EXCERPTED FROM

STEPHEN WOLFRAM

A NEW KIND OF SCIENCE

SECTION 7.5

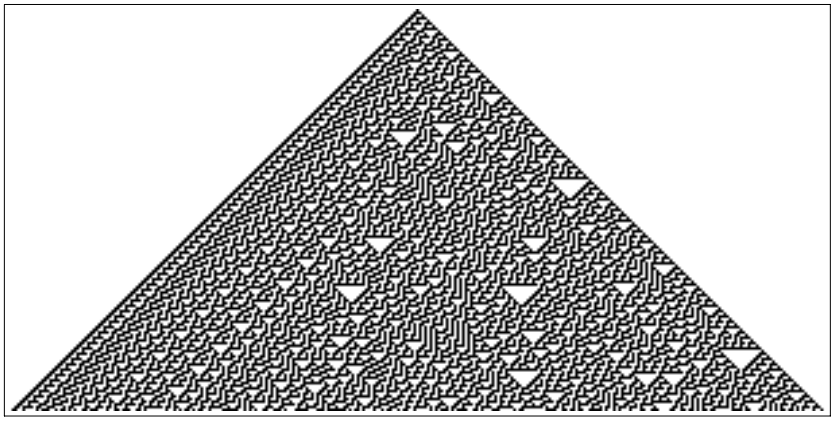
The Intrinsic Generation of Randomness

The Intrinsic Generation of Randomness

In the past two sections, we have studied two possible mechanisms that can lead to observed randomness. But as we have discussed, neither of these in any real sense themselves generate randomness. Instead, what they essentially do is just to take random input that comes from outside, and transfer it to whatever system one is looking at.

One of the important results of this book, however, is that there is also a third possible mechanism for randomness, in which no random input from outside is needed, and in which randomness is instead generated intrinsically inside the systems one is looking at.

The picture below shows the rule 30 cellular automaton in which I first identified this mechanism for randomness. The basic rule for the system is very simple. And the initial condition is also very simple.



The rule 30 cellular automaton from page 27 that was the first example I found of intrinsic randomness generation. There is no random input to this system, yet its behavior seems in many respects random. I suspect that this is how much of the randomness that we see in nature arises.

Yet despite the lack of anything that can reasonably be considered random input, the evolution of the system nevertheless intrinsically yields behavior which seems in many respects random.

As we have discussed before, traditional intuition makes it hard to believe that such complexity could arise from such a simple

underlying process. But the past several chapters have demonstrated that this is not only possible, but actually quite common.

Yet looking at the cellular automaton on the previous page there are clearly at least some regularities in the pattern it produces—like the diagonal stripes on the left. But if, say, one specifically picks out the color of the center cell on successive steps, then what one gets seems like a completely random sequence.

But just how random is this sequence really?

For our purposes here the most relevant point is that so far as one can tell the sequence is at least as random as sequences one gets from any of the phenomena in nature that we typically consider random.

When one says that something seems random, what one usually means in practice is that one cannot see any regularities in it. So when we say that a particular phenomenon in nature seems random, what we mean is that none of our standard methods of analysis have succeeded in finding regularities in it. To assess the randomness of a sequence produced by something like a cellular automaton, therefore, what we must do is to apply to it the same methods of analysis as we do to natural systems.

As I will discuss in Chapter 10, some of these methods have been well codified in standard mathematics and statistics, while others are effectively implicit in our processes of visual and other perception. But the remarkable fact is that none of these methods seem to reveal any real regularities whatsoever in the rule 30 cellular automaton sequence. And thus, so far as one can tell, this sequence is at least as random as anything we see in nature.

But is it truly random?

Over the past century or so, a variety of definitions of true randomness have been proposed. And according to most of these definitions, the sequence is indeed truly random. But there are a certain class of definitions which do not consider it truly random.

For these definitions are based on the notion of classifying as truly random only sequences which can never be generated by any simple procedure whatsoever. Yet starting with a simple initial condition and then applying a simple cellular automaton rule constitutes a simple

procedure. And as a result, the center column of rule 30 cannot be considered truly random according to such definitions.

But while definitions of this type have a certain conceptual appeal, they are not likely to be useful in discussions of randomness in nature. For as we will see later in this book, it is almost certainly impossible for any natural process ever to generate a sequence which is guaranteed to be truly random according to such definitions.

For our purposes more useful definitions tend to concentrate not so much on whether there exists in principle a simple way to generate a particular sequence, but rather on whether such a way can realistically be recognized by applying various kinds of analysis to the sequence. And as discussed above, there is good evidence that the center column of rule 30 is indeed random according to all reasonable definitions of this kind.

So whether or not one chooses to say that the sequence is truly random, it is, as far as one can tell, at least random for all practical purposes. And in fact sequences closely related to it have been used very successfully as sources of randomness in practical computing.

For many years, most kinds of computer systems and languages have had facilities for generating what they usually call random numbers. And in *Mathematica*—ever since it was first released—Random[Integer] has generated 0's and 1's using exactly the rule 30 cellular automaton.

The way this works is that every time Random[Integer] is called, another step in the cellular automaton evolution is performed, and the value of the cell in the center is returned. But one difference from the picture two pages ago is that for practical reasons the pattern is not allowed to grow wider and wider forever. Instead, it is wrapped around in a region that is a few hundred cells wide.

One consequence of this, as discussed on page 259, is that the sequence of 0's and 1's that is generated must then eventually repeat. But even with the fastest foreseeable computers, the actual period of repetition will typically be more than a billion billion times the age of the universe.

Another issue is that if one always ran the cellular automaton from page 315 with the particular initial condition shown there, then one would always get exactly the same sequence of 0's and 1's. But by using different initial conditions one can get completely different

sequences. And in practice if the initial conditions are not explicitly specified, what *Mathematica* does, for example, is to use as an initial condition a representation of various features of the exact state of the computer system at the time when Random was first called.

The rule 30 cellular automaton provides a particularly clear and good example of intrinsic randomness generation. But in previous chapters we have seen many other examples of systems that also intrinsically produce apparent randomness. And it turns out that one of these is related to the method used since the late 1940s for generating random numbers in almost all practical computer systems.

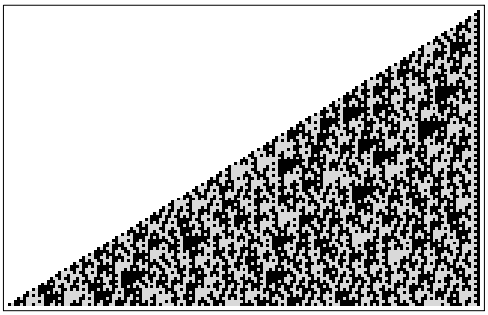
The pictures on the facing page show what happens if one successively multiplies a number by various constant factors, and then looks at the digit sequences of the numbers that result. As we first saw on page 119, the patterns of digits obtained in this way seem quite random. And the idea of so-called linear congruential random number generators is precisely to make use of this randomness.

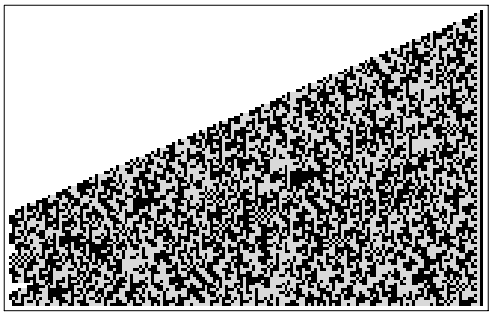
For practical reasons, such generators typically keep only, say, the rightmost 31 digits in the numbers at each step. Yet even with this restriction, the sequences generated are random enough that at least until recently they were almost universally what was used as a source of randomness in practical computing.

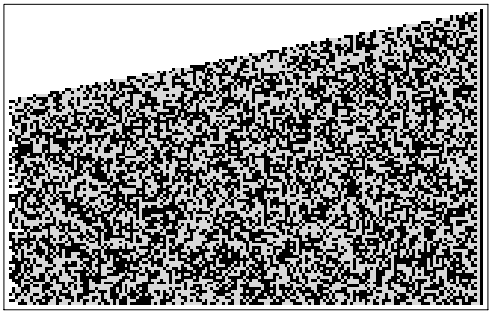
So in a sense linear congruential generators are another example of the general phenomenon of intrinsic randomness generation. But it turns out that in some respects they are rather unusual and misleading.

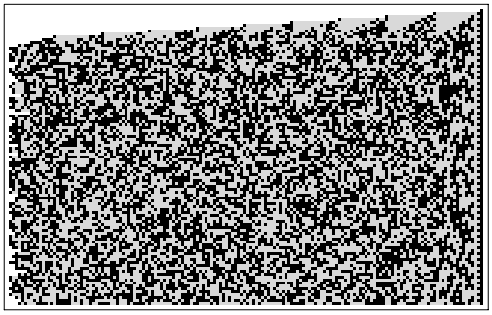
Keeping only a limited number of digits at each step makes it inevitable that the sequences produced will eventually repeat. And one of the reasons for the popularity of linear congruential generators is that with fairly straightforward mathematical analysis it is possible to tell exactly what multiplication factors will maximize this repetition period.

It has then often been assumed that having maximal repetition period will somehow imply maximum randomness in all aspects of the sequence one gets. But in practice over the years, one after another linear congruential generator that has been constructed to have maximal repetition period has turned out to exhibit very substantial deviations from perfect randomness.


multiplier 3


multiplier 5

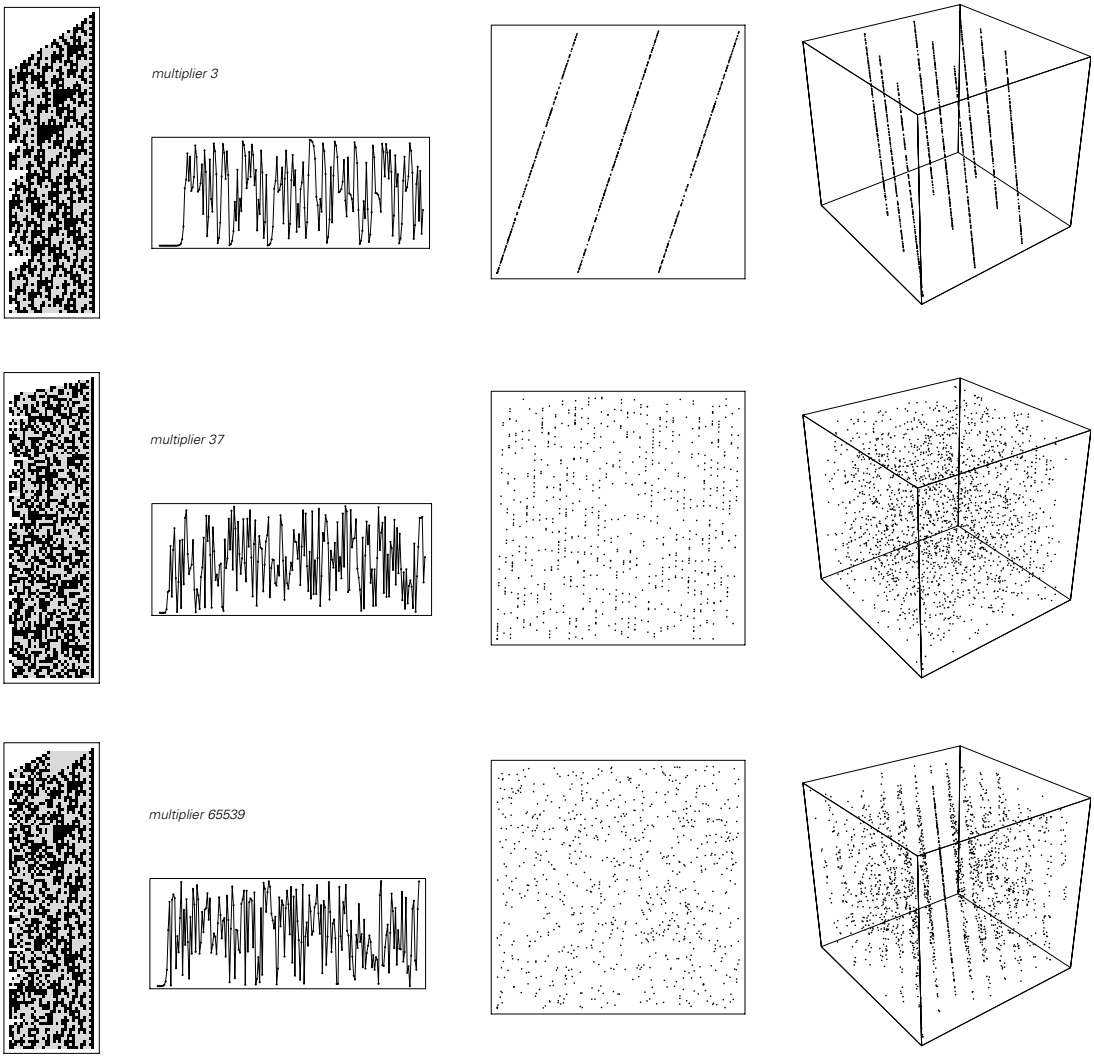

multiplier 37


multiplier 65539

Patterns of digits in base 2 produced by starting with the number 1 and then repeatedly multiplying by various fixed constants. In all cases, the complete pattern has a triangular form, but except in the first case, it is truncated on the left here. The mathematical structure of these systems is nevertheless such that digits further to the left do not affect those shown: at each step the number obtained is effectively reduced modulo 2^n, where n is the width of the picture.

A typical kind of failure, illustrated in the pictures on the next page, is that points with coordinates determined by successive numbers from the generator turn out to be distributed in an embarrassingly regular way. At first, such failures might suggest that more complicated schemes must be needed if one is to get good randomness. And indeed with this thought in mind all sorts of elaborate combinations of linear congruential and other generators have been proposed. But although some aspects of the behavior of such systems can be made quite random, deviations from perfect randomness are still often found.

And seeing this one might conclude that it must be essentially impossible to produce good randomness with any kind of system that has reasonably simple rules. But the rule 30 cellular automaton that we discussed above demonstrates that in fact this is absolutely not the case.



Examples of three so-called linear congruential random number generators. In each case they start with the number 1, then successively multiply by the specified multiplier, keeping only the rightmost 31 digits in the base 2 representation of the number obtained at each step. A version of the case with multiplier 3 was already shown on page 120. Multiplier 65539 was used as the random number generator on many computer systems, starting with mainframes in the 1960s. The last two pictures in each row above give the distribution of points whose coordinates in two and three dimensions are obtained by taking successive numbers from the linear congruential generator. If the output from the generator was perfectly random, then in each case these points would be uniformly distributed. But as the pictures demonstrate, stripes are visible in either two or three dimensions, or both.

Indeed, the rules for this cellular automaton are in some respects much simpler than for even a rather basic linear congruential generator. Yet the sequences it produces seem perfectly random, and do not suffer from any of the problems that are typically found in linear congruential generators.

So why do linear congruential generators not produce better randomness? Ironically, the basic reason is also the reason for their popularity. The point is that unlike the rule 30 cellular automaton that we discussed above, linear congruential generators are readily amenable to detailed mathematical analysis. And as a result, it is possible for example to guarantee that a particular generator will indeed have a maximal repetition period.

Almost inevitably, however, having such a maximal period implies a certain regularity. And in fact, as we shall see later in this book, the very possibility of any detailed mathematical analysis tends to imply the presence of at least some deviations from perfect randomness.

But if one is not constrained by the need for such analysis, then as we saw in the cellular automaton example above, remarkably simple rules can successfully generate highly random behavior.

And indeed the existence of such simple rules is crucial in making it plausible that the general mechanism of intrinsic randomness generations can be widespread in nature. For if the only way for intrinsic randomness generation to occur was through very complicated sets of rules, then one would expect that this mechanism would be seen in practice only in a few very special cases.

But the fact that simple cellular automaton rules are sufficient to give rise to intrinsic randomness generation suggests that in reality it is rather easy for this mechanism to occur. And as a result, one can expect that the mechanism will be found often in nature.

So how does the occurrence of this mechanism compare to the previous two mechanisms for randomness that we have discussed?

The basic answer, I believe, is that whenever a large amount of randomness is produced in a short time, intrinsic randomness generation is overwhelmingly likely to be the mechanism responsible.

We saw in the previous section that random details of the initial conditions for a system can lead to a certain amount of randomness in

the behavior of a system. But as we discussed, there is in most practical situations a limit on the lengths of sequences whose randomness can realistically be attributed to such a mechanism. With intrinsic randomness generation, however, there is no such limit: in the cellular automaton above, for example, all one need do to get a longer random sequence is to run the cellular automaton for more steps.

But it is also possible to get long random sequences by continual interaction with a random external environment, as in the first mechanism for randomness discussed in this chapter.

The issue with this mechanism, however, is that it can take a long time to get a given amount of good-quality randomness from it. And the point is that in most cases, intrinsic randomness generation can produce similar randomness in a much shorter time.

Indeed, in general, intrinsic randomness generation tends to be much more efficient than getting randomness from the environment. The basic reason is that intrinsic randomness generation in a sense puts all the components in a system to work in producing new randomness, while getting randomness from the environment does not.

Thus, for example, in the rule 30 cellular automaton discussed above, every cell in effect actively contributes to the randomness we see. But in a system that just amplifies randomness from the environment, none of the components inside the system itself ever contribute any new randomness at all. Indeed, ironically enough, the more components that are involved in the process of amplification, the slower it will typically be to get each new piece of random output. For as we discussed two sections ago, each component in a sense adds what one can consider to be more inertia to the amplification process.

But with a larger number of components it becomes progressively easier for randomness to be generated through intrinsic randomness generation. And indeed unless the underlying rules for the system somehow explicitly prevent it, it turns out in the end that intrinsic randomness generation will almost inevitably occur—often producing so much randomness that it completely swamps any randomness that might be produced from either of the other two mechanisms.

Yet having said this, one can ask how one can tell in an actual experiment on some particular system in nature to what extent intrinsic randomness generation is really the mechanism responsible for whatever seemingly random behavior one observed.

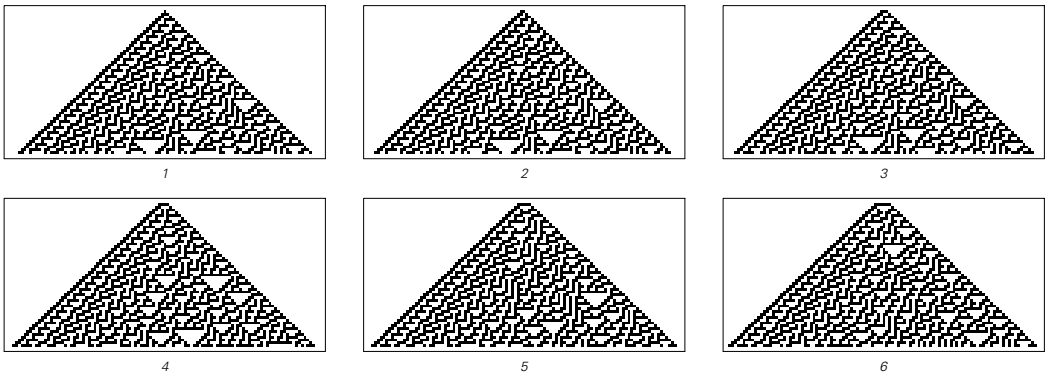
The clearest sign is a somewhat unexpected phenomenon: that details of the random behavior can be repeatable from one run of the experiment to another. It is not surprising that general features of the behavior will be the same. But what is remarkable is that if intrinsic randomness generation is the mechanism at work, then the precise details of the behavior can also be repeatable.

In the mechanism where randomness comes from continual interaction with the environment, no repeatability can be expected. For every time the experiment is run, the state of the environment will be different, and so the behavior one sees will also be correspondingly different. And similarly, in the mechanism where randomness comes from the details of initial conditions, there will again be little, if any, repeatability. For the details of the initial conditions are typically affected by the environment of the system, and cannot realistically be kept the same from one run to another.

But the point is that with the mechanism of intrinsic randomness generation, there is no dependence on the environment. And as a result, so long as the setup of the system one is looking at remains the same, the behavior it produces will be exactly the same. Thus for example, however many times one runs a rule 30 cellular automaton, starting with a single black cell, the behavior one gets will always be exactly the same. And so for example the sequence of colors of the center cell, while seemingly random, will also be exactly the same.

But how easy is it to disturb this sequence? If one makes a fairly drastic perturbation, such as changing the colors of cells all the way from white to black, then the sequence will indeed often change, as illustrated in the pictures at the top of the next page.

But with less drastic perturbations, the sequence can be quite robust. As an example, one can consider allowing each cell to be not just black or white, but any shade of gray, as in the continuous cellular automata we discussed on page 155. And in such systems, one can



The effect of changing the number of initial black cells in the rule 30 cellular automaton shown above. With only 2 or 3 black cells, the sequence in the center of the pattern does not change. But as soon as more black cells are added, it does change.

investigate what happens if at every step one randomly perturbs the gray level of each cell by a small amount.

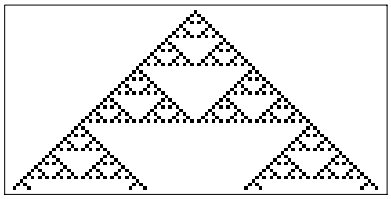
The pictures on the facing page show results for perturbations of various sizes. What one sees is that when the perturbations are sufficiently large, the sequence of colors of the center cell does indeed change. But the crucial point is that for perturbations below a certain critical size, the sequence always remains essentially unchanged.

Even though small perturbations are continually being made, the evolution of the system causes these perturbations to be damped out, and produces behavior that is in practice indistinguishable from what would be seen if there were no perturbations.

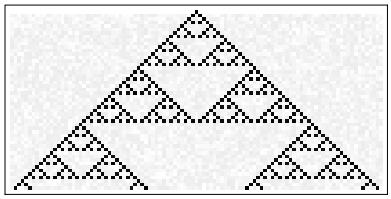
The question of what size of perturbations can be tolerated without significant effect depends on the details of the underlying rules. And as the pictures suggest, rules which yield more complex behavior tend to be able to tolerate only smaller sizes of perturbations. But the crucial point is that even when the behavior involves intrinsic randomness generation, perturbations of at least some size can still be tolerated.

And the reason this is important is that in any real experiment, there are inevitably perturbations on the system one is looking at.

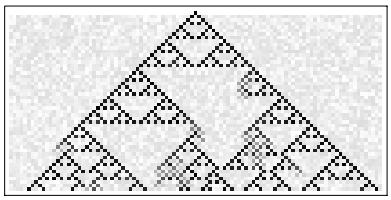
With more care in setting up the experiment, a higher degree of isolation from the environment can usually be achieved. But it is never possible to eliminate absolutely all interaction with the environment.



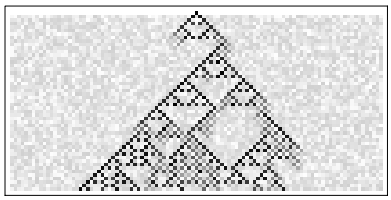
0% perturbations



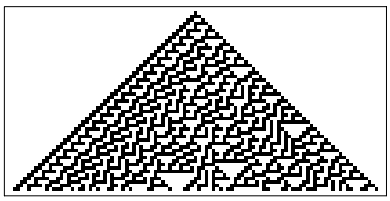
5% perturbations



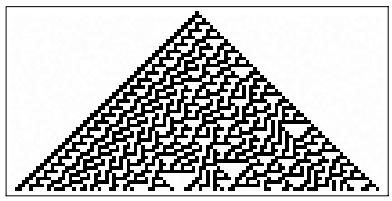
10% perturbations



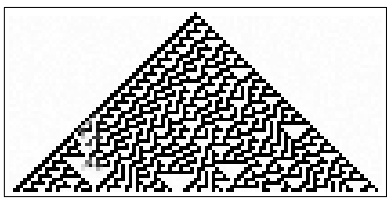
15% perturbations



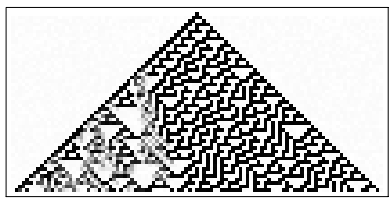
0% perturbations



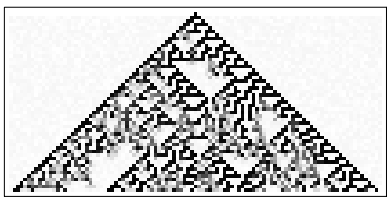
0.5% perturbations



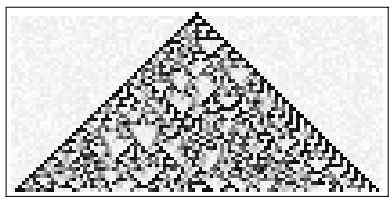
0.8% perturbations



1% perturbations

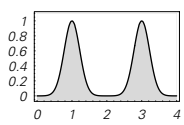


2% perturbations



5% perturbations

The effects of various levels of external randomness on the behavior of continuous cellular automata with generalizations of rules 90 and 30. The value of each cell can be any gray level between 0 and 1. For the generalization of rule 90, the values of the left and right cells are added together, and the value of the cell on the next step is then found by applying the continuous generalization of the modulo 2 function shown at the right. For the generalization of rule 30, a similar scheme based on an algebraic representation of the rule is used. In both cases, every value at each step is also perturbed by a random amount up to the percentage indicated for each picture.



And as a result, the system one is looking at will be subjected to at least some level of random perturbations from the environment.

But what the pictures on the previous page demonstrate is that when such perturbations are small enough, they will have essentially no effect. And what this means is that when intrinsic randomness generation is the dominant mechanism it is indeed realistic to expect at least some level of repeatability in the random behavior one sees in real experiments.

So has such repeatability actually been seen in practice?

Unfortunately there is so far very little good information on this point, since without the idea of intrinsic randomness generation there was never any reason to look for such repeatability when behavior that seemed random was observed in an experiment.

But scattered around the scientific literature—in various corners of physics, chemistry, biology and elsewhere—I have managed to find at least some cases where multiple runs of the same carefully controlled experiment are reported, and in which there are clear hints of repeatability even in behavior that looks quite random.

If one goes beyond pure numerical data of the kind traditionally collected in scientific experiments, and instead looks for example at the visual appearance of systems, then sometimes the phenomenon of repeatability becomes more obvious. Indeed, for example, as I will discuss in Chapter 8, different members of the same biological species often have many detailed visual similarities—even in features that on their own seem complex and apparently quite random.

And when there are, for example, two symmetrical sides to a particular system, it is often possible to compare the visual patterns produced on each side, and see what similarities exist. And as various examples in Chapter 8 demonstrate, across a whole range of physical, biological and other systems there can indeed be remarkable similarities.

So in all of these cases the randomness one sees cannot reasonably be attributed to randomness that is introduced from the environment—either continually or through initial conditions. And instead, there is no choice but to conclude that the randomness must in fact come from the mechanism of intrinsic randomness generation that I have discovered in simple programs, and discussed in this section.