L-Systems

Civilization is founded on language. Our accumulated knowledge has been preserved in writing. We communicate using language.

Everyone knows at least one *natural language*, such as German, French, or English. The term natural has arisen because of languages that have not evolved in the normal course of human civilization. The most well known of these "unnatural" languages are programming languages, such as FORTRAN, C, and Java, which contain instructions for running computers.

Strings

A central component of languages is the *string*, which is a sequence of characters, such as a word, a phrase, or a telephone number. The characters may be letters, digits, punctuation marks, dollar signs, and so forth. Uppercase letters are used in examples to make them stand out, as in ABCBA.

The characters in strings may or may not have meanings associated with them. For the time being, they will just be abstract symbols. Strings may of course, contain patterns; in fact, this is the major interest here. For example, the string ABCBA is a palindrome, reading the same way forward and backward. Another example is DEDEDEDE, which consists of four repetitions of DE.

A string within a string is called a *substring*. For example, DE, ED, and DED are among the many substrings in DEDEDEDE.

The length of a string is the number of characters in it. For example, the length of ABCBA is 5. The *empty string*, consisting of no characters, has length 0. The empty string is denoted by Θ.

A single character is a one-character string.

Concatenation consists of appending one string to another. For example, the concatenation of ABCBA and DE produces ABCBADE.

Formal Language Systems

Linguists studying languages have developed *formal language systems* as models for exploring the expressive powers of different kinds of languages. Most formal language systems are arcane and known only to specialists. A few, however, have practical applications — sometime surprising ones.

A formal language system consists of three parts:

- (1) an *alphabet*, which might consist of letters such as A, B, C, …
- (2) a *grammar* that determines how *sentences* in the system are produced

- (3) a *seed*, which is the starting sentence.

A sentence is a string of symbols in the alphabet, such as BA, BCD, DA, … A *language* for the system is a set of sentences the grammar can produce.

Grammars usually consist of *rewriting rules* that describe how one sentence is produced from another.

Here is an example:

Alphabet: A and B

Rewriting Rules:

1: A ➛ BA

2: B ➛ AB

Seed: A

Grammar: a randomly chosen letter of the current sentence is replaced according to its rewriting rule to produce the next sentence.

The resulting language might be

sentence	*rules*	*generation*
A	seed	0
BA	Rule 1 applied to first character (A)	1
ABA	Rule 2 applied to first character (B)	2
ABBA	Rule 1 applied to last character (A)	3
AABBA	Rule 1 applied to second character (B)	4
…		

Different random choices would, of course, produce different languages.

The process of producing a language, starting with a seed, is called *generation*. The seed is generation 0; subsequent generations are numbered 1, 2, 3, … a shown bove.

Phrase-Structure Languages

One common formal language system uses *phrase-structure grammars*. In such grammars, there are alternative rewriting rules, such as

A ➛ BA

A ➛ AB

The rewriting rule used for a symbol is chosen at random. Sentences are produced starting with the seed and picking one symbol in the sentence to

replace, also at random.

Here's an example:

Alphabet: W, C, S, P, 1, 2, 3, 4

Rewriting Rules:

1: W → 1
2: W → 2
3: W → 3
4: W → 4
5: C → c
6: C → y
7: C → m
8: S → WC
9: P → S
10: P → SP

Seed: P

One language for this grammar is

sentence	*character*	*rule*	*generation*
P			0
SP	1	10	1
SS	2	9	2
WCS	1	8	3
WCWC	3	8	4
4CWC	1	4	5
4CWm	4	7	6
4C2m	3	2	7
4c2m	2	5	8

That's the end of the language, since there are no more symbols with rewriting rules.

This example may seem pointless, but suppose W stands for width, C for color, S for stripe, P for pattern, c for cyan, y for yellow, and so on, with the

numerals standing for themselves. Then this grammar generates examples of simple stripe patterns. The one above has two stripes: cyan of width 4 and magenta of width 2.

Note that P is defined in terms of itself in rule 10:

$$P \rightarrow SP$$

This introduces the concept of recursion, which is a source of complexity. More on this important mechanism later.

L-Systems

Phrase-structure grammars are interesting, but they lack expressive power — that is, the complexity of patterns they can produce is limited and not very interesting. Among the more powerful and interesting formal language systems are Lindenmayer Systems, or L-Systems for short, which are named after their inventor, Aristid Lindenmayer.

Aristid Lindenmayer

1925 - 1985

Aristid Lindenmayer was a Hungarian biologist. His invention of the string rewriting system named after him "grew out of an attempt to describe the development of multicellular organisms in a manner which takes genetic, cytological and physiological observations into account in addition to purely morphological ones" [2].

It was a surprise to him when his system began to be used in many other disciplines and in many ways. In fact, he first rejected as a gimmick the graphical interpretation of L-System strings to draw plants, something he never envisioned when he started to manipulate strings of characters. Drawing ultimately proved to be the most influential use of L-Systems.

There are L-Systems of many different kinds and degrees of complexity, including ones that can be used to represent geometrical forms and even three-dimensional plants in color and [3-5]. Active research in the area continues to this day.

The distinguishing characteristic of L-Systems is that all rules are applied in parallel for each generation and every symbol is rewritten. In the simplest kind of L-System, there are no alternative rules.

Here is an example L-System:

seed: ABCD
rules: A → BD
B → B
C → ACA
D → Θ

The first rule specifies that A is replaced by BD. The second rule specifies that B is replaced by B; that is, it is unchanged. The third rule specifies that C is replaced by ACA, while the fourth rule specified that D is to be replaced by the empty string; that is, deleted.

The generation goes like this:

string	*generation*
ABCD	0
BDBACA	1
BBBDACABD	2
BBBBDACABDB	3
BBBBBDACABDBB	4
BBBBBBDACABDBBB	5
BBBBBBBDACABDBBBB	6
…	…

Bs accumulate and each generation is longer than the previous one.

Despite their apparent simplicity, L-Systems are very powerful. Their languages may contain intricate patterns with infinitely varying subtlety. Among other things, they can describe plant development (their original use), fractals, and complex geometric designs.

The power of L-Systems comes from parallel rewriting and repeated application (iteration) of the rules. These properties are of fundamental importance and apply to entirely different mechanisms, such as cellular automata [1], and to many processes in the physical world.

Example L-Systems

Example 1: The Morse-Thue sequence [6] is produced by a very simple L-System:

seed: A
rules: A → AB
B → BA

The generation goes like this

A
AB
ABBA
ABBABAAB
ABBABAABBAABABBA
…

Note that the lengths of the strings double with each generation.

Example 2: The Fibonacci string sequence, analogous to the Fibonacci number sequence [7], is produced by this L-System:

seed: A
rules: A → B
B → AB

Generation goes like this:

A
B
AB
BAB
ABBAB
BABABBAB
ABBABBABABBAB
BABABBABABBABBABABBAB
…

Note that the lengths of the generations give the Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13, 21, …

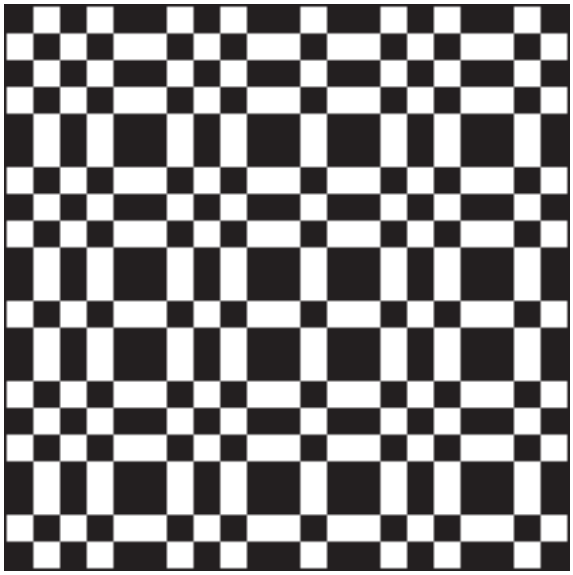
Incidentally, no phrase-structure grammar can generate either the Morse-Thue string sequence or the Fibonacci string sequence.

Interpreting L-System Languages

The strings produced by L-Systems such as those given above have evident patterns but the characters themselves have no meaning. If A and B in the Morse-Thue example are interpreted as 0 and 1, respectively, the results is the usual interpretation of the Morse-Thue sequence as a sequence of binary digits.

Many other kinds of interpretation are possible. One of the most striking methods interprets characters as drawing commands. The next section describes the drawing interpretation.

A very natural interpretation of strings like the Fibonacci strings is as profile sequences. Here is a profile pattern for the seventh-generation Fibonacci string:



Profile Drafting

In fact, the simplest and most straightforward use of L-Systems is in the design of profile drafts.

Here is another example:

seed: A

rules: A → ABB
B → BCC
C → CAA

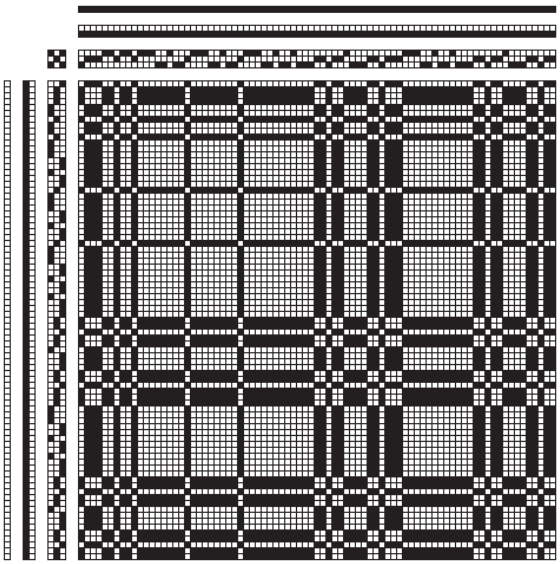
The first four generations are

	generation
ABB	1
ABBBCCBCC	2

ABBBCCBCCBCCCAACAABCCCAACAA 3

ABBBCCBCCBCCCAACAABCCCAACAA 4
BCCCAACAACAAABBABBCAAABBAB
BBCCCAACAACAAABBABBCAAABBABB

Any of these generations beyond the trivial one can be used for profile "threading" and "treadling" sequences. From here on out, we'll drop the quotes. Here is a draft that uses generation 4 for threading and treadling:



The number of blocks is, of course, the number of symbols for which there are replacement rules. By the nature of L-Systems, useful ones for profile drafting are limited to four or five blocks; otherwise the generated strings quickly get too long.

In designing L-systems for profile drafts, the distinguishing characteristic is the repetition of blocks to create varying pattern widths. A profile sequence without block repetitions is, of course, possible, but it is indistinguishable from a treadling sequence, and has limited utility for profile design.

Block repetition is easy to build into L-Systems. The problem is more one of avoiding excessively long generation strings. See the preceding example, which has only three symbols. In this L-System, every generation is three times as long as the preceding one: 3, 9, 27, 81, 243, … Since profile drafts are the source of threading drafts with more than one thread per block, the limitations are clear.

Another consideration is symmetry. Palindromic mirroring is a powerful

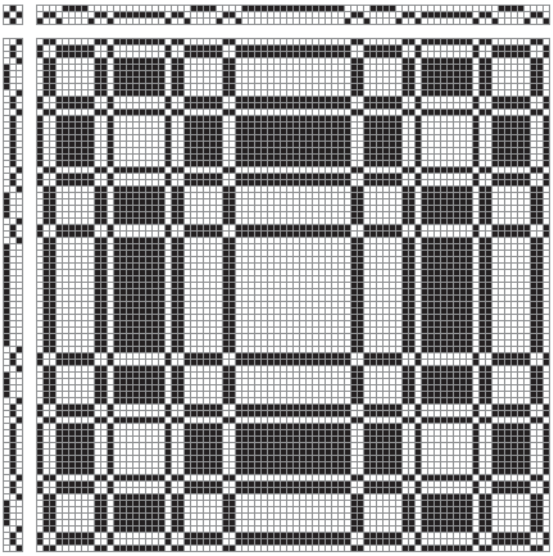
tool for producing attractive, even compelling patterns. Palindromes can be added after the fact, but they can be designed into L-System by the simple expedient of making all the rules palindromic. Here is an example:

seed: A
rules: A → ABBA
B → CC
C → BB

The generations are

	generation
ABBA	1
ABBACCCCABBA	2
ABBACCCCABBABBBBBBBBABBA CCCCABBA	3
ABBACCCCABBABBBBBBBBABBA CCCCABBACCCCCCCCCCCCCCC CCABBACCCCABBABBBBBBBBA BBACCCCABBA	4

A profile draft for generation 4 is



Other Ideas

There is a wealth of ways that profile drafts can be produced from L-Systems.

One idea is to use one L-System for the threading and another for the treadling.

Remember that interpretation is a very powerful tool when using L-Systems. Among the many possibilities is the use of a width sequence to replicate each successive block in a string by some number of times.

T-Sequence Design

One obvious ue of L-Systems is in the design of threading and treadling sequences.

Here is an example:

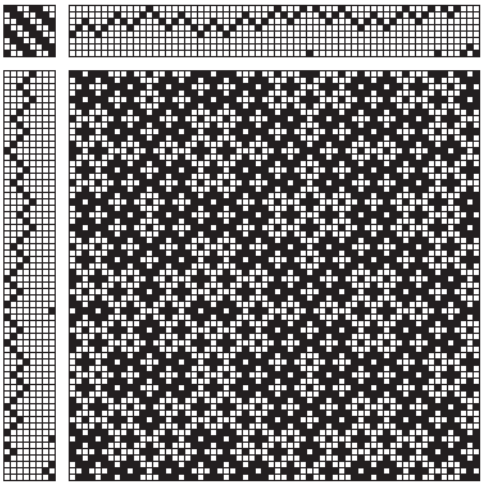
seed: 123456787654321
rules: 1 ➛ 23432
2 ➛ 34543
3 ➛ 45654
4 ➛ 56765
5 ➛ 67876
6 ➛ 78187
7 ➛ 81218
8 ➛ 12321

Thus every 1 in a string is replaced by 23432 and so on.

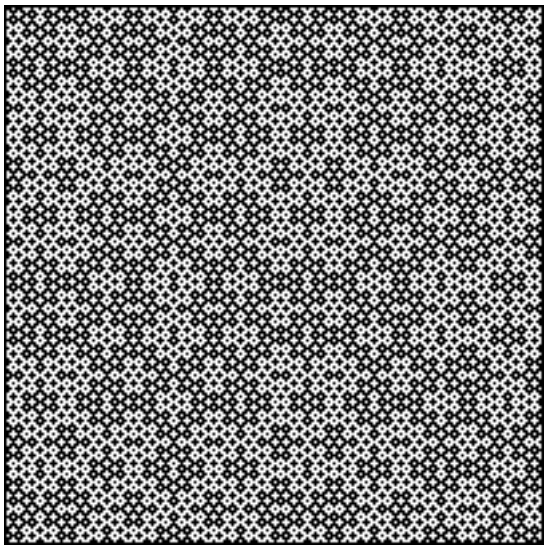
The length of successive generations increases by a factor of 5. The first generation is

2343234543456545676567876781878121 8
12321812187818767876567654565434
5432343

Here is a partial draft using the third generation of this L-System for threading and treadling sequences:



Here's the weave design:



Note: This is a crackle weave.

A Side Trip to Graphics

The first section on L-Systems showed an example in which the characters W, C, S, and P had mnemonic value to suggest widths, colors, stripes, and patterns. These characters themselves have no meaning — they are just suggestions. Striped patterns were an interpretation of the characters. The characters could just as well have stood for wind direction, capacity, speed, and pulse.

L-Systems generate abstract patterns of characters. Interpretation translates these patterns into meaningful, concrete objects.

One of the most striking and well-known interpretations of L-Systems is to produce graphic images. This section illustrates that. It has nothing to do with weaving, *per se*, but it does suggest how L-Systems might be used in weave design.

Consider this L-System:

seed: A

rules: A → BCDDAEFAEFBDFBAECA

B → BB

It's not at all obvious what motivates this particular L-System or why it might be interesting, although the complexity of the first rule suggests some intent. The lack for rules for C, D, E, and F seems curious, although they proliferate during rewriting since the default in such cases is to replace such characters by themselves.

Now consider this L-System, which is the same as the one above except that different characters are used.

seed: X

rules: X → F–[[X]+X]+F[+FX]–X

F → FF

The characters look a bit strange — this is the first L-System with characters other than letters. There is a reason for the characters chosen, however. They serve as mnemonic devices for the intended interpretation, which is as commands for a drawing program:

F move forward a specified length, drawing a line

f move forward a specified length, without drawing a line (not included in the example above)

\+ turn right a specified number of degrees

– turn left a specified number of degrees

[save the current position and direction

] restore the previously saved position and direction

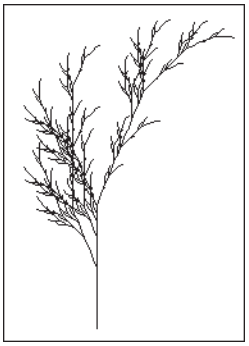
The character X in this L-System is a placeholder. It participates in an important way in the patterns produced, but it is ignored in interpretation.

Two parameters are needed to carry out the interpretation:

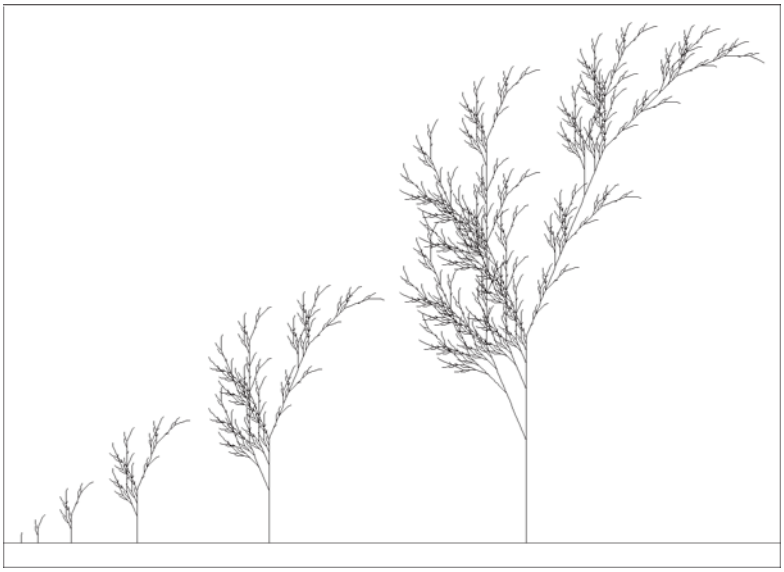
- The length for a move, which determines the scale of the drawing.

- The angle for turns, which is fundamental to the appearance of the result produced. For this L-System it is 22.5º — 1/4 of 90º.

The drawing is accomplished by producing several generations of the L-System and then interpreting the last one. For five generations, the image from interpretation is



Each generation increases the size and detail of the "tree".

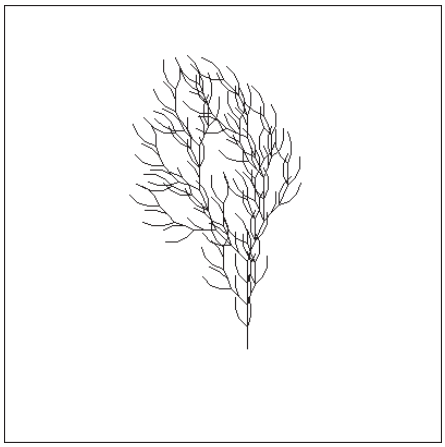


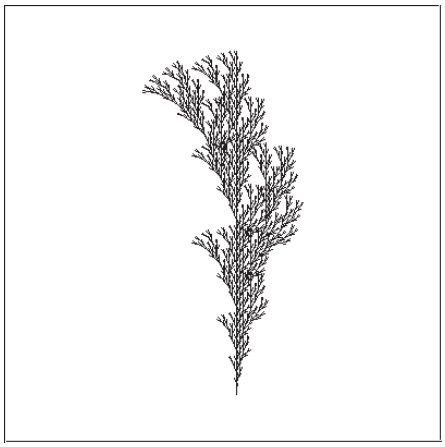
The strings produced by this L-System become very long as rewriting continues:

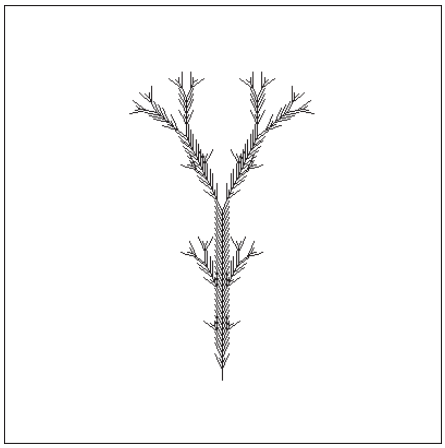
generation	*length*
1	90
2	380
3	1,552
4	6,264
5	25,160
6	100,840
7	403,752
8	1,615,784

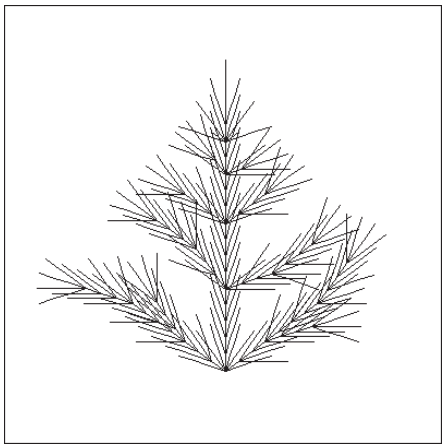
Nevertheless, it is not necessary to look at these strings or even produce them all at once. Only the drawing program uses them, and it can take the characters one by one. What these lengths *do* indicate is how many drawing actions are needed to produce the detail in the images.

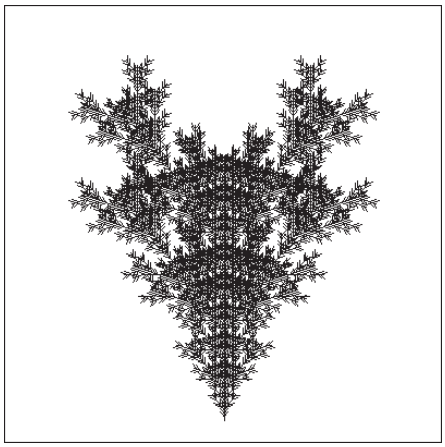
Plants drawn by L-Systems may attempt reality or approch the bizarre. Some examples are shown on the opposite page.

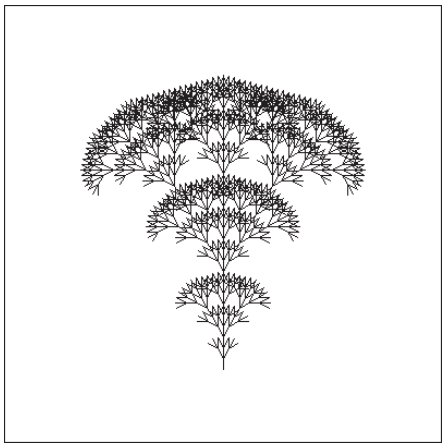




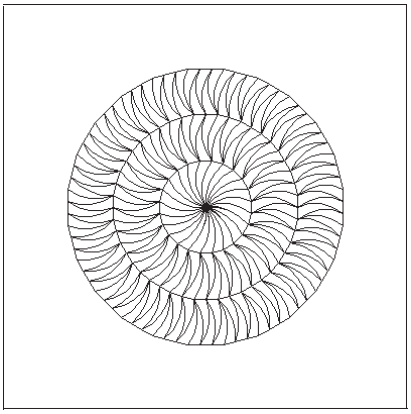


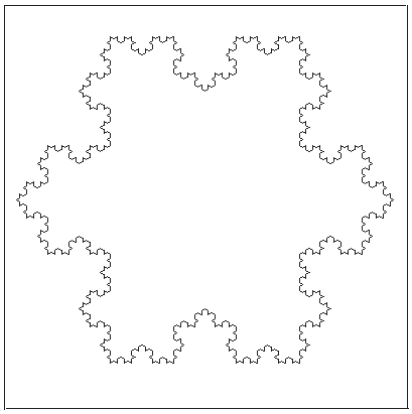


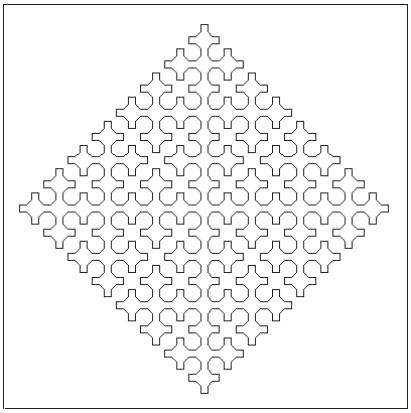


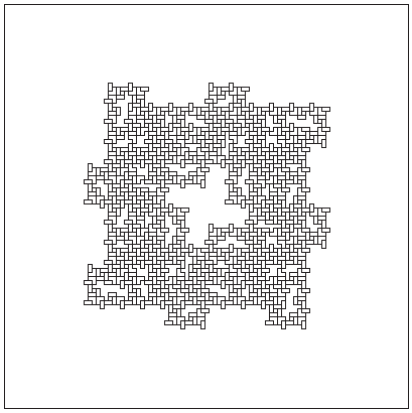


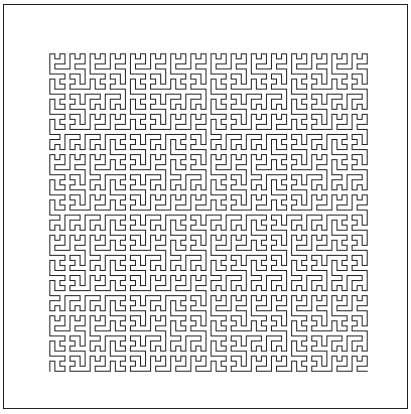
Although the most well-kown drawings produced by L-Systems are those of plants, L-systems also can be used to draw fractals.

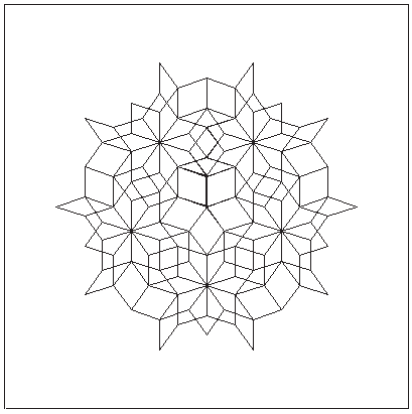












T-Sequence Expressions

The last section showed showed how L-Systems. with appropriate interpretation, can be used to produce pictures. A very different kind of interpretation can be used to produce t-sequence expressions. [Possible problem of reference order.]

A t-sequence expression with undefined variables represents all the possible t-sequences that can be produced by giving all possible values to the undefined variables during interpretation.

The usefulness of this idea is illustrated by the following examples.

Example 1

seed: S

rules: S ➛ pal(T)
T ➛ motif(X,V)
X ➛ hor(Y)
Y ➛ motif(U,V))

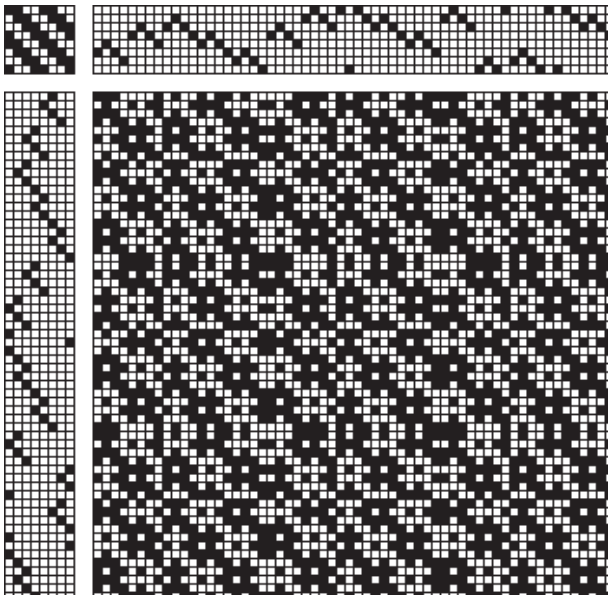
The terminal generation is

pal(motif(hor(motif(U,V)),V))

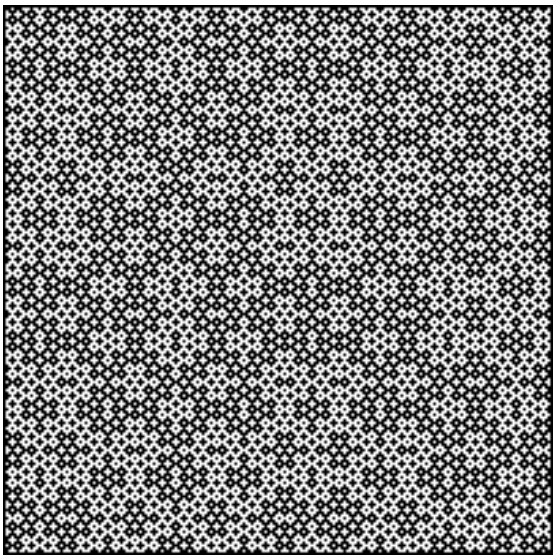
Given the values

U := [1,2,3,2]
V := [1,3, 5, 4, 2]

a draft based on the resulting sequence is:



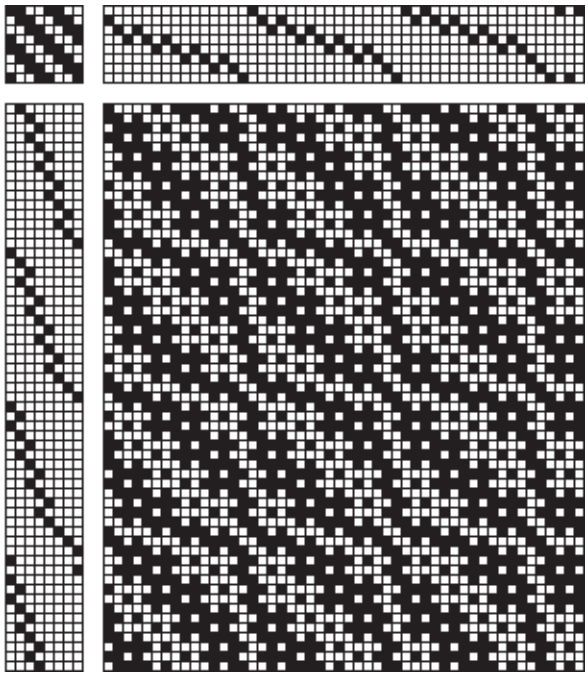
Here is the weave pattern:



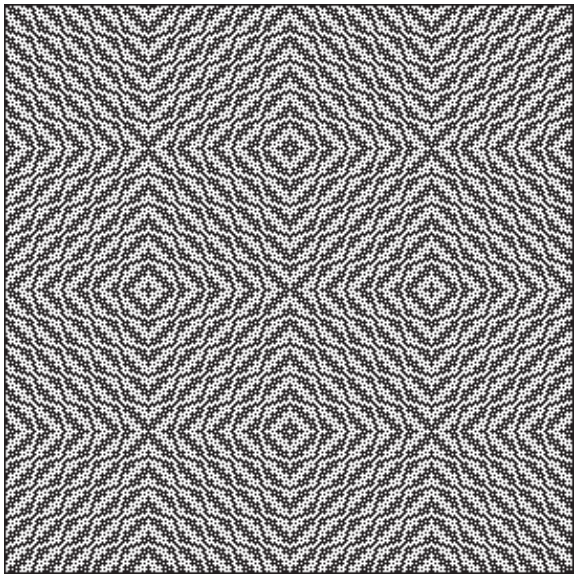
On the other hand, given the values

U := [1,2,3]
V := [1,2,3,4,5]

a draft based on the resulting sequence is:



Here is the weave pattern:



Example 2

seed: S
rules: S → pal(T)
T → coll(U,V)
U → pal(X)
V → pal(Y)

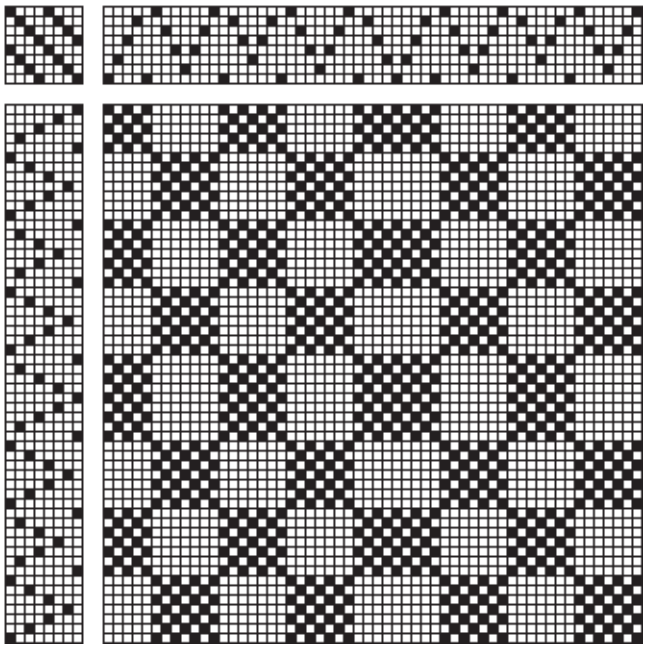
The terminal generation is

pal(coll(pal(X),pal(Y)))

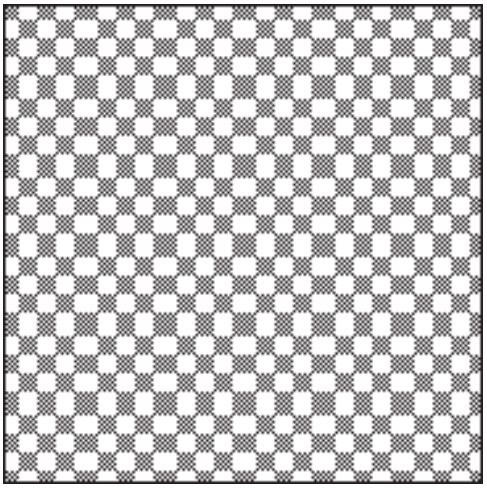
Given the values

X := [1,3,5,7,9,8,6,4,2]
Y := [6,4,2,7,5,3]

a draft based on the resulting sequence is:



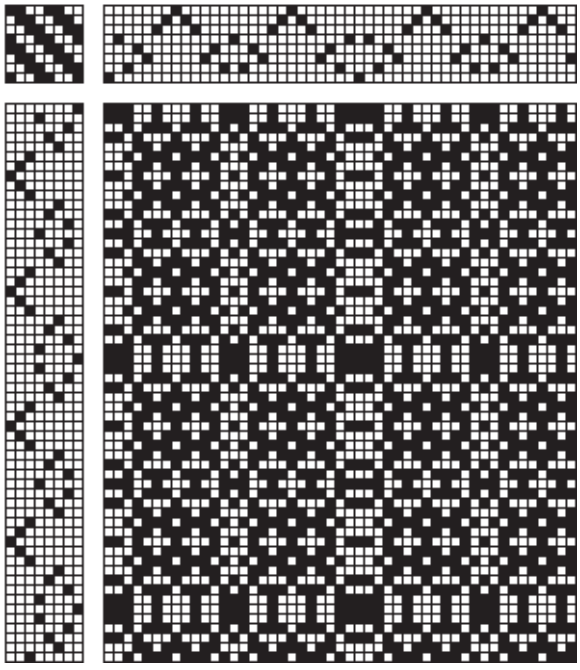
Here is the weave pattern:



On the other hand, given the values

X := [1,5,2,4,3,6,7,8]
Y := [6,4,2,7,5,3]

a draft based on the resulting sequence is:



Here is the weaving pattern:

Resources

Many L-System programs are available on the Web as freeware and shareware programs. Almost all of them are designed to produce images — to such an extent that a person who didn't know otherwise might assume that's all there is to L-Systems.

Lparser [2] is a particularly capable freeware L-System application. Here is an "air horse", which shows what is possible using Lparser: