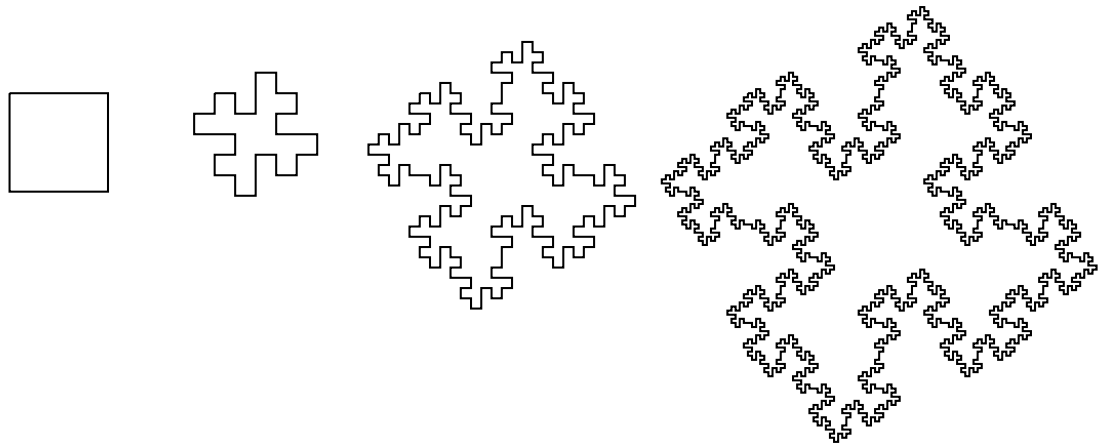
Project 1

Simple L-Systems and Fractal Production



Aristide Lindermayer worked on the understanding and representation of plant growth. He invented a way to describe how simple plants and algua grow. This approach is named L-Systems [**?**]. Although L-Systems are used to describe the plant growth, they can also be applied to produce fractal graphics – graphics based on repetitions of themselves, as the ones above. Moreover, the graphical interpretation of L-Systems is easy with a turtle. This is what we explore in this chapter. In this chapter, we propose you to implement the simplest form of L-System.

1 L-Systems

L-Systems is a generic term of designing different rule-based rewriting systems. A rewriting rule-based system is a system that given an *input* like a sequence of characters, strings, or numbers and the *set of rules* explaining how an element of the input is replaced by new ones, produces an *output*.

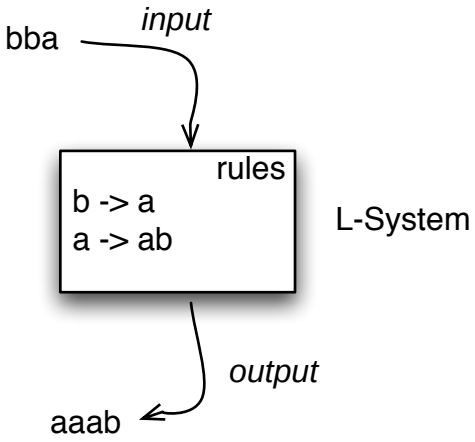
Let us look at the simple example[1] below.

Fibonacci

Rule 1: b → a
Rule 2: a → a b

It is composed by two rules, the first means that every b has to be replaced by an a, the second means that every a has to the replaced by ab. Now if we start with b and apply the two rules we obtain the following results where each line represents the application of the rules to the result of the previous line.

[1] A fun aspect of this L-System is that the length of the production is the Fibonacci suite.



Input	*b*
(Rule 1)	$\rightarrow$ *a*
(Rule 2)	$\rightarrow$ *a b*
(Rules 2, 1)	$\rightarrow$ *a b a*
(Rules 2, 1, 2)	$\rightarrow$ *a b a a b*
(Rules 2, 1, 2, 2, 1)	$\rightarrow$ *a b a a b a b a*

We call the first *input*, in the example *b*, the *original input* of the L-System. Note that we say that we *apply* a rule. We call the symbols a and b that constitutes the first input and output the vocabulary of the L-System.

Analysis. Let us analyse this simple L-System[2] and understand the basic mechanisms used here:

- An L-System can contain several rules. Here we have two rules namely Rule 1 that transforms every element b into a and Rule 2 that transforms every a into the sequence of elements ab.
- A rule is composed by two parts: a *left* and a *right* part. The left part identifies the element in the input that will be replaced by the right part. The left part of a rule only identifies *one* element while the right part can be *one or multiple* elements.
- *Applying a rule* means replacing in the current input the element (left part of the rule) by the sequence of elements described in the right part of the rule. For example, the third line above is produced by applying the Rule 1 and substituing a by ab.
- The rules are applied each time they can be applied. For example, the production of the fourth line is the result of applying the Rule 2 and the Rule 1 to the sequence ab. Note that the application order is irrevelant as long as it is performed in parallel *i.e.*,that a rule is only applied on the input and not on the result of another rule application.
- A rule can be applied several times. The production of the last line is the result of the application of the rules 2, 1, 2, 2 and 1.
- The input does not have to be necessary one single element.

2 The Graphical Interpretation of L-Systems

Besides representing plant growth, L-Systems are also interesting systems because we can use them to produce graphics representing plants or fractals. Fractals are graphics that are built using repetitive patterns

[2]Certain L-Systems are much more complex and take into account the context of the element or some other parameters before applying the rule.

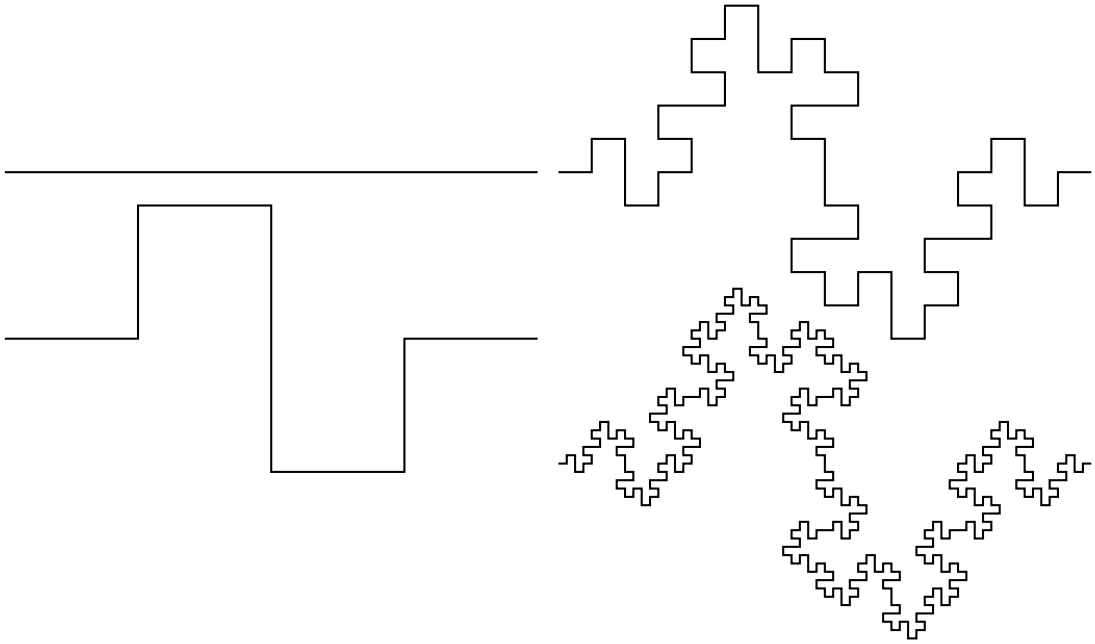


Figure 1.1: Steps 0, 1, 2 and 3 of the L-System defined by Input: F, Angle=90, F → F+F-F-FF+F+F-F. As a segment is cut into 4 segments, we divide the length by 4 from iteration to iteration. The lengths of the picture are 64 * 4, 64, 16 and 4.

and that also partially or fully contain themselves. The idea to produce such graphics based on L-Systems is to use L-System whose vocabulary is mapped to turtle actions.

2.1 A Turtle Dialect for L-Systems

To produce graphics we chose to define the vocabulary of the L-Systems the following way: input and rules only manipulate a limited set of predefined symbols representing turtle methods. These symbols are:

- `F` : the turtle goes forward and leaves a trace from a given distance. It corresponds to our `go:` method.
- `f` : the turtle goes forward without leaving a trace from a given distance. It corresponds to our `jump:` method.
- `+` : the turtle turns on the left from a given angle. It corresponds to our `turnLeft:` method.
- `-` : the turtle turns on the right from a given angle. It corresponds to our `turnRight:` method.

This simplest form of L-System are based on an angle and a length to move forward that are defined *once* for all the rules of the L-System. The value of these data does not change during the rules application or graphical interpretation. This is the reason why you have to pay attention of the length we define if we want to have reasonable output.

2.2 A First L-System based on Turtle Actions

Let us look at the following example whose first application steps are illustrated by the Figure 1.1.

Figure 1.1

Input	*F*
Angle	90 degree
Rule	*F → F+F-F-FF+F+F-F*

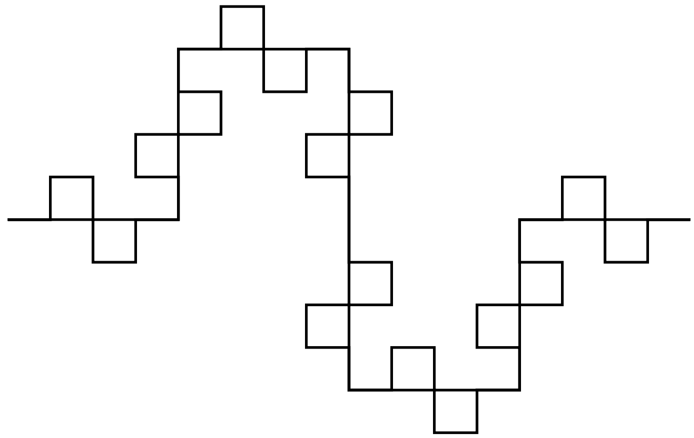


Figure 1.2: The superposition of the graphics produced after the second iteration and after the third iteration shows how the previous figure is decomposed.

- The input represents the geometrical shape that we take as starting point. Here *F* means that we start with a line, *F+F+F+F* a square as the angle is 90 degree.
- The rule describes how one segment is transformed in the second steps of the Figure 1.1.

Applying several times the rule produces the other graphics shown in the Figure 1.1. The Figure 1.2 superposes two graphics on the same picture to show how the rule application decomposes the segments into smaller ones.

Pay attention that applying the rules at the different levels can become really quickly huge output. For example here are the results of this L-System for the level application steps :

- *0* → *F*,
- *1* → *F+F-F-FF+F+F-F*,
- *2* → *F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-FF+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F*,
- *3* → *F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-FF+F-F-FF+F+F-F+F+F-F -FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-FF+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-FF+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-FF+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-FF+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-FF+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-FF+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-FF+F+F-FF+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F*

So take care during your experimentation not to ask crazy numbers, you could think that the system is down but it would be just computing for hours. Finally we could have used directly the name of the turtle methods but this would have led to overly long L-System definitions. Moreover, we prefer to stick with the original L-System notation.

3 Implementing the L-System

As shown in the Chapter **??** it is easy to add functionality to the class `Turtle` so that turtles are able to interpret a string composed by the L-System symbols F, f, +, and - and perform the corresponding actions. You should remember that a string is a collection of characters and that characters are single letter beginning with $. Simple L-Systems defines *once* for all the length of which a turtle should go forward and the angle they should turn. So define the method `interpretChar: aChar length: len angle: degree` that sends the action described by `aChar` to the receiver turtle. If the character is `$F` or `$f` the turtle should go forward (with trace and without trace) from `len` distance. If the character is `$+` or `$-` the turtle receiver should respectively to the left or right from `degree`. The Script 1.1 should draw an U. A possible solution is shown in the method 1.1.

Script 1.1

```
|aTurtle|
aTurtle := Turtle new.
aTurtle interpretChar: $F length: 100 angle: 90.
aTurtle interpretChar: $+ length: 100 angle: 90.
aTurtle interpretChar: $F length: 100 angle: 90.
aTurtle interpretChar: $+ length: 100 angle: 90.
aTurtle interpretChar: $f length: 100 angle: 90.
aTurtle interpretChar: $+ length: 100 angle: 90.
aTurtle interpretChar: $F length: 100 angle: 90.
```

Method 1.1

```
In category L-System
interpretChar: aChar length: len angle: degree

   aChar = $F
      ifTrue: [self go: len]
      ifFalse: [aChar = $+
                  ifTrue: [self turnLeft: degree]
                  ifFalse: [aChar = $-
                                ifTrue: [self turnRight: degree]
                                ifFalse: [aChar =$f
                                            ifTrue: [self jump: len]]]
```

Now we want to treat *sequences* of characters and not characters one by one. Therefore define the method `interpret: aCollection length: len angle: degree` that knows how to interpret a string, *i.e.*,a collection of characters. Hints: String is a subclass of Collection so operations available on collections such as `do:` are also possible on strings. The Script 1.2 should also draw an U.

Script 1.2

```
|aTurtle|
aTurtle := Turtle new.
aTurtle interpret: 'F+F+f+F' length: 100 angle: 90.
```

A possible solution is shown in method 1.2

Method 1.2

```
In category L-System
interpret: aString length: len angle: degree
    aString do: [:each | self interpretChar: each length: len angle: degree]
```

Now we are ready to implement the application of rules on an input.

4 L-Systems with a Single Loops

For now we limit our approach to the simplest L-System: a system containing only one rule. We took this decision so that you could have a fast idea of the process. What is missing now is the application of the rule on an input. We want to be able to specify an input, the rule to apply, a number of times the rule should be applied. Define the method `applyOnInput: aString leftPart: leftPart rightPart: rightPart level: n`. Just remember that computing the input a certain number of time requires to apply any possible rules then reiterate the rule application on the result you obtained previously. To help you, the method `copyReplaceAll:with:` defined on the class `SequenceCollection` replaces all the occurences of a string by another in the receiver as shown by the script 1.3.

Script 1.3

```
'How now brown cow?' copyReplaceAll: 'ow' with: 'ello'
    -Print It-> 'Hello nello brellon cello?'
```

Script 1.4

```
|aTurtle|
aTurtle := Turtle new.
aTurtle applyOnInput: 'F' leftPart: 'F' rightPart: 'F+F-F-FF+F+F-F' level: 1
    -Print It->'F+F-F-FF+F+F-F'

aTurtle applyOnInput: 'F' leftPart: 'F' rightPart: 'F+F-F-FF+F+F-F' level: 0
    -Print It-> 'F'

aTurtle applyOnInput: 'F' leftPart: 'F' rightPart: 'F+F-F-FF+F+F-F' level: 2
    -Print It-> 'F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F-F+F-F-

FF+F+F-FF+F-F-FF+F+F-F+F+F-F-FF+F+F-F+F+F-F-FF+F+F-F-F+F-F-FF+F+F-F'
```

Here is a possible implementation of the method `applyOnInput:leftPart:rightPart:level:` is shown in the method definition 1.3.

Method 1.3

```
In category L-System
applyOnInput: input leftPart: start rightPart: repl level: n
   "Generate the nth output of a simple L-System composed of one rule
   replacing leftPart by rightPart in input"

   |production|
   production := input.
   n timesRepeat: [production := self
                                 replace: start
                                 by: repl
                                 in: production].
   ^ production
```

Method 1.4

```
replace: aCharacter by: aCollection in: aSequence
   ^ aSequence copyReplaceAll: aCharacter with: aCollection
```

Now you are in the position to produce your first graphics. You have to use the methods `applyOnInput:leftPart:rightP` and `interpret:length:angle:`. Try to program the L-System we show previously.

Figure 1.1

Input *F*
Angle 90 degree
Rule *F → F+F-F-FF+F+F-F*

5 A Gallery of 1-rule based L-Systems

We show you some of the pictures that you can now create. The first pictures of this chapter are generated by the following L-System. This family of curves are called Koch curves. The other pictures of this chapter are generated using the described L-Systems. Note that some of them may take some time to draw.

First Pictures of the Chapter
Input *F-F-F-F*
Angle 90 degree
Rule *F → F-F+F+FF-F-F+F*

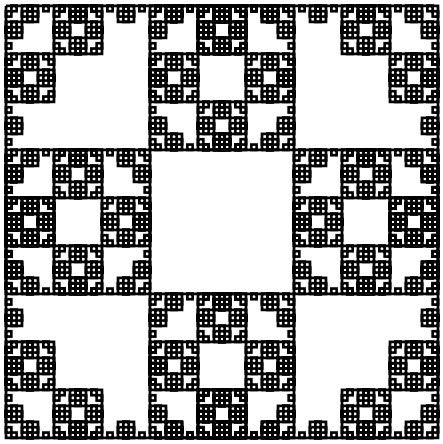


Figure 1.3: Variation on Koch curve 2: n = 4, angle = 90, Input: *F-F-F-F*, *F → FF-F-F-F-FF*

Figure 1.3
Input *F-F-F-F*
Angle 90 degree
Rule *F → FF-F-F-F-FF*

Figure 1.4
Input *F-F-F-F*
Angle 90 degree
Rule *F → FF-F-F-F-F-F+F*

Figure **??**
Input *'F-F-F-F'*
Angle 90 degree
Rule *F → FF-F+F-F-FF*

Figure 1.6 with n=4 and 1.7 with n=5
Input *F-F-F-F*
Angle 90 degree
Rule *F → FF-F- -F-F*

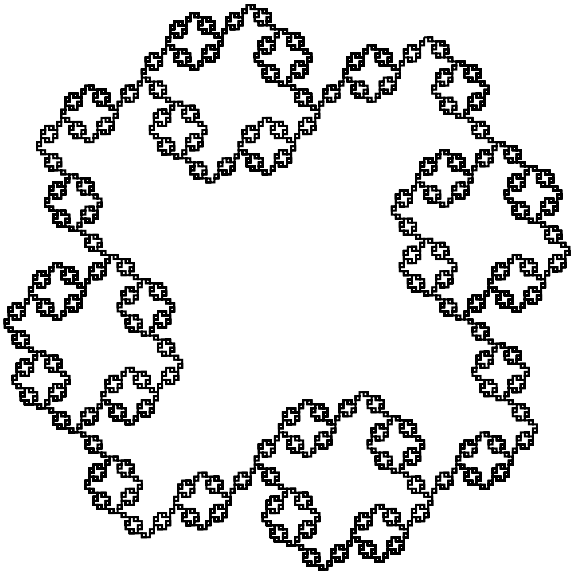


Figure 1.4: Variation on Koch curve 1: n = 4, angle = 90, Input: *F-F-F-F*, *F* → *FF-F-F-F-F-F+F*

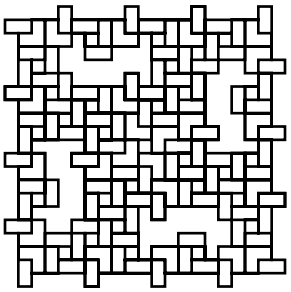


Figure 1.5: Variation on Koch curve 3: n = 3, angle = 90, Input: *F-F-F-F*, *F* → *FF-F-F+F-F-FF*

Figure 1.8
Input *F-F-F-F*
Angle 90 degree
Rule *F* → *F-F+F-F-F*

6 Experimentation

We suggest you to try the following L-Systems that we only describe here then try to create the L-System that produce the snow flakes shown in the Figure 1.9.

Interesting at level 2
Input *F-F-F-F*
Angle 90 degree
Rule *F* → *F+FF-FF-F-F+F+FF-F-F+F+FF+FF-F*

Interesting at level 3
Input *F*

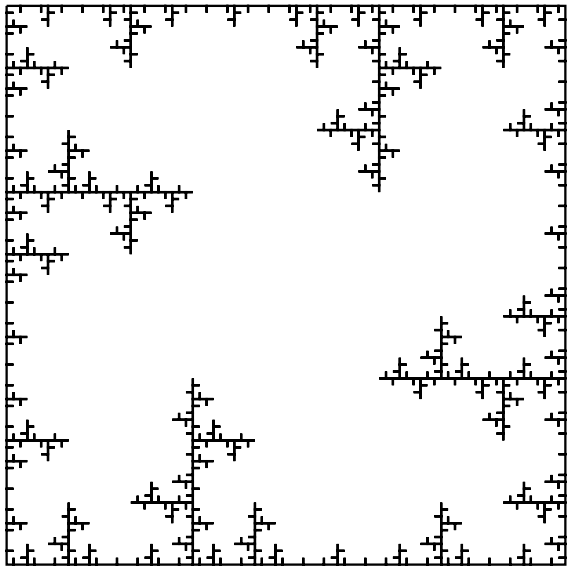


Figure 1.6: Variation on Koch curve 4: n = 4, angle = 90, Input: *F-F-F-F*, *F* → *FF-F- -F-F*

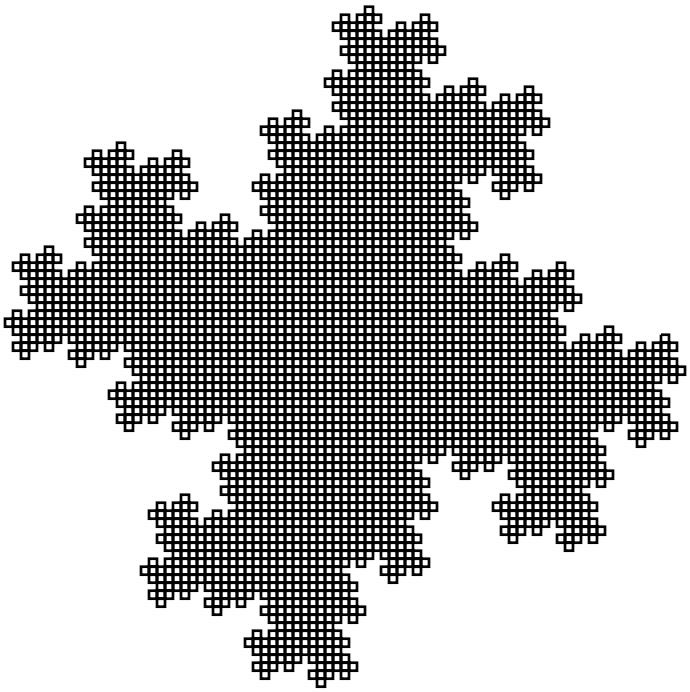


Figure 1.7: Variation on Koch curve 5: n = 5, angle = 90, Input: *F-F-F-F*, *F* → *F-FF–F-F*

Angle 90 degree
Rule *F* → *F+F-F-F+F*

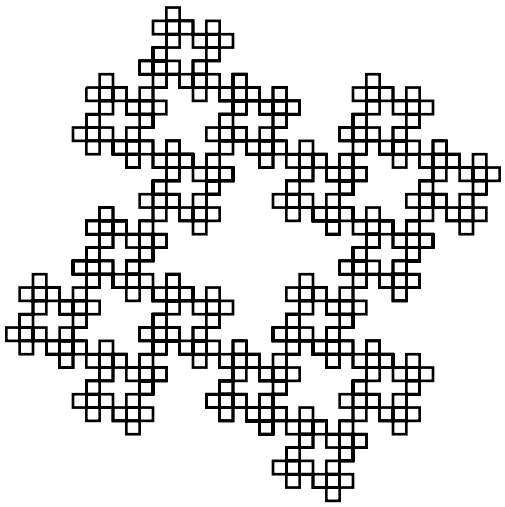


Figure 1.8: Variation on Koch curve 6: n = 4, angle = 90, Input: *F-F-F-F*, $F \rightarrow$ *F-F+F-F-F*

Experiment 1.1
Experiment and find the L-System generating the previous figures. Hints: start with the smallest interesting input, the smallest level of derivation and play with the production rule to generate the transformation of a side.

7 Analysis of the Solution

The fact that the solution only allows the expression of single rule L-System is not really a problem because we learnt this way the essence of the problem and are now ready to develop a more complex system. From a methodological point of view this is a good point. Some people think that producing the best solution handling all kinds of variations is better, however often we cannot forsee where the complex part will be and what are the possible variation. So having a small implementation up front working even in a limited setting is a way to understand the problem. We should only be prepared to change it heavily or to simply throw away what we made.

Even if the solution we proposed here is successful to manage trivial L-Systems, its design is not really good bad from an object-oriented perspective. As this is not the topic of this book this is not really a problem. However, we just want to tell you why. In fact methods should represent the behavior or responsibilities that an object is carrying. Here a method such as `interpret:length:angle:` clearly represents the behavior of a turtle. This method allow one to represent messages sent in another form, here strings. But this is not the case for the methods `replace:by:in:` and `applyOnInput:leftPart:rightPart:level:` that have nothing to do with a turtle. Second these methods do not even use the state of the turtle or its methods. This is a sign that they are defined in the wrong place. A good solution would define a class representing an L-System and these methods would be moved there but this is the topic of another book.

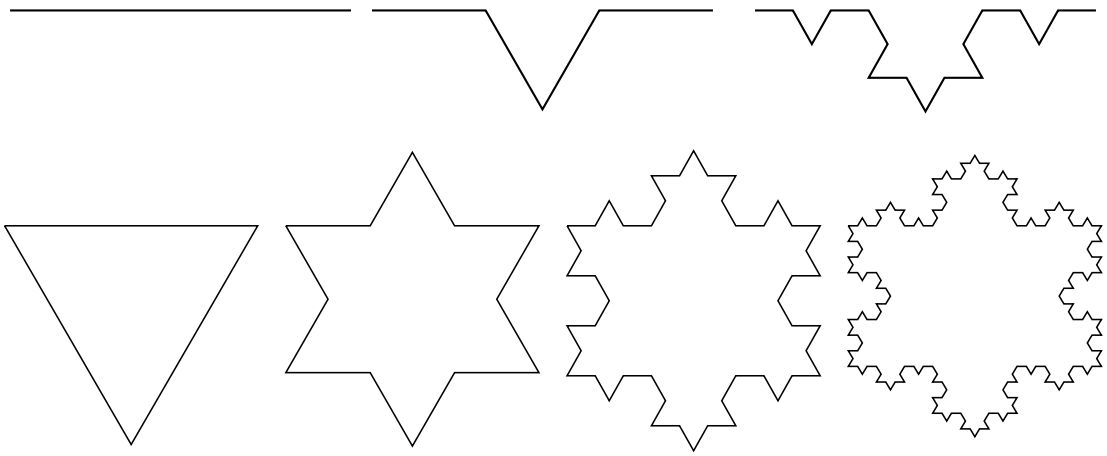


Figure 1.9: Steps to create a nice snow flake.