Tutorial and workshop „Modelling with GroIMP and XL“ / Tutorial for beginners

University of Göttingen, 27 February, 2012

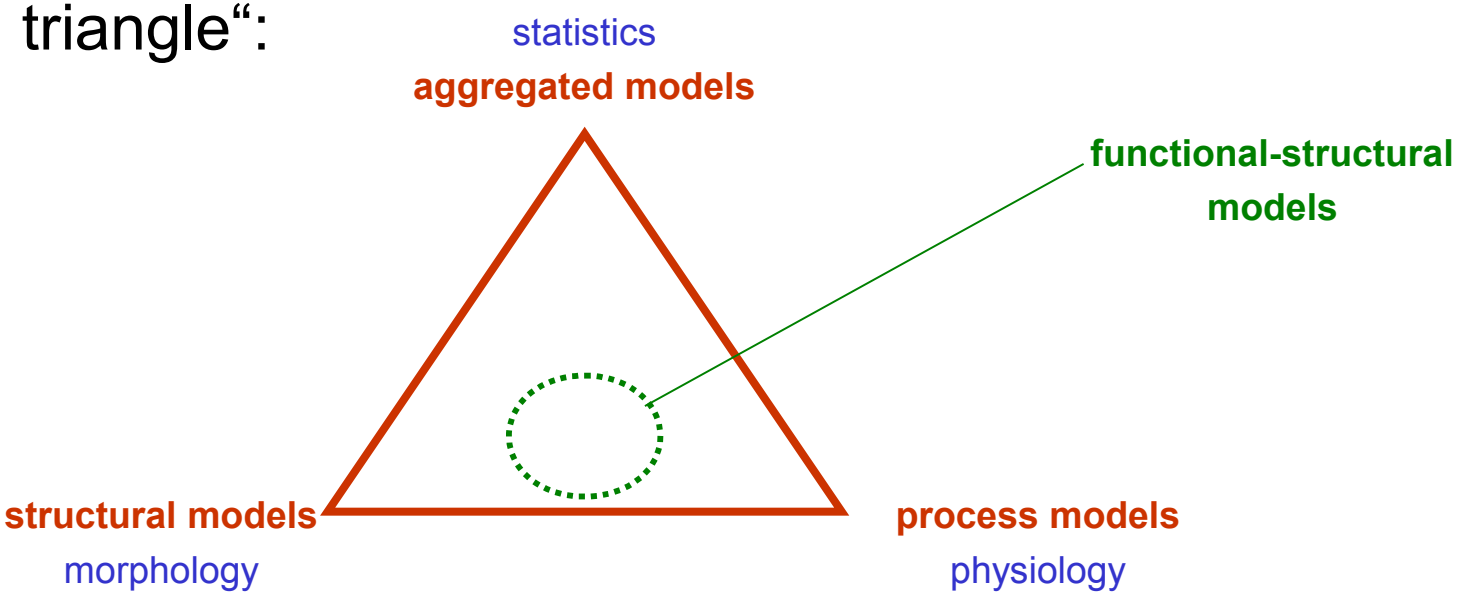
Winfried Kurth

Introduction to rule-based programming, L-systems and XL

Motivation

Functional-structural plant models (FSPM)

„model triangle“:



- Linking of botanical structures and functions (e.g., light interception, water flow) in a *coherent, single* model
- processes linked to morphological objects

Structural models

3 levels:

1. static description of structure

   plant at a fixed date (e.g., at 27 September 2011)

2. dynamic description of structure, non-sensitive

   description of development (ontogenesis) of a plant:
   time series of 3-dimensional structures

3. dynamics, taking causal impacts / conditions into account (sensitive models)

   different paths of development
   logical conditions for the decision between them
   (simplest case: stochastic)

concerning 1.: static description of structure

two approaches:

(a) tables

each morphological unit of a plant = one row

dtd code = „descriptive tree data“, or mtg code

(b) imperative (command-driven):

„Turtle geometry“

virtual turtle „constructs“ the structure,

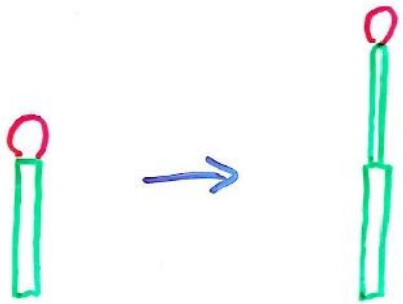
the description are the commands which control it

turtle geometry command language

The second level of description:

Dynamic description of plant structures

- how do plants change during ontogenesis?



The approach of AMAP

Atelier de Modélisation de l'Architecture des Plantes

Montpellier, Paris, Beijing (LIAMA)

Ph. de Reffye, R. Lecoustre, M. Jaeger, E. Costes,
P. Dinouard, F. Blaise, P.-H. Cournède et al.
(agronomists, computer scientists, botanists, mathematicians)

Modelling the activity of meristems

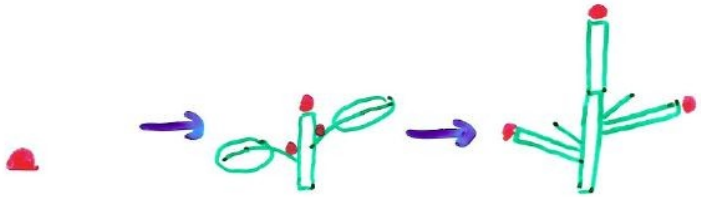
shape of tree = trajectory of its meristems

approach for modelling:

shape of tree = trajectory of meristems

- primary meristem

- branching



- secondary meristem

(to be added:
mechanic deformations, deformations with physiological causes, damages, processes of senescence and mortality)

meristem-based modelling approach

Adrian D. Bell 1979:

3 basic processes

- formation of a shoot (growth)
- transition to resting state (and new activation)
- death

similarly de Reffye 1981:

3 meristem states

- dormance (sleeping)
- croissance (growth)
- mortalité (death)

state transitions with probabilities

$\rightarrow$ binomial distribution, Markov chains

The software GroIMP

„Growth-grammar related Interactive Modelling Platform“

- download from Sourceforge (free & open source)
- rgg files, projects (gsz files)
- editor, environment for development
- window for 3-d view
- 2-d (graph) window (usually hidden!)
- attribute view for each object
- camera position
- navigation
- interactive modelling
- compiler for the programming language XL
- framework for solving ODEs in the context of plant models

XL: a multi-paradigm language

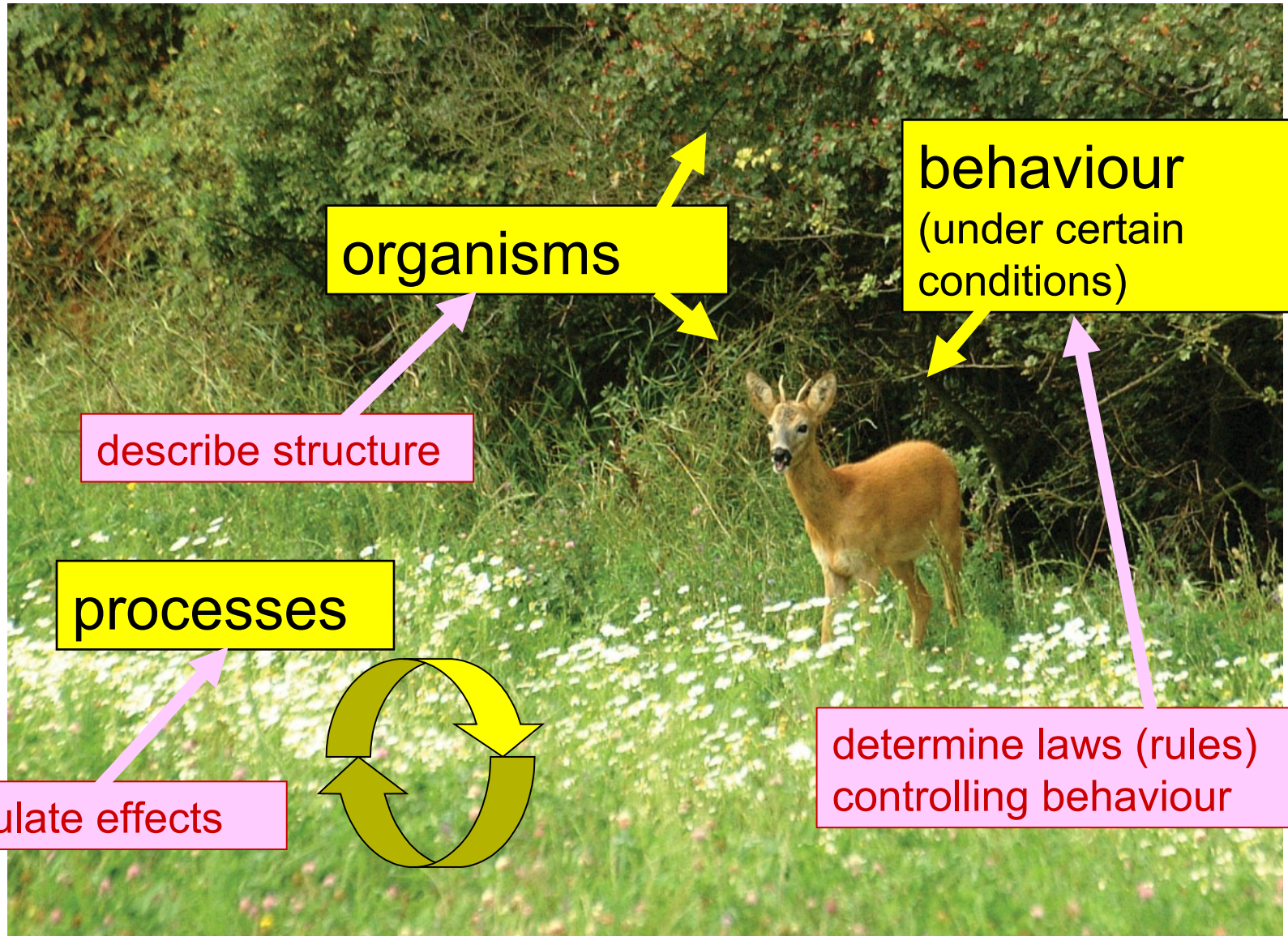
Robert Floyd 1978:

Turing Award Lecture

"The Paradigms
of Programming"

Robert W. Floyd (1936-2001)

Ecosystem:


organisms
behaviour
(under certain
conditions)
describe structure
processes
calculate effects
determine laws (rules)
controlling behaviour

Some important paradigms of programming

- *for numerical simulation of processes:*

imperative paradigm
(also: von-Neumann paradigm, control flow paradigm)

John von Neumann (1903-1957)

imperative programming:

computer = machine for the manipulation of values of variables

(these manipulations can have side effects).

programme = plan for the calculation process with specification of the commands and of the control flow (e.g. loops).

example:

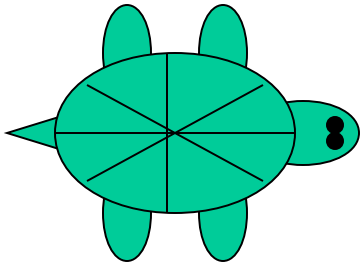
```
x = 0;
while (x < 100)
    x = x + 1;
```

programming languages which support imperative programming:

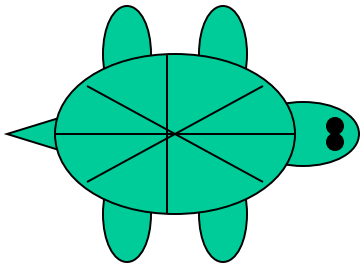
Fortran, Pascal, C, ..., parts of Java, ...,
command language of turtle geometry

Turtle:

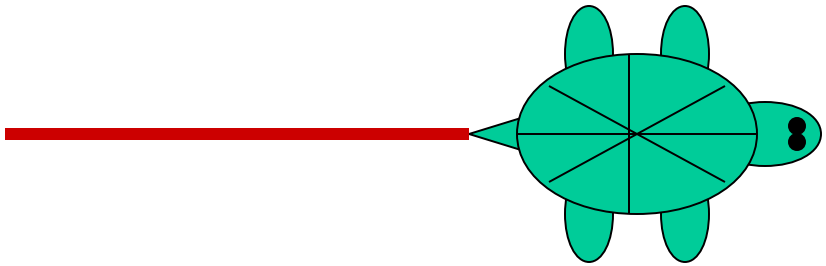
goes according to commands



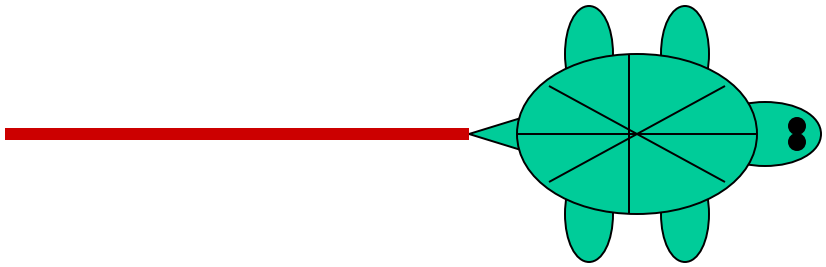
F0



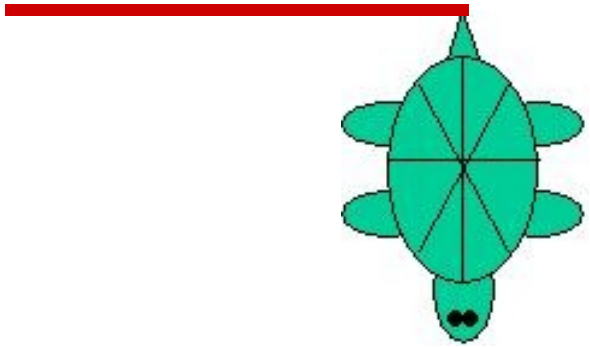
F0



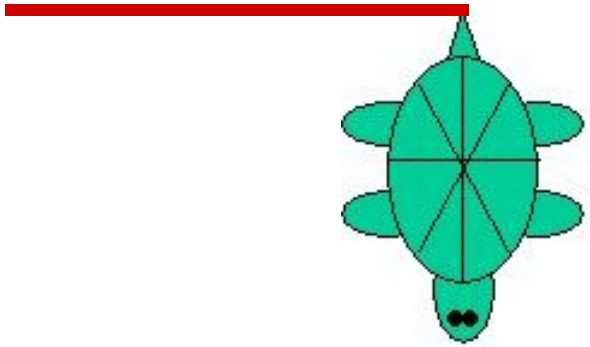
```
F0 RU(90)
```



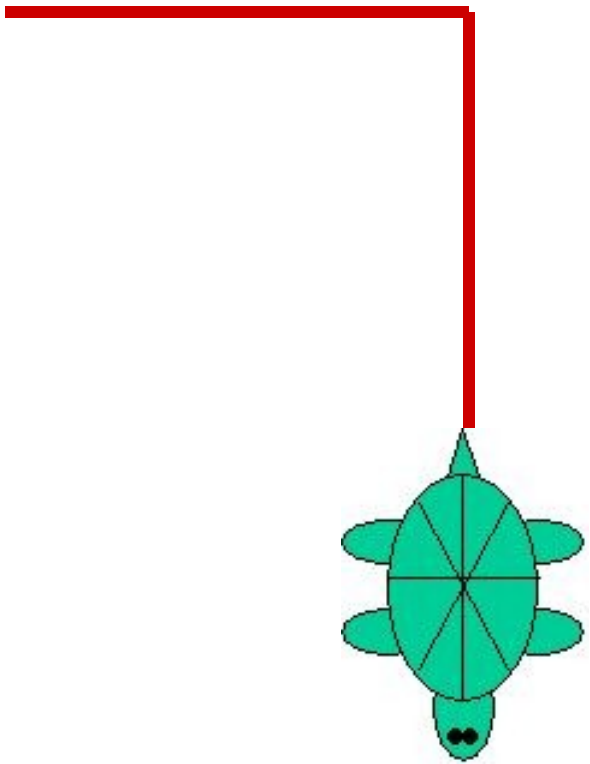
F0 RU(90)



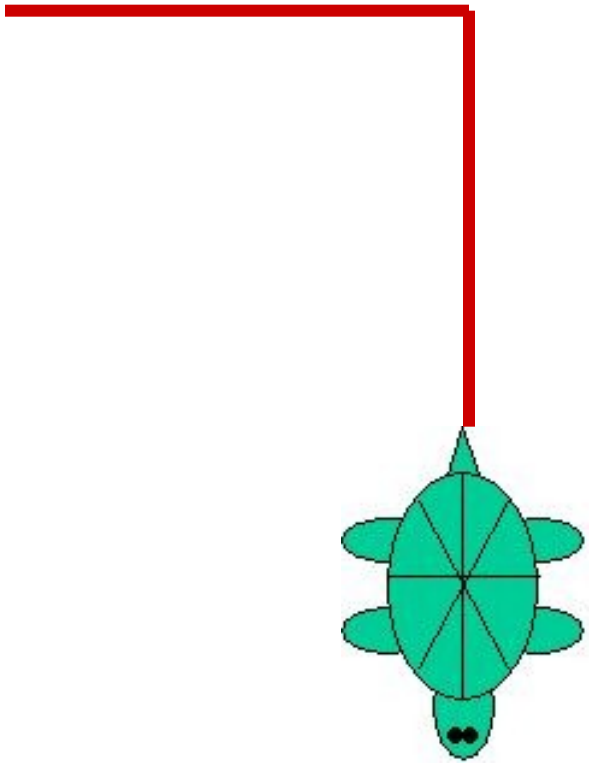
F0 RU(90) F0



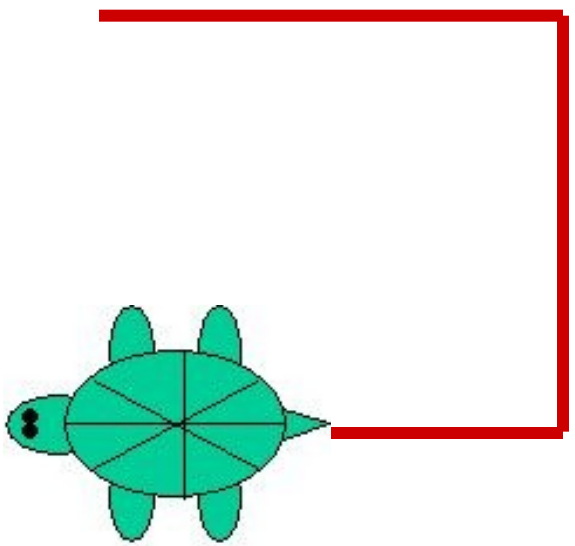
```
F0 RU(90) F0
```



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



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



object-oriented paradigm

computer = environment for virtual objects

programme = list of (object) *classes*, i.e. general specifications of objects, which can be created and destroyed at runtime.

programming languages: Smalltalk, Simula, C++, Java, ...

example:

Inheritance of attributes and methods from superclasses to subclasses

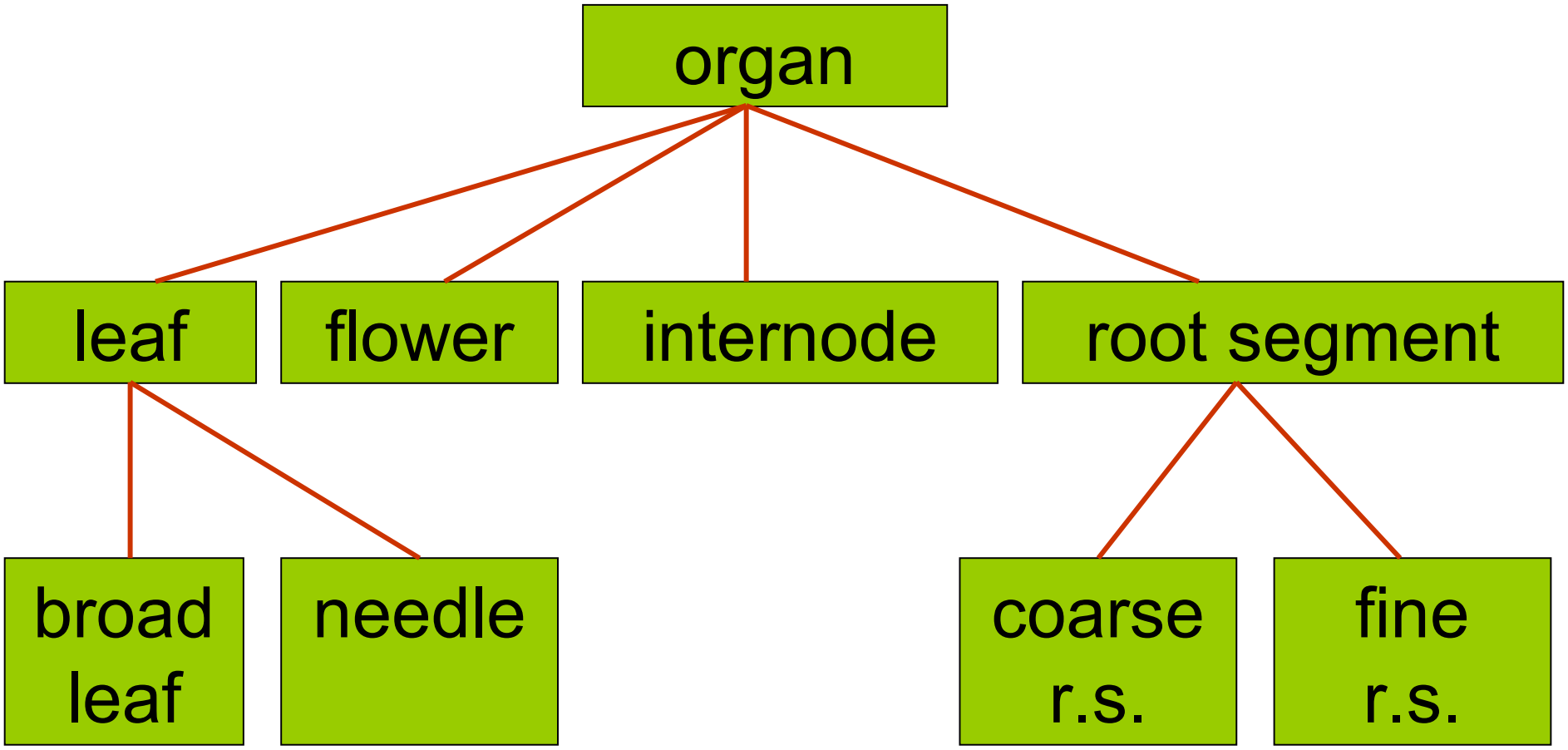
```
public class Car extends Vehicle
    {
    public String name;
    public int places;
    public void show()
         {
         System.out.println("The car is a " + name);
         System.out.println("It has " + places + "places.");
         }
    }
```

typical:

classes (**`Car`**) with data (**`name`**, **`places`**) and methods (**`show`**)

usefulness of object hierarchies in biology

for example:



rule-based paradigm

computer = machine transforming structures

There is a current structure (in XL: a graph) which is transformed as long as it is possible.

Work process: search and application.
matching: search for a suitable rule,
rewriting: application of the rule, thereby transformation of the structure.

programme = set of transformation rules

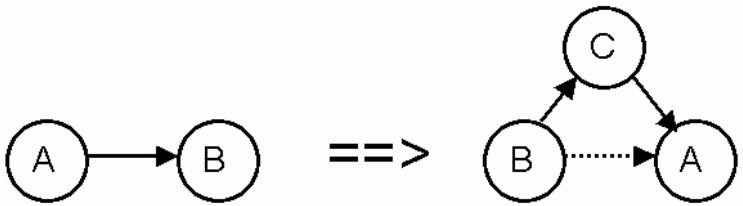
to find a programme: specification of rules.

programming languages: L-system languages, AI languages, Prolog, ...

Example:

a graph grammar

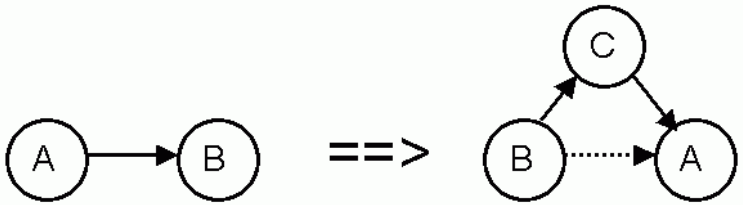
rule:



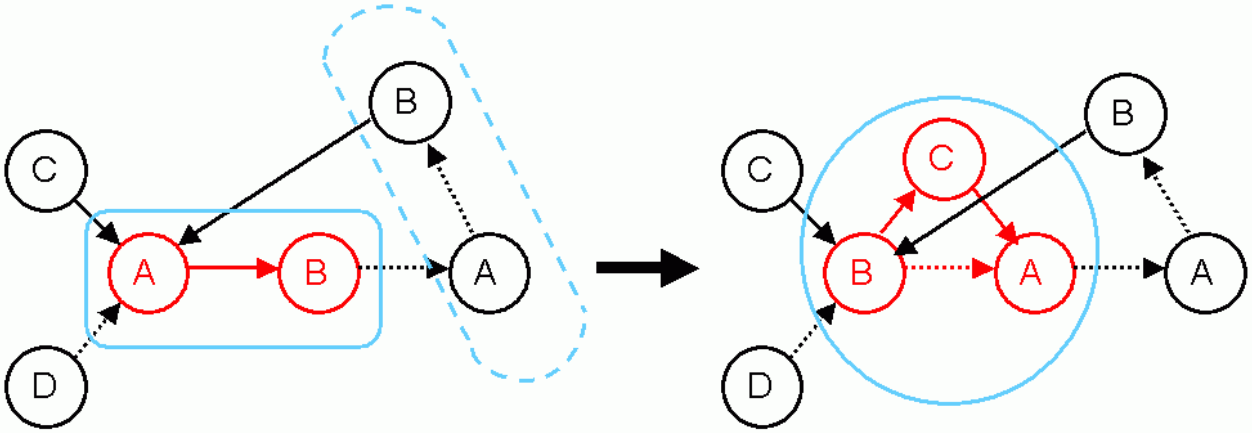
Example:

a graph grammar

rule:



application:



Dynamical description of structures

L-systems (Lindenmayer systems)

rule systems for the replacement of character strings

in each derivation step *parallel* replacement of all characters for which there is one applicable rule

by A. Lindenmayer (botanist) introduced in 1968 to model growth of filamentous algae

Aristid Lindenmayer (1925-1989)

L-systems mathematically:

a triple (Σ, α, *R*) with:

Σ a set of characters, the *alphabet*,

α a string with characters from Σ, the *start word* (also "Axiom"),

R a set of rules of the form

character → string of characters;

with the characters taken from Σ.

A *derivation step* (rewriting) of a string consists of the replacement of all of its characters which occur in left-hand sides of rules by the corresponding right-hand sides.

Convention: characters for which no rule is applicable stay as they are.

Result:

Derivation chain of strings, developed from the start word by iterated rewriting.

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

Example:

alphabet {A, B}, start word A

set of rules:

$A \rightarrow B$
$B \rightarrow AB$

A

Example:

alphabet {A, B}, start word A

set of rules:

$A \rightarrow B$

$B \rightarrow AB$

B

Example:

alphabet {A, B}, start word A

set of rules:

A $\rightarrow$ B

B $\rightarrow$ AB

AB

parallel replacement

Example:

alphabet {A, B}, start word A

set of rules:

$A \rightarrow B$

$B \rightarrow AB$

BAB

Example:

alphabet {A, B}, start word A

set of rules:

$A \rightarrow B$
$B \rightarrow AB$

BAB

Example:

alphabet {A, B}, start word A

set of rules:

$A \rightarrow B$
$B \rightarrow AB$

ABBAB

Example:

alphabet {A, B}, start word A

set of rules:

$A \rightarrow B$
$B \rightarrow AB$

derivation chain:

$A \rightarrow B \rightarrow AB \rightarrow BAB \rightarrow ABBAB \rightarrow BABABBAB$
$\rightarrow ABBABBABABBAB \rightarrow BABABBABABBABBABABBAB$
$\rightarrow ...$

still missing for modelling biological structures in space:

a geometrical interpretation

Thus we add:

a function which assigns to each string a subset of 3-D space

„interpreted“ L-system processing

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

$$\downarrow \quad \downarrow \quad \downarrow$$

$$S_1 \quad S_2 \quad S_3 \quad ....$$

S_1, S_2, S_3, ... can be seen as developmental steps of an object, a scene or an organism.

For the interpretation:

turtle geometry

the turtle command set becomes a subset of the character set of the L-system.

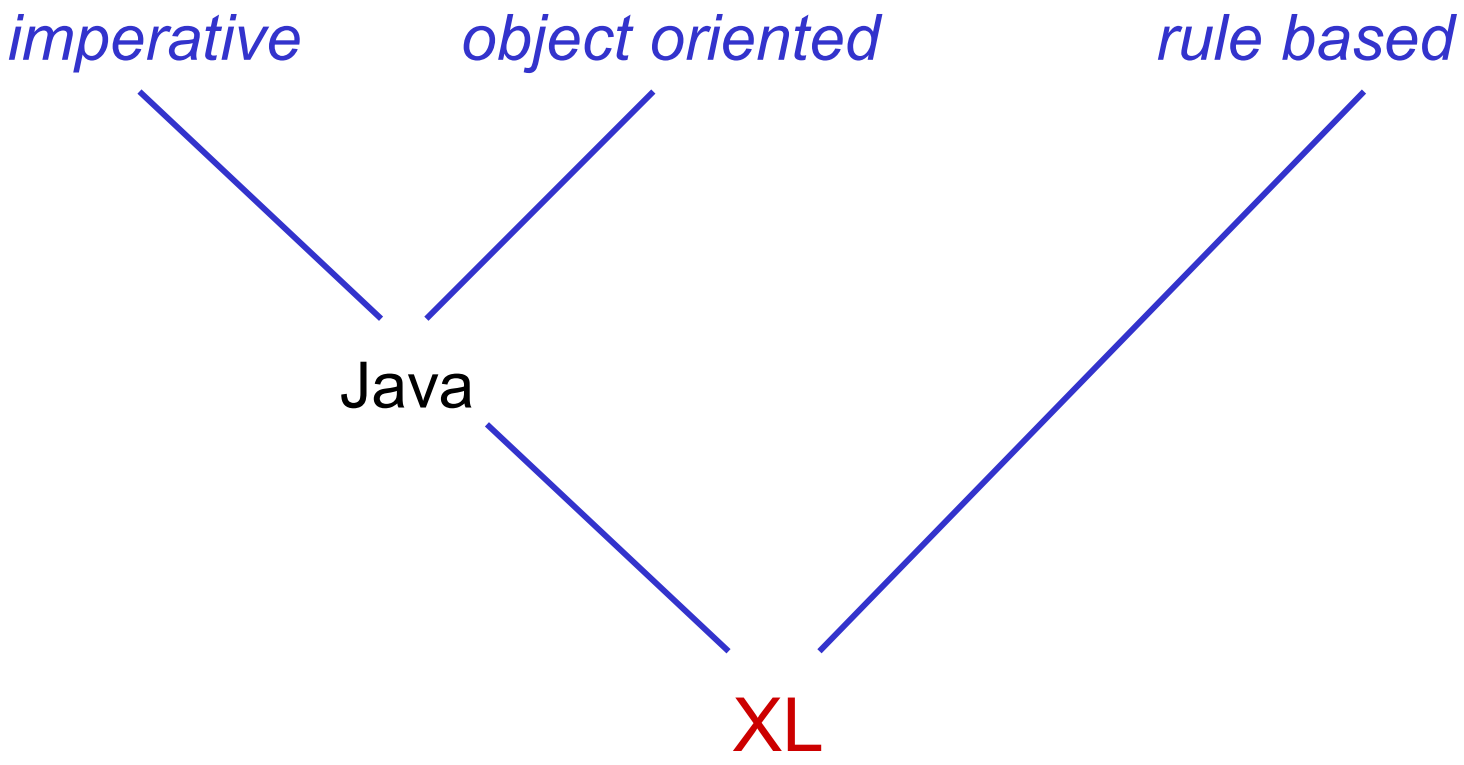
Symbols which are not turtle commands are ignored by the turtle.

$\rightarrow$ connection with imperative paradigm

XL: a synthesis of three paradigms

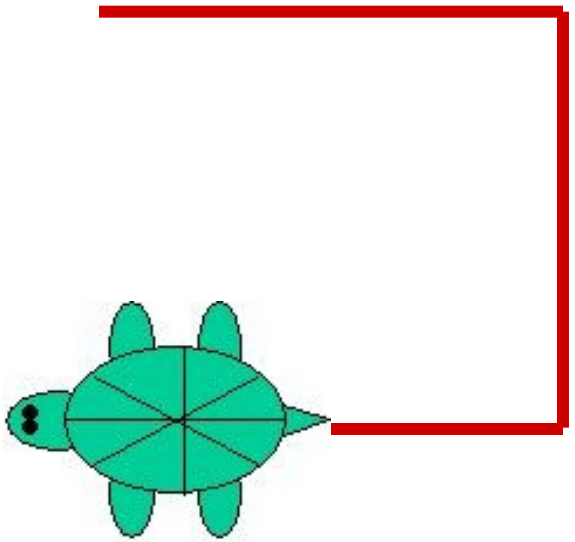
„eXtended L-system language“

programming language which makes parallel graph-grammars (RGG) accessible in a simple way



turtle geometry

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



Turtle geometry

„turtle": virtual device for drawing or construction in 2-D or 3-D space

- able to store information (graphical and non-graphical)

- equipped with a memory containing state information (important for branch construction)

- current turtle state contains e.g. current line thickness, step length, colour, further properties of the object which is constructed next

Turtle commands in XL (selection):

`F0` "Forward", with construction of an element (line segment, shoot, internode...), uses as length the current step size

(the zero stands for „no explicit specification of length")

`M0` forward without construction (*Move*)

`L(x)` change current step size (length) to *x*

`LAdd(x)` increment the current step size to *x*

`LMul(x)` multiply the current step size by *x*

`D(x)`, **`DAdd(x)`**, **`DMul(x)`** analogously for current thickness

Repetition of substrings possible with "**for**"

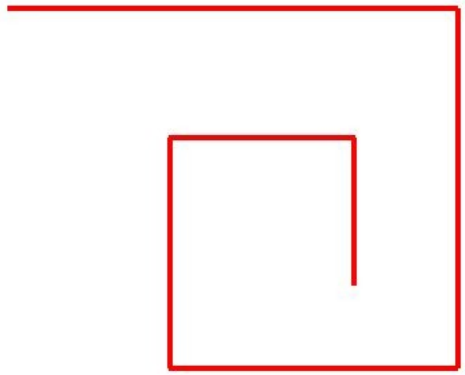
e.g., **for ((1:3)) (A B C)**

yields **A B C A B C A B C**

what is the result of the interpretation of

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

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

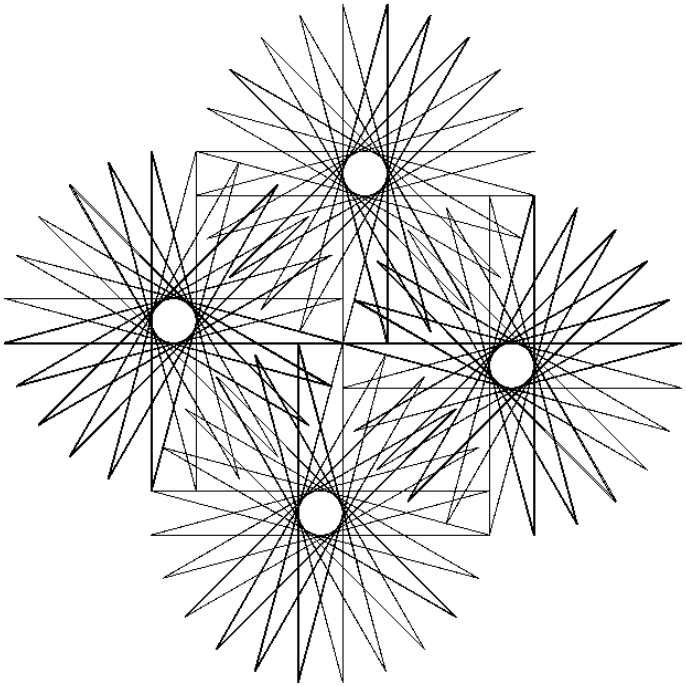


further example:

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

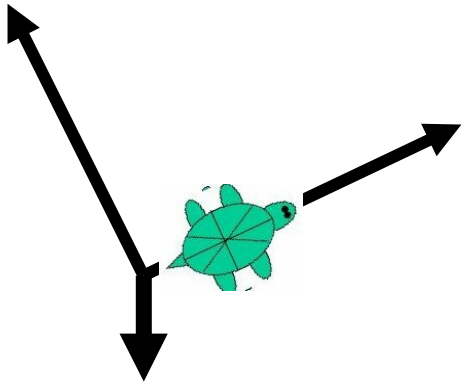
further example:

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



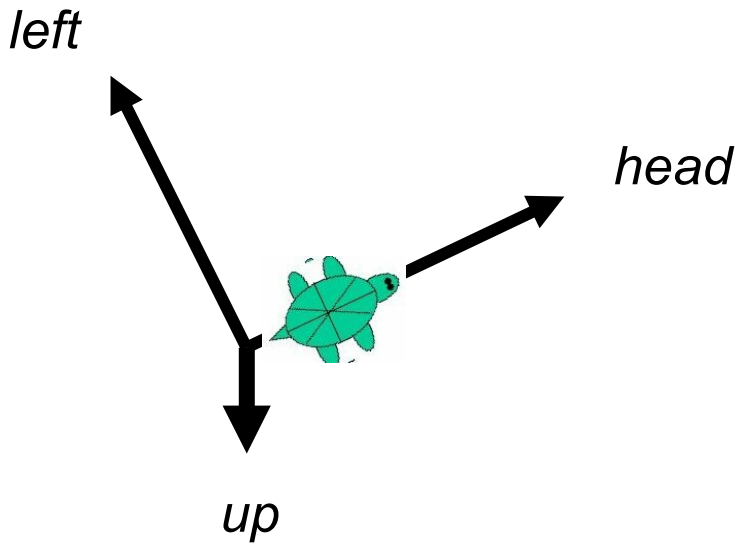
Extension to 3-D graphics:

turtle rotations by 3 axes in space



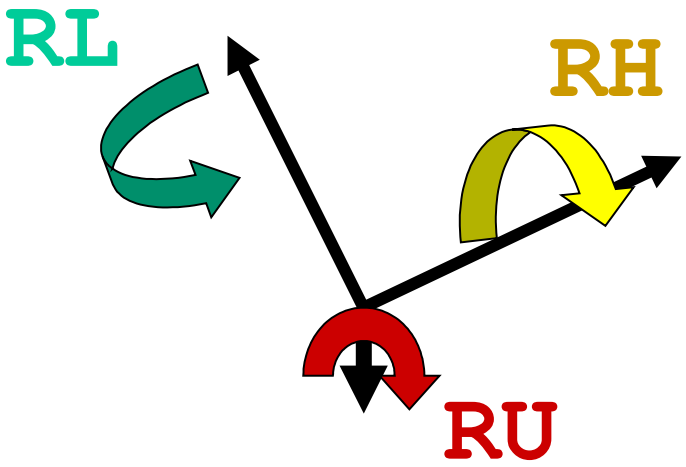
Extension to 3-D graphics:

turtle rotations by 3 axes in space



Extension to 3-D graphics:

turtle rotations by 3 axes in space



3-D commands:

`RU(45)` rotation of the *turtle* around the "up" axis by 45°

`RL(...)`, **`RH(...)`** analogously by "left" and "head" axis

up-, *left*- and *head* axis form an orthogonal spatial coordinate system which is carried by the *turtle*

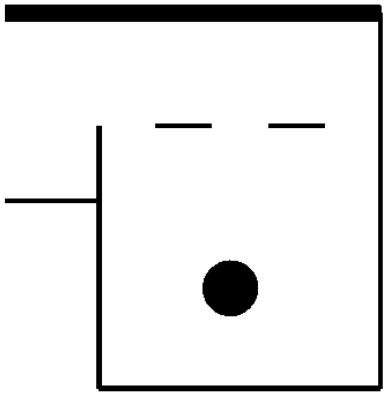
`RV(x)` rotation "to the ground" with strength given by **`x`**

`RG` rotation absolutely to the ground (direction (0, 0, -1))

Example:

```
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)
```

generates

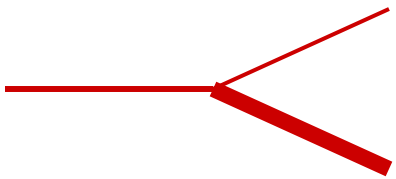


Branches:
realization with memory commands

`[` put current state on stack
("Ablage", Stack)

`]` take current state from stack
and let it become the current state
(thus: end of branch!)

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

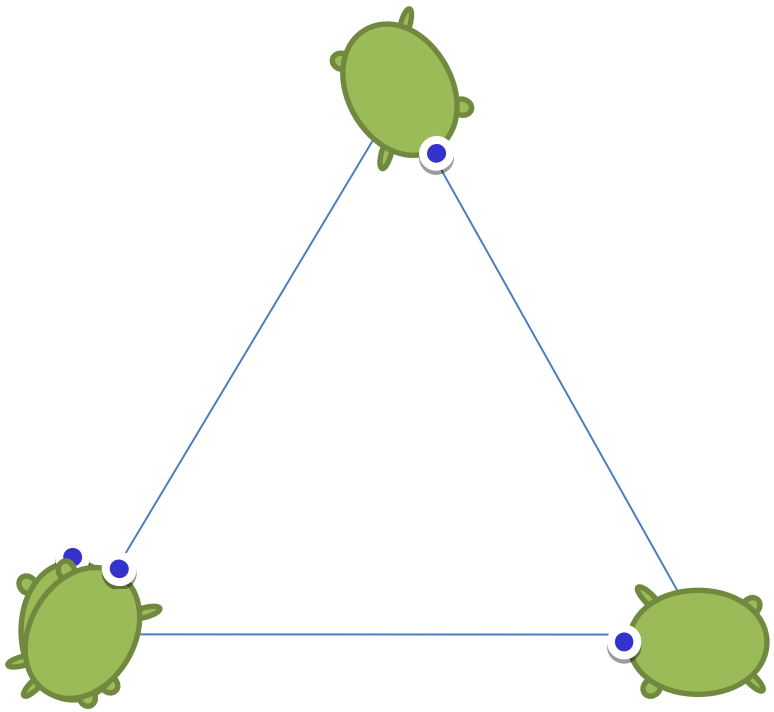


How to execute a turtle command sequence with GroIMP

write into a GroIMP project file (or into a file with filename extension **`.rgg`**):

```
protected void init()
    [
    Axiom ==> turtle command sequence ;
    ]
```

Example: Drawing a triangle



```
protected void init()
    [ Axiom ==> RU(30) F(10) RU(120) F(10) RU(120) F(10) ]
```

see file `sm09_e01.rgg`