EXCERPTED FROM

STEPHEN WOLFRAM

A NEW KIND OF SCIENCE

SECTION 3.6

Sequential Substitution Systems

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

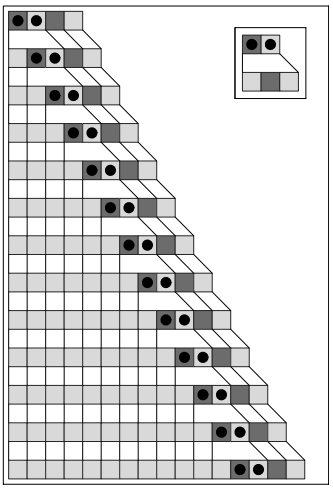
None of the systems we have discussed so far in this chapter might at first seem much like computer programs of the kind we typically use in practice. But it turns out that there are for example variants of substitution systems that work essentially just like standard text editors.

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.

The picture below shows an example of a sequential substitution system in which the rule specifies simply that the first sequence of the form BA found at each step should be replaced with the sequence ABA.

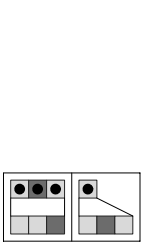


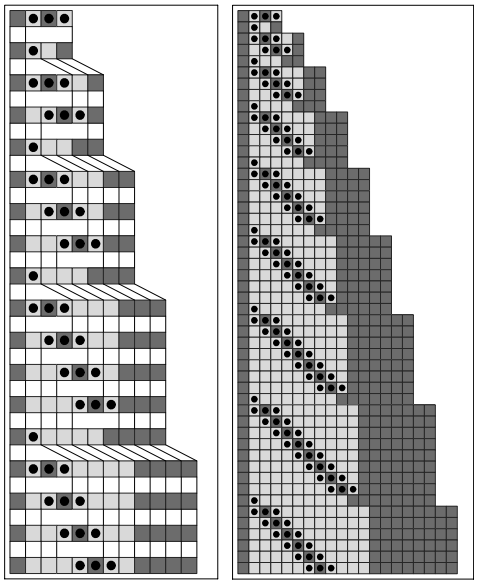
An example of a very simple sequential substitution system. The light squares can be thought of as corresponding to the element *A*, and the dark squares to the element *B*. At each step, the rule then specifies that the string which exists at that step should be scanned from left to right, and the first sequence *BA* that is found should be replaced by *ABA*. In the picture, the black dots indicate which elements are being replaced at each step. In the case shown, the initial string is *BABA*. At each step, the rule then has the effect of adding an *A* inside the string.

The behavior in this case is very simple, with longer and longer strings of the same form being produced at each step. But one can get more complicated behavior if one uses rules that involve more than just one possible replacement. The idea in this case is at each step to scan the string repeatedly, trying successive replacements on successive scans, and stopping as soon as a replacement that can be used is found.

The picture on the next page shows a sequential substitution system with rule {ABA → AAB, A → ABA} involving two possible replacements. Since the sequence ABA occurs in the initial string that is given, the first replacement is used on the first step. But the string BAAB that is produced at the second step does not contain ABA, so now the first replacement cannot be used. Nevertheless, since the string does contain the single element A, the second replacement can still be used.

Despite such alternation between different replacements, however, the final pattern that emerges is very regular. Indeed, if one allows only two possible replacements—and two possible elements—





A sequential substitution system whose rule involves two possible replacements. At each step, the whole string is scanned once to try to apply the first replacement, and is then scanned again if necessary to try to apply the second replacement.

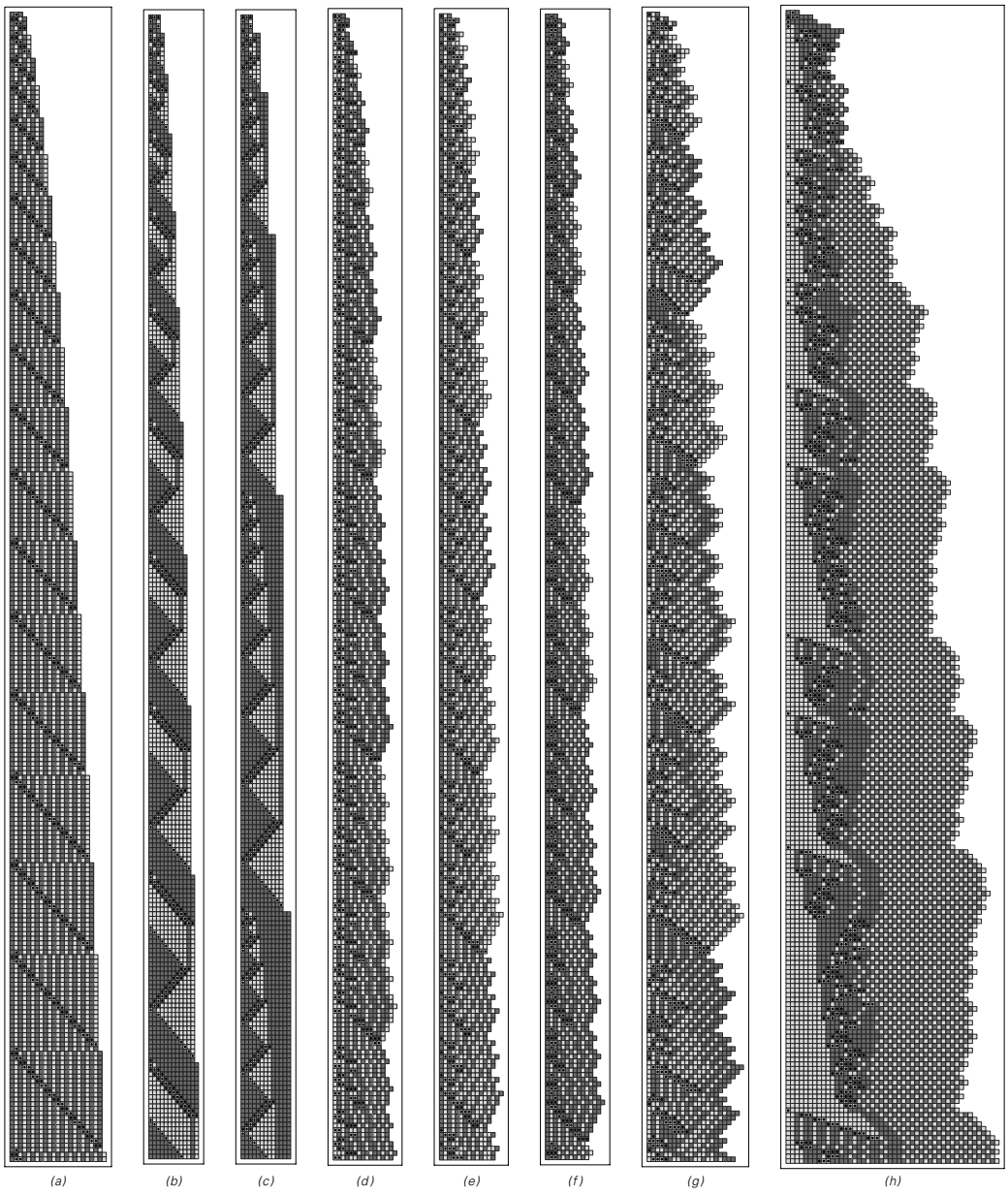
then it seems that no rule ever gives behavior that is much more complicated than in the picture above.

And from this one might be led to conclude that sequential substitution systems could never produce behavior of any substantial complexity. But having now seen complexity in many other kinds of systems, one might suspect that it should also be possible in sequential substitution systems.

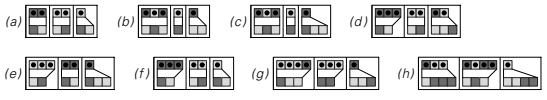
And it turns out that if one allows more than two possible replacements then one can indeed immediately get more complex behavior. The pictures on the facing page show a few examples. In many cases, fairly regular repetitive or nested patterns are still produced.

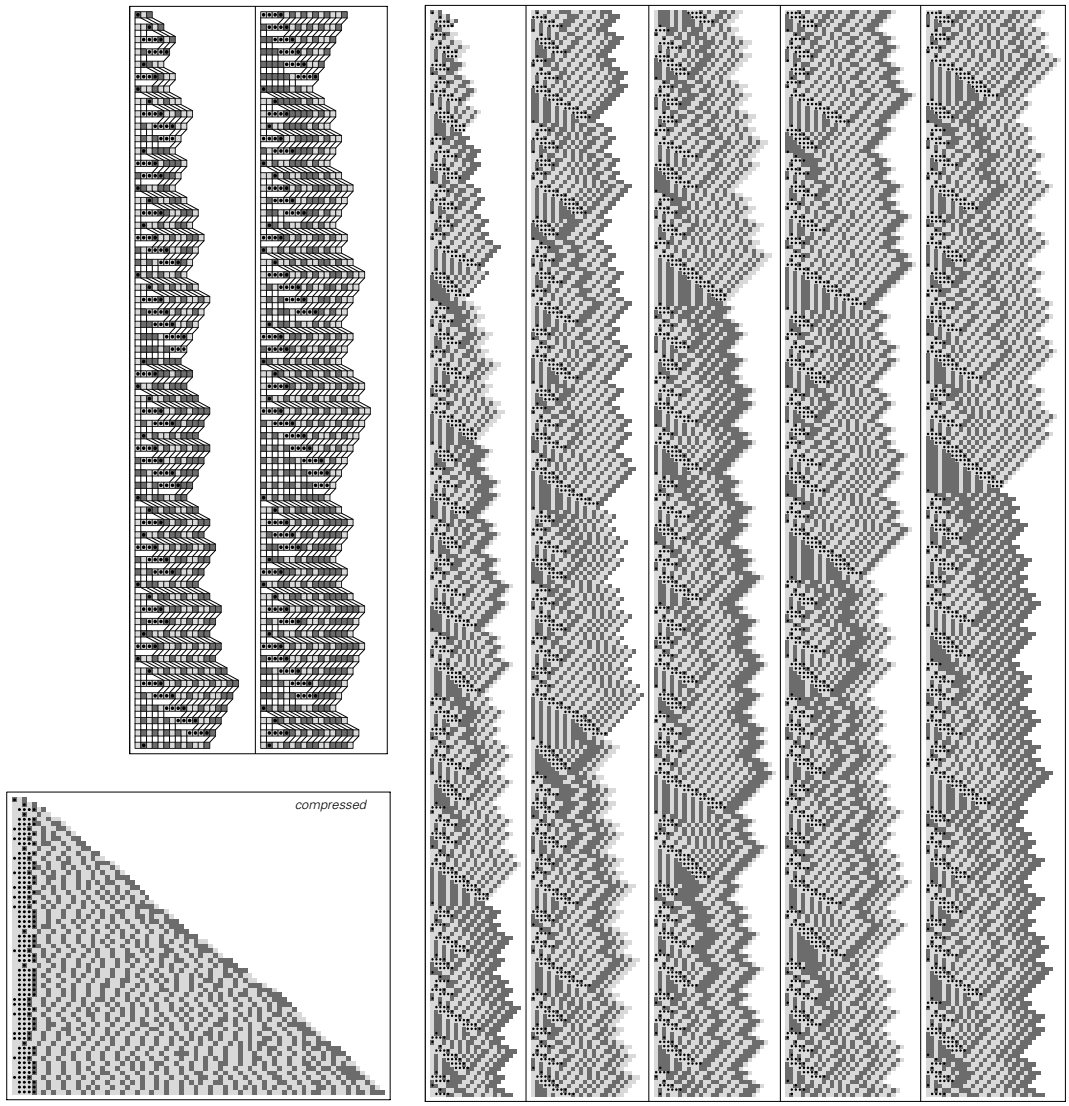
But about once in every 10,000 randomly selected rules, rather different behavior is obtained. Indeed, as the picture on the following page demonstrates, patterns can be produced that seem in many respects random, much like patterns we have seen in cellular automata and other systems.

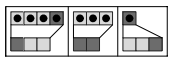
So this leads to the rather remarkable conclusion that just by using the simple operations available even in a very basic text editor, it is still ultimately possible to produce behavior of great complexity.



Examples of sequential substitution systems whose rules involve three possible replacements. In all cases, the systems are started from the initial string *BAB*. The black dots indicate the elements that are replaced at each step.







An example of a sequential substitution system that yields apparently random behavior. Each column on the right-hand side shows the evolution of the system for 250 steps. The compressed picture on the left is made by evolving for a million steps, but showing only steps at which the string becomes longer than it has ever been before. (The rule is the same as (g) on the previous page.)