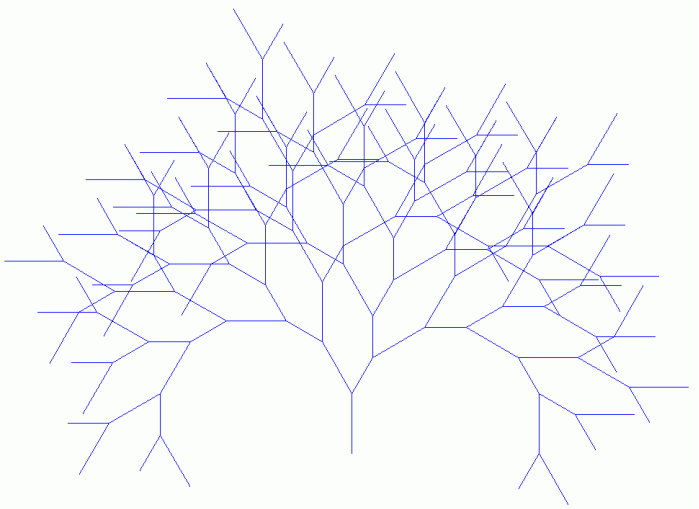
Nichtsensitives und global sensitives L-System im Vergleich

nichtsensitive, dichotome Verzweigung (Syntax von GROGRA):

```
\axiom a 1-8,
\angle 30,
a → RH(180) F(100) [ - b ] + a,
b → RH(180) F(70) [ - b ] + a
```

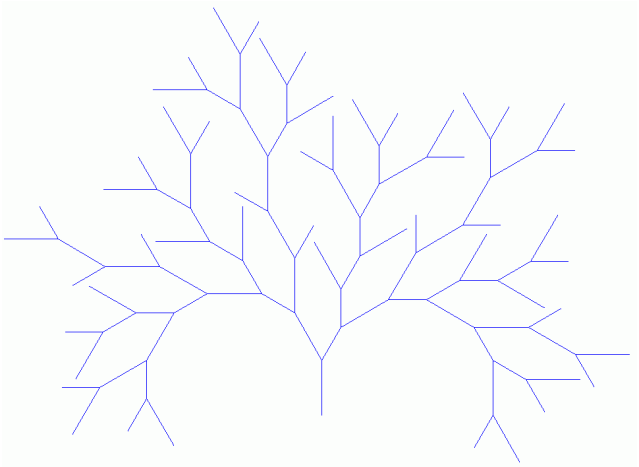
(Beispiel wird später noch einmal mit XL-Syntax aufgegriffen!)

Ergebnis:



sensitive Verzweigung mit Abhängigkeit vom Abstand zum nächsten Nachbar-Element (im Objektraum!):

```
\axiom a 1–8,
\angle 30,
\var f function 2 1,
(f(1) > 60) a → RH(180) F(100) [ – b ] + a,
(f(1) > 60) b → RH(180) F(70) [ – b ] + a
```



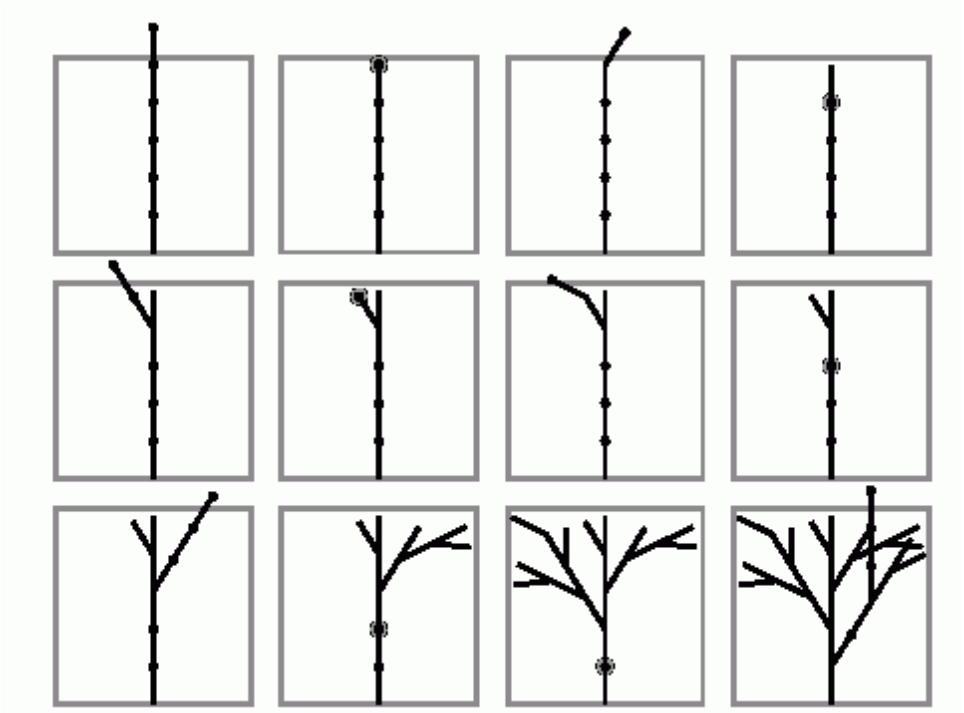
Anwendung in der Pflanzenmodellierung:

- Dichteabhängigkeit des Wachstums
- Einfluss des Neigungswinkels eines Astes auf den Neuaustrieb
- Einfluss der Beschattung
- Wechselwirkung mit Herbivoren (Tiere)

äußere Eingriffe: Beschneiden von Gehölzpflanzen, z.B. im Gartenbau, um eine bestimmte Form zu erzielen

- botanischer Mechanismus: Induktion der Bildung von Ersatztrieben
- Modell: sensitives L-System, das Überschreiten einer räumlich vorgegebenen Grenze registriert und wachstumsanregendes Signal induziert, dieses bringt schlafende Knospen zum Austrieb

aus Prusinkiewicz et al. (1994):



(kleine Kreise: schlafende Knospen, großer Kreis: Wachstumssignal – wandert von oben nach unten bis zur nächsten schlafenden Knospe)

3D-Version, gerendert:

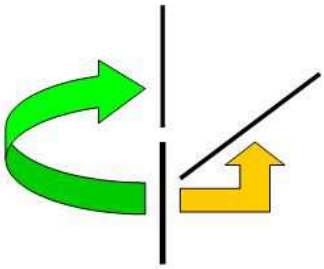
Moseley Old Hall Garden:

Relationale Wachstumsgrammatiken

Der Schritt zu relationalen Wachstumsgrammatiken

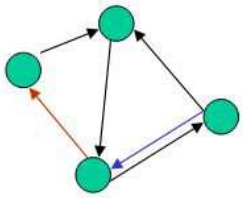
Nachteil von L-Systemen:

- 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)

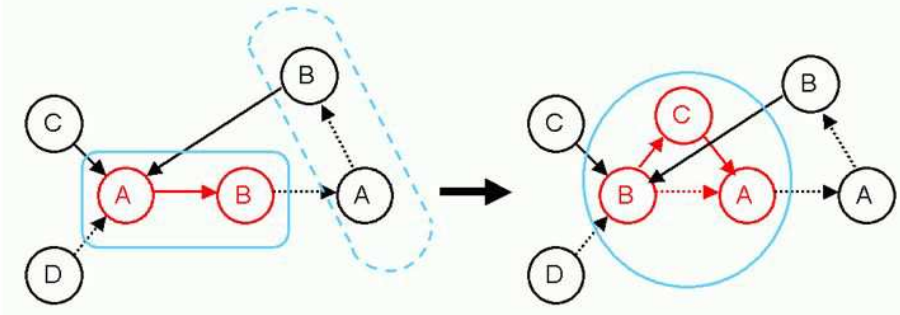


ebenfalls regelbasierter Mechanismus:

Graph-Grammatiken

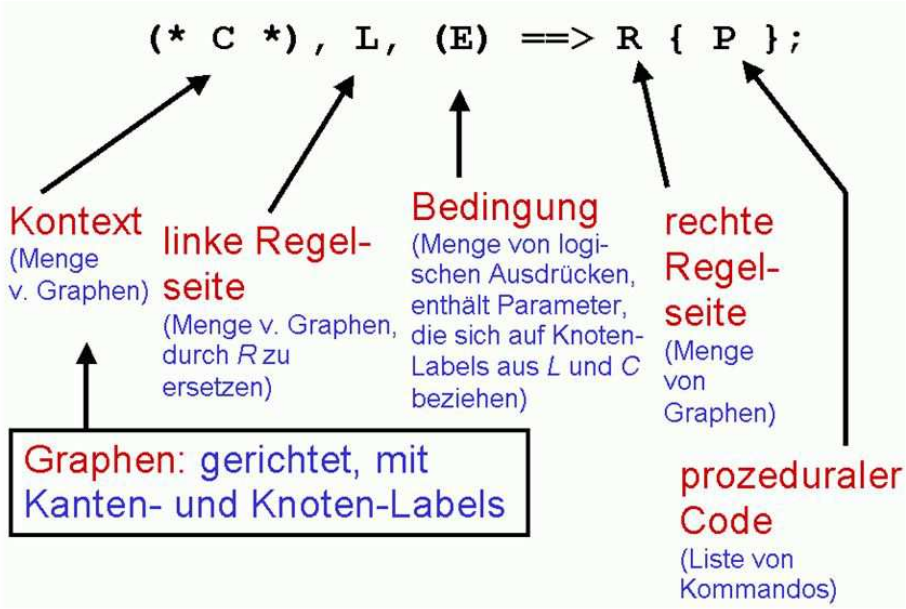
Regel:

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;`

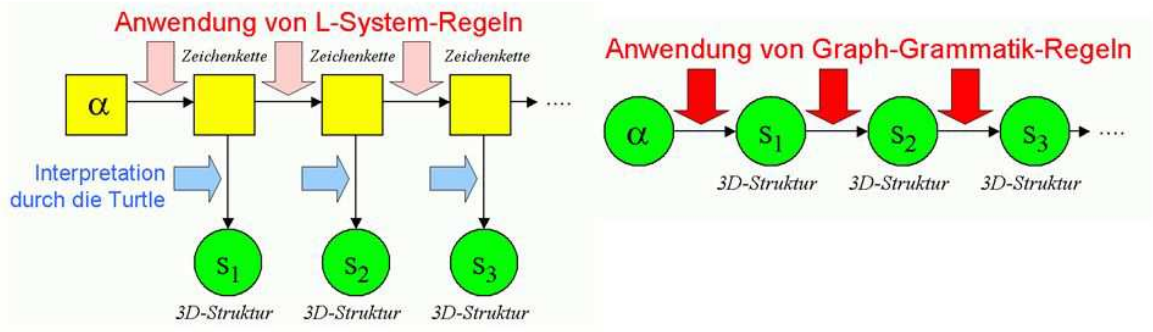
Instanzierungsregeln: einzelne Zeichen werden in Substrukturen aufgelöst, ohne Einfluss auf den nächsten Entwicklungsschritt
(Regel muss dann direkt in der Moduldeklaration stehen)

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:

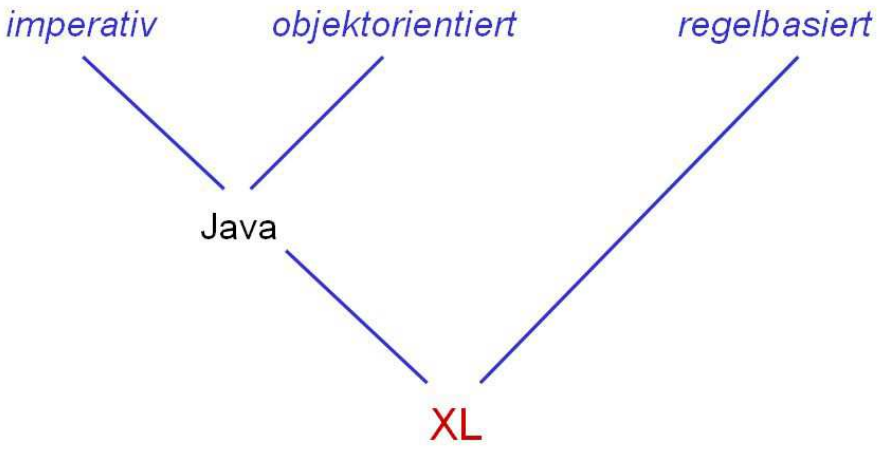
• Knoten des Graphen können beliebige Objekte sein (auch Grafikobjekte)

• 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): [...]

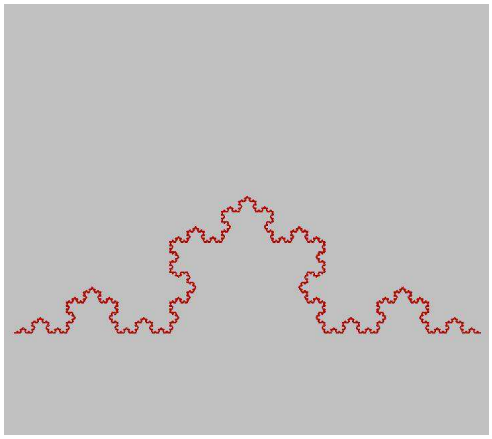
- Knoten der Graphen sind Java-Objekte, auch Geometrie-Objekte

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);
  ]
```

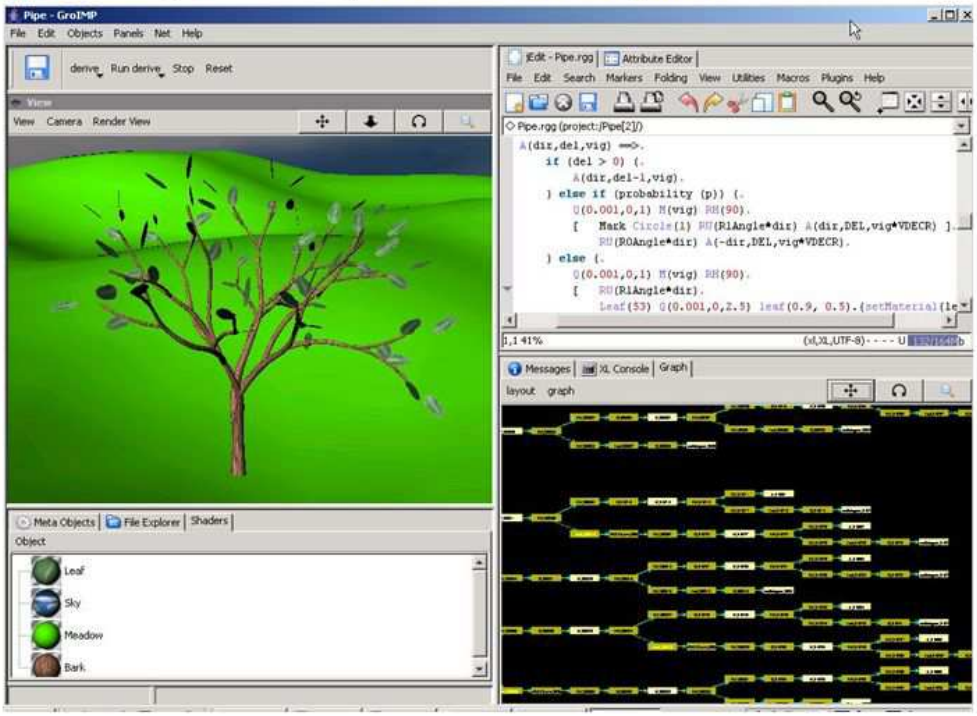
Knoten des Graphen

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

https://gitlab.com/grogra/groimp/-/releases

Anwendungsbeispiel: Modellierung von Parklandschaften

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

mit GroIMP
generierte Erle
in VRML-Welt

Ausschnitte entnommen aus
Beckert (2010), Lange (2002),
Prusinkiewicz et al. (1994), Lindenmayer & Prusinkiewicz (1990)
(genaue Quellenangaben siehe http://www.uni-forst.gwdg.de/~wkurth/fs10_lit.htm)