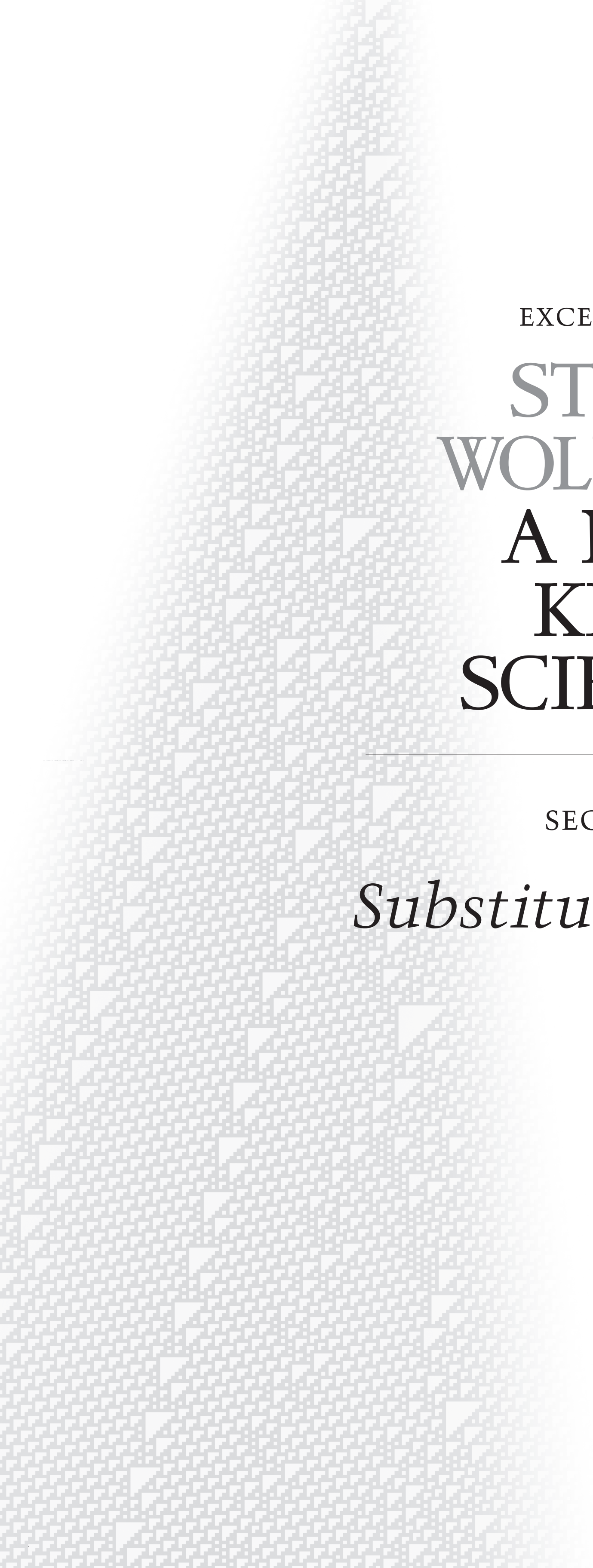


EXCERPTED FROM

STEPHEN WOLFRAM
A NEW KIND OF SCIENCE

SECTION 3.5

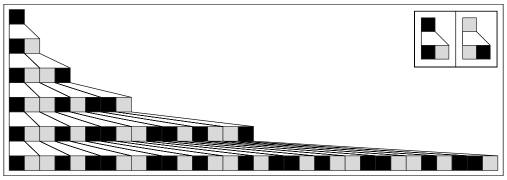
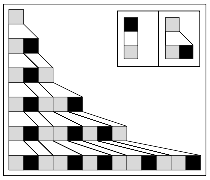
Substitution Systems

Substitution Systems

One of the features that cellular automata, mobile automata and Turing machines all have in common is that at the lowest level they consist of a fixed array of cells. And this means that while the colors of these cells can be updated according to a wide range of different possible rules, the underlying number and organization of cells always stays the same.

Substitution systems, however, are set up so that the number of elements can change. In the typical case illustrated below, one has a sequence of elements—each colored say black or white—and at each step each one of these elements is replaced by a new block of elements.

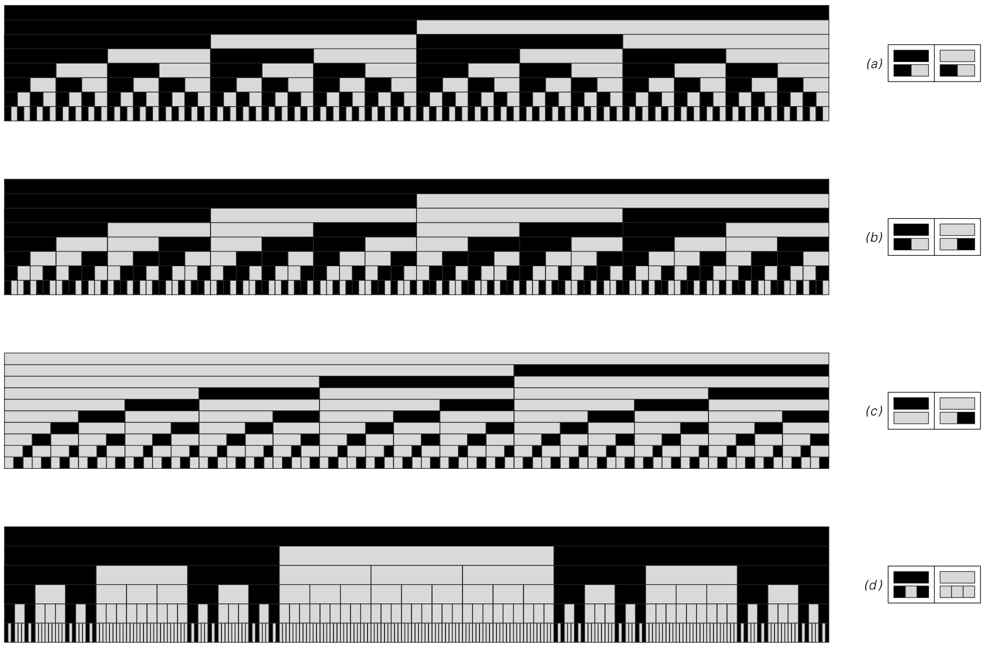
In the simple cases shown, the rules specify that each element of a particular color should be replaced by a fixed block of new elements, independent of the colors of any neighboring elements.

Examples of substitution systems with two possible kinds of elements, in which at every step each kind of element is replaced by a fixed block of new elements. In the first case shown, the total number of elements obtained doubles at every step; in the second case, it follows a Fibonacci sequence, and increases by a factor of roughly $(1+\sqrt{5})/2 \simeq 1.618$ at every step. The two substitution systems shown here correspond to the second and third examples in the pictures on the following two pages.

And with these kinds of rules, the total number of elements typically grows very rapidly, so that pictures like those above quickly become rather unwieldy. But at least for these kinds of rules, one can make clearer pictures by thinking of each step not as replacing every element by a sequence of elements that are drawn the same size, but rather of subdividing each element into several that are drawn smaller.

In the cases on the facing page, I start from a single element represented by a long box going all the way across the picture. Then on successive steps the rules for the substitution system specify how each box should be subdivided into a sequence of shorter and shorter boxes.

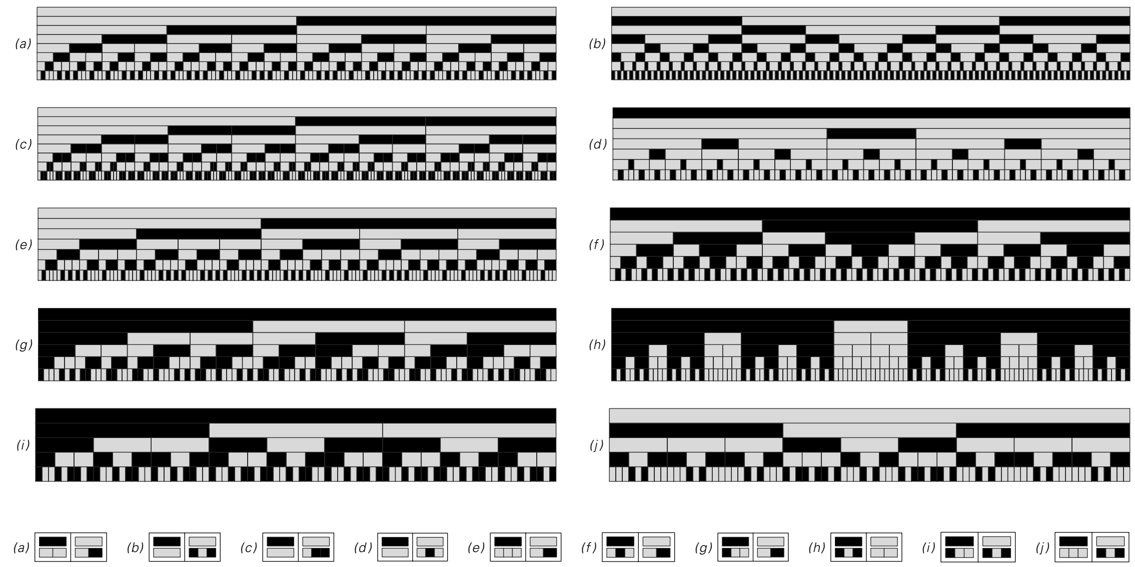


Examples of substitution systems in which every element is drawn as being subdivided into a sequence of new elements at each step. In all cases the overall patterns obtained can be seen to have a very regular nested form. Rule (b) gives the so-called Thue-Morse sequence, which we will encounter many times in this book. Rule (c) is related to the Fibonacci sequence. Rule (d) gives a version of the Cantor set.

The pictures at the top of the next page show a few more examples. And what we see is that in all cases there is obvious regularity in the patterns produced. Indeed, if one looks carefully, one can see that every pattern just consists of a collection of identical nested pieces.

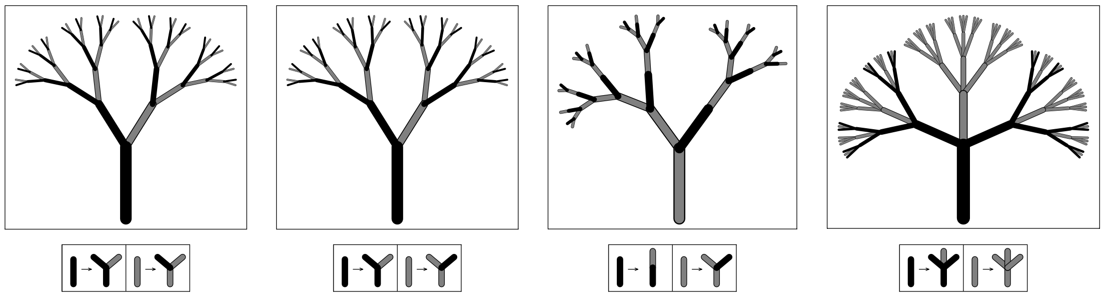
And ultimately this is not surprising. After all, the basic rules for these substitution systems specify that any time an element of a particular color appears it will always get subdivided in the same way.

The nested structure becomes even clearer if one represents elements not as boxes, but instead as branches on a tree. And with this setup the idea is to start from the trunk of the tree, and then at each step to use the rules for the substitution system to determine how every branch should be split into smaller branches.



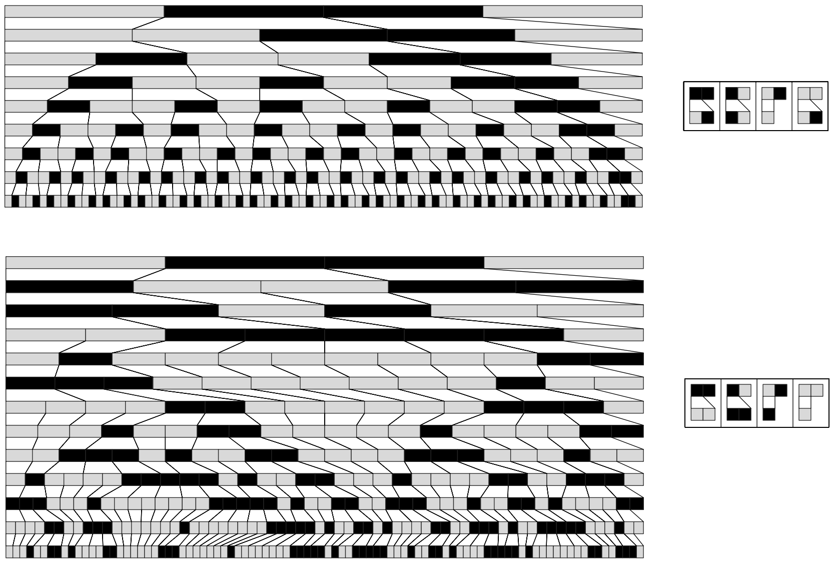
More examples of neighbor-independent substitution systems like those on the previous page. Each rule yields a different sequence of elements, but all of them ultimately have simple nested forms.

Then the point is that because the rules depend only on the color of a particular branch, and not on the colors of any neighboring branches, the subtrees that are generated from all the branches of the same color must have exactly the same structure, as in the pictures below.



The evolution of the same substitution systems as on the previous page, but now shown in terms of trees. Starting from the trunk at the bottom, the rules specify that at each step every branch of a particular color should split into smaller branches in the same way. The result is that each tree consists of a collection of progressively smaller subtrees with the same structure. On page 400 I will use similar systems to discuss the growth of actual trees and leaves.

To get behavior that is more complicated than simple nesting, it follows therefore that one must consider substitution systems whose rules depend not only on the color of a single element, but also on the color of at least one of its neighbors. The pictures below show examples in which the rules for replacing an element depend not only on its own color, but also on the color of the element immediately to its right.



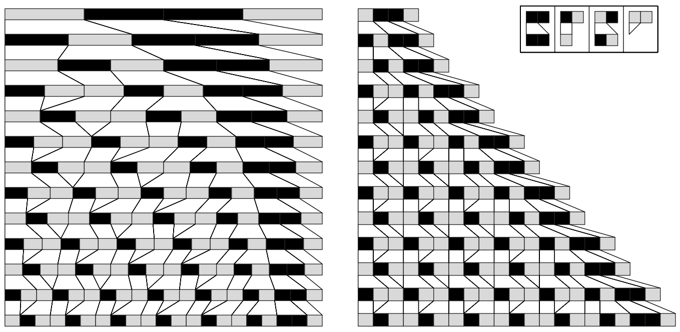
Examples of substitution systems whose rules depend not just on the color of an element itself, but also on the color of the element immediately to its right. Rules of this kind cannot readily be interpreted in terms of simple subdivision of one element into several. And as a result, there is no obvious way to choose what size of box should be used to represent each element in the picture. What I do here is simply to divide the whole width of the picture equally among all elements that appear at each step. Note that on every step the rightmost element is always dropped, since no rule is given for how to replace it.

In the first example, the pattern obtained still has a simple nested structure. But in the second example, the behavior is more complicated, and there is no obvious nested structure.

One feature of both examples, however, is that the total number of elements never decreases from one step to the next. The reason for this is that the basic rules we used specify that every single element should be replaced by at least one new element.

It is, however, also possible to consider substitution systems in which elements can simply disappear. If the rate of such disappearances is too large, then almost any pattern will quickly die out. And if there are too few disappearances, then most patterns will grow very rapidly.

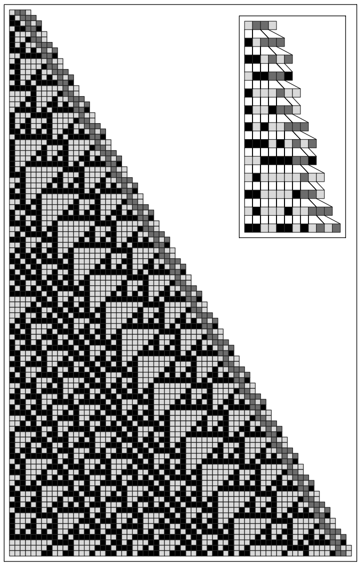
But there is always a small fraction of rules in which the creation and destruction of elements is almost perfectly balanced.

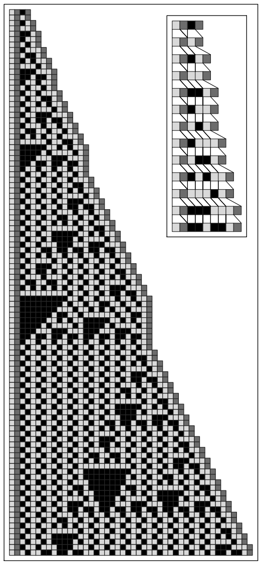

Two views of a substitution system whose rules allow both creation and destruction of elements. In the view on the left, the boxes representing each element are scaled to keep the total width the same, whereas on the right each box has a fixed size, as in our original pictures of substitution systems on page 82. The right-hand view shows that the rates of creation and destruction of elements are balanced closely enough that the total number of elements grows by only a fixed amount at each step.

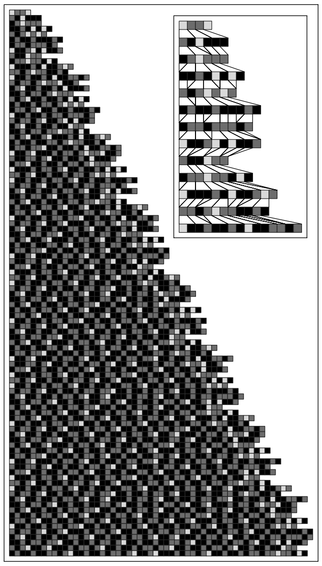
The picture above shows one example. The number of elements does end up increasing in this particular example, but only by a fixed amount at each step. And with such slow growth, we can again represent each element by a box of the same size, just as in our original pictures of substitution systems on page 82.

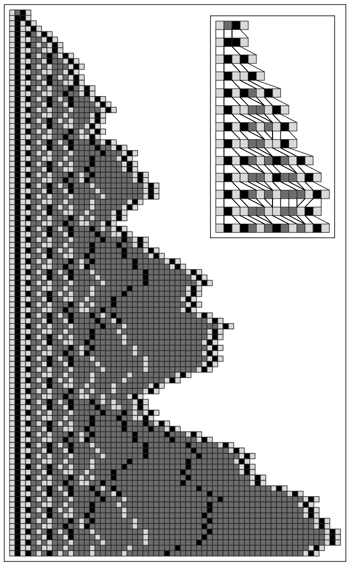
When viewed in this way, however, the pattern produced by the substitution system shown above is seen to have a simple repetitive form. And as it turns out, among substitution systems with the same type of rules, all those which yield slow growth also seem to produce only such simple repetitive patterns.

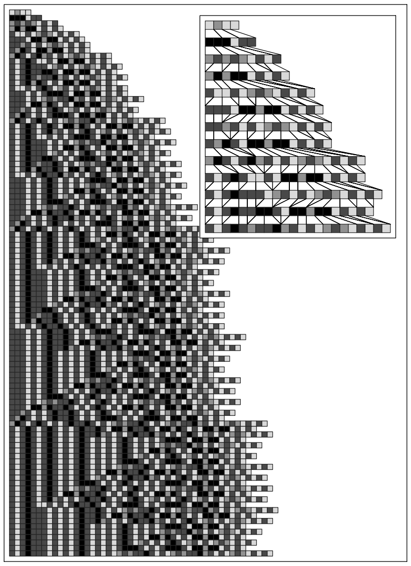
Knowing this, we might conclude that somehow substitution systems just cannot produce the kind of complexity that we have seen in systems like cellular automata. But as with mobile automata and with Turing machines, we would again be wrong. Indeed, as the pictures on the facing page demonstrate, allowing elements to have three or four colors rather than just two immediately makes much more complicated behavior possible.

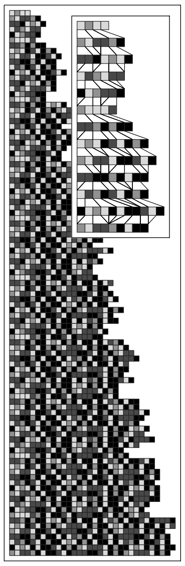

(a)

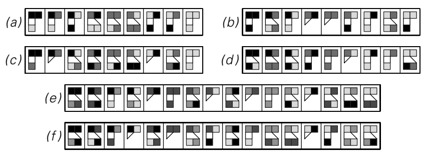

(b)


(c)


(d)


(e)


(f)



Examples of substitution systems that have three and four possible colors for each element. The particular rules shown are ones that lead to slow growth in the total number of elements. Note that on each line in each picture, only the order of elements is ever significant: as the insets show, a particular element may change its position as a result of the addition or subtraction of elements to its left. Note that the pattern in case (a) does eventually repeat, while the one in case (b) eventually shows a nested structure.

As it turns out, the first substitution system shown works almost exactly like a cellular automaton. Indeed, away from the right-hand edge, all the elements effectively behave as if they were lying on a regular grid, with the color of each element depending only on the previous color of that element and the element immediately to its right.

The second substitution system shown again has patches that exhibit a regular grid structure. But between these patches, there are regions in which elements are created and destroyed. And in the other substitution systems shown, elements are created and destroyed throughout, leaving no trace of any simple grid structure. So in the end the patterns we obtain can look just as random as what we have seen in systems like cellular automata.

Sequential Substitution Systems

None of the systems we have discussed so far in this chapter might at first seem much like computer programs of the kind we typically use in practice. But it turns out that there are for example variants of substitution systems that work essentially just like standard text editors.

The first step in understanding this correspondence is to think of substitution systems as operating not on sequences of colored elements but rather on strings of elements or letters. Thus for example the state of a substitution system at a particular step can be represented by the string ABBBABA, where the A's correspond to white elements and the B's to black ones.

The substitution systems that we discussed in the previous section work by replacing each element in such a string by a new sequence of elements—so that in a sense these systems operate in parallel on all the elements that exist in the string at each step.

But it is also possible to consider sequential substitution systems, in which the idea is instead to scan the string from left to right, looking for a particular sequence of elements, and then to perform a replacement for the first such sequence that is found. And this setup is now directly analogous to the search-and-replace function of a typical text editor.