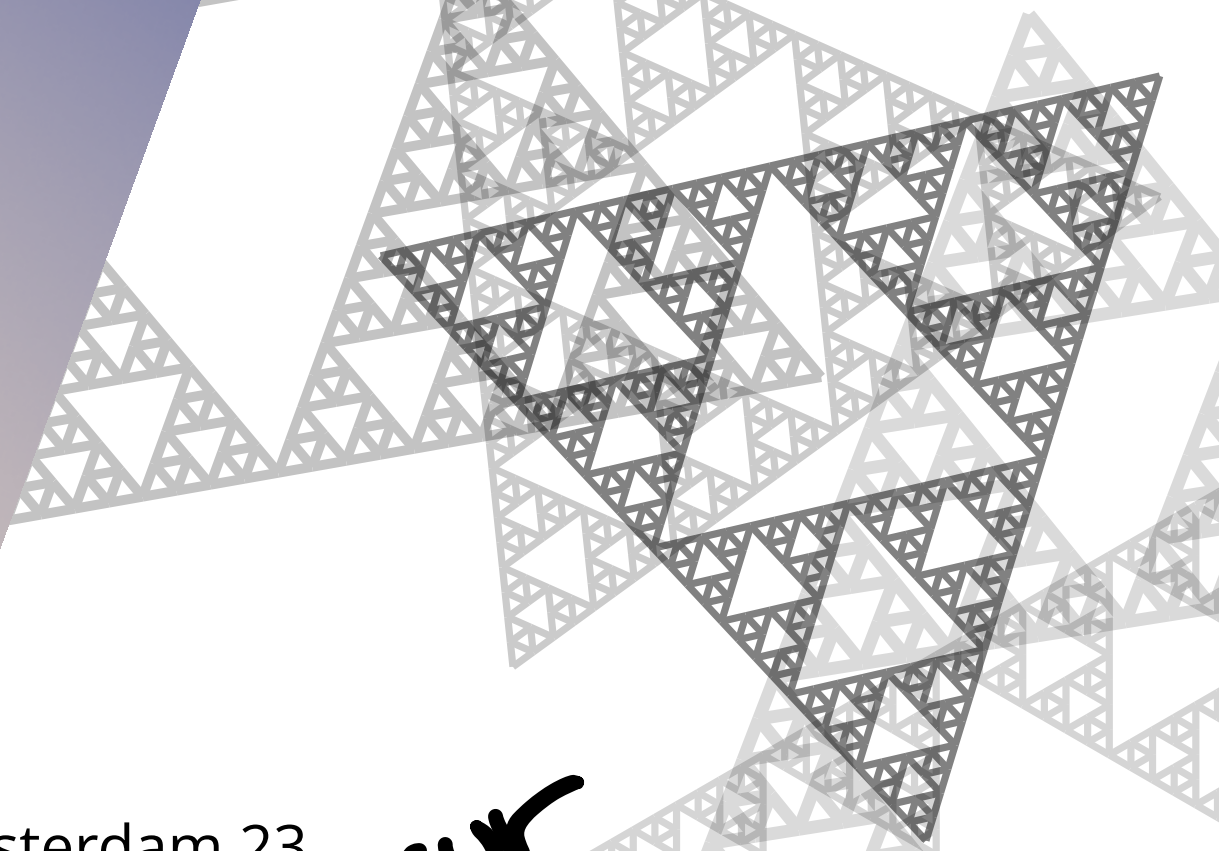


Declarative Amsterdam 23

Talking about ~~my~~ our Generations

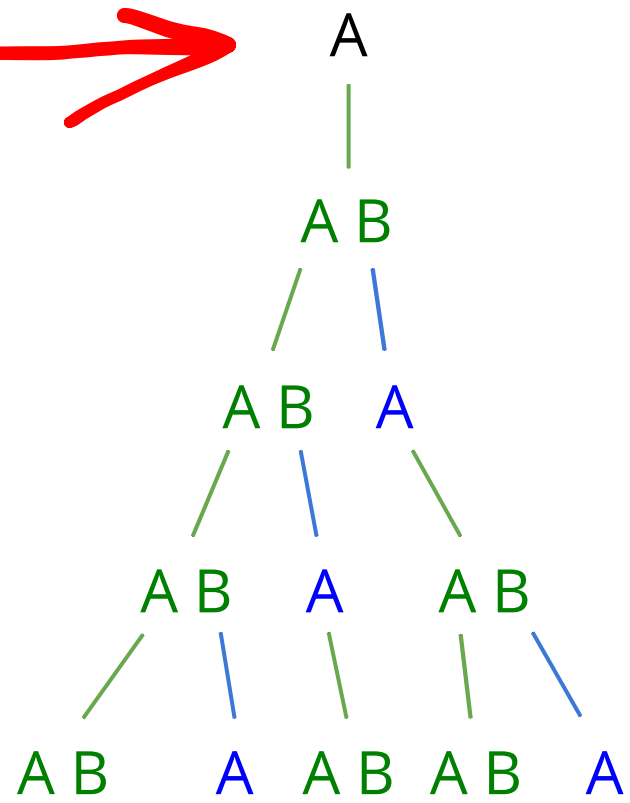

Lindenmayer

start with the symbol A

per iteration

replace any A with A and B

replace any B with A



1

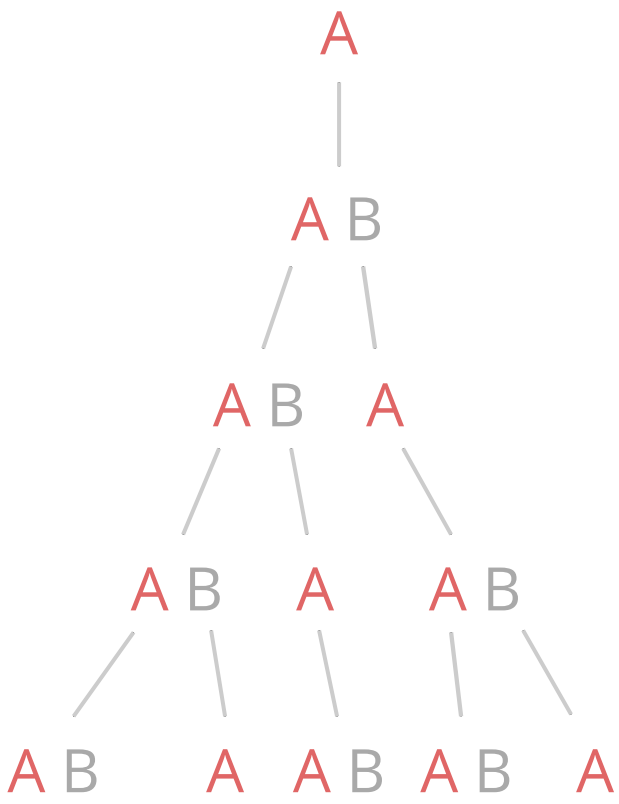
1

2

3

5

?




Systems

Alphabet	`A` `B`
Axiom	`A`
Production Rules	`A` → `A` `B` `B` → `A`

XML Representation In Linsy

```
<system>
  <axiom>A</axiom>
  <grammar>
    <variable match="A">AB</variable>
    <variable match="B">A</variable>
  </grammar>
</system>
```

Binary Tree

```
axiom: 0
alphabet: 0 1 [ ]
rules:
  0 → 1[0]0
  1 → 11
  [ → [
  ] → ]
```

```
0

1[0]0

11[1[0]0]1[0]0

1111111111111111111111111111111
[1111111111111111[11111111[1111[
11[1[0]0]1[0]0]11[1[0]0]1[0]0]11
11[11[1[0]0]1[0]0]11[1[0]0]1[0]0
]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0
]11[1[0]0]1[0]0]1111111111111111
[11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0
]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111
[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```

XML Representation In Linsy

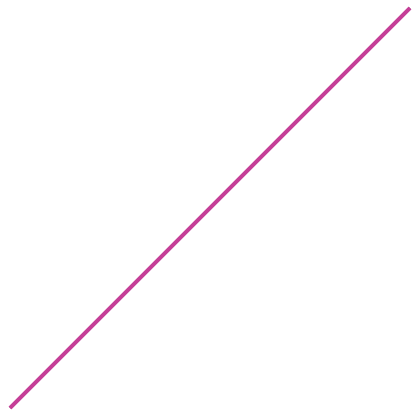
```
<system>
  <axiom>0</axiom>
  <grammar>
    <variable match="0">1[0]0</variable>
    <variable match="1">11</variable>
    <terminal match="[" />
    <terminal match="]" />
  </grammar>
</system>
```

Production Rules

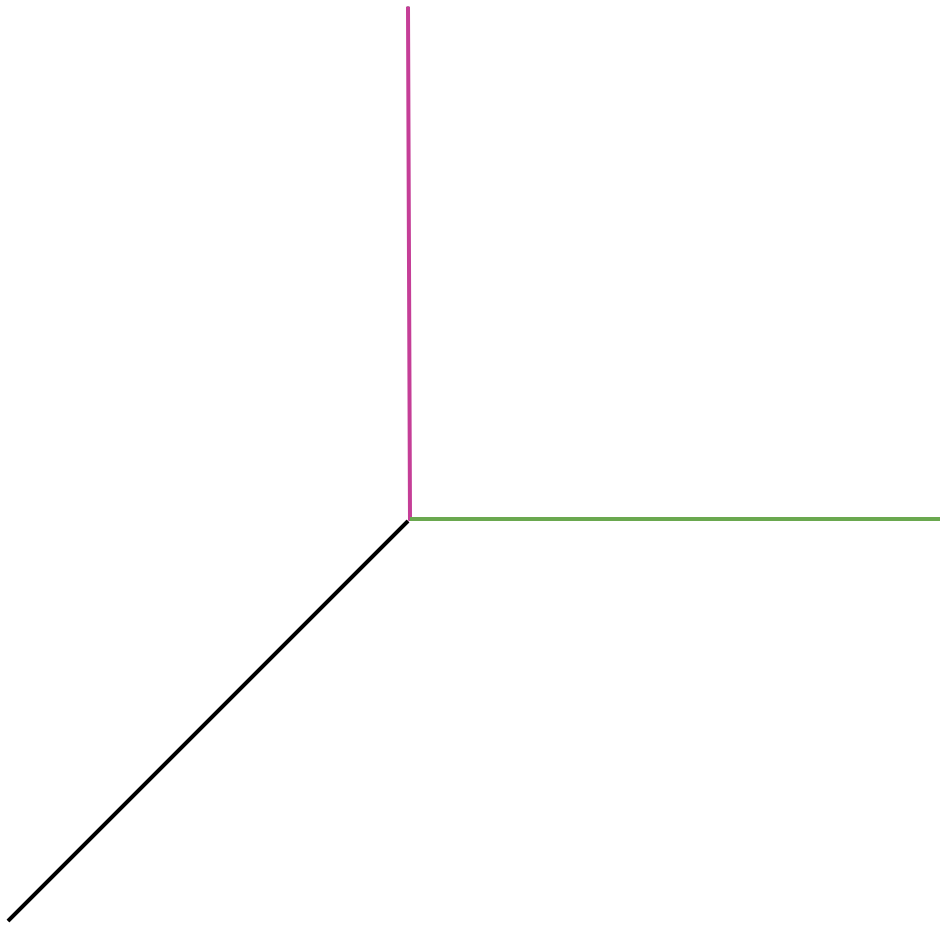
Binary Tree Production Rules

```
1 = draw line forward
0 = draw line forward
[ = push state and turn right (branch)
] = pop state and turn left (end branch)
```

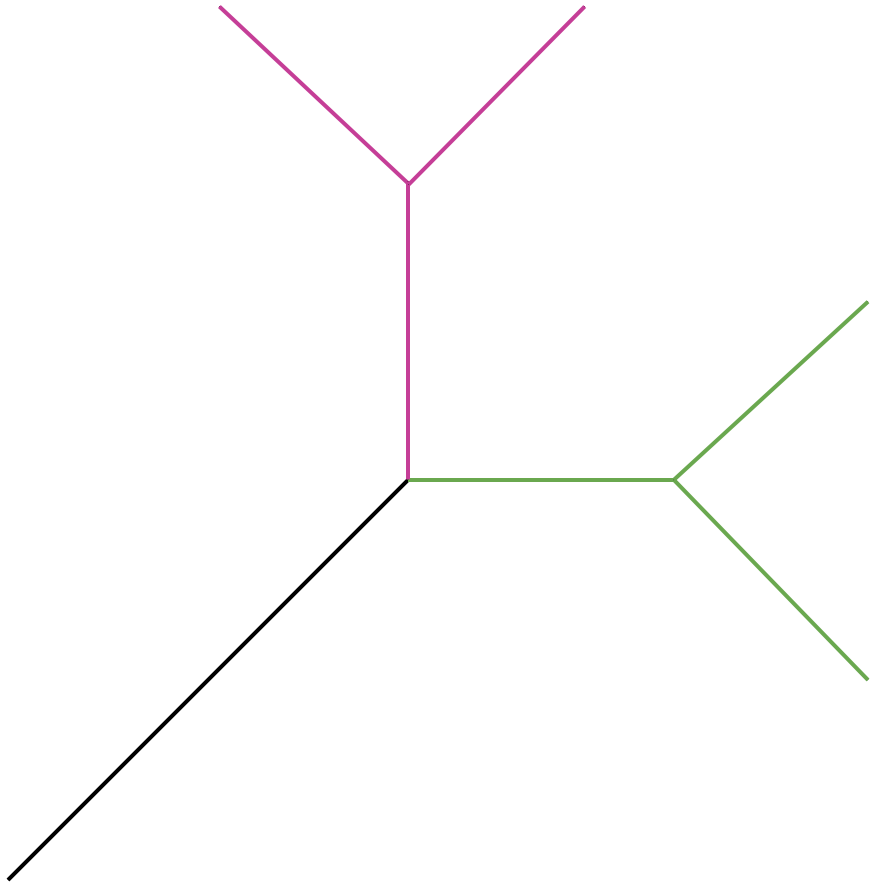
0



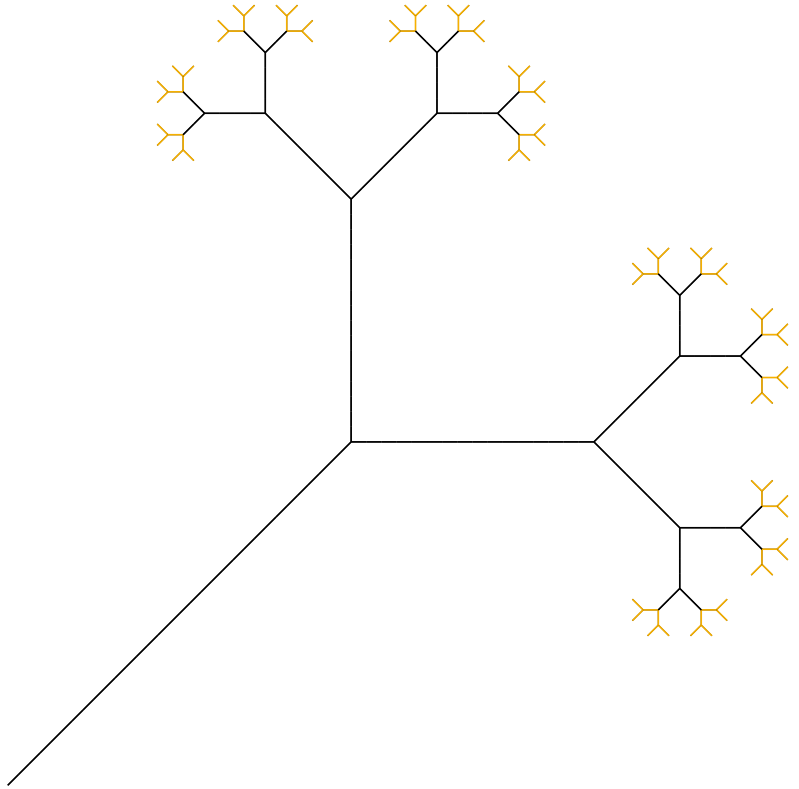
1[0]0



11[1[0]0]1[0]0



```
1111111111111111111111111111111111
[11111111111111111[11111111[1111[
11[1[0]0]1[0]0]11[1[0]0]1[0]0]11
11[11[1[0]0]1[0]0]11[1[0]0]1[0]0
]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0
]11[1[0]0]1[0]0]1111111111111111
[11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0
]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111
[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```



Render

```
<render viewBox="-120 0 240 240">
    <state x="0" y="240" orientation="90"
           angle="45" velocity="16"  />
</render>
```

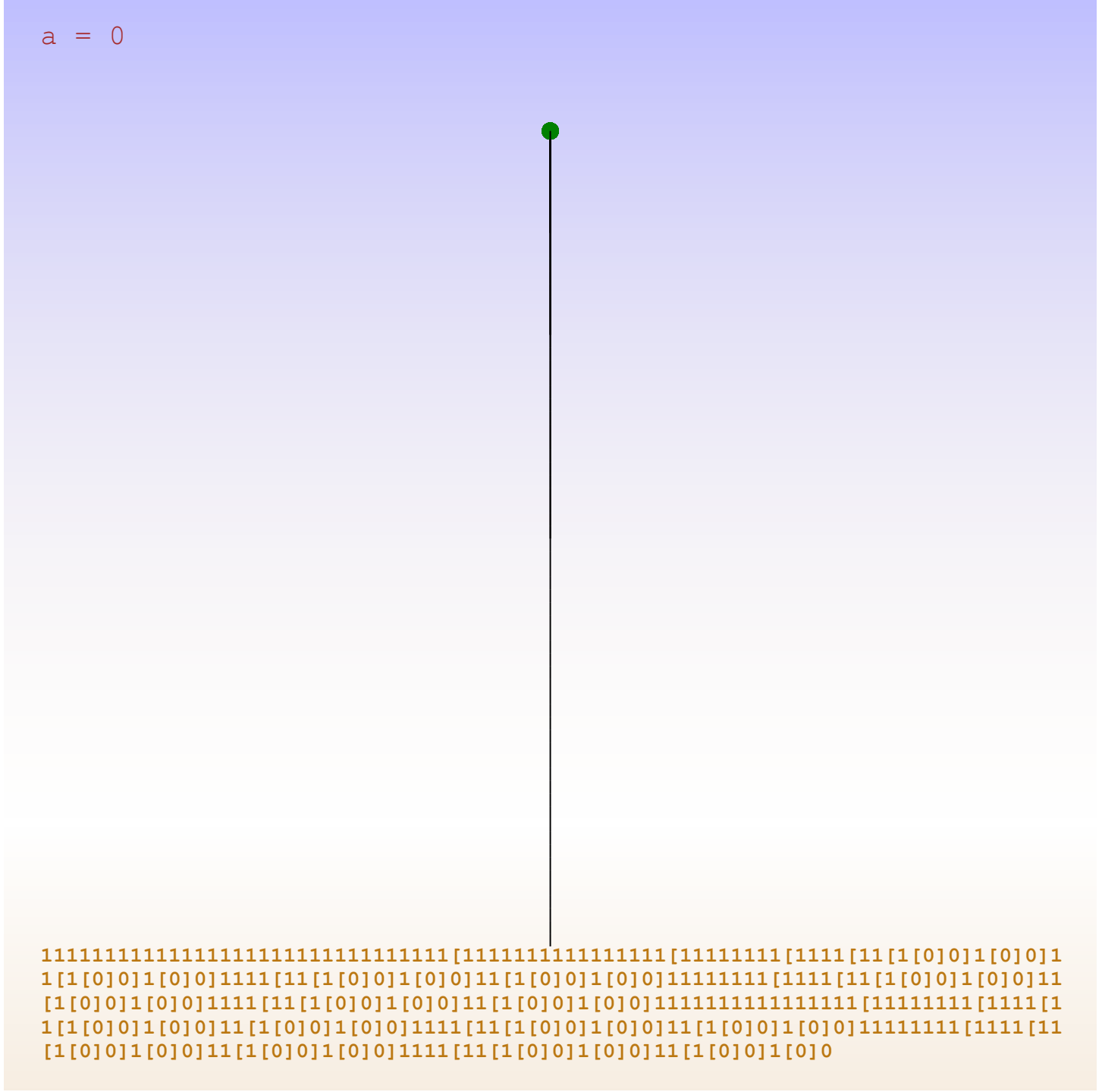
Inkscape

```
A-F = draw line forward
G-L = move forward
+   = turn right
-   = turn left
|   = turn around
[   = push state (branch)
]   = pop state (end branch)
```

```
<system>
  <axiom>B</axiom>
  <grammar>
    <variable match="A">AA</symbol>
    <variable match="B">A[+B]-B</symbol>
    <terminal match="+" />
    <terminal match="-" />
    <terminal match="[" />
    <terminal match="]" />
  </grammar>
  <render />
</system>
```

```
<system iterations="3">
  <axiom>B</axiom>
  <grammar>
    <variable match="A">AA</symbol>
    <variable match="B">A[+B]-B</symbol>
    <terminal match="+" />
    <terminal match="-" />
    <terminal match="[" />
    <terminal match="]" />
  </grammar>
  <render viewBox="-120 0 240 240">
    <state x="0" y="240" angle="45" velocity="16" orientation="90" />
  </render>
</system>
```

Angle


a = 0
1111111111111111111111111111111[1111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0

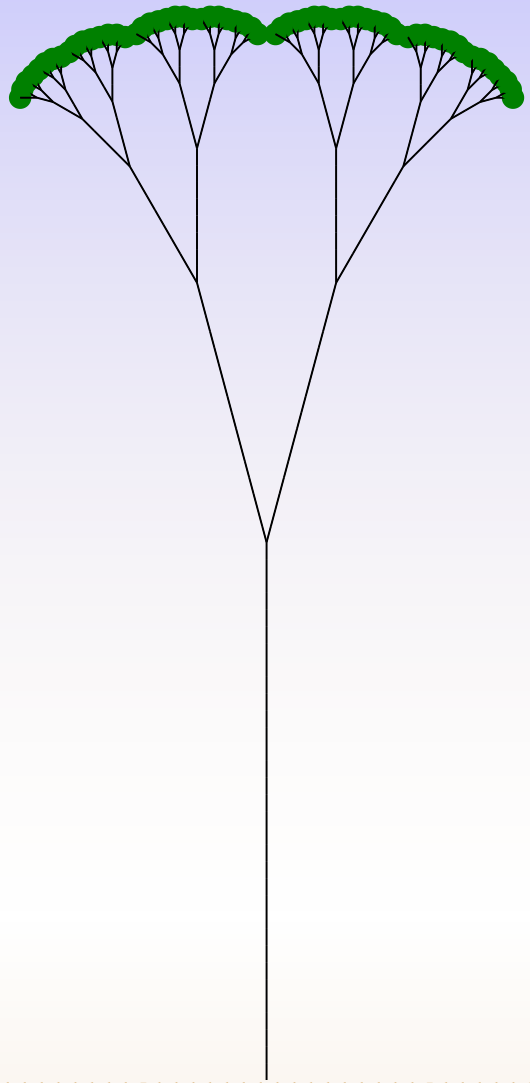
Angle

a = 3

```
11111111111111111111111111111111[11111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```

Angle

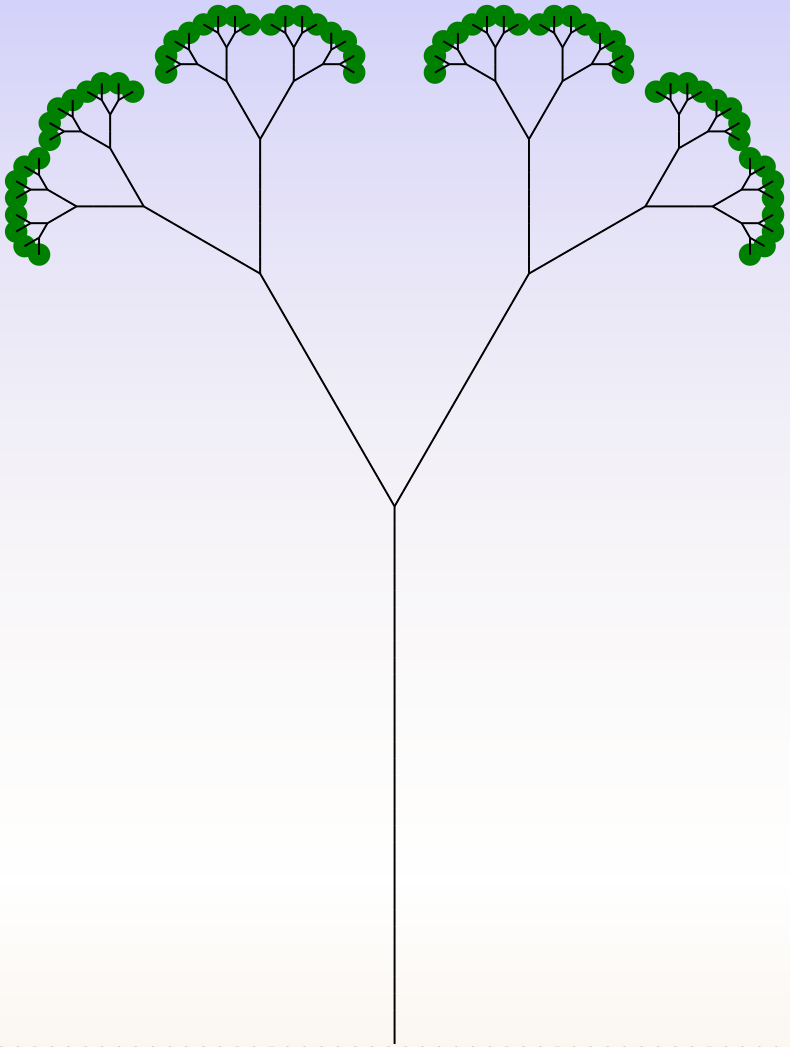
a = 15



```
11111111111111111111111111111111[11111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```

Angle

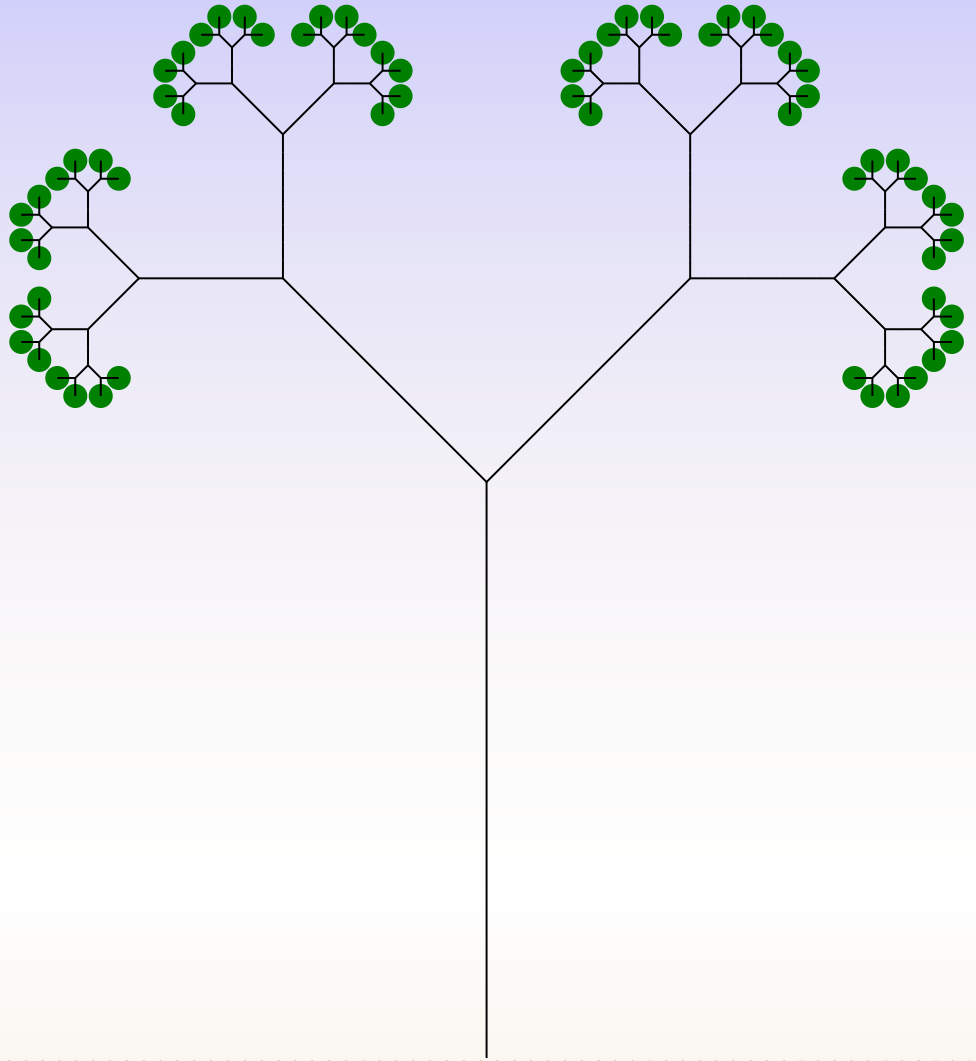
a = 30



```
11111111111111111111111111111111[11111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```

Angle

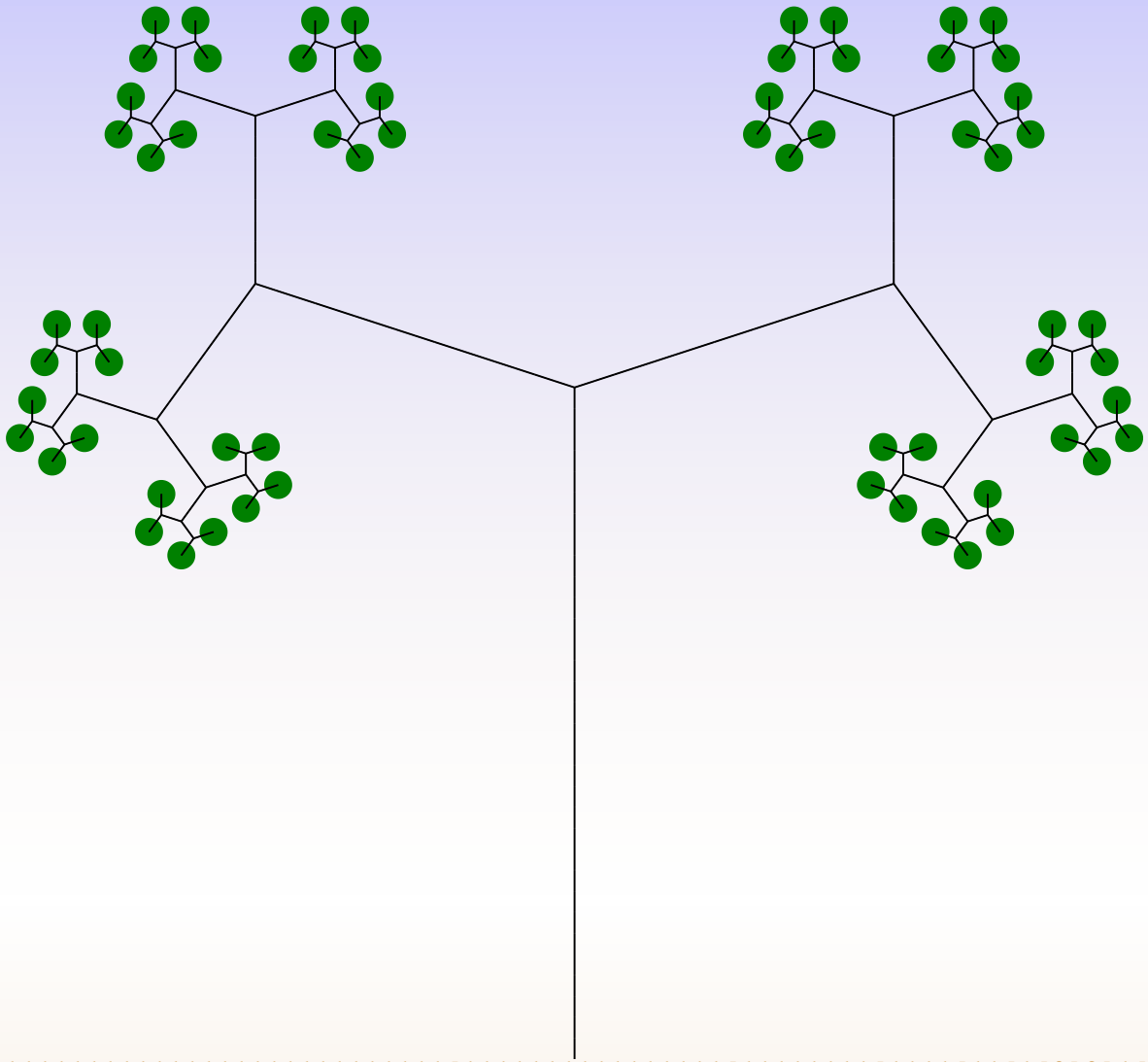
a = 45



```
11111111111111111111111111111111[11111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```

Angle

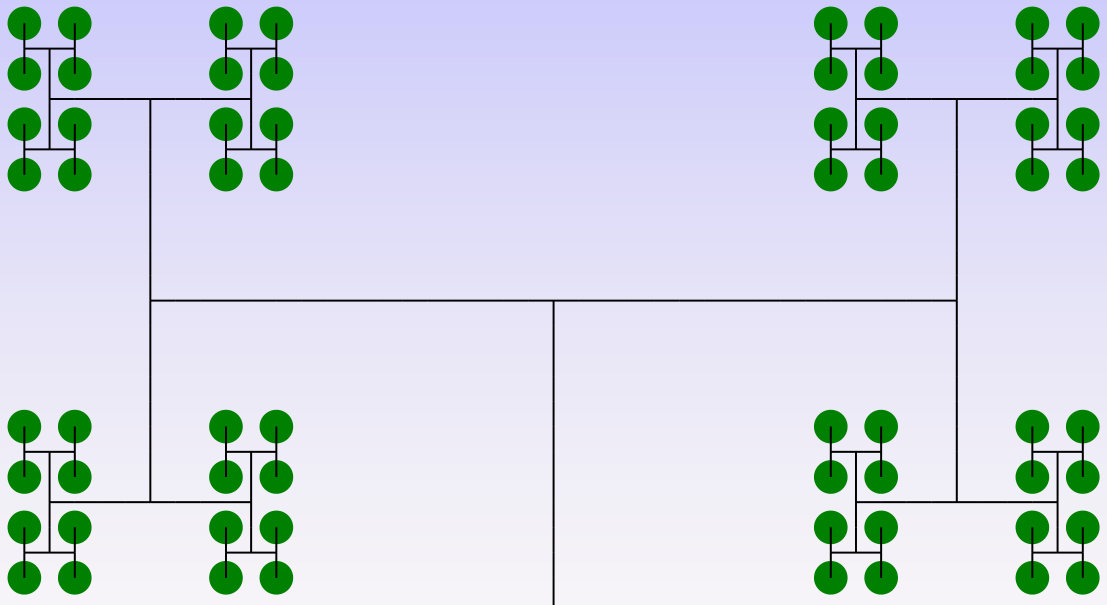
a = 72



```
1111111111111111111111111111111[1111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```

Angle

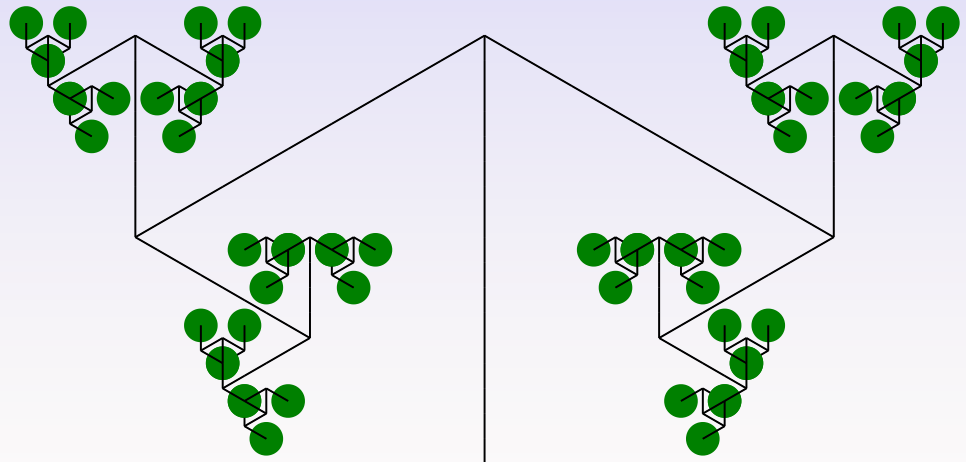
a = 90



```
1111111111111111111111111111111[1111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```

Angle

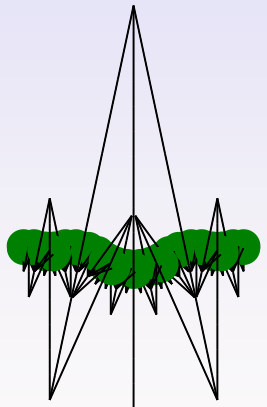
a = 120



```
1111111111111111111111111111111[1111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```

Angle

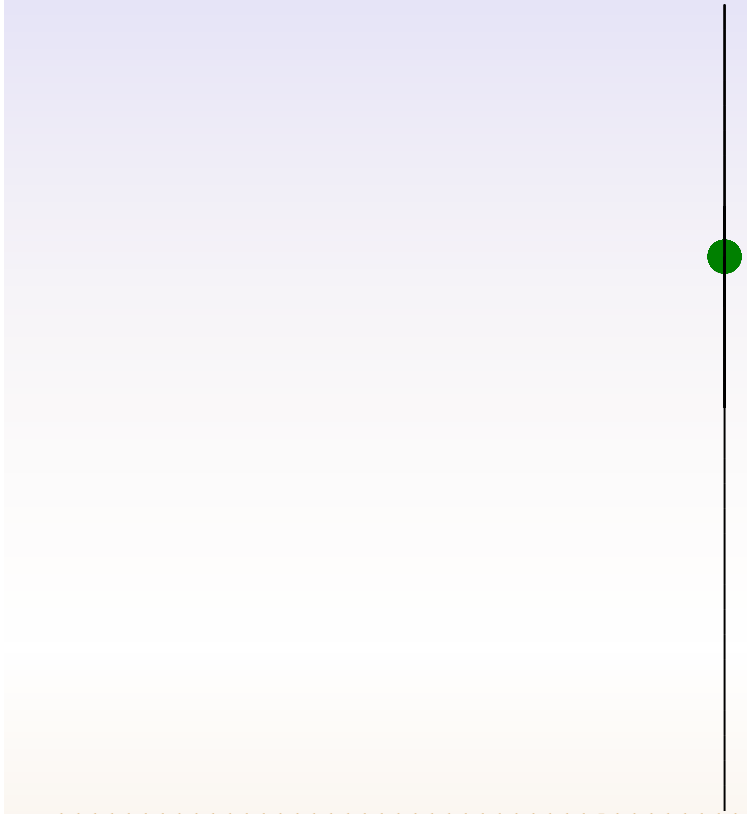
a = 168



```
1111111111111111111111111111111[1111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
```

Angle

a = 180

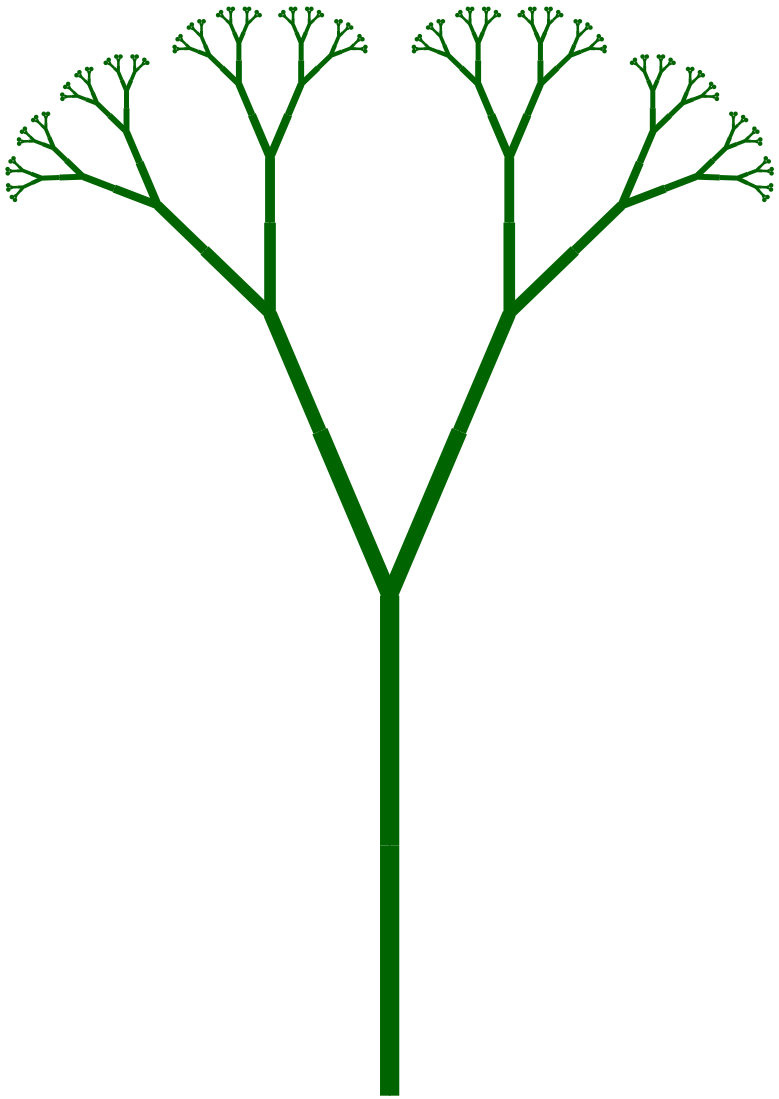


1111111111111111111111111111111[1111111111111111[11111111[1111[11[1[0]0]1[0]0]1
1[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11[1[0]0]1[0]0]11
[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]1111111111111111[11111111[1111[1
1[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0]11111111[1111[11
[1[0]0]1[0]0]11[1[0]0]1[0]0]1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0

paulbourke.net/fractals/lsys/

#	Increment the line width (by line width increment)
!	Decrement the line width (by line width increment)
@	Draw a dot (with line width radius)
{	Open a polygon
}	Close a polygon (and fill it with fill colour)
>	Multiply the line length (by the line length scale factor)
<	Divide the line length (by the line length scale factor)
&	Swap the meaning of + and -
(	Decrement turning angle (by turning angle increment)
)	Increment turning angle (by turning angle increment)

```
axiom: A@
alphabet: A @ - + [ ] < >
rules:
    A → <A>
    @ → <A>[+A>@]-A>@
```



```
<system iterations="7">
    <axiom>A@</axiom>
    <grammar>
        <variable symbol="A">&lt;A&gt;</variable>
        <variable symbol="@">&lt;A&gt;[+A&gt;@]-A&gt;@</variable>
        <terminal symbol="&lt;"/>
        <terminal symbol="&gt;"/>
        <terminal symbol="["/>
        <terminal symbol="]"/>
        <terminal symbol="+"/>
        <terminal symbol="-"/>
    </grammar>
    <render viewBox="0 0 800 800">
        <state x="400" y="800"
            velocity="16"
            angle="23" orientation="90"
            acceleration="1.4"
            color="darkgreen"
        />
    </render>
</system>
```

```
declare function render:symbol (
    $state as map(*),
    $next-symbol as xs:string
) as map(*) {
    switch($next-symbol)
        case "A" case "B" case "C" case "D" case "E" case "F"
            return render:line($state)
        case "G" case "H" case "I" case "J" case "K" case "L"
            return render:move($state)
        case "@" return render:circle($state)
        case "-" return render:turn-left($state)
        case "+" return render:turn-right($state)
        case "[" return render:push-stack($state)
        case "]" return render:pop-stack($state)
        case "<" return render:increase-velocity($state)
        case ">" return render:decrease-velocity($state)
        default return error()
};
```

stochastic

```
axiom: X
angle: 25
alhabet: F X [ ] + -
rules:
  X → F+[[X]-X]-F[-FX]+X (75%)
  X → F-F[-FX]+[[X]-X]+X (25%)
  F → FF (95%)
  F → FFF (5%)
```

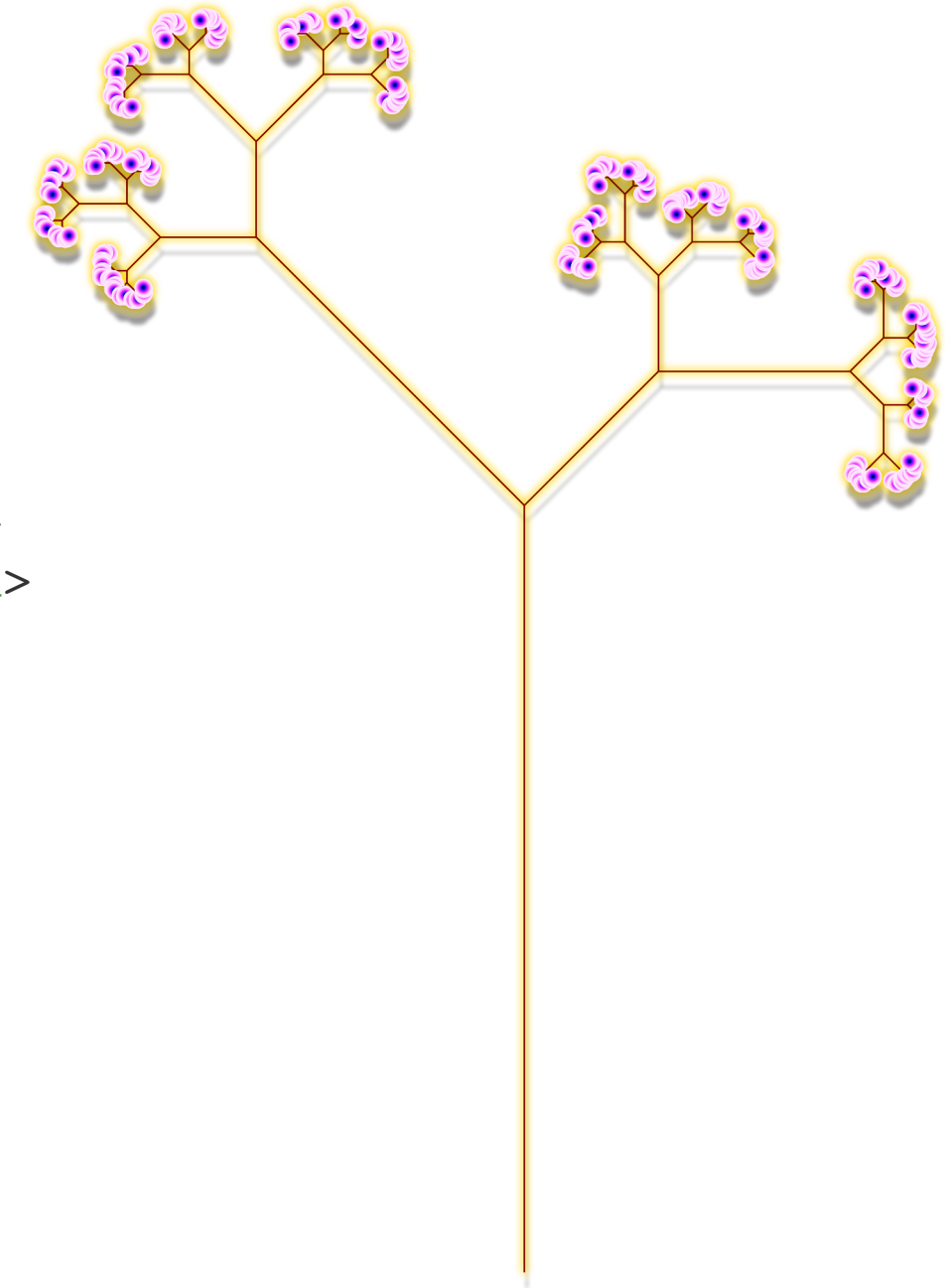
```
<system iterations="4">
  <axiom>0</axiom>
  <grammar type="stochastic">
    <variable match="1">11</variable>
    <variable match="0">
      <option>1[0]0</option>
      <option>11[0]0</option>
      <option>0</option>
    </variable>
    <terminal match="[" />
    <terminal match="]" />
  </grammar>
</system>
```

```
let $random := 0.3456789
```

```
1 [0.33333333333333333, ("1", "[", "0", "]", "0")],
2 [0.66666666666666667, ("1", "1", "[", "0", "]", "0")],
3 [1.0,                 "0"]
```

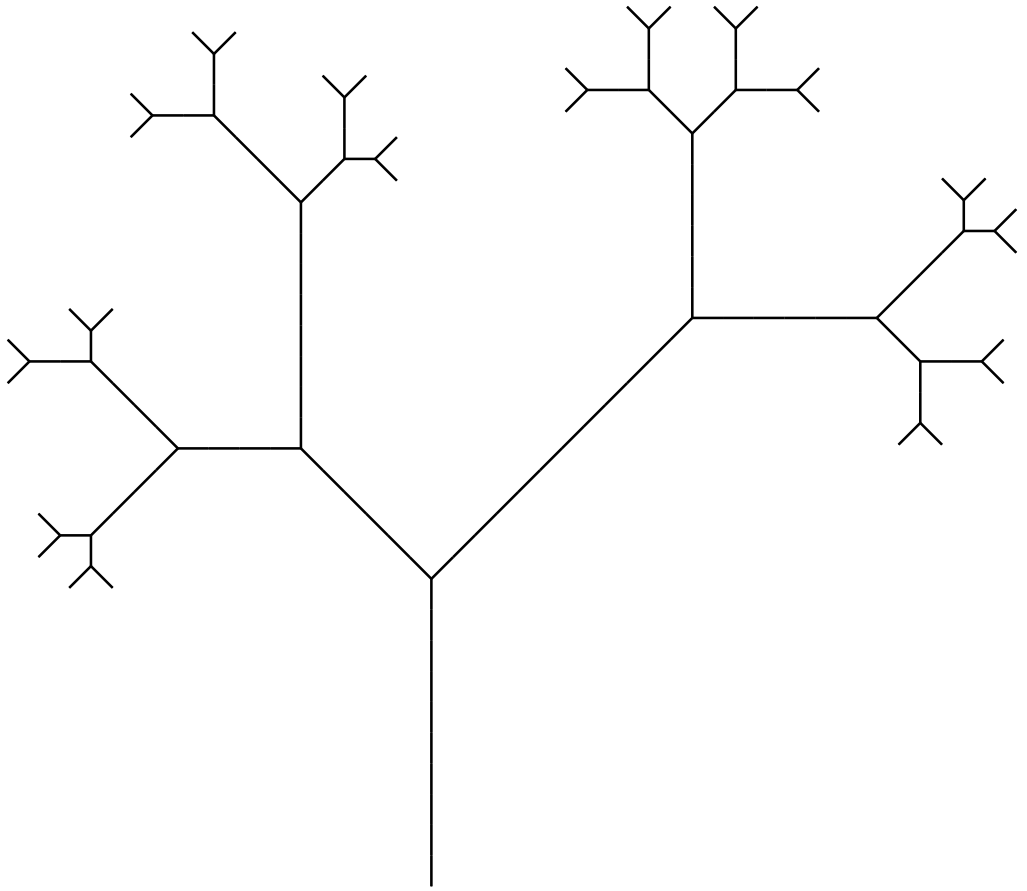
But weight...

```
<variable match="0">
  <option weight="10">1[0]0</option>
  <option weight="10">11[0]0</option>
  <option weight="1">0</option>
</variable>
```



Growing Seeds

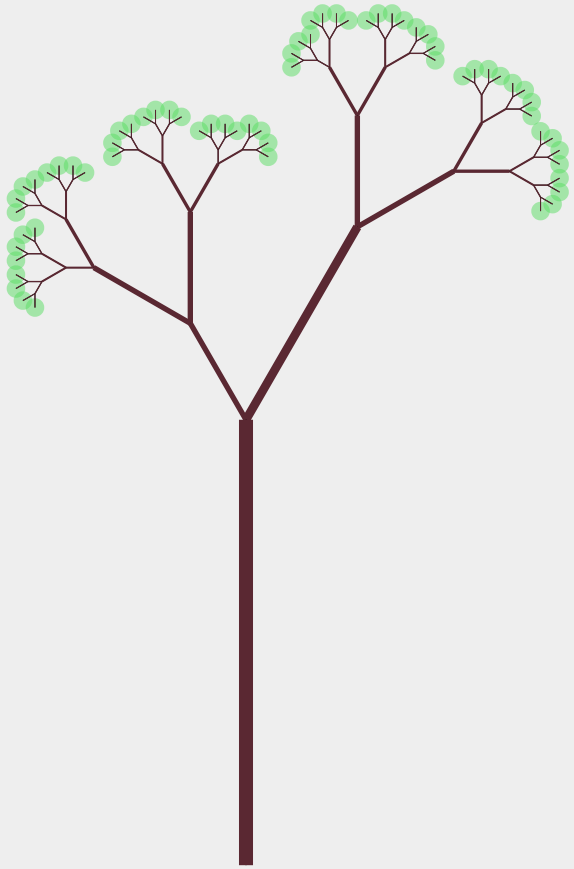
```
1 <system iterations="5" seed="23">
2     <axiom>A</axiom>
3     <grammar type="stochastic">
4       ...
5     </grammar>
6     <render ... />
7 </system>
```




Performance &
Stack Overflows

```
<grammar>
  <variable match="0">1[0]0</variable>

  <variable match="1">2</variable>
  <variable match="2">3</variable>
  <variable match="3">4</variable>
  ...
  <variable match="9">a</variable>
  <variable match="a">b</variable>
  <variable match="b">c</variable>
  <variable match="d">e</variable>
  <variable match="e">f</variable>

  <terminal match="["/>
  <terminal match="]"/>
</grammar>
```

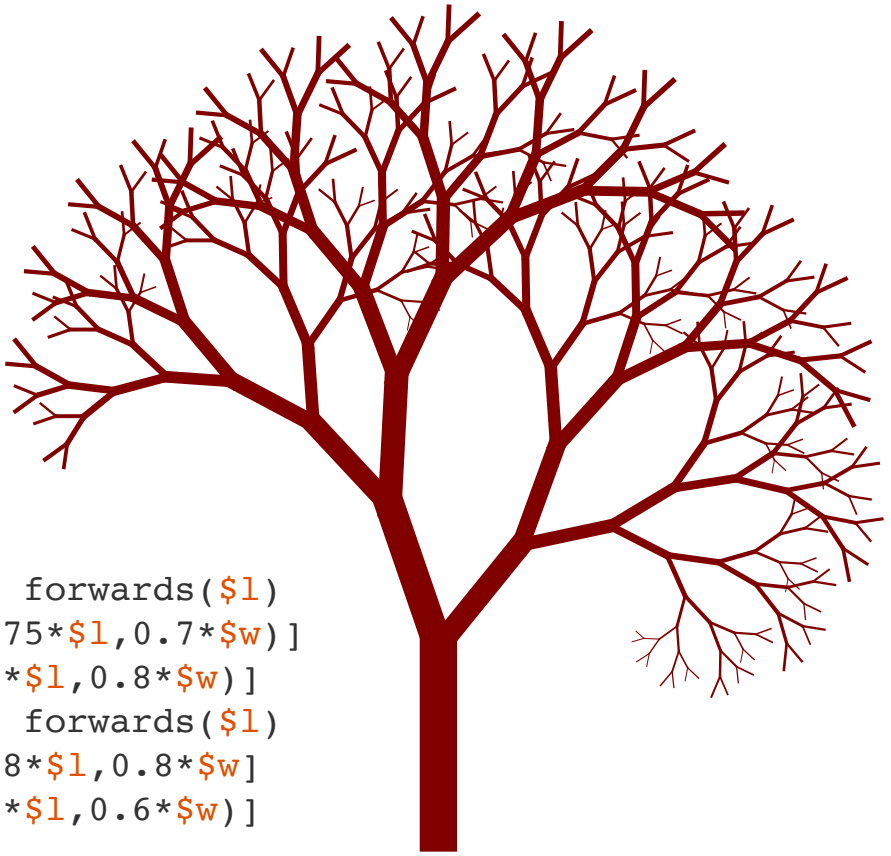
```
a = 30
```



```
6442210]0]10]0]210]0]10]0]3210]0]10]0]210]0]10]0]43210]0]10]0]210]0]10]0]3110]0]
10]0]210]0]10]0]543110]0]10]0]210]0]10]0]3210]0]10]0]210]0]10]0]43210]0]10]0]210
]0]10]0]3210]0]10]0]210]0]10]0
```

```
declare function local:draw ($state as map(*), $next-symbol as xs:string) {
    switch($next-symbol)
        case "0" return $state => local:line(0) => local:leaf()
        case "1" return $state => local:line(0)
        case "2" return $state => local:line(1)
        case "3" return $state => local:line(2)
        case "4" return $state => local:line(3)
                ...
        case "f" return $state => local:line(14)
        case "[" return $state => render:pop-stack() => render:turn-right()
        case "]" return $state => render:push-stack() => render:turn-left()
        default return error()
};
```

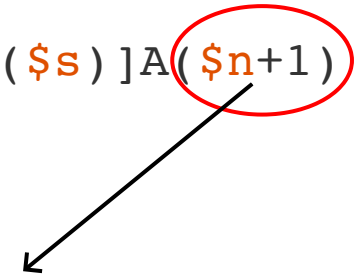
parametric



```
axiom: B(200,40)
A($l,$w) = width($w) forwards($l)
   [rotate(-24) B(0.75*$l,0.7*$w)]
   [rotate(22)A(0.85*$l,0.8*$w)]
B($l,$w) = width($w) forwards($l)
   [rotate(-19) A(0.8*$l,0.8*$w]
   [rotate(39) B(0.7*$l,0.6*$w)]
```

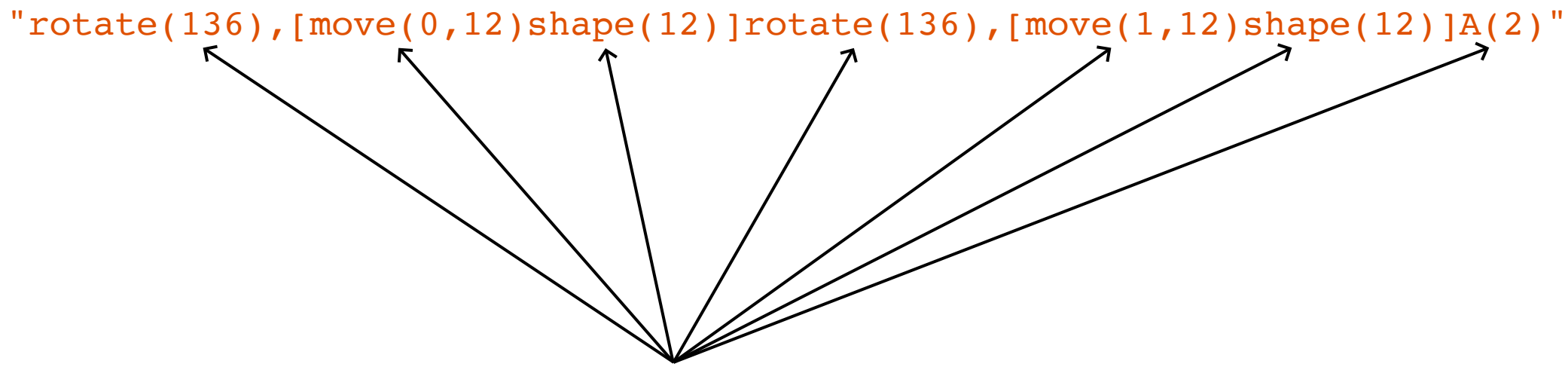
Evaluation in generation

```
A($n) = rotate($a),[move($n,$s)shape($s)]A($n+1)
```



```
"A(0)"
"rotate(136),[move(0,12)shape(12)]A(1)"
"rotate(136),[move(0,12)shape(12)]rotate(136),[move(1,12)shape(12)]A(2)"
```

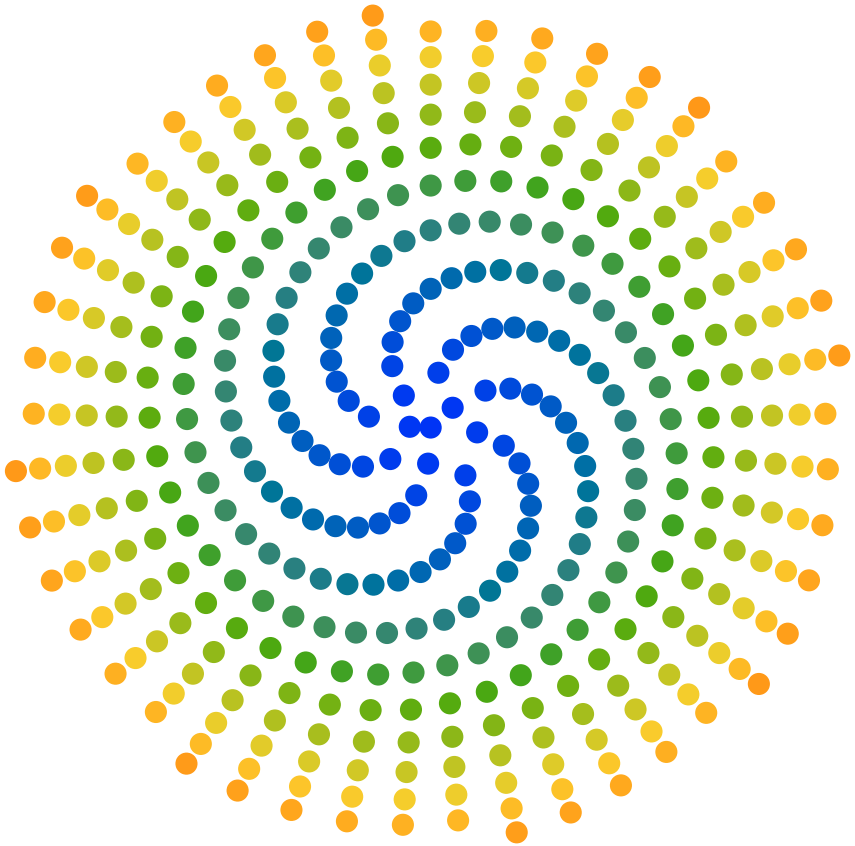
Evaluation in interpretation



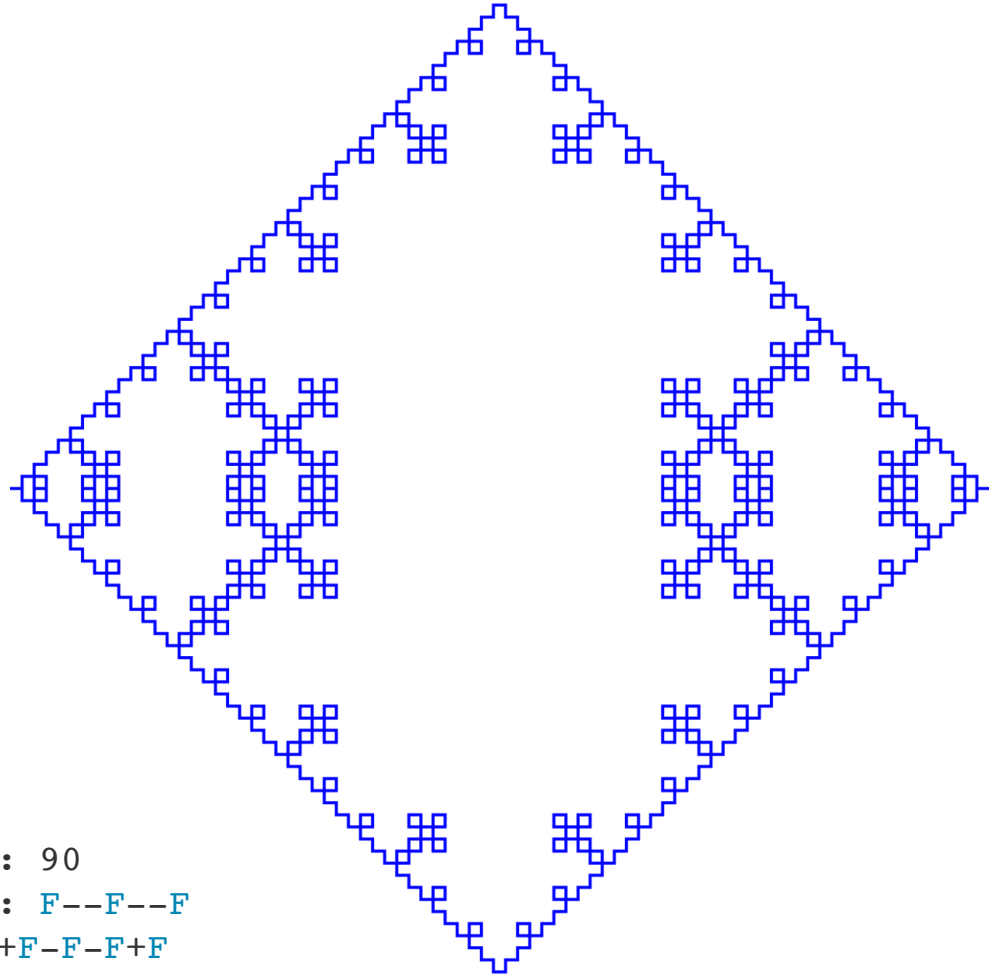
Interpretation functions for each functional form

Putting it all together

```
let $axiom := "A(0)"
let $rules := map {
  "A($n)": "rotate($a),[move($n,$s)shape($s)]A($n+1)"
}
let $parms := map {"a": 136, "s": 12}
let $interpreters := map {
  "move#2": function($stack, $n, $s) {
    $stack=>sys:move(2 * $s * math:sqrt($n))
   },
  "shape#1": function($stack, $s) {
    stack=>sys:circle($s, sys:current($stack)("cix"))
  }
}
```



invisible

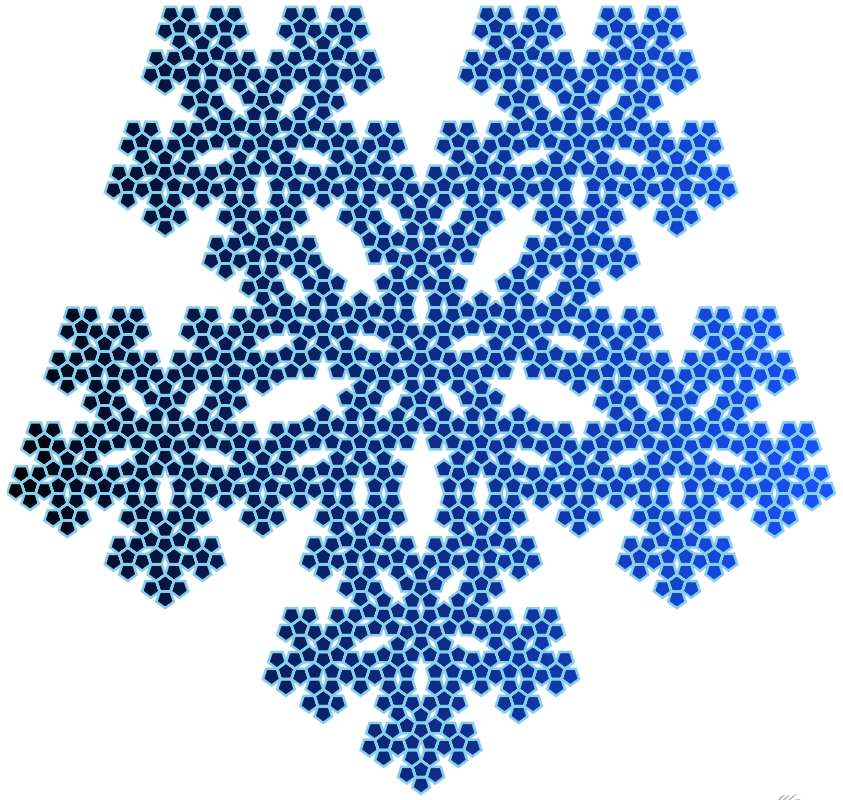


```
angle: 90
axiom: F--F--F
F = F+F-F-F+F
```

Parsing output

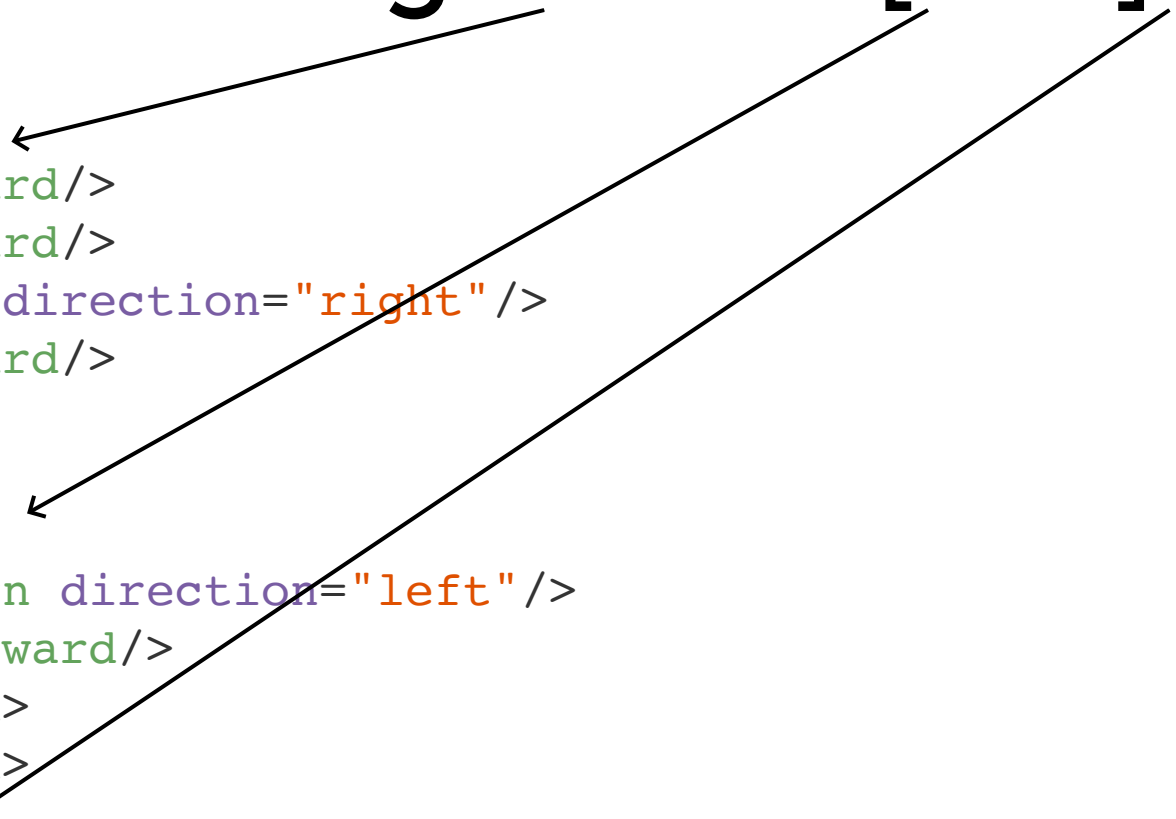
```
1  root = (path | branch)+ .
2  path = (forward | turn | -extra)+ .
3  branch = -"[", (path | branch)+, -"]" .
4  forward = -"F" .
5  turn = direction .
6  @direction = left | right | reverse .
7  left = -"-",+"left"  .
8  right = -"+",+"right"  .
9  reverse = -"|",+reverse" .
10 extra = -~["F";"[";"]";"+";"-"] .
```

CoffeeSacks choose-alternative:
greedy choice (longest leading match)



```
angle: 36
axiom: F++F++F++F++F
F = F++F++F|F-F++F
```

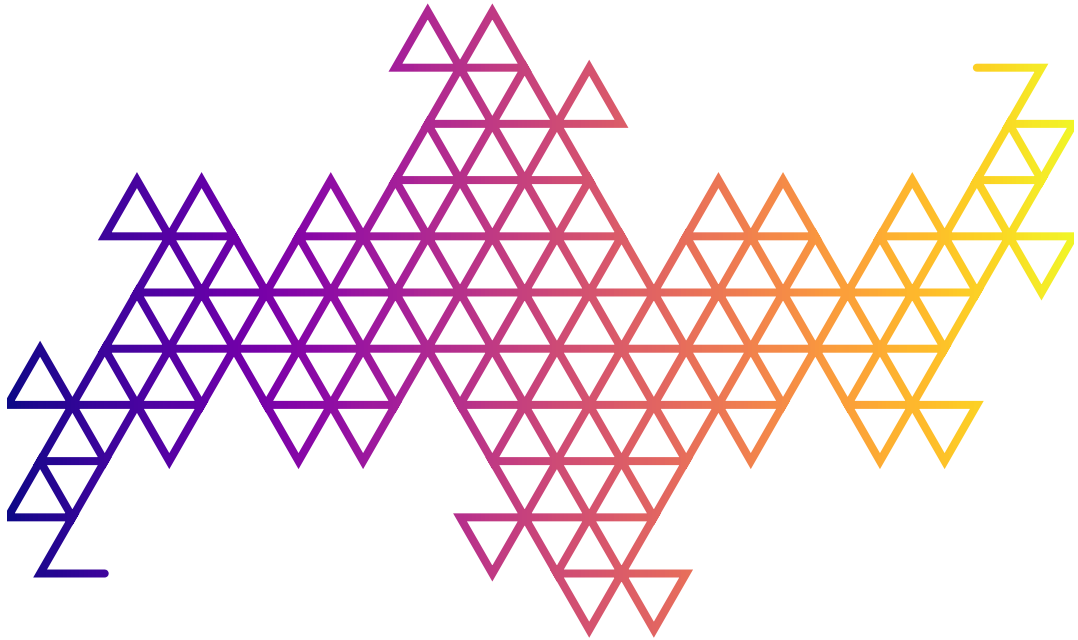
Parsing "FF+XF[X-F]+FF"



```
<root>
  <path>
    <forward/>
    <forward/>
    <turn direction="right"/>
    <forward/>
  </path>
  <branch>
    <path>
      <turn direction="left"/>
      <forward/>
    </path>
  </branch>
  <path>
    <turn direction="right"/>
    <forward/>
    <forward/>
  </path>
</root>
```

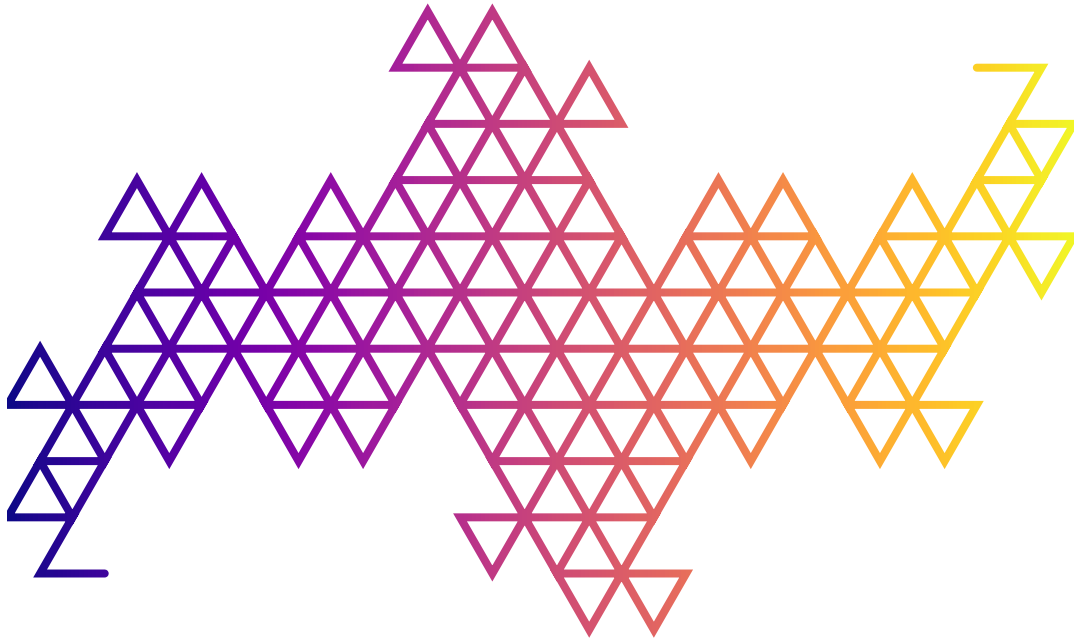
Generation

```
1 { Terdragon curve: axiom: F; F = F+F-F }
2 { F=forward; -=turn 120 left; +=turn 120 right }
3
4 root = (-F|"+"|"-")+ .
5 F = -"F",+"F+F-F" .
```



Generation

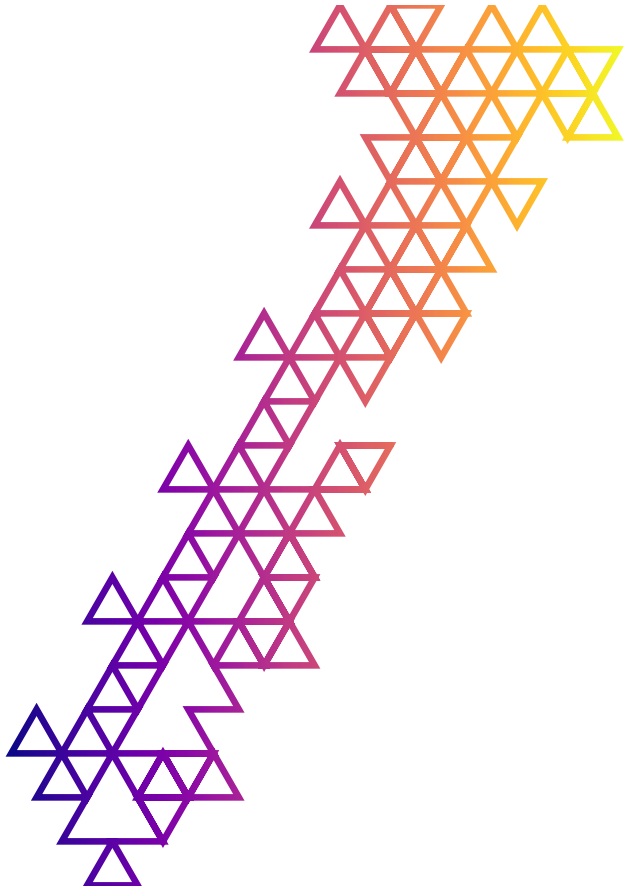
```
1 F => <root>F+F-F</root>
2 F+F-F => <root>F+F-F+F+F-F-F+F-</root>
3 F+F-F+F+F-F-F+F-F
```



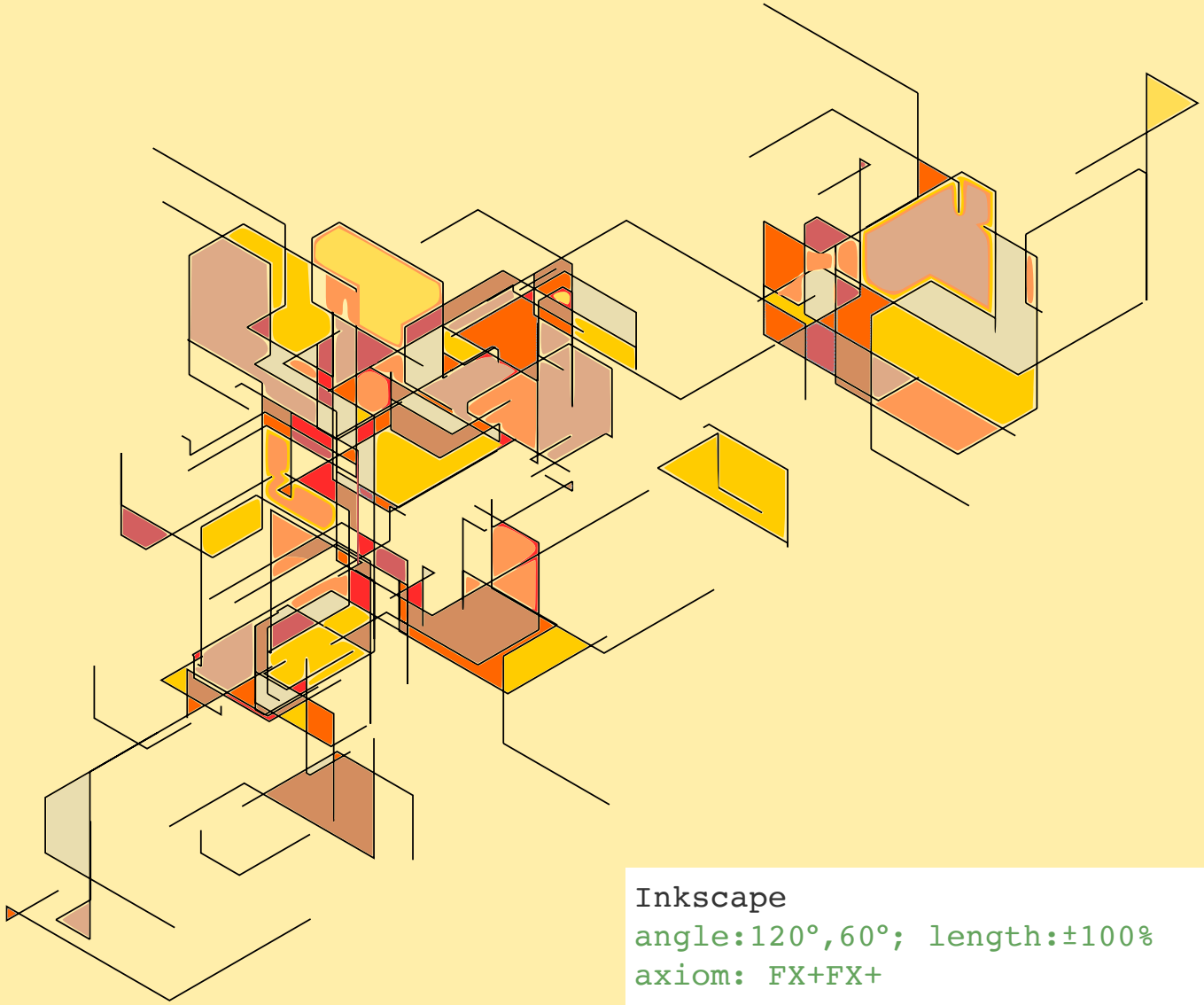
Stochastic Generation

```
1 { Stochastic Terdragon curve: axiom: F; F = F+F-F (80%), F = F-F+F (20%) }
2 { F=forward; -=turn 120 left; +=turn 120 right }
3
4 root = (-F|"+"|"-")+ .
5 F = -"F",+"F+F-F" | -"F",+"F-F+F".
```

Use CoffeeSacks choose-alternative:
weighted random choice



visual

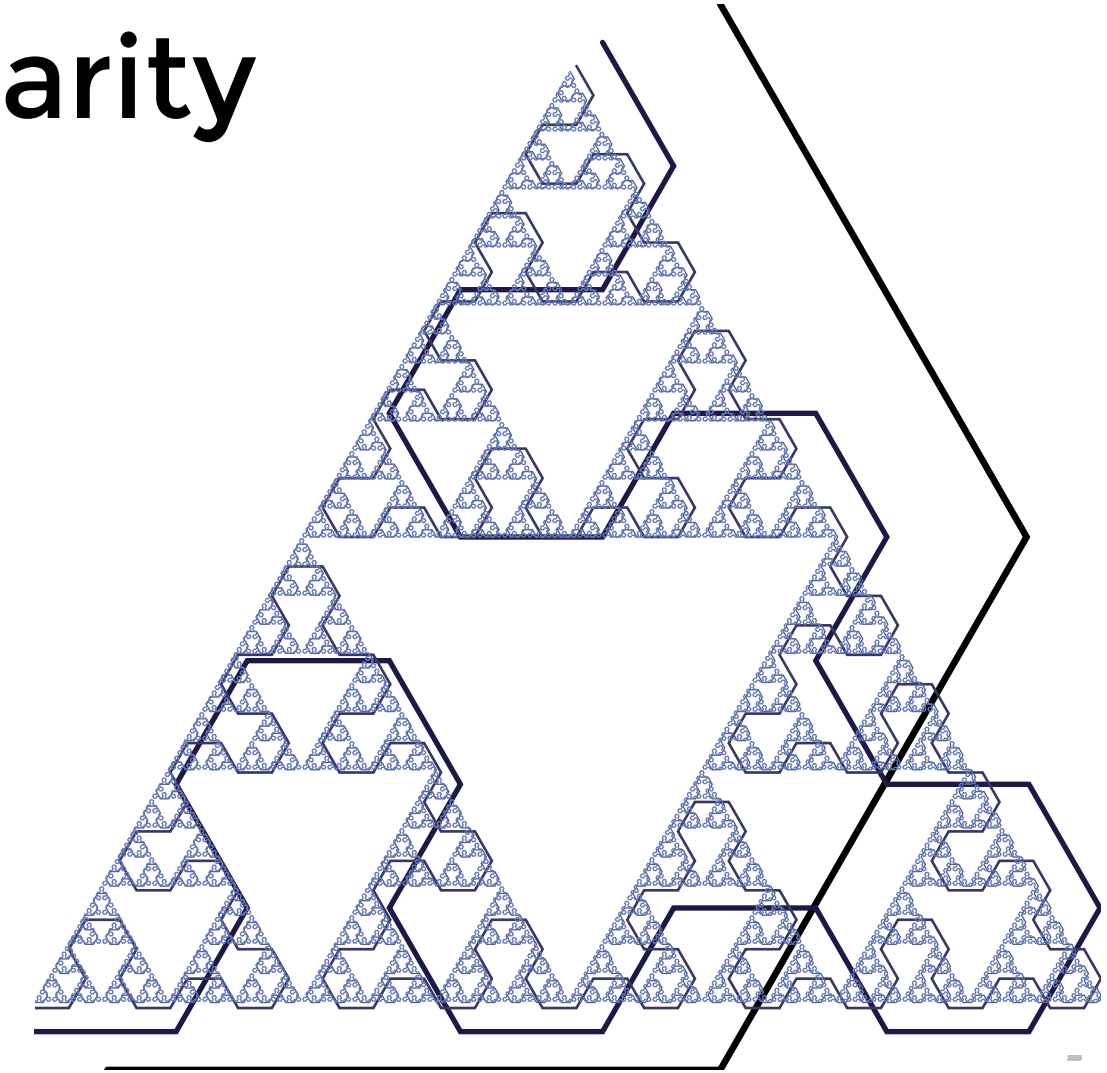


Inkscape
angle:120°,60°; length:±100%
axiom: FX+FX+
X = FX+FY
Y = FX-FY

Recursive self-similarity

Fractal

```
let $axiom := "A"
let $rules := map {
  "A": "B-A-B",
  "B": "A+B+A"
}
```



Recursive self-similarity

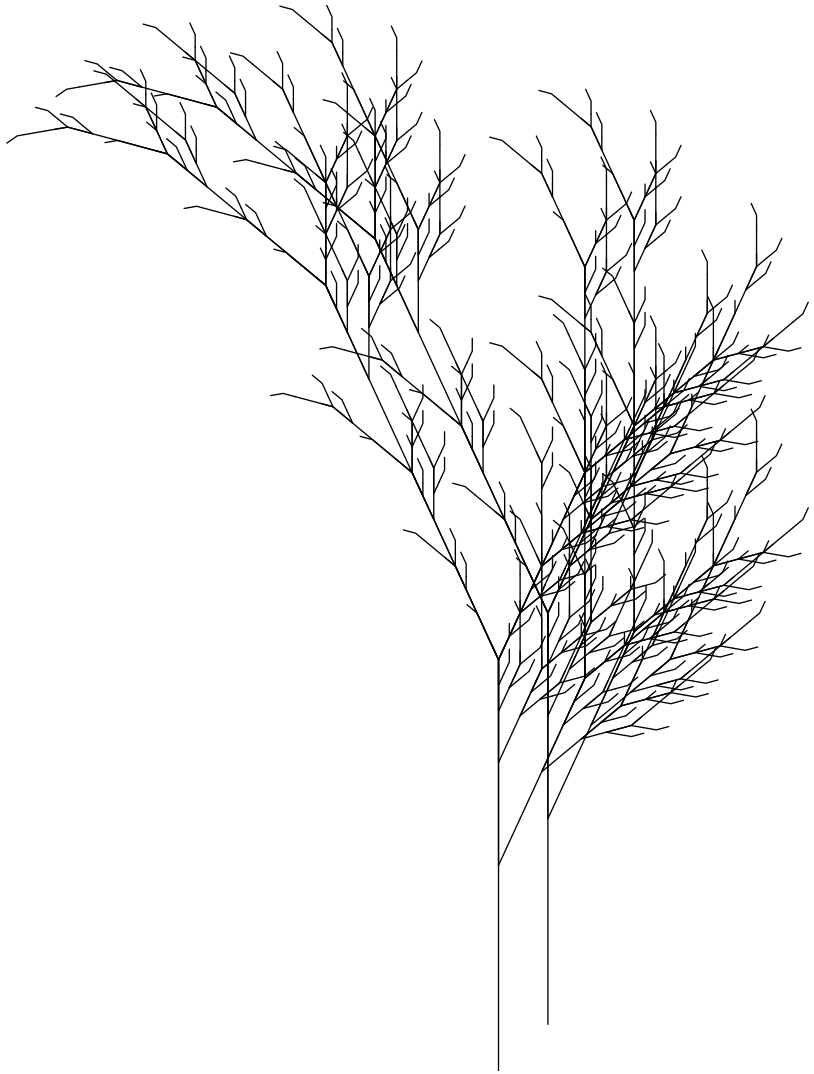
Fractal

```
let $axiom := "A"
let $rules := map {
  "A": "B-A-B",
  "B": "A+B+A"
}
```

Recursive self-similarity

Organic growth

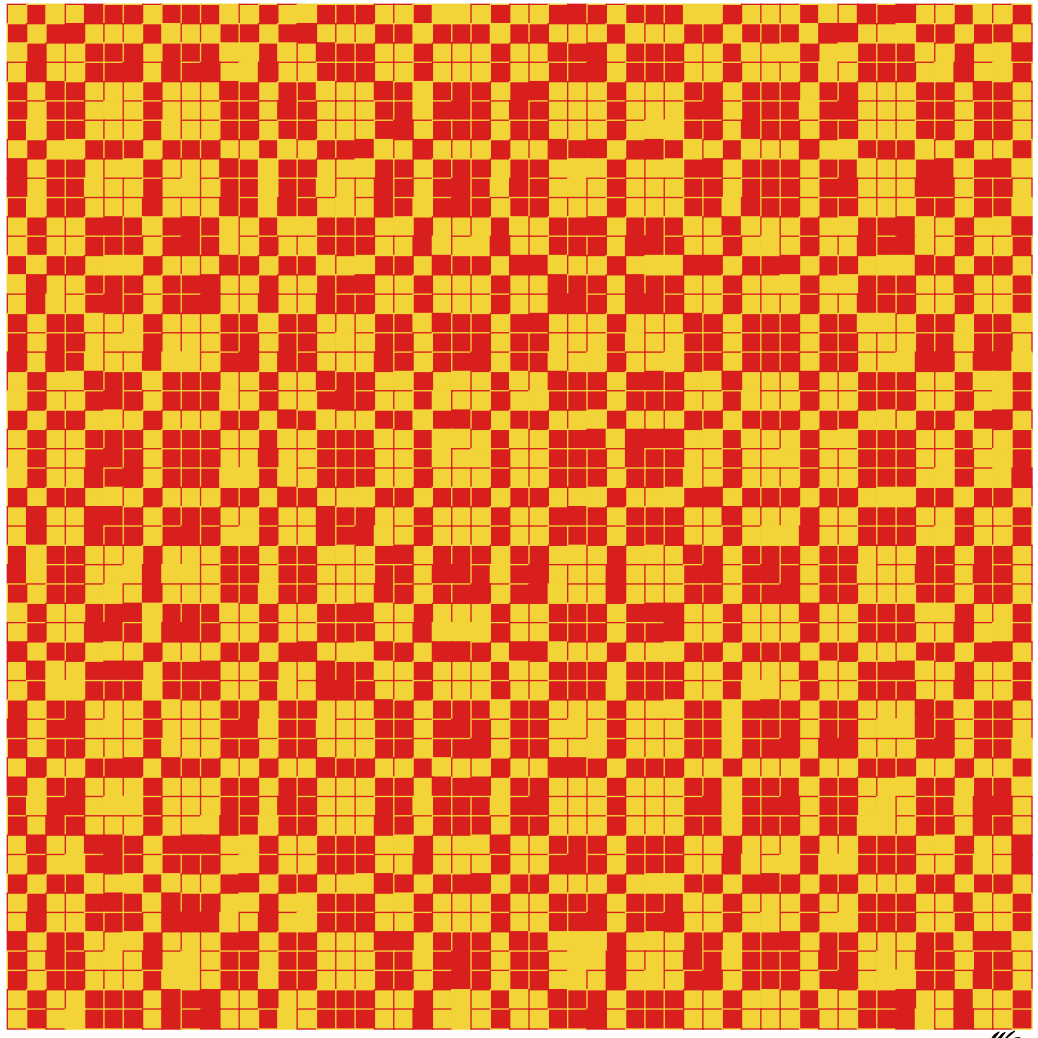
```
1       let $axiom := "X"
2       let $rules := map {
3         "X": "F+[[X]-X]-F[-FX]+X",
4         "F": "FF"
5       }
```



Pattern-generation

Structured, rule-based

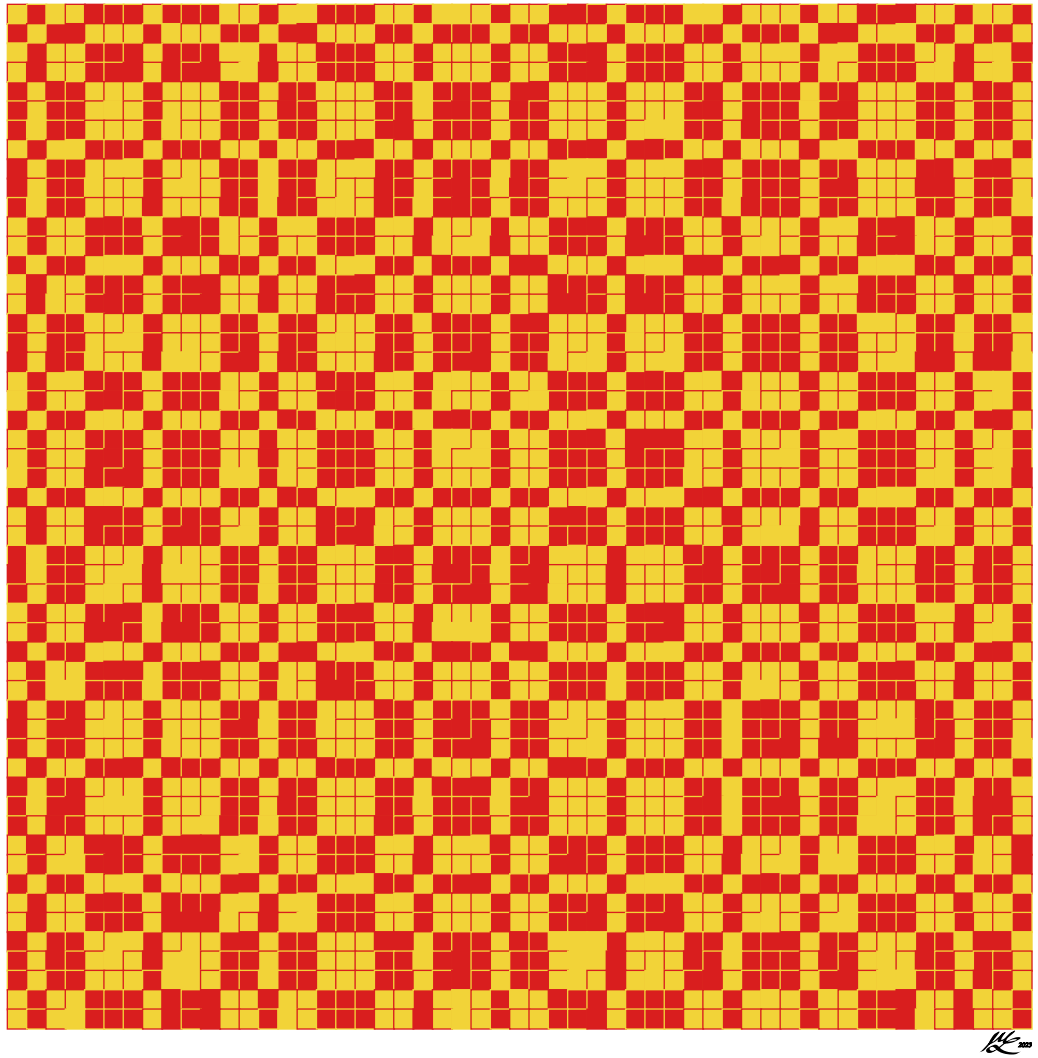
```
let $axiom := this:rbinary(0,31)
let $rules := map {
  "0": "01",
  "1": "10"
}
```



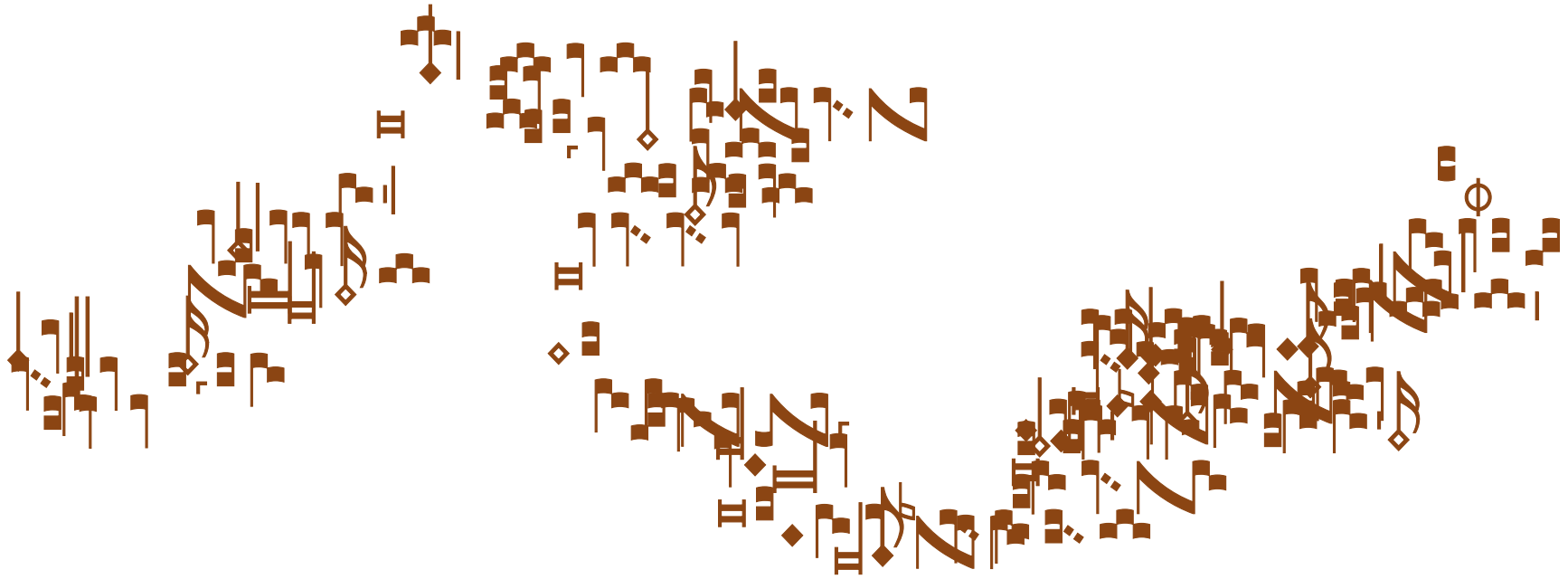
Pattern-generation

Create an integer sequence

```
1  011
2  011010
3  011010010101
4  01101001100101100110011 0
```



musical



```
angle: 90
axiom: ?F
F = F+<GF-GF>
```

linguistic

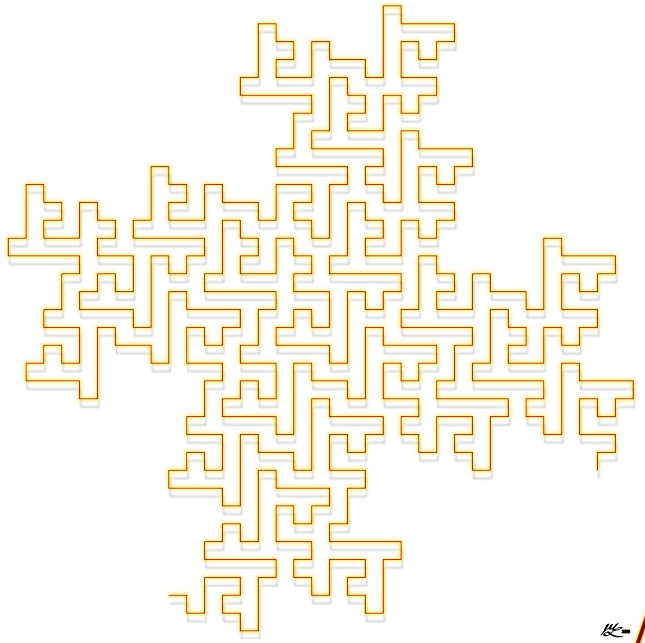
This idiosyncratic Svengali tacitly despised the contractable safe-deposit box.

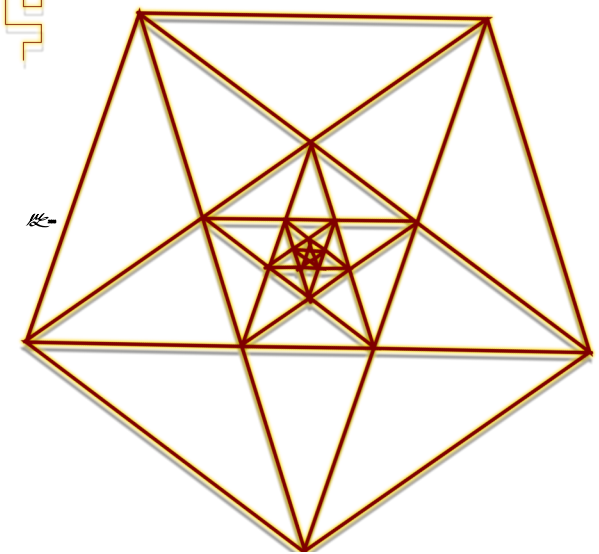
```
axiom: R
R = SAVO (40%)
R = SVBO (30%)
R = SAV (7%)
R = SVB (3%)
R = SAVPO (7%)
R = SVBPO (3%)
R = SAVOCR (7%)
R = SVBOCR (3%)

S = DJN
N = N (95%)
N = NN (5%)
O = DN (50%)
O = DJN (50%)
J = J (95%)
J = JJ (5%)
```

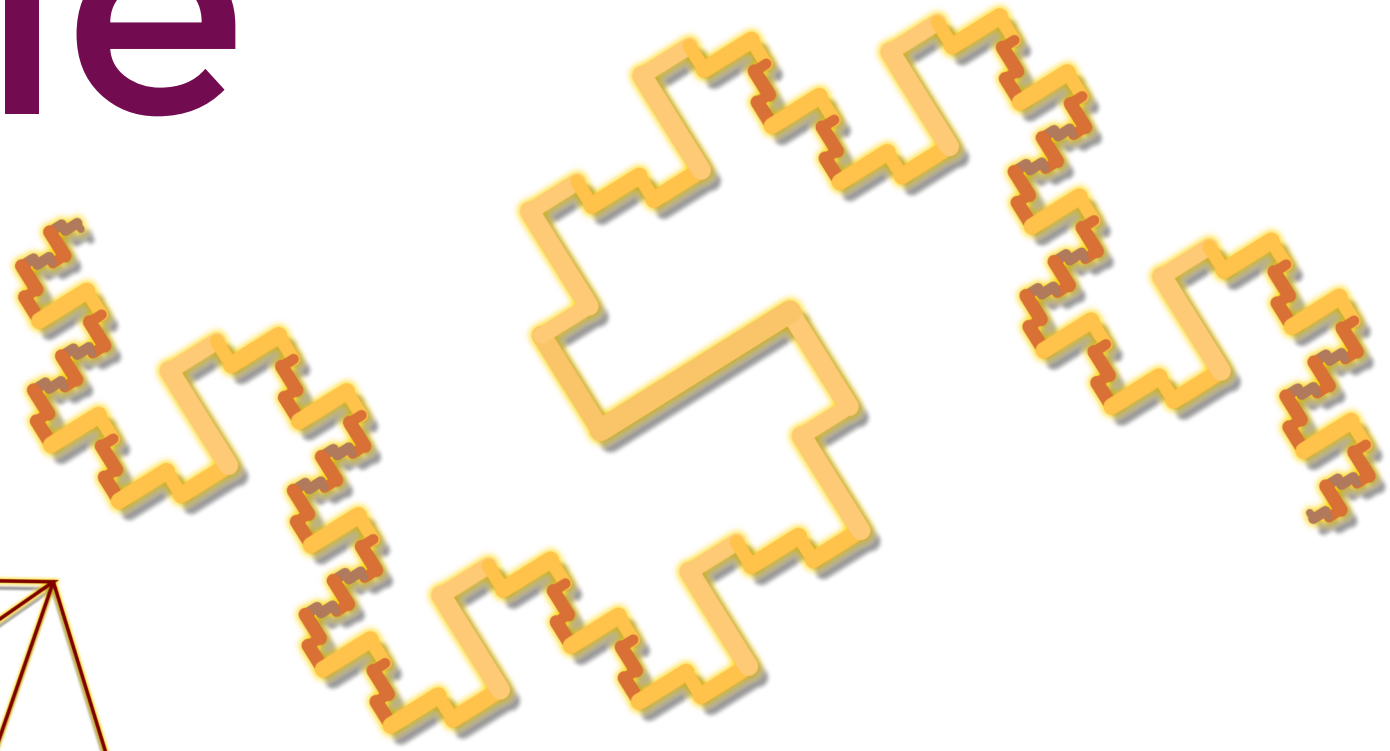
architecture

http://frampetit.com/

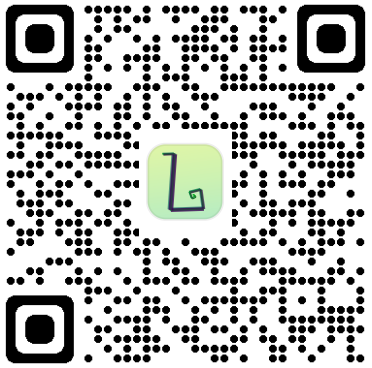
multiple







Mary's art code: https://mathling.com/code/



Juri's library: https://github.com/line-o/linsy