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

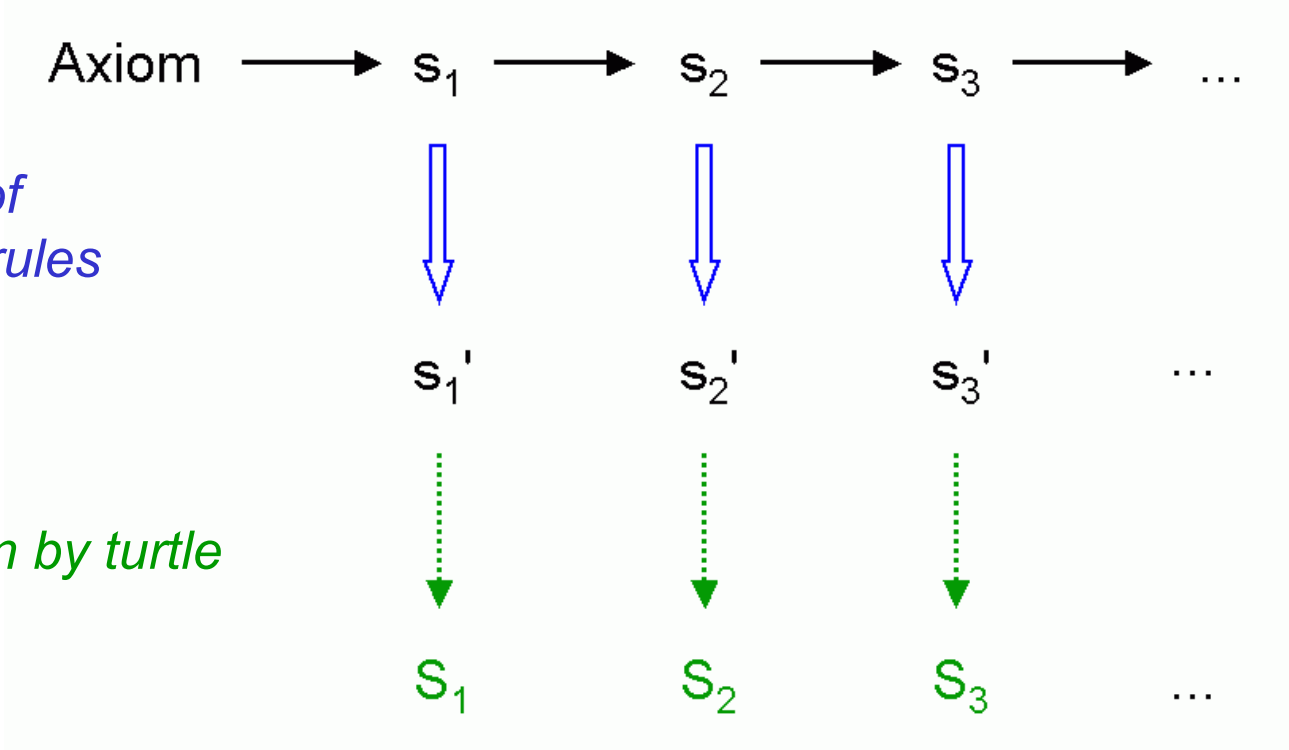
University of Göttingen, 27 February, 2012

Winfried Kurth

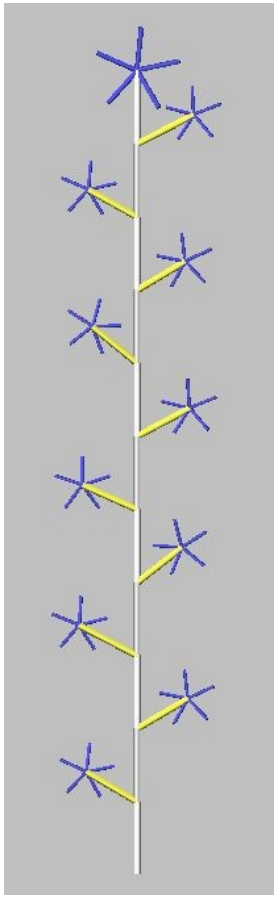
Interpretive rules and instantiation rules

Interpretive rules

insertion of a further phase of rule application directly preceding graphical interpretation (without effect on the next generation)



Example:



```
module MainBud(int x) extends Sphere(3) {{setShader(GREEN);}};
module LBud extends Sphere(2) {{setShader(RED);}};
module LMeris;
module AMeris;
module Flower;

const float d = 30;
const float crit_dist = 21;

protected void init()
   [ Axiom ==> MainBud(10); ]

public void run()
{  [
   MainBud(x) ==> F(20, 2, 15) if (x > 0)
                                 ( RH(180) [ LMeris ] MainBud(x-1) )
                               else
                                 ( AMeris );
   LMeris ==> RU(random(50, 70)) F(random(15, 18), 2, 14) LBud;

   LBud ==> RL(90) [ Flower ];

   AMeris ==> Scale(1.5) RL(90) Flower;          /* Flower: here only a symbol  */
   ]
   applyInterpretation();       /* call of the execution of interpretive rules
                                   (to be done in the imperative part { ... } !) */
}

protected void interpret()                      /* block with interpretive rules */
   [
   Flower ==> RH(30) for ((1:5))
                        ( RH(72) [ RL(80) F(8, 1, 9) ] );
   ]
```

further example:

```
public void run()
{
    [
    Axiom ==> A;
    A ==> Scale(0.3333) for (int i:(-1:1))
                          for (int j:(-1:1))
                            if ((i+1)*(j+1) != 1)
                              ( [ Translate(i, j, 0) A ] );
    ]
    applyInterpretation();
}

public void interpret()
    [
    A ==> Box;
    ]
```

generates the so-called „Menger sponge“ (a fractal)

```
public void run()
{  [
   Axiom ==> A;
   A ==> Scale(0.3333) for (int i:(-1:1))
                         for (int j:(-1:1))
                           if ((i+1)*(j+1) != 1)
                             ( [ Translate(i, j, 0) A ] );
   ]
   applyInterpretation();
}

public void interpret()
   [
   A ==> Box;
   ]
```

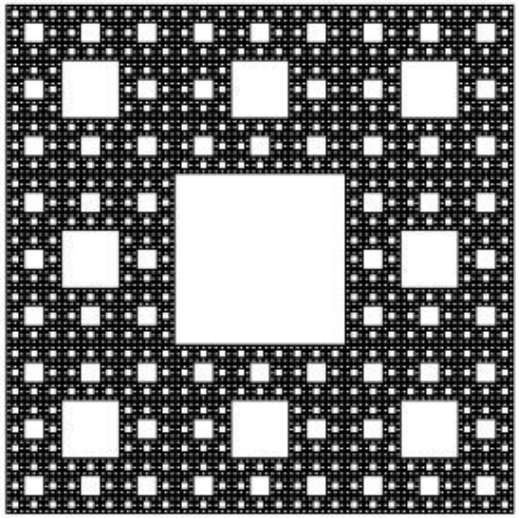
(a)

(b)

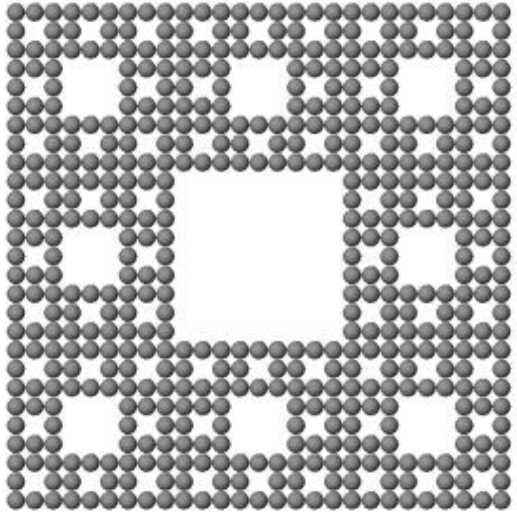
```
A ==> Sphere(0.5);
```

(c)

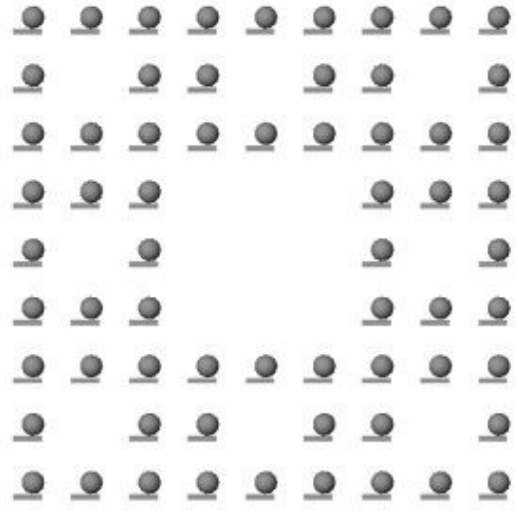
```
A ==> Box(0.1, 0.5, 0.1)
      Translate(0.1, 0.25, 0) Sphere(0.2);
```



(a)



(b)



(c)

what is generated by this example?

```
public void run()
{
    [
    Axiom ==> [ A(0, 0.5) D(0.7) F(60) ] A(0, 6) F(100);
    A(t, speed) ==> A(t+1, speed);
    ]
    applyInterpretation();
}

public void interpret()
    [
    A(t, speed) ==> RU(speed*t);
    ]
```

a very similar type of rules in XL:

instantiation rules

purpose: replacement of single modules by more complicated structures, only for visual representation
(similar as for interpretive rules)

- but: less data are stored (less usage of memory)
- in contrast to interpretive rules, no turtle commands with effect on other nodes can be used

further, arising possibility: „replicator nodes“ for copying and relocation of whole structures

instantiation rules: syntax

no new sort of rule arrow

specification of the instantiation rule directly in the declaration of the module which is to be replaced

```
module A ==> B C D;
```

replaces (instantializes) everywhere **`A`** by **`B C D`**

example

sm09_e43.rgg

```
const int multiply = EDGE_0;      /* user-defined edge type */

module Tree ==> F(20, 1)
    [ M(-8) RU(45) F(6, 0.8) Sphere(1) ]
    [ M(-5) RU(-45) F(4, 0.6) Sphere(1) ] Sphere(2);

module Replicator ==> [ getFirst(multiply) ] Translate(10, 0, 0)
                      [ getFirst(multiply) ];

public void run1()
[
Axiom ==> F(2, 6) P(10) Tree;
]

public void run2()
[
Axiom ==> F(2, 6) P(10) Replicator -multiply-> Tree;
]
```

Tree is instantiated with the red structure

inserts all what comes after the „multiply“ edge