



Generating T-Sequence Expressions

The section describes L-Systems that generate t-sequence expressions, as opposed to actual t-sequences.

This example illustrates the idea:

seed: S
 rules: S \rightarrow pal(T)
 T \rightarrow rpt(U,I)

This is a terminating L-System with only two generations:

pal(T)
 pal(rpt(U,N))

The interpretation of these strings, as the characters used suggest, is that pal is a function that produces a palindrome from its argument and rpt is a function that produces a repeat of its first argument a number of times specified by its second argument. Note that although p, a, l, r, and t, are individual constant characters in the L-System, pal and rpt can be treated as strings during interpretation.

If U is given the value [1, 2, 3, 4] and N the value 2 during interpretation, the result is

[1, 2, 3, 4, 1, 2, 3, 4, 3, 2, 1, 4, 3, 2, 1]

The L-System above could, of course, be derived from its final generation:

pal(rpt(U,N))

The value of using an L-System to characterize the patterns rather than just using the t-sequence expression it produces is that the components are represented in separate rules and hence easy to understand, while t-sequence expressions may be complicated and deeply nested, which is difficult for human beings (but not computer programs) to understand, and may be difficult to construct by hand.

The sections on t-sequences cast operations in an abstract operator notation using a variety of mathematical symbols and typographical devices.

For example, the expression from the example above,

pal(rpt(U,N))

is written in the abstract operator notation as

$\cap(U \times n)$

T-Sequence Models

The last section showed how terminal L-Systems can be used to characterize t-sequences in terms of t-sequence expressions.

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

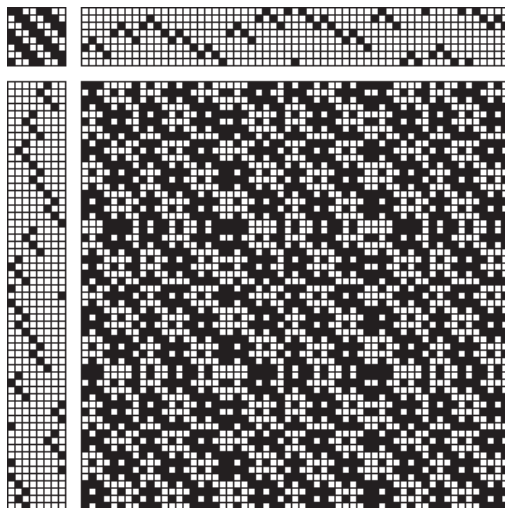
$$\text{pal}(\text{motif}(\text{hor}(\text{motif}(U,V)),V))$$

Given the values

$$U := [1,2,3,2]$$

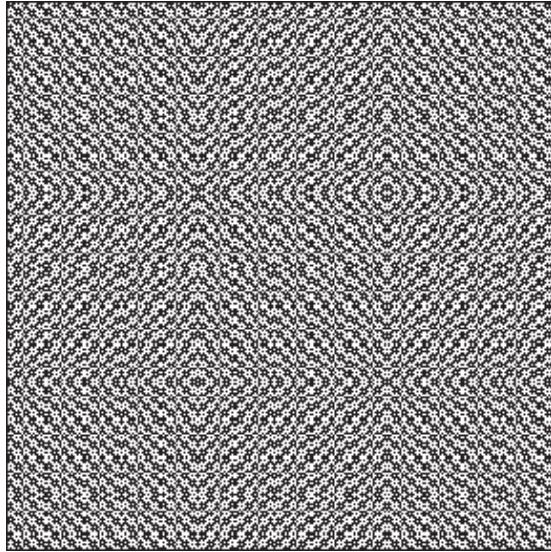
$$V := [1,3, 5, 4, 2]$$

a draft based on the resulting sequence is:





Here is the weave pattern:

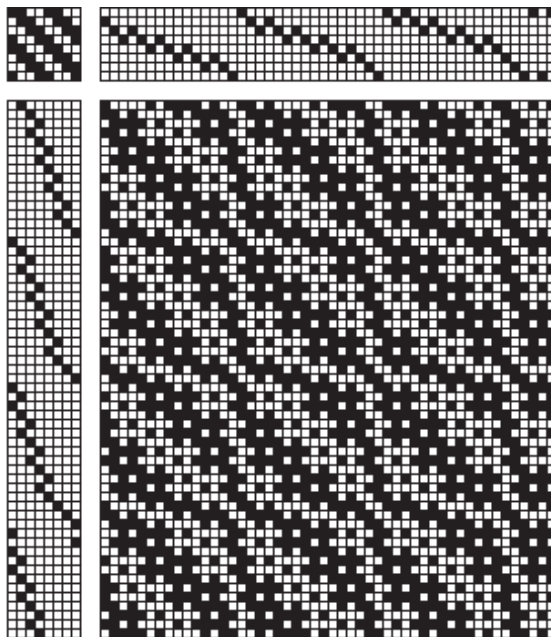


On the other hand, given the values

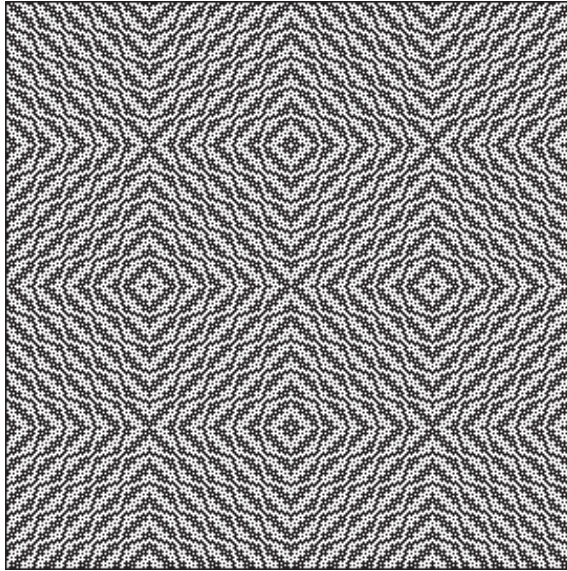
$$U := [1,2,3]$$

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

a draft based on the resulting sequence is:



Here is the weave pattern:



Example 2

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

The terminal generation is

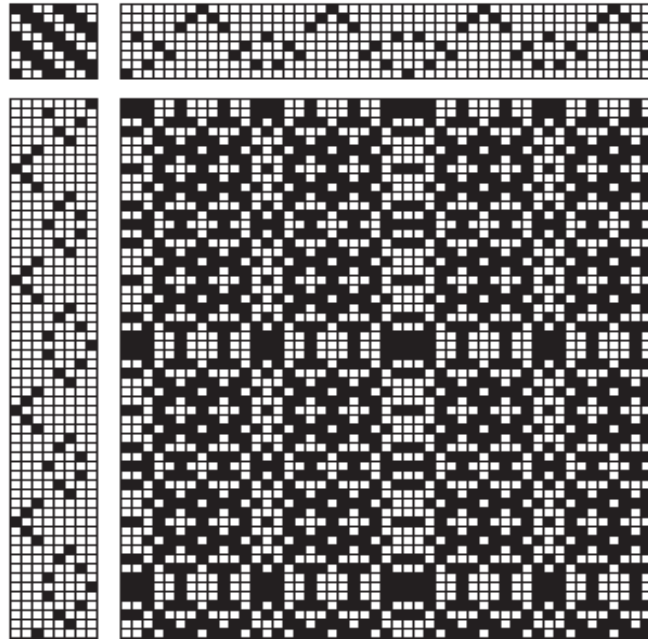
pal(coll(pal(X),pal(Y)))

Given the values

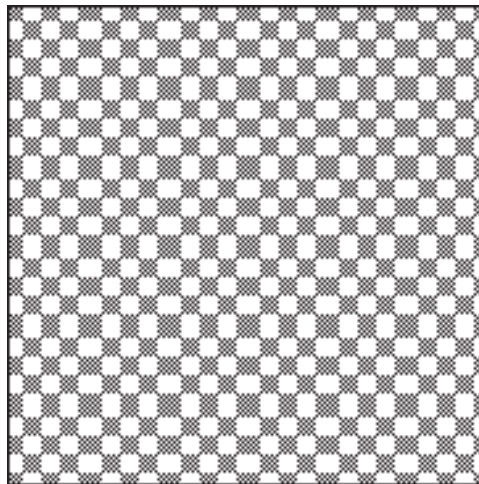
X := [1,3,5,7,9,8,6,4,2]

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

a draft based on the resulting sequence is:



Here is the weave pattern:

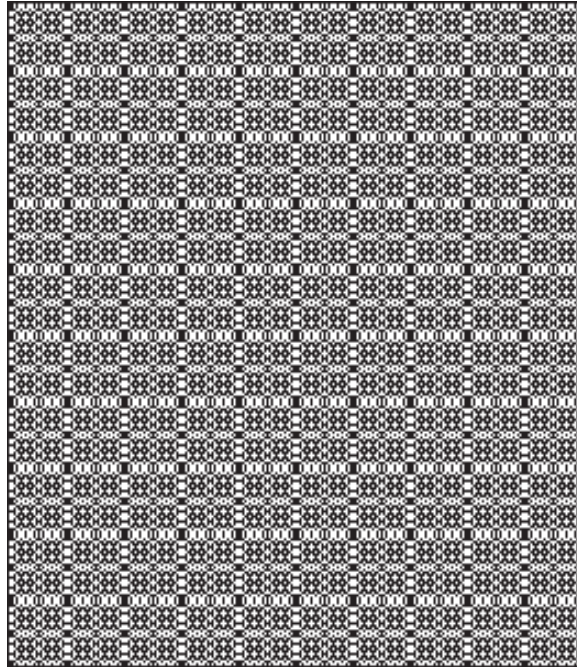


On the other hand, given the values

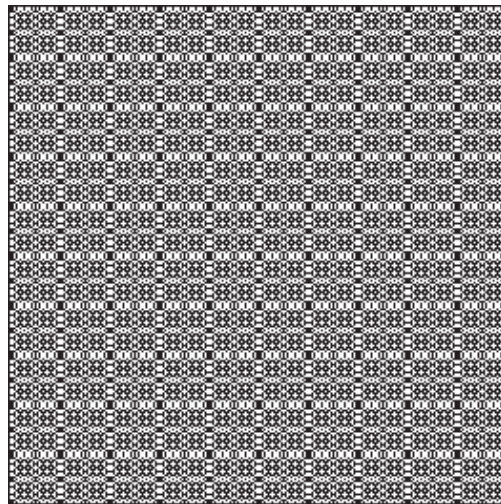
$$X := [1,5,2,4,3,6,7,8]$$

$$Y := [6,4,2,7,5,3]$$

a draft based on the resulting sequence is:



Here is the weave pattern:



Of course, just for these two example L-Systems, there is an infinite number of sequences, not to mention how they are used in drafts.



Non-Terminal T-Sequence Generation

The examples of L-Systems for generating t-sequence expressions in previous sections all have been terminal. While terminal L-Systems provide useful models, they do not exploit the power of L-Systems to generate successively more complex and detailed patterns — in this case, t-sequence expressions.

Consider this L-System:

```
seed:   S
rules:  S → pal(T)
        T → motif(U,V)
        U → hor(T)
```

It generates t-sequences expressions that are palindromes containing motifs along paths with horizontal reflection. But since T and U are defined in terms of themselves, exactly what is going on is hardly clear. The first few generations are:

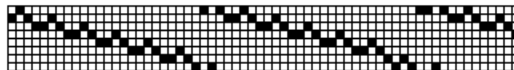
```
pal(T)
pal(motif(U,V))
pal(motif(hor(T),V))
pal(motif(hor(motif(U,V)),V))
...
```

The first three of these generations are comprehensible, but the last one is too intricate to comprehend; it is necessary to try examples and see what results.

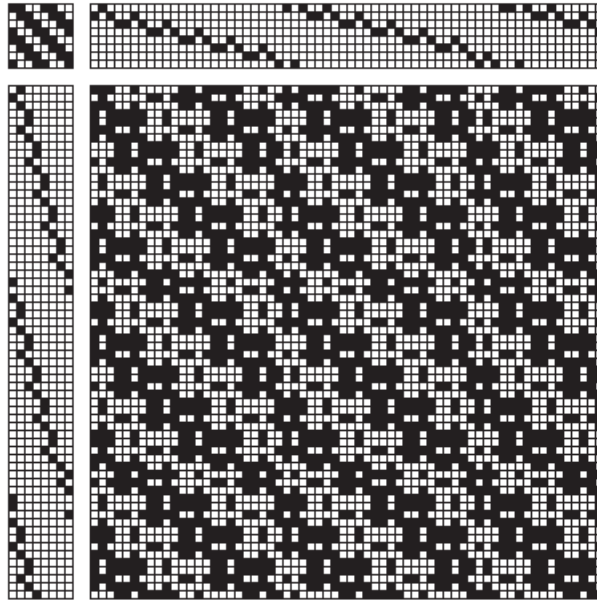
Suppose U and V have the values

```
U := [1,2,3,2]
V := [1,2,3,4,5,6]
```

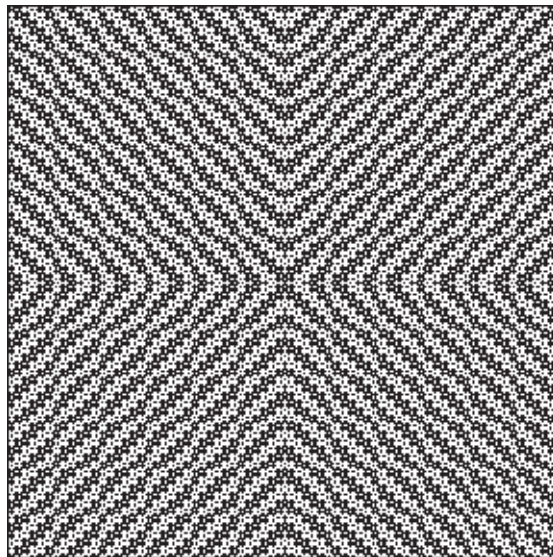
The resulting sequence starts like this:



A partial draft based on this sequence is:



and the weave is:



Here is another example of a non-terminating L-System:

seed: S
 rules: S \rightarrow pal(T)
 T \rightarrow coll(U,V)



U → pal(S)

V → ver(T)

The first few generations are:

```

pal(T)
pal(coll(U,V))
pal(coll(pal(S),vert(T)))
pal(coll(pal(pal(T)),vert(coll(U,V))))
...

```

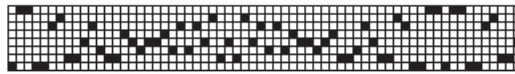
Suppose T, U, and V have the values

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

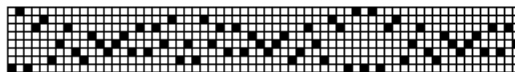
U := [1,1,2,2,3,3,4,4,5,5]

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

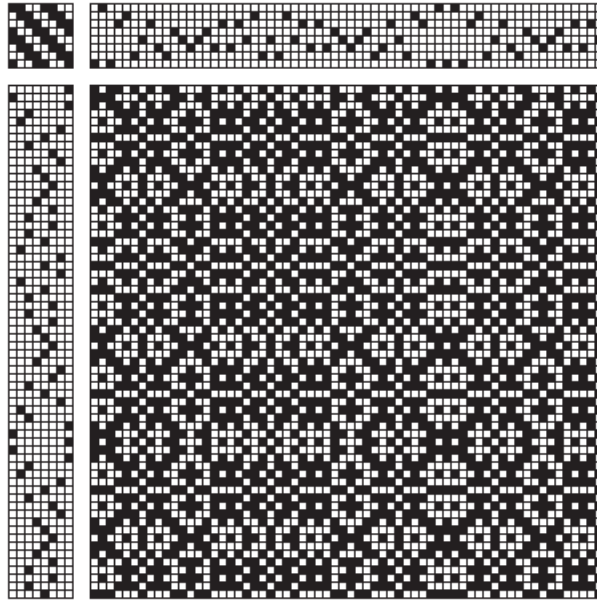
The resulting sequence starts like this:



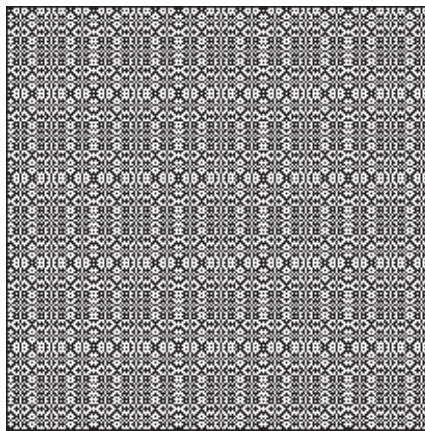
Here is a case of a sequence with adjacent duplicate values. Duplicates are not surprising, since the expression from which it was created is not comprehensible and hence the results unpredictable. Removing adjacent duplicates produces this:



A partial draft based on this sequence is:



and the weave is:



Of course other values for the variables give different results, sometimes very different results.

There are four problems with using L-Systems in the manner described above:

- It is difficult to design useful L-Systems.
- It is difficult to predict the results.
- It is difficult to assign useful values to undefined variables.



- The sequences produced quickly become impossibly long.

The last problem is the easiest to handle: Simply truncate long sequences so they are of a manageable length. Note the t-sequence expressions cannot be truncated; trying to do so generally results in invalid expressions.

The other three problems are essentially intractable if any degree of complexity it to be obtained. The best approach to problems like this is to try many alternatives, extract the useful results, and learn from the process.

For this to be practical, it is necessary to use computer programs.

