EXCERPTED FROM

STEPHEN WOLFRAM
A NEW KIND OF SCIENCE

CHAPTER 6

Starting from Randomness

Starting from Randomness

The Emergence of Order

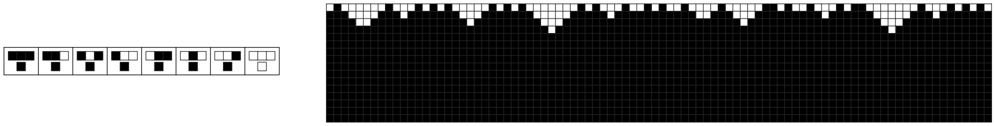
In the past several chapters, we have seen many examples of behavior that simple programs can produce. But while we have discussed a whole range of different kinds of underlying rules, we have for the most part considered only the simplest possible initial conditions—so that for example we have usually started with just a single black cell.

My purpose in this chapter is to go to the opposite extreme, and to consider completely random initial conditions, in which, for example, every cell is chosen to be black or white at random.

One might think that starting from such randomness no order would ever emerge. But in fact what we will find in this chapter is that many systems spontaneously tend to organize themselves, so that even with completely random initial conditions they end up producing behavior that has many features that are not at all random.

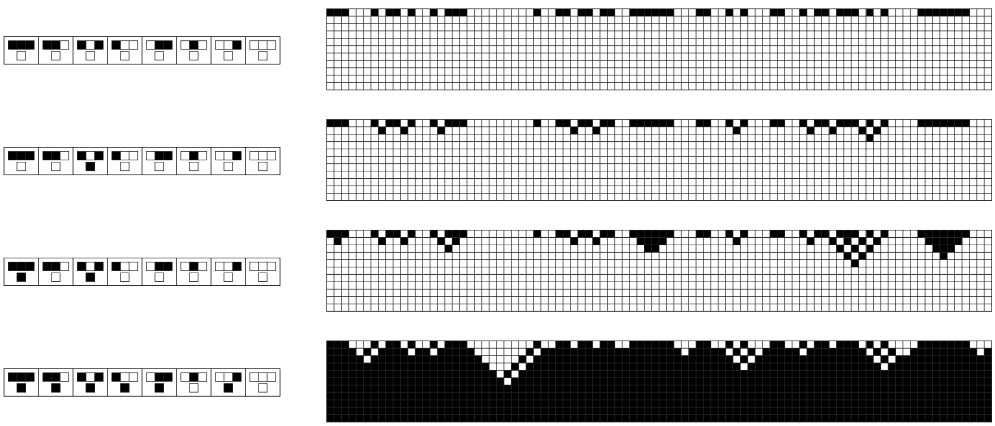
The picture at the top of the next page shows as a simple first example a cellular automaton which starts from a typical random initial condition, then evolves down the page according to the very simple rule that a cell becomes black if either of its neighbors are black.

What the picture then shows is that every region of white that exists in the initial conditions progressively gets filled in with black, so that in the end all that remains is a uniform state with every cell black.



A cellular automaton that evolves to a simple uniform state when started from any random initial condition. The rule in this case was first shown on page 24, and is number 254 in the scheme described on page 53. It specifies that a cell should become black whenever either of its neighbors is already black.

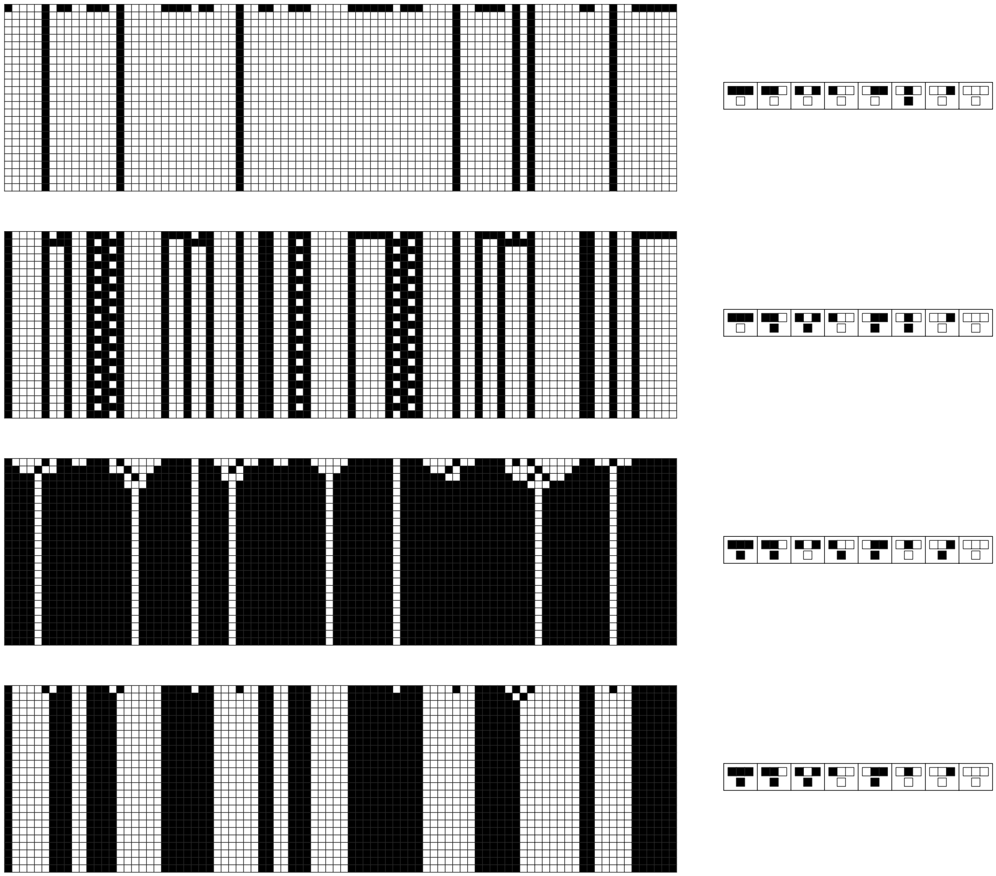
The pictures below show examples of other cellular automata that exhibit the same basic phenomenon. In each case the initial conditions are random, but the system nevertheless quickly organizes itself to become either uniformly white or uniformly black.



Four more examples of cellular automata that evolve from random initial conditions to completely uniform states. The rules shown here correspond to numbers 0, 32, 160 and 250.

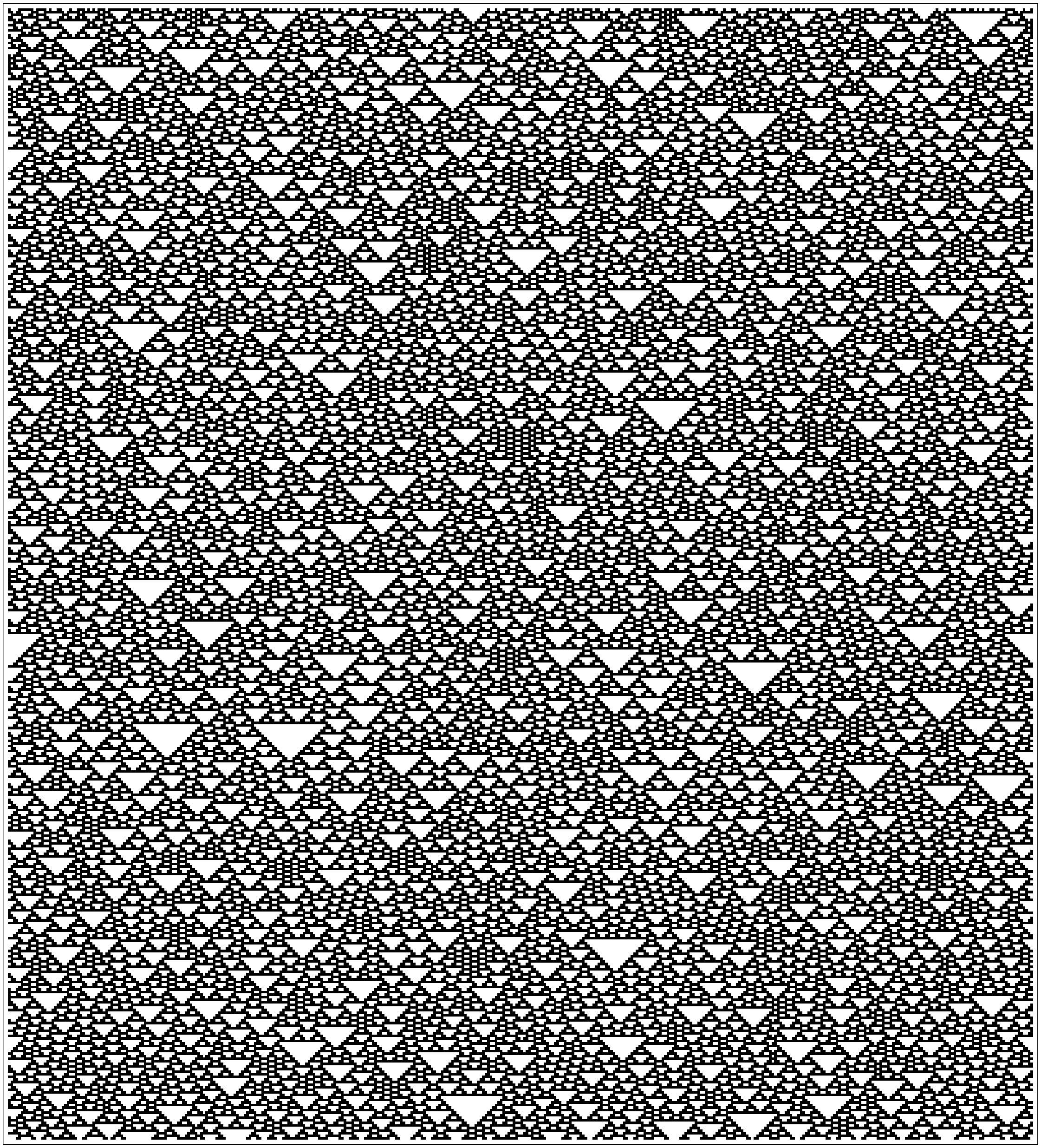
The facing page shows cellular automata that exhibit slightly more complicated behavior. Starting from random initial conditions, these cellular automata again quickly settle down to stable states. But now these stable states are not just uniform in color, but instead involve a collection of definite structures that either remain fixed on successive steps, or repeat periodically.

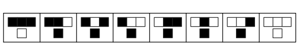
So if they have simple underlying rules, do all cellular automata started from random initial conditions eventually settle down to give stable states that somehow look simple?



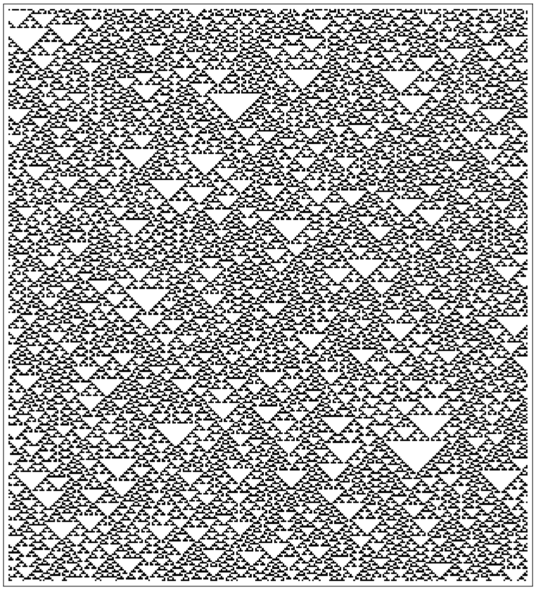
Examples of cellular automata that evolve from random initial conditions to produce a definite set of simple structures. For any particular rule, the form of these structures is always the same. But their positions depend on the details of the initial conditions given, and in many cases the final arrangement of structures can be thought of as a kind of filtered version of the initial conditions. Thus for example in the first rule shown here a structure consisting of a black cell occurs wherever there was an isolated black cell in the initial conditions. The rules shown are numbers 4, 108, 218 and 232.

It turns out that they do not. And indeed the picture on the next page shows one of many examples in which starting from random initial conditions there continues to be very complicated behavior forever. And indeed the behavior that is produced appears in many respects completely random. But dotted around the picture one sees many definite white triangles and other small structures that indicate at least a certain degree of organization.

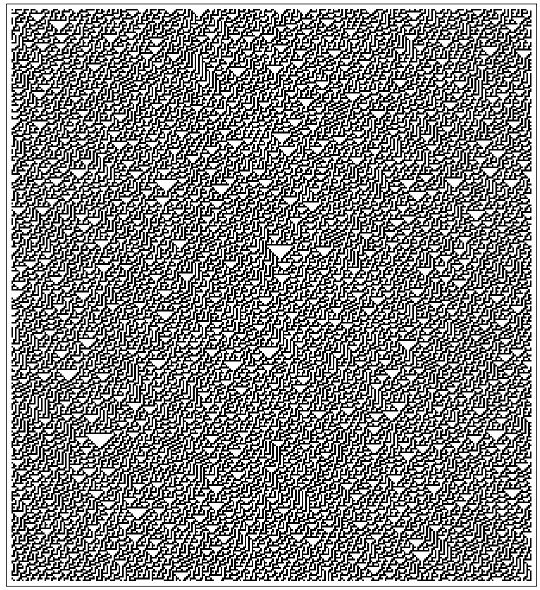




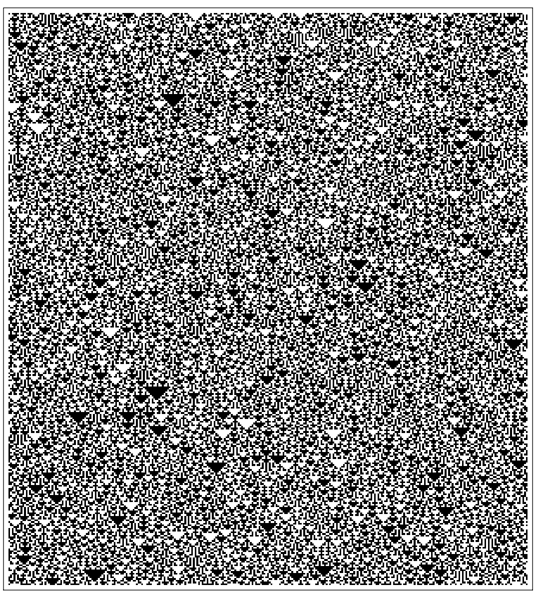
A cellular automaton that never settles down to a stable state, but instead continues to show behavior that seems in many respects random. The rule is number 126.



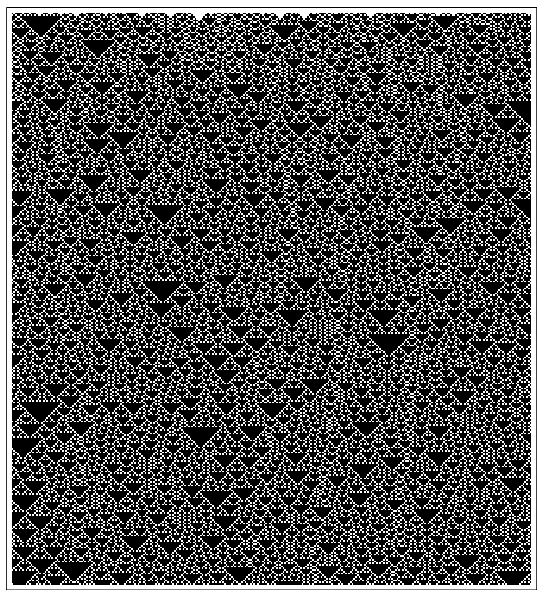
rule 22



rule 30

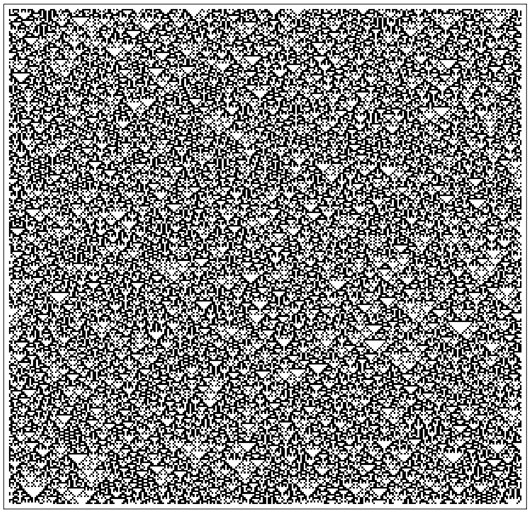


rule 150

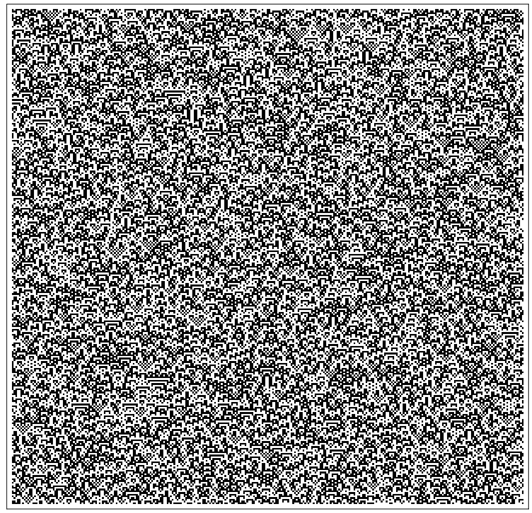


rule 182

Other examples of cellular automata that never settle down to stable states when started from random initial conditions. Each picture is a total of 300 cells across. Note the presence of triangles and other small structures dotted throughout all of the pictures.



rule 90



rule 105

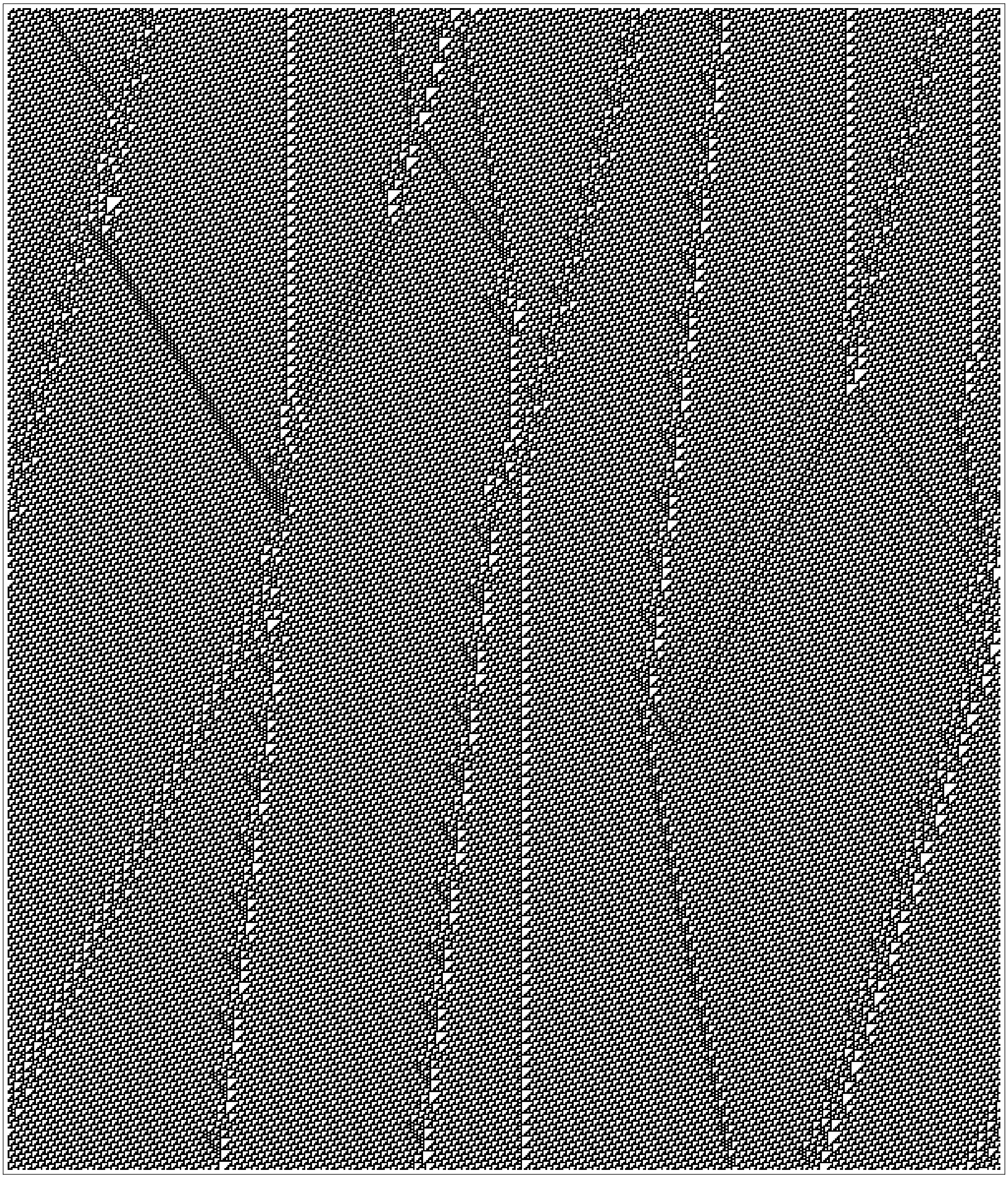
Two more cellular automata that generate various small structures but continue to show seemingly quite random behavior forever.

The pictures above and on the previous page show more examples of cellular automata with similar behavior. There is considerable randomness in the patterns produced in each case. But despite this randomness there are always triangles and other small structures that emerge in the evolution of the system.

So just how complex can the behavior of a cellular automaton that starts from random initial conditions be? We have seen some examples where the behavior quickly stabilizes, and others where it continues to be quite random forever. But in a sense the greatest complexity lies between these extremes—in systems that neither stabilize completely, nor exhibit close to uniform randomness forever.

The facing page and the one that follows show as an example the cellular automaton that we first discussed on page 32. The initial conditions used are again completely random. But the cellular automaton quickly organizes itself into a set of definite localized structures. Yet now these structures do not just remain fixed, but instead move around and interact with each other in complicated ways. And the result of this is an elaborate pattern that mixes order and randomness—and is as complex as anything we have seen in this book.

Complex behavior in the rule 110 cellular automaton starting from a random initial condition. The system quickly organizes itself to produce a set of definite localized structures, which then move around and interact with each other in complicated ways.



A continuation of the pattern from the previous page. Each page shows 700 steps in the evolution of the cellular automaton.

Four Classes of Behavior

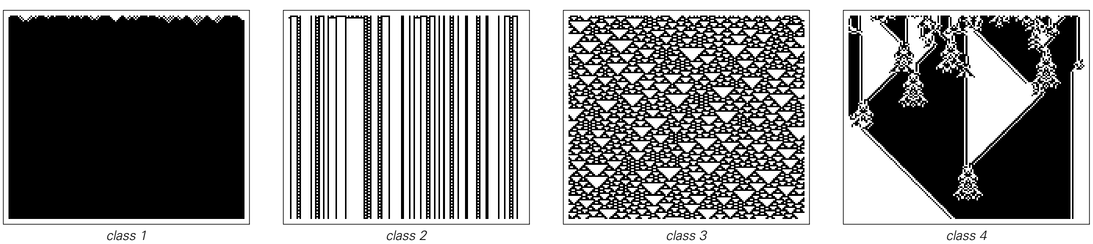
In the previous section we saw what a number of specific cellular automata do if one starts them from random initial conditions. But in this section I want to ask the more general question of what arbitrary cellular automata do when started from random initial conditions.

One might at first assume that such a general question could never have a useful answer. For every single cellular automaton after all ultimately has a different underlying rule, with different properties and potentially different consequences.

But the next few pages show various sequences of cellular automata, all starting from random initial conditions.

And while it is indeed true that for almost every rule the specific pattern produced is at least somewhat different, when one looks at all the rules together, one sees something quite remarkable: that even though each pattern is different in detail, the number of fundamentally different types of patterns is very limited.

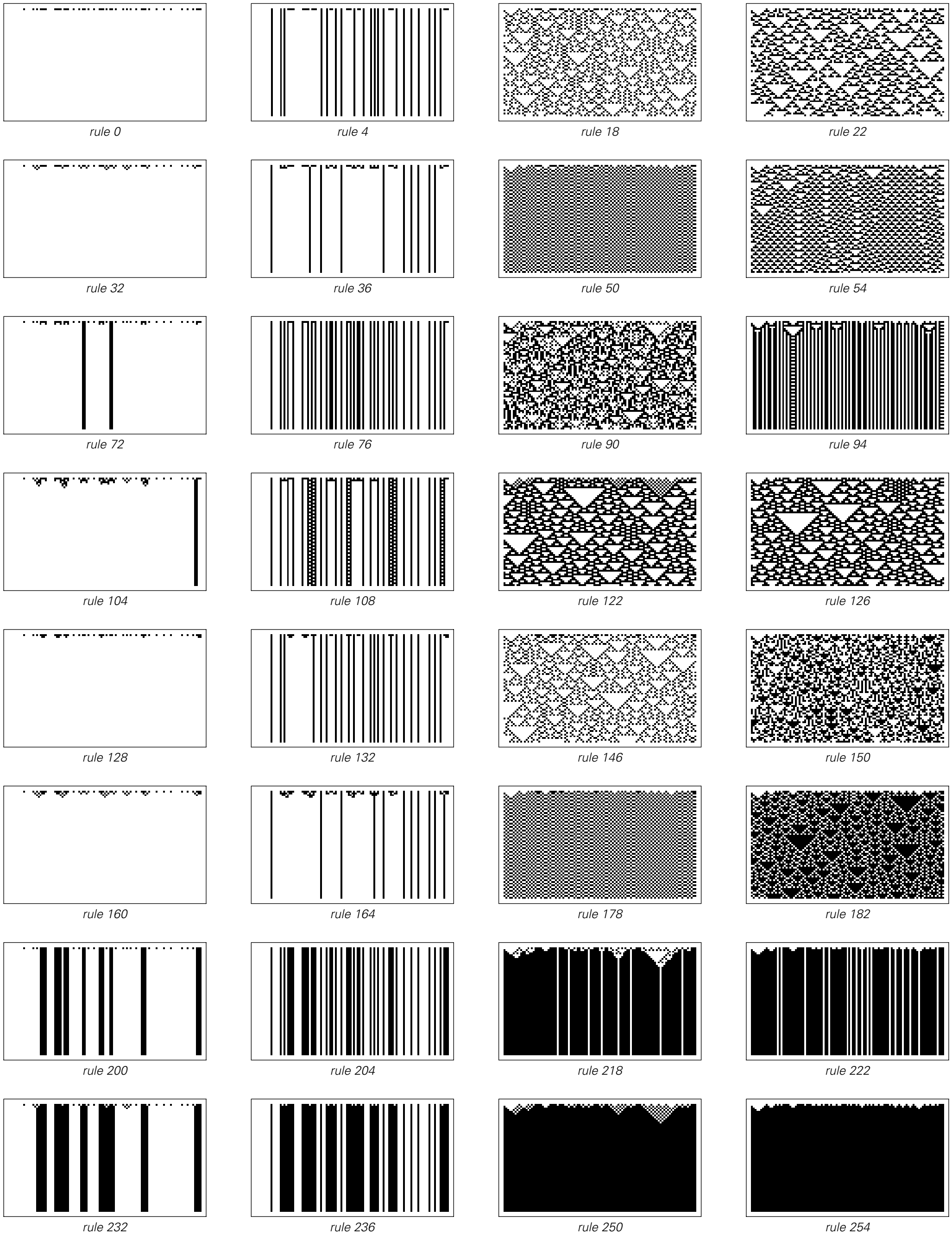
Indeed, among all kinds of cellular automata, it seems that the patterns which arise can almost always be assigned quite easily to one of just four basic classes illustrated below.



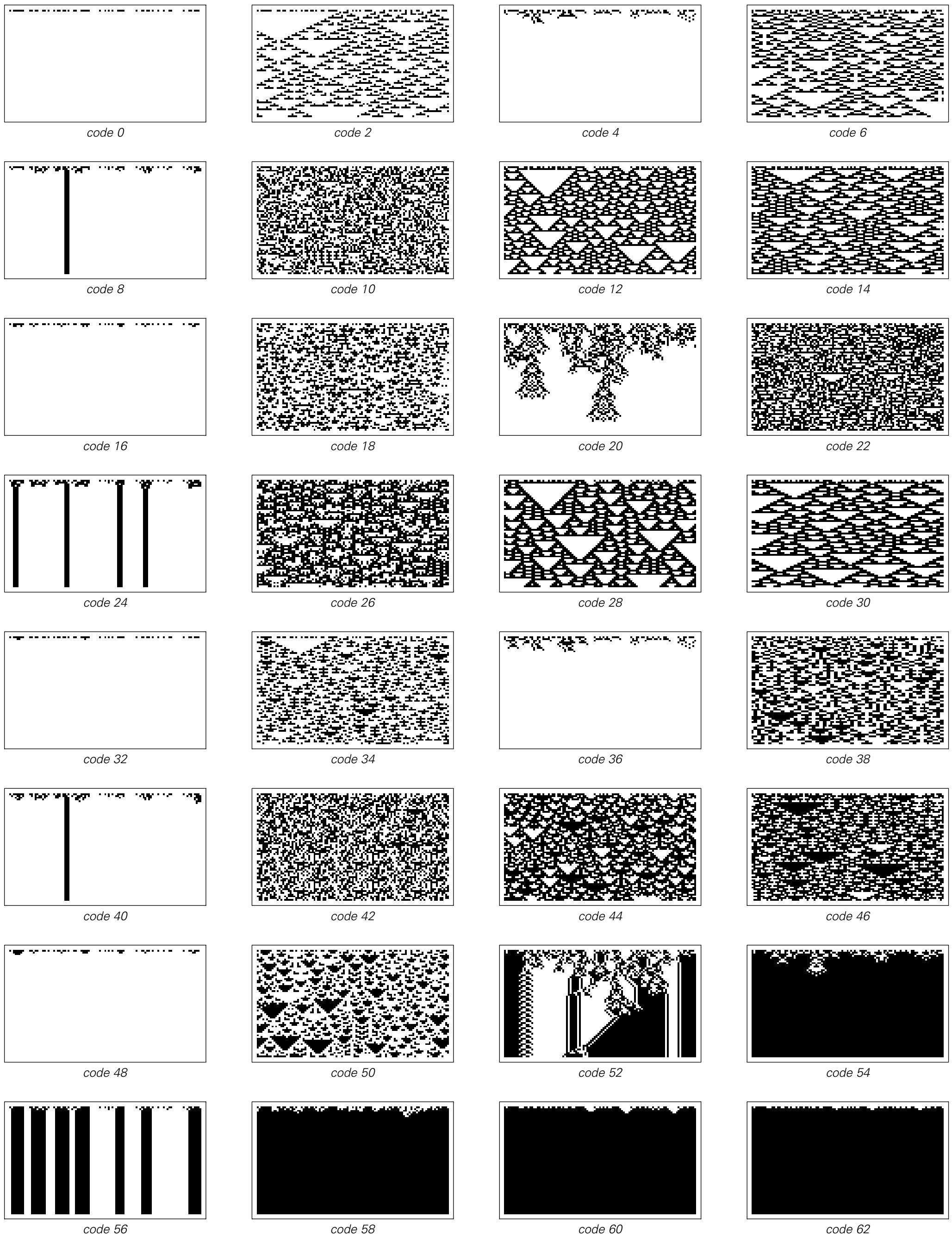
Examples of the four basic classes of behavior seen in the evolution of cellular automata from random initial conditions. I first developed this classification in 1983.

These classes are conveniently numbered in order of increasing complexity, and each one has certain immediate distinctive features.

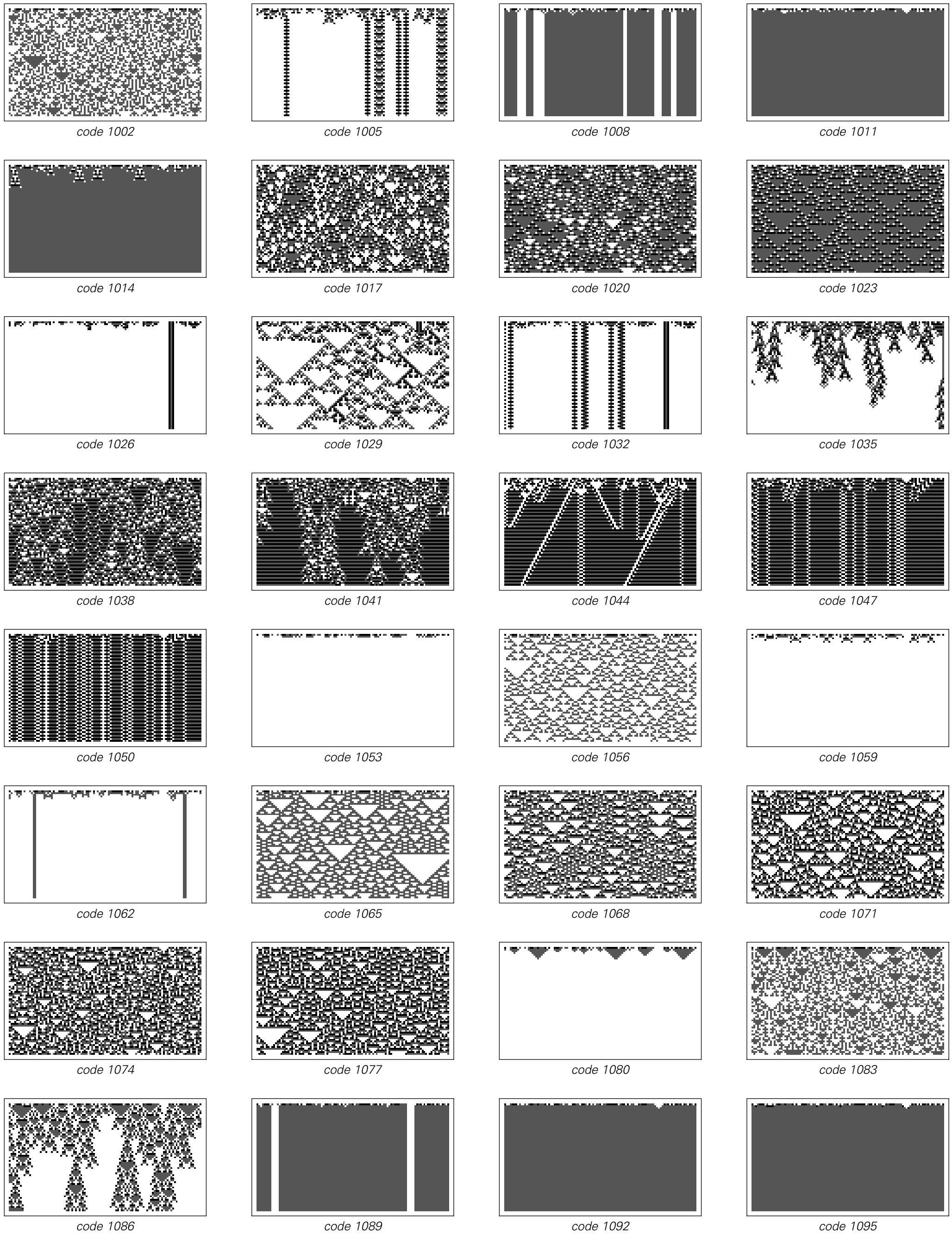
In class 1, the behavior is very simple, and almost all initial conditions lead to exactly the same uniform final state.



The behavior of all cellular automata that involve only nearest neighbors in a symmetrical way, have two possible colors for each cell, and leave states consisting only of white cells unchanged.



Totalistic cellular automata whose rules involve nearest and next-nearest neighbors, and where each cell has two possible colors.



A sequence of totalistic cellular automata with rules that involve only nearest neighbors, but where each cell can have three possible colors.

In class 2, there are many different possible final states, but all of them consist just of a certain set of simple structures that either remain the same forever or repeat every few steps.

In class 3, the behavior is more complicated, and seems in many respects random, although triangles and other small-scale structures are essentially always at some level seen.

And finally, as illustrated on the next few pages, class 4 involves a mixture of order and randomness: localized structures are produced which on their own are fairly simple, but these structures move around and interact with each other in very complicated ways.

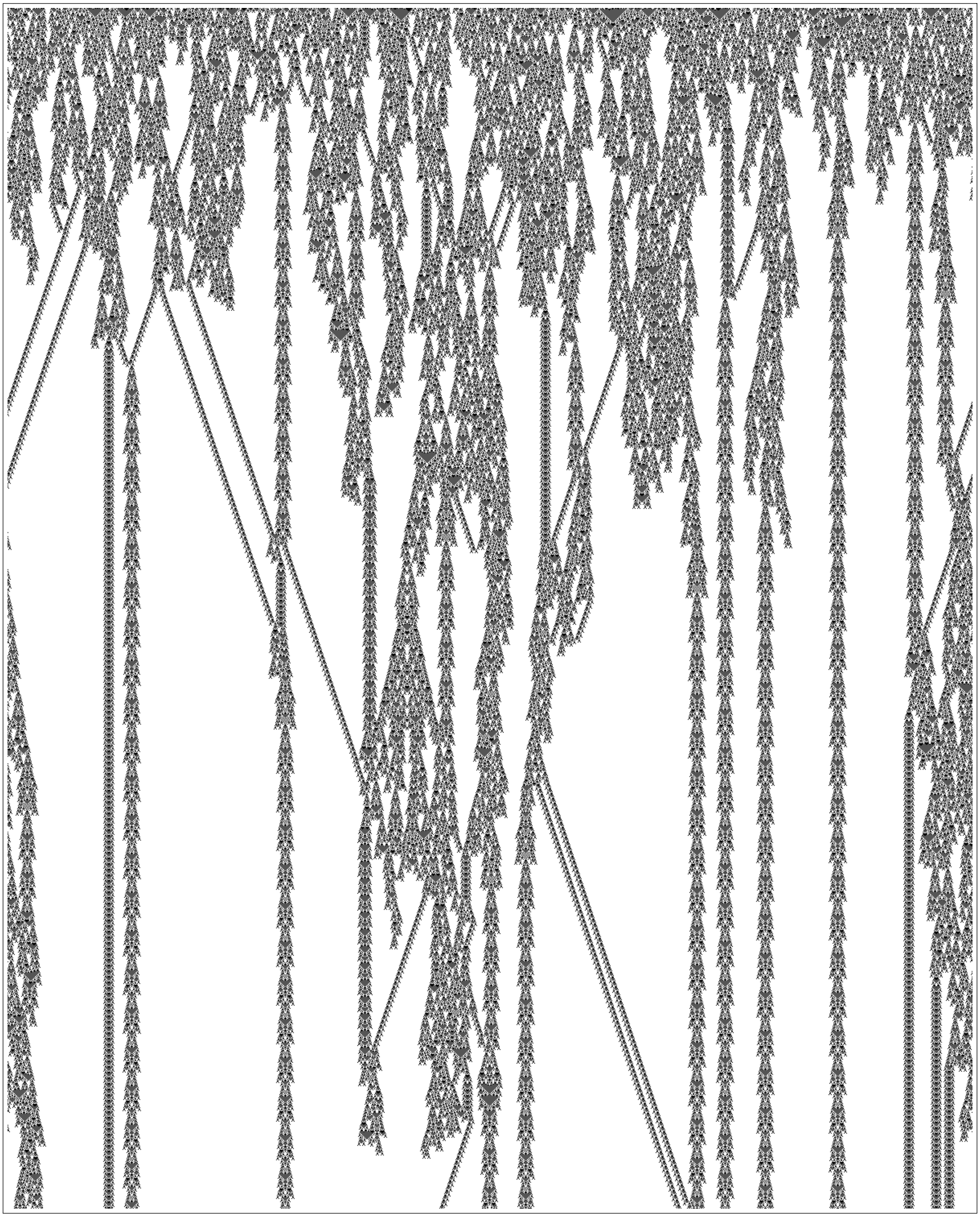
I originally discovered these four classes of behavior some seventeen years ago by looking at thousands of pictures similar to those on the last few pages. And at first, much as I have done here, I based my classification purely on the general visual appearance of the patterns I saw.

But when I studied more detailed properties of cellular automata, what I found was that most of these properties were closely correlated with the classes that I had already identified. Indeed, in trying to predict detailed properties of a particular cellular automaton, it was often enough just to know what class the cellular automaton was in.

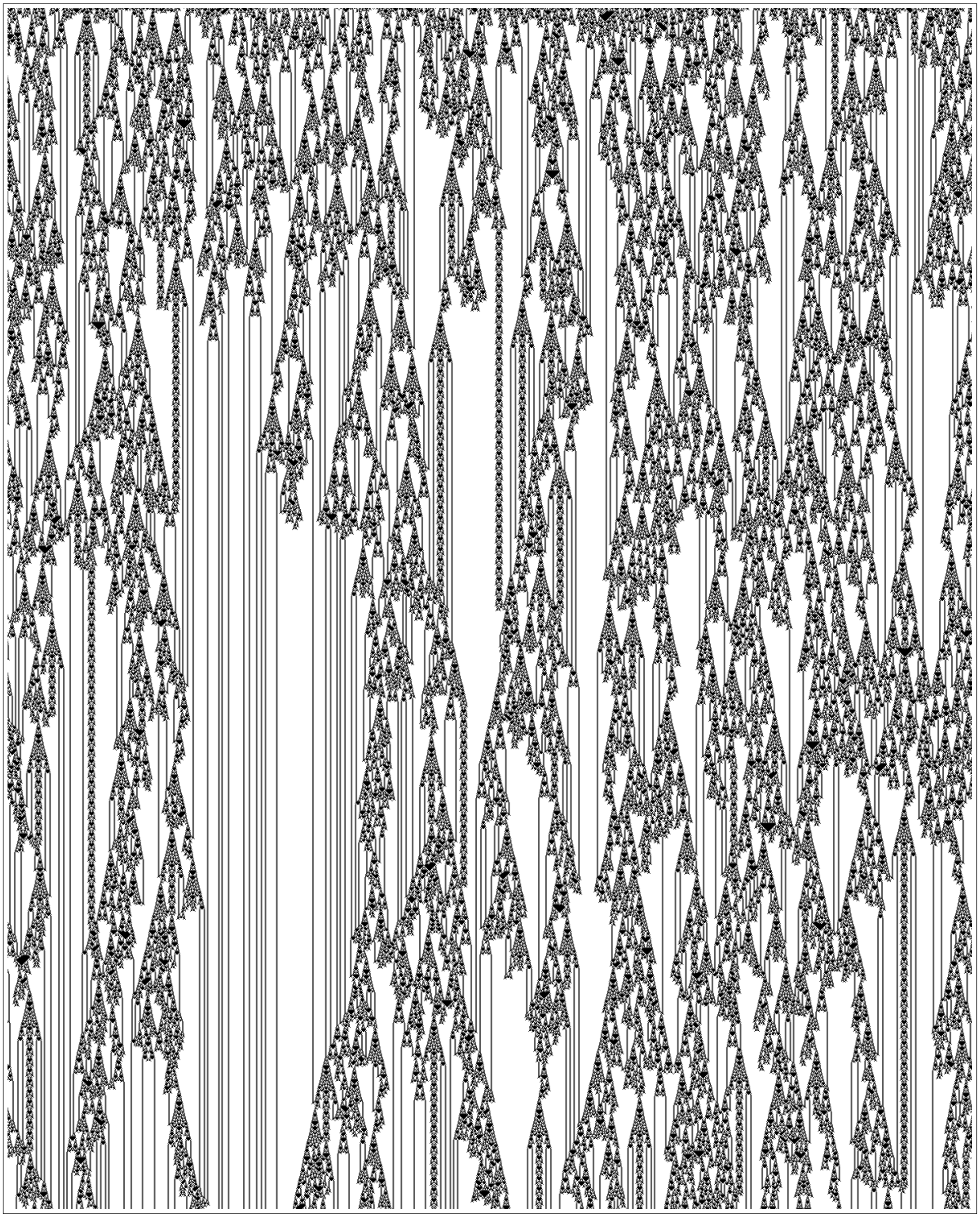
And in a sense the situation was similar to what is seen, say, with the classification of materials into solids, liquids and gases, or of living organisms into plants and animals. At first, a classification is made purely on the basis of general appearance. But later, when more detailed properties become known, these properties turn out to be correlated with the classes that have already been identified.

Often it is possible to use such detailed properties to make more precise definitions of the original classes. And typically all reasonable definitions will then assign any particular system to the same class.

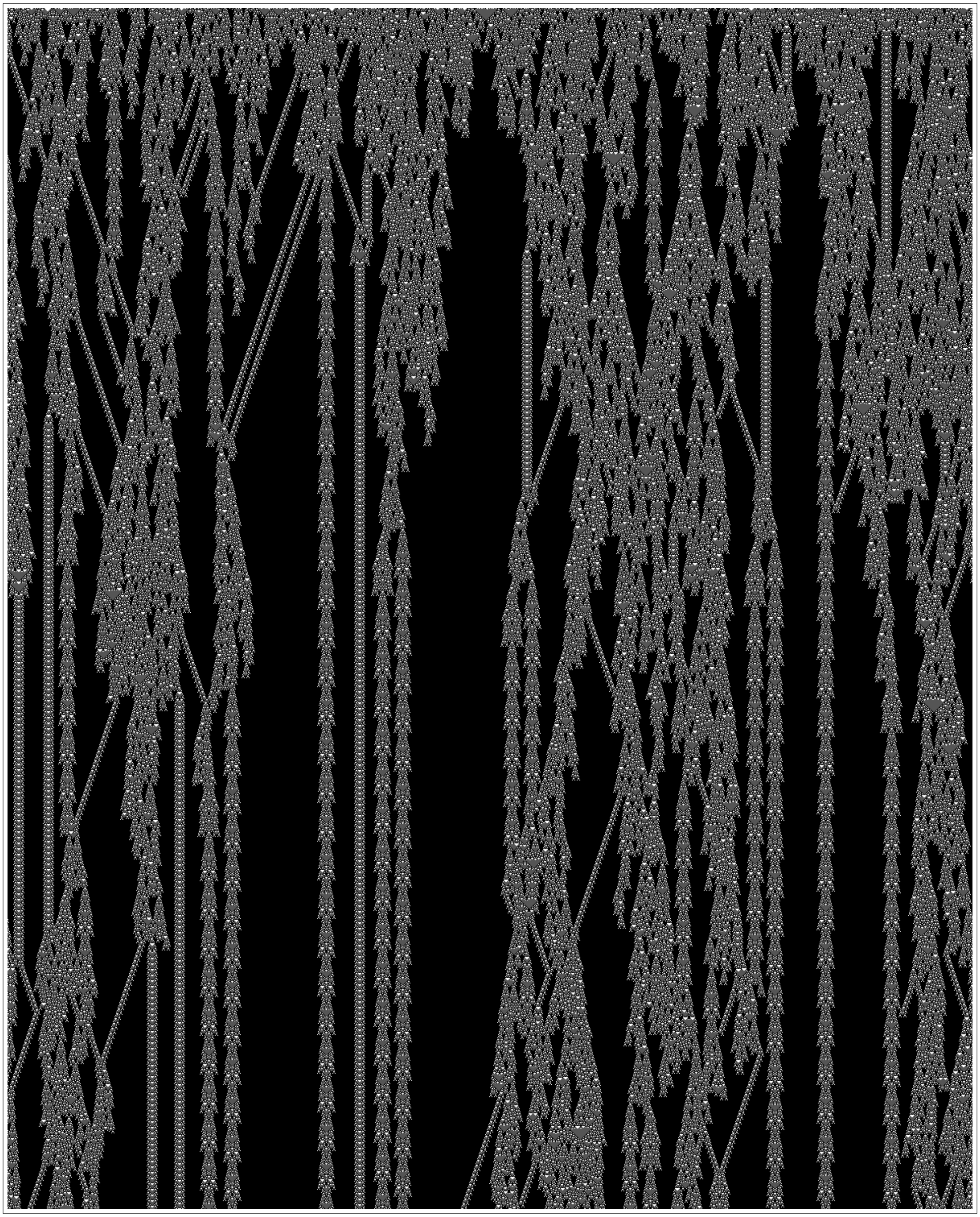
Examples of class 4 cellular automata with totalistic rules involving nearest neighbors and three possible colors for each cell. Each picture shows 1500 steps of evolution from random initial conditions. ▶



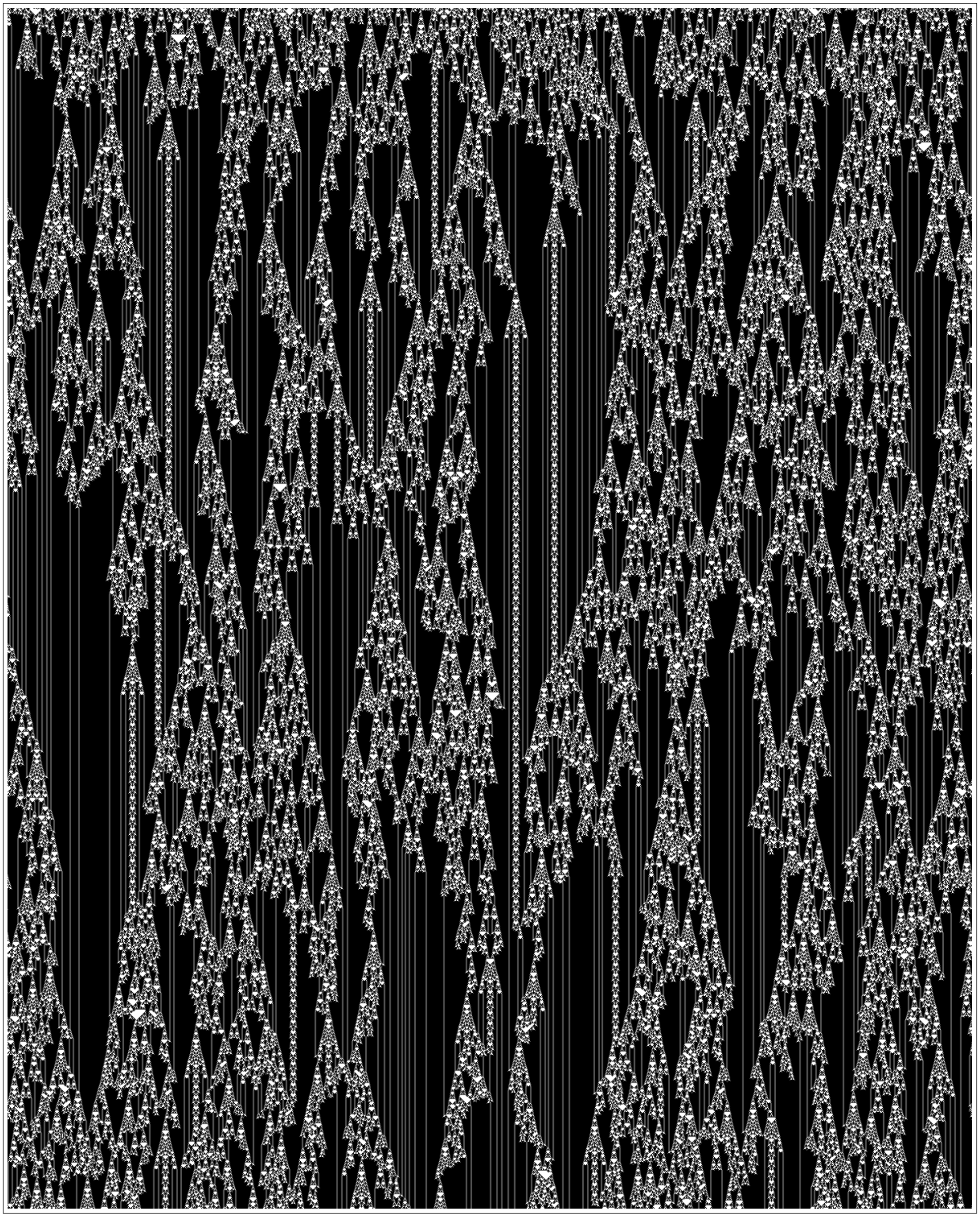
code 1815



code 2007

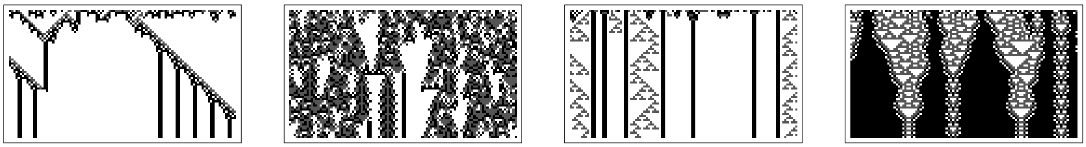


code 1659



code 2043

But with almost any general classification scheme there are inevitably borderline cases which get assigned to one class by one definition and another class by another definition. And so it is with cellular automata: there are occasionally rules like those in the pictures below that show some features of one class and some of another.



code 219 | code 438 | code 1380 | code 1632

Rare examples of borderline cellular automata that do not fit squarely into any one of the four basic classes described in the text. Different definitions based on different specific properties will place these cellular automata into different classes. The rules shown are totalistic ones involving nearest neighbors and three possible colors for each cell. The first rule can be either class 2 or class 4, the second class 3 or 4, the third class 2 or 3 and the fourth class 1, 2 or 3.

But such rules are quite unusual, and in most cases the behavior one sees instead falls squarely into one of the four classes described above.

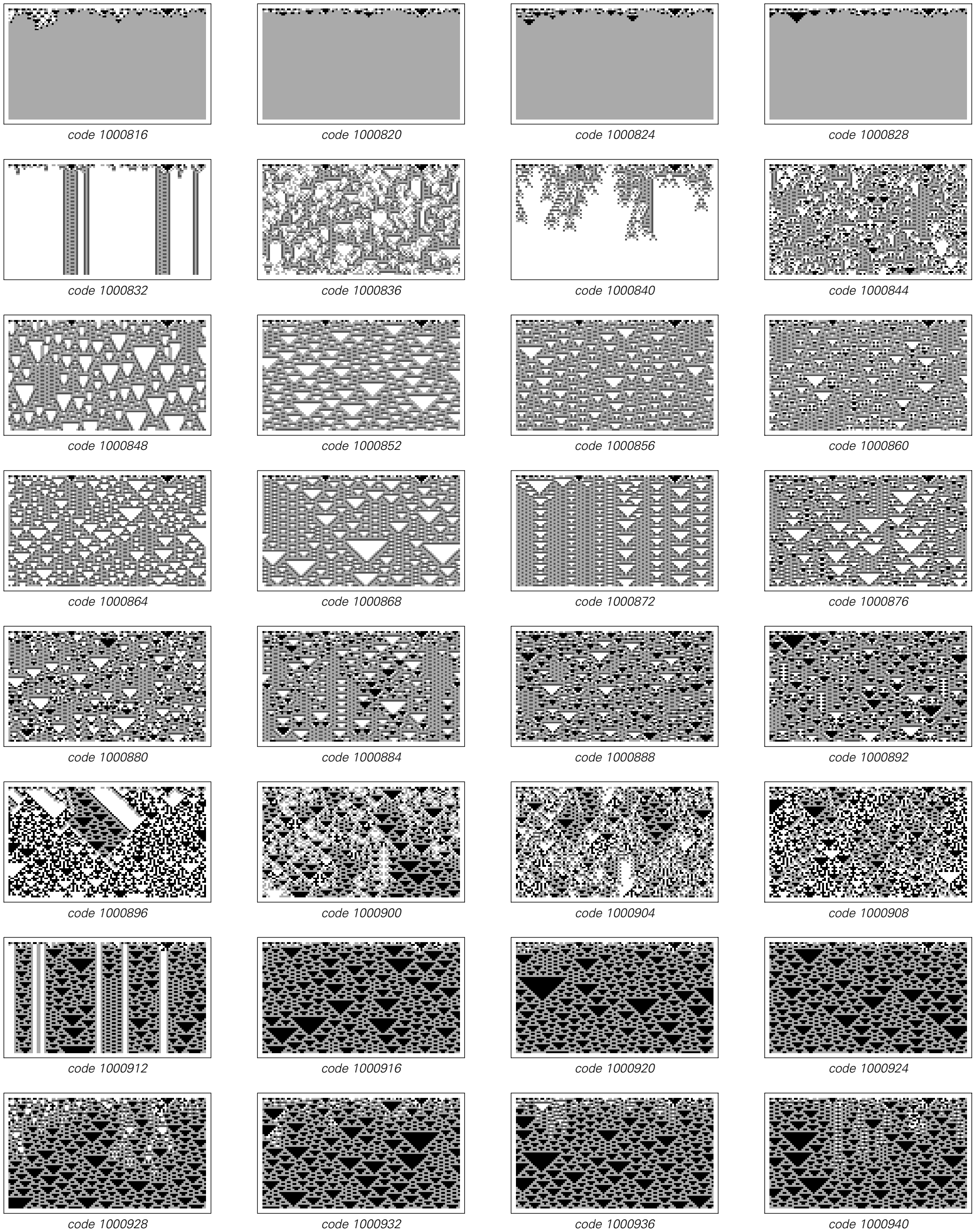
So given the underlying rule for a particular cellular automaton, can one tell what class of behavior the cellular automaton will produce?

In most cases there is no easy way to do this, and in fact there is little choice but just to run the cellular automaton and see what it does.

But sometimes one can tell at least a certain amount simply from the form of the underlying rule. And so for example all rules that lie in the first two columns on page 232 can be shown to be unable ever to produce anything besides class 1 or class 2 behavior.

In addition, even when one can tell rather little from a single rule, it is often the case that rules which occur next to each other in some sequence have similar behavior. This can be seen for example in the pictures on the facing page. The top row of rules all have class 1 behavior. But then class 2 behavior is seen, followed by class 4 and then class 3. And after that, the remainder of the rules are mostly class 3.

The fact that class 4 appears between class 2 and class 3 in the pictures on the facing page is not uncommon. For while class 4 is above class 3 in terms of apparent complexity, it is in a sense intermediate



A sequence of totalistic rules involving nearest neighbors and four possible colors for each cell chosen to show transitions between rules with different classes of behavior. Note that class 4 seems to occur between class 2 and class 3.

between class 2 and class 3 in terms of what one might think of as overall activity.

The point is that class 1 and 2 systems rapidly settle down to states in which there is essentially no further activity. But class 3 systems continue to have many cells that change at every step, so that they in a sense maintain a high level of activity forever. Class 4 systems are then in the middle: for the activity that they show neither dies out completely, as in class 2, nor remains at the high level seen in class 3.

And indeed when one looks at a particular class 4 system, it often seems to waver between class 2 and class 3 behavior, never firmly settling on either of them.

In some respects it is not surprising that among all possible cellular automata one can identify some that are effectively on the boundary between class 2 and class 3. But what is remarkable about actual class 4 systems that one finds in practice is that they have definite characteristics of their own—most notably the presence of localized structures—that seem to have no direct relation to being somehow on the boundary between class 2 and class 3.

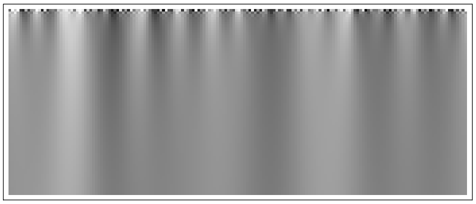
And it turns out that class 4 systems with the same general characteristics are seen for example not only in ordinary cellular automata but also in such systems as continuous cellular automata.

The facing page shows a sequence of continuous cellular automata of the kind we discussed on page 155. The underlying rules in such systems involve a parameter that can vary smoothly from 0 to 1.

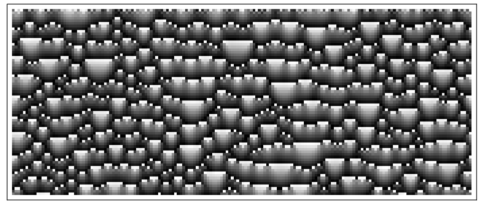
For different values of this parameter, the behavior one sees is different. But it seems that this behavior falls into essentially the same four classes that we have already seen in ordinary cellular automata. And indeed there are even quite direct analogs of for example the triangle structures that we saw in ordinary class 3 cellular automata.

But since continuous cellular automata have underlying rules based on a continuous parameter, one can ask what happens if one smoothly varies this parameter—and in particular one can ask what sequence of classes of behavior one ends up seeing.

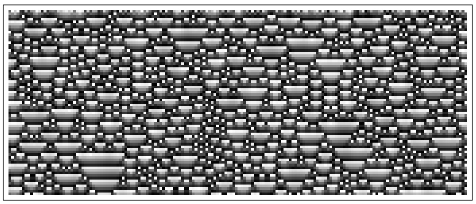
The answer is that there are normally some stretches of class 1 or 2 behavior, and some stretches of class 3 behavior. But at the transitions



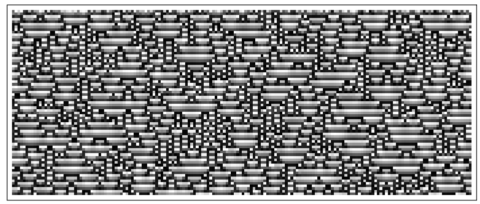
0



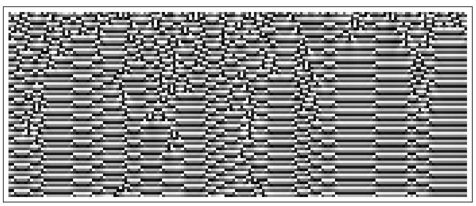
0.1



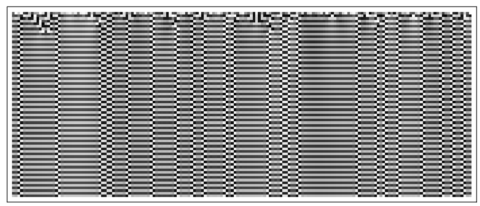
0.2



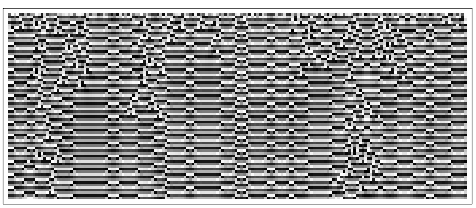
0.3



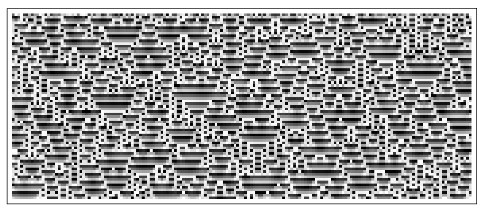
0.4



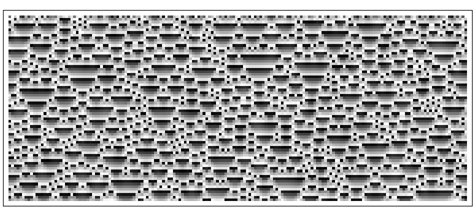
0.5



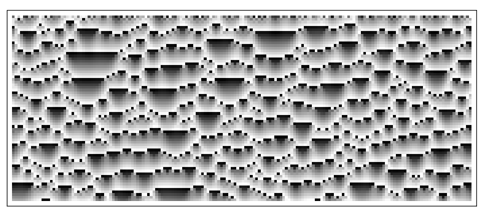
0.6



0.7

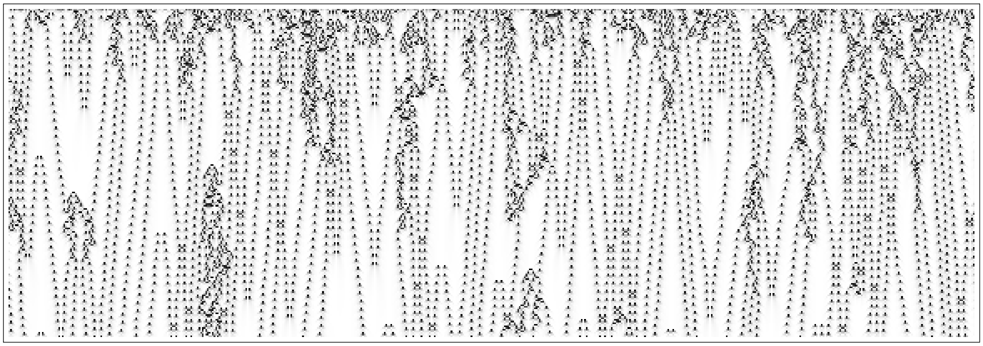


0.8

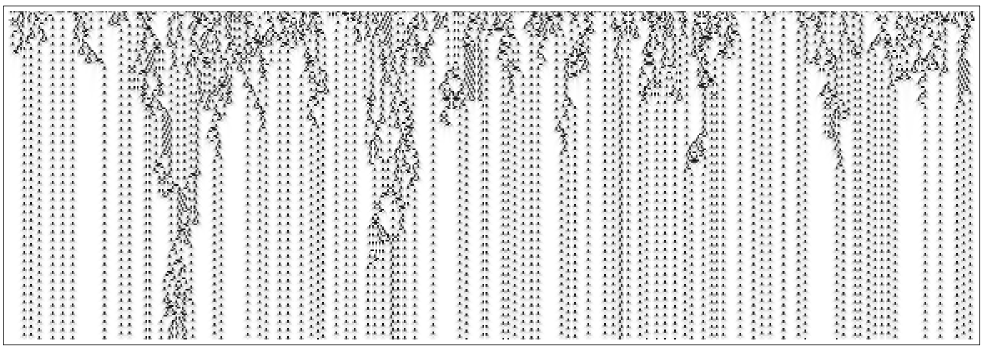


0.9

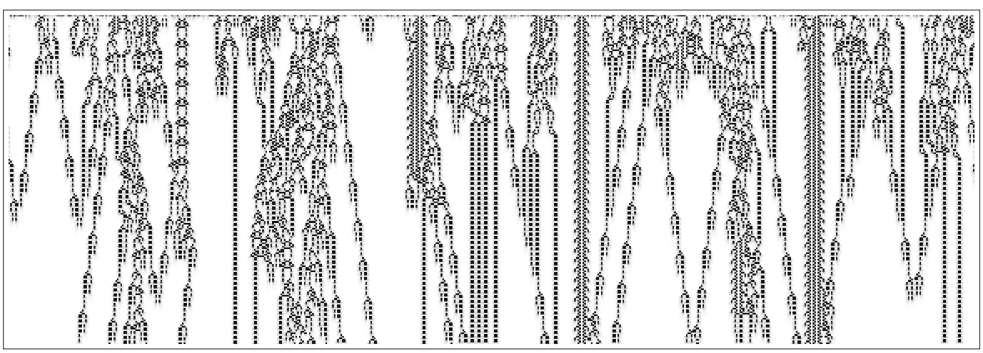
Examples of the evolution of continuous cellular automata from random initial conditions. As discussed on page 155, each cell here can have any gray level between 0 and 1, and at each step the gray level of a given cell is determined by averaging the gray levels of the cell and its two neighbors, adding the specified constant, and then keeping only the fractional part of the result. The behavior produced once again falls into distinct classes that correspond well to the four classes seen on previous pages in ordinary cellular automata.



0.398



0.4



{0.5, 1.13}

Examples of continuous cellular automata that exhibit class 4 behavior. The rules are of the same kind as in the previous picture, except that in the third case shown here, the gray level of each neighboring cell is multiplied by 1.13 before the average is done. In addition, the actual gray levels in these pictures are obtained by taking the difference between the gray level of each cell and its neighbor, thus removing the uniform stripes visible in the previous picture. It is remarkable that class 4 behavior with discrete localized structures can still occur in the continuous systems shown here.

it turns out that class 4 behavior is typically seen—as illustrated on the facing page. And what is particularly remarkable is that this behavior involves the same kinds of localized structures and other features that we saw in ordinary discrete class 4 cellular automata.

So what about two-dimensional cellular automata? Do these also exhibit the same four classes of behavior that we have seen in one dimension? The pictures on the next two pages show various steps in the evolution of some simple two-dimensional cellular automata starting from random initial conditions. And just as in one dimension a few distinct classes of behavior can immediately be seen.

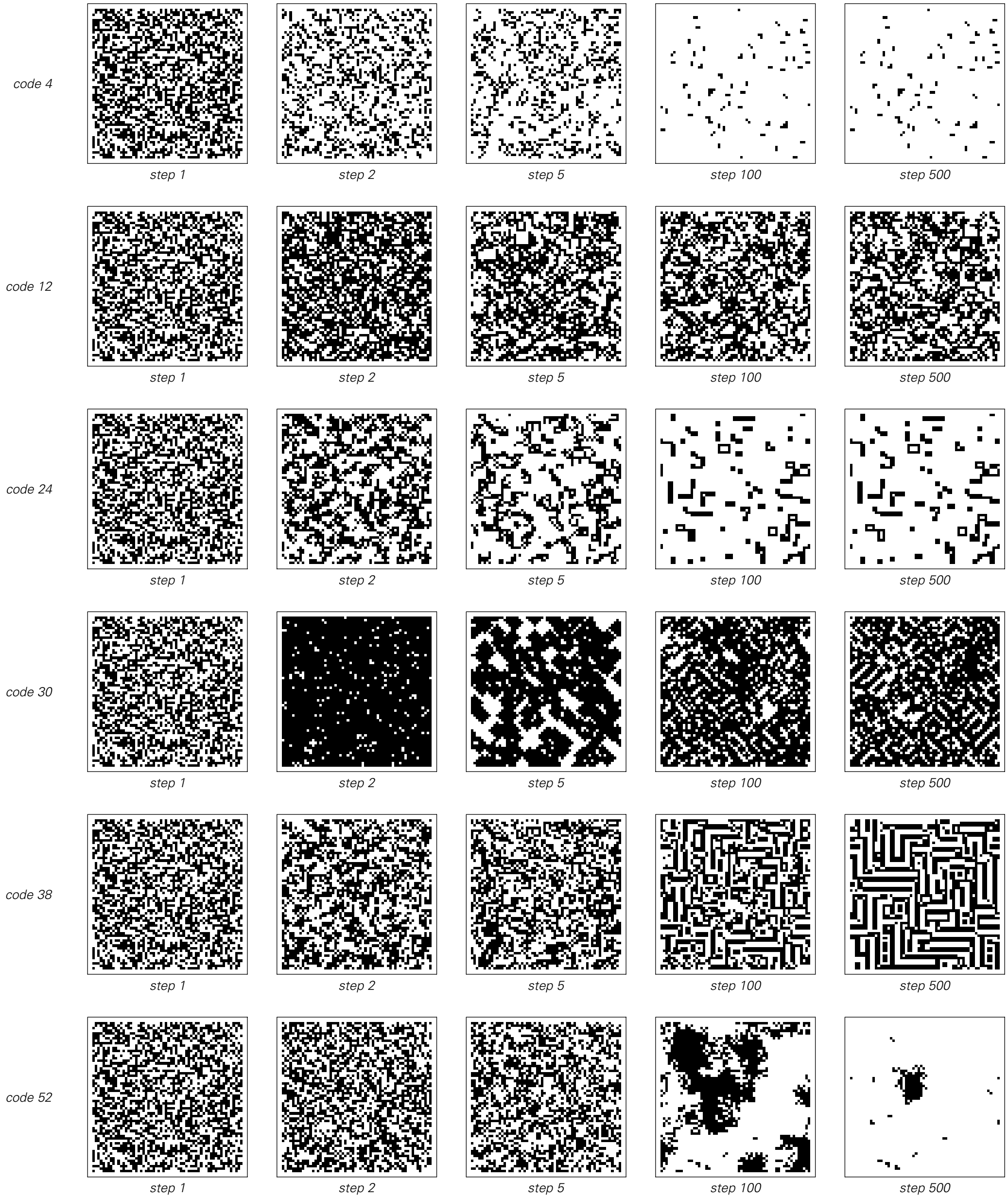
But the correspondence with one dimension becomes much more obvious if one looks not at the complete state of a two-dimensional cellular automaton at a few specific steps, but rather at a one-dimensional slice through the system for a whole sequence of steps.

The pictures on page 248 show examples of such slices. And what we see is that the patterns in these slices look remarkably similar to the patterns we already saw in ordinary one-dimensional cellular automata. Indeed, by looking at such slices one can readily identify the very same four classes of behavior as in one-dimensional cellular automata.

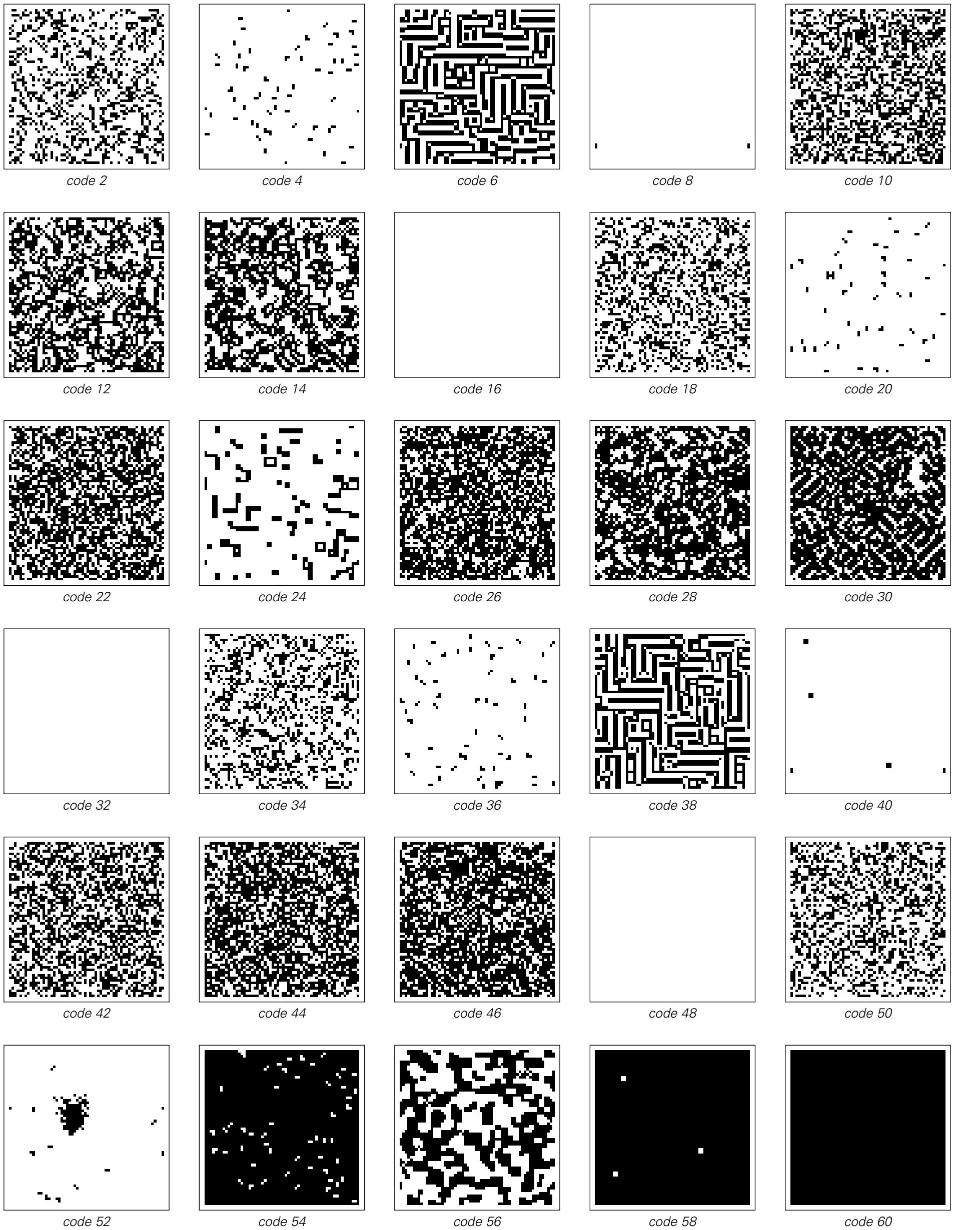
So in particular one sees class 4 behavior. In the examples on page 248, however, such behavior always seems to occur superimposed on some kind of repetitive background—much as in the case of the rule 110 one-dimensional cellular automaton on page 229.

So can one get class 4 behavior with a simple white background? Much as in one dimension this does not seem to happen with the very simplest possible kinds of rules. But as soon as one goes to slightly more complicated rules—though still very simple—one can find examples.

And so as one example page 249 shows a two-dimensional cellular automaton often called the Game of Life in which all sorts of localized structures occur even on a white background. If one watches a movie of the behavior of this cellular automaton its correspondence to a one-dimensional class 4 system is not particularly obvious. But as soon as one looks at a one-dimensional slice—as on page 249—what one sees is immediately strikingly similar to what we have seen in many one-dimensional class 4 cellular automata.

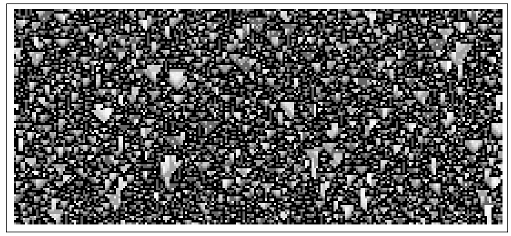


Examples of the evolution of two-dimensional cellular automata with various totalistic rules starting from random initial conditions. The rules involve a cell and its four immediate neighbors. Each successive base 2 digit in the code number for the rule gives the outcome when the total of the cell and its four neighbors runs from 5 down to 0.



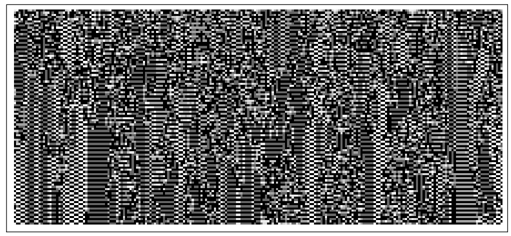
Patterns produced after 500 steps in the evolution of a sequence of two-dimensional cellular automata starting from random initial conditions. The rules shown are of the same kind as on the facing page, and include most of the 64 possibilities that leave a state that contains only white cells unchanged.

code 4

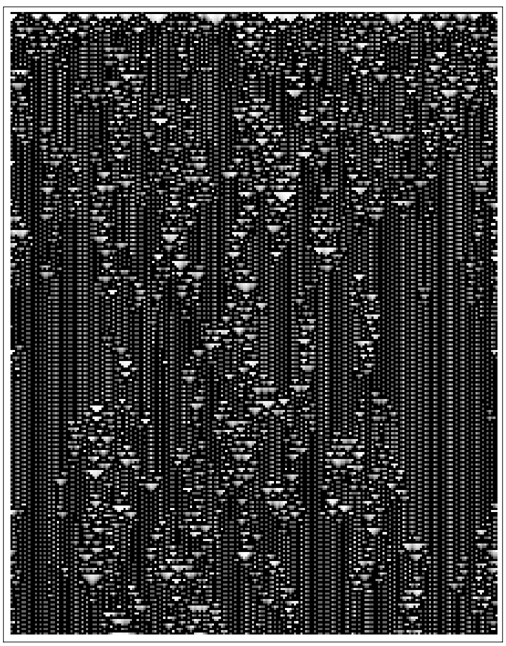


code 12

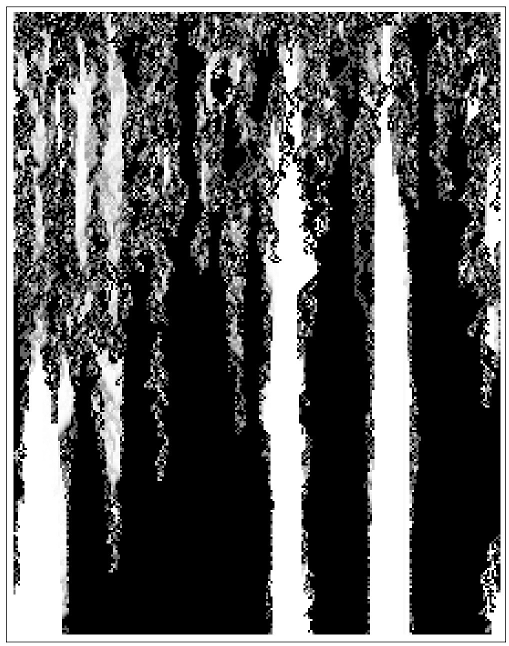
code 24



code 38

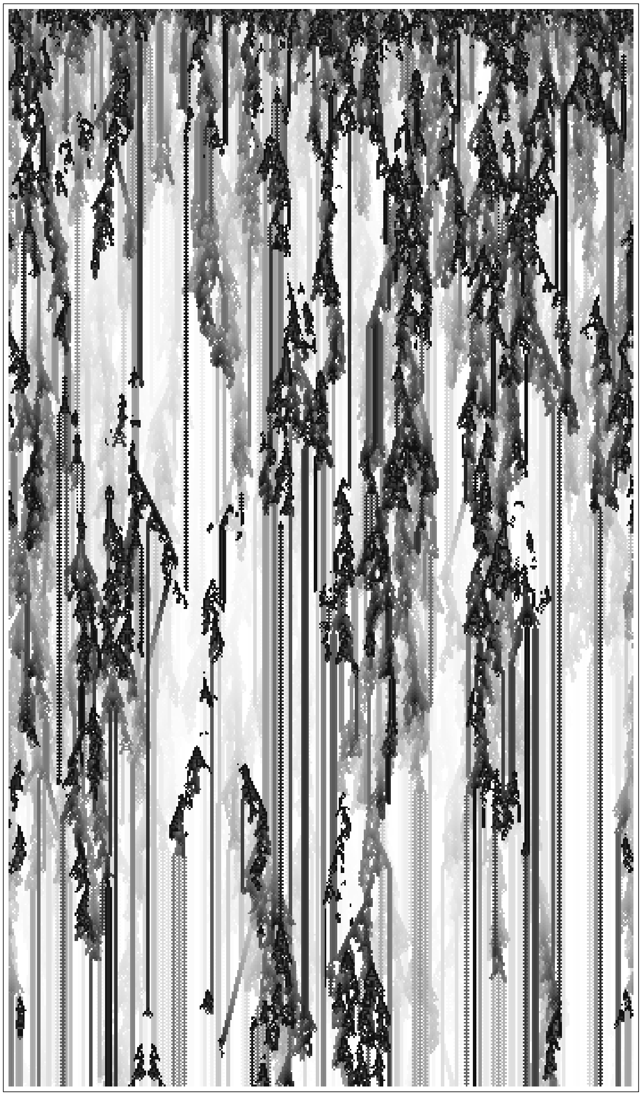


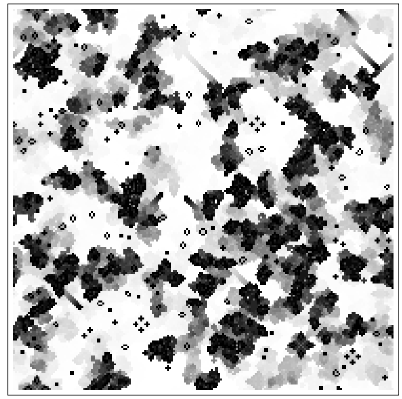
code 30



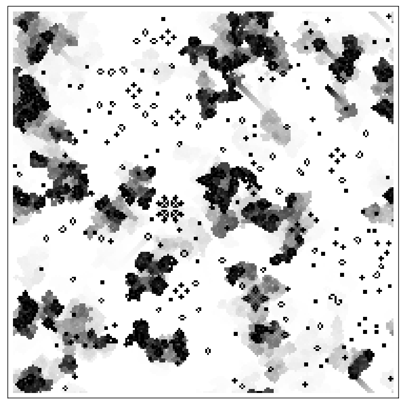
code 52

One-dimensional slices through the evolution of various two-dimensional cellular automata. In each picture black cells further back from the position of the slice are shown in progressively lighter shades of gray, as if they were receding into a kind of fog. Note the presence of examples of both class 3 and class 4 behavior that look strikingly similar to examples in one dimension.

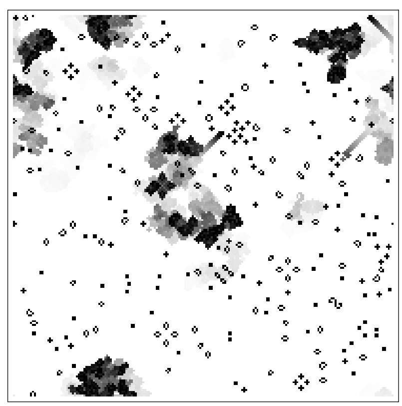




step 200



step 500



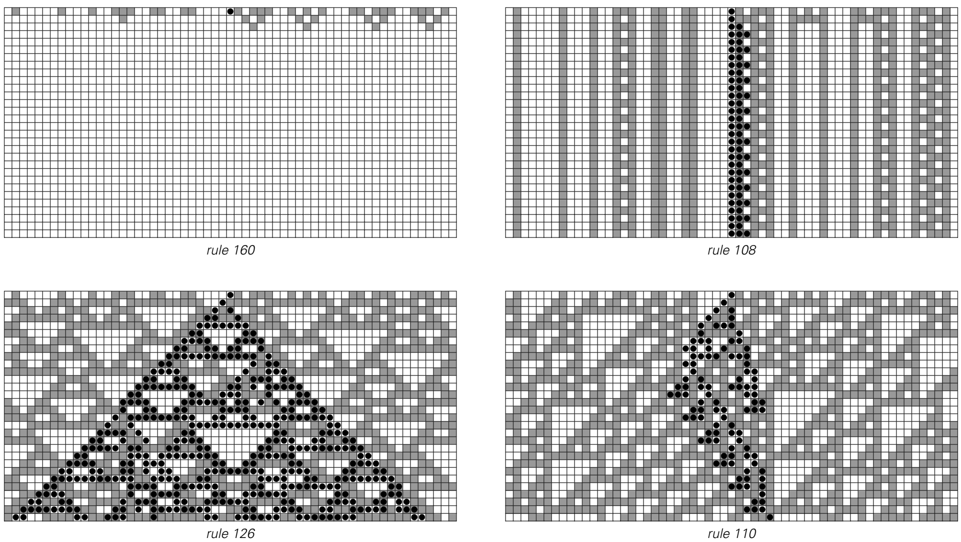
step 1000

The behavior of a class 4 two-dimensional cellular automaton often known in recreational computing as the Game of Life. Localized structures that move (so-called gliders) show up as streaks in the pictures given here. The rule for this cellular automaton considers the 8 neighbors of a cell (including diagonals): if two of these neighbors are black, then the cell stays the same color as before; if three are black, then the cell becomes black; and if any other number of neighbors are black, then the cell becomes white. This rule is outer totalistic 9-neighbor code 224. The pictures on the right show cells that were black on preceding steps in progressively lighter shades of gray.

Sensitivity to Initial Conditions

In the previous section we identified four basic classes of cellular automata by looking at the overall appearance of patterns they produce. But these four classes also have other significant distinguishing features—and one important example of these is their sensitivity to small changes in initial conditions.

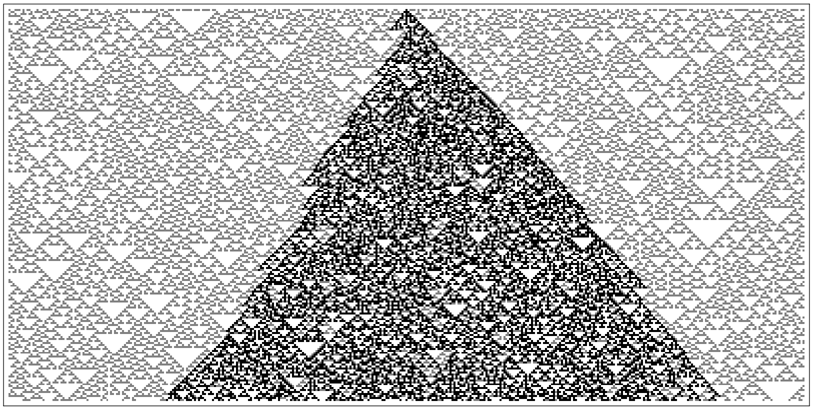
The pictures below show the effect of changing the initial color of a single cell in a typical cellular automaton from each of the four classes of cellular automata identified in the previous section.



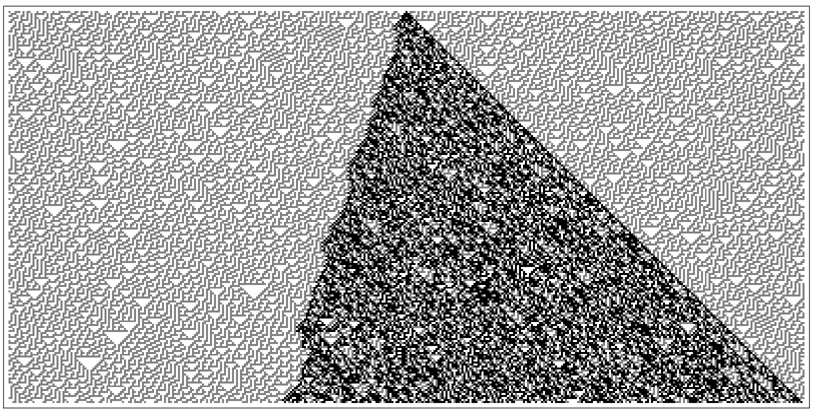
The effect of changing the color of a single cell in the initial conditions for typical cellular automata from each of the four classes identified in the previous section. The black dots indicate all the cells that change. The way that such changes behave is characteristically different for each of the four classes of systems.

The results are rather different for each class.

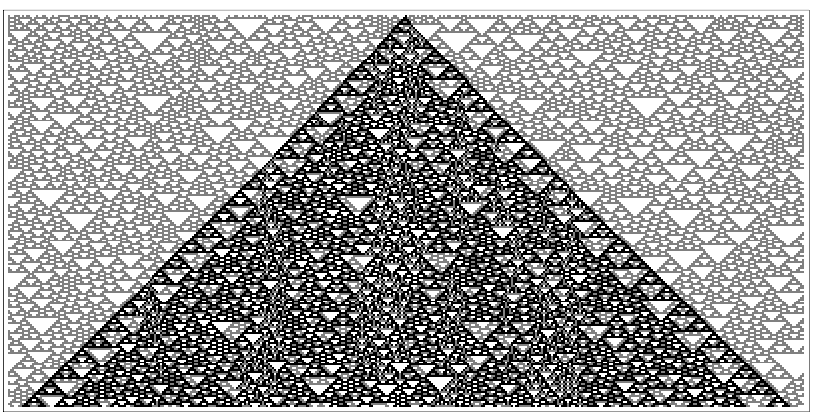
In class 1, changes always die out, and in fact exactly the same final state is reached regardless of what initial conditions were used. In class 2, changes may persist, but they always remain localized in a small region of the system. In class 3, however, the behavior is quite different. For as the facing page shows, any change that is made



rule 22



rule 30



rule 126

The effect of changing the color of a single initial cell in three typical class 3 cellular automata.

typically spreads at a uniform rate, eventually affecting every part of the system. In class 4, changes can also spread, but only in a sporadic way—as illustrated on the facing page and the one that follows.

So what is the real significance of these different responses to changes in initial conditions? In a sense what they reveal are basic differences in the way that each class of systems handles information.

In class 1, information about initial conditions is always rapidly forgotten—for whatever the initial conditions were, the system quickly evolves to a single final state that shows no trace of them.

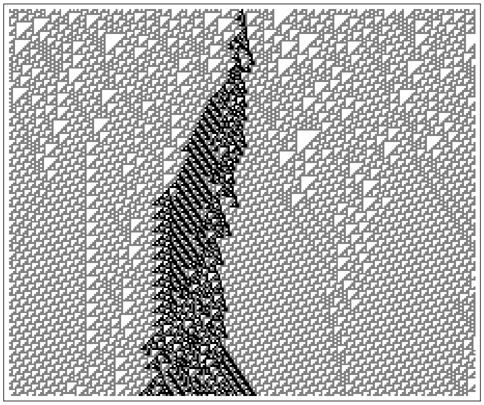
In class 2, some information about initial conditions is retained in the final configuration of structures, but this information always remains completely localized, and is never in any way communicated from one part of the system to another.

A characteristic feature of class 3 systems, on the other hand, is that they show long-range communication of information—so that any change made anywhere in the system will almost always eventually be communicated even to the most distant parts of the system.

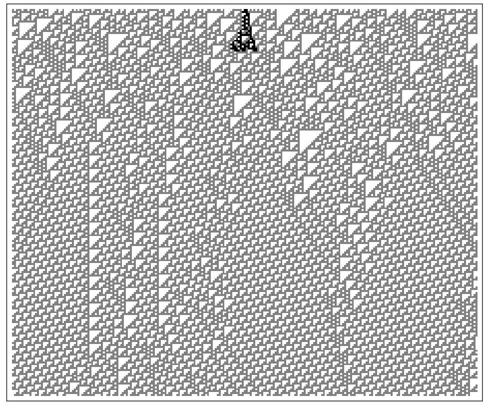
Class 4 systems are once again somewhat intermediate between class 2 and class 3. Long-range communication of information is in principle possible, but it does not always occur—for any particular change is only communicated to other parts of the system if it happens to affect one of the localized structures that moves across the system.

There are many characteristic differences between the four classes of systems that we identified in the previous section. But their differences in the handling of information are in some respects particularly fundamental. And indeed, as we will see later in this book, it is often possible to understand some of the most important features of systems that occur in nature just by looking at how their handling of information corresponds to what we have seen in the basic classes of systems that we have identified here.

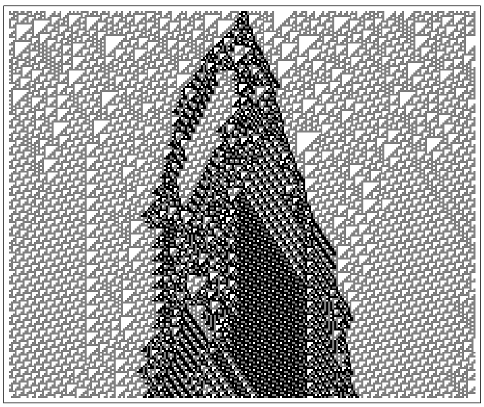
The effect of small changes in initial conditions in the rule 110 class 4 cellular automaton. The changes spread only when they are in effect carried by localized structures that propagates across the system. ▶



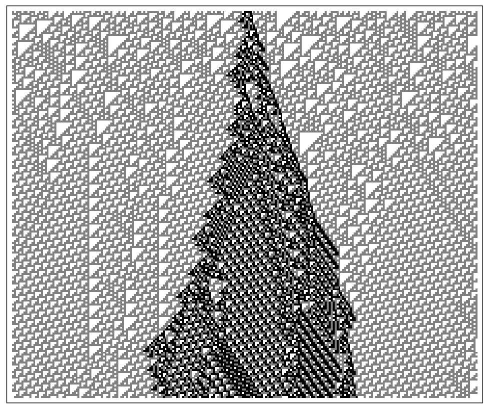
1 cell changed



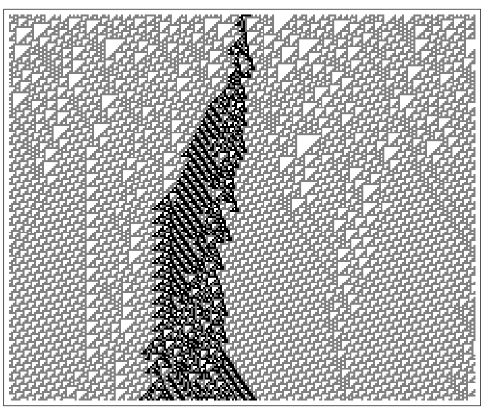
2 cells changed



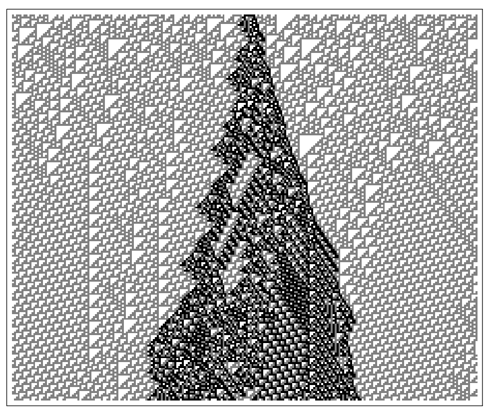
3 cells changed



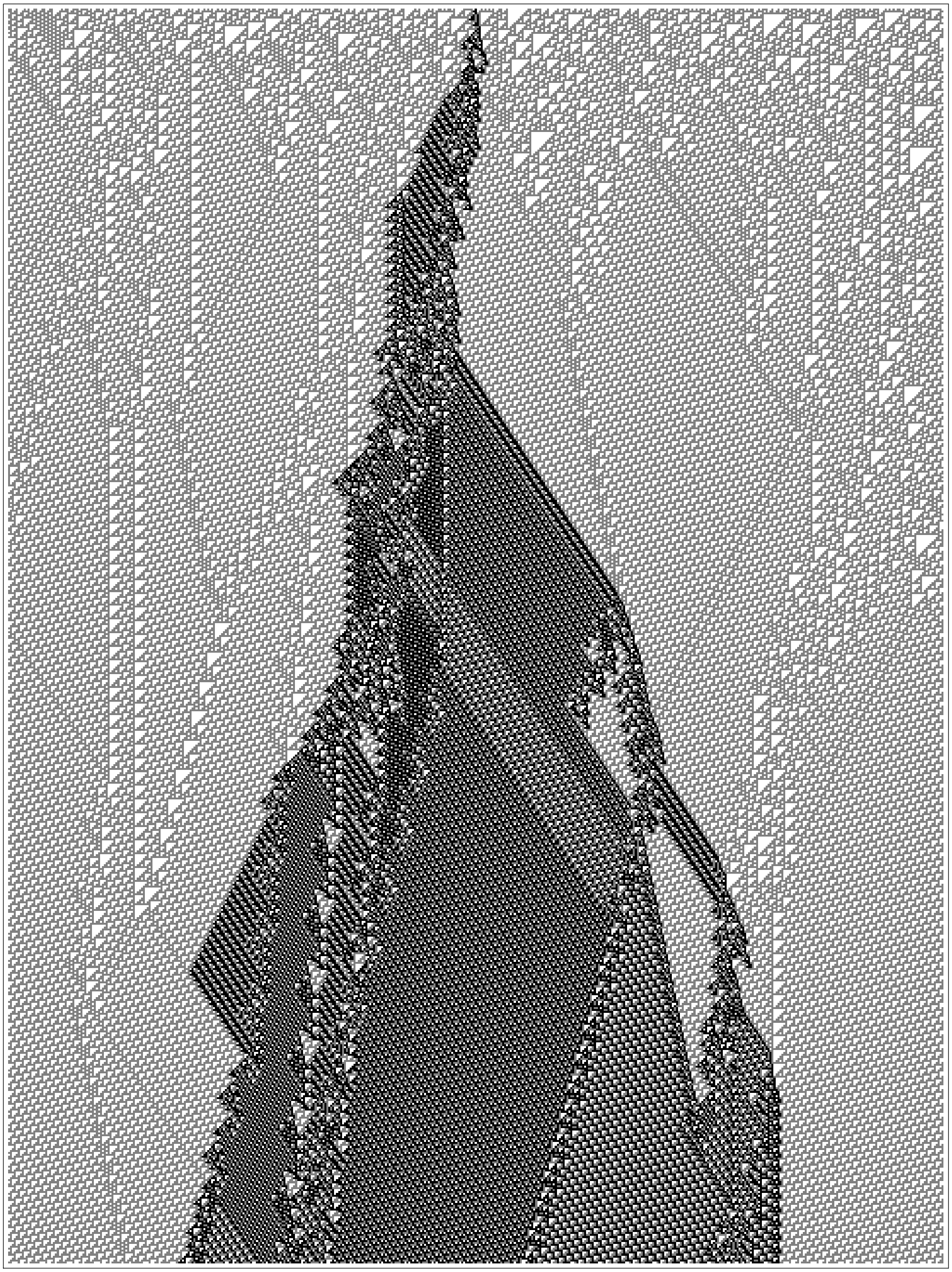
4 cells changed



5 cells changed



6 cells changed



1 cell changed

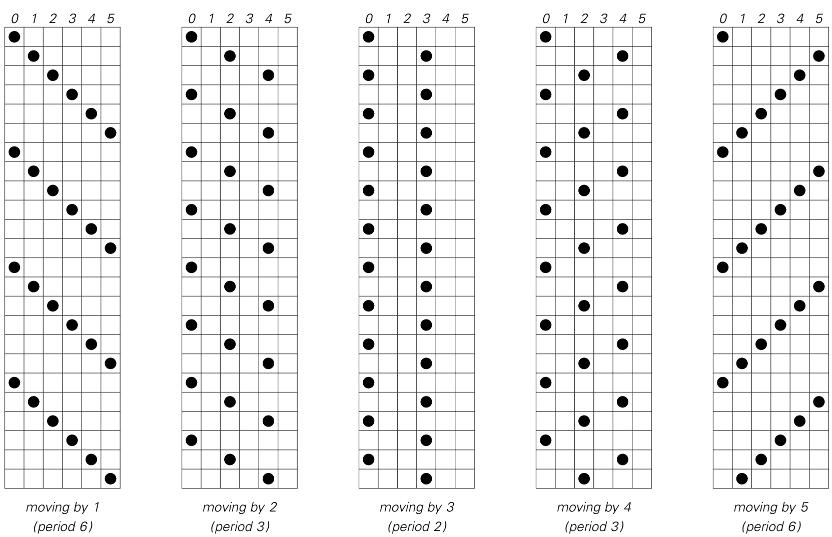
Systems of Limited Size and Class 2 Behavior

In the past two sections we have seen two important features of class 2 systems: first, that their behavior is always eventually repetitive, and second, that they do not support any kind of long-range communication.

So what is the connection between these two features?

The answer is that the absence of long-range communication effectively forces each part of a class 2 system to behave as if it were a system of limited size. And it is then a general result that any system of limited size that involves discrete elements and follows definite rules must always eventually exhibit repetitive behavior. Indeed, as we will discuss in the next chapter, it is this phenomenon that is ultimately responsible for much of the repetitive behavior that we see in nature.

The pictures below show a very simple example of the basic phenomenon. In each case there is a dot that can be in one of six possible positions. And at every step the dot moves a fixed number of positions to the right, wrapping around as soon as it reaches the right-hand end.

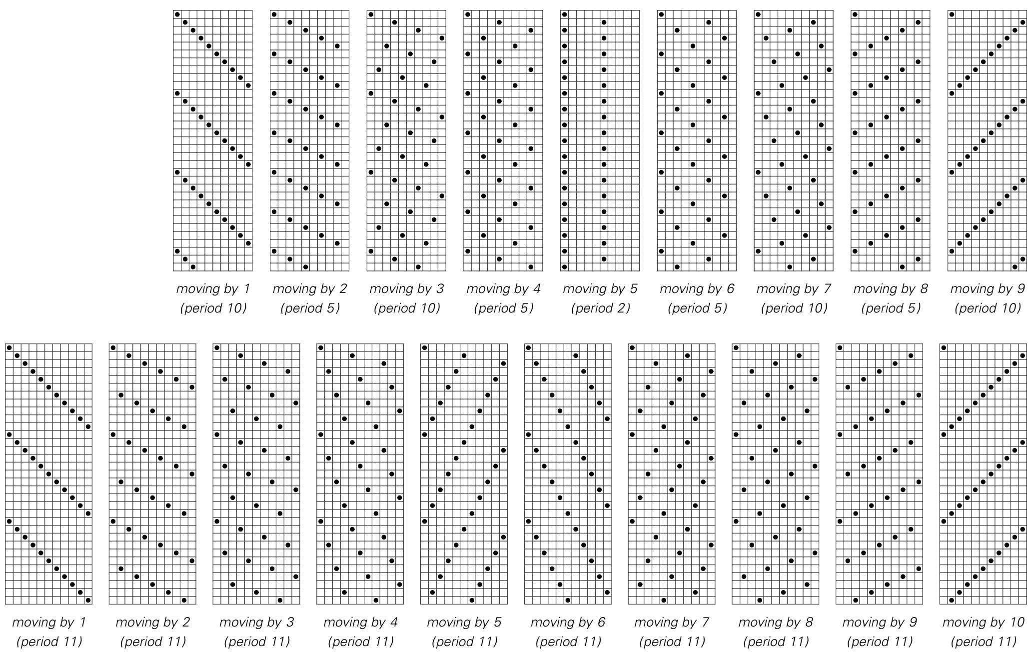


A simple system that contains a single dot which can be in one of six possible positions. At each step, the dot moves some number of positions to the right, wrapping around as soon as it reaches the right-hand end. The behavior of this system, like other systems of limited size, is always repetitive.

Looking at the pictures we then see that the behavior which results is always purely repetitive—though the period of repetition is different in different cases. And the basic reason for the repetitive behavior is that whenever the dot ends up in a particular position, it must always repeat whatever it did when it was last in that position.

But since there are only six possible positions in all, it is inevitable that after at most six steps the dot will always get to a position where it has been before. And this means that the behavior must repeat with a period of at most six steps.

The pictures below show more examples of the same setup, where now the number of possible positions is 10 and 11. In all cases, the behavior is repetitive, and the maximum repetition period is equal to the number of possible positions.

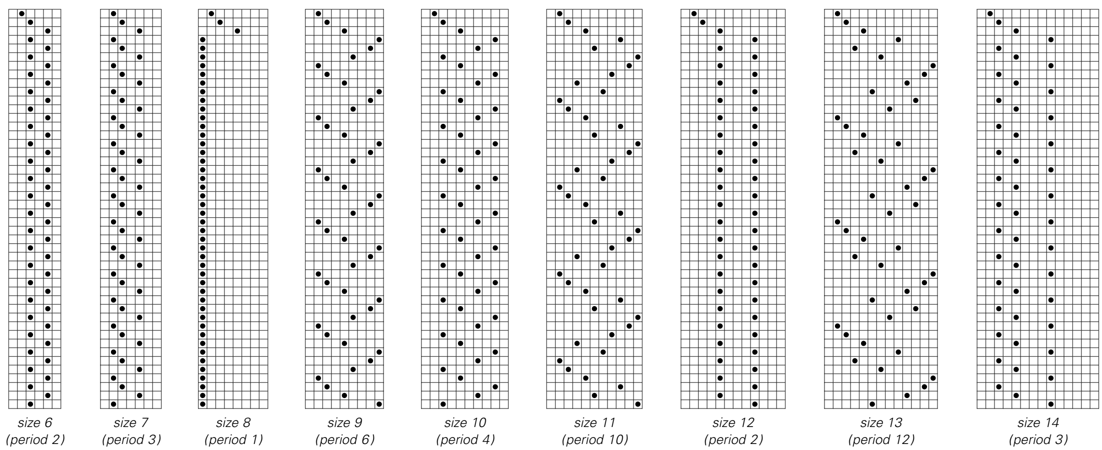


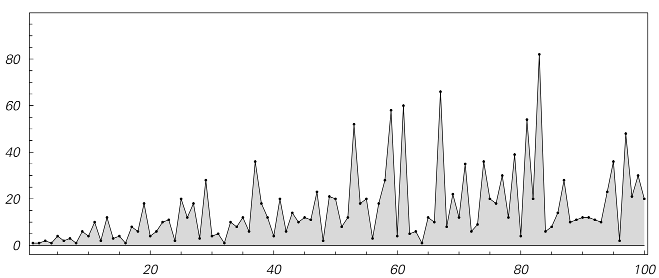
More examples of the type of system shown on the previous page, but now with 10 and 11 possible positions for the dot. The behavior always repeats itself in at most 10 or 11 steps. But the exact number of steps in each case depends on the prime factors of the numbers that define the system.

Sometimes the actual repetition period is equal to this maximum value. But often it is smaller. And indeed it is a common feature of systems of limited size that the repetition period one sees can depend greatly on the exact size of the system and the exact rule that it follows.

In the type of system shown on the facing page, it turns out that the repetition period is maximal whenever the number of positions moved at each step shares no common factor with the total number of possible positions—and this is achieved for example whenever either of these quantities is a prime number.

The pictures below show another example of a system of limited size based on a simple rule. The particular rule is at each step to double the number that represents the position of the dot, wrapping around as soon as this goes past the right-hand end.





A system where the number that represents the position of the dot doubles at each step, wrapping around whenever it reaches the right-hand end. (After t steps the dot is thus at position *Mod[2^t, n]* in a size n system.) The plot at left gives the repetition period for this system as a function of its size; for odd n this period is equal to *MultiplicativeOrder[2, n]*.

Once again, the behavior that results is always repetitive, and the repetition period can never be greater than the total number of possible positions for the dot. But as the picture shows, the actual repetition period jumps around considerably as the size of the system is changed. And as it turns out, the repetition period is again related to the factors of the number of possible positions for the dot—and tends to be maximal in those cases where this number is prime.

So what happens in systems like cellular automata?

The pictures on the facing page show some examples of cellular automata that have a limited number of cells. In each case the cells are in effect arranged around a circle, so that the right neighbor of the rightmost cell is the leftmost cell and vice versa.

And once again, the behavior of these systems is ultimately repetitive. But the period of repetition is often quite large.

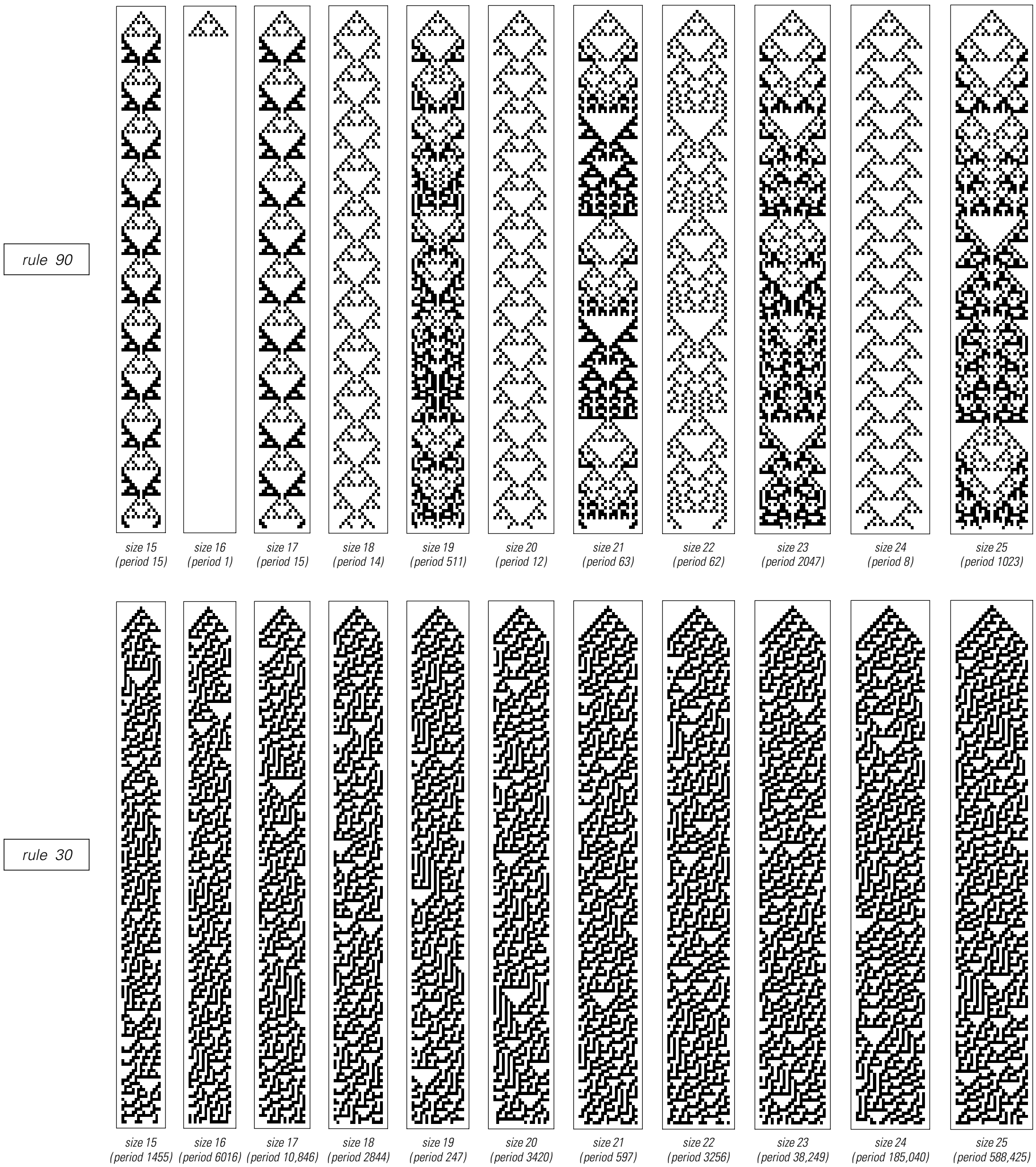
The maximum possible repetition period for any system is always equal to the total number of possible states of the system.

For the systems involving a single dot that we discussed above, the possible states correspond just to possible positions for the dot, and the number of states is therefore equal to the size of the system.

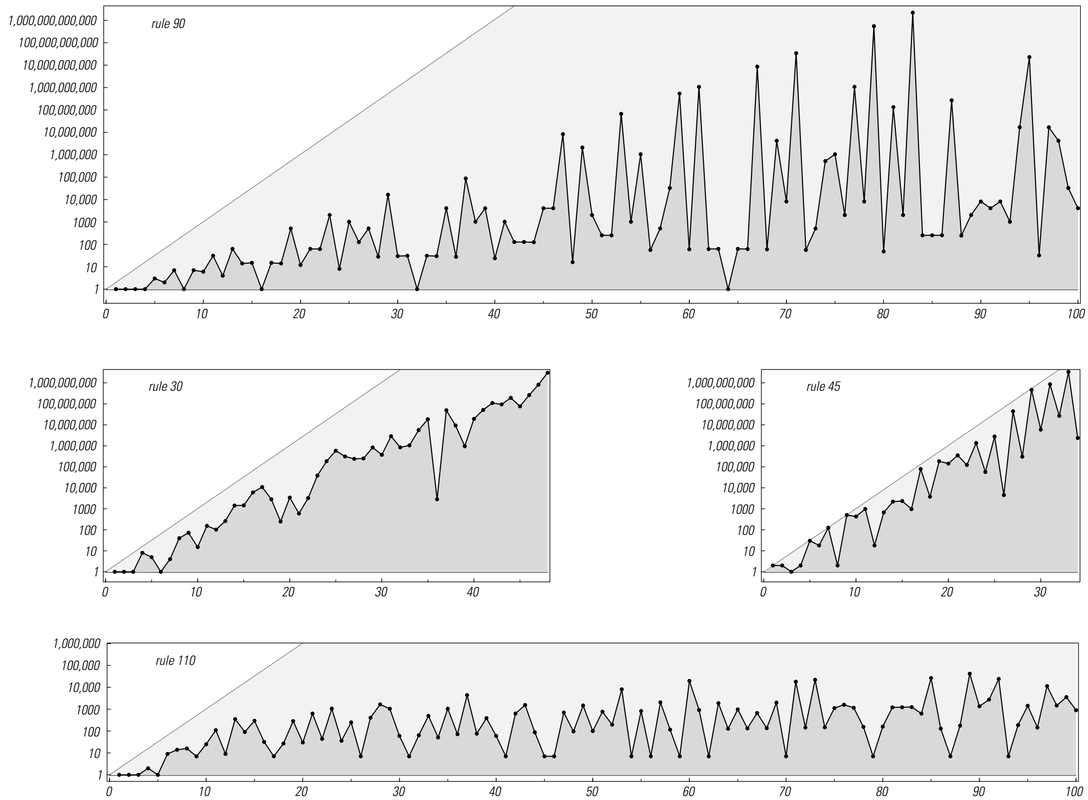
But in a cellular automaton, every possible arrangement of black and white cells corresponds to a possible state of the system. With n cells there are thus 2^n possible states. And this number increases very rapidly with the size n: for 5 cells there are already 32 states, for 10 cells 1024 states, for 20 cells 1,048,576 states, and for 30 cells 1,073,741,824 states.

The pictures on the next page show the actual repetition periods for various cellular automata. In general, a rapid increase with size is characteristic of class 3 behavior. Of the elementary rules, however, only rule 45 seems to yield periods that always stay close to the maximum of 2^n. And in all cases, there are considerable fluctuations in the periods that occur as the size changes.

So how does all of this relate to class 2 behavior? In the examples we have just discussed, we have explicitly set up systems that have limited size. But even when a system in principle contains an infinite number of cells it is still possible that a particular pattern in that system will only grow to occupy a limited number of cells. And in any



The behavior of cellular automata with a limited number of cells. In each case the right neighbor of the rightmost cell is taken to be the leftmost cell and vice versa. The pattern produced always eventually repeats, but the period of repetition can increase rapidly with the size of the system.



Repetition periods for various cellular automata as a function of size. The initial conditions used in each case consist of a single black cell, as in the pictures on the previous page. The dashed gray line indicates the maximum possible repetition period of 2^n. The maximum repetition period for rule 90 is $2^{(n-1)/2} - 1$. For rule 30, the peak repetition periods are of order $2^{0.63 n}$, while for rule 45, they are close to 2^n (for $n = 29$, for example, the period is 463,347,935, which is 86% of the maximum possible). For rule 110, the peaks seem to increase roughly like n^3.

such case, the pattern must repeat itself with a period of at most 2^n steps, where n is the size of the pattern.

In a class 2 system with random initial conditions, a similar thing happens: since different parts of the system do not communicate with each other, they all behave like separate patterns of limited size. And in fact in most class 2 cellular automata these patterns are effectively only a few cells across, so that their repetition periods are necessarily quite short.

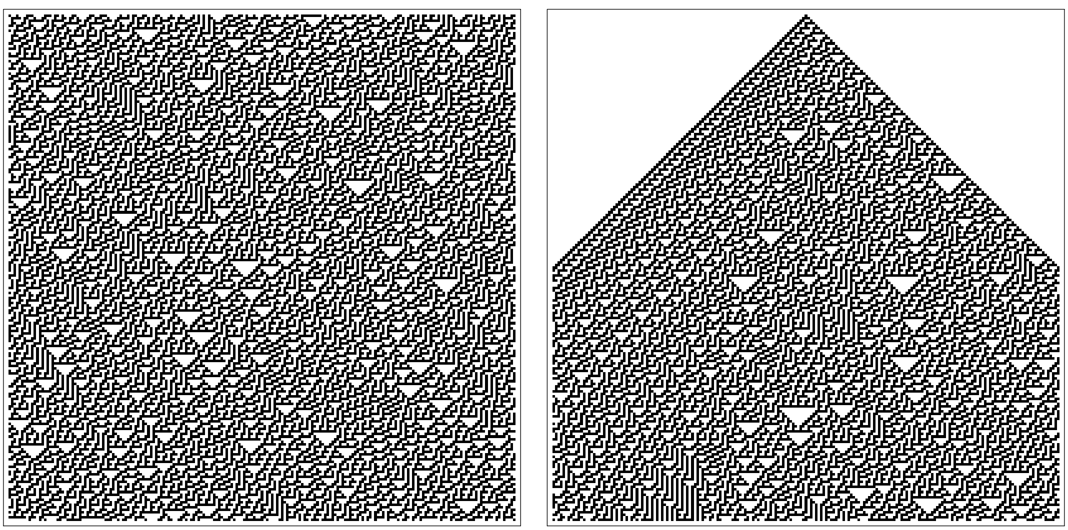
Randomness in Class 3 Systems

When one looks at class 3 systems the most obvious feature of their behavior is its apparent randomness. But where does this randomness ultimately come from? And is it perhaps all somehow just a reflection of randomness that was inserted in the initial conditions?

The presence of randomness in initial conditions—together with sensitive dependence on initial conditions—does imply at least some degree of randomness in the behavior of any class 3 system. And indeed when I first saw class 3 cellular automata I assumed that this was the basic origin of their randomness.

But the crucial point that I discovered only some time later is that random behavior can also occur even when there is no randomness in initial conditions. And indeed, in earlier chapters of this book we have already seen many examples of this fundamental phenomenon.

The pictures below now compare what happens in the rule 30 cellular automaton from page 27 if one starts from random initial conditions and from initial conditions involving just a single black cell.

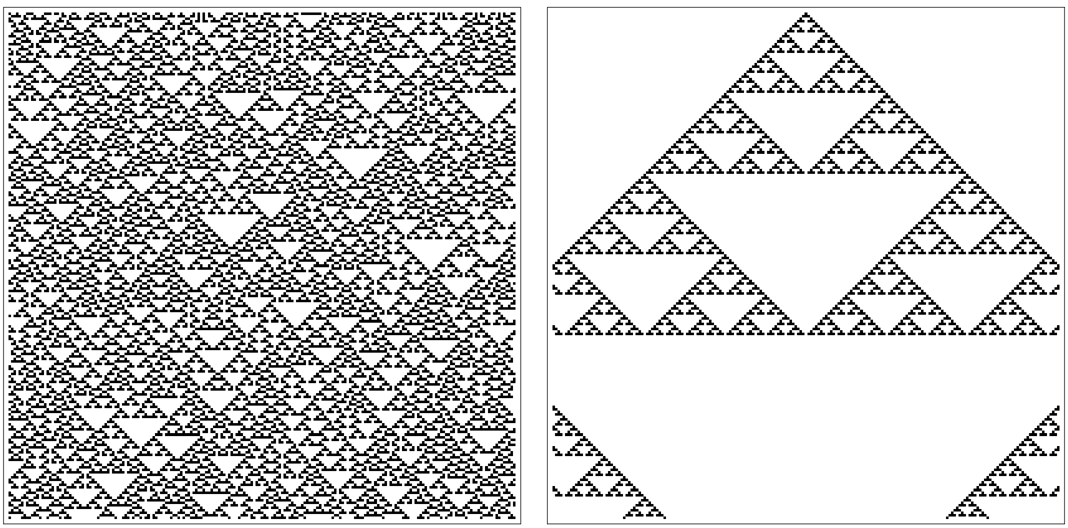


Comparison of the patterns produced by the rule 30 cellular automaton starting from random initial conditions and from simple initial conditions involving just a single black cell. Away from the edge of the second picture, the patterns look remarkably similar.

The behavior we see in the two cases rapidly becomes almost indistinguishable. In the first picture the random initial conditions certainly affect the detailed pattern that is obtained. But the crucial point is that even without any initial randomness much of what we see in the second picture still looks like typical random class 3 behavior.

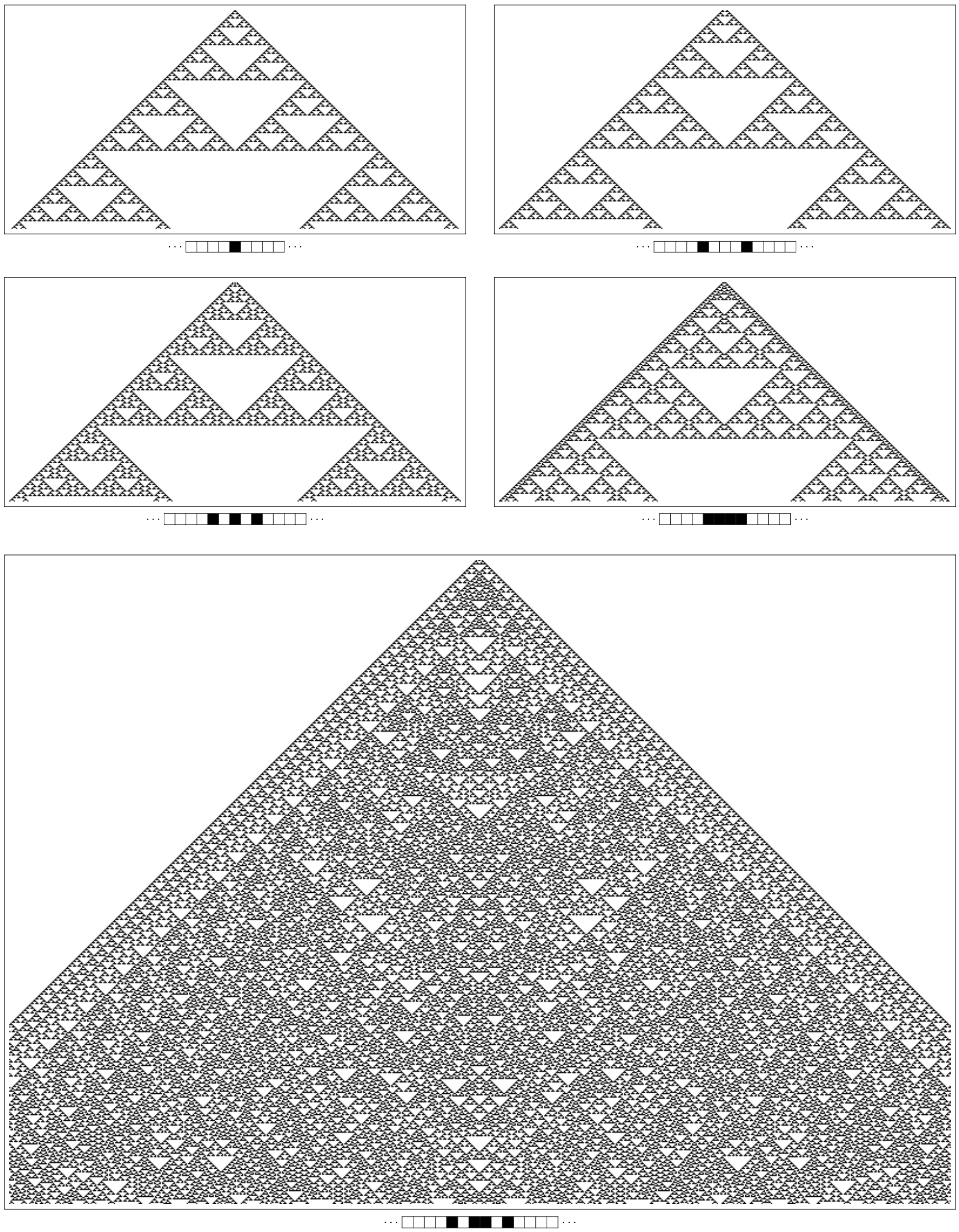
So what about other class 3 cellular automata? Do such systems always produce randomness even with simple initial conditions?

The pictures below show an example in which random class 3 behavior is obtained when the initial conditions are random, but where the pattern produced by starting with a single black cell has just a simple nested form.



Patterns produced by the rule 22 cellular automaton starting from random initial conditions and from an initial condition containing a single black cell. With random initial conditions typical class 3 behavior is seen. But with the specific initial condition shown on the right, a simple nested pattern is produced.

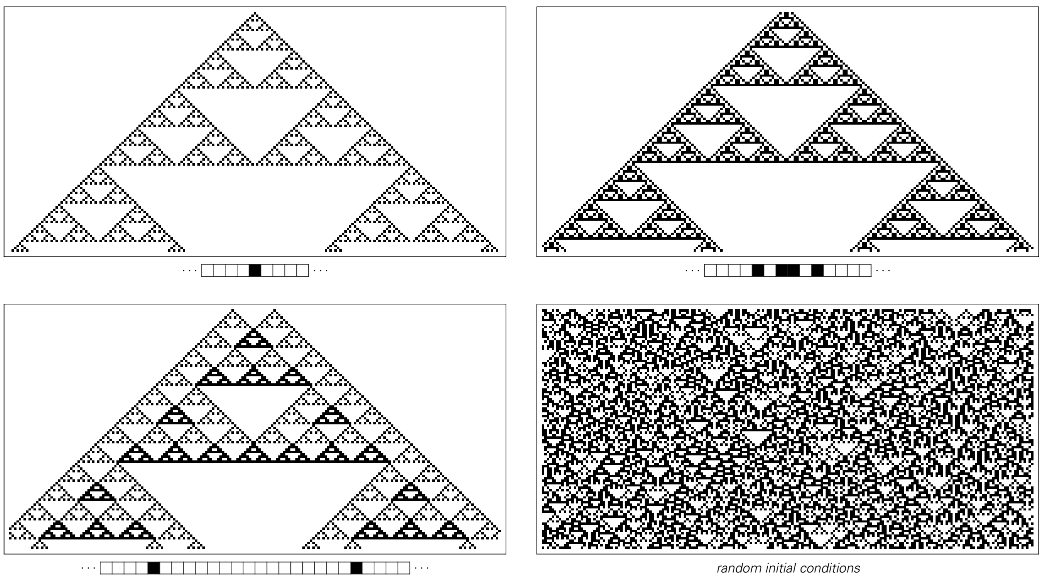
Nevertheless, the pictures on the facing page demonstrate that if one uses initial conditions that are slightly different—though still simple—then one can still see randomness in the behavior of this particular cellular automaton.



Rule 22 with various different simple initial conditions. In the top four cases, the pattern produced ultimately has a simple nested form. But in the bottom case, it is instead in many respects random, much like rule 30.

There are however a few cellular automata in which class 3 behavior is obtained with random initial conditions, but in which no significant randomness is ever produced with simple initial conditions.

The pictures below show one example. And in this case it turns out that all patterns are in effect just simple superpositions of the basic nested pattern that is obtained by starting with a single black cell.



Patterns generated by rule 90 with various initial conditions. This particular cellular automaton rule has the special property of additivity which implies that with any initial conditions the patterns that it produces can be obtained as simple superpositions of the first pattern shown above. Any initial condition that contains black cells only in a limited region will thus lead to a pattern that ultimately has a simple nested form. Unlike rule 30 or rule 22 therefore, rule 90 cannot intrinsically generate randomness starting from simple initial conditions. The randomness in the last picture shown here is thus purely a consequence of the randomness in its initial conditions. Note that the pictures above show only half as many steps of evolution as the corresponding pictures of rule 22 on the previous page.

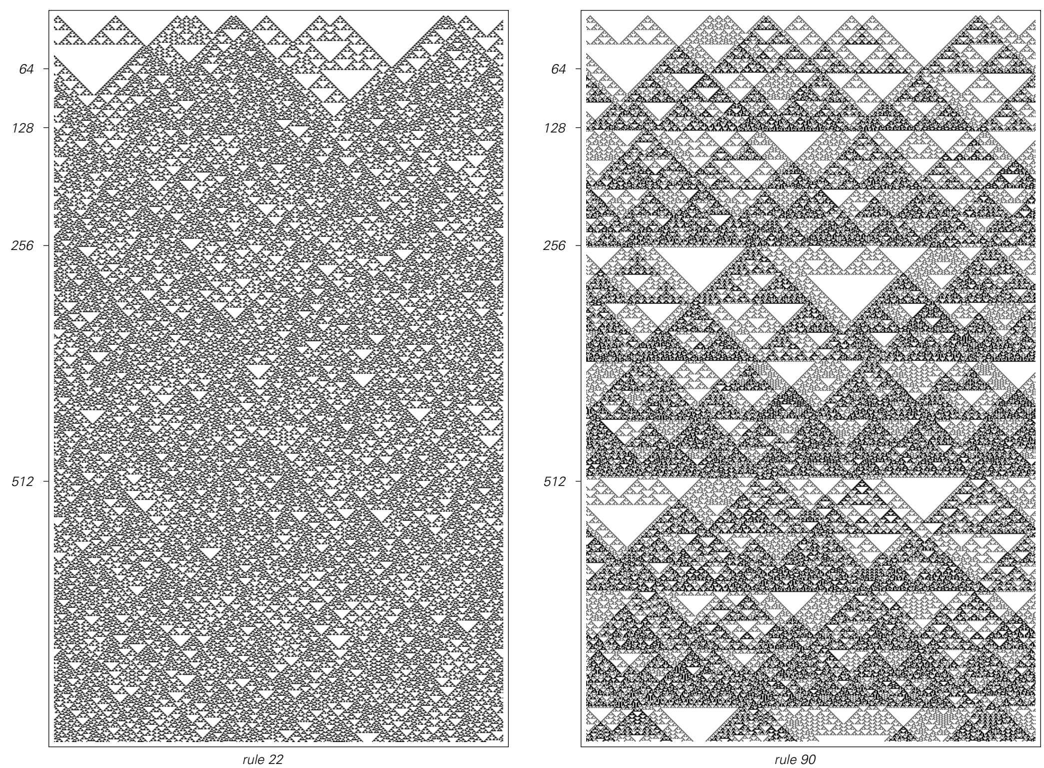
As a result, when the initial conditions involve only a limited region of black cells, the overall pattern produced always ultimately has a simple nested form. Indeed, at each of the steps where a new white triangle starts in the center, the whole pattern consists just of two copies of the region of black cells from the initial conditions.

The only way to get a random pattern therefore is to have an infinite number of randomly placed black cells in the initial conditions.

And indeed when random initial conditions are used, rule 90 does manage to produce random behavior of the kind expected in class 3.

But if there are deviations from perfect randomness in the initial conditions, then these will almost inevitably show up in the evolution of the system. And thus, for example, if the initial density of black cells is low, then correspondingly low densities will occur again at various later steps, as in the second picture below.

With rule 22, on the other hand, there is no such effect, and instead after just a few steps no visible trace remains of the low density of initial black cells.



Examples of evolution from random initial conditions with a low density of black cells. In rule 22 the low initial density has no long-term effect. But in rule 90 its effect continues forever. The reason for this difference is that in rule 22 the randomness we see is intrinsically generated by the evolution of the system, while in rule 90 it comes from randomness in the initial conditions.

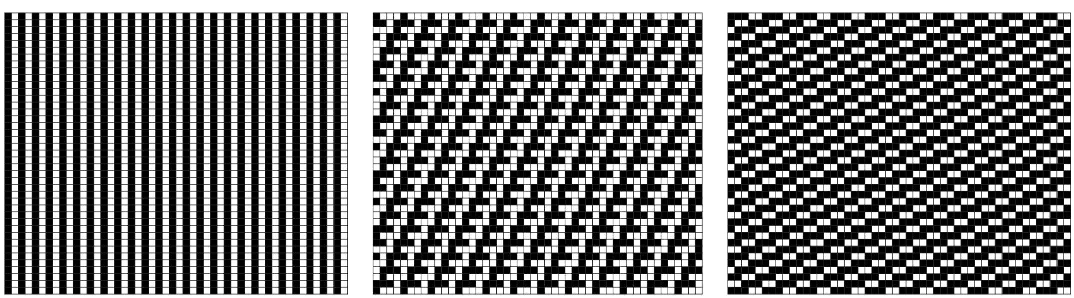
A couple of sections ago we saw that all class 3 systems have the property that the detailed patterns they produce are highly sensitive to detailed changes in initial conditions. But despite this sensitivity at the level of details, the point is that any system like rule 22 or rule 30 yields patterns whose overall properties depend very little on the form of the initial conditions that are given.

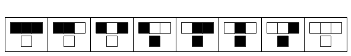
By intrinsically generating randomness such systems in a sense have a certain fundamental stability: for whatever is done to their initial conditions, they still give the same overall random behavior, with the same large-scale properties. And as we shall see in the next few chapters, there are in fact many systems in nature whose apparent stability is ultimately a consequence of just this kind of phenomenon.

Special Initial Conditions

We have seen that cellular automata such as rule 30 generate seemingly random behavior when they are started both from random initial conditions and from simple ones. So one may wonder whether there are in fact any initial conditions that make rule 30 behave in a simple way.

As a rather trivial example, one certainly knows that if its initial state is uniformly white, then rule 30 will just yield uniform white forever. But as the pictures below demonstrate, it is also possible to find less trivial initial conditions that still make rule 30 behave in a simple way.





Examples of special initial conditions that make the rule 30 cellular automaton yield simple repetitive behavior. Small patches with the same structures as shown here can be seen embedded in typical random patterns produced by rule 30. At left is a representation of rule 30. Finding initial conditions that make cellular automata yield behavior with certain repetition periods is closely related to the problem of satisfying constraints discussed on page 210.

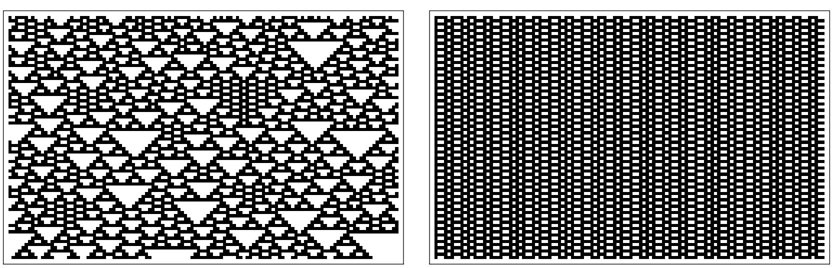
In fact, it turns out that in any cellular automaton it is inevitable that initial conditions which consist just of a fixed block of cells repeated forever will lead to simple repetitive behavior.

For what happens is that each block in effect independently acts like a system of limited size. The right-hand neighbor of the rightmost cell in any particular block is the leftmost cell in the next block, but since all the blocks are identical, this cell always has the same color as the leftmost cell in the block itself. And as a result, the block evolves just like one of the systems of limited size that we discussed on page 255. So this means that given a block that is n cells wide, the repetition period that is obtained must be at most 2^n steps.

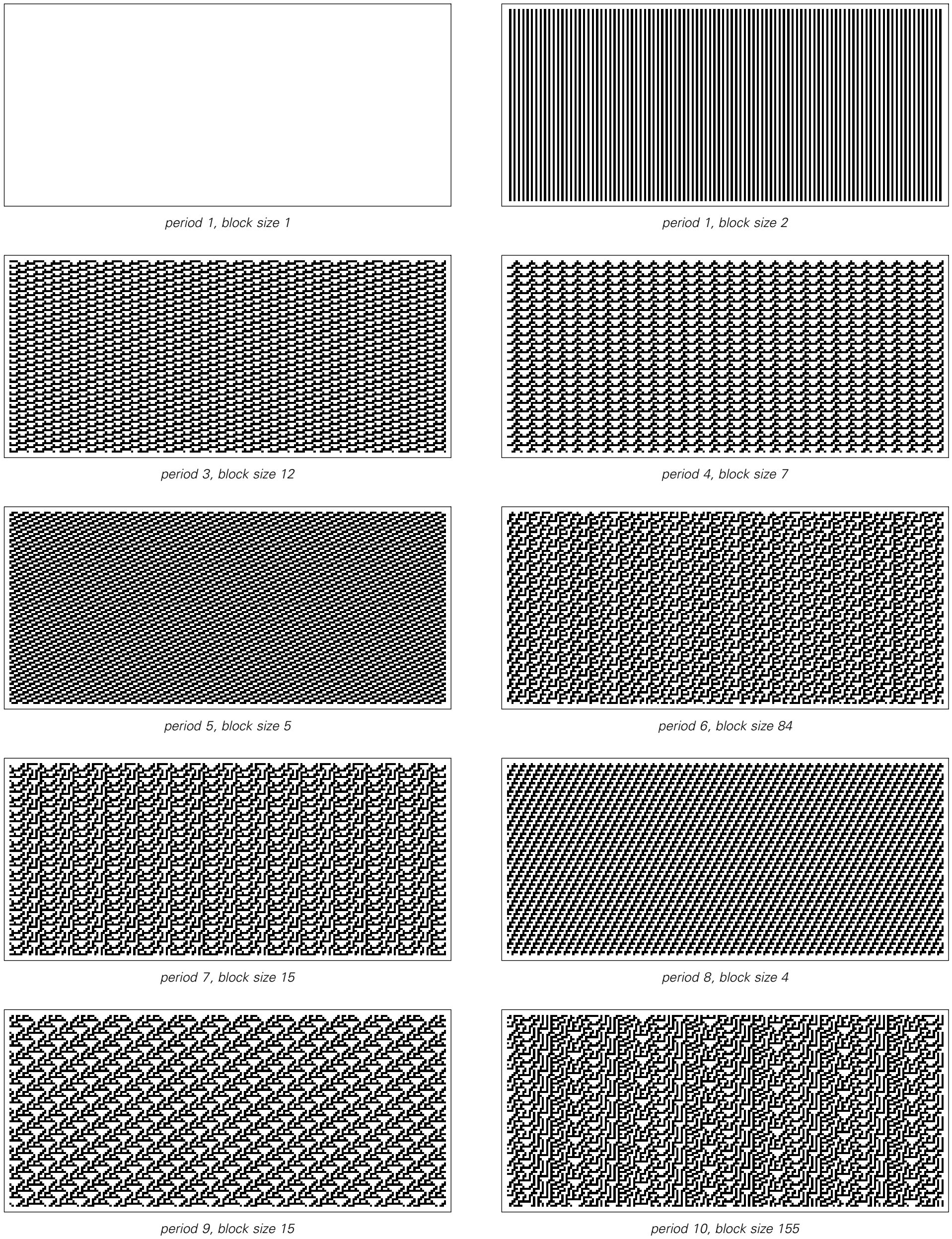
But if one wants a short repetition period, then there is a question of whether there is a block of any size which can produce it. The pictures on the next page show the blocks that are needed to get repetition periods of up to ten steps in rule 30. It turns out that no block of any size gives a period of exactly two steps, but blocks can be found for all larger periods at least up to 15 steps.

But what about initial conditions that do not just consist of a single block repeated forever? It turns out that for rule 30, no other kind of initial conditions can ever yield repetitive behavior.

But for many rules—including a fair number of class 3 ones—the situation is different. And as one example the picture on the right below shows an initial condition for rule 126 that involves two different blocks but which nevertheless yields repetitive behavior.



Rule 126 with a typical random initial condition, and with an initial condition that consists of a random sequence of the blocks ■■□□ and ■■■□. Rule 126 in general shows class 3 behavior, as on the left. But with the special initial condition on the right it acts like a simple class 2 rule. Note the patches of class 2 behavior even in the picture on the left.



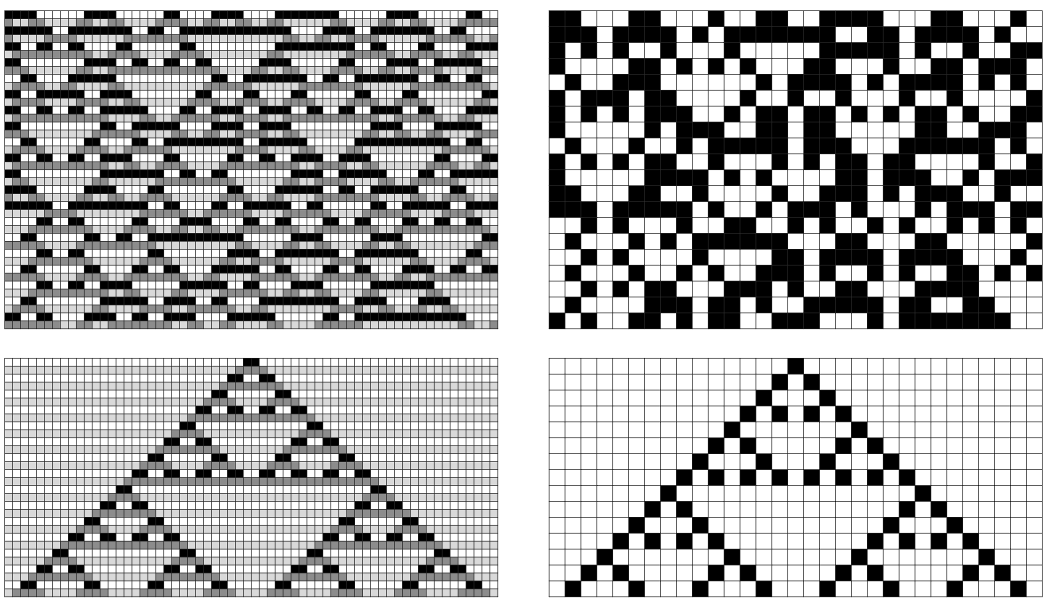
All patterns that repeat in 10 or less steps under evolution according to rule 30. In each case the initial conditions consist of a fixed block of cells that is repeated over and over again. Note that there are no initial conditions that yield a repetition period of exactly 2 steps. To get period 11, a block that contains 275 cells is required.

In a sense what is happening here is that even though rule 126 usually shows class 3 behavior, it is possible to find special initial conditions that make it behave like a simple class 2 rule.

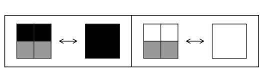
And in fact it turns out to be quite common for there to exist special initial conditions for one cellular automaton that make it behave just like some other cellular automaton.

Rule 126 will for example behave just like rule 90 if one starts it from special initial conditions that contain only blocks consisting of pairs of black and white cells.

The pictures below show how this works: on alternate steps the arrangement of blocks in rule 126 corresponds exactly to the arrangement of individual cells in rule 90. And among other things this explains why it is that with simple initial conditions rule 126 produces exactly the same kind of nested pattern as rule 90.



rule 126 *rule 90*



Two examples of the fact that with special initial conditions rule 126 behaves exactly like rule 90. The initial conditions that are used consist of blocks of cells where each block contains either two black cells or two white cells. If one looks only on every other step, then the blocks behave exactly like individual cells in rule 90. This correspondence is the basic reason that rule 126 produces the same kind of nested patterns as rule 90 when it is started from simple initial conditions.

The point is that these initial conditions in effect contain only blocks for which rule 126 behaves like rule 90. And as a result, the overall patterns produced by rule 126 in this case are inevitably exactly like those produced by rule 90.

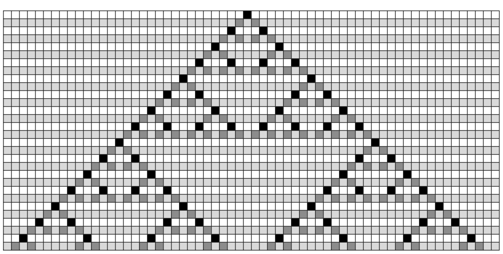
So what about other cellular automata that can yield similar patterns? In every example in this book where nested patterns like those from rule 90 are obtained it turns out that the underlying rules that are responsible can be set up to behave exactly like rule 90. Sometimes this will happen, say, for any initial condition that has black cells only in a limited region. But in other cases—like the example of rule 22 on page 263—rule 90 behavior is obtained only with rather specific initial conditions.

So what about rule 90 itself? Why does it yield nested patterns?

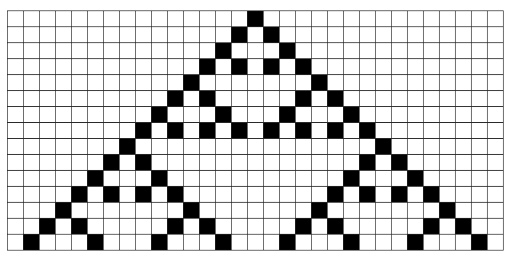
The basic reason can be thought of as being that just as other rules can emulate rule 90 when their initial conditions contain only certain blocks, so also rule 90 is able to emulate itself in this way.

The picture below shows how this works. The idea is to consider the initial conditions not as a sequence of individual cells, but rather as a sequence of blocks each containing two adjacent cells. And with an appropriate form for these blocks what one finds is that the configuration of blocks evolves exactly according to rule 90.

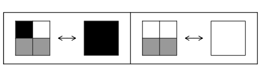
The fact that both individual cells and whole blocks of cells evolve according to the same rule then means that whatever pattern is



rule 90



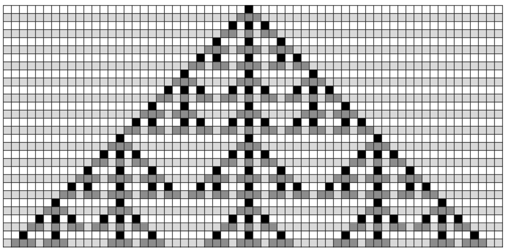
rule 90



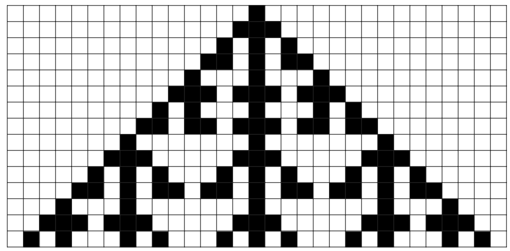
A demonstration of the fact that in rule 90 blocks of cells can behave just like individual cells. One consequence of this is that the patterns produced by rule 90 have a nested or self-similar form.

produced must have exactly the same structure whether it is looked at in terms of individual cells or in terms of blocks of cells. And this can be achieved in only two ways: either the pattern must be essentially uniform, or it must have a nested structure—just like we see in rule 90.

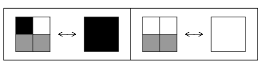
So what happens with other rules? It turns out that the property of self-emulation is rather rare among cellular automaton rules. But one other example is rule 150—as illustrated in the picture below.



rule 150



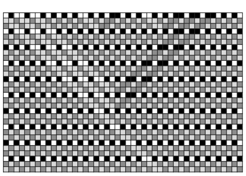
rule 150



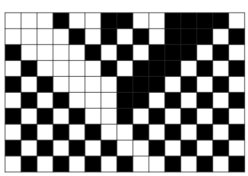
Another example of a rule in which blocks of cells can behave just like individual cells. Rule 90 and rule 150 are also essentially the only fundamentally different elementary cellular automaton rules that have the property of being additive (see page 264).

So what else is there in common between rule 90 and rule 150? It turns out that they are both additive rules, implying that the patterns they produce can be superimposed in the way we discussed on page 264. And in fact one can show that any rule that is additive will be able to emulate itself and will thus yield nested patterns. But there are rather few additive rules, and indeed with two colors and nearest neighbors the only fundamentally different ones are precisely rules 90 and 150.

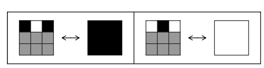
Ultimately, however, additive rules are not the only ones that can emulate themselves. An example of another kind is rule 184, in which blocks of three cells can act like a single cell, as shown below.



rule 184



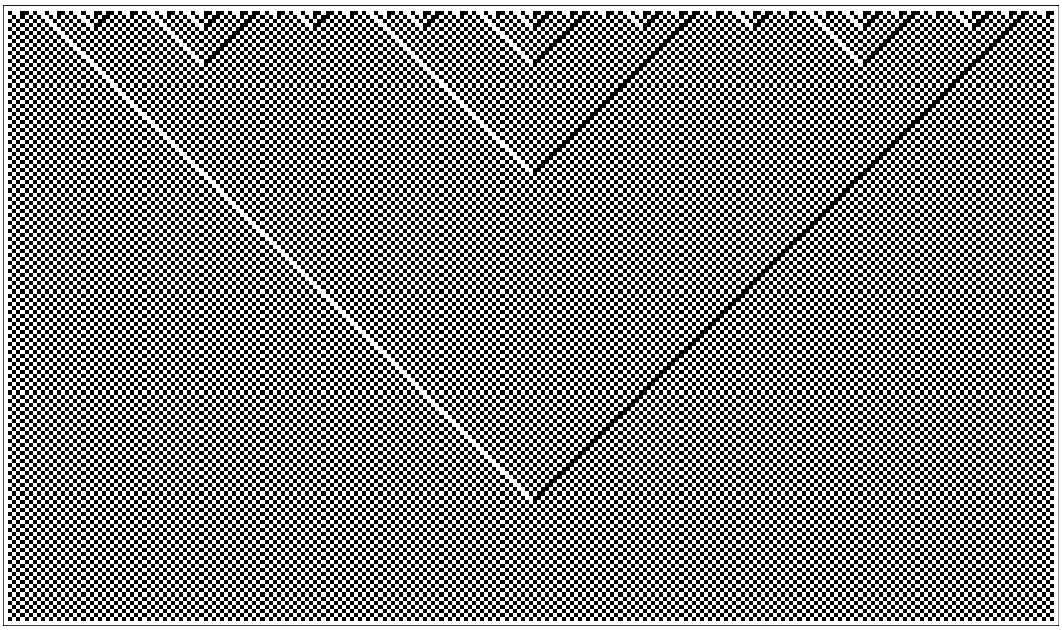
rule 184

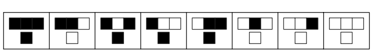


A rule that is not additive, but in which blocks of cells can again behave just like individual cells.

With simple initial conditions of the type we have used so far this rule will always produce essentially trivial behavior. But one way to see the properties of the rule is to use nested initial conditions, obtained for example from substitution systems of the kind we discussed on page 82.

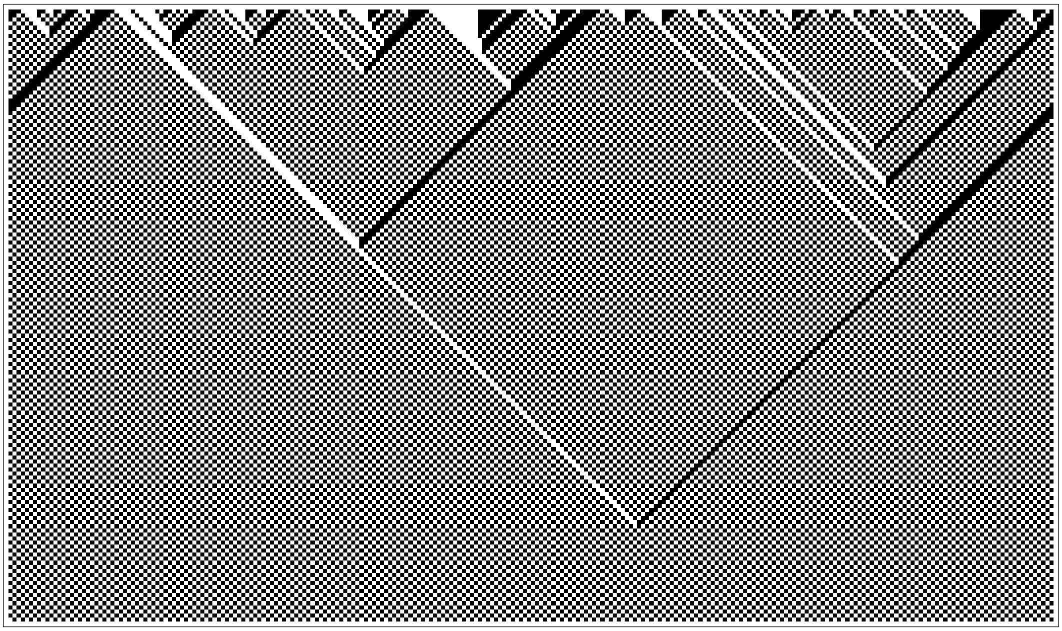
With most rules, including 90 and 150, such nested initial conditions typically yield results that are ultimately indistinguishable from those obtained with typical random initial conditions. But for rule 184, an appropriate choice of nested initial conditions yields the highly regular pattern shown below.





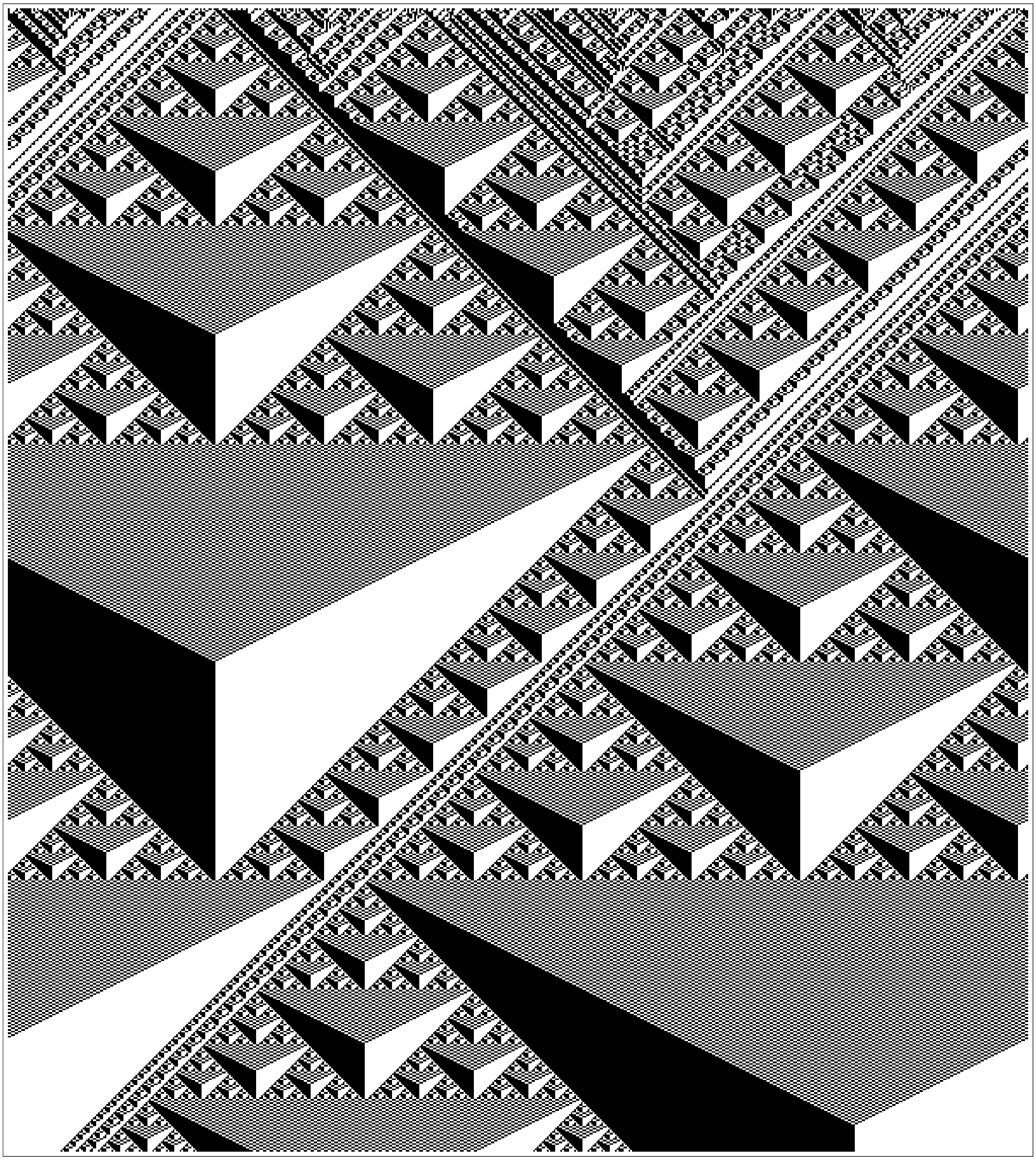
The pattern produced by rule 184 (shown at left) evolving from a nested initial condition. The particular initial condition shown can be obtained by applying the substitution system ■ → ■□■, □ → □□■, starting from a single black element ■ (see page 83). With this initial condition, rule 184 exhibits an equal number of black and white stripes, which annihilate in pairs so as to yield a regular nested pattern.

The nested structure seen in this pattern can then be viewed as a consequence of the fact that rule 184 is able to emulate itself. And the picture below shows that rule 184—unlike any of the additive rules—still produces recognizably nested patterns even when the initial conditions that are used are random.



Rule 184 evolving from a random initial condition. Nested structure similar to what we saw in the previous picture is still visible. The presence of such structure is most obvious when there are equal numbers of black and white cells in the initial conditions, but it does not rely on any regularity in the arrangement of these cells.

As we will see on page 338 the presence of such patterns is particularly clear when there are equal numbers of black and white cells in the initial conditions—but how these cells are arranged does not usually matter much at all. And in general it is possible to find quite a few cellular automata that yield nested patterns like rule 184 even from random initial conditions. The picture on the next page shows a particularly striking example in which explicit regions are formed that contain patterns with the same overall structure as rule 90.



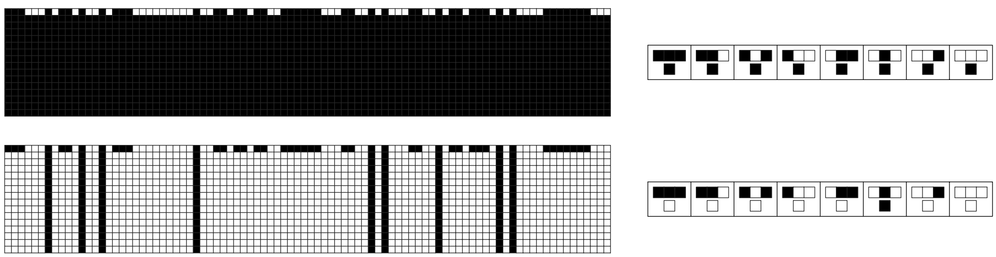
Another example of a cellular automaton that produces a nested pattern even from random initial conditions. The particular rule shown involves next-nearest as well as nearest neighbors and has rule number 4067213884. As in rule 184, the nested behavior seen here is most obvious when the density of black and white cells in the initial conditions is equal.

The Notion of Attractors

In this chapter we have seen many examples of patterns that can be produced by starting from random initial conditions and then following the evolution of cellular automata for many steps.

But what can be said about the individual configurations of black and white cells that appear at each step? In random initial conditions, absolutely any sequence of black and white cells can be present. But it is a feature of most cellular automata that on subsequent steps the sequences that can be produced become progressively more restricted.

The first picture below shows an extreme example of a class 1 cellular automaton in which after just one step the only sequences that can occur are those that contain only black cells.

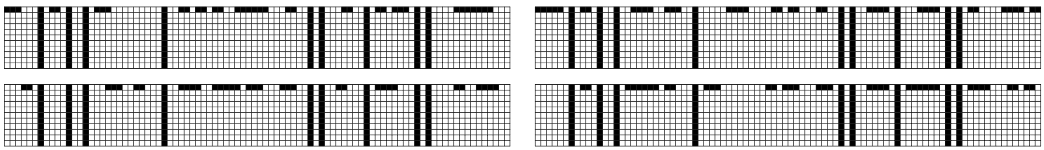


Examples of simple cellular automata that evolve after just one step to attractors in which only certain sequences of black and white cells can occur. In the first case, the sequences that can occur are ones that involve only black cells. In the second case, the sequences are ones in which every black cell is surrounded by white cells. The rules shown are numbers 255 and 4.

The resulting configuration can be thought of as a so-called attractor for the cellular automaton evolution. It does not matter what initial conditions one starts from: one always reaches the same all-black attractor in the end. The situation is somewhat similar to what happens in a mechanical system like a physical pendulum. One can start the pendulum swinging in any configuration, but it will always tend to evolve to the configuration in which it is hanging straight down.

The second picture above shows a class 2 cellular automaton that once again evolves to an attractor after just one step. But now the attractor does not just consist of a single configuration, but instead consists of all configurations in which black cells occur only when they are surrounded on each side by at least one white cell.

The picture below shows that for any particular configuration of this kind, there are in general many different initial conditions that can lead to it. In a mechanical analogy each possible final configuration is like the lowest point in a basin—and a ball started anywhere in the basin will then always roll to that lowest point.



Four different initial conditions that all lead to the same final state in the rule 4 cellular automaton shown on the previous page. The final state can be thought of as one of the possible attractors for the evolution of the cellular automaton; the initial conditions shown then represent different elements in the basin of attraction for this attractor.

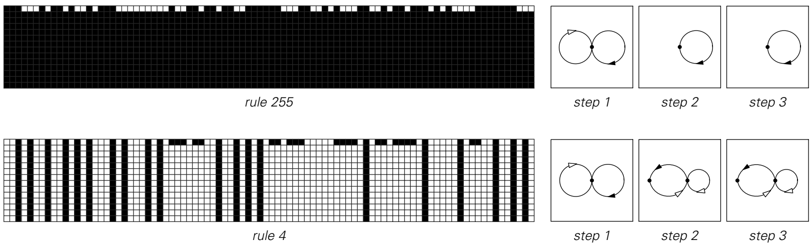
For one-dimensional cellular automata, it turns out that there is a rather compact way to summarize all the possible sequences of black and white cells that can occur at any given step in their evolution.

The basic idea is to construct a network in which each such sequence of black and white cells corresponds to a possible path.

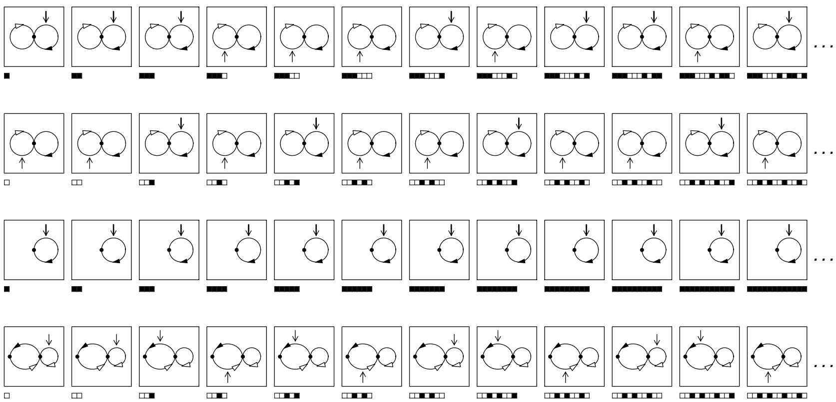
In the pictures at the top of the facing page, the first network in each case represents random initial conditions in which any possible sequence of black and white cells can occur. Starting from the node in the middle, one can go around either the left or the right loop in the network any number of times in any order—representing the fact that black and white cells can appear any number of times in any order.

At step 2 in the rule 255 example on the facing page, however, the network has only one loop—representing the fact that at this step the only sequences which can occur with this rule are ones that consist purely of black cells, just as we saw on the previous page.

The case of rule 4 is slightly more complicated: at step 2, the possible sequences that can occur are now represented by a network with two nodes. Starting at the right-hand node one can go around the loop to the right any number of times, corresponding to sequences of



Networks representing possible sequences of black and white cells that can occur at successive steps in the evolution of the two cellular automata shown on the left. In each case the possible sequences correspond to possible paths through the network. Both rules start on step 1 from random initial conditions in which all sequences of black and white cells are allowed. On subsequent steps, rule 255 allows only sequences containing just black cells, while rule 4 allows sequences that contain both black and white cells, but requires that every black cell be surrounded by white cells.

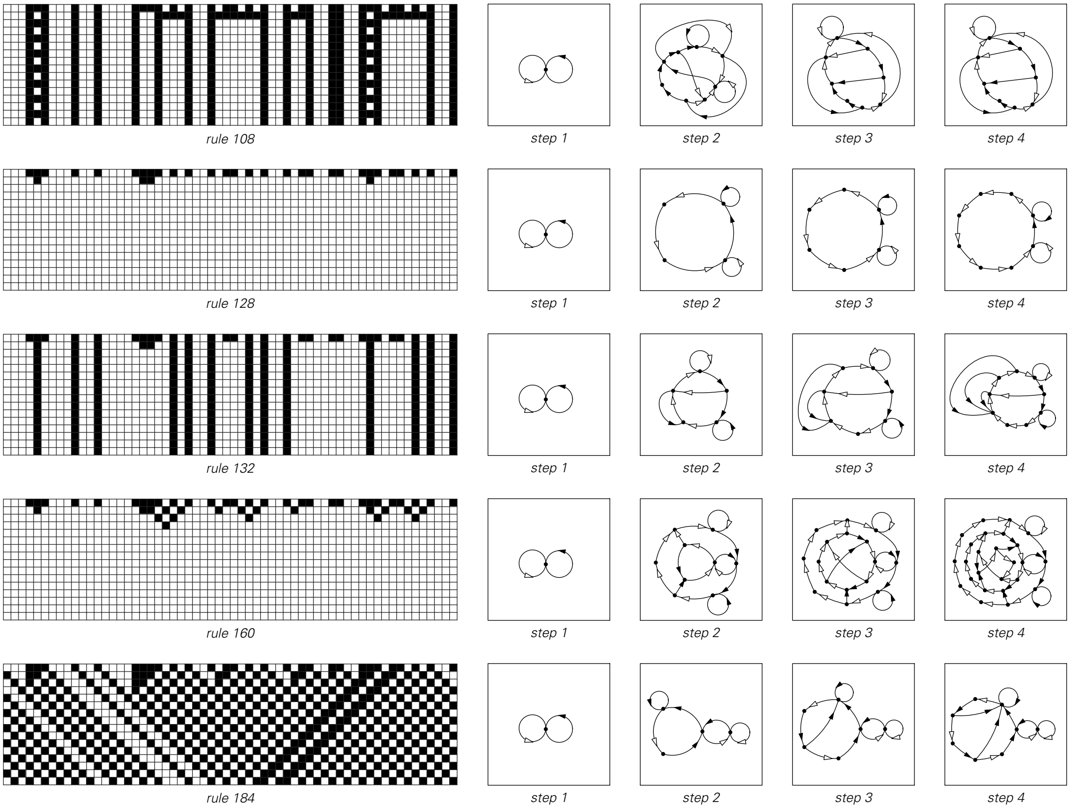


any number of white cells. At any point one can follow the arrow to the left to get a black cell, but the form of the network implies that this black cell must always be followed by at least one white cell.

The pictures on the next page show more examples of class 1 and 2 cellular automata. Unlike in the picture above, these rules do not reach their final states after one step, but instead just progressively evolve towards these states. And in the course of this evolution, the set of sequences that can occur becomes progressively smaller.

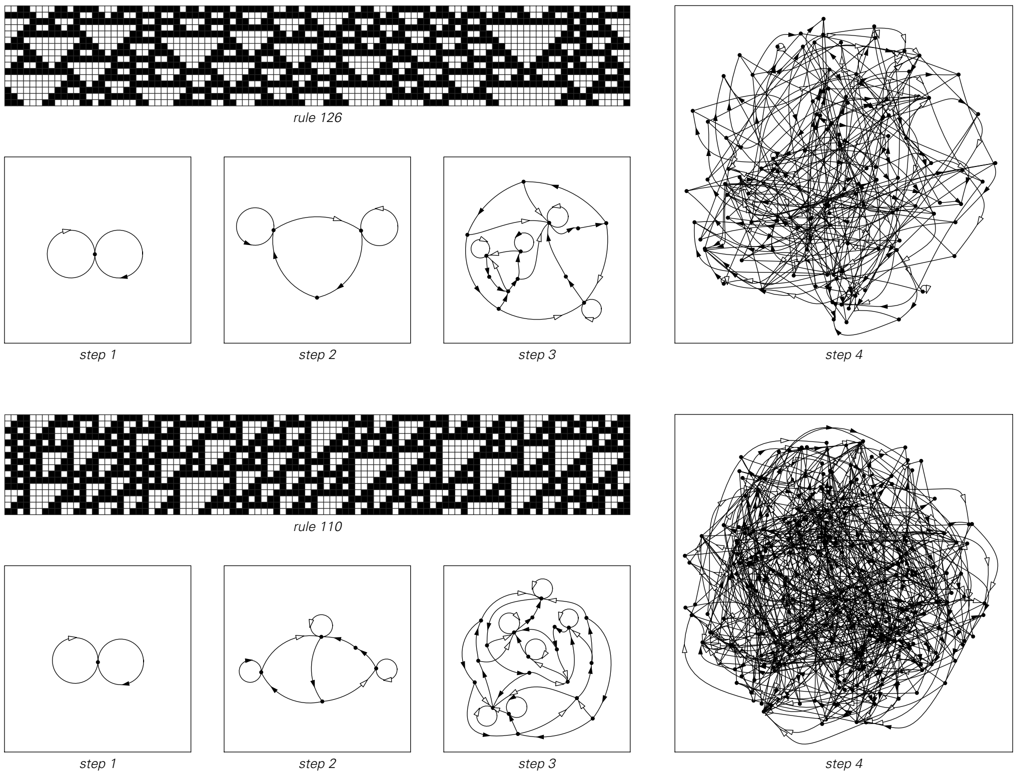
In rule 128, for example, the fact that regions of black shrink by one cell on each side at each step means that any region of black that exists after *t* steps must have at least *t* white cells on either side of it.

The networks shown on the next page capture all effects like this. And to do this we see that on successive steps they become somewhat more complicated. But at least for these class 1 and 2 examples, the progression of networks always continues to have a fairly simple form.



Networks representing possible sequences of black and white cells that can occur at successive steps in the evolution of several class 1 and 2 cellular automata. These networks never have more than about t^2 nodes after t steps.

So what happens with class 3 and 4 systems? The pictures on the facing page show a couple of examples. In rule 126, the only effect at step 2 is that black cells can no longer appear on their own: they must always be in groups of two or more. By step 3, it becomes difficult to see any change if one just looks at an explicit picture of the cellular automaton evolution. But from the network, one finds that now an infinite collection of other blocks are forbidden, beginning with the length 12 block. And on later steps, the set of sequences that are allowed rapidly becomes more complicated—as reflected in a rapid increase in the complexity of the corresponding networks.



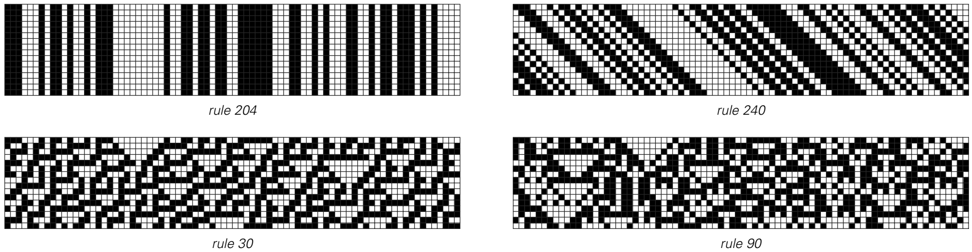
Networks representing possible sequences of black and white cells that can occur at successive steps in the evolution of typical class 3 and 4 cellular automata. The number of nodes in these networks seems to increase at a rate that is at least exponential.

Indeed, this kind of rapid increase in network complexity is a general characteristic of most class 3 and 4 rules. But it turns out that there are a few rules which at first appear to be exceptions.

The pictures at the top of the next page show four different rules that each have the property that if started from initial conditions in which all possible sequences of cells are allowed, these same sequences can all still occur at any subsequent step in the evolution.

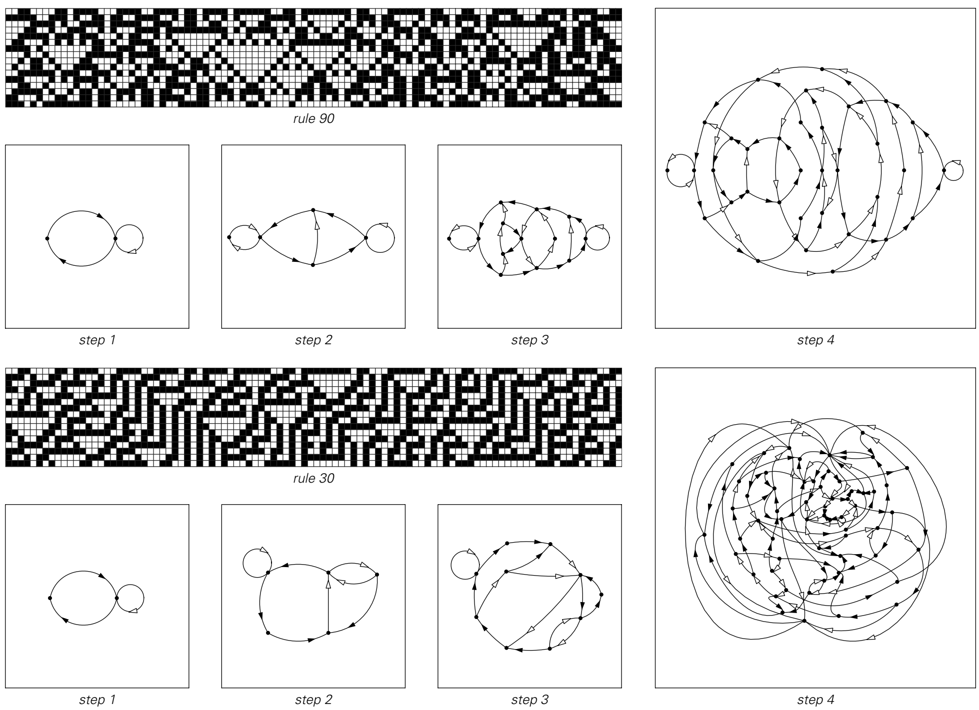
The first two rules that are shown exhibit very simple class 2 behavior. But the last two show typical class 3 behavior.

What is going on, however, is that in a sense the particular initial conditions that allow all possible sequences are special for these rules.



Examples of cellular automata which continue to allow all possible sequences of black and white cells at any step in their evolution. Such cellular automata in effect define what are known as surjective or onto mappings.

And indeed if one starts with almost any other initial conditions—say for example ones that do not allow any pair of black cells together, then as the pictures below illustrate, rapidly increasing complexity in the sets of sequences that are allowed is again observed.



Networks representing possible sequences that can occur in the evolution of the cellular automata at the top of the page, starting from initial conditions in which black cells are only allowed to appear in pairs.

Structures in Class 4 Systems

The next page shows three typical examples of class 4 cellular automata. In each case the initial conditions that are used are completely random. But after just a few steps, the systems organize themselves to the point where definite structures become visible.

Most of these structures eventually die out, sometimes in rather complicated ways. But a crucial feature of any class 4 systems is that there must always be certain structures that can persist forever in it.

So how can one find out what these structures are for a particular cellular automaton? One approach is just to try each possible initial condition in turn, looking to see whether it leads to a new persistent structure. And taking the code 20 cellular automaton from the top of the next page, the page that follows shows what happens in this system with each of the first couple of hundred possible initial conditions.

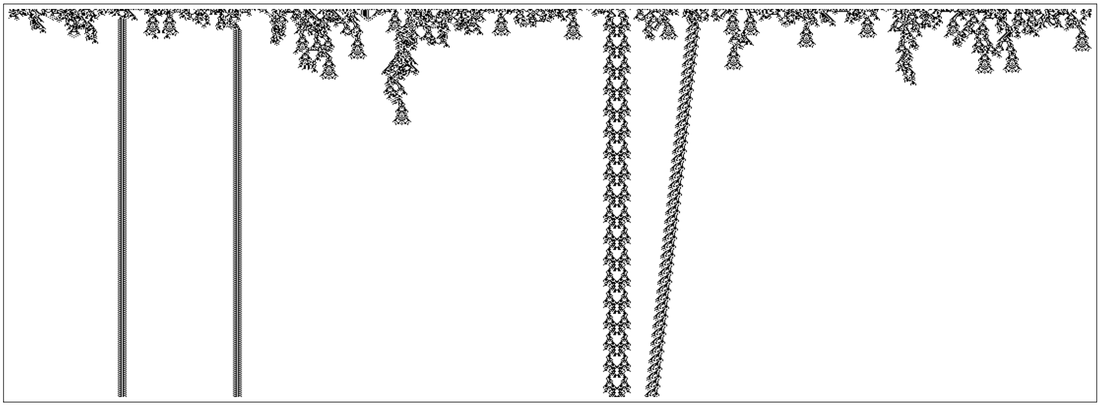
In most cases everything just dies out. But when we reach initial condition number 151 we finally see a structure that persists.

This particular structure is fairly simple: it just remains fixed in position and repeats every two steps. But not all persistent structures are that simple. And indeed at initial condition 187 we see a considerably more complicated structure, that instead of staying still moves systematically to the right, repeating its basic form only every 9 steps.

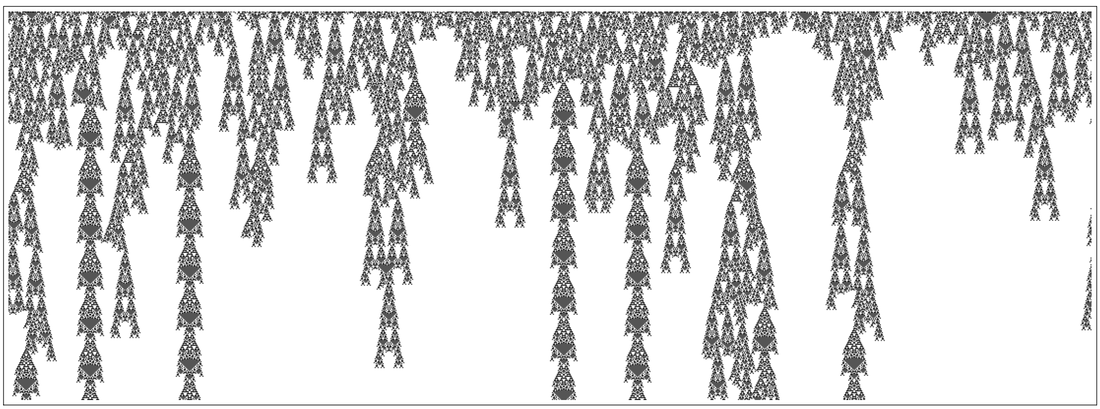
The existence of structures that move is a fundamental feature of class 4 systems. For as we discussed on page 252, it is these kinds of structures that make it possible for information to be communicated from one part of a class 4 system to another—and that ultimately allow the complex behavior characteristic of class 4 to occur.

But having now seen the structure obtained with initial condition 187, we might assume that all subsequent structures that arise in the code 20 cellular automaton must be at least as complicated. It turns out, however, that initial condition 189 suddenly yields a much simpler structure—that just stays unchanged in one position at every step.

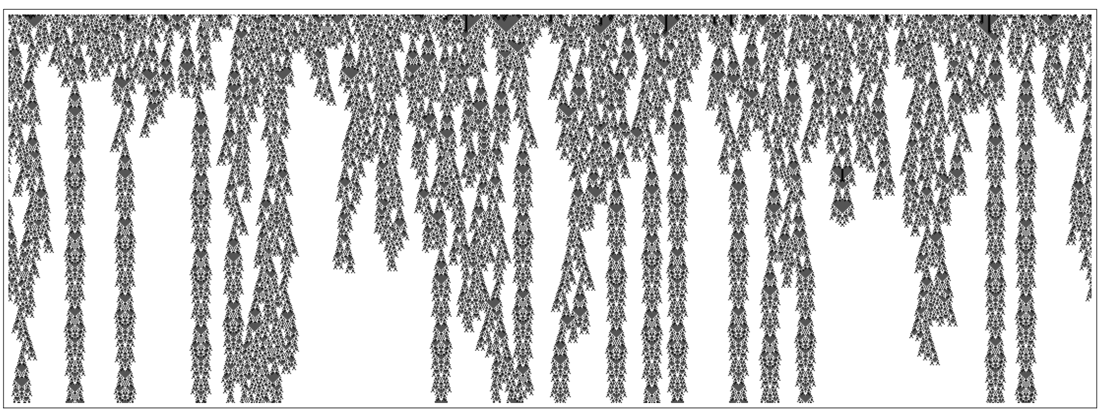
But going on to initial condition 195, we again find a more complicated structure—this time one that repeats only every 22 steps.



2 colors, next-nearest neighbors, code 20

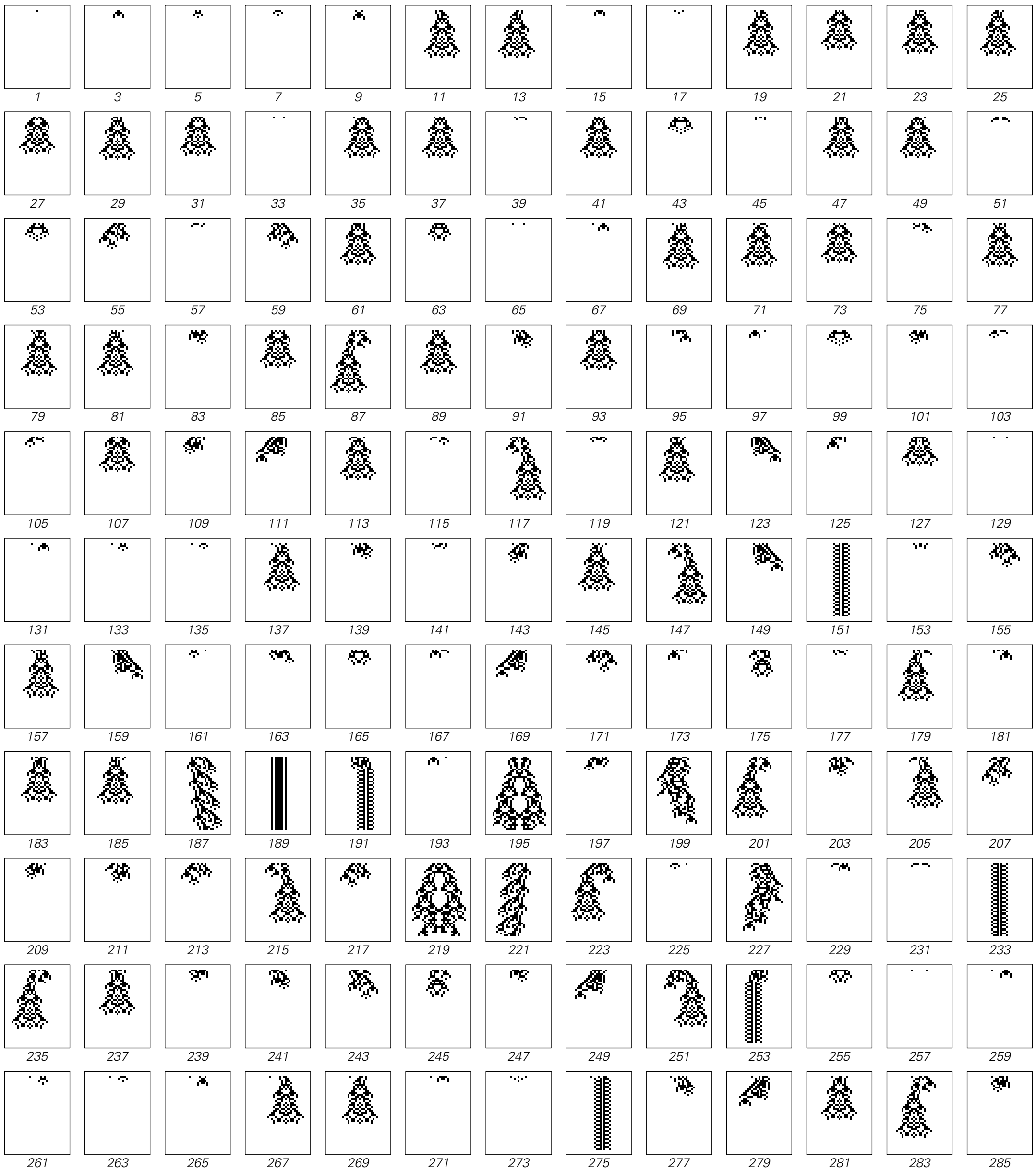


3 colors, nearest neighbors, code 357



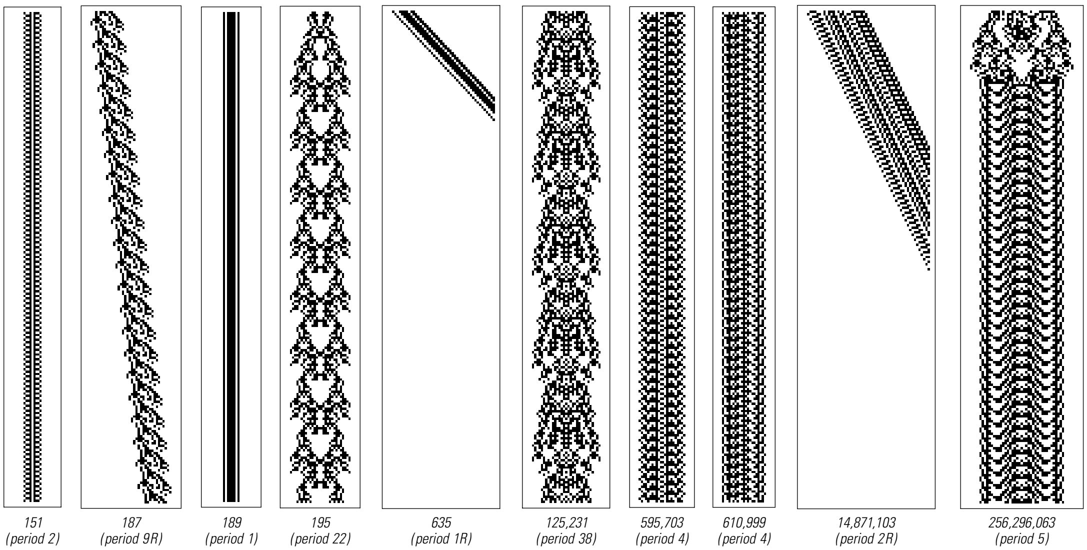
3 colors, nearest neighbors, code 1329

Three typical examples of class 4 cellular automata. In each case various kinds of persistent structures are seen.



The behavior of the code 20 cellular automaton from the top of the facing page for all initial conditions with black cells in a region of size less than nine. In most cases the patterns produced simply die out. But with some initial conditions, persistent structures are formed. Each initial condition is assigned a number whose base 2 digit sequence gives the configuration of black and white cells in that initial condition. Note that initial conditions 195 and 219 both yield the period 22 persistent structure shown on the next page.

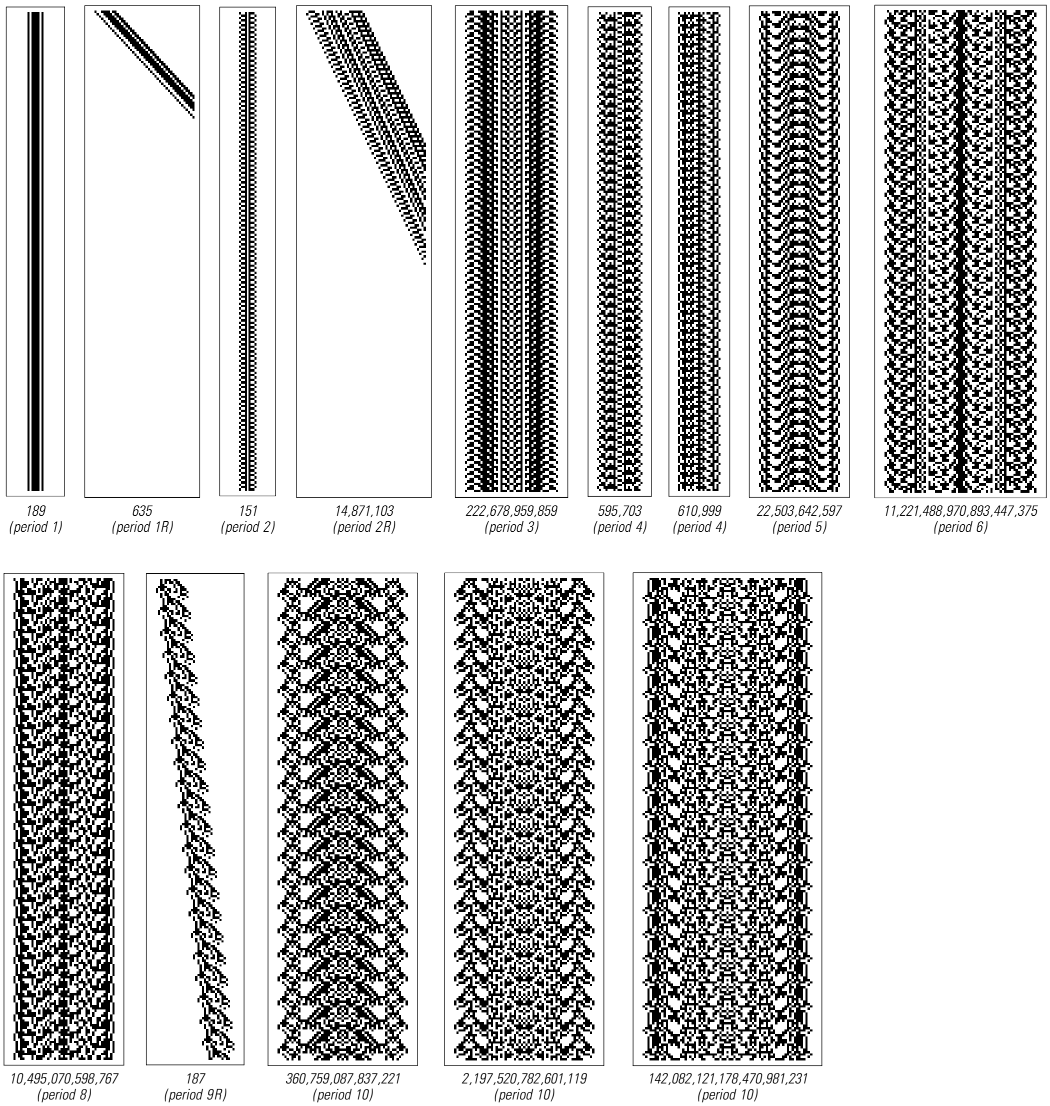
So just what set of structures does the code 20 cellular automaton ultimately support? There seems to be no easy way to tell, but the picture below shows all the structures that I found by explicitly looking at evolution from the first twenty-five billion possible initial conditions.



Persistent structures found by testing the first twenty-five billion possible initial conditions for the code 20 cellular automaton shown on the previous page. Note that reflected versions of the structures shown are also possible. The base 2 digit sequences of the numbers given correspond to the initial conditions in each case, as on the previous page.

Are other structures possible? The largest structure in the picture above starts from a block that is 30 cells wide. And with the more than ten billion blocks between 30 and 34 cells wide, no new structures at all appear. Yet in fact other structures are possible. And the way to tell this is that for small repetition periods there is a systematic procedure that allows one to find absolutely all structures with a given period.

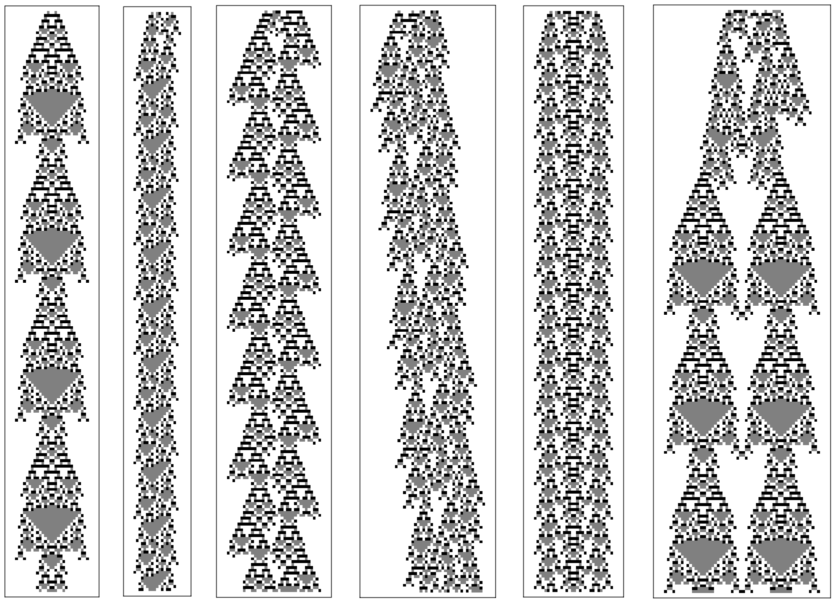
The picture on the facing page shows the results of using this procedure for repetition periods up to 15. And for all repetition periods up to 10—with the exception of 7—at least one fixed or moving structure ultimately turns out to exist. Often, however, the smallest structures for a given period are quite large, so that for example in the case of period 6 the smallest possible structure is 64 cells wide.



All the persistent structures with repetition periods up to 15 steps in the code 20 cellular automaton. The structures shown were found by a systematic method similar to the one used to find all sequences that satisfy the constraints on page 268.

So what about other class 4 cellular automata—like the ones I showed at the beginning of this section? Do they also end up having complicated sets of possible persistent structures?

The picture below shows the structures one finds by explicitly testing the first two billion possible initial conditions for the code 357 cellular automaton from page 282.



28 (period 48) 7,795 (period 19) 1,706,588 (period 26) 4,803,890 (period 41R) 154,596,664 (period 12) 514,454,827 (period 48)

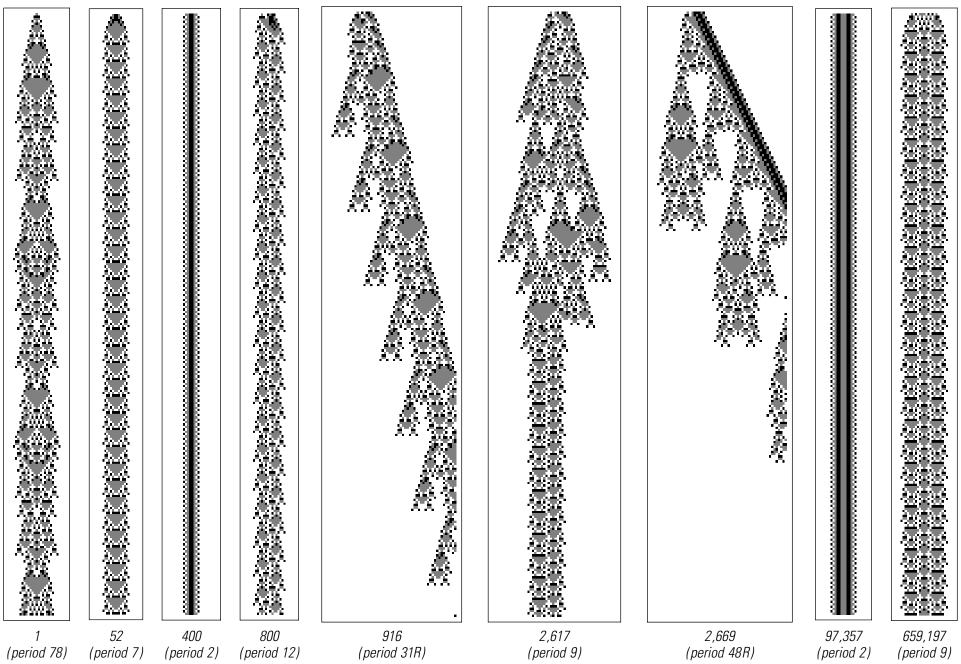
Persistent structures in the code 357 cellular automaton from page 282 obtained by testing the first two billion possible initial conditions. This cellular automaton allows three possible colors for each cell; the initial conditions thus correspond to the base 3 digits of the numbers given. No persistent structures of any size exist in this cellular automaton with repetition periods of less than 5 steps.

Already with initial condition number 28 a fairly complicated structure with repetition period 48 is seen. But with all the first million initial conditions, only one other structure is produced, and this structure is again one that does not move.

So are moving structures in fact possible in the code 357 cellular automaton? My experience with many different rules is that whenever sufficiently complicated persistent structures occur, structures that move can eventually be found. And indeed with code 357, initial condition 4,803,890 yields just such a structure.

So if moving structures are inevitable in class 4 systems, what other fundamentally different kinds of structures might one see if one were to look at sufficiently many large initial conditions?

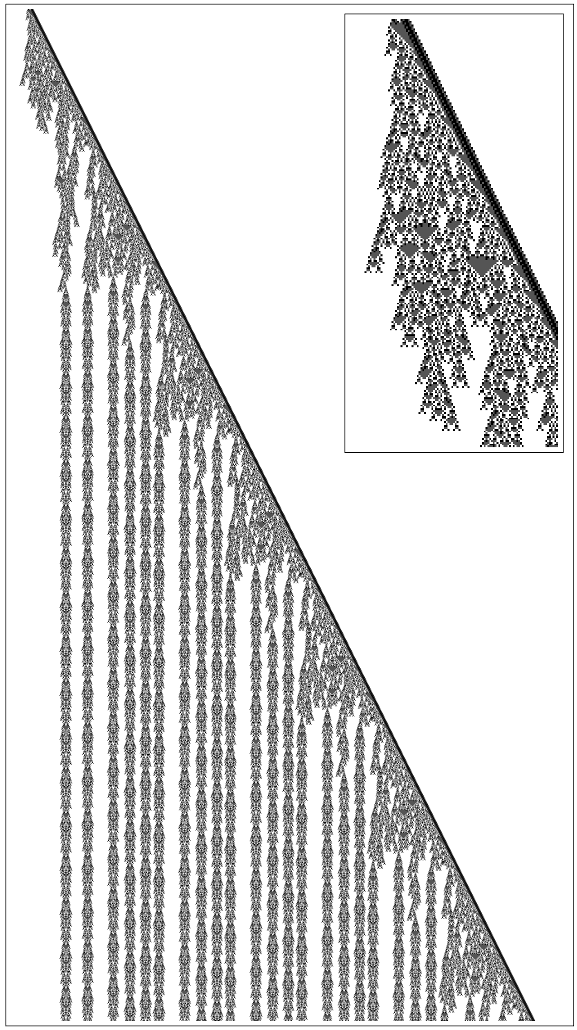
The picture below shows the first few persistent structures found in the code 1329 cellular automaton from the bottom of page 282. The smallest structures are stationary, but at initial condition 916 a structure is found that moves—all much the same as in the two other class 4 cellular automata that we have just discussed.



Persistent structures in the code 1329 cellular automaton shown on page 282.

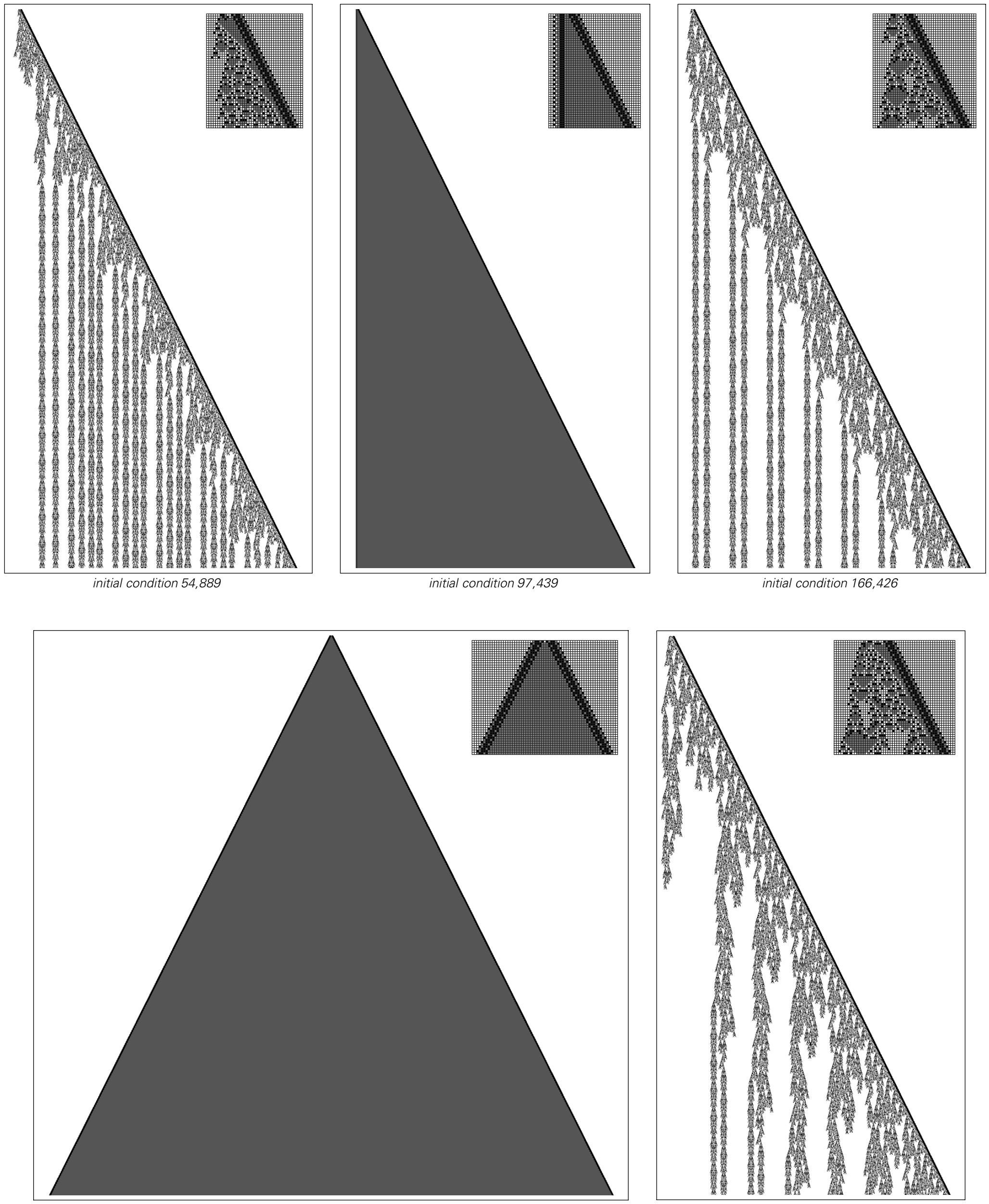
But when initial condition 54,889 is reached, one suddenly sees the rather different kind of structure shown on the next page. The right-hand part of this structure just repeats with a period of 256 steps, but as this part moves, it leaves behind a sequence of other persistent structures. And the result is that the whole structure continues to grow forever, adding progressively more and more cells.

Unbounded growth in code 1329. The initial condition contains a block of 10 cells. The right-hand side of the pattern repeats every 256 steps, and as it moves it leaves behind an infinite sequence of persistent structures.



initial condition number 54,889

Yet looking at the picture above, one might suppose that when unlimited growth occurs, the pattern produced must be fairly complicated. But once again code 1329 has a surprise in store. For the facing page shows that when one reaches initial condition 97,439 there is again unlimited growth—but now the pattern that is produced is very simple. And in fact if one were just to see this pattern, one would probably assume that it came from a rule whose typical behavior is vastly simpler than code 1329.



initial condition 54,889 *initial condition 97,439* *initial condition 166,426*

initial condition 115,396 *initial condition 2,069,116*

Further examples of unbounded growth in code 1329. Most of the patterns produced are complex—but some are simple.

A typical example of the behavior of the rule 110 cellular automaton with random initial conditions. The background pattern consists of blocks of 14 cells that repeat every 7 steps.

Indeed, it is a general feature of class 4 cellular automata that with appropriate initial conditions they can mimic the behavior of all sorts of other systems. And when we discuss computation and the notion of universality in Chapter 11 we will see the fundamental reason this ends up being so. But for now the main point is just how diverse and complex the behavior of class 4 cellular automata can be—even when their underlying rules are very simple.

And perhaps the most striking example is the rule 110 cellular automaton that we first saw on page 32. Its rule is extremely simple—involving just nearest neighbors and two colors of cells. But its overall behavior is as complex as any system we have seen.

The facing page shows a typical example with random initial conditions. And one immediate slight difference from other class 4 rules that we have discussed is that structures in rule 110 do not exist on a blank background: instead, they appear as disruptions in a regular repetitive pattern that consists of blocks of 14 cells repeating every 7 steps.

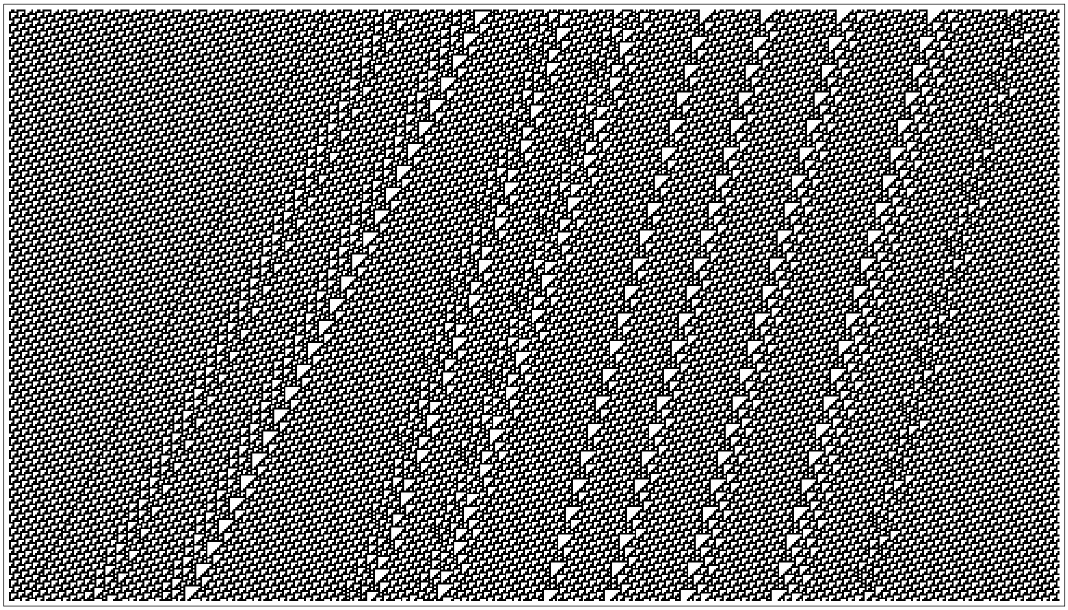
The next page shows the kinds of persistent structures that can be generated in rule 110 from blocks less than 40 cells wide. And just like in other class 4 rules, there are stationary structures and moving structures—as well as structures that can be extended by repeating blocks they contain.

So are there also structures in rule 110 that exhibit unbounded growth? It is certainly not easy to find them. But if one looks at blocks of width 41, then such structures do eventually show up, as the picture on page 293 demonstrates.

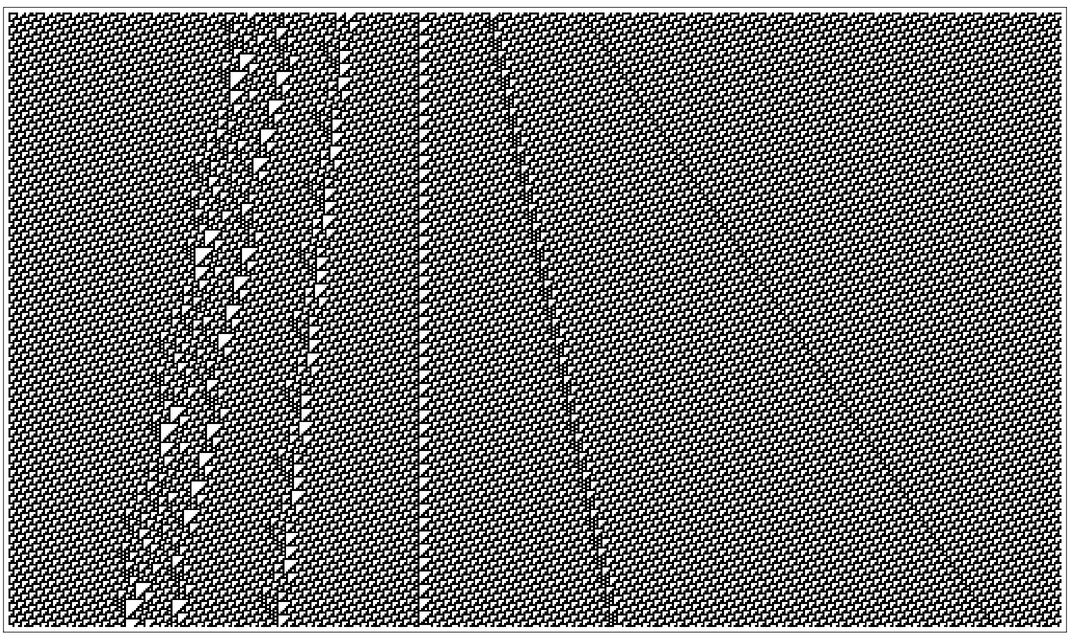
So how do the various structures in rule 110 interact? The answer, as pages 294–296 demonstrate, can be very complicated.

In some cases, one structure essentially just passes through another with a slight delay. But often a collision between two structures produces a whole cascade of new structures. Sometimes the outcome of a collision is evident after a few steps. But quite often it takes a very large number of steps before one can tell for sure what is going to happen.

So even though the individual structures in class 4 systems like rule 110 may behave in fairly repetitive ways, interactions between these structures can lead to behavior of immense complexity.

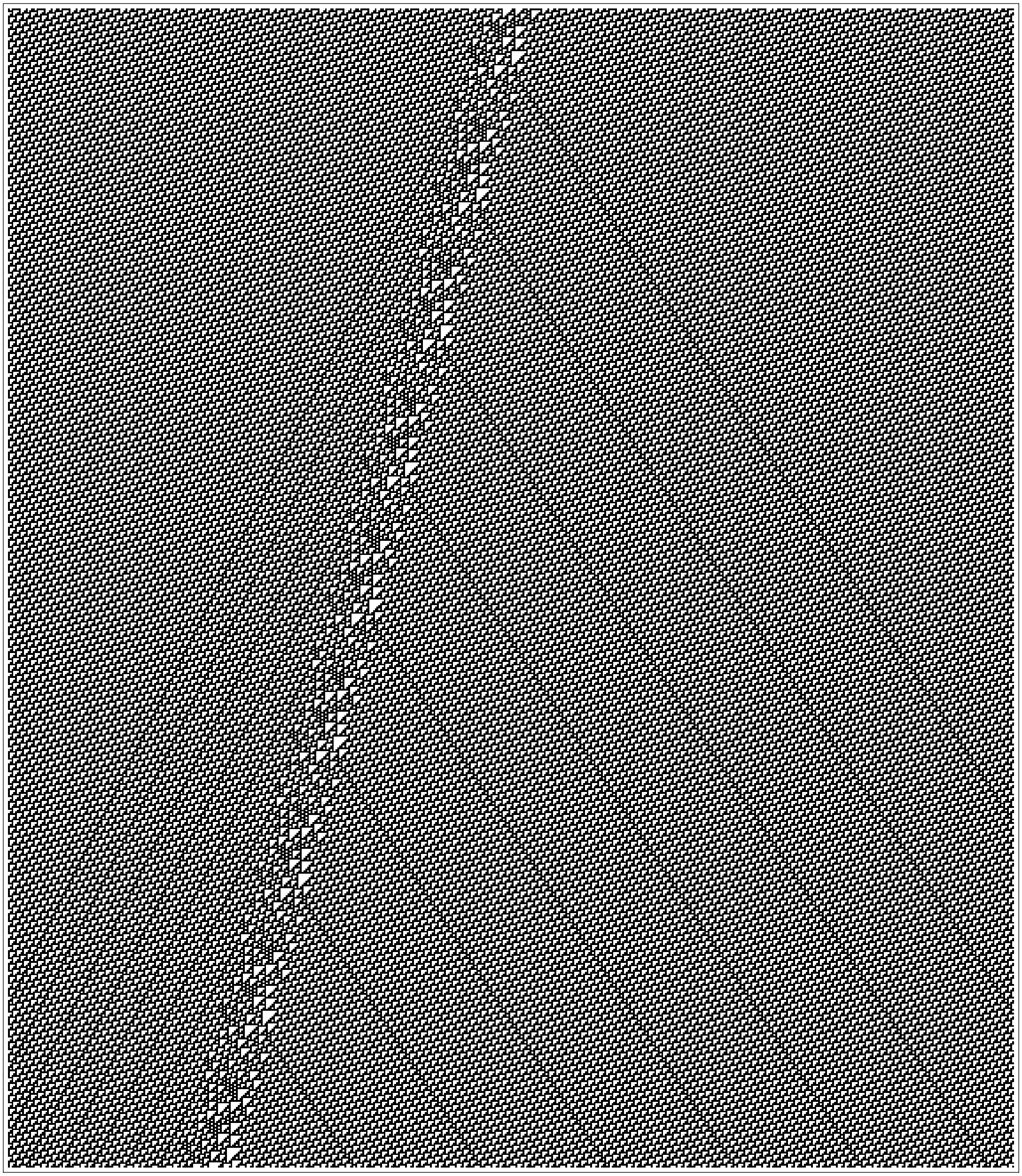


(a) (b) (c) (d) (e) (f) (g) (h) (i) (j)

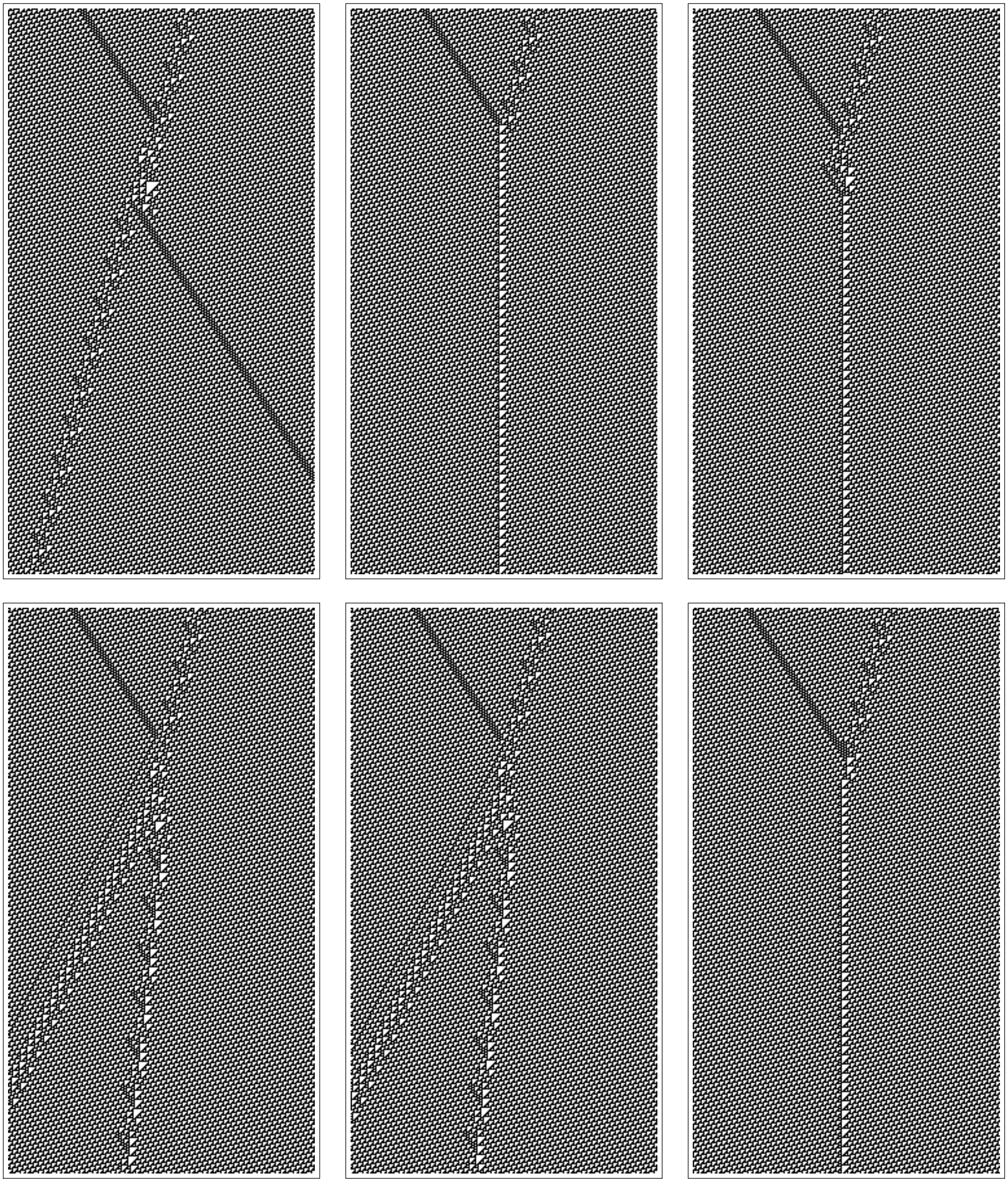


(k) (l) (m) (n) (o)

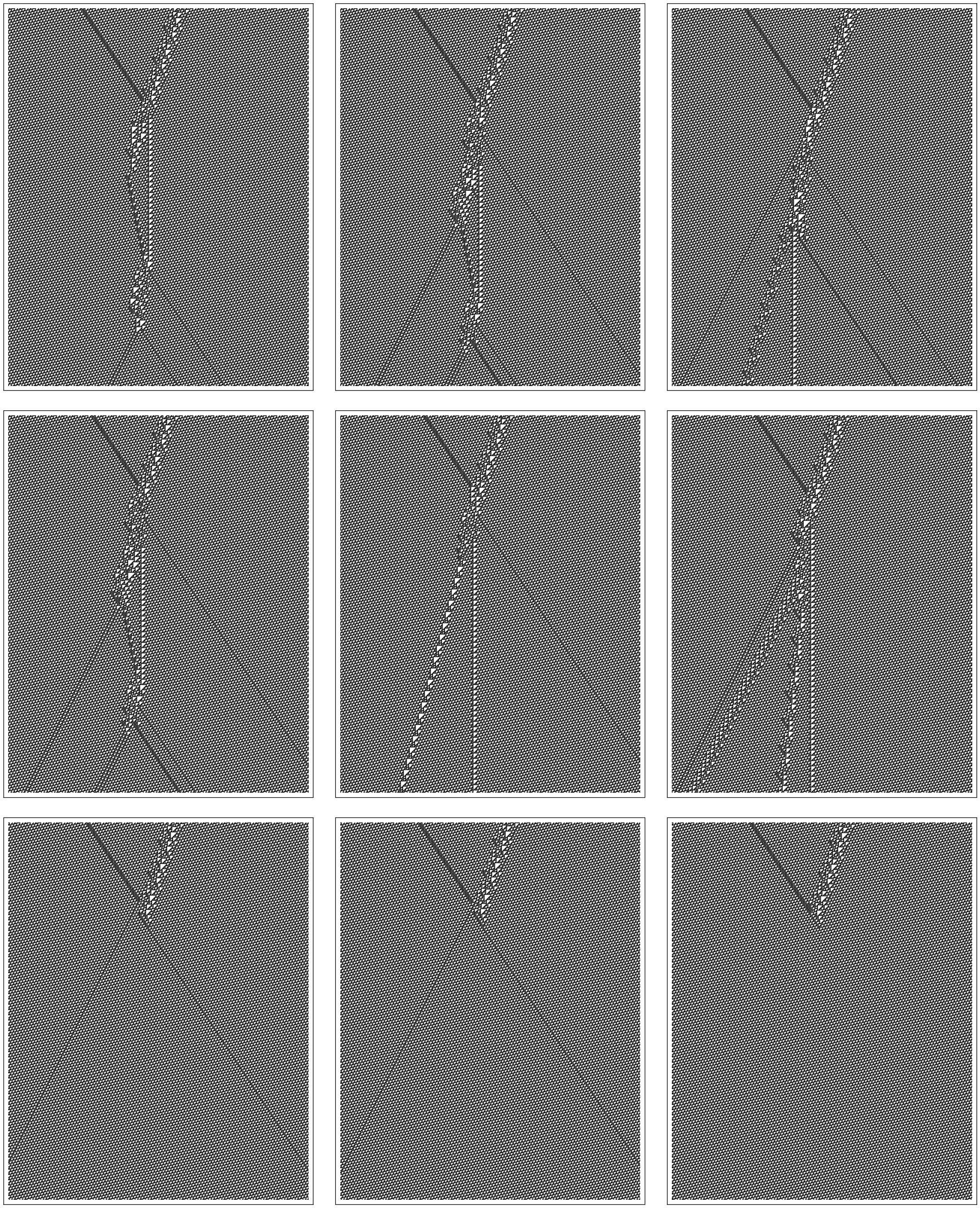
Persistent structures found in rule 110. Extended versions exist of all but structures (a) and (j). Structures (m) and (n) also exist in alternate forms shifted with respect to the background.



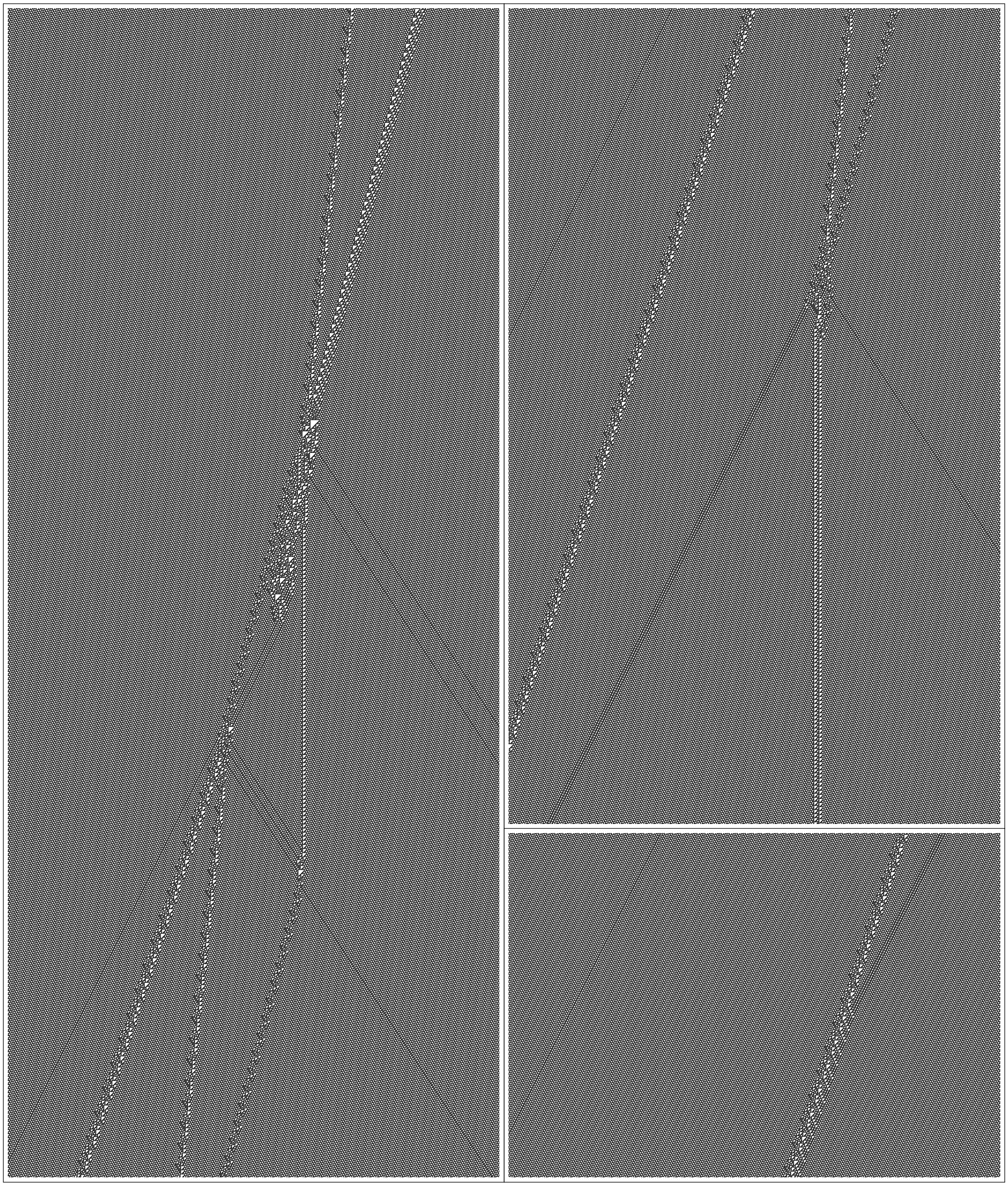
An example of unbounded growth in rule 110. The initial condition consists of a block of length 41 inserted between blocks of the background. New structures on both left and right are produced every 77 steps; the central structure moves 20 cells to the left during each cycle so that the structures on the left are separated by 37 steps while those on the right are separated by 107 steps.



Collisions between persistent structures (o) and (j) from page 292. (The first structure is actually an extended form containing four copies of structure (o) from page 292.) Each successive picture shows what happens when the original structures are started progressively further apart.



Collisions between structures (e) and (o) from page 292.



A collision between structures (l) and (i) from page 292. It takes more than 4000 steps for the final outcome involving 8 separate structures to become clear. The height of the picture corresponds to 2000 steps, and the third picture ends at step 4300.

NOTES FOR CHAPTER 6

Starting from Randomness

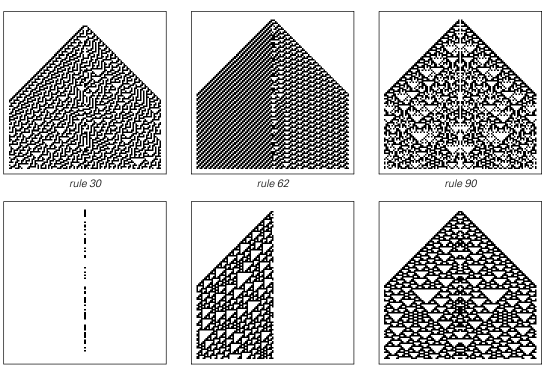
The Emergence of Order

■ **Page 226 · Properties of patterns.** For a random initial condition, the average density of black cells is exactly 1/2. For rule 126, the density after many steps is still 1/2. For rule 22, it is approximately 0.35095. For rule 30 and rule 150 it is exactly 1/2, while for rule 182 it is 3/4. And insofar as rule 110 converges to a definite density, the density is 4/7. (See page 953 for a method of estimating these densities.)

Even after many steps, individual lines in the patterns produced by rules 30 and 150 remain in general completely random. But in rule 126, black cells always tend to appear in pairs, while in rule 182, every white cell tends to be surrounded by black ones. And in rule 22, there are more complicated conditions involving blocks of 4 cells.

The density of triangles of size n goes roughly like 2^{-n} for rules 126, 30 (see also page 871), 150 and 182 and roughly like 1.3^{-n} for rule 22.

In the algebraic representation discussed on page 869, rule 22 is $Mod[p + q + r + p\,q\,r, 2]$, rule 126 is $Mod[(p + q)(q + r) + (p + r), 2]$, rule 150 is $Mod[p + q + r, 2]$ and rule 182 is $Mod[p\,r\,(1 + q) + (p + q + r), 2]$.



rule 30 rule 62 rule 90

rule 108 rule 110 rule 126

■ **Continual injection of randomness.** In the main text we discuss what happens when one starts from random initial conditions and then evolves according to a definite cellular automaton rule. As an alternative one can consider starting with very simple initial conditions, such as all cells white, and then at each step randomly changing the color of the center cell. Some examples of what happens are shown at the bottom of the previous column. The results are usually very similar to those obtained with random initial conditions.

■ **History.** The fact that despite initial randomness processes like friction can make systems settle down into definite configurations has been the basis for all sorts of engineering throughout history. The rise of statistical mechanics in the late 1800s emphasized the idea of entropy increase and the fundamental tendency for systems to become progressively more disordered as they evolve to thermodynamic equilibrium. Theories were nevertheless developed for a few cases of spontaneous pattern formation—notably in convection, cirrus clouds and ocean waves. When the study of feedback and stability became popular in the 1940s, there were many results about how specific simple fixed or repetitive behaviors in time could emerge despite random input. In the 1950s it was suggested that reaction-diffusion processes might be responsible for spontaneous pattern formation in biology (see page 1012)—and starting in the 1970s such processes were discussed as prime examples of the phenomenon of self-organization. But in their usual form, they yield essentially only rather simple repetitive patterns. Ever since around 1900 it tended to be assumed that any fundamental theory of systems with many components must be based on statistical mechanics. But almost all work in the field of statistical mechanics concentrated on systems in or very near thermal equilibrium—in which in a sense there is almost complete disorder. In the 1970s there began to be more discussion of phenomena far from equilibrium, although typically it got no further than to consider how external forces could lead to reaction-diffusion-like phenomena. My own work on cellular automata in 1981 emerged in part from thinking about self-gravitating systems (see page 880) where it seemed conceivable that there might be very basic rules quite different from those usually studied in statistical mechanics. And when I first generated pictures of the behavior of arbitrary cellular automaton rules, what struck me most was the order that emerged even from random initial conditions. But while it was immediately clear that most cellular automata do not have the kind of reversible underlying rules assumed in traditional statistical mechanics, it still seemed initially very surprising that their overall behavior could be so elaborate—and so far from the complete orderlessness one might expect on the basis of traditional ideas of entropy maximization.

Four Classes of Behavior

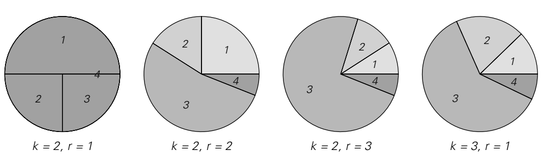
■ **Different runs.** The qualitative behavior seen with a given cellular automaton rule will normally look exactly the same for essentially all different large random initial conditions—just as it does for different parts of a single initial condition. And as discussed on page 597 any obvious differences could in effect be thought of as revealing deviations from randomness in the initial conditions.

■ **Page 232 · Elementary rules.** The examples shown have rule numbers *n* for which *IntegerDigits[n, 2, 8]* matches *{_, i_, _, j_, i_, _, j_, 0}*.

■ **Page 235 · States of matter.** As suggested by pages 944 and 1193, working out whether a particular substance at a particular temperature will be a solid, liquid or gas may in fact be computationally comparable in difficulty to working out what class of behavior a particular cellular automaton will exhibit.

■ **Page 235 · Class 4 rules.** Other examples of class 4 totalistic rules with $k = 3$ colors include 357 (page 282), 438, 600, 792, 924, 1038, 1041, 1086, 1329 (page 282), 1572, 1599 (see page 70), 1635 (see page 67), 1662, 1815 (page 236), 2007 (page 237) and 2049 (see page 68).

■ **Frequencies of classes.** The pie charts below show results for 1D totalistic cellular automata with k colors and range r. Class 3 tends to become more common as the number of elements in the rule increases because as soon as any of these elements yield class 3 behavior, that behavior dominates the system.



■ **History.** I discovered the classification scheme for cellular automata described here late in 1983, and announced it in January 1984. Much work has been done by me and others on ways to make the classification scheme precise. The notion that class 4 can be viewed as intermediate between class 2 and class 3 was studied particularly by Christopher Langton, Wentian Li and Norman Packard in 1986 for ordinary cellular automata, by Hyman Hartman in 1985 for probabilistic cellular automata and by Hugues Chaté and Paul Manneville in 1990 for continuous cellular automata.

■ **Subclasses within class 4.** Different class 4 systems can show localized structures with strikingly similar forms, and this may allow subclasses within class 4 to be identified. In addition, class 4 systems show varying levels of activity, and it is possible that there may be discrete transitions—perhaps analogous to percolation—that can be used to define boundaries between subclasses.

■ **Page 240 · Undecidability.** Almost any definite procedure for determining the class of a particular rule will have the feature that in borderline cases it can take arbitrarily long, often formally showing undecidability, as discussed on page 1138. (An example would be a test for class 1 based on checking that no initial pattern of any size can survive. Including probabilities can help, but there are still always borderline cases and potential undecidability.)

■ **Page 244 · Continuous cellular automata.** In ordinary cellular automata, going from one rule to the next in a sequence involves some discrete change. But in continuous cellular automata, the parameters of the rule can be varied smoothly. Nevertheless, it still turns out that there are discrete transitions in the overall behavior that is produced. In fact, there is often a complicated set of transitions that depends more on the digit sequence of the parameter than its size. And between these transitions there are usually ranges of parameter values that yield definite class 4 behavior. (Compare page 922.)

■ **Nearby cellular automaton rules.** In a range r cellular automaton the new color of a particular cell depends only on cells at most a distance r away. One can make an equivalent cellular automaton of larger range by having a rule in which cells at distance more than r have no effect. One can then define nearby cellular automata to be those where the differences in the rule involve only cells close to the edge of the range. With larger and larger ranges one can then construct closer approximations to continuous sequences of cellular automata.

■ **2D class 4 cellular automata.** No 5- or 9-neighbor totalistic rules nor 5-neighbor outer totalistic ones appear to yield class 4 behavior with a white background. But among 9-neighbor outer totalistic rules there are examples with codes 224 (Game of Life), 226, 4320 (sometimes called HighLife), 5344, 6248, 6752, 6754 and 8416, etc. It turns out that the simplest moving structures are the same in codes 224, 226 and 4320.

■ **Page 249 · Game of Life.** Invented by John Conway around 1970 (see page 877), the Life 2D cellular automaton has been much studied in recreational computing, and as described on page 964 many localized structures in it have been identified. Each step in its evolution can be implemented using

```
LifeStep[a_List] :=
  MapThread[If[#1 == 1 && #2 == 4 || #2 == 3, 1, 0] &,
    {a, Sum[RotateLeft[a, {i, j}], {i, -1, 1}, {j, -1, 1}]}, 2]
```

A more efficient implementation can be obtained by operating not on a complete array of black and white cells but rather just on a list of positions of black cells. With this setup, each step then corresponds to

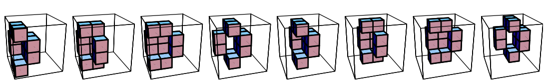
```
LifeStep[list_] :=
  With[{p = Flatten[Array[List, {3, 3}, -1], 1]},
    With[{u = Split[Sort[Flatten[Outer[Plus, list, p, 1], 1]]]},
      Union[Cases[u, {x_, _, _} → x],
        Intersection[Cases[u, {x_, _, _, _} → x], list]]]]
```

(A still more efficient implementation is based on finding runs of length 3 and 4 in *Sort[u]*.)

■ **3D class 4 rules.** With a cubic lattice of the type shown on page 183, and with updating rules of the form

```
LifeStep3D[{p_, q_, r_}, a_List] := MapThread[If[
 #1 == 1 && p ≤ #2 ≤ q || #2 == r, 1, 0] &, {a, Sum[RotateLeft[
  a, {i, j, k}], {i, -1, 1}, {j, -1, 1}, {k, -1, 1}] - a}, 3]
```

Carter Bays discovered between 1986 and 1990 the three examples *{5, 7, 6}*, *{4, 5, 5}*, and *{5, 6, 5}*. The pictures below show successive steps in the evolution of a moving structure in the second of these rules.



■ **Random initial conditions in other systems.** Whenever the initial conditions for a system can involve an infinite sequence of elements these elements can potentially be chosen at random. In systems like mobile automata and Turing machines the colors of initial cells can be random, but the active cell must start at a definite location, and depending on the behavior only a limited region of initial cells near this location may ever be sampled. Ordinary substitution systems can operate on infinite sequences of elements chosen at random. Sequential substitution systems, however, rely on scanning limited sequences of elements, and so cannot readily be given infinite random initial conditions. The same is true of ordinary and cyclic tag systems. Systems based on continuous numbers involve infinite sequences of digits which can readily be chosen at random (see page 154). But systems based on integers (including register machines) always deal with finite sequences of digits, for which there is no unique definition of randomness. (See however the discussion of number representations on page 1070.) Random networks (see pages 963 and 1038) can be used to provide random initial conditions for network systems. Multiway systems cannot meaningfully be given infinite random initial conditions since these would typically lead to an infinite number of possible states. Systems based on constraints do not have initial conditions. (See also page 920.)

Sensitivity to Initial Conditions

■ **Page 251 · Properties.** In rule 126, the outer edges of the region of change always expand by exactly one cell per step. The same is true of the right-hand edge in rule 30—though the left-hand edge in this case expands only about 0.2428 cells on average per step. In rule 22, both edges expand about 0.7660 cells on average per step.

The motion of the right-hand edge in rule 30 can be understood by noting that with this rule the color of a particular cell will always change if the color of the cell to its left is changed on the previous step (see page 601). Nothing as simple is true for the left-hand edge, and indeed this seems to execute an essentially random walk—with an average motion of about 0.2428 cells per step. Note that in the approximation that the colors of all cells in the pattern are assumed completely independent and random there should be motion by 0.25 cells per step. Curiously, as discussed on page 871, the region of non-repetitive behavior in evolution from a single black cell according to rule 30 seems to grow at a similar but not identical rate of about 0.252 cells per step. (For rule 45, the left-hand edge of the difference pattern moves about 0.1724 cells per step; for rule 54 both edges move about 0.553 cells per step.)

■ **Difference patterns.** The maximum rate at which a region of change can grow is determined by the range of the underlying cellular automaton rule. If the rule involves up to r nearest neighbors, then at each step a change in the color of a given cell can affect cells up to r away—so that the edge of the region of change can move by r cells.

For most class 3 rules, once one is inside the region of change, the colors of cells usually become essentially uncorrelated.

However, for additive rules the pattern of differences is just exactly the pattern that would be obtained by evolution from an initial condition consisting only of the changes made. In general the pattern of probabilities for changes can be thought of as being somewhat like a Green's function in mathematical physics—though the nonadditivity of most cellular automata makes this analogy less useful. (Note that the pattern of differences between two initial conditions in a rule with k possible colors can always be reproduced by looking at the evolution from a single initial condition of a suitable rule with $2k$ colors.) In 2D class 3 cellular automata, the region of change usually ends up having a roughly circular shape—a result presumably related to the Central Limit Theorem (see page 976).

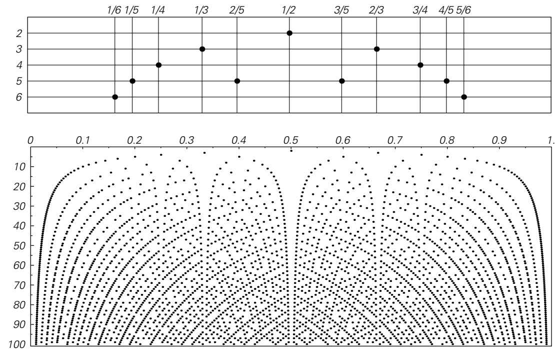
For any additive or partially additive class 3 cellular automaton (such as rule 90 or rule 30) any change in initial conditions will always lead to expanding differences. But in other rules it sometimes may not. And thus, for example, in rule 22, changing the color of a single cell has no effect after even one step if the cell has a ■■ block on either side. But while there are a few other initial conditions for which differences can die out after several steps most forms of averaging will say that the majority of initial conditions lead to growing patterns of differences.

■ **Lyapunov exponents.** If one thinks of cells to the right of a point in a 1D cellular automaton as being like digits in a real number, then linear growth in the region of differences associated with a change further to the right is analogous to the exponentially sensitive dependence on initial conditions shown on page 155. The speed at which the region of differences expands in the cellular automaton can thus be thought of as giving a Lyapunov exponent (see page 921) that characterizes instability in the system.

Systems of Limited Size and Class 2 Behavior

■ **Page 255 · Cyclic addition.** After t steps, the dot will be at position $Mod[m\,t, n]$ where n is the total number of positions, and m is the number of positions moved at each step. The repetition period is given by $n/GCD[m, n]$. The picture on page 613 shows the values of m and n for which this is equal to n.

An alternative interpretation of the system discussed here involves arranging the possible positions in a circle, so that at each step the dot goes a fraction m/n of the way around the circle. The repetition period is maximal when m/n is a fraction in lowest terms. The picture below shows the repetition periods as a function of the numerical size of the quantity m/n.



■ **Page 257 · Cyclic multiplication.** With multiplication by k at each step the dot will be at position $Mod[k^t, n]$ after t steps. If k and n have no factors in common, there will be a t for which $Mod[k^t, n] == 1$, so that the dot returns to position 1. The smallest such t is given by $MultiplicativeOrder[k, n]$, which always divides $EulerPhi[n]$ (see page 1093), and has a value between $Log[k, n]$ and $n-1$, with the upper limit being attained only if n is prime. (This value is related to the repetition period for the digit sequence of $1/n$ in base k, as discussed on page 912). When $GCD[k, n] == 1$ the dot can never visit position 0. But if $n == k^s$, the dot reaches 0 after s steps, and then stays there. In general, the dot will visit position $m = k\,^\wedge IntegerExponent[n, k]$ every $MultiplicativeOrder[k, n/m]$ steps.

■ **Page 260 · Maximum periods.** A cellular automaton with n cells and k colors has k^n possible states, but if the system has cyclic boundary conditions, then the maximum repetition period is smaller than k^n. The reason is that different states of the cellular automaton have different symmetry properties, and thus cannot be on the same cycle. In particular, if a state of a cellular automaton has a certain spatial period, given by the minimum positive m for which $RotateLeft[list, m] == list$, then this state can never evolve to one with a larger spatial period. The number of states with spatial period m is given by

```
s[m_, k_] :=
  k^m - Apply[Plus, Map[s[#, k] &, Drop[Divisors[m], -1]]]
```

or equivalently

```
s[m_, k_] := Apply[Plus,
    (MoebiusMu[m/#] k^# &)[Divisors[m]]]
```

In a cellular automaton with a total of n cells, the maximum possible repetition period is thus $s[n, k]$. For $k = 2$, the maximum periods for n up to 10 are: $\{2, 2, 6, 12, 30, 54, 126, 240, 504, 990\}$. In all cases, $s[n, k]$ is divisible by n. For prime n, $s[n, k]$ is $k^n - k$. For large n, $s[n, k]$ oscillates between about $k^n - k^{n/2}$ and $k^n - k$. (See page 963.)

■ **Additive cellular automata.** In the case of additive rules such as rule 90 and rule 60, a mathematical analysis of the repetition periods can be given (as done by Olivier Martin, Andrew Odlyzko and me in 1983). One starts by converting the list of cell colors at each step to a polynomial *FromDigits[list, x]*. Then for the case of rule 60 with n cells and cyclic boundary conditions, the state obtained after t steps is given by

$$PolynomialMod[(1+x)^t\, z, \{x^n - 1, 2\}]$$

where z is the polynomial representing the initial state, and $z = 1$ for a single black cell in the first position. The state $z = 1$ evolves after one step to the state $z = 1 + x$, and for odd n this latter state always eventually appears again. Using the result that $1 + x^{2^m} == (1+x)^{2^m}$ modulo 2 for any m, one then finds that the repetition period always divides the quantity $p[n] = 2\,\hat{}\,MultiplicativeOrder[2, n] - 1$, which in turn is at most $2^{n-1} - 1$. The actual periods are often smaller than $p[n]$, with the following ratios occurring:

n	11	13	19	25	27	29	37	41	43	53
ratio	3	5	27	41	19	565	21255	25	3	1266205

There appears to be no case for $n > 5$ where the period achieves the absolute maximum $2^{n-1} - 1$.

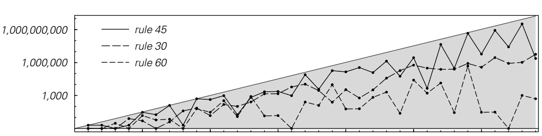
In the case of rule 90 a similar analysis can be given, with the $1 + x$ used at each step replaced by $1/x + x$. And now the repetition period for odd n divides

$$q[n] = 2\,\hat{}\,MultiplicativeOrder[2, n, \{1, -1\}] - 1$$

The exponent here always lies between $Log[k, n]$ and $(n-1)/2$, with the upper bound being attained only if n is prime. Unlike for the case of rule 60, the period is usually equal to $q[n]$ (and is assumed so for the picture on page 260), with the first exception occurring at $n = 37$.

■ **Rules 30 and 45.** Maximum periods are often achieved with initial conditions consisting of a single black cell. Particularly for rule 30, however, there are quite a few exceptions. For $n = 13$, for example, the maximum period is 832 but the period with a single black cell is 260. For rule 45, the maximum possible period discussed above is achieved for $n = 9$, but does not appear to be achieved for any larger n. (See page 962.)

■ **Comparison of rules.** Rules 45, 30 and 60, together with their conjugates and reflections, yield the longest repetition periods of all elementary rules (see page 1087). The picture below compares their periods as a function of n.

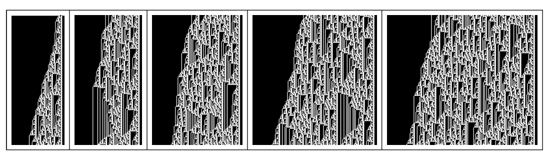


■ **Implementing boundary conditions.** In the bitwise representation discussed on page 865, 0's outside of a width n can be implemented by applying $BitAnd[a, 2^n - 1]$ at each step. Cyclic boundary conditions can be implemented efficiently in assembler on computers that support cyclic shift instructions.

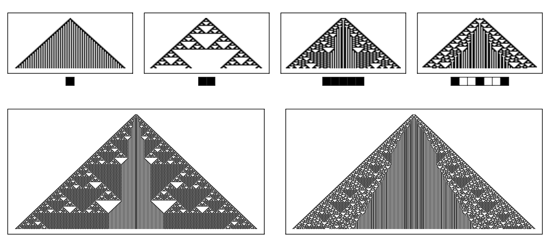
Randomness in Class 3 Systems

■ **Page 263 · Rule 22.** Randomness is obtained with initial conditions consisting of two black squares $4m$ positions apart for any $m \geq 2$. The base 2 digit sequences for 19, 25, 37, 39, 41, 45, 47, 51, 57, 61, … also give initial conditions that yield randomness. Despite its overall randomness there are some regularities in the pattern shown at the bottom of the page. The overall density of black cells is not 1/2 but is instead approximately 0.35, just as for random initial conditions. And if one looks at the center cell in the pattern one finds that it is never black on two successive steps, and the probability for white to follow white is about twice the probability for black to follow white. There is also a region of repetitive behavior on each side of the pattern; the random part in the middle expands at about 0.766 cells per step—the same speed that we found on page 949 that changes spread in this rule.

■ **Rule 225.** With initial conditions consisting of a single black cell, this class 3 rule yields a regular nested pattern, as shown on page 58. But with the initial condition ■■□■, it yields the much more complicated pattern shown below. With a background consisting of repetitions of the block ■□, insertion of a single initial white cell yields a largely random pattern that expands by one cell per step. Rule 225 can be expressed as $\neg\, p \veebar (q \vee r)$.



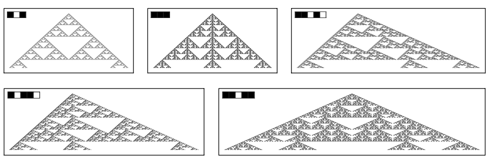
■ **Rule 94.** With appropriate initial conditions this class 2 rule can yield both nested and random behavior, as shown below.



■ **Rule 218.** If pairs of adjacent black cells appear anywhere in its initial conditions this class 2 rule gives uniform black, but if none do it gives a rule 90 nested pattern.

■ **Additive rules.** Of the 256 elementary cellular automata 8 are additive: {0, 60, 90, 102, 150, 170, 204, 240}. All of these are either trivial or essentially equivalent to rules 90 or 150.

Of all $k^{k^{2r+1}}$ rules with k colors and range r it turns out that there are always exactly k^{2r+1} additive ones—each obtained by taking the cells in the neighborhood and adding them modulo k with weights between 0 and $k-1$. As discussed on page 955, any rule based on addition modulo k must yield a nested pattern, and it therefore follows that any rule that is additive must give a nested pattern, as in the examples below. (See also page 870.)



Note that each step in the evolution of any additive cellular automaton can be computed as

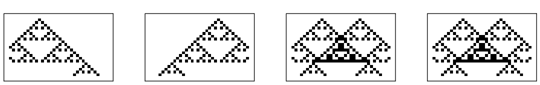
Mod[ListCorrelate[w, list, Ceiling[Length[w]/2]], k]

(See page 1087 for a discussion of partial additivity.)

■ **Page 264 · Generalized additivity.** In general what it means for a system to be additive is that some addition operation $\oplus$ can be used to combine any set of evolution histories to yield another possible evolution history. If ϕ is the rule for the system, this then requires for any states u and v the distributive property

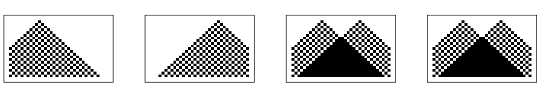
$\phi[u \oplus v] == \phi[u] \oplus \phi[v]$

(In mathematical terms this is equivalent to the statement that ϕ is conjugate to itself under the action of $\oplus$—or alternatively that ϕ defines a homomorphism with respect to the $\oplus$ operation.) In the usual case, $u \oplus v$ is just Mod[u + v, k], yielding say for rule 90 the results below.



$\phi[u]$ $\phi[v]$ $\phi[u \oplus v]$ $\phi[u] \oplus \phi[v]$

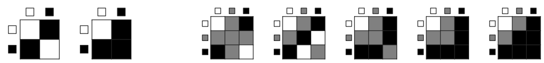
But it turns out that some elementary rules show additivity with respect to other addition operations. An example as shown below is rule 250 with $u \oplus v$ taken as Max[u, v] (Or).



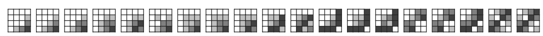
$\phi[u]$ $\phi[v]$ $\phi[u \oplus v]$ $\phi[u] \oplus \phi[v]$

If a system is additive it means that one can work out how the system will behave from any initial condition just by combining the patterns ("Green's functions") obtained from certain basic initial conditions—say ones containing a single black cell. To get all the familiar properties of additivity one needs an addition operation that is associative (Flat) and commutative (Orderless), and has an identity element (white or 0 in the cases above)—so that it defines a commutative monoid. (Usually it is also convenient to be able to get all possible elements by combining a small number of basic generator elements.)

The inequivalent commutative monoids with up to $k = 4$ colors are (in total there are 1, 2, 5, 19, 78, 421, 2637, … such objects):



6 14 4017 8229 13008 19569 19650



For $k = 2$, $r = 1$ the number of rules additive with respect to these is respectively: {8, 9}; for $k = 2$, $r = 2$: {32, 33}; for $k = 3$, $r = 1$: {28, 27, 35, 244, 28}; for $k = 4$, $r = 1$:

{1001, 65, 540, 577, 126, 4225, 540, 9065,
757, 408, 65, 133, 862, 224, 72, 72, 91, 4096, 64}

It turns out to be possible to show that any rules ϕ additive with respect to some addition operation $\oplus$ must work by applying that operation to values associated with cells in their neighborhood. The values are obtained by applying to cells at each position one of the unary operations (endomorphisms) σ that satisfy $\sigma[a \oplus b] == \sigma[a] \oplus \sigma[b]$ for individual cell values a and b. (For Xor, there are 2 possible σ, while for Or there are 3.)

The basic examples are then rules of the form RotateLeft[a] ⊕ RotateRight[a]—analogs of rule 90, but with other addition operations (compare page 886). The σ can be used to give analogs of the weights that appear in the note above. And rules that involve more than two cells can be obtained by having several instances of $\oplus$—which can always be flattened. But in all cases the general results for associative rules on page 956 show that the patterns obtained must be at most nested.

If instead of an ordinary cellular automaton with a limited number of possible colors one considers a system in which every cell can have any integer value then additivity with respect to ordinary addition becomes just traditional linearity. And the only way to achieve this is to have a rule in which the new value of a cell is given by a linear form such as $a x + b y$. If the values of cells are allowed to be any real number then linear forms such as $a x + b y$ again yield additivity with respect to ordinary addition. But in general one can apply to each cell value any function σ that obeys the so-called Cauchy functional equation $\sigma[x + y] == \sigma[x] + \sigma[y]$. If $\sigma[x]$ is required to be continuous, then the only form it can have is $c x$. But if one allows σ to be discontinuous then there can be some other exotic possibilities. It is inevitable that within any rationally related set of values x one must have $\sigma[x] = c x$ with fixed c. But if one assumes the Axiom of Choice then in principle it becomes possible to construct $\sigma[x]$ which have different c for different sets of x values. (Note however that I do not believe that such σ could ever actually be constructed in any explicit way by any real computational system—or in fact by any system in our universe.)

In general $\oplus$ need not be ordinary addition, but can be any operation that defines a commutative monoid—including an infinite one. An example is ordinary numbers modulo an irrational. And indeed a cellular automaton whose rule is based on *Mod[x + y, π]* will show additivity with respect to this operation (see page 922). If $\oplus$ has an inverse, so that it defines a group, then the only continuous (Lie group) examples turn out to be combinations of ordinary addition and modular addition (the group U(1)). This assumes, however, that the underlying cellular automaton has discrete cells. But one can also imagine setting up systems whose states are continuous functions of position. ϕ then defines a mapping from one such function to another. To be analogous to cellular automata one can then require this mapping to be local, in which case if it is continuous it must be just a linear differential operator involving *Derivative[n]*—and at some level its behavior must be fairly simple. (Compare page 161.)

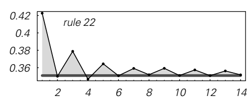
■ **Probabilistic estimates.** One way to get estimates for density and other properties of class 3 cellular automata is to make the assumption that the color of each cell at each step is completely random. And with this assumption, if the overall density of black cells at a particular step is p, then each cell at that step should independently have probability p to be black. This means that for example the probability to find a black cell followed by two white cells is $p(1-p)^2$. And in general, the probabilities for all 8 possible combinations of 3 cells are given by

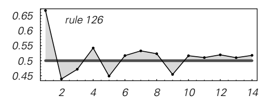
```
probs = Apply[Times, Table[IntegerDigits[8 - i, 2, 3],
      {i, 8}] /. {1 → p, 0 → 1 - p}, {1}]
```

In terms of these probabilities the density at the next step in the evolution of cellular automaton with rule number m is then given by

```
Simplify[probs . IntegerDigits[m, 2, 8]]
```

For rule 22, for example, this means that if the density at a particular step is p, then the density on the next step should be $3p(1-p)^2$, and the densities on subsequent steps should be obtained by iterating this function. (At least for the 256 elementary cellular automata this iterated map is never chaotic.) The stable density after many steps is then given by *Solve[3 p (1 - p)^2 == p, p]*, so that $p \to 1 - 1/\sqrt{3}$ or approximately 0.42. The actual density for rule 22 is however 0.35095. The reason for the discrepancy is that the probabilities for different cells are in fact correlated. One can systematically include more such correlations by looking at more steps of evolution at once. For two steps, one must consider probabilities for all 32 combinations of 5 cells, and for rule 22 the function becomes $p(1-p)^2(2+3p^2)$, yielding density 0.35012; for three steps it is $p(1-p)^2(p^4 - 18p^3 + 41p^2 - 22p + 6)$ yielding density 0.379. The plot below shows what happens with more steps: the results seem to converge slowly to the exact result indicated by the gray line.





(For rules 90 and 30 the functions obtained after one step are respectively $2p(1-p)$ and $p(2p^2 - 5p + 3)$, both of which turn out to imply correct final densities of $1/2$).

Probabilistic approximation schemes like this are often used in statistical physics under the name of mean field theories. In general, such approximations tend to work better for systems in larger numbers of dimensions, where correlations tend to be less important.

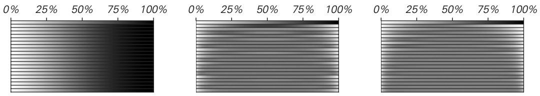
Probabilistic estimates can also be used for other quantities, such as growth rates of difference patterns (see page 949). In most cases, however, buildup of correlations tends to prevent systematic improvement of such approximations.

■ **Density in rule 90.** From the superposition principle above and the number of black cells at step t in a pattern starting from a single black cell (see page 870) one can compute the density after t steps in the evolution of rule 90 with initial conditions of density p to be (see also page 602)

```
1/2 (1 - (1 - 2 p))^(2^DigitCount[t, 2, 1])
```

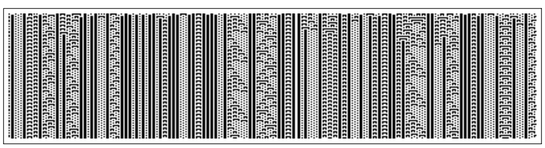
■ **Densities in other rules.** The pictures below show how the densities on successive steps depend on the initial density. Densities are indicated by gray levels. Initial densities are shown across each picture. Successive steps are shown down the page. Rule 236 is class 2, and the density retains a memory of its initial value. But in the class 3 rules 126 and 30, the densities converge quickly to a fixed value.

Page 339 shows a cellular automaton with very different behavior.



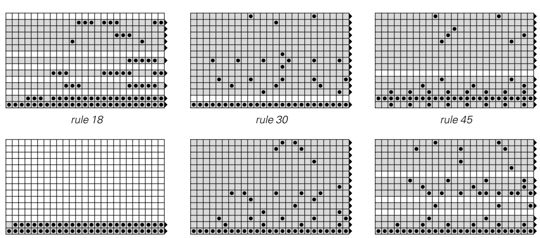
rule 236 rule 126 rule 30

■ **Density oscillations in rule 73.** Although there are always some fluctuations, most rules yield densities that converge more or less uniformly to their final values. One exception is rule 73, which yields densities that continue to oscillate with a period of 3 steps forever. The origin of this phenomenon is that with completely random initial conditions rule 73 evolves to a collection of independent regions, as in the picture below, and many of these regions contain patterns that repeat with period 3. The boundaries between regions come from blocks of even numbers of black cells in the initial conditions, and if one does not allow any such blocks, the density oscillations no longer occur. (See also page 699.)



Special Initial Conditions

■ **Page 267 · Repeating blocks.** The discussion in the main text is mostly about repetition strictly every p steps, and no sooner. (If a system repeats for example every 3 steps, then it is inevitable that it will also repeat in the same way every 6, 9, 12, 15, etc. steps.) Finding configurations in a 1D cellular automaton that repeat with a particular period is equivalent to satisfying the kind of constraints we discussed on page 211. And as described there, if such constraints can be satisfied at all, then it must be possible to satisfy them with a configuration that consists of a repetition of identical blocks. Indeed, for period p, the length of blocks required is at most 2^{2p} (or 2^{2pr} for range r rules).


rule 18 rule 30 rule 45
rule 50 rule 90 rule 110

The pictures at the bottom of the previous column summarize which periods can be obtained with various rules. Periods from 1 to 15 are represented by different rows, with period 1 at the bottom. Within each row a gray bar indicates that a particular period can be obtained with blocks of some length. The black dots indicate specific block sizes up to 25 that work.

In rule 90 (as well as other additive rules such as 60 and 150) any period can occur, but all configurations that repeat must consist of a sequence of identical blocks. For periods up to 10, examples of such blocks in rule 90 are given by the digits of

{0, 40, 24, 2176, 107904, 640, 96, 8421376, 7566031296234863392, 15561286137}

For period 1 the possible blocks are □ and ■■□; for period 2 ■□■□□□ and ■■■■□□. The total number of configurations in rule 90 that repeat with any period that divides p is always 4^p.

Rules 30 and 45 (as well as other one-sided additive rules) also have the property that all configurations that repeat must consist of a sequence of identical blocks. The total number of configurations in rule 30 that repeat with periods that divide 1 through 10 are {3, 3, 15, 10, 8, 99, 18, 14, 30, 163}. In general for one-sided additive rules the number of such configurations increases for large p like $k^{h_{tx} p}$, where h_{tx} is the spacetime entropy of page 960. (This is the analog of a standard result in dynamical systems theory about expansive homeomorphisms.)

For rules that do not show at least one-sided additivity there can be an infinite number of configurations that repeat with a given period. To find them one considers all possible blocks of length $2pr+1$ and picks out those that after p steps evolve so that their center cell ends up the same color as it was originally. The possible configurations that repeat with period p then correspond to the finite complement language (see page 958) obtained by stringing together these blocks. For $p = 2$, rule 18 leaves 20 of the 32 possible length 5 blocks invariant, but these blocks can only be strung together to yield repetitions of {a, b, 0, 0}, where now a and b are not fixed, but in every case can each be either {1} or {0, 1}.

(See also page 700.)

■ **Localized structures.** See pages 281 and 1118.

■ **2D cellular automata.** As expected from the discussion of constraints on page 942, the problem of finding repeating configurations is much more difficult in two dimensions than in one dimension. Thus for example unlike in 1D there is no guarantee in 2D that among repeating configurations of a particular period there is necessarily one that consists just of a repetitive array of fixed blocks. Indeed, as discussed on page 1139, it is in a sense possible that the only repeating configurations are arbitrarily complex. Note that if one considers configurations in 2D that consist only of infinitely long stripes, then the problem reduces again to the 1D case. (See also page 349.)

■ **Systems based on numbers.** An iterated map of the kind discussed on page 150 with rule *x → Mod[a x, 1]* (with rational *a*) will yield repetitive behavior when its initial condition is a rational number. The same is true for higher-dimensional generalizations such as so-called Anosov maps *{x, y} → Mod[m . {x, y}, 1]*. The continued fraction map *x → Mod[1/x, 1]* discussed on page 914 becomes repetitive whenever its initial condition is a solution to a quadratic equation.

For a map *x → f[x]* where *f[x]* is a polynomial such as *a x (1 - x)* the real initial conditions that yield period *p* are given by

```
Select[x /. Solve[Nest[f, x, p] == x, x], Im[#] == 0 &]
```

For *x → a x (1 - x)* the results usually cannot be expressed in terms of explicit radicals beyond period 2. (See page 961.)

■ **Sarkovskii's theorem.** For any iterated map based on a continuous function such as a polynomial it was shown in 1962 that if an initial condition exists that gives period 3, then other initial conditions must exist that give any other period. In general, if a period *m* is possible then so must all periods *n* for which *p = {m, n}* satisfies

```
OrderedQ[(Transpose[If[MemberQ[p/#, 1], Map[Reverse,
        {p/#, #}], {#, p/#}]] &)[2^IntegerExponent[p, 2]]]
```

Extensions of this to other types of systems seem difficult to find, but it is conceivable that when viewed as continuous mappings on a Cantor set (see page 869) at least some cellular automata might exhibit similar properties.

■ **Page 269 · Rule emulations.** See pages 702 and 1118.

■ **Renormalization group.** The notion of studying systems by seeing the effect of changing the scale on which one looks at them has been widely used in physics since about 1970, and there is some analogy between this and what I do here with cellular automata. In the lattice version in physics one typically considers what happens to averages over all possible configurations of a system if one does a so-called blocking transformation that replaces blocks of elements by individual elements. And what one finds is that in certain cases—notably in connection with nesting at critical points associated with phase transitions (see page 981)—certain averages turn out to be the same as one would get if one did no blocking but just changed parameters ("coupling constants") in the underlying rules that specify the weighting of different configurations. How such effective parameters change with scale is then governed by so-called renormalization group differential equations. And when one looks at large scales the versions of these equations that arise in practice essentially always show fixed points, whose properties do not depend much on details of the equations—leading to certain universal results across many different underlying systems (see page 983).

What I do in the main text can be thought of as carrying out blocking transformations on cellular automata. But only rarely do such transformations yield cellular automata whose rules are of the same type one started from. And in most cases such rules will not suffice even if one takes averages. And indeed, so far as I can tell, only in those cases where there is fairly simple nested behavior is any direct analog of renormalization group methods useful. (See page 989.)

■ **Page 271 · Self-similarity of additive rules.** The fact that rule 90 can emulate itself can be seen fairly easily from a symbolic description of the rule. Given three cells *{a_1, a_2, a_3}* the rule specifies that the new value of the center cell will be *Mod[a_1 + a_3, 2]*. But given *{a_1, 0, a_2, 0, a_3, 0}* the value after one step is *{Mod[a_1 + a_2, 2], 0, Mod[a_2 + a_3, 2], 0}* and after two steps is again *{Mod[a_1 + a_3, 2], 0}*. It turns out that this argument generalizes (by interspersing *k - 1* 0's and going for *k* steps) to any additive rule based on reduction modulo *k* (see page 952) so long as *k* is prime. And it follows that in this case the pattern generated after a certain number of steps from a single non-white cell will always be the same as one gets by going *k* times that number of steps and then keeping only every k^{th} row and column. And this immediately implies that the pattern must always have a nested form. If *k* is not prime the pattern is no longer strictly invariant with respect to keeping only every k^{th} row and column—but is in effect still a superposition of patterns with this property for factor of *k*. (Compare page 870.)

■ **Fractal dimensions.** The total number of nonzero cells in the first *t* rows of the pattern generated by the evolution of an additive cellular automaton with *k* colors and weights *w* (see page 952) from a single initial *1* can be found using

```
g[w_, k_, t_] := Apply[Plus, Sign[NestList[Mod[
        ListCorrelate[w, #, {-1, 1}, 0], k] &, {1}, t - 1]], {0, 1}]
```

The fractal dimension of this pattern is then given by the large *m* limit of

```
Log[k, g[w, k, k^(m+1)]/g[w, k, k^m]]
```

When *k* is prime it turns out that this can be computed as

```
d[w_, k_ : 2] := Log[k, Max[Abs[Eigenvalues[With[
  {s = Length[w] - 1}, (Map[Function[u, Map[Count[u, #] &,
   #1]], Map[Flatten[Map[Partition[Take[#, k + s - 1], s, 1] &,
   NestList[Mod[ListConvolve[w, #], k] &, #, k - 1]], 1] &,
   Map[Flatten[Map[{Table[0, {k - 1}], #} &, Append[#,
    0]]] &, #]]] &)[Array[IntegerDigits[#, k, s] &, k^s - 1]]]]]]]
```

For rule 90 one gets *d[{1, 0, 1}] = Log[2, 3] ≃ 1.58*. For rule 150 *d[{1, 1, 1}] = Log[2, 1 +* $\sqrt{5}$ *] ≃ 1.69*. (See page 58.) For the other rules on page 952:

d[{1, 1, 0, 1, 0}] =
  Log[2, Root[4 + 2 # - 2 #^{2} *- 3 #*^{3} *+ #*^{4} *&, 2]] ≃ 1.72*
d[{1, 1, 0, 1, 1}] =
  Log[2, Root[-4 + 4 # + #^{2} *- 4 #*^{3} *+ #*^{4} *&, 2]] ≃ 1.8*

Other cases include (see page 870):

d[{1, 0, 1}, k] = 1 + Log[k, (k + 1)/2]

d[{1, 1, 1}, 3] = Log[3, 6] ≃ 1.63

d[{1, 1, 1}, 5] = Log[5, 19] ≃ 1.83

d[{1, 1, 1}, 7] = Log[7, Root[-27136 + 23280 # -
  7288 #^{2} *+ 1008 #*^{3} *- 59 #*^{4} *+ #*^{5} *&, 1]] ≃ 1.85*

■ **General associative rules.** With a cellular automaton rule in which the new color of a cell is given by *f[*a_1*,* a_2*]* (compare page 886) it turns out that the pattern generated by evolution from a single non-white cell is always nested if the function *f* has the property of being associative or *Flat*. In fact, for a system involving *k* colors the pattern produced will always be essentially just one of the patterns obtained from an additive rule with *k* or less colors. In general, the pattern produced by evolution for *t* steps is given by

```
NestList[
  Inner[f, Prepend[#, 0], Append[#, 0], List] &, {a}, t]
```

so that the first few steps yield

```
{a}
{f[0, a], f[a, 0]}
{f[0, f[0, a]], f[f[0, a], f[a, 0]], f[f[a, 0], 0]}
{f[0, f[0, f[0, a]]], f[f[0, f[0, a]], f[f[0, a], f[a, 0]]],
  f[f[f[0, a], f[a, 0]], f[f[a, 0], 0]], f[f[f[a, 0], 0], 0]}
```

If *f* is *Flat*, however, then the last two lines here become

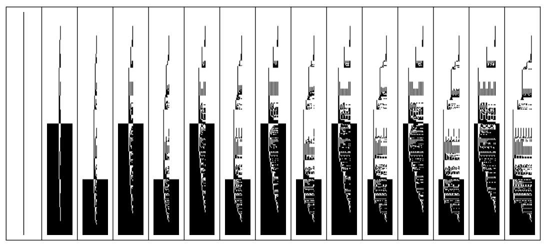
```
{f[0, 0, a], f[0, a, a, 0], f[a, 0, 0]}
{f[0, 0, 0, a], f[0, 0, a, 0, a, a, 0],
  f[0, a, a, 0, a, 0, 0], f[a, 0, 0, 0]}
```

and in general the number of *a*’s that appear in a particular element is given as in Pascal’s triangle by a binomial coefficient. If *f* is commutative (*Orderless*) then all that can ever matter to the value of an element is its number of *a*’s. Yet since there are a finite set of possible values for each element it immediately follows that the resulting pattern must be essentially Pascal’s triangle modulo some integer. And even if *f* is not commutative, the same result will hold so long as *f[0, a] == a* and *f[a, 0] == a*—since then any element can be reduced to *f[a, a, a, …]*. The result can also be generalized to cellular automata with basic rules involving more than two elements—since if *f* is *Flat*, *f[*a_1*,* a_2*,* a_3*]* is always just *f[f[*a_1*,* a_2*],* a_3*]*.

If one starts from more than a single non-*0* element, then it is still true that a nested pattern will be produced if *f* is both associative and commutative. And from the discussion on page 952 this means that any rule that shows generalized additivity must always yield a nested pattern. But if f is not commutative, then even if it is associative, non-nested patterns can be produced. And indeed page 887 shows an example of this based on the non-commutative group S_3. (In general *f* can correspond to an almost arbitrary semigroup, but with a single initial element only a cyclic subgroup of it is ever explored.)

■ **Nesting in rule 45.** As illustrated on page 701, starting from a single black cell on a background of repeated ■□ blocks, rule 45 yields a slanted version of the nested rule 90 pattern.

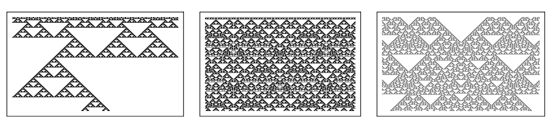
■ **Uniqueness of patterns.** Starting from a particular initial condition, different rules can often yield the same pattern. The picture below shows in sorted order the configurations obtained at each successive step in the evolution of all 256 elementary cellular automata starting from a single black cell. After a large number of steps, between 94 and 105 distinct individual configurations are obtained, together with 143 distinct complete patterns. (Compare page 1186.)



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

■ **Square root of rule 30.** Although rule 30 cannot apparently be decomposed into other *k = 2*, *r = 1* cellular automata, it can be viewed as the square of the *k = 3*, *r = 1/2* cellular automata with rule numbers 11736, 11739 and 11742.

■ **Page 272 · Nested initial conditions.** The pictures below show patterns generated by rule 90 starting from the nested sequences on page 83. (See page 1091.)



(a) (b) (c)

The Notion of Attractors

■ **Page 275 · Discrete systems.** In traditional mathematics mechanical and other systems are assumed continuous, so that for example a pendulum may get exponentially close to the attractor state where it has stopped, but it will never strictly reach this attractor. In discrete systems like cellular automata, however, there is no problem in explicitly reaching at least simple attractors.

■ **Implementation.** One can represent a network by a list such as *{{1 → 2}, {0 → 3, 1 → 2}, {0 → 3, 1 → 1}}* where each element represents a node whose number corresponds to the position of the element, and for each node there are rules that specify to which nodes arcs with different values lead. Starting with a list of nodes, the nodes reached by following arcs with value *a* for one step are given by

```
NetStep[net_, i_, a_] :=
  Union[ReplaceList[a, Flatten[net⟦i⟧]]]
```

A list of values then corresponds to a path in the network starting from any node if

```
Fold[NetStep[net, #1, #2] &,
    Range[Length[net]], list] =!= {}
```

Given a set of sequences of values represented by a particular network, the set obtained after one step of cellular automaton evolution is given by

```
NetCAStep[{k_, r_, rtab_}, net_] := Flatten[
  Map[Table[# /. (a_ → s_) :→ rtab⟦i k + a + 1⟧ → k^(2 r) (s - 1) +
        1 + Mod[i k + a, k^(2 r)], {i, 0, k^(2 r) - 1}] &, net], 1]
```

where here elementary rule 126 is specified for example by *{2, 1, Reverse[IntegerDigits[126, 2, 8]]}*. Starting from the set of all possible sequences, as given by

```
AllNet[k_ : 2] := {Thread[Range[k] - 1 → 1]}
```

this then yields for rule 126 the network

```
{{0 → 1, 1 → 2}, {1 → 3, 1 → 4}, {1 → 1, 1 → 2}, {1 → 3, 0 → 4}}
```

It is always possible to find a minimal network that represents a set of sequences. This can be done by first creating a “deterministic” network in which at most one arc of each value comes out of each node, then combining equivalent nodes. The whole procedure can be performed using

```
MinNet[net_, k_ : 2] := Module[{d = DSets[net, k], q, b},
  If[First[d] =!= {}, AllNet[k], q = ISets[b = Map[Table[
      Position[d, NetStep[net, #, a]]⟦1, 1⟧, {a, 0, k - 1}] &, d]];
   DeleteCases[MapIndexed[#2⟦2⟧ - 1 → #1 &, Rest[
     Map[Position[q, #]⟦1, 1⟧ &, Transpose[Map[#⟦Map[
        First, q]⟧ &, Transpose[b]]], {2}]] - 1, {2}], _ → 0, {2}]]]
DSets[net_, k_ : 2] :=
  FixedPoint[Union[Flatten[Map[Table[NetStep[net, #, a],
            {a, 0, k - 1}] &, #], 1]] &, {Range[Length[net]]}]
ISets[list_] := FixedPoint[Function[g, Flatten[Map[
   Map[Last, Split[Sort[Transpose[{Map[Position[g, #]⟦1,
      1⟧ &, list, {2}], Range[Length[list]]}]⟦#⟧], First[#1] ==
       First[#2] &], {2}] &, g], 1]], {{1}, Range[2, Length[list]]}]
```

If *net* has *q* nodes, then in general *MinNet[net]* can have as many as $2^q - 1$ nodes. The form of *MinNet* given here can take up to about n^2 steps to generate a result with *n* nodes; an *n Log[n]* procedure is known. The result from *MinNet* for rule 126 is *{{1 → 3}, {0 → 2, 1 → 1}, {0 → 2, 1 → 3}}*.

In general *MinNet* will yield a network with the property that any allowed sequence of values corresponds to a path which starts from node 1. In the main text, however, the networks allow paths that start at any node. To obtain such trimmed networks one can apply the function

```
TrimNet[net_] :=
  With[{m = Apply[Intersection, Map[FixedPoint[
        Union[#, Flatten[Map[Last, net⟦#⟧], {2}]]] &,
        #] &, Map[List, Range[Length[net]]]]]},
   net⟦m⟧ /. Table[(a_ → m⟦i⟧) → a → i, {i, Length[m]}]]
```

■ **Finite automata.** The networks discussed in the main text can be viewed as finite automata (also known as finite state machines). Each node in the network corresponds to a state in the automaton, and each arc represents a transition that occurs when a particular value is given as input. *NetCAStep* above in general produces a non-deterministic finite automaton (NDFA) for which a particular sequence of values does not determine a unique path through the network. *MinNet* creates an equivalent DFA, then minimizes this. The Myhill-Nerode theorem ensures that a unique minimal DFA can always be found (though to do so is in general a PSPACE-complete problem).

The total number of distinct minimal finite automata with $k = 2$ possible labels for each arc grows with the number of nodes as follows: 3, 7, 78, 1388, … (The simple result $(n+1)^{n k}$ based on the number of ways to connect up *n* nodes is a significant overestimate because of equivalence between automata with different patterns of connections.)

■ **Regular languages.** The set of sequences obtained by following possible paths through a finite network is often called a regular language, and appears in studies of many kinds of systems. (See page 939.)

■ **Regular expressions.** The sequences in a regular language correspond to those that can be matched by *Mathematica* patterns that use no explicit pattern names. Thus for example *{(0 | 1) ...}* corresponds to all possible sequences of *0*’s and *1*’s, while *{1, 1, (1) ..., 0, (0) ...} ...* corresponds to the sequences that can occur after 2 steps in rule 126 and *{(0) ..., 1, {0, (0) ..., 1, 1} | {1, (1) ..., 0}} ...* to those that can occur after 2 steps in rule 110 (see page 279).

■ **Generating functions.** The sequences in a regular language can be thought of as corresponding to products of non-commuting variables that appear as coefficients in a formal power series expansion of a generating function. A basic result is that for regular languages this generating function is always rational. (Compare the discussion of entropies below.)

■ **History.** Simple finite automata have implicitly been used in electromechanical machines for over a century. A formal version of them appeared in 1943 in McCulloch-Pitts neural network models. (An earlier analog had appeared in Markov chains.) Intensive work on them in the 1950s (sometimes under the name sequential machines) established many basic properties, including interpretation as regular languages and equivalence to regular expressions. Connections to formal power series and to substitution systems (see page 891) were studied in the 1960s. And with the development of the Unix operating system in the 1970s regular expressions began to be widely used in practical computing in lexical analysis (lex) and text searching (ed and grep). Regular languages also arose in dynamical systems theory in the early 1970s under the name of sofic systems.

■ **Page 278 · Network properties.** The number of nodes and connections at step $t > 1$ are: rule 108: 8, 13; rule 128: $2t$, $2t+2$; rule 132: $2t+1$, $3t+3$; rule 160: $(t+1)^2$, $(t+1)(t+3)$; rule 184: $2t$, $3t+1$. For rule 126 the first few cases are

{{1, 2}, {3, 5}, {13, 23}, {106, 196}, {2866, 5474}}

and for rule 110 they are

{{1, 2}, {5, 9}, {20, 38}, {206, 403}, {1353, 2666}}

The maximum size of network that can possibly be generated after t steps of cellular automaton evolution is $2^{k^{2rt}} - 1$. For $t = 1$ the maximum of 15 (with 29 connections) is achieved for 16 out of the 256 possible elementary rules, including 22, 37, 73, 94, 104, 122, 146 and 164. For $t = 2$, rule 22 gives the largest network, with 280 nodes and 551 arcs. The $k = 2$, $r = 2$ totalistic rule with code 20 gives a network with 65535 nodes after just 1 step. Note that rules which yield maximal size networks are in a sense close to allowing all possible sequences. (The shortest excluded block for code 20 is of length 36.)

■ **Excluded blocks.** As the evolution of a cellular automaton proceeds, the set of sequences that can appear typically shrinks, with progressively more blocks being excluded. In some cases the set of allowed sequences forms a so-called finite complement language (or subshift of finite type) that can be characterized completely just by saying that some finite set of blocks are excluded. But whenever the overall behavior is at all complex, there tend to be an infinite set of blocks excluded, making it necessary to use a network of the kind discussed in the main text. If there are n nodes in such a network, then if any blocks are excluded, the shortest one of them must be of length less than n. And if there are going to be an infinite number of excluded blocks, there must be additional excluded blocks with lengths between n and $2n$. In rule 126, the lengths of the shortest newly excluded blocks on successive steps are 0, 3, 12, 13, 14, 14, 17, 15. It is common to see such lengths progressively increase, although in principle they can decrease by as much as $2r$ from one step to the next. (As an example, in rule 54 they decrease from 9 to 7 between steps 4 and 5.)

■ **Entropies and dimensions.** There are 2^n sequences possible for n cells that are each either black or white. But as we have seen, in most cellular automata not all these sequences can occur except in the initial conditions. The number of sequences s_n of length n that can actually occur is given by

```
Apply[Plus, Flatten[MatrixPower[m, n]]]
```

where the adjacency matrix m is given by

```
MapAt[1 + # &, Table[0, {Length[net]}, {Length[net]}],
  Flatten[MapIndexed[{First[#2], Last[#1]} &, net, {2}], 1]]
```

For rule 32, for example, s_n turns out to be Fibonacci[n + 3], so that for large n it is approximately GoldenRatio^n. For any rule, s_n for large n will behave like κ^n, where κ is the largest eigenvalue of m. For rule 126 after 1 step, the characteristic polynomial for m is $x^3 - 2x^2 + x - 1$, giving $\kappa \simeq 1.755$. After 2 steps, the polynomial is

$$x^{13} - 4x^{12} + 6x^{11} - 5x^{10} + 3x^9 - 3x^8 + 5x^7 - 3x^6 - x^5 + 4x^4 - 2x^3 + x^2 - x + 1$$

giving $\kappa \simeq 1.732$. Note that κ is always an algebraic number—or strictly a so-called Perron number, obtained from a polynomial with leading coefficient 1. (Note that any possible Perron number can be obtained for example from some finite complement language.)

It is often convenient to fit s_n for large n to the form 2^{hn}, where h is the so-called spatial (topological) entropy (see page 1084), given by Log[2, κ]. The value of this for successive t never increases; for the first 3 steps in rule 126 it is for example approximately 1, 0.811, 0.793. The exact value of h after more steps tends to be very difficult to find, and indeed the question of whether its limiting value after infinitely many steps satisfies a given bound—say even being nonzero—is in general undecidable (see page 1138).

If one associates with each possible sequence of length n a number Sum[$a_i 2^{-i}$, {i, n}], then the set of sequences that actually occur at a given step form a Cantor set (see note below), whose Hausdorff dimension turns out to be exactly h.

■ **Cycles and zeta functions.** The number of sequences of n cells that can occur repeatedly, corresponding to cycles in the network, is given in terms of the adjacency matrix m by Tr[MatrixPower[m, n]]. These numbers can also be obtained as the coefficients of x^n in the series expansion of

$x\,\partial_x$ *Log[ζ[m, x]]*, with the so-called zeta function, which is always a rational function of x, given by

ζ[m_, x_] := 1/Det[IdentityMatrix[Length[m]] - m x]

and corresponds to the product over all cycles of $1/(1 - x^n)$.

▪ 2D generalizations. Above 1D no systematic method seems to exist for finding exact formulas for entropies (as expected from the discussion at the end of Chapter 5). Indeed, even working out for large n how many of the 2^{n^2} possible configurations of a $n \times n$ grid of black and white squares contain no pair of adjacent black cells is difficult. Fitting the result to $2^{h\,n^2}$ one finds $h \simeq 0.589$, but no exact formula for h has ever been found. With hexagonal cells, however, the exact solution of the so-called hard hexagon lattice gas model in 1980 showed that $h \simeq 0.481$ is the logarithm of the largest root of a degree 12 polynomial. (The solution of the so-called dimer problem in 1961 also showed that for complete coverings of a square grid by 2-cell dominoes *h = Catalan/(π Log[2]) ≃ 0.421*.)

▪ Probability-based entropies. This section has concentrated on characterizing what sequences can possibly occur in 1D cellular automata, with no regard to their probability. It turns out to be difficult to extend the discussion of networks to include probabilities in a rigorous way. But it is straightforward to define versions of entropy that take account of probabilities—and indeed the closest analog to the usual entropy in physics or information theory is obtained by taking the probabilities *p[i]* for the k^n blocks of length n (assuming k colors), then constructing

-Limit[Sum[p[i] Log[k, p[i]], {i, k^n}]/n, n → ∞]

I have tended to call this quantity measure entropy, though in other contexts, it is often just called entropy or information, and is sometimes called information dimension. The quantity

Limit[Sum[UnitStep[p[i]], {i, k^n}]/n, n → ∞]

is the entropy discussed in the notes above, and is variously called set entropy, topological entropy, capacity and fractal dimension. An example of a generalization is the quantity given for blocks of size n by

h[q_, n_] := Log[k, Sum[p[i]^q, {i, k^n}]]/(n (q - 1))

where $q = 0$ yields set entropy, the limit $q \to 1$ measure entropy, and $q = 2$ so-called correlation entropy. For any q the maximum *h[q, n] == 1* occurs when all *p[i] == k^{-n}*. It is always the case that *h[q + 1, n] ≤ h[q, n]*. The *h[q]* have been introduced in almost identical form several times, notably by Alfréd Rényi in the 1950s as information measures for probability distributions, in the 1970s as part of the thermodynamic formalism for dynamical systems, and in the 1980s as generalized dimensions for multifractals. (Related objects have also arisen in connection with Hölder exponents for discontinuous functions.)

▪ Entropy estimates. Entropies *h[n]* computed from blocks of size n always decrease with n; the quantity *n h[n]* is always convex (negative second difference) with respect to n. At least at a basic level, to compute topological entropy one needs in effect to count every possible sequence that can be generated. But one can potentially get an estimate of measure entropy just by sampling possible sequences. One problem, however, is that even though such sampling may give estimates of probabilities that are unbiased (and have Gaussian errors), a direct computation of measure entropy from them will tend to give a value that is systematically too small. (A potential way around this is to use the theory of unbiased estimators for polynomials just above and below *p Log[p]*.)

▪ Nested structure of attractors. Associating with each sequence of length n (and k possible colors for each element) a number *Sum[a[i] k^{-i}, {i, n}]*, the set of sequences that occur in the limit $n \to \infty$ forms a Cantor set. For $k = 3$, the set of sequences where the second color never occurs corresponds to the standard middle-thirds Cantor set. In general, whenever the possible sequences correspond to paths through a finite network, it follows that the Cantor set obtained has a nested structure. Indeed, constructing the Cantor set in levels by considering progressively longer sequences is effectively equivalent to following successive steps in a substitution system of the kind discussed on page 83. (To see the equivalence first set up s kinds of elements in the substitution system corresponding to the s nodes in the network.) Note that if the possible sequences cannot be described by a network, then the Cantor set obtained will inevitably not have a strictly nested form.

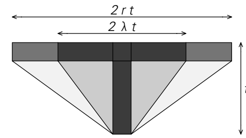
▪ Surjectivity and injectivity. One can think of a cellular automaton rule as a mapping (endomorphism) from the space of possible states of the cellular automaton to itself. (See page 869.) Usually this mapping is contractive, so that not all the states which appear as input to the mapping can also appear as output. But in some cases, the mapping is surjective or onto, meaning that any state which appears as input can also appear as output. Among $k = 2$, $r = 1$ elementary cellular automata it turns out that this happens precisely for those 30 rules that are additive with respect to at least the first or last position on which they depend (see pages 601 and 1087); this includes both rules 90 and 150 and rules 30 and 45. With $k = 2$, $r = 2$ there are a total of 4,294,967,296 possible rules. Out of these 141,884 are onto—and 11,388 of these turn out not to be additive with respect to any position. The easiest way to test whether a particular rule is onto seems to be essentially just to construct the minimal finite automaton discussed on page 957. The onto

$k = 2$, $r = 2$ rules were found in 1961 in a computer study by Gustav Hedlund and others; they later apparently provided input in the design of S-boxes for DES cryptography (see page 1085).

Even when a cellular automaton mapping is surjective, it is still often many-to-one, in the sense that several input states can yield the same output state. (Thus for example additive rules such as 90 and 150, as well as one-sided additive rules such as 30 and 45 are always 4-to-1.) But some surjective rules also have the property of being injective, so that different input states always yield different output states. And in such a case the cellular automaton mapping is one-to-one or bijective (an automorphism). This is equivalent to saying that the rule is reversible, as discussed on page 1017.

(In 2D such properties are in general undecidable; see page 1138.)

■ **Temporal sequences.** So far we have considered possible sequences of cells that can occur at a particular step in the evolution of a cellular automaton. But one can also consider sequences formed from the color of a particular cell on a succession of steps. For class 1 and 2 cellular automata, there are typically only a limited number of possible sequences of any length allowed. And when the length is large, the sequences are almost always either just uniform or repetitive. For class 3 cellular automata, however, the number of sequences of length n typically grows rapidly with n. For additive rules such as 60 and 90, and for partially additive rules such as 30 and 45, any possible sequence can occur if an appropriate initial condition is given. For rule 18, it appears that any sequence can occur that never contains more than one adjacent black cell. I know of no general characterization of temporal sequences analogous to the finite automaton one used for spatial sequences above. However, if one defines the entropy or dimension h_t for temporal sequences by analogy with the definition for spatial sequences above, then it follows for example that $h_t \leq 2 \lambda h_x$, where λ is the maximum rate at which changes grow in the cellular automaton. The origin of this inequality is indicated in the picture below. The basic idea is that the size of the region that can affect a given cell in the course of t steps is $2 \lambda t$. But for large sizes x the total number of possible configurations of this region is $k^{h_x x}$. (Inequalities between entropies and Lyapunov exponents are also common in dynamical systems based on numbers, but are more difficult to derive.) Note that in effect, h_x gives the information content of spatial sequences in units of bits per unit distance, while h_t gives the corresponding quantity for temporal sequences in units of bits per unit time. (One can also define directional entropies based on sequences at different slopes; the values of such entropies tend to change discontinuously when the slope crosses λ.)



Different classes of cellular automata show characteristically different entropy values. Class 1 has $h_x = 0$ and $h_t = 0$. Class 2 has $h_x \neq 0$ but $h_t = 0$. Class 3 has $h_x \neq 0$ and $h_t \neq 0$. Class 4 tends to show fluctuations which prevent definite values of h_x and h_t from being found.

■ **Spacetime patches.** One can imagine defining entropies and dimensions associated with regions of any shape in the spacetime history of a cellular automaton. As an example, one can consider patches that extend x cells across in space and t cells down in time. If the color of every cell in such a patch could be chosen independently then there would be $k^{t x}$ possible configurations of the complete patch. But in fact, having just specified a block of length $x + 2 r t$ in the initial conditions, the cellular automaton rule then uniquely determines the color of every cell in the patch, allowing a total of at most $s[t, x] = k^{x+2 r t}$ configurations. One can define a topological spacetime entropy h_{tx} as

$$Limit[Limit[Log[k, s[t, x]]/t, t \to \infty], x \to \infty]$$

and a measure spacetime entropy h^{μ}_{tx} by replacing s with $p\, Log[p]$. In general, $h_t \leq h_{tx} \leq 2 \lambda h_x$ and $h \leq 2 r h_t$. For additive rules like rule 90 and rule 150 every possible configuration of the initial block leads to a different configuration for the patch, so that $h_{tx} = 2 r = 2$. But for other rules many different configurations of the initial block can lead to the same configuration for the patch, yielding potentially much smaller values of h_{tx}. Just as for most other entropies, when a cellular automaton shows complicated behavior it tends to be difficult to find much more than upper bounds for h_{tx}. For rule 30, $h^{\mu}_{tx} < 1.155$, and there is some evidence that its true value may actually be 1. For rule 18 it appears that $h^{\mu}_{tx} = 1$, while for rule 22, $h^{\mu}_{tx} < 0.915$ and for rule 54 $h^{\mu}_{tx} < 0.25$.

■ **History.** The analysis of cellular automata given in this section is largely as I worked it out in the early 1980s. Parts of it, however, are related to earlier investigations, particularly in dynamical systems theory. Starting in the 1930s the idea of symbolic dynamics began to emerge, in which one partitions continuous values in a system into bins represented by discrete symbols, and then looks at the sequences of such symbols that can be produced by the evolution of the system. In connection with early work on chaos theory, it was noted that there are some systems that act like "full shifts", in the sense that the set of sequences they generate includes all possibilities—and corresponds to what one would get by starting with any possible number, then successively shifting digits to the left, and at each step picking off the leading digit. It was noted that some systems could also yield various kinds of subshifts that are subsets of full shifts. But since—unlike in cellular automata—the symbol sequences being studied were obtained by rather arbitrary partitionings of continuous values, the question arose of what effect using different partitionings would have. One approach was to try to find invariants that would remain unchanged in different partitionings—and this is what led, for example, to the study of topological entropy in the 1960s. Another approach was to look at actual possible transformations between partitionings, and this led from the late 1950s to various studies of so-called shift-commuting block maps (or sliding-block codes)—which turn out to be exactly 1D cellular automata (see page 878). The locality of cellular automaton rules was thought of as making them the analog for symbol sequences of continuous functions for real numbers (compare page 869). Of particular interest were invertible (reversible) cellular automaton rules, since systems related by these were considered conjugate or topologically equivalent.

In the 1950s and 1960s—quite independent of symbolic dynamics—there was a certain amount of work done in connection with ideas about self-reproduction (see page 876) on the question of what configurations one could arrange to produce in 1D and 2D cellular automata. And this led for example to the study of so-called Garden of Eden states that can appear only in initial conditions—as well as to some general discussion of properties such as surjectivity.

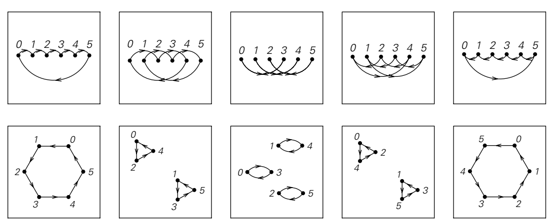
When I started working on cellular automata in the early 1980s I wanted to see how far one could get by following ideas of statistical mechanics and dynamical systems theory and trying to find global characterizations of the possible behavior of individual cellular automata. In the traditional symbolic dynamics of continuous systems it had always been assumed that meaningful quantities must be invariant under continuous invertible transformations of symbol sequences. It turns out that the spacetime (or "invariant") entropy defined in the previous note has this property. But the spatial and temporal entropies that I introduced do not—and indeed in studying specific cellular automata there seems to be no particular reason why such a property would be useful.

■ **Attractors in systems based on numbers.** Particularly for systems based on ordinary differential equations (see page 922) a geometrical classification of possible attractors exists. There are fixed points, limit cycles and so-called strange attractors. (The first two of these were identified around the end of the 1800s; the last with clarity only in the 1960s.) Fixed points correspond to zero-dimensional subsets of the space of possible states, limit cycles to one-dimensional subsets (circles, solenoids, etc.). Strange attractors often have a nested structure with non-integer fractal dimension. But even in cases where the behavior obtained with a particular random initial condition is very complicated the structure of the attractor is almost invariably quite simple.

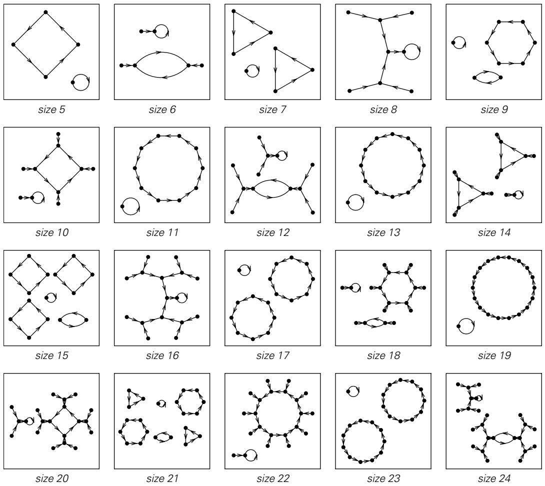
■ **Iterated maps.** For maps of the form $x \to a\,x\,(1-x)$ discussed on page 920 the attractor for small a is a fixed point, then a period 2 limit cycle, then period 4, 8, 16, etc. There is an accumulation of limit cycles at $a \simeq 3.569946$ where the system has a special nested structure. (See pages 920 and 955.)

■ **Attractors in Turing machines.** In theoretical studies Turing machines are often set up so that if their initial conditions follow a particular formal grammar (see page 938) then they evolve to "accept" states—which can be thought of as being somewhat like attractors.

■ **Systems of limited size.** For any system with a limited total number of states, it is possible to create a finite network that gives a global representation of the behavior of the system. The idea of this network (which is very different from the finite automata networks discussed above) is to have each node represent a complete state of the system. At each step in the evolution of the system, every state evolves to some new state, and this process is represented in the network by an arc that joins each node to a new node. The picture below gives the networks obtained for systems of the kind shown on page 255. Each node is labelled by a possible position for the dot. In the first case shown, starting for example at position 4 the dot then visits positions 5, 0, 1, 2 and so on, at each step going from one node in the network to the next.

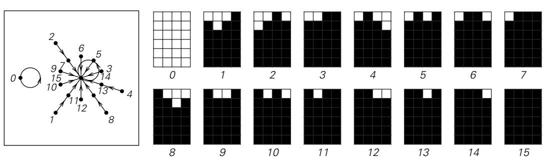


The pictures below give networks obtained from the system shown on page 257 for various values of n. For odd n, the networks consist purely of cycles. But for even n, there are also trees of states that lead to these cycles.

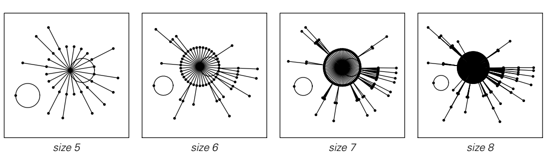


In general, any network that represents the evolution of a system with definite rules will have the same basic form. There are cycles which contain states that are visited repeatedly, and there can also be trees that represent transient states that can each only ever occur at most once in the evolution of the system.

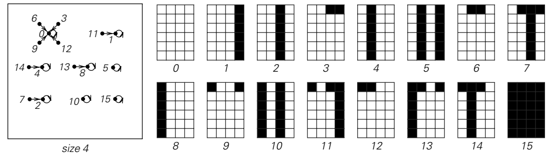
The picture below shows the network obtained from a class 1 cellular automaton (rule 254) with 4 cells and thus 16 possible states. All but one of these 16 states evolve after at most two steps to state 15, which corresponds to all cells being black.

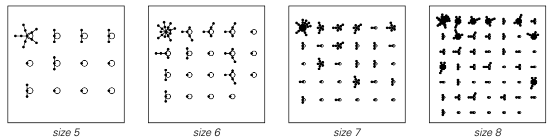


The pictures below show networks obtained when more cells are included in the cellular automaton above. The same convergence to a single fixed point is observed.

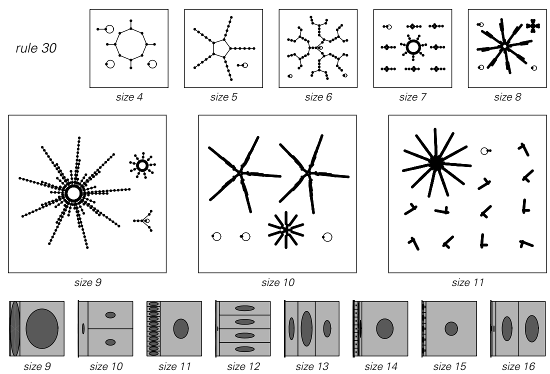


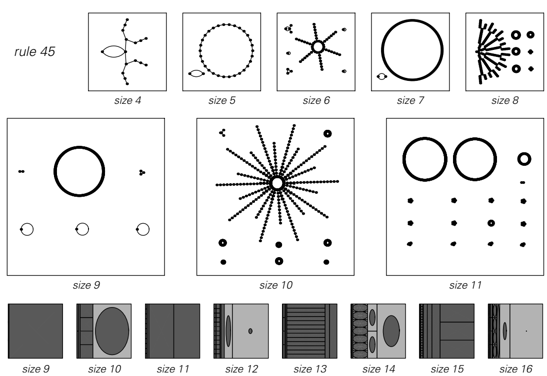
The pictures below give corresponding results for a class 2 cellular automaton (rule 132). The number of distinct cycles now increases with the size of the system. (As discussed below, identical pieces of the network are often related by symmetries of the underlying cellular automaton system.)

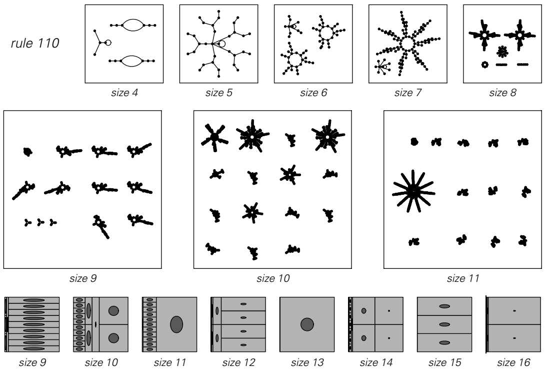




In class 3, larger cycles are usually obtained, and often the whole network is dominated by a single largest cycle. The second set of pictures below summarize the results for some larger cellular automata. Each distinct region corresponds to a disjoint part of the network, with the area of the region being proportional to the number of nodes involved. The dark blobs represent cycles. (See page 1087.)







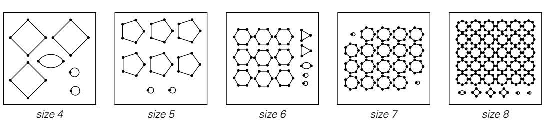
For large sizes there is a rough correspondence with the infinite size case, but many features are still different. (To recover correct infinite size results one must increase size while keeping the number of steps of evolution fixed; the networks shown above, however, effectively depend on arbitrarily many steps of evolution.)

▪ **Symmetries.** Many of the networks above contain large numbers of identical pieces. Typically the reason is that the states in each piece are shifted copies of each other, and in such cases the number of pieces will be a divisor of n. (See page 950.) If the underlying cellular automaton rule exhibits an invariance—say under reflection in space or permutation of colors—this will also often lead to the presence of identical pieces in the final network, corresponding to cosets of the symmetry transformation.

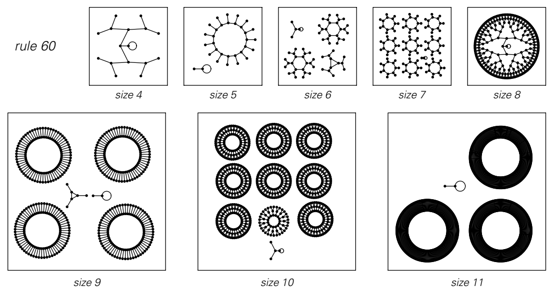
▪ **Shift rules.** The pictures below show networks obtained with rule 170, which just shifts every configuration one position to the left at each step. With any such shift rule, all states lie on cycles, and the lengths of these cycles are the divisors of the size n. Every cycle corresponds in effect to a distinct necklace with n beads; with k colors the total number of these is

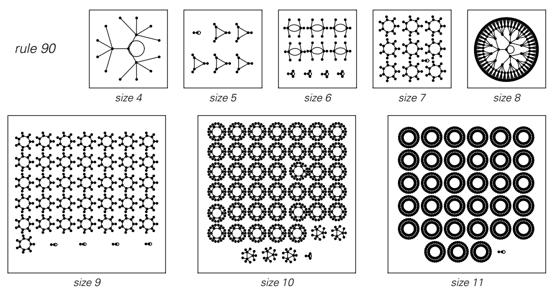
Apply[Plus, (EulerPhi[n/#] $k^{\#}$ &)[Divisors[n]]]/n

The number of cycles of length exactly m is s[m, k]/m, where s[m, k] is defined on page 950. For prime k, each cycle (except all 0's) corresponds to a term in the product Factor[x^{k^n-1} - 1, Modulus → k]. (See page 975.)

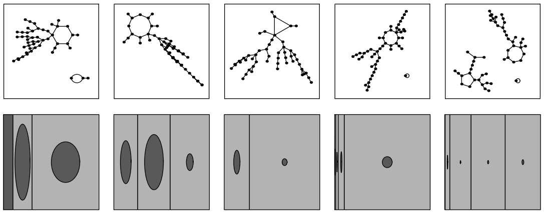


▪ **Additive rules.** The pictures below show networks obtained for the additive cellular automata with rules 60 and 90. The networks are highly regular and can be analyzed by the algebraic methods mentioned on page 951. The lengths of the longest cycles are given on page 951; all other cycles must have lengths which divide these. Rooted at every state on each cycle is an identical structure. When the number of cells n is odd this structure consists of a single arc, so that half of all states lie on cycles. When n is even, the structure is a balanced tree of depth 2^IntegerExponent[n, 2] and degree 2 for rule 60, and depth 2^IntegerExponent[n/2, 2] and degree 4 for rule 90. The total fraction of states on cycles is in both cases 2^(-2^IntegerExponent[n, 2]). States with a single black cell are always on the longest cycles. The state with no black cells always forms a cycle of length 1.





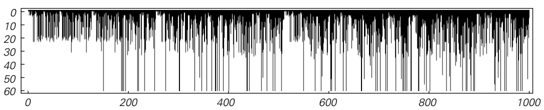
▪ **Random networks.** The pictures below show networks in which each of a set of n nodes has as its successor a node that is chosen at random from the set. The total number of possible such networks is n^n. For large n, the average number of distinct cycles in all such networks is Sqrt[π/2] Log[n], and the average length of these cycles is Sqrt[π n/8]. The average fraction of nodes that have no predecessor is $(1 - 1/n)^n$ or 1/ℯ in the limit $n \to \infty$. Note that processes such as cellular automaton evolution do not yield networks whose properties are particularly close to those of purely random ones.



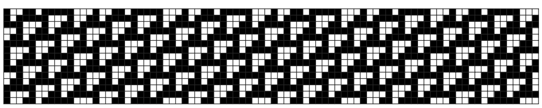
size 10 size 100 size 1000 size 10000 size 100000

Structures in Class 4 Systems

■ **Page 283 · Survival data.** The number of steps for which the pattern produced by each of the first 1000 initial conditions in code 20 survive are indicated in the picture below. 72 of these initial conditions lead to persistent structures. Among the first million initial conditions, 60,171 lead to persistent structures and among the first billion initial conditions the number is 71,079,205.



■ **Page 290 · Background.** At every step the background pattern in rule 110 consists of repetitions of the block *b = {1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0}*, as shown in the picture below. On step *t* the color of a cell at position *x* is given by *b[[Mod[x + 4 t, 14] + 1]]*.



■ **Page 292 · Structures.** The persistent structures shown can be obtained from the following *{n, w}* by inserting the sequences *IntegerDigits[n, 2, w]* between repetitions of the background block *b*:

{{152, 8}, {183, 8}, {18472955, 25}, {732, 10}, {129643, 18}, {0, 5}, {152, 13}, {39672, 21}, {619, 15}, {44, 7}, {334900605644, 39}, {8440, 15}, {248, 9}, {760, 11}, {38, 6}}

The repetition periods and distances moved in each period for the structures are respectively

{{4, -2}, {12, -6}, {12, -6}, {42, -14}, {42, -14}, {15, -4}, {15, -4}, {15, -4}, {15, -4}, {30, -8}, {92, -18}, {36, -4}, {7, 0}, {10, 2}, {3, 2}}

Note that the periodicity of the background forces all rule 110 structures to have periods and distances given by *{4, -2} r + {3, 2} s* where *r* and *s* are non-negative integers. Extended versions of structures (d)–(i) can be obtained by collisions with (a). Extended versions of (b) and (c) can be obtained from

Flatten[{IntegerDigits[1468, 2], Table[IntegerDigits[102524348, 2], {n}], IntegerDigits[v, 2]}]

where *n* is a non-negative integer and *v* is one of

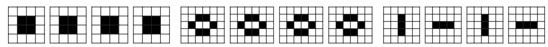
{1784, 801016, 410097400, 13304, 6406392, 3280778648}

Note that in most cases multiple copies of the same structure can travel next to each other, as seen on page 290.

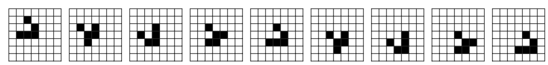
■ **Page 293 · Glider gun.** The initial conditions shown correspond to *{n, w} = {1339191737336, 41}*.

■ **Page 294 · Collisions.** A fundamental result is that the sum of the widths of all persistent structures involved in an interaction must be conserved modulo 14.

■ **The Game of Life.** The 2D cellular automaton described on page 949 supports a whole range of persistent structures, many of which have been given quaint names by its enthusiasts. With typical random initial conditions the most common structures to occur are:

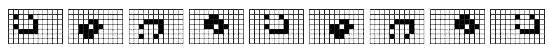


"block" "beehive" "blinker"

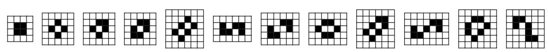


"glider"

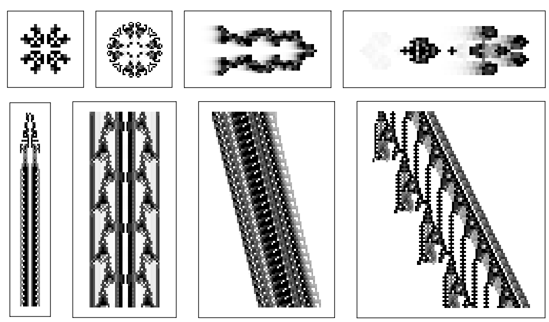
The next most common moving structure is the so-called "spaceship":



The complete set of structures with less than 8 black cells that remain unchanged at every step in the evolution are:

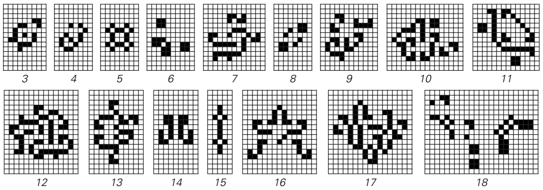


More complicated repetitive and moving structures are shown in the pictures below. If one looks at the history of a single row of cells, it typically looks much like the complete histories we have seen in 1D class 4 cellular automata.

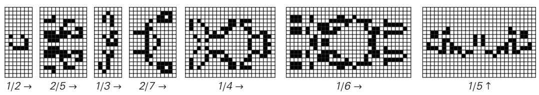


"pulsar" "26 oscillator" "slow ship" "blinker ship"

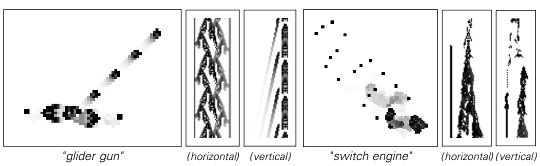
Structures with all repetition periods up to 18 have been found in Life; examples are shown in the pictures below.



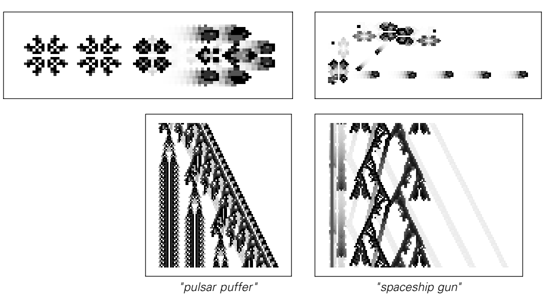
Persistent structures with various speeds in the horizontal and vertical direction have also been found, as shown below.



The first example of unbounded growth in Life was the so-called "glider gun", discovered by William Gosper in 1970 and shown below. This object emits a glider every 30 steps. The simplest known initial condition which leads to a glider gun contains 21 black cells. The so-called "switch engine" discovered in 1971 generates unbounded growth by leaving a trail behind when it moves; it is now known that it can be obtained from an initial condition with 10 black cells, or black cells in just a 5×5 or 39×1 region. It is also known that from less than 10 initial black cells no unbounded growth is ever possible.

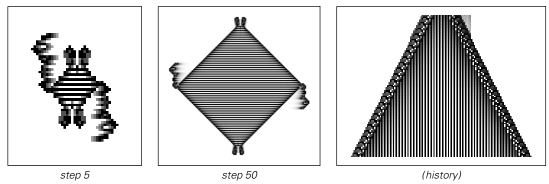


Many more elaborate structures similar to the glider gun were found in the 1970s and 1980s; two are illustrated below.

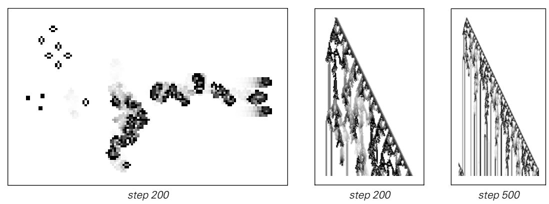


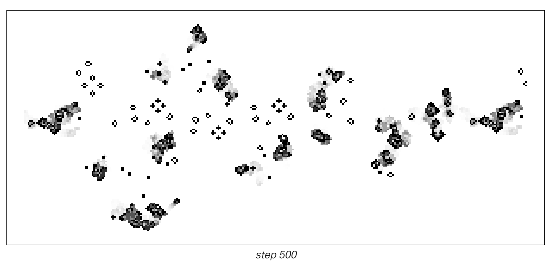
A simpler kind of unbounded growth occurs if one starts from an infinite line of black cells. In that case, the evolution is effectively 1D, and turns out to follow elementary rule 22, thus producing the infinitely growing nested pattern shown on page 263.

For a long time it was not clear whether Life would support any kind of uniform unbounded growth from a finite initial region of black cells. However, in 1993 David Bell found starting from 206 black cells the "spacefiller" shown below. This object is closely analogous to those shown for code 1329 on page 287.



As in other class 4 cellular automata, there are structures in Life which take a very long time to settle down. The so-called "puffer train" below which starts from 23 black cells becomes repetitive with period 140 only after more than 1100 steps.





■ **Other 2D cellular automata.** The general problem of finding persistent structures is much more difficult in 2D than in 1D, and there is no completely general procedure, for example, for finding all structures of any size that have a certain repetition period.

■ **Structures in Turing machines.** See page 888.